Apprentissage semi-supervisé de dictionnaire et de réseaux de neurones profonds

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 127, Astronomie et Astrophysique d'Ile de France (AAIF)

Spécialité de doctorat: Astronomie et astrophysique

Unité de recherche: DAp/UMR AIM

Référent: CEA Saclay

Thèse présentée et soutenue en visio, le 16/02/2021, par

Khanh Hung TRAN

Composition du jury:

Alexandre GRAMFORT	Président
Directeur de recherche, INRIA Paris-Saclay	
David PICARD	Rapporteur
Directeur de recherche, École des ponts ParisTech	
Julien MAIRAL	Rapporteur
Chargé de recherche, INRIA Grenoble	
Antitza DANTCHEVA	Examinatrice
Chargée de recherche, INRIA Sophia Antipolis	
Jean-Luc STARCK	Directeur
Directeur de recherche, CEA Saclay	
Fred Maurice NGOLE-MBOULA	Coencadrant
Chercheur, CEA Saclay	

Thèse de doctorat

NNT: 2021UPASP014

Remerciements

Je voudrais remercier les nombreuses personnes qui ont contribué au succès de ma thèse et qui m'a accompagné jusqu'au bout de cette mission, chronologiquement :

Je tiens à remercier l'équipe pédagogique de l'INSA de Lyon et de master IMA à Paris 6 pour les cours de grande qualité qui m'ont passioné pour lancer une thèse.

Mes remerciements vont ensuite à mon encadrant Fred-Maurice Ngole Mboula et mon directeur de thèse Jean-Luc Starck pour leur orientation et leur confiance à moi.

J'ai beaucoup apprécié l'ambiance de recherche au sein de Service Intelligence des Données et du groupe CosmoStat, où j'ai réalisé ma thèse. Je remercie tous ses membres de ces deux laboratoires.

J'ai passé de très bons moments avec mes camarades : Vincent, Baptiste, Edwin, Arnaud, Roman, Sandra, Etienne, Oudom, Régis, Shivani, Andrey, Meritxell, Ismaïl, Marisnel, Camille, Alyssa, Noëlie. Je leur remercie pour les escalades, les foots, les repas ...

Je tiens à remercier également le responsable à mon école doctorat, M. Alain Abergel pour sa gentillesse et pour ses conseils, ainsi que Florent Sureau et François Orieux pour leur disponibilité de participation dans mon comité de suivi.

Je remercie bien évidemment tous les membres du jury pour l'intérêt qu'ils ont porté à mes travaux et ainsi que mes deux rapporteurs pour leur lecture attentive de ce manuscrit.

Pour finir, plus personnel, je tiens à témoigner ma reconnaissance à ma famille, notamment mes parents qui m'ont soutenu tout au long de nombreuses années d'étude.

Abtract

Since the 2010's, machine learning (ML) has been the object of intense scientific activity. Numerous ML models have given rise to new results in various fields such as Computer Vision, Natural Language Processing, Robotics ... However, most of these models use supervised learning, which requires a massive amount of annotated examples. Thus, the objective of this thesis is to study and propose new methods for weakly supervised learning.

First, we present the mathematical and algorithmic tools on which our work is based. This includes dictionary and deep neural networks learning, manifold learning, and pseudo-labelling.

In a second step, we propose a new method of semi-supervised dictionary learning to deal with the problem of classification when a small number of training samples (labeled and unlabeled) is available. This method is based on the one hand on the preservation of the spatial organization of the original data in the sparse codes space, using cost functions derived from manifold learning. On the other hand, a classifier is learned in the sparse codes space in a semi-supervised way and jointly with the dictionary and the sparse codes. This method provides an improvement in the classification accuracy rate compared to state-of-the-art methods in semi-supervised dictionary learning.

The limitations of this first approach led us to propose a semi-supervised learning method for neural networks. This second approach is inspired by adversarial learning: we generate virtual points for which the structural deviation between the original space and the latent space is the strongest; then the model parameters are updated to minimize this distortion. This approach brings not only an improvement in classification accuracy rates compared to state-of-the-art methods, but also a big difference in robustness to adversarial examples.

Finally, we analyze similarities and differences, as well as advantages and disadvantages between dictionary learning and neural networks. We conclude with some perspectives on these two types of models.

Résumé étendu

Depuis les années 2010, l'apprentissage automatique (ML[1]) fait l'objet d'une activité scientifique intense. De nombreux modèles de ML ont donné lieu à des résultats inédits dans des divers domaines tels que la Vision par ordinateur, le Traitement automatique des langues, la Robotique... Toutefois, la plupart de ces modèles emploient l'apprentissage supervisé, qui requiert une quantité massive d'exemples annotés. Ainsi, l'objectif de cette thèse est d'étudier et de proposer de nouvelles méthodes d'apprentissage faiblement supervisé.

En plus que la perte supervisée $\mathcal{L}_s$, nous introduissons la perte non-supervisée $\mathcal{L}_u$ afin de régulariser mieux le modèle f :

$$\mathcal{L}_s(f, \mathcal{X}^l, \mathcal{Y}) + \mathcal{L}_u(f, \mathcal{X}) \tag{1}$$

où $f, \mathcal{X}^l, \mathcal{Y}$ are respectivement le modèle, les échantillons labélisés et les labels. $\mathcal{X}$ signifie les conventionels échantillons qui sont tous les échantillons labélisés et non-labélisé. Dans le premier temps, nous contruissons une approche d'apprentissage semi-supervisé sur le modèle d'apprentissage de dictionnaire et dans le deuxième temps, nous contruissons une autre approche d'apprentissage semi-supervisé sur un réseau de neurones profonds. Avant d'aborder ces deux approches, nous commençons en présentant l'apprentissage de variétés, qui est le point-clé pour construire la perte non-supervisée.

Apprentissage de variétés

L'apprentissage de variétés fait partie de l'apprentissage de représentation (non-supervisé). Pour une donnée de grande dimension, c'est-à-dire la représentation originale des échantillons a de nombreuses caractéristiques, il peut être difficile d'interpréter et de visualiser l'organisation globale des données. Nous supposons que, pour une tâche donnée, les données d'intérêt se trouvent sur une variété de faible nombre de dimensions. Par conséquent, les coordonnées des données par rapport

[1]Machine Learning

à cette variété peuvent être appliquées pour la visualisation et la réduction de dimensionnalité.

Note that, il existe plusieurs méthodes pour apprendre une variété les données d'intérêt. Dans cette thèse, pour la simplicité, nous convenons que l'apprentissage de variétés est la préservation de structure géométrique (PSG). Dans la plupart des cas, la démarche de PSG comporte deux étapes: nous extrayons tout d'abord une propriété géométrique, parmi des échatillons dans la représentation originale. Puis nous cherchons une représentation latente qui préserve cette propriété géométrique pour tous les échatillons.

Étant donné un ensemble de données $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{x}_i \in \mathbb{R}^n$ et sa représentation latente $\mathcal{A} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}, \mathbf{a}_i \in \mathbb{R}^p$, où $\mathbf{a}_i$ est la representation latente de $\mathbf{x}_i$, nous notons:

$$\min_{\mathcal{A}} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c), \tag{2}$$

la perte de PSG d'une représentation latente $\mathbf{a}_i$ versus autres représentations latentes. $\mathcal{A}_i^c$ est le complément de $\{\mathbf{a}_i\}$ dans $\mathcal{A}$. La perte totale de PSG est alors définie comme :

$$\min_{\mathcal{A}} \mathcal{L}_t = \min_{\mathcal{A}} \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) \tag{3}$$

Nous présentons ci-dessous quelques pertes PSG populaires :

- *Multi-Dimensional scaling* (MDS) [Kruskal & Wish 1978] :

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} (d_a(\mathbf{a}_i, \mathbf{a}_j) - d_x(\mathbf{x}_i, \mathbf{x}_j))^2, \tag{4}$$

où $d_a()$ et $d_x()$ sont des métriques de dissimilarité. Par défaut, elles sont toutes deux des distances euclidiennes. Cette méthode vise à préserver les distances par paire depuis la représentation originale dans la représentation latente.

- *Laplacian eigenmaps* (LE) [Belkin & Niyogi 2003] :

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} d_x(\mathbf{x}_i, \mathbf{x}_j) d_a(\mathbf{a}_i, \mathbf{a}_j), \tag{5}$$

où $d_x()$ est une métrique de similarité (par exemple $d_x(\mathbf{x}_i, \mathbf{x}_j) = \exp(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma_i^2})$) et $d_a()$ est une métrique de dissimilarité (par exemple $d_a(\mathbf{a}_i, \mathbf{a}_j) = \|\mathbf{a}_i - \mathbf{a}_j\|_2^2$). La variété est apprise en préservant des distances locales entre deux échantillons. Afin de réduire l'impact des longues distances, dans certains travaux, $d_x(\mathbf{x}_i, \mathbf{x}_j)$ est directement mis à zéro si $\mathbf{x}_j$ ne fait pas partie des k plus proches voisins de $\mathbf{x}_i$ ou vice versa si $\mathbf{x}_i$ ne fait pas partie des k plus proches voisins de $\mathbf{x}_j$. Nous pouvons également définir $d_x(\mathbf{x}_i, \mathbf{x}_j) = 0$ si $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 > \kappa$. Soit $\mathbf{W}$ la matrice définie par $\mathbf{W}_{ij} = d_x(\mathbf{x}_i, \mathbf{x}_j)$, en conséquence $\mathbf{W}$ est une matrice symétrique si $d_x()$ est symétrique. Nous représentons la perte totale de la méthode Laplacien eigenmaps à l'aide des matrices :

$$\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ neqi}}^{N} \mathbf{W}_{ij} \left\| \mathbf{a}_i - \mathbf{a}_j \right\|_2^2 = 2\,\mathrm{tr}(\mathbf{A}\mathbf{L}\mathbf{A}^\top), \tag{6}$$

où $\mathbf{L} = \mathbf{D} - \mathbf{W}$, $\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{W}_{ij}$ ($\mathbf{D}$ est une matrice diagonale). $\mathbf{L}$ est une matrice laplacienne car elle est symétrique, la somme de chaque ligne est égale à 0 et ses éléments sont négatifs à l'exception des éléments diagonaux.

- *Locally Linear Embedding* (LLE) [Roweis & Saul 2000] :

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \left\| \mathbf{a}_i - \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} \lambda_{ij} \mathbf{a}_j \right\|_2^2, \tag{7}$$

où λ_{ij} sont déterminés en résolvant le problème suivant:

$$\begin{aligned} &\min_{\lambda_{ij}} \left\| \mathbf{x}_i - \sum_j \lambda_{ij} \mathbf{x}_j \right\|_2^2, \\ &\text{soumis à: } \begin{cases} \sum_j \lambda_{ij} = 1, \text{ si } \mathbf{x}_j \in \mathrm{knn}(\mathbf{x}_i), \\ \lambda_{ij} = 0 \text{ sinon.} \end{cases} \end{aligned} \tag{8}$$

où $\mathrm{knn}(\mathbf{x}_i)$ désigne l'ensemble qui contient les indices des k échantillons les plus proches voisins (par distance euclidienne) de l'échantillon $\mathbf{x}_i$. En supposant que les données observées $\mathcal{X}$ sont échantillonnées à partir d'une variété lisse et à condition que l'échantillonnage soit suffisamment dense, nous pouvons supposer que l'échantillon se trouve localement sur des patchs linéaires. Ainsi, LLE calcule d'abord les coordonnées barycentriques de chaque échantillon par rapport à ses plus proches voisins. Ces coordonnées barycentriques caractérisent la géométrie locale de la variété sous-jacente. Ensuite, LLE calcule une représentation latente qui est compatible avec ces coordonnées barycentriques locales. En introduisant $\mathbf{V} \in \mathbb{R}^{N \times N}$ une représentation matricielle pour λ : $\mathbf{V}[j,i] = \lambda_{ij}$ alors la perte totale $\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c)$ est peut être réécrite : $\mathcal{L}_t = \|\mathbf{A} - \mathbf{A}\mathbf{V}\|_F^2 = \mathrm{tr}(\mathbf{A}\mathbf{L}\mathbf{A}^\top)$, où $\mathbf{L} = \mathbf{I}_N - \mathbf{V} - \mathbf{V}^\top + \mathbf{V}^\top\mathbf{V}$. Ainsi, la perte $\mathcal{L}_t$ est peut être interprétée comme la perte de Laplacien eigenmaps dont la métrique d_x qui est implicite.

Dans le cas non-paramétré, apprentissage de variétés par PSG est réalisé en optimisant la perte totale directement sur la représentation latente $\mathcal{A}$. Par contre, dans le cas paramétré, nous nous servons d'une fonction $g()$ qui part de la représentation originale dans la représentation latente :

$\mathbf{a} = g(\mathbf{x})$ et la PSG est réalisée en optimisant des paramètres entraînables de la fonction $g()$. Par exemple, $g()$ est un réseau de neurones.

Apprentissage semi-supervisé de dictionnaire

Nous présentons d'abord l'apprentissage de dictionnaire et ensuite notre méthode proposée sur ce modèle.

Apprentissage de dictionnaire. L'apprentissage de dictionnaire englobe des méthodes et algorithmes qui, pour un type de signal donné, visent à dériver un ensemble de caractéristiques cardinales permettant de décrire de manière concise les signaux de ce type. Plus précisément, étant donné un ensemble de vecteurs $(\mathbf{x}_j)_{1\leq j\leq N}$ dans $\mathbb{R}^n$ qui représentent des signaux (électrocardiographie, séquences de gènes...), l'apprentissage de dictionnaire calcule un ensemble de vecteurs $(\mathbf{d}_i)_{1\leq i\leq p}$ dans $\mathbb{R}^n$ et un ensemble de coefficients $(\mathbf{a}_j)_{1\leq j\leq N}$ dans $\mathbb{R}^p$ de sorte que:

$$\mathbf{x}_j \approx \sum_{i=1}^{p} \mathbf{a}_j[i]\mathbf{d}_i \tag{9}$$

où il y seulement quelque coefficients $\mathbf{a}_j[i]$ qui ne sont pas nul. Nous appelons $\mathbf{a}_j$ la représentation parcimonieuse ou le code parcimonieux du signal $\mathbf{x}_j$. La matrice $\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_p], \mathbf{D} \in \mathbb{R}^{n\times p}$ est le dictionnaire qui contient les caractéristiques cardinales que nous appelons les atomes. En général, le problème d'apprentissage de dictionnaire est écrit sous la forme :

$$\min_{\mathbf{A},\mathbf{D}\in\mathcal{C}} \mathcal{R}(\mathbf{A},\mathbf{D}) = \min_{\mathbf{A},\mathbf{D}\in\mathcal{C}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_2^2 + \lambda \|\mathbf{A}\|_q ,$$

où $0 \leq q \leq 1$ indique la norme appliquée sur le code $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_N]$ pour que le dernier devienne parcimonieux. Pour rappel, $\mathbf{X} \in \mathbb{R}^{n\times N}$ et $\mathbf{A} \in \mathbb{R}^{p\times N}$. $\mathcal{C}$ désigne l'ensemble de dictionnaire dont tous les atomes vérifient $\|\mathbf{d}_i\|_2 \leq 1$, pour éviter le transfer d'énergie entre $\mathbf{D}$ et $\mathbf{A}$. Les applications les plus connues de l'apprentissage de dictionnaire sont l'acquisition comprimée et la restauration du signal. En plus, l'apprentissage de dictionnaire est aussi utilisé pour la classification. La figure 1 montre un exemple que la représentation parcimonieuse est plus discriminante que la représentation originale.

Méthode proposée. Nous considérons que la donnée $\mathbf{X}$ contient deux parties $\mathbf{X}^l$ et $\mathbf{X}^u$, qui designent respectivement la donnée labélisée et non-labélisé, $\mathbf{X} = [\mathbf{X}^l, \mathbf{X}^u]$. Par correspondance, $\mathbf{A} = [\mathbf{A}^l, \mathbf{A}^u]$ qui contient le code parcimonieux labélisé et non-labélisé.

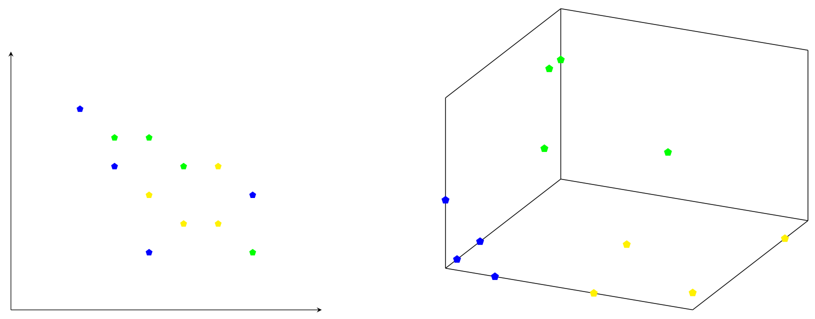


Figure 1: Gauche : La représentation originale de 2 dimensions avec 12 échantillons en 3 classes. Nous voyons que c'est difficile pour avoir un classifieur linéaire pour séparer ces 3 classes. Droite : La représentation parcimonieuse de 3 dimensions. Nous voyons que avec cette représentation, c'est plus simple pour séparer ces 3 classes par un classifieur linéaire.

Nous formulons littéralement la fonction de coût pour l'apprentissage semi-supervisé de dictionnaire :

$$\begin{aligned} &\min_{\Theta} \left[\mathcal{R}(\mathbf{A},\mathbf{D}) + \mathcal{D}(\theta,\mathbf{A}^l,\mathbf{A}^u,\mathbf{D},\mathbf{Y},\mathbf{P}) + \mathcal{F}(\mathbf{A},\mathbf{D})\right], \\ &\text{où } \theta = \{\theta,\mathbf{A}^l,\mathbf{A}^u,\mathbf{D}\in\mathcal{C},\mathbf{P}\}. \end{aligned} \tag{10}$$

- $\mathcal{R}$ signifie la perte de reconstruction avec la contrainte parcimonieuse.
- $\mathcal{D}$ signifie la perte de classification, qui a pour but de rendre le code $\mathbf{A}$ ou le dictionnaire $\mathbf{D}$ plus discriminant. θ signifie des paramrètres d'un classifieur et $\mathbf{Y},\mathbf{P}$ signifient respectivement les labels et les peuso-labels.
- $\mathcal{F}$ signifie la perte de régularisation, qui peut être appliquée sur le code parcimonieux $\mathbf{A}$ ou le dictionnaire $\mathbf{D}$.

Basé sur l'équation (10), nous proposons une méthode d'apprentissage semi-supervisé de dictionnaire afin de traiter le problème de la classification lorsqu'un faible nombre d'échantillons labélisés. Premièrement, cette méthode repose sur la préservation de structure géométrique de la représentation originale dans l'espace des codes parcimonieux, c'est à dire la perte de régularisation $\mathcal{F}(\mathbf{A}) = \beta\mathcal{L}_t = \beta\,\mathrm{tr}(\mathbf{A}\mathbf{L}\mathbf{A}^\top)$ avec la technique Locally Linear Embedding (LLE) dans la section précédente. Deuxièmement, en notant C le nombre de classes, un classifieur linéaire $(\mathbf{w}_c,\mathbf{b}_c)$ où $c = 1,..,C$ est appris dans l'espace des codes parcimonieux de façon semi-supervisée par la technique peuso-label.

$$\min_{\mathbf{W},\mathbf{b},\mathbf{A},\mathbf{P},\mathbf{D}\in\mathcal{C}} \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 + \lambda\|\mathbf{A}\|_1 + \gamma\Big(\sum_{i=1}^{N_l}\sum_{c=1}^{C}\left\|\mathbf{w}_c^\top\mathbf{a}_i^l + b_c - y_i^c\right\|_2^2$$
$$+\sum_{j=1}^{N_u}\sum_{k=1}^{C}(\mathbf{P}_{kj})^r\sum_{c=1}^{C}\left\|\mathbf{w}_c^\top\mathbf{a}_j^u + b_c - y_j^c(k)\right\|_2^2\Big) + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2) + \beta\,\mathrm{tr}(\mathbf{A}\mathbf{L}\mathbf{A}^\top), \tag{11}$$

où y_i^c signifie le label d'échantillon labélisé $\mathbf{x}_i^l$ pour une classe c; $y_j^c(k)$ signifie le label d'échantillon non-labélisé $\mathbf{x}_i^l$ pour une classe c en supposant que cette échantillon appartient à la classe k. Troisièmement, nous réalisons le codage parcimonieux pour un nouvelle échantillon $\mathbf{x}_{N+i}$ en prenant en compte la PSG :

$$\min_{\mathbf{a}_{N+i}} \|\mathbf{x}_{N+i} - \mathbf{D}\mathbf{a}_{N+i}\|_2^2 + \lambda\|\mathbf{a}_{N+i}\|_1 + \beta\omega_A\left\|\mathbf{a}_{N+i} - \sum_{j\in\mathrm{knn}'(N+i)}\hat{\lambda}_{ij}V\mathbf{a}_j\right\|_2^2 \tag{12}$$

L'ensemble knn $'(N+i)$ contient les indices des k échantillons les plus proches parmi les N échantillons d'entraînement pour le nouvelle échantillon $\mathbf{x}_{N+i}$. Nous obtenons les coefficients λ_{ij} en résolvant le problème (8).

Dans la partie d'expérience numérique, nous montrons que dans le cas de faible nombre d'échantillons labélisés, notre méthode proposée apporte une amélioration sur la précision par rapport aux méthodes de l'état de l'art en apprentissage semi-supervisé de dictionnaire, avec les données standards comme MNIST, USPS, YaleB et AR. Cependant, dans nos expérimentations, le modèle d'apprentissage de dictionnaire n'est pas pertinent pour traiter directement les données où le nombre de dimensions (caractéristiques) est supérieur à 1000, parce que nous arrivons à un problème de computation que nous ne disposons pas d'ordinateur efficace pour réaliser l'optimisation. Une solution est de faire un prétraitement pour réduire le nombre de dimensions (par exemple PCA) mais ce prétraitement peut entraîner une perte importante d'informations discriminantes.

Apprentissage semi-supervisé de réseaux de neurones profonds

Les limitations de cette première approche nous ont conduit à proposer une méthode d'apprentissage semi-supervisé de réseaux de neurones profonds. Le modèle de réseau de neurones, en particulier le réseau neuronal convolutif a montré la capacité de traiter efficacement les données (images) dont le nombre de dimensions est beaucoup plus grand, par exemple $224 \times 224 \times 3 \approx 150K$ dimensions.

Nous notons f_θ est un réseau de neurones avec des paramètres θ. Comme le modèle d'apprentissage de dictionnaire, nous intégrons la préservation de structure géométrique dans la fonction de coût :

$$\min_\theta L_s + \lambda L_{psg} + \beta L_u \tag{13}$$

où L_s est la perte supervisée, nous prenons souvent l'entropie croisée; L_{psg} est la perte PSG (non-supervisé), de la représentation originale vers une représentation latente dans un réseau de neurones; L_u signifie des autres éventuelles pertes non-supervisées, par exemple pseudo-label, apprentissage auto-supervisé...

Avant de détailler notre approach proposée, nous présentons d'abord l'apprentissage antagoniste de variétés.

Apprentissage antagoniste de variétés. Tout d'abord, la tâche PSG est paramétrée par un réseau de neurones, ç'est à dire une représentation latente $\mathbf{a}_i$ d'un échantillon $\mathbf{x}_i$ est exprimée par $\mathbf{a}_i = f_\theta^{(l)}(\mathbf{x}_i)$, où (l) est l'indice d'une couche intermédiaire du réseau de neurones $f()$. Par conséquent, nous avons le problème de PSG avec model de réseau de neurones :

$$\min_\theta L_{psg} = \mathcal{L}_t\left(f_\theta^{(l)}(\mathbf{x}_1^l), .., f_\theta^{(l)}(\mathbf{x}_N^l)\right) \tag{14}$$

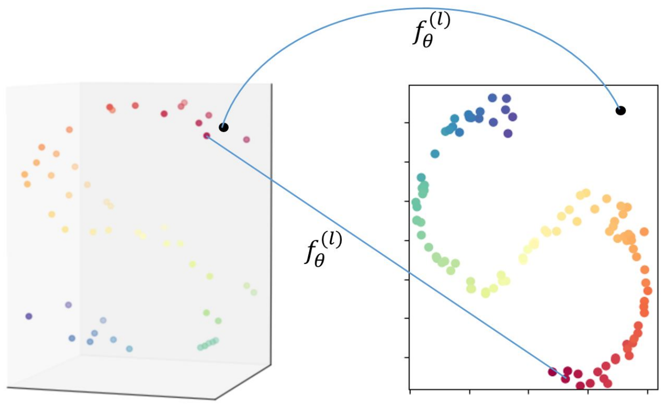


Figure 2: Préservation de structure géométrique et un cas d'échec. A gauche : la représentation originale de la donnée de forme S en 3 dimensions. A droite : une représentation latente de cette donnée en 2 dimensions. La similarité entre les échantillons est indiquée par des couleurs différentes.

La figure 2 montre un exemple de PSG réalisée par un réseau de neurone $f_\theta^{(l)}$, en résolvant le problème (14). Les points en différentes couleurs sauf le noir sont les échatillons d'entraînement et nous voyons bien que la structure géométrique de ces échatillons est bien préservé dans la représentation latente. Néanmoins, si l'architechture du modèle $f_\theta^{(l)}$ est compliquée, par exemple, avec un grand nombre de paramètres, nous arrivons évantuellement à un cas d'échec pour PSG. Dans la représentation originale, l'échantillon noir est proche des échantillons rouges mais sa représentation latente est éloignée celle des échantillons rouges. Potentiellement, c'est parce que les échantillons d'entraînment ne sont pas assez dense et cette échantillon noir n'est pas utilisé pour entraîner le modèle.

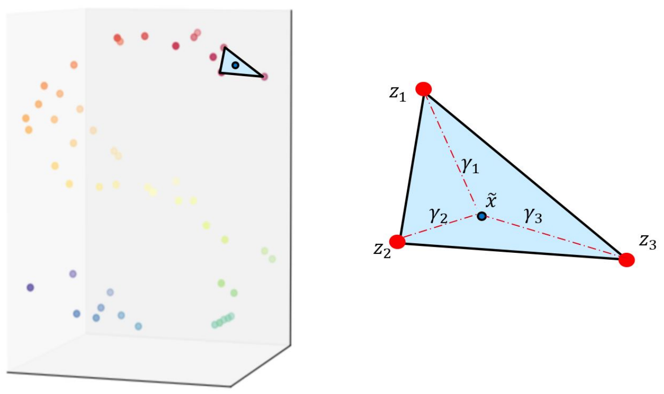


Figure 3: A gauche : les échantillons de forme S avec une enveloppe convexe définie par trois points d'ancrage. A droite : l'extraction et l'agrandissement de cet enveloppe convexe avec un point virtuel $\tilde{\mathbf{x}}$ et des points d'ancrage $\mathbf{z}$. Dans ce cas, les points d'ancrage sont des échantillons proches les uns des autres.

Dans l'objective de régulariser le modèle $f_\theta^{(l)}$ pour éviter le cas d'échec pour PSG, nous introduissons le concept des points virtuels et des points d'attaque. La figure 3 montre un exemple pour un point virtuel $\tilde{\mathbf{x}}$, qui est coordonné à l'aide des points d'ancrage $\mathbf{z}$ avec les coefficients γ :

$$\begin{aligned} &\tilde{\mathbf{x}} = \gamma_1\mathbf{z}_1 + \gamma_2\mathbf{z}_2 + ... + \gamma_p\mathbf{z}_p, \\ \text{subject to: } &\gamma_1, \gamma_2, .., \gamma_p \geq 0, \\ &\gamma_1 + \gamma_2 + ... + \gamma_p = 1. \end{aligned} \tag{15}$$

Les contraintes sur γ garantissent que le point virtuel est toujours à l'intérieur de l'enveloppe convexe définie par des points d'ancrage. Concernant le point d'attaque, c'est un point virtuel qui maximise localement une perte de PSG avec le modèle $f_\theta^{(l)}$:

$$\max_\gamma \mathcal{L}_e(\tilde{\mathbf{a}}, \mathcal{A}) = \max_\gamma \mathcal{L}_e\big(f_\theta^{(l)}(\tilde{\mathbf{x}}), \{f_\theta^{(l)}(\mathbf{x}_1), .., f_\theta^{(l)}(\mathbf{x}_N)\}\big). \tag{16}$$

En intégrant des points d'attaque comme des échantillons synthétiques supplémentaires pour la tâche de PSG comme dans le problème (17), nous souhaitons avoir une meilleur régularisation pour le modèle $f_\theta^{(l)}$. Afin d'optimiser le problème (17), nous effectuons alternativement entre deux phases, phase d'attaque avec γ comme variable et mise à jour des paramètres du modèle avec θ

comme variable.

$$\min_{\theta} \max_{\gamma} \mathcal{L}_t \left(f_\theta^{(l)}(\mathbf{x}_1), .., f_\theta^{(l)}(\mathbf{x}_N), f_\theta^{(l)}(\tilde{\mathbf{x}}^1), .., f_\theta^{(l)}(\tilde{\mathbf{x}}^M) \right) \tag{17}$$

Méthode proposée. En remplaçant la perte PSG dans le problem (13) par l'apprentissage antagoniste de variétés, nous obtenons:

$$\begin{aligned} &\min_{\theta} \max_{\gamma} (L_s + \lambda L_{pgs} + \beta L_u) \\ =&\min_{\theta} \max_{\gamma} \frac{1}{N_l} \sum_{i=1}^{N_l} \mathcal{L}_c(f_\theta(\mathbf{x}_i^l), y_i) + \lambda \mathcal{L}_t \left(f_\theta^{(l)}(\mathbf{x}_1), .., f_\theta^{(l)}(\mathbf{x}_N), f_\theta^{(l)}(\tilde{\mathbf{x}}^1), .., f_\theta^{(l)}(\tilde{\mathbf{x}}^M) \right) + \beta L_u \end{aligned} \tag{18}$$

où N_l signifie le nombre d'échantillons labélisés et N signifie le nombre d'échantillons (labélisé et non-labélisé). $\mathcal{L}_c$ signifie la perte de l'entropie croisée.

Nous appliquons notre proposé méthode sur quelques bases de donnée standard comme CIFAR10, SVHN, ImageNet. Nous montrons que cette approche apporte non seulement une amélioration des taux de précision de classification par rapport aux méthodes de l'état de l'art, mais aussi une grande amélioration de robustesse exemples antagonistes.

Contents

I Introduction **1**

A Context . . . 1

B Learning paradigms . . . 3

C Motivations . . . 6

D Manuscript outlines . . . 7

II Methodological pillars **9**

A Dictionary learning . . . 9

A.1 Sparse coding . . . 10

A.2 Dictionary update . . . 13

A.3 Conclusion on dictionary learning . . . 14

B Supervised dictionary learning . . . 15

B.1 SDL with internal classifier . . . 16

B.2 SDL with atoms discriminative . . . 18

B.3 Conclusion about SDL . . . 19

C Deep Learning . . . 20

C.1 Neural networks . . . 20

C.2 Convolutional neural networks . . . 20

C.3 Standard CNNs . . . 23

C.4 Optimizers . . . 30

D Manifold learning . . . 31

III Semi-supervised dictionary learning **37**

A Introduction . . . 37

A.1 Generalities . . . 38

A.2 Related works . . . 39

B Proposed method 41

B.1 Construction of objective function 42

B.2 Optimization 44

B.3 Numerical experiments 47

B.4 Conclusion about proposed method 55

C Conclusion 55

IV Semi-supervised deep learning 57

A Related works 57

A.1 Notations 57

A.2 Auxiliary task as regularization 59

A.3 Pseudo labeling 70

A.4 Generative models 72

A.5 Virtual Adversarial Training 76

A.6 Holistic methods 77

A.7 Partial conclusion for semi-supervised neural networks 79

B Manifold attack 79

B.1 Individual attack point versus data points 79

B.2 Attack points as data augmentation 82

B.3 Pairwise manifold learning 83

B.4 Settings of anchor points and initialization of virtual points 84

C Applications of manifold attack 85

C.1 Manifold learning on a small dataset 85

C.2 Robustness to adversarial examples 87

C.3 Semi-supervised manifold attack 91

C.4 Conclusion about manifold attack 92

V General Conclusion 95

Appendix A 101

1 Optimization for Laplacian Learning 101

2 Probability update 102

3 Sparse coding 103

4 Dictionary update 105

5 Classifier update 106

6 Sparse coding with epoch and batch 107
7 Parallel training for dictionary learning model 108
8 Pseudo labeling 109
8.1 Label Propagation 110
8.2 Sharpening 112

Appendix B **115**
1 Projection for sum and positive constraint 115
2 Adversarial examples 116
3 Hyper-parameters 121
3.1 Robustness to adversarial examples 121
3.2 Semi-supervised manifold attack 121
4 Parallel programming 121
4.1 Data Parallel 122
4.2 Model Parallel 122
4.3 Discussion about parallel training 123
5 System of index for anchor points 124

Bibliography **127**

Acronyms

AE	Auto-Encoder.
AI	Artificial Intelligence.
BN	Batch Normalization.
CIFAR	Canadian Institute For Advanced Research.
CNN	Convolutional Neural Network.
CV	Computer Vision.
DeL	Deep Learning.
DiL	Dictionary Learning.
EMA	Exponentially Moving Average.
ERM	Empirical Risk Minimization.
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm.
GAN	Generative Adversarial Network.
GNN	Graph Neural Network.
GPU	Graphics Processing Unit.
ICT	Interpolation Consistency Training.
IHT	Iterative Hard Thresholding.
JDL	Joint Dictionary Learning.

KL Kullback–Leibler.

LC-KSVD Label Consistent K-SVD.

LC-RLSDLA Label-Consistent - Recursive Least Squares Dictionary Learning Algorithm.

LE Laplacian Eigenmaps.

LL Laplacian Learning.

LLE Locally Linear Embedding.

LP Label Propagation.

LSTM Long short-term memory.

MDS Multi-Dimensional Scaling.

ML Machine Learning.

MNIST Modified National Institute of Standards and Technology.

MP Matching Pursuit.

NLP Natural Language Processing.

NNM Neural Network Model.

ODL Online Dictionary Learning.

OMP Orthogonal Matching Pursuit.

OSSDL Online Semi-Supervised Dictionary Learning.

PSG Préservation de Structure Géométrique.

PSSDL Probabilistic Semi-Supervised Dictionary Learning.

RNN Recurrent Neural Network.

S2D2 Semi-Supervised Discriminative Dictionary.

SC Sparse Code.

SDGDL Supervised Dual Graph Dictionary Learning.

SDL Supervised Dictionary Learning.

SE Squeeze and Excitation.

SL Supervised Learning.

SNE Stochastic Neighbor Embedding.

SR Sparse Representation.

SSD-LP Semi-Supervised Dictionary - Label Propagation.

SSDL Semi-Supervised Dictionary Learning.

SSDL-GA Semi-Supervised Dictionary Learning with Graph regularization and Active points.

SSL Semi-Supervised Learning.

SSNN Semi-Supervised Neural Network.

SSP-DL Structural Sparse Preserving - Dictionary Learning.

SSR-D Semi-Supervised Robust Dictionary.

SVHN Street View House Numbers.

SVM Support Vector Machine.

UL Unsupervised Learning.

USPS United States Postal Service.

USSDL Unified Semi-Supervised Dictionary Learning.

VAE Variational Autoencoder.

VAT Virtual Adversarial Training.

Chapter I

Introduction

"Data is the new oil."
– Clive Humby

This thesis aims at improving the performance of semi-supervised learning by Dictionary Learning (DiL) model with Sparse Representation (SR) at the first time and by Deep Learning (DeL) model at the second time. We focus on the geometric preservation between the original representation of data and its hidden representation. In this chapter, we present firstly the context and learning paradigms, then our motivations. Finally, we present the outline of this thesis.

A Context

We live in an era where fossil fuels are running out and data are increasing. Soon, the mining tools will not be oil-shores or big excavators, but rather high-performance computing machines with complex algorithms. Tremendous volume of data are now available in regards to virtually everything. It includes biomedical signals, genomic data, industrial systems signals, weather and environment related data, astronomical surveys data, satellite earth images and a plethora of human activities generated data, *e.g.* images or videos captured then uploaded on social media, exchanged emails and messages, rating notes for films...

"Without big data, you are blind and deaf and in the middle of a freeway."
– Geoffrey Moore

Officially named in 1956 at Dartmouth Conference, Artificial Intelligence (AI) is a field of computer sciences aimed at emulating different aspects of human and animal intelligence for automated tasks. Its subfield coined Machine Learning (ML) consists in using data to build and improve the ability of a software agent to perform AI related tasks. It has become one of the most interesting research subject for the last two decades. Applications of AI in general and ML in particular are found in various domains such as:

- Computer Vision (CV): Image Recognition (objects, persons, places), Image Segmentation, Object Tracking, Video Surveillance...
- Natural Language Processing (NLP): Question Answering, Machine Translation, Voice Verification (Authentication), Text Generation, Information Retrieval (search engines), Email Filtering, Document Analysis, Speech Recognition (virtual assistant) ...
- Robotic: Imitation Learning, Self-driving cars...
- Finance: Portfolio Management, Stock market Prediction, Fraud Detection...
- Entertainment: Recommendation System (films, musics), Game bots...

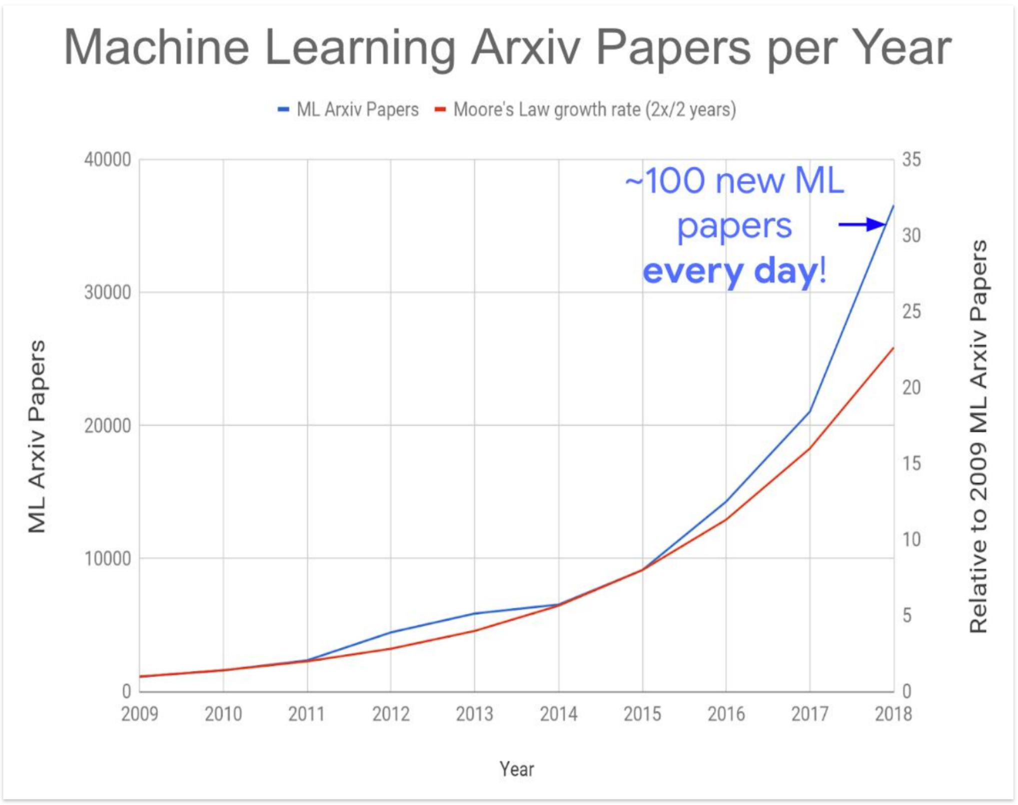


Figure I.1: Number of ML articles per year on Arxiv, a popular public repository of research papers. They have more than doubled every two years, which is more than Moore's prediction for chips in 1975. Credits: [Dean *et al.* 2018].

The global ML market is worth about $7.3B in 2020 and will reach to $30.6B in 2024, reaching an annual growth rate of 43% [1] (without taking into account impacts of coronavirus). The "GAFA" (Google, Amazon, Facebook, and Apple) have invested heavily in ML research and in the development of ML platform services, *e.g.* renting high performance GPUs, cloud storage. On the hardware side, NVIDIA develops specific GPUs for ML tasks, Google develops TPUs (Tensor Processing Unit)... The number of ML start-ups is ever growing and the number of ML articles has been increasing very quickly over the last decade and reached to about 100 papers per day in 2018, as shown in figure I.1.

Jointly to the depicted corporate and scientific investment, the ML wave is strongly sustained by the growing access to free ML training resources of quality. Anyone can find lectures of leading experts on platforms such as fast.ai, Coursera, Udemy, Udacity. Moreover, ML communities or media such as Kaggle, Medium, Towards Data Science contribute a great deal to ML important ideas popularizing, as well as practionning. We review some of the main learning paradigms in the next section.

B Learning paradigms

In this section, we present some popular learning paradigms in ML, namely supervised learning, unsupervised learning and semi-supervised learning. Figure I.2 shows an illustration for a general ML task with three main factors: data, model or mapping function (with parameters) and optimization process. The target is optional (required only for supervised learning).

"...what we want is a machine that can learn from experience."

– Alan Turing

Supervised learning. (SL) This type of learning requires labelled data, which means that each sample in the training set has a corresponding label, which is often provided by human. The training stage consists in computing, through an optimization algorithm, a model that maps a sample taken as input to its corresponding label. The computed model is then used to predict the labels of samples which are not in the training set. There are two main types of SL problems: classification and regression. In classification, the model predicts a categorical variable, while in regression, the model predicts a continuous variable. For illustration purposes and with no loss of generality, we only present SL for classification problems hereafter. In this case, a sample's label is typically represented in a one-hot encoded form. For instance, if there are three classes, "dog", "cat", and "bird", then one-hot encoded label associated to the class "dog" is $[1, 0, 0]$. For a

[1]Market Research Future `https://www.marketresearchfuture.com/reports/machine-learning-market-2494`

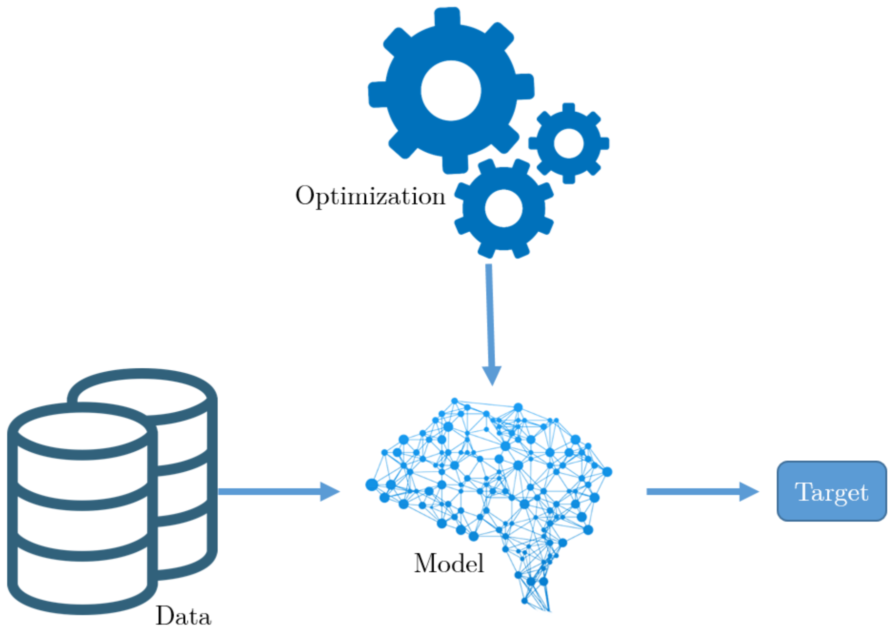


Figure I.2: Illustration of a ML task. After fitting a model to data (with or without annotation) via an optimization process, the model outputs a prediction or decision for an input sample which is not in data.

given input sample, the model actually returns a probability vector, for instance $[0.8, 0.15, 0.05]$. Then the sample is assigned to the class that is given the strongest probability, in our example the "dog" class. Let d denote a dissimilarity metric, $f()$ a model that can map a sample to a classes probability vector, (x_i, y_i) a sample/label couple and N_l the number of training samples. The loss function $\mathcal{L}$ to be optimized for a classification task is defined as:

$$\mathcal{L} = \frac{1}{N_l} \sum_{i=1}^{N_l} d(f(x_i), y_i) \tag{I.1}$$

Different SL approaches for classification have been developed and become standard over years, from linear models such as Linear Discriminant Analysis, Support Vector Machine (SVM) [Vapnik *et al.* 1992] to non-linear models such as Decision Trees, k-Nearest Neighbors, Naive Bayes or Logistic Regression... One can add to these standard approaches, **ensemble learning** techniques which consist in combining several weak predictive models into a stronger aggregated predictive model. Two important families among these techniques are Boosting [Schapire 1999, Chen & Guestrin 2016] and Bagging [Bühlmann 2004].

Reinforcement learning can be considered to be part of SL, as the model is optimized to output a user provided target at each state during the training phase. For example, in playing chess, machine is learnt to perform action that optimizes the reward at each state and relations between

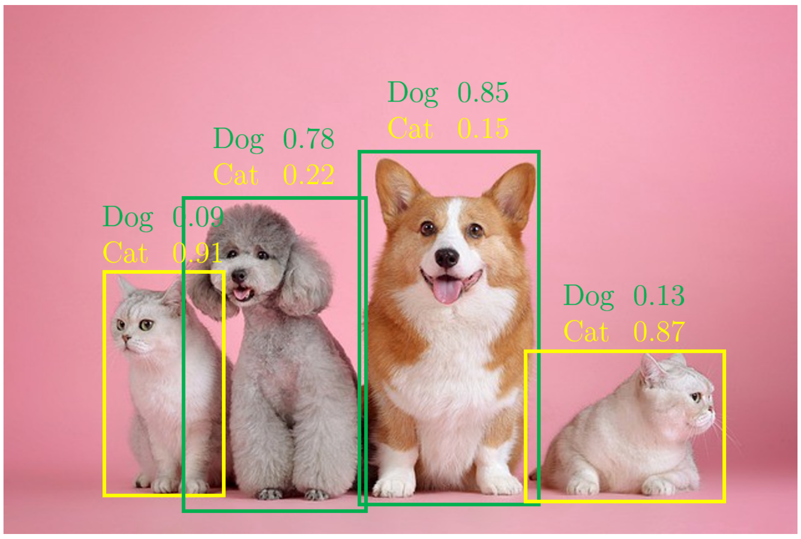


Figure I.3: Image recognition task.

actions and rewards are considered as user provided targets.

On the contrary, under the unsupervised learning paradigm, one aims at eliciting meaningful structures solely based on data samples. It is the point treated in the next section.

Unsupervised learning. (UL) In contrast to supervised learning, this learning paradigm does not require labels. It rather aims at uncovering assumed regularities in the data. UL can be achieved through clustering or representation learning.

Clustering consists in partitioning samples into different groups (clusters) in such a way that samples in the same group are more similar to each other than to those in other groups. Clustering can be considered as a classification where the classes has to be discovered. The most popular clustering technique is certainly the k-means algorithm [MacQueen 1967]. On the connexionnist side, we can cite the self-organizing maps (SOM). However, these techniques operate directly in the samples ambient space, which might contain useless features in regard to a particular application. This problem can be addressed by representation learning techniques.

Representation learning (also called feature extraction) encompasses methods that tries to capture relevant information in the samples, given a specific task or prior knowledge of the data. This information is ceased through computed features that constitute the new or latent representation of the original samples. In general, the number of features in latent representation is much smaller than the one of original representation. Among representation learning techniques, we can cite Dictionary Learning, Manifold Learning Techniques but also Auto-Encoder neural networks, Self-Supervised Learning [Zisserman 2018].

Beyond the clustering and representation learning techniques, we can mention active learning from robotics which is about unsupervised mechanisms set up to push an autonomous agent toward discovery maximizing situations [Oudeyer *et al.* 2007].

It is worth noting that labels can also be used for clustering and representation learning when they are partly available. Hence Semi-Supervised Learning paradigm.

Semi-supervised learning. (SSL) In this learning paradigm, one makes use of both labelled and unlabelled data during training. The core idea is that structural information that can be extracted from the samples in an unsupervised manner can complement labels information in the learning task. For instance, one of the earliest semi-supervised classification methods, label spreading [Zhu & Ghahramani 2002], relies on ones ability to find structured communities in the dataset, considered as a network. Thus, each community is affected to a class, based only on a few labeled examples.

Models trained in a SSL fashion can often achieve the performance of their SL counterparts with smaller amount of labelled data, thanks to relevant structural information extracted from a large amount of data available. SSL can be put into perspective with the Weakly Supervised Learning methodology in which model training relies on inexpensive "weak" or inaccurate labels, with different mechanisms to account for labels uncertainty [Ratner *et al.* 2019].

C Motivations

In this thesis, we focus on SSL, which appears to be a growth research area for different reasons.

Annotation cost. In SL, annotation is primordial. The amount of annotated samples need to be large enough to give an acceptable accuracy. However, annotating samples is a costly work, which at times even requires expertise (medical diagnostic). Annotation task can vary from simple tasks such as labeling object in an image to more complicated tasks such as locating object by a box, segmenting object. . . SSL offers the perspective to alleviate this burden.

Scientific perspectives. One can consider that the SL field is scientifically and technically mature, in the sense that when sufficient one has at disposal computationally efficient algorithms and affordable architectures to reach critical level of performances, provided that a sufficient amount of labelled data are available. On the contrary, SSL still offers a great deal of challenges along with promising research tracks that we will try to convey through our contributions.

Availability of unlabelled data. Additionally, important volumes of unlabelled data are virtually always available. This calls for research investments into UL and SSL, which in turn drives data owning corporations interest.

Feasibility. On a more fundamental level, we see SSL at play everytime a new born progressively learns to categorize and interpret the different stimuli from his environment. Biological learning systems generally display an impressive capacity to generalize knowledge from only few examples. This indicates that efficient SSL is at least algorithmically feasible.

D Manuscript outlines

In this thesis, we develop one SSL approach based Dictionary Learning (DiL) and another in the Deep Learning framework (DeL).

Here is the outline and resume for each chapter:

- **Chapter II: Methodological pillars**

  In this chapter, we present the building blocks on which our approaches rely, namely Dictionary Learning model and Convolutional Neural Network model and Manifold Learning.

- **Chapter III: Semi-supervised dictionary learning**

  We present our first contribution on semi-supervised dictionary learning (SSDL).

- **Chapter IV: Semi-supervised deep learning**

  We present our second contribution on semi-supervised neural network (SSNN).

Finally, we present our conclusions and perspectives in **Chapter V**.

Chapter II

Methodological pillars

Chapter abstract

In this chapter, we give a general overview of Dictionary Learning (DiL) and their extension to classification tasks. Then, we introduce Convolutional Neural Network (CNN) models. Finally, we remind manifold learning.

A Dictionary learning

Dictionary Learning (DiL) encompasses methods and algorithms which, for a given type of signal, aim at deriving a set of cardinal features which enables one to concisely describe signals of this type. Precisely, given a set of vectors $(\mathbf{x}_j)_{1\leq j\leq N}$ in $\mathbb{R}^n$ that represent signals (Electrocardiography, genes sequences...), DiL techniques compute a set vectors $(\mathbf{d}_i)_{1\leq i\leq p}$ in $\mathbb{R}^n$ and a set of coefficients $(\mathbf{a}_j)_{1\leq j\leq N}$ in $\mathbb{R}^p$ so that:

$$\mathbf{x}_j \approx \sum_{i=1}^{p} \mathbf{a}_j[i]\mathbf{d}_i \tag{II.1}$$

with a constraint of sparsity on the vectors $\mathbf{a}_j$. This approximation is illustrated by figure II.1.

As for the terminology:

- the matrix $\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_p]$ is the dictionary.
- its columns vectors $(\mathbf{d}_i)$ are the atoms.
- the vectors $\mathbf{a}_j$ are the sparse codes as they represent each of the samples.

Hence, the learnt dictionary is specifically adapted for describing signals that are structurally similar to the training samples. The benefit of such dictionaries in sparsity-driven signal recovery

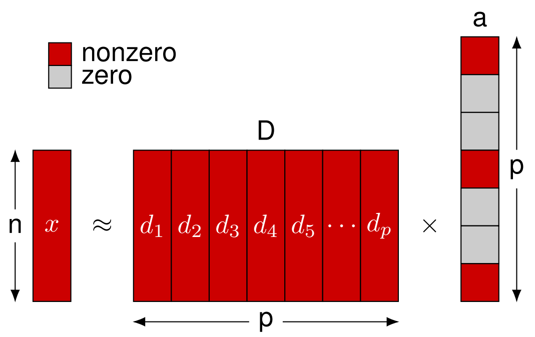


Figure II.1: Illustration of sparse representation. Here we approximate signal x by only 3 atoms.

has been shown in several applications (see for example [Elad & Aharon 2006]).

A dictionary and the associated sparse codes is generally learnt by alternating between two optimization steps: **sparse coding** and **dictionary update**. In this section, we present some optimization methods for each step.

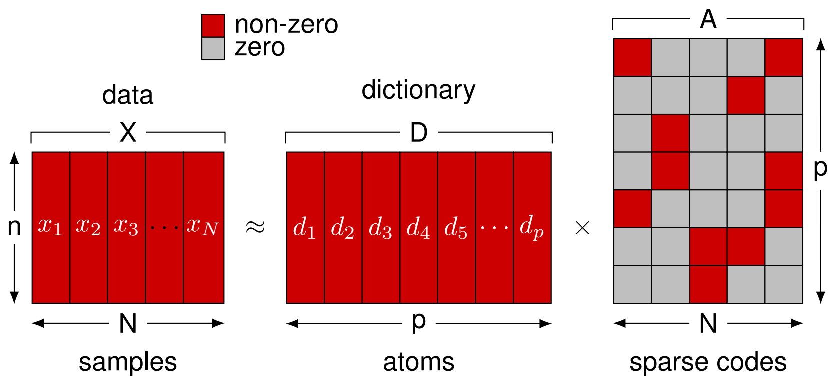

non-zero
zero
data
X
n
x_1 x_2 x_3 $\cdots$ x_N
N
samples
$\approx$
dictionary
D
d_1 d_2 d_3 d_4 d_5 $\cdots$ d_p
p
atoms
$\times$
A
p
N
sparse codes

Figure II.2: Illustration of sparse dictionary learning task. We try to approximate signals $\mathbf{X}$ by $\mathbf{DA}$, where $\mathbf{D}$ is a base or dictionary and $\mathbf{A}$ is sparse code of $\mathbf{X}$, which means each column $\mathbf{A}[:,i]$ is sparse code of sample $\mathbf{x}_i$.

A.1 Sparse coding. Given a dictionary $\mathbf{D} \in \mathbb{R}^{n\times p}$, which contains p atoms, $\mathbf{d}_i \in \mathbb{R}^n$ $(i = 1, ..., p)$, we compute a signal $\mathbf{x} \in \mathbb{R}^n$ sparse code by solving $\mathbf{a} \in \mathbb{R}^p$ by solving the following optimization problem:

$$\min_{\mathbf{a}} \|\mathbf{x} - \mathbf{Da}\|_F^2 \text{ s.t. } \|\mathbf{a}\|_0 \leq k, \tag{II.2}$$

where $k \in \mathbb{N}^*$ sets the maximum number of non-zero elements in $\mathbf{a}$. The signal $\mathbf{x}$ is approximated by a linear combination of at most k atoms, *i.e.* its reconstruction $\hat{\mathbf{x}} = \mathbf{d}_1\mathbf{a}[1] + \cdots + \mathbf{d}_p\mathbf{a}[p]$ has only k non-zero coefficients $\mathbf{a}[i]$. We impose $k \ll n$, which means that we look for an approximation of $\mathbf{x}$ by only a small number of atoms in the dictionary $\mathbf{D}$.

Usually, one seeks for a redundant or over-complete dictionary, which means the number of atoms is far higher than the number of original features, *i.e.* $p \gg n$. This makes possible to find a small number of atoms in $\mathbf{D}$ that are sufficient to approximate the sample $\mathbf{x}$.

Solving the problem II.2 exactly would have a prohibitive computational cost because it is a NP-hard problem. Thus, one has to settle for an approximate resolution. Two main types of algorithm can be distinguished. On the one hand, we have greedy algorithms such as Matching Pursuit (MP) [Mallat & Zhang 1993], Orthogonal Matching Pursuit (OMP) [Pati *et al.* 1993], Regularized Orthogonal Matching Pursuit (ROMP) [Needell & Vershynin 2007]. In short, MP finds iteratively the most correlated atom with the residue among unused atoms until having exactly k atoms. OMP is a variant of MP which updates the sparse codes associated with the previously selected atoms at each iteration. ROMP reconstructs the sparse representation of signal based on Restricted Isometry Property. One the other hand, we have iterative shrinkage algorithms among which the well known Iterative Hard Thresholding (IHT) algorithm [Blumensath & Davies 2008]. IHT is a gradient-based method in which hard thresholding operator plays the role of a projector.

One can make use of other sparsity promoting penalties than the l_0 pseudo-norm, such as the so-called l_q, norms defined as

$$\|\mathbf{x}\|_q = (\sum_{i=1}^{n}(\mathbf{x}[i])^q)^{\frac{1}{q}} \tag{II.3}$$

where $q \in (0, 1)$. Let focus on the important case of the l_1 norm, that is found in the Lasso problem formulation [Tibshirani 1996]. The sparse coding problem can be written into three equivalent forms:

$$\min_{\mathbf{a}} \|\mathbf{x} - \mathbf{D}\mathbf{a}\|_F^2 \text{ s.t. } \|\mathbf{a}\|_1 \leq \mu, \tag{II.4}$$

where $\mu \in \mathbb{R}$.

$$\min_{\mathbf{a}} \frac{1}{2} \|\mathbf{x} - \mathbf{D}\mathbf{a}\|_2^2 + \lambda \|\mathbf{a}\|_1 , \tag{II.5}$$

where $\lambda > 0$.

$$\min_{\mathbf{a}} \|\mathbf{a}\|_1 \text{ s.t. } \|\mathbf{x} - \mathbf{D}\mathbf{a}\|_F^2 \leq \varepsilon. \tag{II.6}$$

With l_1 norm, the sparse coding problem is convex and can be solved in polynomial time. Several algorithms can be used including Coordinate descent (see an overview [Wright 2015]), Fixed-point continuation [Hale *et al.* 2007] and Proximal gradient [Combettes & Pesquet 2009] which are fundamentally subgradient descent methods. In a different approach, feature-sign search algorithm [Lee *et al.* 2007] proposes to search for the optimal sign of each element in sparse code in order to remove the l_1 norm's absolute values, then solves for a quadratic problem with linear constraints. Gradient projection for sparse reconstruction [Figueiredo *et al.* 2007] proposes to replace sparse code $\mathbf{a}$ with two positive terms $\mathbf{u}, \mathbf{v}$, with $\mathbf{a} = \mathbf{u} - \mathbf{v}$. One justifies that the problem II.5 can be rewritten as:

$$\min_{\substack{\mathbf{u},\mathbf{v}\in\mathbb{R}^p \\ \mathbf{u},\mathbf{v}\geq\mathbb{0}}} \frac{1}{2}\|\mathbf{x} - \mathbf{D}(\mathbf{u} - \mathbf{v})\|_2^2 + \lambda(\sum_{i=1}^{p}\mathbf{u}[i] + \sum_{i=1}^{p}\mathbf{v}[i]) \tag{II.7}$$

This quadratic problem with the linear constraints ($\mathbf{u}, \mathbf{v} \geq \mathbb{0}$) is then solved by a gradient-based method with backtracking line search for tuning gradient descent step. [Kim *et al.* 2007] proposes another constrained quadratic form of the problem II.5:

$$\min_{\substack{\mathbf{a},\mathbf{u}\in\mathbb{R}^p \\ -\mathbf{u}[i]\leq\mathbf{a}[i]\leq\mathbf{u}[i]}} \|\mathbf{x} - \mathbf{Da}\|_2^2 + \lambda\sum_{i=1}^{p}\mathbf{u}[i] \tag{II.8}$$

In fact, the problem II.8 is equivalent to the problem II.5 and is solved using an Interior-Point Method (also referred to as *barrier methods*).

[Efron *et al.* 2004] propose Least Angle Regression (LARS), which can be considered a variant of OMP with l_1 norm in the formulations II.4 and II.6 of the lasso problem. The signal $\mathbf{a}$ is initialized by $\mathbb{0}$ (in $\mathbb{R}^p$) and progressively increased in the direction created by a set of active atoms, to reduce the residue $\mathbf{r} = \mathbf{x} - \mathbf{Da}$. The new active atom is added in the set if it has as much correlation with the residue as each existing active atom has.

The most well-known applications of sparse representation are compressed sensing and signal recovery. For instance, if a signal is corrupted, we first transform it to the frequency domain by the Fourier transform. Then we reconstruct its frequency representation by sparse coding with hypothesis that signals are sparse in the frequency domain. The dictionary $\mathbf{D}$ in this case is Fourier basis functions. The recovered signal is obtained from the reconstructed frequency representation by the inverse Fourier transform. The construction of the dictionary follows two strategies. The first one called analytical dictionaries are designed for certain classes of signal. Wavelets (Haar 1909, [Meyer 1993, Starck *et al.* 2007, Mallat 2008]), Discrete Cosine Transform (DCT) [Ahmed *et al.* 1974], ridgelets [Candès 1998], curvelets [Candès & Donoho 2000, Starck *et al.* 2002] are examples of analytical dictionaries. Signal decomposition and reconstruction through these dictionaries benefit fast implicit transforms which is a considerable advantage from a practical standpoint. On the other hand, the learned dictionaries are optimized according to a specific dataset. Therefore, they describe better suited for these data compared to the first more generic ones. However, they are in general unstructured so that signals analysis using these dictionaries involves explicit matrices products that can be computationally costly for large dictionaries. In next subsection, we present optimization methods to learn a dictionary from a given dataset.

A.2 Dictionary update. [Olshausen & Field 1996] were the first to propose a way to learn the dictionary from data and to insist on the redundancy of dictionary. Given a matrix $\mathbf{X} \in \mathbb{R}^{n\times N}$ that contains N signals $\mathbf{x}_i \in \mathbb{R}^n$, *i.e.* $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_N]$ and the matrix $\mathbf{A} \in \mathbb{R}^{p\times N}$ contains sparse code $\mathbf{a}_i \in \mathbb{R}^p$ of signal $\mathbf{x}_i$, *i.e.* $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, .., \mathbf{a}_N]$, the dictionary computation problem is given by:

$$\min_{\mathbf{D}\in\mathcal{C}} \|\mathbf{X} - \mathbf{DA}\|_2^2 , \tag{II.9}$$

where $\mathcal{C} = \{\mathbf{D}, \|\mathbf{d}_i\|_2 \leq \alpha, \forall i = 1, 2, ..., p\}$ is a subset of $\mathbb{R}^{n\times p}$ that contains all dictionaries whose atoms l_2 norms are less or equal to a given α (generally set to 1).

The problem II.9 is convex. We note $\Pi_{\mathcal{C}}$ the projection of a point onto the convex set $\mathcal{C}$. For solving this problem, one can use the projected gradient method, with the projector $\Pi_{\mathcal{C}}$.

Method of optimal directions (MOD) [Engan *et al.* 1999] proposes to solve II.9 using the first order optimality condition without constraint $\mathbf{D} \in \mathcal{C}$. Then the optimum is projected using $\Pi_{\mathcal{C}}$. MOD is simple but does not give the optimal solution to the problem II.9. [Lee *et al.* 2007] propose to use Lagrange duality, solving the following problem:

$$\begin{aligned} &\underset{\gamma_1,..,\gamma_p}{\text{maximize}} \quad \inf_{\mathbf{D}} \left(\|\mathbf{X} - \mathbf{DA}\|_F^2 + \sum_{i=1}^{p} \gamma_i (\|\mathbf{d}_i\|_2^2 - \alpha) \right) \\ &\text{subject to} \quad \gamma_i \geq 0, \forall i = 1, \ldots, p. \end{aligned} \tag{II.10}$$

with fixed γ, the infimum is obtained using first order optimality condition *w.r.t.* $\mathbf{D}$, then Newton's method is applied to solve for maximizing. Finally, K-SVD [Aharon *et al.* 2006] is a revolutionary method in DiL for its simplicity and efficacy. It differs from other mentioned methods in the way it couples the dictionary $\mathbf{D}$ and the sparse codes $\mathbf{A}$ update. The algorithm K-SVD updates for each $k = 1, .., p$ the couple $(\mathbf{d}_k, \mathbf{A}[k, \Omega_k])$ by assuming that other atoms are fixed:

$$(\mathbf{d}_k, \mathbf{A}[k, \Omega_k]) = \underset{\|\mathbf{d}\|_2=\alpha, \mathbf{a}\in\mathbb{R}^{|\Omega_k|}}{\text{argmin}} \|\underbrace{\mathbf{X}[:, \Omega_k] - \sum_{j\neq k} \mathbf{d}_j \mathbf{A}[j, \Omega_k]}_{\mathbf{E}_k} - \mathbf{d}\mathbf{a}^\top\|_F^2, \tag{II.11}$$

where $\Omega_k = \{i|\mathbf{A}[k, i] \neq 0\}$, $\mathbf{A}[k, :]$ being the the k^{th} line of sparse codes matrix. Then, the matrix $\mathbf{E}_k = \mathbf{X}[:, \Omega_k] - \sum\limits_{j\neq k} \mathbf{d}_j \mathbf{A}[j, \Omega_k]$ is decomposed using the singular value decomposition (SVD):

$$\mathbf{E}_k = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top , \tag{II.12}$$

where $r \leq \min(n, |\Omega_k|)$ and $\sigma_1 \geq \sigma_2 \geq .. \geq \sigma_r \geq 0$. $\mathbf{E}_k$ is approximated with the term $\sigma_1 \mathbf{u}_1 \mathbf{v}_1^\top$ to approximate $\mathbf{E}_k$. Hence, the k^{th} dictionary atom and the corresponding sparse codes are identified as $\mathbf{d}_k = \alpha \mathbf{u}_1$ and $\mathbf{A}[k, \Omega_k] = \frac{1}{\alpha}\sigma_1 \mathbf{v}_1$.

Besides, online dictionary learning (ODL) methods appeared for data stream processing. These methods allow dictionary updates solely based on one or few signals at the time. In most papers, two supplement matrix: $\mathbf{B}_t = \mathbf{B}_{t-1} + \mathbf{x}_t \mathbf{a}_t^\top$ and $\mathbf{C}_t = \mathbf{C}_{t-1} + \mathbf{a}_t \mathbf{a}_t^\top$ are required to stock only necessary information at the time t for updating dictionary, therefore previous signals and their sparse codes do not need to be saved. [Mairal *et al.* 2009a] propose to update each atom while fixing other ones until the convergence. [Skretting & Engan 2010] propose Recursive Least Squares Dictionary Learning Algorithm (RLS-DLA) that updates directly all atoms at once time, with a forgetting factor for former signals (exponentially). Apparently, RLS-DLA is fast since it does not solve for the optimal solution with the constraint $\mathbf{D} \in \mathcal{C}$. A survey for optimizing DiL problem can be found in [Mairal *et al.* 2014] or [Zhang *et al.* 2015].

A.3 Conclusion on dictionary learning . Most DiL involve a sparse coding step and a dictionary update. Figure II.2 shows an illustration of DiL, which corresponds to the following objective function:

$$\min_{\mathbf{D}\in\mathcal{C},\mathbf{A}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_2^2 + \lambda \|\mathbf{A}\|_q \,, \tag{II.13}$$

where $0 \leq q \leq 1$.

Beyond this formalism, DiL methods rely also on the Bayesian inference. For instance, [Zhou *et al.* 2009] proposes a non-parametric approach in which the number of atoms or sparse penalty parameter are optimization variables as well as the dictionary and the atoms. The optimization is performed using the Beta process [Hjort 1990]. However, as nature of non-parametric models, this dictionary learning model can become more and more complicated with an increasing amount of training data.

As aforementioned, traditional DiL methods produce unstructured dictionaries in that there are no relationships between atoms. However, structured DiL methods have been proposed. For instance, [Jenatton *et al.* 2010] proposed a tree-structured dictionary learning, where l_0 or l_1 norms are replaced with Hierarchical Sparsity-Inducing Norm in the optimization process.

For completeness sake, let mention an interesting variant of DiL as presented so far, which is analysis dictionary learning [Rubinstein *et al.* 2013, Tang *et al.* 2019]:

$$\min_{\mathbf{\Omega}\in\mathcal{C}',\mathbf{U}} \|\mathbf{U} - \mathbf{\Omega}\mathbf{X}\|_2^2 + \lambda \|\mathbf{U}\|_q \tag{II.14}$$

where $\mathcal{C}'$ is a set that constraints $\mathbf{\Omega}$. The atoms in an analysis dictionary $\mathbf{\Omega}$ can be interpreted as filters, which try to activate relevant features.

Globally speaking, DiL leads to state-of-the-art results in many applications such as denoising [Elad & Aharon 2006], colorization [Mairal *et al.* 2008], super resolution [Yang *et al.* 2010a], data clustering [Elhamifar & Vidal 2012], and in specific fields such as medical imaging [Gangeh *et al.* 2010], astronomy [Ngolè Mboula *et al.* 2014], *etc.* DiL also plays the role of representation learning, which is a step in the pipeline of bag-of-words (BoW) models in Computer Vision [Fei-Fei & Perona 2005]. Recall that BoW models consist of four steps: descriptor extraction, representation learning, spatial pooling and classification. In the first versions of BoW, representation learning is performed by Vector Quantization (K-Means [MacQueen 1967], DBSCAN [Ester *et al.* 1996] or other clustering algorithms). After that, Vector Quantization is replaced by DiL, which has shown an improvement for BoW models [Jianchao Yang *et al.* 2009, Coates & Ng 2011].

We now turn to supervised dictionary learning (SDL) applications.

B Supervised dictionary learning

As previously, DiL yields a new data representation, the sparse codes, which are optimized so that one can get an accurate sparse description of the samples. Driven by the ML trend, several approaches have been developed by applying dictionary learning in classification tasks. In this section, we present how this can be done in supervised manners.

Let $\mathbf{X}^l$ denote the labelled data from C classes. $\mathbf{X}^l$ has N_l samples $\mathbf{x}_i^l$ $(i = 1,..,N_l)$. Each sample $\mathbf{x}_i$ is associated with label $\mathbf{y}_i = [y_i^1, y_i^2, ..., y_i^C]^\top$, where:

$$y_i^j = \begin{cases} 1 \text{ if sample } i \text{ belongs to } j^{th} \text{ class} \\ -1 \text{ otherwise.} \end{cases}$$

$\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{N_l}]$ is the label matrix for all labelled samples, $\mathbf{Y} \in \mathbb{R}^{C\times N_l}$. $\mathbf{A}^l$ denote the sparse codes of labelled data $\mathbf{X}^l$. $\mathbf{W} \in \mathbb{R}^{C\times p}$ is a linear classifier consisting of C binary classifiers (with the strategy "one vs all") in the sparse code space, $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_C]^\top$ and $\mathbf{b} = [b_1, b_2, ..., b_C]^\top$ denotes the associated bias. SDL can be generically formalized as follows:

$$\begin{gathered} \min_{\Theta} \left[\mathcal{R}(\mathbf{A}^l, \mathbf{D}) + \mathcal{D}(\mathbf{W}, \mathbf{b}, \mathbf{A}^l, \mathbf{D}) \right], \\ \text{where } \Theta = \{\mathbf{W}, \mathbf{b}, \mathbf{A}^l, \mathbf{D} \in \mathcal{C}\}. \end{gathered} \tag{II.15}$$

$\mathcal{R}$ denotes the reconstruction error with the sparsity constraint. The penalty $\mathcal{D}$ aims at making the dictionary $\mathbf{D}$ or the sparse codes $\mathbf{A}$ discriminative, possibly including an internal classifier $(\mathbf{W}, \mathbf{b})$ learning loss. Linear classifier $\mathbf{W}$ is described as internal in the sense that it is jointly learned with the dictionary and the sparse codes during the optimization process.

In the next subsections, we present SDL methods by making sparse codes discriminative then SDL methods by making atoms discriminative.

B.1 SDL with internal classifier. This type of SDL trains a classifier in the sparse code space during the training process. Hence, in prediction stage, an unlabelled sample has to be firstly sparse coded with the learnt dictionary in order that the learnt internal classifier to be applied. Here is the general objective function on which almost SDL with internal classifier approaches rely:

$$\begin{array}{cc} \textbf{Objective function} & \textbf{Prediction} \\ \min_{\mathbf{W},\mathbf{A}^l,\mathbf{D}\in\mathcal{C}} \underbrace{\left\|\mathbf{X}^l - \mathbf{D}\mathbf{A}^l\right\|_F^2 + \lambda\left\|\mathbf{A}^l\right\|_q}_{\mathcal{R}} + \underbrace{\gamma\left\|\mathbf{Y} - \mathbf{W}\mathbf{A}^l\right\|_F^2}_{\mathcal{D}} & \underset{c}{\operatorname{argmax}}\ \mathbf{w}_c^\top \mathbf{a}^u \end{array} \tag{II.16}$$

The simplest SDL method is probably DK-SVD proposed by [Zhang & Li 2010], which solves the following problem:

$$\begin{aligned} \min_{\substack{\mathbf{A}^l,\mathbf{W},\mathbf{D} \\ \|\mathbf{d}_i\|_2=1}} \quad & \left\|\mathbf{X}^l - \mathbf{D}\mathbf{A}^l\right\|_F^2 + \gamma\left\|\mathbf{Y} - \mathbf{W}\mathbf{A}^l\right\|_F^2 \\ \text{subject to} \quad & \|\mathbf{a}_i\|_0 \le T_0, \forall i = 1,\ldots,N_l. \end{aligned} \tag{II.17}$$

We can rewrite the objective function II.17 by piling $\left\|\mathbf{X}^l - \mathbf{D}\mathbf{A}^l\right\|_F^2$ on $\left\|\mathbf{Y} - \mathbf{W}\mathbf{A}^l\right\|_F^2$:

$$\begin{aligned} \min_{\substack{\mathbf{A}^l,\mathbf{W},\mathbf{D} \\ \|\mathbf{d}_i\|_2=1}} \quad & \left\|\begin{bmatrix}\mathbf{X}^l \\ \sqrt{\gamma}\ \mathbf{Y}\end{bmatrix} - \begin{bmatrix}\mathbf{D} \\ \sqrt{\gamma}\ \mathbf{W}\end{bmatrix}\mathbf{A}^l\right\|_F^2 \\ \text{subject to} \quad & \|\mathbf{a}_i\|_0 \le T_0, \forall i = 1,\ldots,N_l. \end{aligned} \tag{II.18}$$

Then the problem II.18 can be solved using the K-SVD algorithm. Inspired by DK-SVD, [Jiang *et al.* 2013] propose label consistent K-SVD (LCK-SVD) by adding an other quadratic loss into the objective function, in order to encourage samples belonging to the same classes to be described by the same subset of atoms, i.e. to have common sparse codes supports. To do so, an atoms labels matrix $\mathbf{Q} \in \mathbb{R}^{p\times N_l}$ is defined as $\mathbf{Q}[i,j] = 1$ if j^{th} training sample and i^{th} atom are in the same class, 0 otherwise, yielding the following optimization problem:

$$\begin{aligned} \min_{\substack{\mathbf{A}^l,\mathbf{W},\mathbf{U},\mathbf{D} \\ \|\mathbf{d}_i\|_2=1}} \quad & \left\|\mathbf{X}^l - \mathbf{D}\mathbf{A}^l\right\|_F^2 + \gamma\left\|\mathbf{Y} - \mathbf{W}\mathbf{A}^l\right\|_F^2 + \eta\left\|\mathbf{Q} - \mathbf{U}\mathbf{A}^l\right\|_F^2 \\ \text{subject to} \quad & \|\mathbf{a}_i\|_0 \le T_0, \forall i = 1,\ldots,N_l. \end{aligned} \tag{II.19}$$

where $\mathbf{U} \in \mathbb{R}^{p\times p}$ is a linear transformation matrix to optimize. Thus, this method also makes the dictionary atoms discriminative. As DK-SVD, the objective function II.19 of LCK-SVD can be rewritten as:

$$\min_{\substack{\mathbf{A}^l,\mathbf{W},\mathbf{U},\mathbf{D}\\ \|\mathbf{d}_i\|_2=1}} \left\| \begin{bmatrix} \mathbf{X}^l \\ \sqrt{\eta}\ \mathbf{Q} \\ \sqrt{\gamma}\ \mathbf{Y} \end{bmatrix} - \begin{bmatrix} \mathbf{D} \\ \sqrt{\eta}\ \mathbf{U} \\ \sqrt{\gamma}\ \mathbf{W} \end{bmatrix} \mathbf{A}^l \right\|_F^2 \quad \text{subject to} \quad \|\boldsymbol{\alpha}_i\|_0 \leq T_0, \forall i = 1, \dots, p, \tag{II.20}$$

and solved using K-SVD algorithm. [Mairal *et al.* 2009b] proposed two SDL methods, respectively named SDL-G and SDL-D:

$$\min_{\mathbf{D}\in\mathcal{C},\mathbf{A}^l,\theta_j} \sum_{i=1}^{N_l} \Big(\underbrace{\mathcal{B}(y_i^j f(\mathbf{a}_i^l, \theta_j)) + \lambda_0 \left\|\mathbf{x}_i^l - \mathbf{D}\mathbf{a}_i^l\right\|_2^2 + \lambda_1 \left\|\mathbf{a}_i^l\right\|_1}_{\mathcal{S}(\mathbf{a}_i^l,\mathbf{x}_i^l,\mathbf{D},\theta_j,y_i^j)} \Big) + \lambda_2 \left\|\theta_j\right\|_2^2, \tag{SDL-G}$$

where $\mathcal{B}(x) = log(1 + e^{-x})$ is the logistic function and $f(\mathbf{a}_i^l, \theta_j) = \mathbf{w}_j^\top \mathbf{a}_i^l + b_j$ where $\theta_j = [\mathbf{w}_j^\top, b_j]$ is a binary classifier for class j. Given dictionary $\mathbf{D}$ and classifier θ_j, the *supervised sparse coding* loss $\mathcal{S}^\star(\mathbf{x}_i^l, \mathbf{D}, \theta_j, y_i^j)$ for sample $\mathbf{x}_i^l$ relatively to class j is defined as:

$$\mathcal{S}^\star(\mathbf{x}_i^l, \mathbf{D}, \theta_j, y_i^j) = \min_{\mathbf{a}} \mathcal{S}(\mathbf{a}, \mathbf{x}_i^l, \mathbf{D}, \theta_j, y_i^j), \tag{II.21}$$

where we also take into account of classifier j compared to regular sparse coding. Hence, to classify a test sample $\mathbf{x}$, its sparse code is computed for each classifier θ_c, and then the sample is assigned to the optimal class as follows:

$$\operatorname*{argmin}_{c} \mathcal{S}^\star(\mathbf{x}, \mathbf{D}, \theta_c, 1). \tag{II.22}$$

SDL-D is an extension of SDL-G on dictionary update, which aims at making the model more discriminative, solving the following problem:

$$\min_{\mathbf{D}\in\mathcal{C},\theta_j} \sum_{i=1}^{N_l} \Big(\mu\mathcal{B}\big(\mathcal{S}^\star(\mathbf{x}_i^l, \mathbf{D}, \theta_j, -y_i^j) - \mathcal{S}^\star(\mathbf{x}_i^l, \mathbf{D}, \theta_j, y_i^j)\big) + (1-\mu)\mathcal{S}^\star(\mathbf{x}_i^l, \mathbf{D}, \theta_j, y_i^j) \Big) + \lambda_2 \left\|\theta_j\right\|_2^2, \tag{SDL-D}$$

where μ balances the additional discriminative penalty with respect to the SDL-G form. Note SDL-D boils down to SDL-G when $\mu = 0$. Compared to SDL-G, first, SDL-D is rather a dictionary update problem than dictionary learning problem as SDL-G because only the dictionary $\mathbf{D}$ is variable to be optimized. Second, SDL-D optimizes the dictionary $\mathbf{D}$ in order to make a better discrimination between false prediction $(-y_i^j)$ and true prediction (y_i^j).

SDL-G and SDL-D have shown impressive performances on multiple benchmark (digits recognition, texture. . .). This can be explained the use of a non-linear classifier, with the logistic function $\mathcal{B}(x)$, which is rather similar to activation functions in DeL field.

B.2 SDL with atoms discriminative. This category relies on the relationship between atoms and class labels: a class-specific sub-dictionary is learnt for each class. Hence, an unlabelled sample is classified by reconstruction error given by each class-specific sub-dictionary. Here is the general objective function on which almost SDL with atoms discriminative approaches rely:

$$\begin{array}{cc} \textbf{Objective function} & \textbf{Prediction} \\ \min_{\mathbf{A}^l_{c,c},\mathbf{D}\in\mathcal{C}} \underbrace{\sum_{c=1}^{C}\left(\left\|\mathbf{X}^l_c - \mathbf{D}_c\mathbf{A}^l_{c,c}\right\|_F^2 + \lambda\left\|\mathbf{A}^l_{c,c}\right\|_q\right)}_{\mathcal{R}\text{ and }\mathcal{D}} & \operatorname*{argmin}_{c} \left\|\mathbf{x}^u - \mathbf{D}_c\mathbf{a}^u\right\|_2 \end{array} \tag{II.23}$$

where $\mathbf{D}_c$ is a class-specific sub-dictionary reserved for class c, $\mathbf{X}^l_c$ contains all samples of class c and $\mathbf{A}^l_{c,c}$ is sparse code of $\mathbf{X}^l_c$ on class-specific sub-dictionary $\mathbf{D}_c$.

[Wright *et al.* 2009] proposed Sparse Representation-based Classification (SRC), which uses all training samples as the dictionary $\mathbf{D}$. There is no optimization process in this approach to learn the dictionary. A test sample is first sparse coded on $\mathbf{D}$ and classified by reconstruction error given by each class-specific sub-dictionary $\mathbf{D}_c$, which contains all training samples of class c. However, dictionary size in SRC could be large if the number of training samples is large. In order to obtain a smaller dictionary and make atoms more discriminative, [Yang *et al.* 2010b] proposed to learn class-specific sub-dictionaries $\mathbf{D}_c$ for each class c. [Ramirez *et al.* 2010] extended the form II.23 by penalizing cross-correlations between sub-dictionaries, yielding Dictionary Learning with Structured Incoherence (DLSI):

$$\min_{\mathbf{A}^l_{c,c},\mathbf{D}\in\mathcal{C}} \sum_{c=1}^{C}\left(\left\|\mathbf{X}^l_c - \mathbf{D}_c\mathbf{A}^l_{c,c}\right\|_F^2 + \lambda\left\|\mathbf{A}^l_{c,c}\right\|_1\right) + \eta\sum_{c=1}^{C}\sum_{j=c+1}^{C}\left\|\mathbf{D}_c^\top\mathbf{D}_j\right\|_F^2 \tag{II.24}$$

Indeed, training samples can have shared features such as background or in the multiclass setting, they can belong to more than one class, which can induce undesirable correlations between class-specific sub-dictionaries. Therefore, [Kong & Wang 2012] extended furthermore DLSI by introducing in addition a synthesized sub-dictionary that captures trans-classes features. This approach called DL-COPAR (COmmonality and PARticularity) solves the following problem:

$$\begin{aligned}\min_{\substack{\mathbf{A}^l,\mathbf{D}\\ \|\mathbf{d}_i\|_2=1}} \sum_{c=1}^{C}\Big(&\left\|\mathbf{X}_c^l - \mathbf{D}\mathbf{A}_c^l\right\|_F^2 + \lambda\left\|\mathbf{A}_c^l\right\|_1 + \sum_{\substack{j=1\\ j\neq c}}^{C}\left\|\mathbf{A}_{c,j}^l\right\|_F^2 \\ &+ \left\|\mathbf{X}_c^l - \mathbf{D}_c\mathbf{A}_{c,c}^l - \mathbf{D}_{C+1}\mathbf{A}_{c,C+1}^l\right\|_F^2\Big) + \eta\sum_{c=1}^{C}\sum_{j=c+1}^{C}\left\|\mathbf{D}_c^\top\mathbf{D}_j\right\|_F^2\end{aligned} \tag{II.25}$$

In this approach, $\mathbf{D} = [\mathbf{D}_1, .., \mathbf{D}_C, \mathbf{D}_{C+1}]$, where $\mathbf{D}_{C+1}$ contains the common features of all classes and $\mathbf{A}_c^l = [\mathbf{A}_{c,1}^{l\top}, .., \mathbf{A}_{c,C+1}^{l\top}]^\top$. Unlike DLSI which performs sparse coding on each class-specific sub-dictionary, DL-COPAR performs sparse coding on the whole dictionary. For all training samples of class c, the coefficients firstly coded in a class-specific sub-dictionary j $(j \neq c)$ are forced to zero by minimizing $\left\|\mathbf{A}_{c,j}^l\right\|_F^2$. Secondly, these samples are represented by using its class-specific sub-dictionary on one hand and the synthesized sub-dictionary on the other hand.

[Yang *et al.* 2014] proposed an approach called Fisher Discrimination based Dictionary Learning (FDDL), which tries to minimize the intra-class energy and maximize inter-class energy:

$$\begin{aligned}\min_{\substack{\mathbf{A}^l,\mathbf{D}\\ \|\mathbf{d}_i\|_2=1}} \sum_{c=1}^{C}\Big(&\left\|\mathbf{X}_c^l - \mathbf{D}\mathbf{A}_c^l\right\|_F^2 + \left\|\mathbf{X}_c^l - \mathbf{D}_c\mathbf{A}_{c,c}^l\right\|_F^2 + \sum_{\substack{j=1\\ j\neq c}}^{C}\left\|\mathbf{D}_j\mathbf{A}_{c,j}^l\right\|_F^2 + \lambda_1\left\|\mathbf{A}_c^l\right\|_1 \\ &+ \lambda_2\big(\left\|\mathbf{A}_c^l - \mathrm{M}(\mathbf{A}_c^l)\right\|_F^2 - N_l^c\left\|\mathrm{m}(\mathbf{A}_c^l) - \mathrm{m}(\mathbf{A}^l)\right\|_2^2\big)\Big) + \eta\left\|\mathbf{A}^l\right\|_F^2,\end{aligned} \tag{II.26}$$

where N_l^c denotes the number of samples in class c, $\mathrm{m}(\mathbf{Z})$ is the column mean for matrix $\mathbf{Z}$ and $\mathrm{M}(\mathbf{Z})$ is the matrix with the same size as $\mathbf{Z}$ by repeating column $\mathrm{m}(\mathbf{Z})$. $\left\|\mathbf{A}_c^l - \mathrm{M}(\mathbf{A}_c^l)\right\|_F^2$ represents within-class energy and $N_l^c\left\|\mathrm{m}(\mathbf{A}_c^l) - \mathrm{m}(\mathbf{A}^l)\right\|_2^2$ represents between-class energy. The penalty $\eta\left\|\mathbf{A}^l\right\|_F^2$ guarantees that sparse coding is convex since we introduce a negative term for between-class energy: $-N_l^c\left\|\mathrm{m}(\mathbf{A}_c^l) - \mathrm{m}(\mathbf{A}^l)\right\|_2^2$.

B.3 Conclusion about SDL. Labels can be incorporated into classical DiL in other to learn discriminative dictionaries or sparse codes in different manners. This can be done by using an internal classifier to make sparse codes discriminative or by defining class-specific sub-dictionaries to make atoms discriminative. Note that, we can apply both of these two techniques, *e.g.* LCK-SVD by problem II.19. Interestingly, having a look at sparse codes and corresponding active atoms might provide simple interpretation to classification decision based upon this modelling. The interested reader can find a review of SDL in [Gangeh *et al.* 2015].

C Deep Learning

Conventionally, only in this thesis, we understand Deep Learning as deep Neural Network Model (NNM) or Convolutional Neural Network (CNN) model because Deep Learning can have other deep architectures which differ from Neural Network Model.

C.1 Neural networks. Let $1 \leq l \leq L$ denote index of layer in a deep neural network and let $n^{(l)}$ denote the number of features of hidden representation $z^{(l)}$, which means $z^{(l)} \in \mathbb{R}^{n^{(l)}}$. In layer (l), $z^{(l)}$ is computed as follows:

$$z^{(l)} = \sigma(z^{(l-1)}W^{(l)} + b^{(l)}) \tag{II.27}$$

where σ: $\mathbb{R}^{n^{(l)}} \rightarrow \mathbb{R}^{n^{(l)}}$ is an element-wise activation function, for adapting the non-linearity of data. $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ and $b^{(l)} \in \mathbb{R}^{n^{(l)}}$ are model parameters which represent for a Fully Connected layer. Figure II.3 shows an illustration for a neural network model.

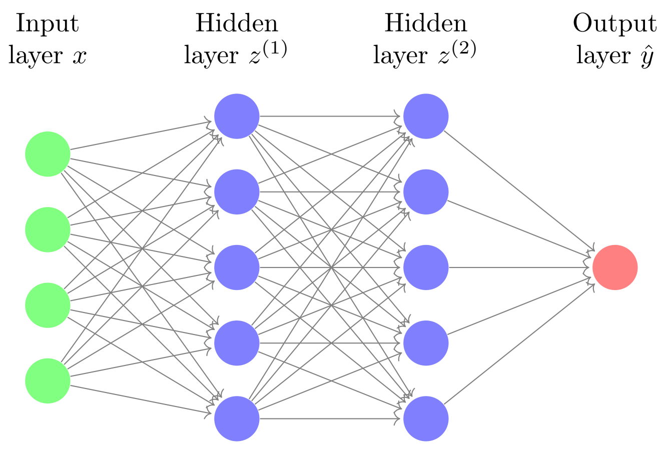


Figure II.3: A simple neural network model with three layers.

C.2 Convolutional neural networks. Since a vanilla neural network model by equation II.27 is composed only of Fully Connected layers, it has several following shortcomings. Firstly, units in a signal vector or pixels in an image are independent, which means do not take into account the specific topological structure (here the locality) among these units or pixels. Secondly, for a deep vanilla NNM, using Fully Connected layers increases quickly the number of model parameters, which can lead to an over-fitting problem. Finally, Fully Connected layer (as filter) is not robust to some transformations, such as translation. In order to overcome the above limitations, [Fukushima 1980] proposed Convolutional Neural Network (CNN) and over the years, it was improved and refined in by [Lecun *et al.* 1998]. For a simple and general view, we present CNN in case of images (especially square images).

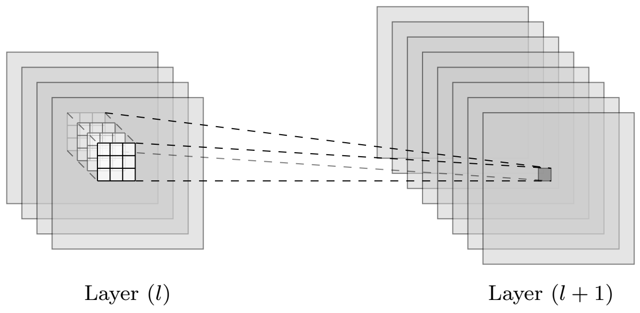


Figure II.4: Illustration for a convolutional operator in CNNs. Input layer (l) has dimension 4x16x16, output layer $(l+1)$ has dimension 8x16x16 and 8 convolutional filters, each of them has dimension 4x3x3. Source: developed from [Lanusse *et al.* 2017].

Let consider a square hidden representation: $z^{(l)} \in \mathbb{R}^{K_{(l)} \times H_{(l)} \times H_{(l)}}$, where $K_{(l)}$ means the number of channels, $H_{(l)}$ means both height and width in the layer (l). A convolutional layer with parameters $(W^{(l)}, b^{(l)})$ is thus a transition between layer (l) and $(l+1)$. $W^{(l)}$ consists of $K_{(l+1)}$ square convolutional filters, each filter has size of $K_{(l)} \times I_{(l)} \times I_{(l)}$ where $I_{(l)} \times I_{(l)}$ is also called kernel size. In short, $W^{(l)} \in \mathbb{R}^{K_{(l+1)} \times K_{(l)} \times I_{(l)} \times I_{(l)}}$. The bias is represented by $b^{(l)} \in \mathbb{R}^{K_{(l+1)}}$. After performing $K_{(l+1)}$ convolutions, the output $z^{(l+1)}$ has $K_{(l+1)}$ channels or shortly $z^{(l+1)} \in \mathbb{R}^{K_{(l+1)} \times H_{(l+1)} \times H_{(l+1)}}$. Here is the details of the operations performed in a convolutional layer:

$$z^{(l+1)}[k,:,:] = \sigma\big(\sum_{h=1}^{K_{(l)}} W^{(l)}[k,h,:,:] * z^{(l)}[h,:,:] + b^{(l)}[k]\big), \tag{II.28}$$

where $k = 1, .., K_{(l+1)}$. Figure II.4 shows an illustration for a convolutional operator in CNNs. In first CNN models, the architecture consists of convolutional layers and Max-Pooling layers (alternatively) at the beginning, then a Flatten layer followed by a Fully Connected layers at the end (figure II.5).

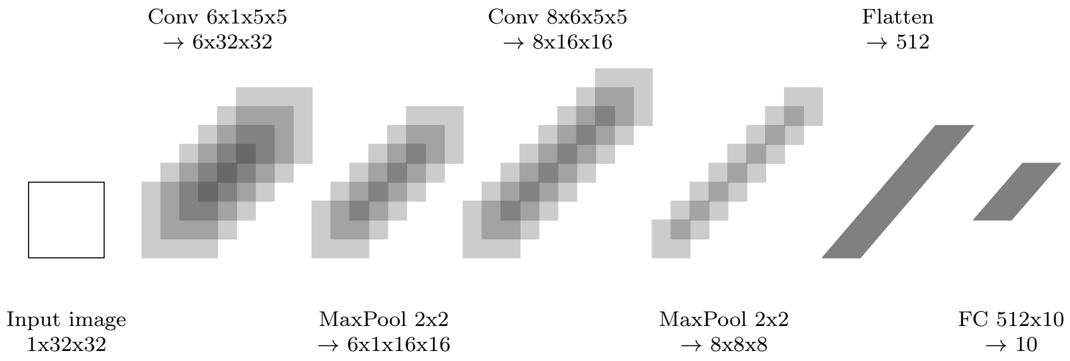


Figure II.5: An example inspired from LeNet architecture. Source: developed from [Stutz 2016].

In practice, the size of conventional images is often hundreds times hundreds pixels (even more in some specific fields) and the number of categories (classes) can reach to the thousands, instead of 32×32 with 10 categories in figure II.5. Therefore, we need to increase the capabilities of CNN models, usually by going deeper or by widening the number of channels. Hence we obtain architectures with higher complexity. One can think of broadening kernel sizes. However, this may lead very quickly to an immense number of model parameters and may not be efficient for conventional images. For NNMs and also CNNs that have a high complexity, two classical problems need to be considered:

- **Vanishing gradient.** This problem relates to deep architecture models that use gradient-based methods as optimizer. In this case, backward gradients in upstream layers are very small, which implies that parameters in these layers are almost fixed during the training stage. For example, activation functions such as tanh or sigmoid have their derivatives in the range $(-1, 1)$ so that when backward gradients are computed using the chain rule, they get fainter as one moves towards upstream layers.

- **Over-fitting.** This problem relates to neural network models that have a large number of model parameters compared to number of training samples. Although training set and test set are sampled from the same data distribution, a model may display a low training loss and a high testing loss.

Solutions for the vanishing gradient problem though various CNN architectures will be detailed in section II.C.3 (Standard CNNs). The over-fitting problem is tackled through various regularization strategies:

- Through the data, *e.g.* data input augmentation (rotation, translation, flip, color transformations,...). A review of data augmentation field can be found in [Shorten & Khoshgoftaar 2019]. The regularization might also be applied to the outputs using techniques such as Label Smoothing [Szegedy *et al.* 2015] that proposed to modify slightly the target, *e.g.* from [1,0,0] to [0.8,0.1,0.1]. Instead of manually modifying the target label, an other technique to perform Label Smoothing is to slightly maximize entropy of predicted output [Pereyra *et al.* 2017]. Note that, Label Smoothing is applied only for labelled samples.

- Through the model parameters, *e.g.* weights decay [Krogh & Hertz 1992] or other strategies which are presented in section IV.A.2.

- Through the optimization scheme, *e.g.* Early Stopping [Morgan & Bourlard 1990, Prechelt 1997].

C.3 Standard CNNs. In this subsection, we present some popular CNN architectures, which are considered as the backbone of many applications. Figure II.6 gives a simple illustrations for these CNN architectures.

AlexNet. This is an architecture proposed by [Krizhevsky *et al.* 2012] that revives CNN after the work of [Lecun *et al.* 1998] (LeNet). It has several improved points compared to LeNet. First, AlexNet is deeper and has much more number of parameters than LeNet (60M vs 60K). The training has been made possible using GPUs. Second, AlexNet uses the non-saturating ReLU activation function [Nair & Hinton 2010]: $\max(0, x)$, which showed more efficient than tanh or sigmoid function. This is because the derivative of ReLU is 1 if $x > 0$ instead of a value in $(-1, 1)$ as derivatives of tanh or sigmoid. Thus ReLU is better to deal with the vanishing gradient problem and provides even a faster learning. In addition, ReLUs is sparse if $x \leq 0$ and sparse representations seem to be more efficient for regularization than dense representations.

VGG. This approach [Simonyan & Zisserman 2014] aims at making improvements over AlexNet. First, it is deeper and larger than AlexNet (about 2 times in term of number of parametric layers and in term of number of parameters). Second, large kernel sizes (11×11 or 5×5) in somes first convolutional layers of AlexNet is replaced by multiple 3×3 ones, which helps to learn better the texture of data.

Inception. Also known as GoogleNet [Szegedy *et al.* 2014a], this architecture appeared almost at the same time as VGG. Again, it aims at making improvements over AlexNet by several following significant modifications. Firstly, 1×1 convolutional filters is applied to reduce the number of channels before applying larger convolutional filters. For a short comparison, let take an example that we want to pass from a hidden representation $z^{(l)}$ of 32 channels to the next one $z^{(l+1)}$ of 64 channels with 3×3 convolutional filters, then we need $W^{(l)}$ of size $64 \times 32 \times 3 \times 3 = 18432$ parameters. On the contrary, if we apply 1×1 convolutional filters, we firstly pass from $z^{(l)}$ of 32 channels to an intermediate representation of 16 channels by $W_0^{(l+1)}$ of size $16 \times 32 \times 1 \times 1 = 512$ parameters. Then we pass from the intermediate representation of 16 channels to $z^{(l+1)}$ of 64 channels by $W_1^{(l+1)}$ of size $64 \times 16 \times 3 \times 3 = 9216$ parameters. Consequently, we need only 9728 parameters in this case instead of 18432 in the previous case and this reduces considerably the number of parameters.

Secondly, inception modules contain various kernel sizes (1×1, 3×3 and 5×5) in parallel, which reinforces for the learning of texture. Inception has many other variants that are described in [Szegedy *et al.* 2015] and [Szegedy *et al.* 2016].

ResNet. To efficiently handle the vanishing gradient problem, ResNet [He *et al.* 2015] was created by adding additive skip operators into CNN architecture, *e.g.* a forward pass through a

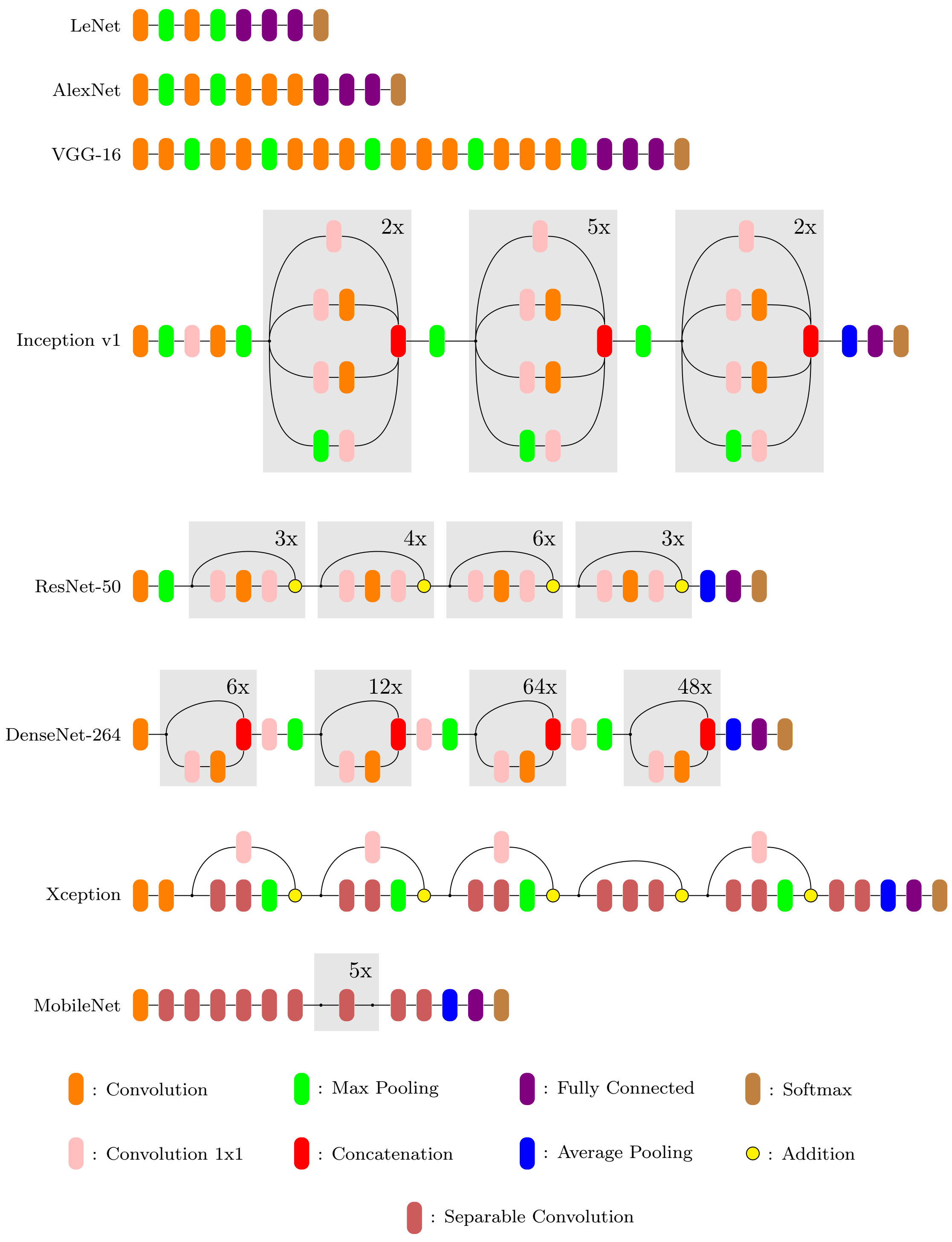


Figure II.6: Illustration for several standard CNN architectures. For convolution layers, they can have different parameter such as kernel size, stride or zero padding. Gray rectangles mean to repeat exactly the included module. Source: inspired by [Alemi 2016].

series of Residual modules that can be illustrated by:

$$z_0^{(0)} \prod_{j=0}^{N^m} (1 + \prod_{i=1}^{N_j^l} W_i^{(j)}), \tag{II.29}$$

where $z_0^{(0)}$ is the first layer of the first Residual module. $W_i^{(j)}$ can be seen as parameters of layer i in Residual module j. N_j^m and N_m are respectively the number of layers in module j and the number of Residual modules. For the first layer in a Residual module, besides backward gradient comes from a series of multiplications through intermediate layers, we have also backward gradient transferred directly to this first layer thanks to an additive skip operator. Hence, we can avoid the vanishing gradient. In NLP field, Long short-term memory (LSTM) [Hochreiter & Schmidhuber 1997] also uses additive operators to solve for the vanishing gradient problem. ResNet has 5 versions: 18, 34, 50, 101 and 152 layers. In the deeper versions (the last three), Residual modules have a bottleneck architecture, which means a main convolutional layer of kernel size 3×3 is sandwiched between two auxiliary convolutional layers of kernel size 1×1, which helps to control the number of channels. We call them bottleneck architecture because in these Residual modules, the number of channels in the input and the number of channels in the output are larger than the one in the main convolutional layer of kernel size 3×3.

DenseNet. As ResNet, DenseNet [Huang *et al.* 2016] aim also at handling the vanishing gradient problem. The fundamental difference between these two architectures is that DenseNet uses Dense modules with concatenation operators, instead of additive skip operators as in ResNet. Therefore, output of a Dense module contains feature maps of the first layer and also of all intermediate layers. It is very important to note that DenseNet always takes advantage of additive operators to avoid the vanishing gradient problem. Thus, additive operators come from the 1×1 convolutional layer right after a Dense module. These 1×1 convolutional layers help to sum up (with weights) all concatenated feature maps.

Consequently, backward gradients are transferred to all layers in a module, not just for the first layer as in ResNet. Hence, DenseNet alleviates even further the vanishing-gradient problem. Besides, for the same number of layers, DenseNet has less parameters than ResNet: given a convolutional filter $W^{(l)} \in \mathbb{R}^{K_{(l+1)} \times K_{(l)} \times I_{(l)} \times I_{(l)}}$, both ResNet and DenseNet apply $I_{(l)} = 3$, but there is a considerable difference on $K_{(l+1)} \times K_{(l)}$ between these two approaches. DenseNet has 4 versions: 121, 169, 201 and 264 layers.

Xception. The novelty of this architecture [Chollet 2016] is to introduce Separable Convolutional layers, which consists of channel-wise convolutions (also called depth-wise convolutions) followed by a point-wise convolutions (1×1 convolutional filters) and eventually an activation function. In the channel-wise convolution, each channel is treated independently:

$$z^{(l+1)}[k,:,:] = W^{(l)}[k,:,:] * z^{(l)}[k,:,:] + b^{(l)}[k], \tag{II.30}$$

where now $W^{(l)} \in \mathbb{R}^{K_{(l)} \times I_{(l)} \times I_{(l)}}$, hence the input and output of this operator have the same number of channels. In order to modify the number of channels from $K_{(l)}$ to $K_{(l+1)}$, a point-wise convolution is applied. Thus, Separable Convolutional layers has much less parameters than their corresponding convolutional layers. This is the main advantage compared to other CNNs.

Xception paved the way designing "light" models, which are compatible for learning on portative devices, such as cellphones. Thus, Xception propelled Federated Learning field, which aims at training a model across multiple decentralized devices.

MobileNet. MobileNets [Howard *et al.* 2017] consist of a series of Separable Convolutional layers and aim at alleviating the model as much as possible. MobileNets do not even need additive operators to deal with the vanishing gradient problem because they are not really deep (28 layers including 13 point-wise convolution layers). MobileNets use stride in Separable Convolution layers to reduce size of feature maps (height and width) instead of Max-Pooling layers. From a basic architecture of MobileNet, two factors are introduced to reduce more the number of parameters: width multiplier α and resolution R. Firstly, width multiplier means that for a given Separable Convolutional layer, the number of input channels $K_{(l)}$ becomes $\alpha K_{(l)}$ and the number of output channels $K_{(l+1)}$ becomes $\alpha K_{(l+1)}$. Secondly, all height and width of feature maps are multiplied by $\rho = R/224$ since 224 is the resolution of MobileNet basic architecture, R being the input resolution. Hence, name for a MoblileNet has the following form "α MobileNet-R". Note that, resolution changing does not impact the number of model parameters in CNN architectures, but it impacts the computational cost and the required memory to stock hidden representations.

Table II.1 shows several standard CNN architecture with top5-error on the standard dataset ImageNet [Deng *et al.* 2009], which is a project designed for visual object recognition. For now, about ten million images have been hand-annotated and eventually bounding boxes (for object location) are also provided.

Add-on modules

In addition to the CNN architectures, there are also several add-on modules which help to enhance the performance of a CNN architecture. We present some popular add-on modules in this subsection.

Dropout. [Srivastava *et al.* 2014] introduced Dropout which can be considered as an auxiliary

Network	**Year**	**top5-error**	**Number of parametric layers**	**Number of parameters**
LeNet	1998		5	60K
AlexNet	2012	16.4%	8	62.3M
VGG-16	2014	8.1%	16	138.3M
Inception v1	2014	10.1%	22	5M
Inception v3	2015	5.6%	48	23M
ResNet-50	2015	6.7%	50	25.8M
ResNet-101	2015	6.0%	101	40M
ResNet-152	2015	5.7%	152	60.3M
DenseNet-264	2016	6.1%	264	73M
Xception	2016	5.5%	71	22.8M
1.0 MobileNet-224	2017	10.5%	28	4.2M

Table II.1: top5-error on ImageNet and number of parameters by several standard CNN architectures. Source: `https://paperswithcode.com/sota/image-classification-on-imagenet`.

layer and it consists in dropping out (setting to 0) random units in a hidden representation $z^{(l)}$. Dropout acts as a regularization because it increases the number of samples. Thus, this process can be considered as data augmentation in hidden representations. Besides, if Dropout is applied for a Fully Connected layer, we can consider that Dropout deactivates randomly an amount of parameters in this layer instead of units in hidden representations, because the role of an unit $z[i]$ and the role of a parameter $W[j]$ are the same, to make $z[i]W[j] = 0$. However, this idea can not be applied for convolutional layers, where the role of an unit $z[i]$ and the role a parameter $W[j]$ are not the same for convolutional operators, which is illustrated in [Reinhold 2019].

In practice, Dropout is widely applied for Fully Connected layers but it is often less effective for convolutional layers ([Ghiasi *et al.* 2018]). Therefore, these authors proposed another version of Dropout, called DropBlock, which drops out randomly blocks instead of units in feature maps to deal with convolutional layers.

Normalization. We present Batch Normalization (BN) proposed by [Ioffe & Szegedy 2015]. Given a batch $\mathcal{B}$ that has m samples x_i, these samples are normalized as:

$$\begin{aligned}
\mu_{\mathcal{B}} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i && \text{(batch mean)}\\
\sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 && \text{(batch variance)}\\
\hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && \text{(normalize)}\\
y_i &= \gamma \hat{x}_i + \beta && \text{(scale and shift)},
\end{aligned}$$

where γ, β are trainable parameters of BN layer and $\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}$ are non-trainable parameters. Here are several advantages of BN:

- BN mitigates "covariate shift" effect which refers to changes in the distribution of the inputs of each layer.
- BN smooths the objective function [Santurkar *et al.* 2019].
- BN is beneficial in regard of the vanishing gradient problem. Indeed, when used immediately before non-linearity layers such as sigmoid or tanh, BN modifies the input distribution bringing the values closer to 0. This results in greater derivatives values, which mitigates vanishing gradient problem.

In the case that a BN layer succeeds a Fully Connected layer (n dimensions), BN is applied unit-wise and there are $4n$ parameters. Otherwise, if a BN layer succeeds a convolutional layer ($H \times W \times K$ dimensions), BN is applied feature map-wise. This reduces the number of parameters to $4K$ instead of $4HWK$.

In evaluation mode, given a BN layer, $(\mu_{\mathcal{B}}^{ev}, \sigma_{\mathcal{B}}^{ev})$ is required to perform the forward pass through this layer for a single test sample. Therefore, it needs to be estimated during the training to be used afterward. For example, EMA with momentum α might by used to update $(\mu_{\mathcal{B}}^{ev}, \sigma_{\mathcal{B}}^{ev})$ as:

$$(\mu_{\mathcal{B}}^{ev}, \sigma_{\mathcal{B}}^{ev}) \leftarrow (1-\alpha)(\mu_{\mathcal{B}}^{ev}, \sigma_{\mathcal{B}}^{ev}) + \alpha(\mu_{\mathcal{B}}^{tr}, \sigma_{\mathcal{B}}^{tr}), \tag{II.31}$$

where $(\mu_{\mathcal{B}}^{tr}, \sigma_{\mathcal{B}}^{tr})$ are obtained after each forward batch during the training.

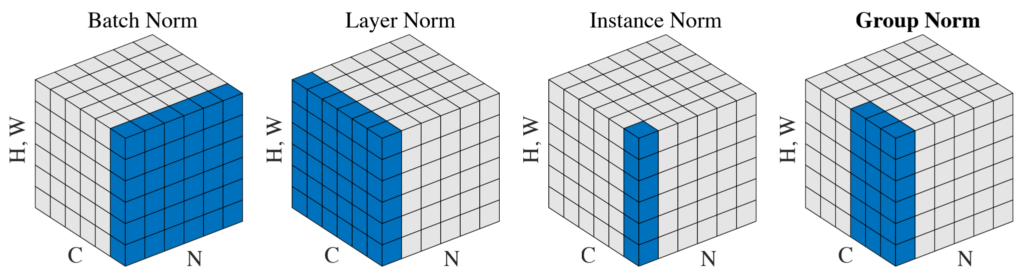


Figure II.7: Different normalization methods. N, C and (H, W) means respectively for batch axis, channel axis and spatial axes. Blue blocks indicate units (pixels) to be normalized by the same mean and variance. Source: [Wu & He 2018].

Although BN has shown impressive performances for very deep architectures, it has several drawbacks such as using a small batch size in training, multi-GPUs training or compatibility for RNNs (Recurrent Neural Network). Other normalization methods are developed to overcome these drawbacks. For examples, Weight Normalization [Salimans & Kingma 2016] proposed to normalizes model parameters instead of feature maps. Layer Normalization [Ba *et al.* 2016] proposes to normalize feature maps across all features instead of samples. Instance Normalization [Ulyanov *et al.* 2016] proposed to normalize across features in a channel. Group Normalization [Wu & He 2018] is a mix of layer normalization and instance normalization, proposed to normalize across features in a group of channels. Figure II.7 shows position of units to be normalized using the same mean and variance in different normalization methods. An overview and more normalization methods can be found in [Kurita 2018].

Squeeze and Excitation. This add-on module [Hu *et al.* 2017] aims at biasing the contribution of channels in feature maps (figure II.8). As suggested by its name, this module has two stages: squeeze and excitation (SE). Squeeze consists of a Global Pooling layer and excitation is a little bottleneck architecture which consists of two Fully Connected layers with activation. Given feature maps of a layer, SE computes a scalar weight for each channel, then multiplies each channel by its weight. The mechanism used in SE is thus a gate mechanism. The later is applied in general for units of a representation and not necessarily channels as in SE. We can take for example the units in each hidden state vector in LSTM [Hochreiter & Schmidhuber 1997].

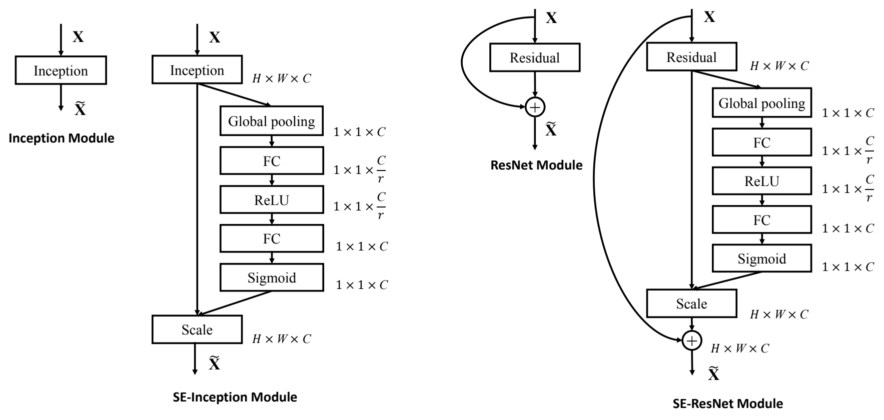


Figure II.8: Squeeze and Excitation implemented in Inception module (left) and Residual module (right). Only in these figures, H, W, C means for height, width and number of channels. Source: [Hu *et al.* 2017].

It is worth noting that there is also attention mechanism (appeared a long time ago but firstly introduced in deep learning by [Bahdanau *et al.* 2014]). Gate and attention mechanisms both use trainable parameters to weight different units, but there is just a slight difference between them. Attention mechanism is global and aggregating while gate mechanism is applied to each elementary unit.

Holistic networks

As each single architecture has different advantages and shortcomings, we quickly go through several architectures that consists of at least two single architectures. These hybrid architectures are expected to give better performances or acceptable performances with the minimum possible number of parameters. For example, Inception-ResNet [Szegedy *et al.* 2016]; MobileNet v2 [Sandler *et al.* 2018] that uses modules inspired from Residual modules (ResNet) but replacing normal convolution layer with Separation Convolution (figure II.9 stride = 1); MnasNet [Tan *et al.* 2018] goes a little further by using in addition add-on modules such as SE and BN to form its own elementary modules (MBConv); EfficientNet [Tan & Le 2019] is a scalable network, which means from a reference architecture composed of MBConv layers, the number of layers, number of channels and number of feature in a feature maps are modified to get a new architecture.

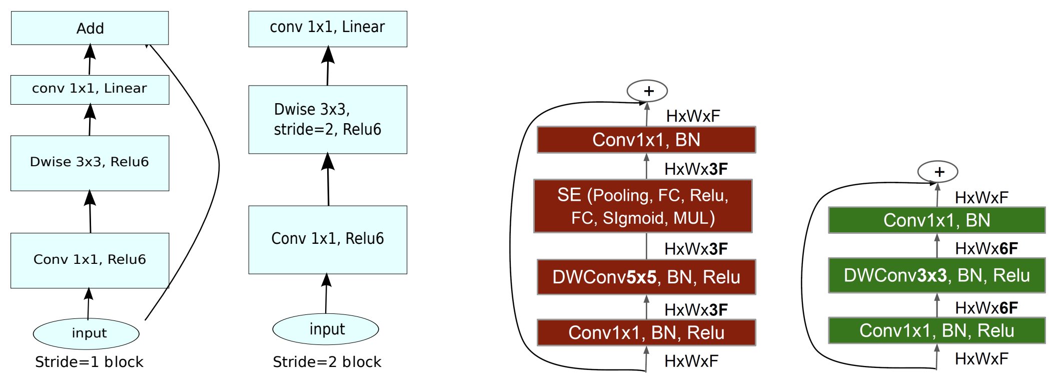


Figure II.9: Left: Modules used in MobileNet v2 with stride = 1 and stride = 2. Relu6 means max(0,6). Source: [Sandler *et al.* 2018]. Center: MBConv3 (5x5) used in MnasNet. Right: MBConv6 (3x3) used in MnasNet. Source: [Tan *et al.* 2018].

Conclusion about CNN architectures

There are several from now on standard existing architectures available for the practitioner. If the CNN architecture is complex, several issues have to be dealt with including the vanishing gradient problem, the over-fitting problem, the training time and computational cost. A recent trend consists in replacing classical convolutional layers (equation II.28) with channel-wise convolutions (or depth-wise convolutions, see equation II.30) and point-wise convolutions, which reduces dramatically the model's complexity, so that these models can be trained on computationally limited devices. We precisely turn to CNN's optimization methods in the next section.

C.4 Optimizers. In order to optimize objective function for NNMs and also CNNs, gradient-based algorithms are almost the only choice. Table II.2 shows popular gradient-based optimizers.

Momentum takes into account the prior update step Δ_{t-1} to accelerate and dampens oscillations (caused by narrow local optimas). Nesterov accelerated gradient (NAG) uses anticipatory gradient instead of current gradient to regularize update steps. However, Momentum and NAG apply the same learning rate (step size) η for all parameters. On the contrary, Adagrad weights the learning rate η by using the sum of the squares of the gradients up to time t. RMSprop goes a little further by using EMA instead of normal sum, in order to avoid quickly decreasing learning rate. Adadelta replaces fixed learning rate η with an EMA of the squares of the update steps up to time t. Finally, Adam and its variants AdaMax, Nadam, AMSGrad preserve the learning η but replacing gradients $g_{t,i}$ with an EMA of gradients. Depending on each method, there are other slight improvements. An overview of gradient-based optimizers can be found in [Ruder 2017].

Algorithms	**Description**	**Update**
Momentum [Qian 1999]	$\Delta_t = \gamma\Delta_{t-1} + \eta\nabla_\theta J(\theta)$	$\theta \leftarrow \theta - \Delta_t$
Nesterov accelerated gradient [Nesterov 1983]	$\Delta_t = \gamma\Delta_{t-1} + \eta\nabla_\theta J(\theta - \gamma\Delta_{t-1})$	$\theta \leftarrow \theta - \Delta_t$
Adagrad [Duchi *et al.* 2011]	$g_{t,i} = \nabla_\theta J(\theta_{t,i})$ $G_{t,i} = \sum_{k=1}^{t} g_{k,i}^2$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}+\epsilon}} g_{t,i}$
RMSprop [Hinton *et al.*]	$G_{t,i} = 0.9G_{t-1,i} + 0.1g_{t,i}^2$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}+\epsilon}} g_{t,i}$
Adadelta [Zeiler 2012]	$G_{t,i} = \gamma G_{t-1,i} + (1-\gamma)g_{t,i}^2$ $\Delta_{t,i} = \theta_{t+1,i} - \theta_{t,i}$ $D_{t,i} = \gamma D_{t-1,i} + (1-\gamma)\Delta_{t,i}^2$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\sqrt{D_{t-1,i}+\epsilon}}{\sqrt{G_{t,i}+\epsilon}} g_{t,i}$
Adam [Kingma & Ba 2014]	$m_{t,i} = \gamma_1 m_{t-1,i} + (1-\gamma_1)g_{t,i}$ $G_{t,i} = \gamma_2 G_{t-1,i} + (1-\gamma_2)g_{t,i}^2$ $\hat{m}_{t,i} = \frac{m_{t,i}}{1-\gamma_1^t}$ and $\hat{G}_{t,i} = \frac{G_{t,i}}{1-\gamma_2^t}$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{G}_{t,i}+\epsilon}} \hat{m}_{t,i}$
AdaMax [Kingma & Ba 2014]	$u_{t,i} = \max(\gamma_2 u_{t-1,i}, \|g_{t,i}\|)$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{u_{t,i}} \hat{m}_{t,i}$
Nadam [Dozat 2016]	$\hat{m}_{t,i} = \frac{\gamma_1 m_{t,i}}{1-\gamma_1^{t+1}} + \frac{(1-\gamma_1)g_{t,i}}{1-\gamma_1^t}$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{G}_{t,i}+\epsilon}} \hat{m}_{t,i}$
AMSGrad [Reddi *et al.* 2019]	$\hat{G}_{t,i} = \max(\hat{G}_{t-1,i}, G_{t,i})$	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{G}_{t,i}+\epsilon}} \hat{m}_{t,i}$

Table II.2: Various gradient-based optimizers for neural networks. AdaMax, Nadam, AMSGrad are variant of or developed from Adam.

D Manifold learning

Manifold learning is a part of the unsupervised representation learning, which consists of both linear (PCA [Pearson 1901], ICA [Jutten & Herault 1991]) and nonlinear methods. For a high-dimensional data, *i.e.* the original representation of sample has many features. It can be difficult to interpret and visualize the global organization of the data. We assume that, for a given task, the data of interest lies on a low dimensional manifold. Thus, computing data coordinates with respect to this manifold can ease their visualization. Figure II.10 show an illustration of manifold learning.

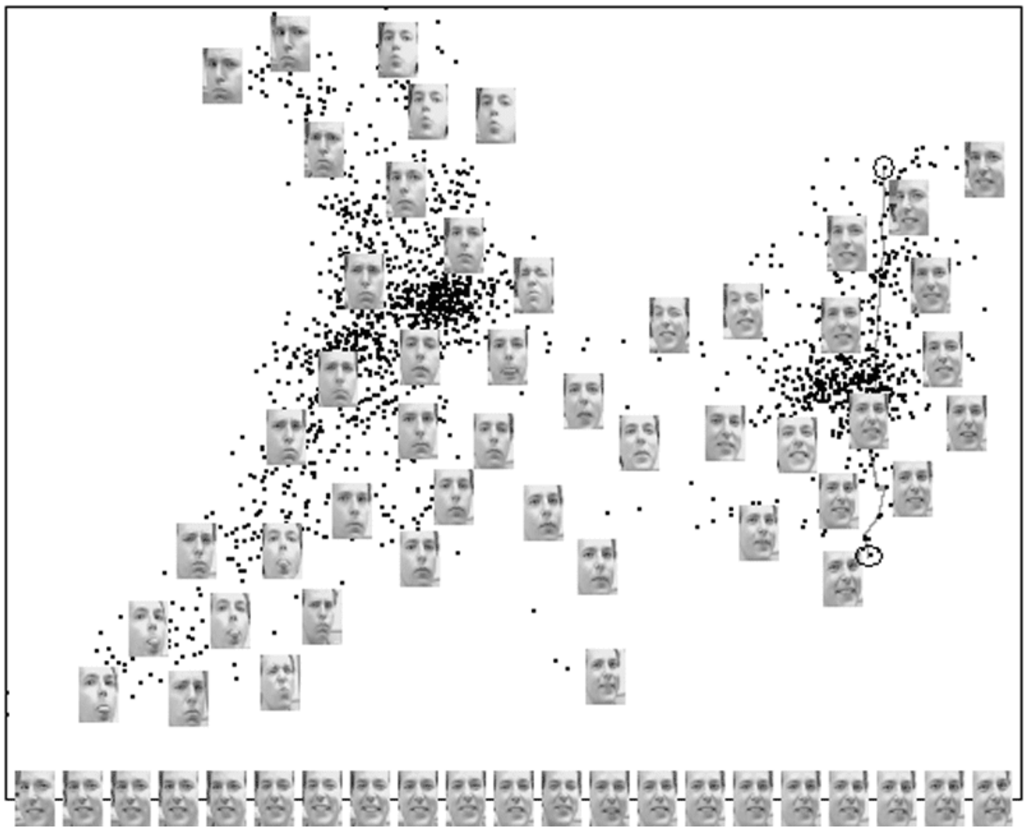


Figure II.10: Two-dimensional embedding of face images by a dimensionality reduction method. The face images are divided into two parts, the faces with open mouth and the faces with closed mouth. Moreover, the pose and expression of human faces change continuously and smoothly, from the top to the bottom, from the left to the right. The bottom images correspond to points along the right path (linked by solid line), illustrating one particular mode of variability in the pose. Source: [He *et al.* 2005].

In most cases, a manifold learning process has two stages: extracting properties of original data then computing a new representation - an embedding which preserves these properties into a low dimensional space. Given a data set $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{x}_i \in \mathbb{R}^n$ and its embeddings set $\mathcal{A} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}, \mathbf{a}_i \in \mathbb{R}^p$, where $\mathbf{a}_i$ is the embedded representation of $\mathbf{x}_i$, we note:

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c), \tag{II.32}$$

the embedding loss, where $\mathcal{A}_i^c$ is the complement of $\{\mathbf{a}_i\}$ in $\mathcal{A}$. The objective function of manifold learning problem is then defined as:

$$\min_{\mathcal{A}} \mathcal{L}_t = \min_{\mathcal{A}} \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) \tag{II.33}$$

We present some popular embedding losses hereafter:

- *Multi-Dimensional scaling* (MDS) [Kruskal & Wish 1978]:

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} (d_a(\mathbf{a}_i, \mathbf{a}_j) - d_x(\mathbf{x}_i, \mathbf{x}_j))^2, \tag{II.34}$$

where $d_a()$ and $d_x()$ are measures of dissimilarity. By default, they are both Euclidean distances. This method aims at preserving pairwise distances from the original representation in the embedding space.

- *Laplacian eigenmaps* (LE) [Belkin & Niyogi 2003]:

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} d_x(\mathbf{x}_i, \mathbf{x}_j) d_a(\mathbf{a}_i, \mathbf{a}_j), \tag{II.35}$$

where $d_x()$ is a measure of similarity measure (for instance $d_x(\mathbf{x}_i, \mathbf{x}_j) = \exp\big(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma_i^2}\big))$ and $d_a()$ is a measure of dissimilarity (for instance $d_a(\mathbf{a}_i, \mathbf{a}_j) = \|\mathbf{a}_i - \mathbf{a}_j\|_2^2)$. This method learns manifold structure by emphasizing the preservation of local distances. In order to further reduce effect of large distances, in some papers, $d_x(\mathbf{x}_i, \mathbf{x}_j)$ is set directly to zero if $\mathbf{x}_j$ is not in the k nearest neighbors of $\mathbf{x}_i$ or vice versa if $\mathbf{x}_i$ is not in the k nearest neighbors of $\mathbf{x}_j$. Alternatively, one can set $d_x(\mathbf{x}_i, \mathbf{x}_j) = 0$ if $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 > \kappa$. Let $\mathbf{W}$ be the matrix defined as $\mathbf{W}_{ij} = d_x(\mathbf{x}_i, \mathbf{x}_j)$ hence $\mathbf{W}$ is a symmetric matrix if $d_x()$ is symmetric. We can represent the objective function of Laplacian eigenmaps method using the matrices:

$$\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathbf{W}_{ij} \|\mathbf{a}_i - \mathbf{a}_j\|_2^2 = 2\,\mathrm{tr}(\mathbf{A}\mathbf{L}\mathbf{A}^\top), \tag{II.36}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{W}$, $\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{W}_{ij}$ ($\mathbf{D}$ being a diagonal matrix). $\mathbf{L}$ a graph Laplacian matrix because it is symmetric, the sum of each row equals to 0 and its elements are negatives except for the diagonal elements.

- *Locally Linear Embedding* (LLE) [Roweis & Saul 2000]:

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \left\| \mathbf{a}_i - \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} \lambda_{ij} \mathbf{a}_j \right\|_2^2, \tag{II.37}$$

where λ_{ij} are determined by solving the following problem:

$$\begin{aligned} &\min_{\lambda_{ij}} \left\| \mathbf{x}_i - \sum_j \lambda_{ij} \mathbf{x}_j \right\|_2^2, \\ &\text{subject to: } \begin{cases} \sum\limits_j \lambda_{ij} = 1, \text{ if } \mathbf{x}_j \in \text{knn}(\mathbf{x}_i), \\ \lambda_{ij} = 0 \text{ if not.} \end{cases} \end{aligned} \tag{II.38}$$

where $\text{knn}(\mathbf{x}_i)$ denote a set containing indices of the k nearest neighbors samples (in Euclidean distance) of the sample $\mathbf{x}_i$. Assuming that the observed data $\mathcal{X}$ is sampled from a smooth manifold and provided that the sampling is dense enough, one can assume that the sample lies locally on linear patches. Thus, LLE first computes the barycentric coordinate for each sample *w.r.t.* its nearest neighbors. These barycentric coordinates characterize the local geometry of the underlying manifold. Then, LLE computes a low dimensional representation (embedded) which is compatible with these local barycentric coordinates. Introducing $\mathbf{V} \in \mathbb{R}^{N \times N}$ a matrix representation form for λ as: $\mathbf{V}[j, i] = \lambda_{ij}$ then $\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c)$ can be rewritten as $\mathcal{L}_t = \|\mathbf{A} - \mathbf{AV}\|_F^2 = \text{tr}(\mathbf{ALA}^\top)$, where $\mathbf{L} = \mathbf{I}_N - \mathbf{V} - \mathbf{V}^\top + \mathbf{V}^\top\mathbf{V}$. Thus, the loss $\mathcal{L}_t$ can be interpreted as Laplacian eigenmaps loss, based on an implicit metric d_x for measuring distance between two samples.

- *Laplacian Learning* (LL) [Dong *et al.* 2016]:

$$\mathcal{L}_t = \text{tr}(\mathbf{ALA}^\top), \tag{II.39}$$

where $\mathbf{L}$ is learnt by solving the following problem:

$$\begin{aligned} \min_{\mathbf{L} \in \mathbb{R}^{N \times N}} \quad & \text{tr}\left(\mathbf{XLX}^\top\right) + \theta \|\mathbf{L}\|_F^2 \\ \text{subject to} \quad & \text{tr}(\mathbf{L}) = N, \\ & \mathbf{L}_{ij} = \mathbf{L}_{ji} < 0 (i \neq j), \\ & \sum_j \mathbf{L}_{ij} = 0. \end{aligned} \tag{II.40}$$

The constraint $\text{tr}(\mathbf{L}) = N$ is set in order to control the energy of $\mathbf{L}$. Instead of creating $\mathbf{L}$ using a predefined generic metric as in LE manifold learning method, in LL, the laplacian matrix is learnt directly from the data. To optimize problem II.40, we can use interior point method [Boyd & Vandenberghe 2004] or ADMM [Boyd *et al.* 2011]). We propose also a method to solve problem II.40 in appendix A.1.

- *Contrastive loss* :

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} \Big(d_x(\mathbf{x}_i, \mathbf{x}_j) d_a(\mathbf{a}_i, \mathbf{a}_j) + (1 - d_x(\mathbf{x}_i, \mathbf{x}_j)) \max(0, \tau - d_a(\mathbf{a}_i, \mathbf{a}_j)) \Big),$$

where $d_x(\mathbf{x}_i, \mathbf{x}_j)$ is a discrete similarity metric which is equal to 1 if $\mathbf{x}_j$ is in the neighborhood of $\mathbf{x}_j$ and 0 otherwise. $d_a()$ is a measure of dissimilarity.

- *Stochastic Neighbor Embedding* (SNE) [Hinton & Roweis 2003]:

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \sum_{\mathbf{a}_j \in \mathcal{A}_i^c} P_{ij} \log \frac{P_{ij}}{Q_{ij}},$$

where $P_{ij} = \frac{d_x(\mathbf{x}_i, \mathbf{x}_j)}{\sum_{k \neq i} d_x(\mathbf{x}_i, \mathbf{x}_k)}$ and $Q_{ij} = \frac{d_a(\mathbf{a}_i, \mathbf{a}_j)}{\sum_{k \neq i} d_a(\mathbf{a}_i, \mathbf{a}_k)}$, d_x and d_a are both similarity metric. The objective of this method is to preserve the similarity between two distributions of pairwise distances, one in original representation and the other in embedded representation, by Kullback–Leibler (KL) divergence.

Traditionally, manifold learning finds its applications in dimensionality reduction or data visualization, which refers to the techniques used to help the analyst see the underlying structure of data and explore it. For instance, [van der Maaten & Hinton 2008] proposes a variant of SNE, which has been used in a wide range of fields. Classical nonlinear manifold learning methods do not require a mapping model, which is a function $g()$ with trainable parameters that maps a sample $\mathbf{x}$ to its embedded representation $\mathbf{a}$ as $\mathbf{a} = g(\mathbf{x})$. $\mathbf{a}$ is directly used as the optimization variable.

Finally, it is worth noting that for several manifold learning methods such as LE and LLE, some supplement constraints are required to avoid a trivial solution (for instance with all embedded points are collapsed into only one point). Usually, mean and co-variance constrains are applied:

$$\begin{aligned} \mathrm{m}(\mathbf{A}) &= [\mu(\mathbf{A}[1,:]), .., \mu(\mathbf{A}[p,:])]^\top = \mathbb{0} \\ \mathrm{Cov}(\mathbf{A}, \mathbf{A}) &= \big(\mathbf{A} - \mathrm{M}(\mathbf{A})\big)\big(\mathbf{A} - \mathrm{M}(\mathbf{A})\big)^\top = \mathbf{I}_p \end{aligned} \tag{II.41}$$

Chapter III

Semi-supervised dictionary learning

Chapter abstract

In this chapter, we introduce Semi-Supervised Dictionary Learning (SSDL). Then, we present our proposed SSDL method. We show that our approach provides an improvement over state-of-the-art semi-supervised dictionary learning methods on some handwritten digit datasets and face datasets.

A Introduction

Although SDL has gained much interest and has shown significant performance improvements in classification, it requires a large number of labelled samples per class to be accurate, as for supervised learning methods in general. In order to deal with databases which have just a few labelled samples per class, semi-supervised learning, which also exploits unlabelled samples in training phase is used. Indeed, unlabelled samples can help to regularize the learning model, yielding an improvement of classification accuracy. In this section, we address semi-supervised dictionary learning (SSDL) by introducing some methods to convert SDL to SSDL and the related works. But before going further, let state three assumptions proposed by [Chapelle *et al.* 2006] on which most SSL methods rely.

- *The smoothness assumption.* If two points x_1, x_2 are close, then so should be the corresponding model outputs y_1, y_2.
- *The clustering assumption.* If points are in the same cluster, they are likely to be of the same class. The cluster assumption can be equivalently formulated as follows: the decision boundary should lie in a low-density region (low density separation).

- *The manifold assumption.* While being in a high dimensional ambient space, the data roughly lies on a low-dimensional manifold. As a consequence, some properties of the data organization can be preserved in a low-dimensional embedding space.

A.1 Generalities. We remind some notations used in this section:

- $\mathbf{a}_{i,j}$ is sparse code on the class-specific sub-dictionary $\mathbf{D}_j$ for sample $\mathbf{x}_i$, therefore $\mathbf{a}_i = [\mathbf{a}_{i,1}^\top, .., \mathbf{a}_{i,C}^\top]^\top$.
- $\mathbf{A}_{i,j}^l$ is sparse code corresponds to the sub-dictionary $\mathbf{D}_j$ for all samples of i^{th} class $\mathbf{X}_i^l$, hence $\mathbf{A}_i^l = [\mathbf{A}_{i,1}^{l\top}, ..., \mathbf{A}_{i,c}^{l\top}, ..., \mathbf{A}_{i,C}^{l\top}]^\top$ and $\mathbf{A}^l = [\mathbf{A}_1^l, ..., \mathbf{A}_i^l, ..., \mathbf{A}_C^l]$.
- $\mathbf{Q}$ is defined as in LCK-SVD method (problem II.19).
- $\mathbf{V}$ is learnt from data samples, in order to apply to sparse codes (which can be considered as a manifold learning method).
- $\mathrm{m}(\mathbf{Z})$ is the column mean for matrix $\mathbf{Z}$ and $\mathrm{M}(\mathbf{Z})$ is the matrix with the same size as $\mathbf{Z}$ by repeating column $\mathrm{m}(\mathbf{Z})$.
- l_q, l_{q_1} indicate norms with $0 < q, q_1 < 1$.
- $y_j^c(k) = 1$ if $k = c$, otherwise $y_j^c(k) = -1$.
- $\mathbf{L}$ denotes a Laplacian matrix, potentially used to regularize sparse code $(\mathbf{L}_A, \mathbf{L}^{LW}, \mathbf{L}^{LB})$ and eventually feature in dictionary $(\mathbf{L}_D)$.
- $r \geq 1$, activation hyper-parameter in $(\mathbf{P}_{ij})^r$, used for sharpening the estimated probabilities (see appendix A.8.2).

Besides, $\mathbf{X}^u$ denotes unlabelled data, which have N_u samples $\mathbf{x}_j^u$ $(j = 1, .., N_u)$, hence data is now composed of labelled data and unlabelled data: $\mathbf{X} = [\mathbf{X}^l, \mathbf{X}^u]$. In the same manner, $\mathbf{A}^u$ denotes sparse code of $\mathbf{X}^u$ that contains N_u sparse codes $\mathbf{a}_i^u$ $(j = 1, .., N_u)$ and $\mathbf{A} = [\mathbf{A}^l, \mathbf{A}^u]$.

We find three ways of extending a SDL method to the semi-supervised setting in the literature. The first one is used to convert a SDL into SSDL, *i.e.* to incorporate unlabelled samples in the learning, the second and third ones are used to reinforce semi-supervised learning. We present it hereafter from the objective function II.15 point of view.

(i) *The first straightforward step* consists in modifying $\mathcal{R}$ by incorporating the reconstruction error and the sparse codes penalty for the unlabelled data: $\|\mathbf{X}^u - \mathbf{D}\mathbf{A}^u\|_F^2 + \lambda \|\mathbf{A}^u\|_q$. It can be considered as a regularization for the dictionary $\mathbf{D}$ because we use more samples to train it. This setting is found in most SSDL approaches (see for example [Pham & Venkatesh 2008, Zhang *et al.* 2013, Wang *et al.* 2013]).

(*ii*) *The second one* goes further by adding a term $\mathcal{F}(\mathbf{A}^l, \mathbf{A}^u)$ into the objective function II.15, in order to enforce the preservation of the original data structure in the sparse code space [Zheng *et al.* 2011]. By considering that sparse coding amounts to embed data in the union of low dimensional subspaces, this constraint can be related to the manifold assumption. A similar constraint can be used to preserve features relationship within the learnt atoms [Yankelevsky & Elad 2017], then $\mathcal{F}$ can contain in addition $\mathbf{D}$ as variable such as $\mathcal{F}(\mathbf{A}^l, \mathbf{A}^u, \mathbf{D})$.

(*iii*) *A third one* would consist in modifying the functional $\mathcal{D}$ in II.15 to train the internal classifier in a semi-supervised way, therefore including unlabelled sparse codes in the learning procedure. To do so, several works introduce a new matrix variable $\mathbf{P} \in \mathbb{R}^{C \times N_u}$, whose entry $\mathbf{P}_{kj}$ is positive and indicates the estimated probability that an unlabelled sample j belongs to the class k (hence $\sum\limits_{k=1}^{C} \mathbf{P}_{kj} = 1$). Matrix $\mathbf{P}$ is learned during training phase as $\mathbf{D}$ and $\mathbf{A}$. It can be estimated by Label Propagation based methods (see appendix A.8.1), which rely on the smoothness assumption. In some methods, $\mathbf{P}$ is then sharpened (see appendix A.8.2), which is grounded on the clustering assumption.

Integrating the three aspects addressed above, we can formulate the following generic optimization problem for SSDL:

$$\begin{aligned}&\min_{\Theta} \left[\mathcal{R}(\mathbf{A}^l, \mathbf{A}^u, \mathbf{D}) + \mathcal{D}(\mathbf{W}, \mathbf{b}, \mathbf{A}^l, \mathbf{A}^u, \mathbf{D}, \mathbf{P}) + \mathcal{F}(\mathbf{A}^l, \mathbf{A}^u, \mathbf{D})\right], \\ &\text{where } \Theta = \{\mathbf{W}, \mathbf{b}, \mathbf{A}^l, \mathbf{A}^u, \mathbf{D} \in \mathcal{C}, \mathbf{P}\}.\end{aligned} \tag{III.1}$$

A.2 Related works. In table III.1, we list SSDL methods that follow the form III.1. We note that all methods use unlabelled data for dictionary fitting which makes sense, Dictionary Learning being by construction unsupervised. Thus JDL, OSSDL and LC-RLSDLA are SSDL extensions of DK-SVD (II.17) and LCK-SVD (II.19) for the last two.

Following (*ii*) that concerns $\mathcal{F}$, SSP-DL approach proposes to learn a matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$ as:

$$\min_{\mathbf{V}_{ii}=0} \|\mathbf{V}\|_{q,q}^q + \lambda \|\mathbf{X} - \mathbf{X}\mathbf{V}\|_F^2 \tag{III.2}$$

Thus, it is a sparse coding problem with $l_{q,q}$ norm ($0 < q < 1$) for sample $\mathbf{x}_i$, where the corresponding dictionary is composed of other samples. Then the matrix $\mathbf{V}$ captures linear relationship between the samples in the original space that are enforced in the sparse code space by minimizing $\|\mathbf{A} - \mathbf{A}\mathbf{V}\|_F^2$. SDGDL and PSSDL use Laplacian-based matrix $\mathbf{L}_A$, derived from LE and LLE

Method		$\mathcal{R}$	$\mathcal{D}$	$\mathcal{F}$
JDL	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}, \mathbf{W}$ s.t $\lVert \mathbf{a}_i \rVert_0 \leq \epsilon$	$\lVert \mathbf{X}^l - \mathbf{D}\mathbf{A}^l \rVert_F^2$ $+\rho \lVert \mathbf{X}^u - \mathbf{D}\mathbf{A}^u \rVert_F^2$	$\gamma \lVert \mathbf{Y} - \mathbf{W}\mathbf{A}^l \rVert_F^2$ $+\mu \lVert \mathbf{W} \rVert_F^2$	
OSSDL LC-RLSDLA	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}, \mathbf{W}, \mathbf{U}$ s.t $\lVert \mathbf{a}_i \rVert_0 \leq \epsilon$	$\lVert \mathbf{X}^l - \mathbf{D}\mathbf{A}^l \rVert_F^2$ $+\rho \lVert \mathbf{X}^u - \mathbf{D}\mathbf{A}^u \rVert_F^2$	$\gamma \lVert \mathbf{Y} - \mathbf{W}\mathbf{A}^l \rVert_F^2$ $+\psi \lVert \mathbf{Q} - \mathbf{U}\mathbf{A}^l \rVert_F^2$	
S2D2	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}$	$\lVert \mathbf{X} - \mathbf{D}\mathbf{A} \rVert_F^2$ $+\lambda \lVert \mathbf{A} \rVert_1$	$\sum_{i=1}^{N_l} \sum_{c=1}^{C} \lVert \mathbf{x}_i^l - \mathbf{D}_c \mathbf{a}_{i,c}^l \rVert_2^2$ $+ \sum_{j=1}^{N_u} \sum_{c=1}^{C} \Big(\lVert (\mathbf{x}_j^u - \mathbf{D}_c \mathbf{a}_{j,c}^u) \mathbf{P}_{cj} \rVert_2^2$ $+ \lVert \mathbf{D}_c \mathbf{a}_{j,c}^u (1 - \mathbf{P}_{cj}) \rVert_2^2 \Big)$ $+ \sum_{c=1}^{C} \Big(\lVert \mathbf{A}_c^l - \mathrm{M}[\mathbf{A}_c^l] \rVert_F^2$ $- \lVert \mathrm{m}[\mathbf{A}_c^l] - \mathrm{m}[\mathbf{A}^l] \rVert_2^2 \Big)$	
SSR-D *	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}$	$\lVert (\mathbf{X} - \mathbf{D}\mathbf{A})^\top \rVert_{2,q_1}^{q_1}$ $+\lambda \sum_{c=1}^{C} \lVert \mathbf{A}_c^l \rVert_{2,q}^q$ $+\lambda \lVert \mathbf{A}^u \rVert_{2,q}^q$		
SSP-DL *	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}$	$\lVert \mathbf{X} - \mathbf{D}\mathbf{A} \rVert_F^2$ $+\lambda_1 \sum_{c=1}^{C} \lVert \mathbf{A}_c^l \rVert_{2,q}^q$ $+\lambda_2 \lVert \mathbf{A}^u \rVert_{q,q}^q$		$\beta \lVert \mathbf{A} - \mathbf{A}\mathbf{V} \rVert_F^2$
SDGDL $^\diamond$	$\min$ $\lVert \mathbf{d}_i \rVert_2 = 1, \mathbf{A}^l, \mathbf{W}$ s.t $\lVert \mathbf{a}_i^l \rVert_0 \leq \epsilon$	$\lVert \mathbf{X}^l - \mathbf{D}\mathbf{A}^l \rVert_F^2$	$\gamma \lVert \mathbf{Y} - \mathbf{W}\mathbf{A}^l \rVert_F^2$	$\beta_1 \operatorname{tr}(\mathbf{A}^l \mathbf{L}_A \mathbf{A}^{l\top})$ $+\beta_2 \operatorname{tr}(\mathbf{D}^\top \mathbf{L}_D \mathbf{D})$
USSDL	$\min$ $\mathbf{D} \in \mathcal{C}, \mathbf{A}$ $\mathbf{W}, \mathbf{b}, \mathbf{P}$	$\lVert \mathbf{X} - \mathbf{D}\mathbf{A} \rVert_F^2$ $+\lambda \lVert \mathbf{A} \rVert_1$	$\gamma \big(\sum_{i=1}^{N_l} \sum_{c=1}^{C} \lVert \mathbf{w}_c^\top \mathbf{a}_i^l + b_c - y_i^c \rVert_2^2$ $+ \sum_{j=1}^{N_u} \sum_{k=1}^{C} (\mathbf{P}_{kj})^r \sum_{c=1}^{C} \lVert \mathbf{w}_c^\top \mathbf{a}_j^u + b_c - y_j^c(k) \rVert_2^2 \big)$ $+\mu \lVert \mathbf{W} \rVert_F^2$	
PSSDL	$\min$ $\mathbf{D}, \mathbf{A}, \mathbf{W}$ $\sigma_i, z, \sigma_d, \sigma_w$	$\sum_{i=1}^{N} \big(\frac{\lVert \mathbf{x}_i - \mathbf{D}\mathbf{a}_i \rVert_2^2}{2\sigma_i^2}$ $+ \log \sigma_i^{n+2} \big) + \frac{\lVert \mathbf{A} \rVert_1}{z}$ $+(N+1) \log z$ $+ \frac{\lVert \mathbf{D} \rVert_F^2}{2\sigma_d^2} + p \log \sigma_d^{n+2}$	$\sum_{i}^{N_l} \big(- \mathbf{w}_{c, y_i^c = 1}^\top \mathbf{a}_i^l$ $+ \log \big(\sum_{c=1}^{C} \exp(\mathbf{w}_c^\top \mathbf{a}_i^l) \big)$ $+ \frac{\lVert \mathbf{W} \rVert_F^2}{2\sigma_w^2} + C \log \sigma_w^{p+2}$ $+(1-\beta) \big(\operatorname{tr}(\mathbf{A}\mathbf{L}^{LW}\mathbf{A}^\top)$ $- \operatorname{tr}(\mathbf{A}\mathbf{L}^{LB}\mathbf{A}^\top) \big)$	$\beta \operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top)$

SSD-LP	$\min\limits_{\mathbf{D} \in \mathcal{C}, \mathbf{A}} \sum_{c=1}^{C} \left(\left\Vert \mathbf{X}_c^l - \mathbf{D}_c \mathbf{A}_{c,c}^l \right\Vert_F^2 + \lambda \left\Vert \mathbf{A}_{c,c}^l \right\Vert_1 \right) - \gamma \sum_{c=1}^{C} \left\Vert \mathbf{A}_{c,c}^l - \mathrm{M}[\mathbf{A}_{c,c}^l] \right\Vert_F^2 + \sum_{j}^{N_u} \sum_{c=1}^{C} \left(\mathbf{P}_{cj} \left\Vert \mathbf{x}_j^u - \mathbf{D}_c \mathbf{a}_{j,c}^u \right\Vert_2^2 + \lambda \left\Vert \mathbf{a}_{j,c}^u \right\Vert_1 \right)$

Table III.1: SSDL objective functions. OSSDL, LC-RLSDLA: online dictionary learning methods, they have the same objective function but using different methods for optimization. JDL: Joint Dictionary Learning [Pham & Venkatesh 2008], OSSDL: Online Semi-Supervised Dictionary Learning [Zhang *et al.* 2013], LC-RLSDLA: Label Consistent - Recursive Least Squares Dictionary Learning Algorithm [Matiz & Barner 2016], S2D2: Semi-Supervised Discriminative Dictionary [Shrivastava *et al.* 2012], SSR-D: Semi-Supervised Robust Dictionary [Wang *et al.* 2013], SSP-DL: Structural Sparse Preserving - Dictionary Learning [Wang *et al.* 2016], SDGDL: Supervised Dual Graph Dictionary Learning [Yankelevsky & Elad 2017], USSDL: Unified Semi-Supervised Dictionary Learning [Wang *et al.* 2015], PSSDL: Probabilistic Semi-Supervised Dictionary Learning [Babagholami-Mohamadabadi *et al.* 2013], SSD-LP: Semi-Supervised Dictionary - Label Propagation [Chen & Yang 2017].

methods respectively, to constraint sparse code $\mathbf{A}$. PSSDL uses also Local Fisher Discriminant Analysis [Sugiyama 2007], which is a semi-supervised manifold learning technique. Thus, it introduces the following penalty $(1-\beta)\big(\operatorname{tr}(\mathbf{A}\mathbf{L}^{LW}\mathbf{A}^\top) - \operatorname{tr}(\mathbf{A}\mathbf{L}^{LB}\mathbf{A}^\top)\big)$. We refer readers to the original paper for the explicit form of $\mathbf{L}^{LW}$ and $\mathbf{L}^{LB}$.

Finally, let turn to some SSDL methods that learn a semi-supervised internal classifier ((iii)). S2D2 uses pseudo-label $\mathbf{P}_{cj}$ to weight the reconstruction loss between class-specific sub-dictionaries $\mathbf{D}_c$: $\left\Vert (\mathbf{x}_j^u - \mathbf{D}_c \mathbf{a}_{j,c}^u)\mathbf{P}_{cj} \right\Vert_2^2$, and uses $(1-\mathbf{P}_{cj})$ to force unrelated class components of sparse codes to zero: $\left\Vert \mathbf{D}_c \mathbf{a}_{j,c}^u (1-\mathbf{P}_{cj}) \right\Vert_2^2$. The Probability matrix $\mathbf{P}$ is estimated based on reconstruction error given by the updated sparse codes dictionary. In SSD-LP, the probability matrix $\mathbf{P}$ is estimated using a rule inspired from the Label Propagation ([Zhu & Ghahramani 2002, Zhou *et al.* 2004] in appendix A.8.1). Finally, USSDL uses $\mathbf{P}$ to weight the classification loss between sub-classifiers $(\mathbf{w}_c, b_c)$: $(\mathbf{P}_{kj})^r \sum_{c=1}^{C} \left\Vert \mathbf{w}_c^\top \mathbf{a}_j^u + b_c - y_j^c(k) \right\Vert_2^2$, where $r > 1$ is an activation hyper-parameter used for sharpening the estimated probabilities (see appendix A.8.2). Finally, in USSDL, $\mathbf{P}$ is estimated based on classification error given by updated sparse code and updated classifier.

B Proposed method

In our work, we decided to design the objective function based on the form II.16, assuming that samples in the different classes have similar textures and need to be reconstructed from the common dictionary. In addition, we want to train simultaneously the internal classifier and the dictionary to make sparse code more discriminative. In the following, we refer to our method as

Semi-Supervised Dictionary Learning with Graph regularization and Active points method (SSDL-GA for short). It can be viewed is an extended version of USSDL by combining with manifold preservation for both sparse code and dictionary. Our proposed method also takes into account the manifold preservation for sparse coding out-of-sample data points. In this section, we construct the objective function of SSDL-GA step by step, then we present the optimization process and we conclude with some numerical experiments.

B.1 Construction of objective function. In our method, we use the reconstruction error $\mathcal{R}$ as:

$$\mathcal{R}(\mathbf{A},\mathbf{D}) = \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 + \lambda\|\mathbf{A}\|_1 \tag{III.3}$$

It uses both labelled and unlabelled samples. The manifold learning for preservation $\mathcal{F}$ is applied for both sparse code and dictionary:

$$\mathcal{F}(\mathbf{A}^l,\mathbf{A}^u,\mathbf{D}) = \beta\,\mathrm{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) + \varphi\,\mathrm{tr}(\mathbf{D}^\top\mathbf{L}_D\mathbf{D}), \tag{III.4}$$

where $\mathbf{L}_A, \mathbf{L}_D$ are both Laplacian-based matrix. These matrices can be considered as hyper-parameters, which can be set using LE, LLE or LL manifold learning method. Laplacian-based matrices are used for manifold learning in DiL thanks to their simplicity (quadratic form) and in addition, they guarantee the convexity for sparse coding. For the functional $\mathcal{D}$, we construct this term in the same way as in USSDL. Indeed, USSDL uses an internal semi-supervised classifier, which is developed from the Adaptive Semi-Supervised Learning [Wang *et al.* 2014] and then combined with the Active points method. Following Adaptive Semi-Supervised Learning approach, $\mathcal{D}$ is split into two functionals $\mathcal{D}^l$ and $\mathcal{D}^u$ defined as:

$$\mathcal{D}^l(\mathbf{W},\mathbf{b},\mathbf{A}^l) = \gamma\sum_{i=1}^{N_l}\sum_{c=1}^{C}\left\|\mathbf{w}_c^\top\mathbf{a}_i^l + b_c - y_i^c\right\|_2^2 = \gamma\left\|\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y}\right\|_F^2, \tag{III.5}$$

and

$$\begin{aligned}\mathcal{D}^u(\mathbf{W},\mathbf{b},\mathbf{A}^u,\mathbf{P}) &= \gamma\sum_{j=1}^{N_u}\sum_{k=1}^{C}(\mathbf{P}_{kj})^r\sum_{c=1}^{C}\left\|\mathbf{w}_c^\top\mathbf{a}_j^u + b_c - y_j^c(k)\right\|_2^2\\ &= \gamma\sum_{k=1}^{C}\left\|(\mathbf{1}_{N_u}^\top\mathbf{P}[k,:])^{r/2}\circ(\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\right\|_F^2,\end{aligned} \tag{III.6}$$

where:

- the matrices $\mathbf{1}_{N_l}$ and $\mathbf{1}_{N_u}$ are matrix One of size $1\times N_l$ and $1\times N_u$, respectively.

- ($\circ$) is the Hadamard product.

- $y_j^c(k) = 1$ if $k = c$, otherwise $y_j^c(k) = -1$. $\mathbf{y}_j(k) = [y_j^1(k), y_j^2(k), ..., y_j^C(k)]^\top$ and $\mathbf{Y}_k = [\mathbf{y}_1(k), \mathbf{y}_2(k), ..., \mathbf{y}_{N_u}(k)]$.

- $r \geq 1$ is an activation hyper-parameter in function x^r, with $0 \leq x \leq 1$.

As a reminder, $\mathbf{P}$ is a $C \times N_u$ matrix, whose entry $\mathbf{P}_{kj}$ is positive and indicates the estimated probability that an unlabelled sample j belongs to class k. In $\mathcal{D}^u$, $\mathbf{P}$'s entries are used as weight parameters associated with classification error for unlabelled samples. The decision boundary is the sparse code space hyperplan defined by the equation $\mathbf{w}_c^\top \mathbf{a} + b_c = 0$. Similarly to SVM, the decision boundary is ajusted only based on nearby samples, hence the concept of *active points*. Active points are the sparse codes that verify the following constraint:

$$\begin{cases} \mathbf{w}_c^\top \mathbf{a}_i + b_c < 1, \text{ if } \mathbf{a}_i \text{ belongs to this class } c \\ \mathbf{w}_c^\top \mathbf{a}_i + b_c > -1, \text{if not.} \end{cases} \tag{III.7}$$

Then, for labelled samples, matrix $\mathbf{Q}^l \in \mathbb{R}^{C \times N_l}$ indicates active points for a given class as follows: $\begin{cases} \mathbf{Q}^l[c,i] = 1, \text{ if } y_i^c(\mathbf{w}_c^\top \mathbf{a}_i^l + b_c) < 1 \\ \mathbf{Q}^l[c,i] = 0, \text{ otherwise} \end{cases}$. In a similar fashion, we define the matrix $\mathbf{Q}_k^u \in \mathbb{R}^{C \times N_u}$ for unlabeled samples: $\begin{cases} \mathbf{Q}_k^u[c,j] = 1 \text{ if } y_j^c(k)(\mathbf{w}_c^\top \mathbf{a}_j^u + b_c) < 1 \\ \mathbf{Q}_k^u[c,j] = 0, \text{ otherwise} \end{cases}$, $\forall k \in [1,..,C]$. Finally, $\mathcal{D}$ can be rewritten as follows, with l_2 regularization on classifier:

$$\begin{aligned} \mathcal{D}(\mathbf{W}, \mathbf{b}, \mathbf{A}^l, \mathbf{A}^u, \mathbf{P}) = \gamma\Big(& \left\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\right\|_F^2 \\ & + \sum_{k=1}^{C} \left\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\right\|_F^2 \Big) + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2), \end{aligned}$$

Aggregating the reconstruction $\mathcal{R}$, the manifold learning for structure preservation $\mathcal{F}$ and the classification $\mathcal{D}$, we end up with the following objective function:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{b},\mathbf{A},\mathbf{P},\mathbf{D}\in\mathcal{C}} & \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \beta \operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) + \varphi \operatorname{tr}(\mathbf{D}^\top \mathbf{L}_D \mathbf{D}) + \gamma\Big(\left\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\right\|_F^2 \\ & + \sum_{k=1}^{C} \left\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\right\|_F^2 \Big) + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2) + \lambda \|\mathbf{A}\|_1, \end{aligned} \tag{III.8}$$

where:

- $\mathbf{Q}^l = (\mathbf{Y} \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l}) < \mathbb{1})$ and $\mathbf{Q}^u_k = (\mathbf{Y}_k \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u}) < \mathbb{1})$, $\forall k \in [1, .., C]$
- $\mathbf{P} \in \mathbb{R}^{C\times N_u}, \mathbf{P}_{kj} \in [0,1]$ and $\sum\limits_{k=1}^{C} \mathbf{P}_{kj} = 1, \forall j$

We propose a minimization scheme in the following subsection.

B.2 Optimization.

B.2.1 Alternate update

In the optimization process, the five following steps are repeated until convergence is reached:

- *Active elements update*:

$$\begin{aligned} \mathbf{Q}^l &= (\mathbf{Y} \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l}) < \mathbb{1}) \\ \mathbf{Q}^u_k &= (\mathbf{Y}_k \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u}) < \mathbb{1}), \forall k \in [1, .., C] \end{aligned} \tag{III.9}$$

The complexity for this step is $\mathcal{O}(pCN_l + pC^2N_u)$.

- *Probability update*:

$$\begin{aligned} &\min_{\mathbf{P}\geq \mathbb{0}} \sum_{j=1}^{N_u}\sum_{k=1}^{C}(\mathbf{P}_{kj})^r \sum_{c=1}^{C} \mathbf{Q}^u_k[c,j] \left\| y^c_j(k)(\mathbf{w}_c^\top \mathbf{a}^u_j + b_c) - 1 \right\|_2^2, \\ &\text{subject to } \sum_{k=1}^{C} \mathbf{P}_{kj} = 1, \forall j. \end{aligned} \tag{III.10}$$

This problem can be solved efficiently using Lagrange multiplier as in appendix A.2. The complexity for this step is $\mathcal{O}(pC^2N_u)$.

- *Sparse coding*:

$$\begin{aligned} &\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \lambda \|\mathbf{A}\|_1 + \beta \operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) + \varphi \operatorname{tr}(\mathbf{D}^\top \mathbf{L}_D \mathbf{D}) \\ &+\gamma\Big(\left\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\right\|_F^2 + \sum_{k=1}^{C} \left\| \mathbf{Q}^u_k \circ (\mathbf{1}^\top_{N_u}\mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\right\|_F^2 \Big) \end{aligned} \tag{III.11}$$

The problem can be solved efficiently using FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) with backtracking [Beck & Teboulle 2009]. We give more details in appendix A.3. The complexity for this step is $\mathcal{O}((p^2N + pN^2 + npN + pN_lC + pC^2N_u)s_s)$, where s_s is number of iterations.

- *Dictionary update* :

$$\min_{\mathbf{D}\in\mathcal{C}} \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 + \varphi\, \mathrm{tr}(\mathbf{D}^\top \mathbf{L}_D \mathbf{D}) \tag{III.12}$$

As in Sparse Coding, we use FISTA with backtracking to solve this problem. We give more details in appendix A.4. The complexity for this step is $\mathcal{O}(p^2N + (p^2n + pNn)s_d)$, where s_p is the number of iterations.

- *Classifier update*:

$$\min_{\mathbf{W},\mathbf{b}} \gamma\Big(\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\|_F^2 + \sum_{k=1}^{C} \Big\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\Big\|_F^2 \Big) + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2) \tag{III.13}$$

We provide a solution for this problem in appendix A.5. The complexity of this step is $\mathcal{O}(p^2N_uC^2 + p^2N_lC)$.

Algorithm 1 SSDL-GA.

```
Require: $\mathbf{X}, \mathbf{Y}, \beta, \varphi, \gamma, \lambda, \mu, p, r$.
  Initialize: $\mathbf{L}_A, \mathbf{L}_D$, $\mathbf{D}, \mathbf{A}, \mathbf{W}, \mathbf{b}, \mathbf{Y}_k$
  Normalization: $\omega_A = \frac{N}{\mathrm{tr}\,(\mathbf{L}_A)}$, $\omega_D = \frac{n}{\mathrm{tr}\,(\mathbf{L}_D)}\omega_D$, $\mathbf{L}_A \leftarrow \omega_A \mathbf{L}_A, \mathbf{L}_D \leftarrow \omega_D \mathbf{L}_D$
  while not converged do
    Update $\mathbf{Q}^l, \mathbf{Q}_k^u$.
    Update the probability matrix $\mathbf{P}$
    Update sparse code $\mathbf{A}$ (Sparse coding)
    Update dictionary $\mathbf{D}$
    Update classifier $\mathbf{W}, \mathbf{b}$
  end while
  Output: $\mathbf{D}, \mathbf{A}, \mathbf{W}, \mathbf{b}, \mathbf{P}$
```

The global optimization is summarized in algorithm 1. We normalize energy of Laplacian matrix ($\mathrm{tr}(\mathbf{L}_A) = N$ and $\mathrm{tr}(\mathbf{L}_D) = n$) for avoiding scalar transfer between β and $\mathrm{tr}(\mathbf{L}_A)$, also between ϕ and $\mathrm{tr}(\mathbf{L}_D)$ in tuning hyper-parameter stage for β, ϕ. In addition, since $\mathbf{L}_A, \mathbf{L}_D$ can be constructed or learnt by different manifold learning Laplacian-based methods, this normalization guarantees the equity in a comparison of effect between these methods.

In general, $p \sim \mathcal{O}(n)$, the complexity for the global algorithm is $\mathcal{O}\big((n^2N + nN^2 + nC^2N)s_ss_t + (n^3 + n^2N)s_ds_t + n^2NC^2s_t\big)$, where s_t is the number of iteration for global algorithm. The memory complexity is proportional to N^2 because of the manifold structure preservation, which can be prohibitive for large data sets. Thus, we developed a sparse coding with epoch and batch strategy which is explained in appendix A.6. Once we have the optimal internal classifier through the matrix $\mathbf{W}$ and the bias $\mathbf{b}$, the unlabelled sample $\mathbf{a}_i^u$ is classified into the class $\hat{j}$ according to the following equation:

$$\hat{j} = \underset{j}{\operatorname{argmax}}\, \mathbf{w}_j^\top \mathbf{a}_i^u + b_j, \quad \text{where } \mathbf{w}_j^\top \text{ is } j^{th} \text{ row of } \mathbf{W} \tag{III.14}$$

B.2.2 Initialization

The dictionary $\mathbf{D}$ is initialized as follows: if there are more atoms than labelled samples ($p > N_l$), all the labelled samples are used as initial atoms and the remaining initial atoms are selected randomly from the unlabelled samples. Otherwise, we select randomly a labelled sample for each class until we obtain p samples. Then, each atom $\mathbf{d}_i$ is projected on the l_2 sphere of radius α ($\mathbf{D} \in \mathcal{C}$). The sparse codes $\mathbf{A}$ are initialized by solving the following LASSO problem with the initial dictionary $\mathbf{D}$:

$$\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{DA}\|_F^2 + \lambda \|\mathbf{A}\|_1 \tag{III.15}$$

Finally, the linear classifier $\mathbf{W}$ and $\mathbf{b}$ is initialized using only the labelled sparse codes by solving the following problem:

$$\begin{aligned} &\min_{\mathbf{W},\mathbf{b}} \gamma \left\|\mathbf{Y} - \mathbf{WA}^l - \mathbf{b1}_{N_l}\right\|_F^2 + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2) \\ =&\min_{\mathbf{W}'} \gamma \left\|\mathbf{Y} - \mathbf{W}'\mathbf{A}^{l*}\right\|_F^2 + \mu \|\mathbf{W}'\|_F^2, \end{aligned} \tag{III.16}$$

where $\mathbf{W}' = [\mathbf{W}, \mathbf{b}] \in \mathbb{R}^{C\times(p+1)}$ (we add $\mathbf{b}$ as a column after the last one of $\mathbf{W}$) and $\mathbf{A}^{l*} = \begin{bmatrix} \mathbf{A}^l \\ \mathbf{1}_{N_l} \end{bmatrix}$. The solution $\hat{\mathbf{W}}'$ is given in closed-form by:

$$\hat{\mathbf{W}}' = \mathbf{Y}(\mathbf{A}^{l*})^\top \left(\mathbf{A}^{l*}(\mathbf{A}^{l*})^\top + \frac{\mu}{\gamma} I\right)^{-1} \tag{III.17}$$

We set $\frac{\mu}{\gamma} = 2$ in the initialization and in the optimization process.

B.2.3 Out-of-sample data points

We suppose that the dictionary $\mathbf{D}$ and the sparse codes $\mathbf{A} \in \mathbb{R}^{p\times N}$ have been calculated for N training samples. If we have q new unlabelled data points $\mathbf{X}^{new} = [\mathbf{x}_{N+1}, \mathbf{x}_{N+2}, ..., \mathbf{x}_{N+q}]$, we perform a simple sparse coding step for each new unlabelled data point $\mathbf{x}_{N+i}$, by taking into account manifold structure preservation as follows, depending on manifold learning method:

- *Laplacian eigenmaps*:

$$\min_{\mathbf{a}_{N+i}} \|\mathbf{x}_{N+i} - \mathbf{D}\mathbf{a}_{N+i}\|_2^2 + \lambda \|\mathbf{a}_{N+i}\|_1 + \beta\omega_A \sum_{j=1}^{N} \frac{1}{2} d_x(\mathbf{x}_{N+i}, \mathbf{x}_j) \|\mathbf{a}_{N+i} - \mathbf{a}_j\|_2^2 , \tag{III.18}$$

where d_x is the metric used for calculating pairwise distances among N training samples. We multiply new pairwise distances by ω_A (in algorithm 1) for the equity with existing pairwise distances.

- *Locally Linear Embedding*:

$$\min_{\mathbf{a}_{N+i}} \|\mathbf{x}_{N+i} - \mathbf{D}\mathbf{a}_{N+i}\|_2^2 + \lambda \|\mathbf{a}_{N+i}\|_1 + \beta\omega_A \left\| \mathbf{a}_{N+i} - \sum_{j \in \mathrm{knn}'(N+i)} \hat{\lambda}_{ij} V \mathbf{a}_j \right\|_2^2 \tag{III.19}$$

The set $\mathrm{knn}'(N+i)$ contains the indices of the k nearest samples among the N training samples for $\mathbf{x}_{N+i}$. We get the coefficients λ_{ij} by solving a problem of the form (II.38), as previously mentioned.

- *Laplacian Learning*:

$$\begin{aligned} \mathbf{L}'_A = & \operatorname*{argmin}_{\mathbf{L} \in \mathbb{R}^{(N+1)\times(N+1)}} && \mathrm{tr}\left([\mathbf{X}, \mathbf{x}_{N+i}]\mathbf{L}[\mathbf{X}, \mathbf{x}_{N+i}]^\top\right) + \theta \|\mathbf{L}\|_F^2 \\ & \text{subject to} && \mathrm{tr}(\mathbf{L}) = N+1, \\ &&& \mathbf{L}_{ij} = \mathbf{L}_{ji} < 0 (i \neq j), \\ &&& \sum_j \mathbf{L}_{ij} = 0. \end{aligned} \tag{III.20}$$

$$\begin{aligned} \min_{\mathbf{a}_{N+i}} \|\mathbf{x}_{N+i} - \mathbf{D}\mathbf{a}_{N+i}\|_2^2 + \lambda \|\mathbf{a}_{N+i}\|_1 + \beta\Big(& 2\,\mathrm{tr}\left(\mathbf{a}_{N+i}\mathbf{L}'_A[N+1, 1:N]\mathbf{A}^\top\right) \\ & + \mathrm{tr}\left(\mathbf{a}_{N+i}\mathbf{L}'_A[N+1, N+1]\mathbf{a}_{N+i}^\top\right)\Big) \end{aligned} \tag{III.21}$$

We first recompute $\mathbf{L}'_A$ to take into account a new sample.

B.3 Numerical experiments. We organize this subsection as follows: firstly, we show the advantage of the manifold structure preservation constraint on the USPS database (United States Postal Service). Then we assess the impact of the number of unlabelled samples involved in the training on both USPS and MNIST datasets [LeCun *et al.* 2010]. Using the same datasets, we compare the performance of our approach with other SSDL methods, as well as some popular classifiers. Finally, we evaluate our approach in the setting where very few labels are available using the two faces databases, Extended YaleB [Georghiades *et al.* 2001] and AR [Martinez & Benavente 1998]. Note that, the pre-processing of data is very important and will be detailed in each experiment.

B.3.1 Sparse code regularization

We evaluate the regularization effect of different Laplacian-based matrices $\mathbf{L}_A$ for the accuracy of classification and their robustness to noise, on the USPS handwritten digits dataset. This dataset consists of 9298 images, where 7291 images are used for training and 2007 images are used for testing. Each image has size 16×16, which is represented by 256-dimensional vectors. In this experiment, the training set contains N_l labelled samples which are extracted from 7291 images and the testing set contains all 2007 images. In order to assess manifold structure preservation, we use the following objective function to obtain the dictionary and labelled samples sparse code:

$$\min_{\mathbf{A}^l,\mathbf{D}\in\mathcal{C}} \left\|\mathbf{X}^l - \mathbf{D}\mathbf{A}^l\right\|_F^2 + \beta\,\mathrm{tr}\,(\mathbf{A}^l\mathbf{L}_A\mathbf{A}^{l\top}) + \lambda\left\|\mathbf{A}^l\right\|_1 \tag{III.22}$$

Method and hyper-parameters	Construction for Laplacian matrix $\mathbf{L}_A$
LE-knn k, σ	$\mathbf{W}_{ij} = d_x(\mathbf{x}_i^l, \mathbf{x}_j^l) = \begin{cases} \exp\left(\frac{-\left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|_2^2}{2\sigma^2}\right), \text{ if } \mathbf{x}_j^l \in \mathrm{knn}(\mathbf{x}_i^l) \text{ or } \mathbf{x}_i^l \in \mathrm{knn}(\mathbf{x}_j^l) \\ 0, \text{ otherwise} \end{cases}$ $\mathbf{L}_A = \mathbf{D} - \mathbf{W}, \text{ where } \mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij} \text{ and } \mathbf{D}_{ij} = 0 (i \neq j)$
LE-Threshold κ, σ	$\mathbf{W}_{ij} = d_x(\mathbf{x}_i^l, \mathbf{x}_j^l) = \begin{cases} \exp\left(\frac{-\left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|_2^2}{2\sigma^2}\right), \text{ if } \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|_2 < \kappa \\ 0, \text{ otherwise} \end{cases}$ $\mathbf{L}_A = \mathbf{D} - \mathbf{W}, \text{ where } \mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij} \text{ and } \mathbf{D}_{ij} = 0 (i \neq j)$
LL θ	$\mathbf{L}_A = \min_{\mathbf{L}} \mathrm{tr}\,(\mathbf{X}^l\mathbf{L}\mathbf{X}^{l\top}) + \theta\left\|\mathbf{L}\right\|^2$ $\text{s.t tr}(\mathbf{L}) = N_l, \mathbf{L}_{ij} = \mathbf{L}_{ji} < 0 (i \neq j), \Sigma_j \mathbf{L}_{ij} = 0$
LLE k	$\min_{\lambda_{ij}} \|\mathbf{x}_i^l - \sum_j \lambda_{ij}\mathbf{x}_j^l\|_2^2 \text{ subject to: } \begin{cases} \sum_j \lambda_{ij} = 1, \text{ if } \mathbf{x}_j^l \in \mathrm{knn}(\mathbf{x}_i^l), \\ \lambda_{ij} = 0 \text{ if not.} \end{cases}$ $\mathbf{L}_A = \mathbf{I}_N - \mathbf{V} - \mathbf{V}^\top + \mathbf{V}^\top\mathbf{V}, \text{ where } \mathbf{V}_{ji} = \lambda_{ij}$

Table III.2: Laplacian matrix $\mathbf{L}_A$ constructed by different methods and their corresponding hyper-parameters.

We use four configurations of $\mathbf{L}_A$: LE-knn, LE-Threshold, LL, LLE as described in table III.2.

The metric used to find k nearest neighbor of a sample in LE-knn and LLE is Euclidean metric. Since no internal classifier is learnt while optimizing the objective function eq. (III.22), the computed sparse codes are used to train an external linear SVM classifier. This classifier is tuned with different box constraint values $\{0.1, 1, 10\}$ (a hyper-parameter of SVM). We used a "one against all" strategy and a five-fold cross validation. Then each testing sample is sparse coded with the manifold learning regularization as described in section III.B.2.3. Finally, the trained SVM predicts labels for testing sparse code.

The experiment is performed as follows. Firstly, we set $\beta = 0$ to find the pair (λ, p), with $\lambda \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1\}$ and $p \in \{32, 64, 128, 256\}$, that gives the best accuracy rate. Secondly, we fix the best pair $(\lambda, p) = (0.5, 128)$ to tune the remaining hyper-parameters (which depends on the method): $\beta, \theta, \sigma, \kappa, k$ (table III.3). For tuning κ, we introduce its replacement ζ. We sort all pairwise distances $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|_2$ then take ζ percentage of them (from the smallest value) as active pairwise distances. σ is chosen in a range based upon the mean distance $\|\mathbf{x}_i^l - \mathbf{x}_j^l\|_2$ in the dataset, which approximately equal to 10. In all settings, the experiment is repeated three times with random initializations of the dictionary and the best score is retained.

Hyper-parameter	Values	Method
β	$\{0.01, 0.1, 1, 10, 100\}$	All
k	$\{2, 3, 4, 5, 6, 7, 8, 9\}$	LE-knn, LLE
σ	$\{0.1, 1, 10, 15, 30, 1000\}$	LE-knn, LE-Threshold
ζ (for κ)	$\{0.03, 0.05, 0.1, 0.15, 0.3, 0.5, 0.7\}$	LE-Threshold
θ	$\{10^{-3}, 5 \times 10^{-3}, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 100\}$	LL

Table III.3: Hyper-parameter values for sparse code regularization tuning.

Table III.4 shows the best error rates (with the best hyper-parameters) for each Laplacian-based matrix. Firstly, we see that using sparse code regularization ($\beta \neq 0$) systematically decrease the error rate. Secondly, the matrix $\mathbf{L}_A$ constructed using the LLE approach gives the best error rate. Besides, computing $\mathbf{L}_A$ using LLE or LL requires only one hyper-parameter, k or θ respectively. Therefore the tuning is less time consuming. Nevertheless, interpreting geometrically the matrix $\mathbf{L}_A$ derived from the LL approach is not straightforward. Hence, a new matrix manifold constraint $\mathbf{L}'_A$ is learnt when sparse coding testing samples, which increases the computational cost when the LL approach is used.

Finally, we evaluate the robustness of the Laplacian matrix constructed by different methods to noise in samples. We fix the number of labelled training samples at 500 and add different noise amplitudes into both training and testing samples. The hyper-parameters are tuned as in previous experiment to get the best error rate for each method. Table III.5 shows these best error rates for

Methods / N_l	100	500	1000	2000
SC ($\beta = 0$)	19.6	10.9	8.0	7.5
SC LE-knn	16.6	9.1	6.8	6.0
SC LE-Threshold	18.1	9.4	7.9	7.1
SC LL	17.33	9.4		
SC LLE	16.6	8.3	6.4	5.8

Table III.4: The error rate of classification on different types of Laplacian-based matrix used for sparse code regularization, with different number of labelled samples in training (same number of samples per class). SC means sparse coding.

each type of Laplacian matrix used. Globally, we observe that SC LLE gives the best error rate among compared methods.

Methods / σ	0	0.2	0.4	0.6	0.8	1
SC ($\beta = 0$)	10.9	10.9	11.8	12.7	15.7	17.6
SC LE-knn	9.1	9.3	9.9	10.7	12.3	13.9
SC LE-Threshold	9.4	9.5	10.5	11.9	13.7	15.2
SC LL	9.4	9.5	9.8	10.5	12.0	14.6
SC LLE	8.3	9.0	9.8	10.9	12.0	13.9

Table III.5: The error rate of classification on different types of Laplacian matrix used for sparse codes regularization, with different additive noise levels. Here we fix the number of labelled samples in training $N_l = 500$ and use Gaussian noise following the distribution $\mathcal{N}(0, \sigma_N)$ where $\sigma_N = \sigma\mu(\mathbf{X}^2)$.

Given the previous results, we retain the LLE approach for computing the matrix $\mathbf{L}_A$ henceforward. Regarding to $\mathbf{L}_D$ used for features structural dependencies preservation in the dictionary atoms, we use LL instead, as proposed in [Yankelevsky & Elad 2017]. The number of atoms being fixed and we do not need to relearn new matrix as for the test samples sparse coding.

B.3.2 Low number of labelled samples

In this part, we evaluate our approach SSDL-GA on the USPS and MNIST datasets. MNIST contains images (28×28) of 10 handwritten digits, 60000 images for the training set and 10000 images for the testing set.

In the first test, for each dataset, we select for each class: 20 images as labelled samples, 100 images as testing samples and an increasing number of images 20, 40, 60, 80, 100, 150 as unlabelled samples. Each image (under form of vector) is normalized to have unit l_2 norm. Since we constrain $\|\mathbf{d}_i\|_2 \leq \alpha$ and we want to tune α for several values, an other way to do this is to fix $\alpha = 1$ and multiply normalized images by a scalar, *e.g.* two following problems are equivalent:

$$\min_{\mathbf{D},\mathbf{A}} \|5 \times \mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \lambda \|\mathbf{A}\|_1 , \text{ s.t } \|\mathbf{d}_i\|_2 \leq 1 \tag{III.23}$$

$$\min_{\mathbf{D},\mathbf{A}} ||\mathbf{X} - \mathbf{DA}||_F^2 + \frac{\lambda}{25} \|\mathbf{A}\|_1 \,, \text{ s.t } \|\mathbf{d}_i\|_2 \le \frac{1}{5} \tag{III.24}$$

Hence, we multiply normalized images by 5 which is equivalent to constrain $\|\mathbf{d}_i\|_2 \le 0.2$ and scale down other hyper-parameters by 25.

Five random samplings were conducted and the average scores for this test are shown in figure III.1. To tune hyper-parameters, we perform a grid search. We take $\mu = 2\gamma$ since μ is less sensitive compared to other hyper-parameters. As in the previous subsection, we first optimize the pair (λ, p) while fixing $(\gamma = 0, \beta = 0, \varphi = 0)$. Then we fix the best pair (λ, p) found and tune the remaining hyper-parameters. For the USPS dataset, we used the hyper parameters: $p = 200, \lambda = 0.3, \alpha = 1, \beta = 0.5, \varphi = 0, \gamma = 0.5, \mu = 1, k = 8, r = 1.7$ and for the MNIST datasets, we used $p = 200, \lambda = 0.5, \alpha = 1, \beta = 1, \varphi = 0, \gamma = 1, \mu = 2, k = 8, r = 2$. From this test, we can make two observations. Firstly, the accuracy rate can be significantly improved by increasing the number of unlabelled samples in training and it converges to a stable value. This suggests that, for a given dictionary size, there is a threshold number of unlabelled additional samples above which the sparse codes discriminative power stops improving significantly. Secondly, with a sufficient number of unlabelled samples in training, the accuracy rates for these unlabelled samples and for testing samples are similar. Hence, integrating unlabelled samples in the training is beneficial for generalization.

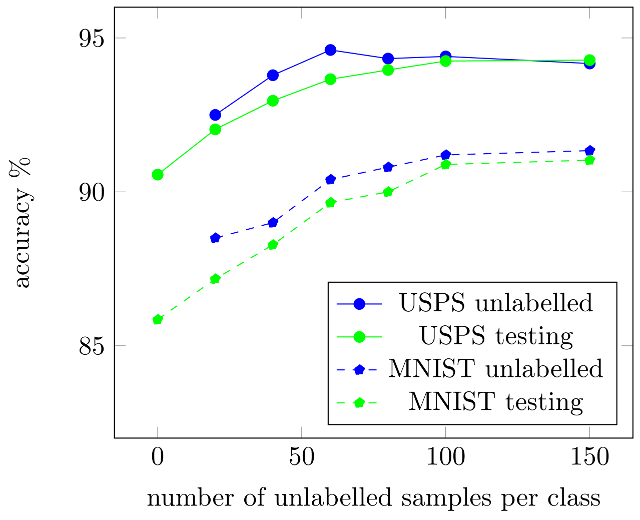


Figure III.1: The accuracy rate for unlabelled samples and for testing samples in two databases USPS and MNIST with different number of unlabelled samples per class: 20, 40, 60, 80, 100, 150.

In the second test, we compare SSDL-GA with other SSDL approaches. For SSDL-GA, we keep the same pre-processing and hyper-parameters as in the first test. We set up the training set (labelled samples, unlabelled samples) and testing set as follows:

- For the MNIST database, we randomly select 200 images from each class, in which 20 images are used for labelled samples, 80 images are used for unlabelled samples and 100 images remain as testing samples.
- For the USPS database, we randomly select 110 images from each class, in which 20 images are used for labelled samples, 40 images are used for unlabelled samples and 50 images remain as testing samples.

Five random samplings were conducted to calculate the mean and standard deviation on testing set. The table III.6 shows the accuracy rate of various SSDL approaches: OSSDL [Zhang *et al.* 2013], S2D2 [Shrivastava *et al.* 2012], SSR-D [Wang *et al.* 2013], SSP-DL [Wang *et al.* 2016], USSDL [Wang *et al.* 2015], PSSDL [Babagholami-Mohamadabadi *et al.* 2013], SSD-LP [Chen & Yang 2017], also Convolutional Neural Network (CNN, supervised) and Label Propagation (LP, semi-supervised) [Zhou *et al.* 2004]. For CNN, only labelled samples are used for training and the shown results are the best average accuracy rate after testing three different simple CNN models. Here are configurations used for MNIST and USPS respectively:

- $Conv[32 \times 3 \times 3] \rightarrow ReLU \rightarrow BNorm \rightarrow Conv[32 \times 3 \times 3] \rightarrow ReLU \rightarrow Pool[2 \times 2] \rightarrow BNorm \rightarrow Conv[64 \times 3 \times 3] \rightarrow ReLU \rightarrow BNorm \rightarrow Conv[64 \times 3 \times 3] \rightarrow ReLU \rightarrow Pool[2 \times 2] \rightarrow BNorm \rightarrow FC[512] \rightarrow ReLU \rightarrow BNorm \rightarrow Dropout[0.25] \rightarrow FC[10] \rightarrow softmax$.
- $Conv[16 \times 3 \times 3] \rightarrow ReLU \rightarrow BNorm \rightarrow Conv[32 \times 3 \times 3] \rightarrow tanh \rightarrow Pool[2 \times 2]\ BNorm \rightarrow FC[128] \rightarrow tanh \rightarrow BNorm \rightarrow Dropout[0.25] \rightarrow FC[10] \rightarrow softmax$.

For both dataset, the contribution of the manifold structure preservation with LLE can be assessed by comparing SSDL-GA and USSDL (SSDL-GA is equivalent to USSDL when $\beta = 0$). Secondly, the SSDL-GA outperforms other SSDL methods, as well as the CNN and LP. We notice that on MNIST, SSDL-GA is the only one in the SSDL family that can outperform the CNN. This can be trivially explained by the fact that the CNN does not make use of unlabelled samples. Therefore, we evaluate this test with a semi-supervised neural network models: the Ladder net [Rasmus *et al.* 2015]. We provide more details of this model in section IV.A.2 (denoising). Here is configuration used in Ladder net: $FC[1000] \rightarrow ReLU \rightarrow FC[500] \rightarrow ReLU \rightarrow FC[250] \rightarrow softmax$ and $\sigma_{noise} = 0.3$. We see that our model gets slightly better accuracy rate compared to the Ladder net.

Complexity comparison. As the complexity for several SSDL algorithms (in table III.1) are not communicated and we do not have access to the implementation for these algorithms, we only compare the complexity for each step in the optimization process. First of all, objective

Method / Data	USPS	MNIST	Sparse coding nature
LP	90.3 ± 1.3	85.12 ± 0.6	
OSSDL*	80.8 ± 2.8	73.2 ± 1.8	individual, l_0
S2D2*	86.6 ± 1.6	77.6 ± 0.8	group, l_1
SSR-D*	87.2 ± 0.5	83.8 ± 1.2	individual, $l_{2,p}$
SSP-DL*	87.8 ± 1.1	85.8 ± 1.2	group, $l_{2,p}$ and $l_{p,p}$
USSDL	91.56 ± 1.15	84.8 ± 1.7	individual, l_1
PSSDL$^\diamond$	86.9 ± 1.0	87.4 ± 1.2	group, l_1
SSD-LP*	90.3 ± 1.3	87.8 ± 1.6	group, l_1
SSDL-GA	$\mathbf{93.6 \pm 1.0}$	$\mathbf{90 \pm 0.8}$	group, l_1
CNN	89.28 ± 1.4	88.4 ± 1.1	
Ladder	92.68 ± 1.0	89.84 ± 0.8	

Table III.6: Accuracy rate and nature of sparse coding for various SSDL methods, with handwritten digits databases USPS and MNIST. CNN means Convolutional Neural Network. The Ladder net [Rasmus *et al.* 2015] is neural network model trained in a semi-supervised way. ($\diamond$) In PSSDL, for each class, we use 25 images as labelled samples instead of 20 and the rest of the training data as unlabelled samples; its corresponding accuracy rate is extracted from the original paper. (*) Accuracy rate are extracted from [Chen & Yang 2017]. In sparse coding nature column, individual means that sparse codes computation is independent for each sample and group means that sparse codes computation is dependent between samples.

function of SSDL algorithms are iteratively optimized by following steps: sparse coding, dictionary update, classifier update and probability update. The first two steps (solved by iterative method) are the most essential and influence mostly to the complexity. The remaining steps can be solved trivially by first order optimality. Secondly, dictionary update is just slightly different among the compared algorithms (which depends on the shared dictionary approach or the specific class sub-dictionaries approach). However, in general, the quadratic cost $\|\mathbf{X} - \mathbf{DA}\|_2^2$ is minimized for $\mathbf{D}$ with the constraint $\mathbf{D} \in \mathcal{C}$. Therefore, the complexity of this step can be regarded as equivalent among the compared algorithms. By two observations, sparse coding step is the essential factor for the complexity comparison.

The complexity of sparse coding then depends essentially on two factors: the type of sparse coding and the norm used in sparse coding. On the one hand, if the sparse codes computation is independent for each sample, then the sparse coding step is parallelizable. This does not hold true for sparse coding in the approaches that use manifold structure preservation $\mathcal{F}$ or Fisher Discriminant Analysis. Besides, there are several norms used in sparse coding $l_0; l_1; l_{2,q}; l_{q,q}; (0 < q < 1)$. With the pseudo-norm l_0, low complexity greedy algorithm can be used (i.e. MP-derived algorithm). These two factors are showed for each SSDL method in table III.6, and we can see clearly the trade-off between complexity and accuracy.

B.3.3 Face databases

In this subsection, we evaluate our approach with Extended YaleB cropped and AR cropped dataset for which the size of each image is respectively 192×168 and 165×120 pixels.

The YaleB dataset contains 2432 frontal-face images of 38 individuals (64 images for each individual), captured under various illumination conditions and expressions. We first resize images to 54×48 before applying a Principal Component Analysis (PCA) to obtain 300 dimensional feature vectors (same process as [Wang *et al.* 2015, Chen & Yang 2017]). Then each vector coordinate is normalized to have zero mean and unit variance. Finally, each vector is normalized to have l_2 unit norm and then multiplied by 2. We randomly select $N = 20$ images for each person to create a training set and use the remaining images for the testing set. In the training set, for each person, we use $N_l = \{2, 5, 10\}$ images as labelled samples and the remaining images as unlabelled samples. Five independent evaluations were conducted to compute the mean and standard deviation.

The results are shown in table III.7 with various SSDL approaches. In the case $N_l = 2$ (very few labelled samples), our approach improves significantly the accuracy rate compared to other SSDL methods but it is less accurate in the cases $N_l = \{5, 10\}$ compared to SSD-LP. The hyper-parameters values in three cases are $N_l = \{2, 5, 10\}$: $p = 380, \lambda = 0.005, \beta = 2, \gamma = 0.5, \varphi = 0, \mu = 1, k = 4, r = 1.5$.

Method / N_l	2	5	10
S2D2*	53.4 ± 2.1	76.1 ± 1.3	83.2 ± 1.9
JDL*	55.2 ± 1.8	77.4 ± 2.8	85.3 ± 1.6
USSDL*	60.5 ± 2.1	86.5 ± 2.1	93.6 ± 0.8
SSD-LP*	67.0 ± 2.9	$\mathbf{89.8 \pm 0.9}$	$\mathbf{95.2 \pm 0.2}$
SSDL-GA	$\mathbf{73.62 \pm 3.1}$	86.6 ± 1.6	90.7 ± 0.4

Table III.7: Accuracy rate for YaleB database with various SSDL approaches and different number of labelled samples in training. (*) Accuracy rate are extracted from [Chen & Yang 2017].

The AR Face database consists of more than 4000 images but we evaluate our approach with its subset that consists of 2600 images (26 images per person for 50 male subjects and 50 female subjects). These 26 images are taken from different facial expressions, illumination conditions, and occlusions (sun-glasses and scarves). First, this dataset is projected into a 540-dimensional feature vector by a randomly generated matrix. Then we perform the same pre-processing as described in YaleB. For each person, 20 images are randomly selected for training set and 6 resting images are for testing set. In two SDL methods: LC-KSVD and SDGDL, all 20 training images are labelled samples. On the other hand, in SSDL-GA, 15 images are randomly selected from 20 training samples and used as labelled samples and the 5 resting images are used as unlabelled training samples.

Table III.8 shows accuracy rates for above mentioned methods. Although SSDL-GA uses fewer labelled samples in the training set, it gives a better accuracy rate. This can be explained by the manifold learning method used (LLE instead of LE in SDGDL) and by using internal semi-supervised classifier. Although LLE and LE are both Laplacian-based methods but LLE is not sensible to the similarity metric $d_x()$ used to measure distance between two samples in the original representation. The hyper-parameters used are $p = 300, \lambda = 0.0015, \beta = 0.3, \gamma = 0.08, \varphi = 0, \mu = 0.016, k = 8, r = 1.5$.

Method	Accuracy rate
LC-KSVD1*	84.17
LC-KSVD2*	85
SDGDL*	84.93
SDGDL-L*	85.33
SSDL-GA	**92.09 $\pm$ 1.16**

Table III.8: Accuracy rate for the AR database with SDL methods. (*) Accuracy rate are extracted from [Yankelevsky & Elad 2017]

.

B.4 Conclusion about proposed method. We have presented a SSDL method by integrating manifold structure preservation and an internal semi-supervised classifier (by pseudo-label) to the classical DiL problem. This helps to exploit more information from unlabelled samples to reinforce the model. In addition, new unlabelled samples are also sparse coded by taking into account manifold structure preservation. Experimental results on several benchmark databases have shown the advantage of our approach, especially in the case of and low number of unlabelled samples in training, it performs about 2% better than the state-of-art for digit recognition compared to other SSDL approaches and gets slightly better accuracy compared to Ladder net, a semi-supervised neural network. We also propose a batch and epoch version in the appendix to deal with the case of the large number of training samples.

C Conclusion

We have presented a SSDL method by integrating manifold structure preservation to the classical DiL problem. This helps to exploit more information from unlabelled samples to reinforce the model. In addition, new unlabelled samples are also sparse coded by taking into account manifold structure preservation. Finally, we train an internal classifier in a semi-supervised way, using pseudo-labelling technique.

Experimental results on several benchmark databases have shown the advantage of our approach, especially in the case of and low number of unlabelled samples in training, it performs about 2% better than the state-of-art for digit recognition compared to other SSDL approaches and gets slightly better accuracy compared to semi-supervised neural network. We also propose a batch and epoch version in the appendix to accelerate the optimization process. However, in general, dictionary learning methods for classification objectives, due to limits of computation, require a dimensionality reduction to fewer than about 10^3 dimensions. Applying dimensionality reduction can make an important loss of discriminatory information. Hence, in the next chapter, we develop a semi-supervised deep learning method to deal with dataset that has large size of sample, which is a shortcoming of dictionary learning model.

Chapter IV

Semi-supervised deep learning

Chapter abstract

In this chapter, we present a survey of semi-supervised learning methods which have been developed for neural network model (NNM) and CNN in particular. Then we introduce a new semi-supervised NNM learning method, grounded in manifold learning methodology. We show that our approach provides non only an improvement for the accuracy rate but also for the robustness to adversarial examples, compared to state-of-the-art semi-supervised NNM learning.

A Related works

We provide first notations used in this section and a taxonomy of semi-supervised neural networks (SSNN). Then we present SSNN models for each category.

A.1 Notations.

- $\mathcal{L}_s$ and $\mathcal{L}_u$ are respectively supervised cost and unsupervised cost.
- $f()$ the complete model, which outputs the predicted class probability.
- $f^{(:l)}()$ upstream part of $f()$ that outputs the intermediate (latent) representation $z^{(l)}$ of layer (l) and takes sample x as input.
- $f^{(l:)}()$ downstream part of $f()$ that outputs the predicted class probability and takes intermediate (latent) representation $z^{(l)}$ of layer (l) as input.
- $f^{(l)}()$ layer (l) in model $f()$; it takes $z^{(l-1)}$ as input and outputs $z^{(l)}$.

- x, x^l and x^u are respectively a sample, a labelled sample and an unlabelled sample.
- n is the number of dimensions of x, hence $x \in \mathbb{R}^n$.
- $\mathcal{X}^l$ and $\mathcal{X}^u$ are sets that contain all labelled and unlabelled samples respectively. $\mathcal{X} = \mathcal{X}^l \cap \mathcal{X}^u$ is the complete set.
- N_l is the number of labelled samples and N_u is the number of unlabelled samples, hence $N = N_u + N_l$ is the total number of samples.
- Conventionally, samples are sorted as $\mathcal{X} = \{x^l_1, x^l_2, .., x^l_{N_l}, x^u_1, x^u_2, .., x^u_{N_u}\}$.
- C is the number of classes.
- y_i is the class of the sample x^l_i and $y_i \in \{1, 2, .., C\}$ or is the corresponding one hot vector when y_i is used in a cost function, *e.g.* $d(f(x_i), y_i)$.
- d_s and d_r are dissimilarity metric, respectively for measuring supervision loss and reconstruction loss.
- d_s is usually Cross Entropy d_{CE} and d_r is usually Mean Square Error d_{MSE} or Binary Cross Entropy d_{BCE}. In addition, to measure the dissimilarity between two distributions, d_{KL} is usually used.

$$d_{\text{CE}}(a,b) = -\sum_k^n \log(a[k])b[k] \text{ where } a, b \text{ are two distributions.} \tag{IV.1}$$

$$d_{\text{BCE}}(a,b) = -\sum_k^n \big(a[k]\log(b[k]) + (1-a[k])\log(1-b[k])\big) \text{ where } 0 < a[k], b[k] < 1. \tag{IV.2}$$

$$d_{\text{MSE}}(a,b) = \frac{1}{n}\sum_k^n (a[k]-b[k])^2 \tag{IV.3}$$

$$d_{\text{KL}}(a,b) = \sum_k^n a[k]\log\frac{a[k]}{b[k]} \text{ where } a, b \text{ are two distributions.} \tag{IV.4}$$

In general, the cost function for semi supervised neural network learning is defined as follows:

$$\mathcal{L} = \mathcal{L}_s + \lambda\mathcal{L}_u \tag{IV.5}$$

where λ weights the unlabelled samples related component of the cost. In some works, λ is a function that increases progressively from 0 (at epoch $t = 1$) to a fixed value at a given epoch.

By default, $\mathcal{L}_s$ is given by:

$$\mathcal{L}_s = \frac{1}{N_l}\sum_{i=1}^{N_l} d_s\big(f(x_i^l), y_i\big) \tag{IV.6}$$

For each method, we provide the explicit form of $\mathcal{L}_u$.

Taxonomy. We group SSNN methods according to the following categories:

- **Auxiliary task as regularization**
  + *Reconstruction*
  + *Denoising*
  + *Entropy minimization*
  + *Consistency constraint*
  + *Contrastive representation*
  + *Self-supervised learning*
  + *Manifold learning*
- **Pseudo labeling**
- **Generative models**
  + *Variational Autoencoders*
  + *Generative Adversarial Networks*
- **Virtual Adversarial Training**
- **Holistic methods**

It is worth noting that there are also semi-supervised neural network approaches based on graph, called GNN (Graph Neural Network, see [Zhou *et al.* 2019] for a review). However, these approaches require graph information between samples. Since the latter does not exist in several standard datasets (image classification) that we work on, we do not present GNNs in this thesis.

A.2 Auxiliary task as regularization. In the first category, an auxiliary loss is added to the label prediction loss. This auxiliary loss which does not use label information aims at enforcing model outputs properties derived from the assumptions stated at the beginning of section III.A. In the following, we analyze each auxiliary task from this point of view before diving into more details.

A.2.1 Reconstruction

Here the auxiliary task is the reconstruction task. The mechanism of reconstruction task relies on the manifold assumption. One assumes that data lies on a low-dimensional manifold. The model can in principle produce simpler data representations through its hidden layers without loss of information. We call these representations *latent representations* hereafter. In order to ensure that latent representations are relevant with respect to unlabeled samples, they are used to reconstruct the unlabeled samples in the original space using loss $\mathcal{L}_u$ as:

$$\mathcal{L}_u = \frac{1}{N} \sum_{x \in \mathcal{X}} d_r(\hat{x}, x) \tag{IV.7}$$

where $\hat{x} = g \circ f^{(:l)}(x)$, with $g()$ is a NNM that maps a latent representation $f^{(:l)}(x)$ back to the original representation x. $g \circ f^{(:l)}()$ is basically an AE. d_r is a dissimilarity measure (for reconstruction), that can chosen as the MSE. However, reconstructing x from its latent representation $f^{(:l)}(x)$ can be difficult because $f()$ is primarily optimized for classification and tends to filter out non-discriminative features, which are necessary for reconstruction task. Therefore [Robert *et al.* 2018] proposed the HybridNet, in which, the following AE architecture h is used instead:

$$\mathcal{L}_u = \frac{1}{N} \sum_{x \in \mathcal{X}} d_r(\hat{x} + \hat{x}_c, x) \tag{IV.8}$$

where $\hat{x}_c = h(x)$ complements $\hat{x}$ in the sense that it is the reconstruction from non-discriminative features.

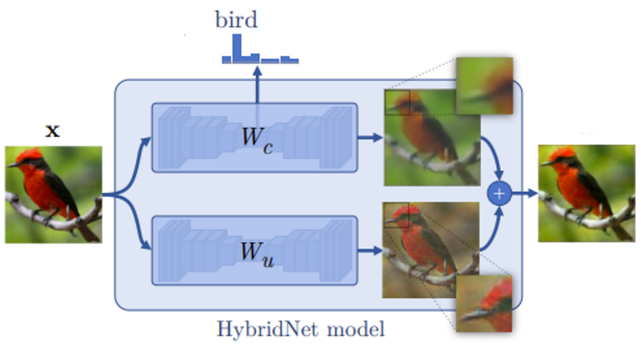


Figure IV.1: HybridNet model for semi-supervised learning by reconstruction. Source: [Robert *et al.* 2018]. W_c corresponds to $g \circ f^{(:l)}()$ and W_u corresponds to h in our notations.

The resulting cost $\mathcal{L}_u$ is illustrated in figure IV.1. If encoder $f^{(:l)}()$ uses convolutional layers, decoder $g()$ should contain some operations (layers) which aim at inverting convolutional layers, in order to be compatible with the encoder. These operations can be inspired from CNN models for

Image Segmentation such as: Fully Convolutional Networks [Long *et al.* 2014], U-Net [Ronneberger *et al.* 2015], V-Net [Milletari *et al.* 2016] for 3D Image.

A.2.2 Denoising

This auxiliary loss complexifies the previous one introducing noise at different levels. The smoothness assumption is used by enforcing reference entries and their slightly perturbed versions to have close latent representations throughout the network.

Enforcing noise robustness into neural network training was introduced in [Valpola 2014] with the Ladder network architecture. Then [Rasmus *et al.* 2015] proposed to use a denoising loss as an auxiliary constraint to integrate unlabeled data in the neural network learning. Figure IV.2 shows an example of Ladder network with 2 layers ($L = 2$). It is an AE architecture, for which the decoder, i.e the reconstruction branch, has the same (but reversed) architecture as the encoder, with the so-called skip connections that link layers in encoder to the corresponding ones in decoder. Hence the name Ladder network. For simplicity sake, we consider a Ladder network which consists of only Fully Connected layers, but it can be extended to CNN.

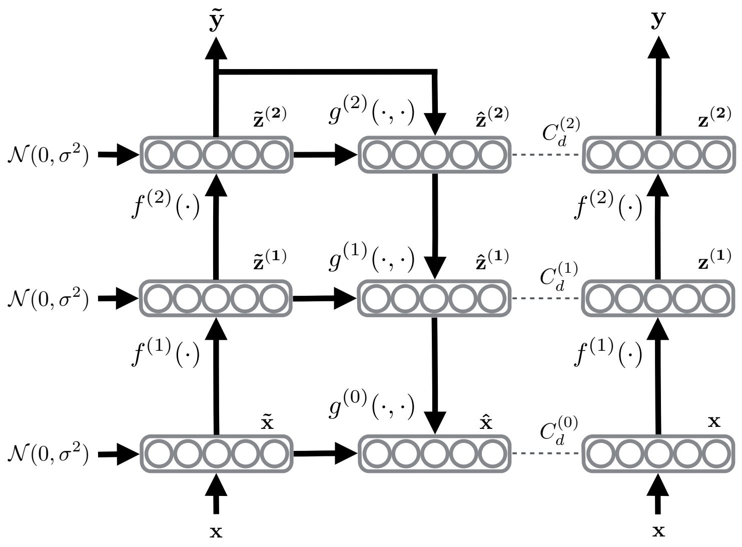


Figure IV.2: Semi-Supervised Learning with Ladder Network, using denoising as auxiliary task. Source: [Rasmus *et al.* 2015].

As illustrated in figure IV.2, the encoder performs two forward passes, a corrupted one and clean one. In the corrupted forward pass, Gaussian noise $\mathcal{N}(0, \sigma^2)$ is injected at input layer and each layer (l), yielding a noisy hidden representation $\tilde{z}^{(l)}$. The clean forward pass outputs noise-free hidden representation $z^{(l)}$ at the layer (l).

In the decoder, a denoiser $g^{(l)}(.,.)$ is added at each layer (l), as follows: $f_{\text{dec}}()^{(l-1)} \rightarrow g^{(l-1)}(.,.) \rightarrow$

$f_{\text{dec}}()^{(l)} \rightarrow g^{(l)}(.,.) \rightarrow \ldots$ Let $u^{(l)}$ be the output of $f_{\text{dec}}^{(l)}()$. Then $g^{(l)}(.,.)$ transforms its noisy input $\tilde{z}^{(l)}$ into its denoised version $\hat{z}^{(l)}$ as follows:

$$\hat{z}^{(l)}[j] = g^{(l)}(\tilde{z}^{(l)}[j], u^{(l)}[j]) = \big(\tilde{z}^{(l)}[j] - \mu^{(l)}[j]\big)v^{(l)}[j] + \mu^{(l)}[j] \tag{IV.9}$$

where j is a unit index in a hidden representation. $\mu^{(l)}[j]$ and $v^{(l)}[j]$ are computed from $u^{(l)}[j]$ as

$$\begin{aligned}\mu^{(l)}[j] &= d_1^{(l)}[j]\text{sigmoid}(d_2^{(l)}[j]u^{(l)}[j] + d_3^{(l)}[j]) + d_4^{(l)}[j]u^{(l)}[j] + d_5^{(l)}[j]\\ v^{(l)}[j] &= d_6^{(l)}[j]\text{sigmoid}(d_7^{(l)}[j]u^{(l)}[j] + d_8^{(l)}[j]) + d_9^{(l)}[j]u^{(l)}[j] + d_{10}^{(l)}[j]\end{aligned} \tag{IV.10}$$

where the vectors $d_i^{(l)}$ have the same size as $u^{(l)}$ and $\tilde{z}^{(l)}$, and $(i = 1, .., 10)$.

Then, the auxiliary cost $\mathcal{L}_u$ is the sum of the dissimilarities between denoised representations $\hat{z}^{(l)}$ and noise-free representations $z^{(l)}$:

$$\mathcal{L}_u = \frac{1}{N}\sum_{l=1}^{L}\sum_{i=1}^{N}\lambda_l d_{\text{MSE}}(z_i^{(l)}, \hat{z}_i^{(l)}) \tag{IV.11}$$

where λ_l weights the contribution of layer (l). The denoisers parameters are trainable, as well as the model parameters on the encoder and the decoder branch.

Despite its good performance for classification task, semi-supervised learning through the Ladder network architecture is computionally expensive for large datasets and/or very deep model, compared to a conventional training. In terms of memory, noisy latent representation $\tilde{z}$, and denoised latent representation $\hat{z}$ have to be stored additionally. In terms of algorithmic complexity, instead of one backpropagation (gradient) from the last layer as usual, additional backpropagations from all intermediate layers (see IV.11) have to be performed.

To mitigate these complexity issues, the authors proposed a simplified variant called Γ-Model, where $\lambda_l = 0$ when $l < L$, which amounts to perform denoising at the top layer only.

A.2.3 Entropy minimization

The auxiliary loss penalizes the entropy of output probabilities. This constraint relies on the clustering assumption. Entropy minimization (EntMin) [Grandvalet & Bengio 2004] encourages the network to make a low-entropy prediction for each unlabelled sample, regardless of its predicted class:

$$\mathcal{L}_u = -\frac{1}{N_u}\sum_{i=1}^{N_u}\sum_{k=1}^{C} f(x^u)[k]\log(f(x^u)[k]) \tag{IV.12}$$

Figure IV.3 shows two output sets of probabilities, one with high entropy and the other with low entropy. With low entropy of output probability, we expect to have a dominant predicted class probability (sharpness, or low uncertainty).

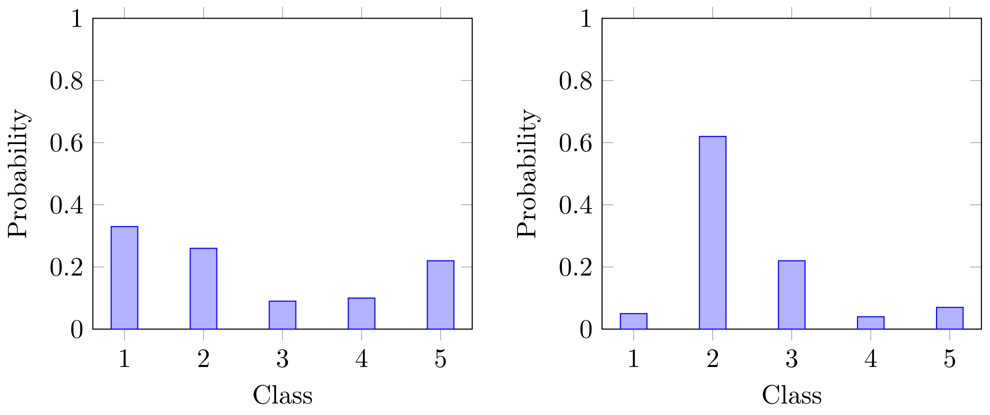


Figure IV.3: Left: High entropy $H(x) = 1.5$. Right: Low entropy $H(x) = 1.1$. Output probability $f(x)$ is "sharper" with lower entropy. In this case the number of classes $C = 5$.

However, three issues need to be considered with EntMin. Firstly, in the case of multi-label problem, the number of classes that a sample belongs to, must be small compared to the total number of classes. Secondly, lower entropy does not guarantee a higher accuracy rate, because false predictions with high certainty would result in a low entropy. Finally, it appears that using only EntMin as auxiliary cost seems inefficient to produce competitive results compared to other SSL methods ([Oliver *et al.* 2018]).

A.2.4 Consistency constraint

The auxiliary loss in this case enforces the *consistency* between the output associated to a sample x and the one associated to its perturbed version $\tilde{x}$. The notions of consistency and perturbation will be clarified thought the examples of the Π-Model, Temporal Ensembling, Mean teachers and Dual Student presented hereafter.

Π-Model proposed by [Laine & Aila 2016] uses the following auxiliary loss:

$$\mathcal{L}_u = \frac{1}{N} \sum_{i=1}^{N} d_{\text{MSE}}(z_i, \tilde{z}_i) \tag{IV.13}$$

where z_i and $\tilde{z}_i$ are the model outputs associated with the inputs x_i and its perturbed version respectively. Perturbation are performed using two methods: stochastic augmentation and stochastic model.

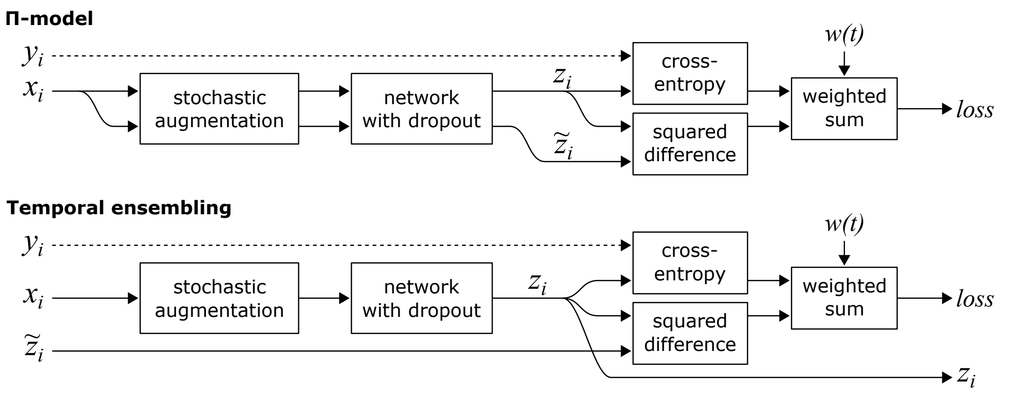


Figure IV.4: Π-Model and Temporal Ensembling model. Source: [Laine & Aila 2016].

Stochastic augmentation consists in applying a transformation with a random magnitude to x_i each time it is presented to the network. It might be a translation with a random step size or an additive noise with a random standard deviation. The stochastic model affects the neural network model parameters themselves, for instance using the dropout technique.

Temporal Ensembling is an improved version of Π-Model. In Π-Model, two stochastic forward passes are performed for all samples in a batch and then the model is updated with a consistency constraint between two stochastic outputs. Temporal Ensembling goes a step further by taking into account also the irregularities along the model parameters optimization path. The training epoch outputs z_i is cumulated into an ensemble output Z_i using an exponentially moving average (EMA) update:

$$Z_i \leftarrow \alpha Z_i + (1-\alpha) z_i \tag{IV.14}$$

where $0 \leq \alpha < 1$. The auxiliary cost in Temporal Ensembling is the same as in Π-Model except that $\tilde{z}_i$ is replaced by Z_i which stabilizes the model updates. Moreover, Temporal Ensembling requires only a single forward pass for a given sample instead of two forward passes in the Π-Model. However, the ensemble output Z_i needs to be stored across epochs.

Mean teachers: In this approach, one considers that there are a teacher model and a student model. The role of teacher is to generate a target, *i.e.* $\tilde{z}_i$ in Π-Model or Z_i in Temporal Ensembling, for a given unlabelled sample x_i. The role of the student model is to update its parameters for mapping x_i to the target generated by the teacher model. Following this teacher-student strategy, [Tarvainen & Valpola 2017] proposed a method called Mean teachers that updates the teacher model using an EMA:

$$\theta'_t \leftarrow \alpha \theta'_{t-1} + (1-\alpha)\theta_t \tag{IV.15}$$

where θ'_t denotes the parameters of the teacher model $f_{\theta'}$ and θ_t the parameters of student model f_θ, at time t. The form of the auxiliary cost for Mean Teachers remains the same as for the Π-Model and Temporal Ensembling:

$$\mathcal{L}_u = \frac{1}{N}\sum_{i=1}^{N} d_{\text{MSE}}\big(f_\theta(x_i), z'_i\big), \tag{IV.16}$$

where $z'_i = f_{\theta'}(x_i)$ is the target generated by teacher.

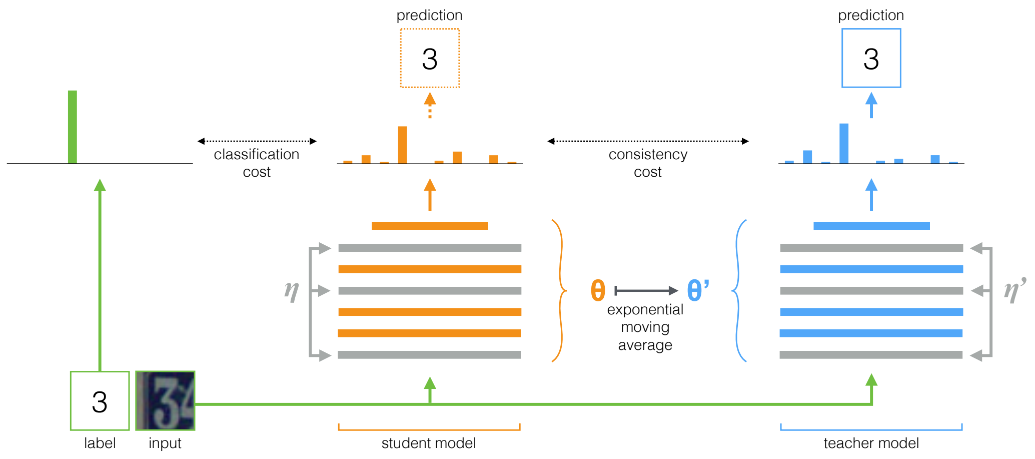


Figure IV.5: Mean Teachers model. Source: [Tarvainen & Valpola 2017]

Figure IV.5 illustrates Mean Teachers model. η and η' are perturbations in the student and the teacher model respectively. Both student model and teacher model can be used for prediction of test samples, but teacher model is more likely to be correct.

Dual Student In Mean Teachers, since the parameters of the teacher model are updated from the student model's, there is a strong dependency between the two models. If the student misclassifies an unlabelled sample after several epochs, so does the teacher and this can not be changed because they are closely related. To deal with this problem, [Ke *et al.* 2019] proposed an approach called Dual Student which uses two independent models f_{θ^1} and f_{θ^2}. At a given training step and for a given sample x_i, one plays role of teacher and other plays role of student. To choose which model plays the role of the teacher and the role of the student, the stability and confidence of each model is estimated. Here are the two auxiliary costs associated to the two models, $\mathcal{L}_u^1$ and $\mathcal{L}_u^2$:

$$\begin{aligned}\mathcal{L}_u^1 &= \frac{1}{N}\sum_{i=1}^{N}\Big(d_{\text{MSE}}\big(f_{\theta^1}(x_i), \tilde{z}_i^1\big) + \gamma\Omega_i^1 H\big(\mathcal{E}_i^2 - \mathcal{E}_i^1\big) d_{\text{MSE}}\big(f_{\theta^1}(x_i), z_i^2\big)\Big)\\ \mathcal{L}_u^2 &= \frac{1}{N}\sum_{i=1}^{N}\Big(d_{\text{MSE}}\big(f_{\theta^2}(x_i), \tilde{z}_i^2\big) + \gamma\Omega_i^2 H\big(\mathcal{E}_i^1 - \mathcal{E}_i^2\big) d_{\text{MSE}}\big(f_{\theta^2}(x_i), z_i^1\big)\Big)\end{aligned} \tag{IV.17}$$

where Ω_i^j and $\mathcal{E}^j$ quantify the confidence of the model j for the sample i, and its stability respectively, H is the Heaviside function, $z_i^j = f_{\theta^j}(x_i)$ and $\tilde{z}_i^j = f_{\theta^j}(\tilde{x}_i)$ are respectively the output of model j for sample x_i and its perturbed version $\tilde{x}_i$. The confidence Ω_i^j takes value 1 if both two following conditions are satisfied:

1. The predicted label for sample x_i and for its perturbed version $\tilde{x}_i$ are the same, which means $\underset{c}{\operatorname{argmax}}\, f_{\theta^j}(x_i)[c] = \underset{c}{\operatorname{argmax}}\, f_{\theta^j}(\tilde{x}_i)[c]$.
2. $\|f_{\theta^j}(x_i)\|_\infty > \xi$, where ξ is a threshold. This means the largest class probability in output must be greater than a threshold.

Otherwise, Ω_i^j takes value 0. In the case that both two models satisfy the confidence conditions for a given sample, the stability criterion is used, which is computed as $\mathcal{E}_i^j = \|f_{\theta^j}(x_i) - f_{\theta^j}(\tilde{x})\|_2^2$.

The drawback of this approach is that it is computationally heavier than the basic Teacher-Student approach.

A.2.5 Contrastive representation

This task can be considered an extended version of consistency task, but is usually applied to hidden representations instead of the output as in consistency representation. In addition to the consistency constrain for two similar inputs, the hidden representations of two dissimilar inputs are forced to be far away. Hence, this strategy relies on the smoothness assumption and the manifold assumption. Here is an example of contrastive loss function, sometimes called triplet loss:

$$\mathcal{L}_{triplet} = \max\Big(d\big(h(x_i), h(x_p)\big) - d\big(h(x_i), h(x_n)\big) + \alpha, 0\Big) \tag{IV.18}$$

where x_i is an input, x_p is a positive input and x_n is a negative input. $h()$ is a mapping function and $d()$ is the pairwise dissimilarity metric, *e.g.* $d\big(h(x_i), h(x_p)\big) = \|h(x_i) - h(x_p)\|_2^2$. While a consistency loss would only consist in $d\big(h(x_i), h(x_p)\big)$, contrastive representation goes further by adding the term $-d\big(h(x_i), h(x_n)\big)$ which enforces $h(x_i)$ and $h(x_n)$ to be far apart.

We illustrate this approach with slimCLR proposed by [Chen *et al.* 2020]. For a batch that contains N_b samples x_i where $i = 1, .., N_b$, two stochastic forward passes are performed with model $g \circ f^{(:l)}$ where g is a NNM that maps the latent representation $f^{(:l)}$ to the expected contrastive representation. Hence, two outputs are obtained: $z_{2i-1} = g \circ f^{(:l)}(x_i)$ and $z_{2i} = g \circ f^{(:l)}(x_i)$. The objective is to force z_{2i-1} and z_{2i} to have the same representation. Moreover, this representation has to be far from those of other samples. The pairwise similarity between z_i and z_j is defined as

$s_{i,j} = \frac{z_i^\top z_j}{\|z_i\|\|z_j\|}$, where $i = 1, .., 2N_b$ and $j = 1, .., 2N_b$, then the contrastive loss is defined as:

$$l(i,j) = -\log \frac{\exp(s_{i,j})^{\frac{1}{\tau}}}{\sum_{k=1,k\neq j}^{2N_b} \exp(s_{i,k})^{\frac{1}{\tau}}} \tag{IV.19}$$

where $x^{\frac{1}{\tau}}$ can be considered as activation function with parameter τ (in appendix A.8.2). Then the auxiliary cost $\mathcal{L}_u$ for a batch is described as following:

$$\mathcal{L}_u = \frac{1}{2N_b} \sum_{i=1}^{N_b} \big(l(2i-1, 2i) + l(2i, 2i-1)\big) \tag{IV.20}$$

A.2.6 Self-supervised learning

This field encompasses all methods that generate automatically some kind of supervisory signal. The latter can be used to train a model to learn relevant features, hence the most obvious application of self-supervised learning is the transfer learning. For example, a model is first trained using a generate supervisory signal. Then the upper layers of the model are used for features extraction. Since it does not require annotation by human, self-supervised learning can be considered as a type of unsupervised learning. Therefore it can be combined with supervised learning to create a semi-supervised learning method. Figure IV.6 gives an example of such an architecture. Each sample x_i is rotated by a random angle $\bar{y}_i \in \{0°, 90°, 180°, 270°\}$, yielding couples $(\bar{x}_i, \bar{y}_i)$. These synthetic labels are used to learn the upstream part of the model, while the actual labels available are used to optimize the model globally. The reconstruction task based regularization can be considered a special case of self-supervised learning.

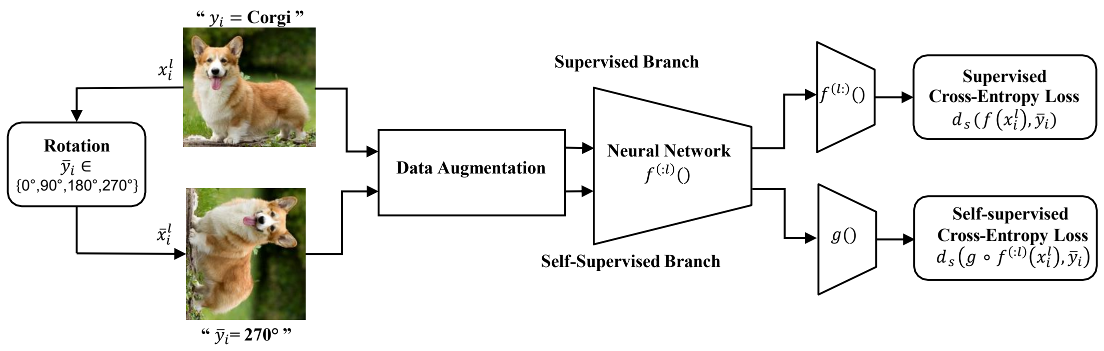


Figure IV.6: Semi-supervised learning by combining supervised learning with self-supervised learning. Here the classification task in self-supervised learning consists in recognizing the angle of rotation. The couple $(\bar{x}_i, \bar{y}_i)$ is automatically generated. Source: developed from [Tran 2019].

The auxiliary cost in this case takes the following form:

$$\mathcal{L}_u = \frac{1}{N}\sum_{i=1}^{N} d_s\big(g \circ f^{(:l)}(\bar{x}_i), \bar{y}_i\big) \tag{IV.21}$$

where $g()$ is a NNM that maps a latent representation $f^{(:l)}()$ to self-generated corresponding target $\bar{y}_i$. In the following, we present some popular mock tasks used in self-supervised learning:

- Relative Patch Location [Doersch *et al.* 2015]. This method proposes to crop a patch in the center of image and an other patch in a given relative position with respect to the first one (8 positions corresponding to 8 classes). The auxiliary task is to predict the position of the second patch relative to the first. In this case, the model takes two patches as inputs and has to output a target within the eight positions. In same spirit, Jigsaw puzzle was proposed by [Noroozi & Favaro 2016]. In this task, nine patches are randomly permuted and fed into the model and the objective is to predict which permutation has been performed.

- Rotation [Gidaris *et al.* 2018]. This mock task is illustrated by figure IV.6 and presented in details at the beginning of this subsection.

- Exemplar [Dosovitskiy *et al.* 2014]. In this mock task, N patches are sampled from different images by focusing on areas that have considerable gradients because they can contain objects or parts of objects. They are called exemplary patches. Then, each patch is distorted by applying a variety of random transformations, *i.e.* translation, rotation, scaling, color jittering, etc. All the resulting distorted patches from the same exemplary patch are in the same class, hence we have N classes. Finally, the objective is to predict the class of a distorted patch.

- Colorization [Zhang *et al.* 2016]. As suggested by its name, this method takes as input a grey level image and outputs its colored version. Each unlabelled image is converted from RBG to L*a*b* color space. Only the luminance information (L*) is kept for each pixel, which results into a grey level image. Finally, the objective is to predict a* and b* parameters.

- One can think of mock tasks consisting of reconstructing the original signal from an altered version (values removal, noise, ...).

A survey for self-supervised learning can be found in [Kolesnikov *et al.* 2019, Weng 2019]. Some results with rotation and exemplar as mock tasks can be found in [Zhai *et al.* 2019].

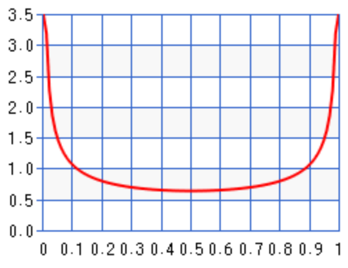
A.2.7 Manifold learning based

Manifold learning losses have used in conjunction with several models to build semi-supervised learning techniques. We have presented some examples for DiL including our own contribution in III.A. An example for SVM model can be found in [Zhili Wu *et al.* 2006]. It has been introduced for neural network models in [Weston & Ratle 2008]. The auxiliary task $\mathcal{L}_u$ in this case takes the form:

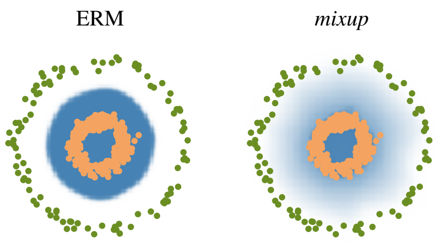
$$\mathcal{L}_u = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_e(z_i^{(l)}, \mathcal{A}\backslash\{z_i^{(l)}\}) \tag{IV.22}$$

where $\mathcal{L}_e$ is the embedding loss defined in section II.D (Manifold learning), $\mathcal{A} = \{z_1^{(l)}, ..., z_N^{(l)}\}$ with $z_i^{(l)} = f^{(:l)}(x_i)$. As opposed to DiL model and SVM model, in neural network model, the manifold learning loss can be plugged at different locations in the architecture and a careful choice has to be made. Let consider the **Interpolation Consistency Training** (ICT) proposed in [Verma *et al.* 2019b]. Interestingly, its manifold learning loss differs from all traditional ones reviewed in section II.D. ICT is based on a supervised learning method called **Mix-up** [Zhang *et al.* 2017]. Mix-up learning objective is given by:

$$\begin{aligned}\mathcal{L}_{mu} =& \frac{1}{N_l^2 N_\lambda}\sum_{k=1}^{N_\lambda}\sum_{i=1}^{N_l}\sum_{j=1}^{N_l}\Big[d_s\big(f(\lambda x_i^l + (1-\lambda)x_j^l), \lambda y_i + (1-\lambda)y_j\big)\Big] \\ &\text{where: } \lambda \sim \text{Beta}(\alpha,\alpha), \alpha > 0. \\ &\qquad N_\lambda \text{ is the number of samplings } \lambda \text{ from Beta}(\alpha,\alpha).\end{aligned} \tag{IV.23}$$



(a)



(b)

Figure IV.7: (a): density function for Beta(0.5, 0.5). (b): Effect of Mix-up on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates $p(y = 1|x)$. Source: [Zhang *et al.* 2017].

The objective of Mix-up is to regularize the model by favoring a linear separation of classes. Figure IV.7 (b) shows the effect of Mix-up compared to empirical risk minimization (ERM), which is described by equation IV.6. The decision boundary between two classes is smoother compared to ERM's. In a semi-supervised setting, the ICT auxiliary is given by

$$\mathcal{L}_u = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_e(z_i, \mathcal{A}\backslash\{z_i\}) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} l_e(z_i, z_j) \tag{IV.24}$$

where:

$$\begin{aligned} l_e(z_i, z_j) =& \frac{1}{N_\lambda}\sum_{k=1}^{N_\lambda} d_{\text{MSE}}\Big(f_\theta\big(\lambda x_i + (1-\lambda)x_j\big), \lambda z_i + (1-\lambda)z_j\Big) \\ \text{where: } & \lambda \sim \text{Beta}(\alpha,\alpha), \alpha > 0. \\ & N_\lambda \text{ is the number of } \lambda \text{ samplings from Beta}(\alpha,\alpha). \end{aligned} \tag{IV.25}$$

l_e here can be considered as a manifold learning loss, which favors a linear behavior of the model. In addition, ICT uses Mean Teachers scheme to enforce the consistency. An illustration for ICT is shown in figure IV.8.

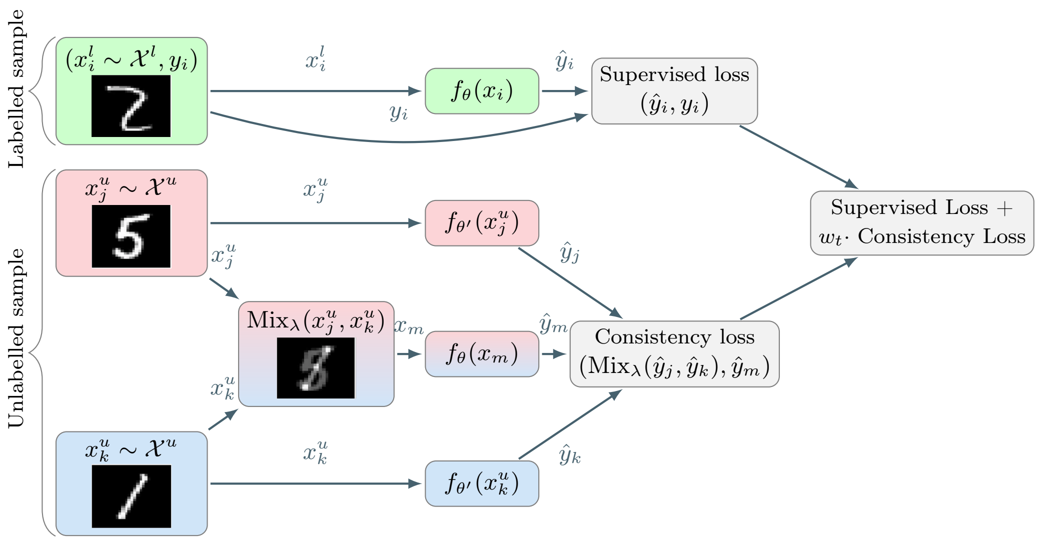
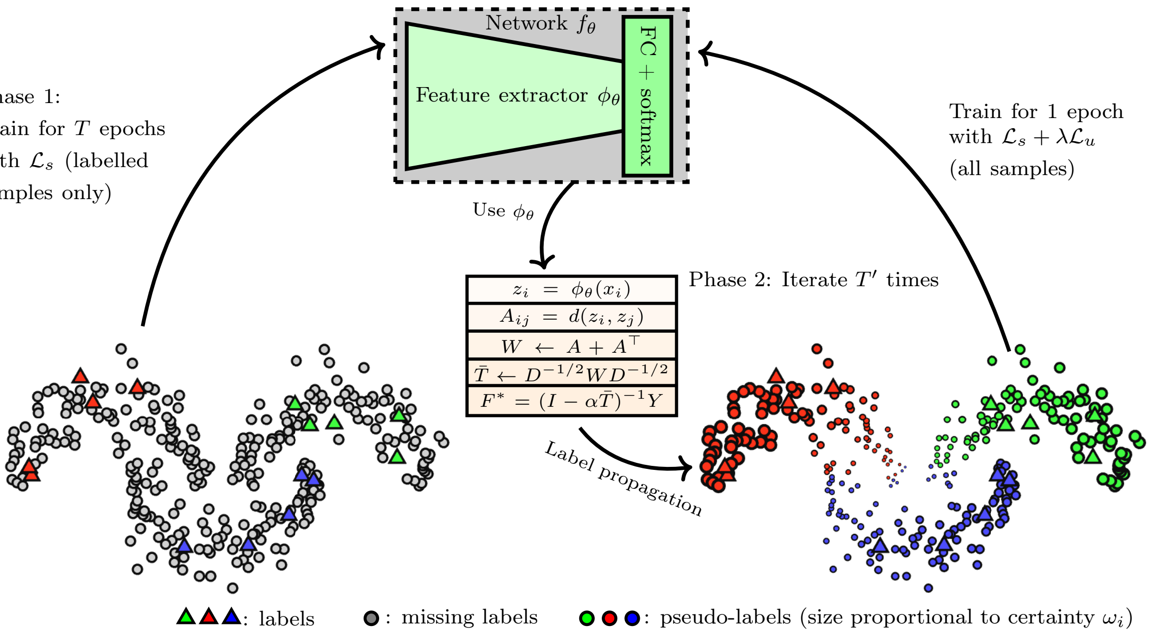

Labelled sample
$(x_i^l \sim \mathcal{X}^l, y_i)$
x_i^l
$f_\theta(x_i)$
$\hat{y}_i$
y_i
Supervised loss $(\hat{y}_i, y_i)$
Unlabelled sample
$x_j^u \sim \mathcal{X}^u$
x_j^u
$f_{\theta'}(x_j^u)$
$\hat{y}_j$
Supervised Loss + $w_t\cdot$ Consistency Loss
$\text{Mix}_\lambda(x_j^u, x_k^u)$
x_m
$f_\theta(x_m)$
$\hat{y}_m$
Consistency loss $(\text{Mix}_\lambda(\hat{y}_j, \hat{y}_k), \hat{y}_m)$
x_k^u
$x_k^u \sim \mathcal{X}^u$
$f_{\theta'}(x_k^u)$
$\hat{y}_k$

Figure IV.8: An illustration for Interpolation Consistency Training (ICT). Source: [Verma *et al.* 2019b].

A.3 Pseudo labeling. [Iscen *et al.* 2019] proposed to use pseudo-labelling to train a neural network in a semi-supervised manner. This approach relies on the smoothness assumption. Figure IV.9 illustrates this method, which can divided into two phases. In the first phase, the model f_θ is trained for T epochs by optimizing supervised cost $\mathcal{L}_s$ as in equation IV.6. Let ϕ_θ denote the part of the model going from the input layer to the layer just before the $fully-connected$ layer (FC). In a second phase, the following operations are performed iteratively:

- $z_i = \phi_\theta(x_i), \forall i = 1, .., N;$

- A where $A_{ij} = d(z_i, z_j)$, with d is a pairwise similarity metric;
- $W = A + A^\top$ if $d(z_i, z_j) \neq d(z_j, z_i)$.
- $\bar{T} = D^{-1/2} W D^{-1/2}$, with symmetric normalization on W (see effect of symmetric normalization in appendix A.8.1);
- Perform Label Propagation by taking equation $F^* = (I - \alpha\bar{T})^{-1} Y$;
- Train for one epoch with all samples, labelled samples and unlabelled samples with their pseudo labels $\bar{y}_i = \underset{k}{\operatorname{argmax}}\, F^*_u[i,k]$, where F^*_u is unlabelled part of F^*.



Network f_θ
Feature extractor ϕ_θ
FC + softmax
Phase 1:
Train for T epochs
with $\mathcal{L}_s$ (labelled
samples only)
Train for 1 epoch
with $\mathcal{L}_s + \lambda\mathcal{L}_u$
(all samples)
Use ϕ_θ
$z_i = \phi_\theta(x_i)$
$A_{ij} = d(z_i, z_j)$
$W \leftarrow A + A^\top$
$\bar{T} \leftarrow D^{-1/2} W D^{-1/2}$
$F^* = (I - \alpha\bar{T})^{-1} Y$
Phase 2: Iterate T' times
Label propagation
▲▲▲: labels
●: missing labels
●●●: pseudo-labels (size proportional to certainty ω_i)

Figure IV.9: An illustration for semi-supervised neural network with pseudo labels obtained by Label Propagation. Source: [Iscen *et al.* 2019].

Since Label Propagation can misclassify an unlabelled sample, the *pseudo label certainty* are used to control it. The certainty ω_i for an unlabelled sample x_i^u is measured by:

$$\omega_i = 1 - \frac{H(F^*_u[i,:])}{\log(C)} \tag{IV.26}$$

where H is the entropy function, C is the number of classes. In addition, a *class balancing* function ζ_k for class k is defined as $\zeta_k = \frac{1}{N_l^k + N_u^k}$, where N_l^k is the number of labelled samples in class k and N_u^k is the number of unlabelled samples whose pseudo labels are in class k. Here is the details of the complete cost $\mathcal{L}$ in the second phase:

$$\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u = \frac{1}{N_l}\sum_{i=1}^{N_l} \zeta_{y_i} d_s\big(f(x_i^l), y_i\big) + \frac{1}{N_u}\sum_{i=1}^{N_u} \zeta_{\bar{y}_i}\omega_i d_s\big(f(x_i^u), \bar{y}_i\big). \tag{IV.27}$$

Here $\lambda = 1$ would mean that the contribution of pseudo labels (after multiplying by ω_i) is as important as the one of real label. A combination with other approaches can be considered to improve the performance of this method, such as Mean Teacher by updating model parameters.

A.4 Generative models. A generative model is a model that can describe how sample (observation) x is generated from a latent representation z. We use the probabilistic notation $p(x|z)$. By sampling an instance from the distribution of z and passing it though the generative model, one can generate new data samples. Hence, the main points considered for creating a generative model are its architecture and the distribution that z is enforced to follow. In this subsection, we revisit two popular generative models, Variational AutoEncoder (VAE) [Kingma & Welling 2013] and Generative Adversarial Network (GAN) [Goodfellow *et al.* 2014a] with their applications for semi-supervised learning.

A.4.1 Variational Autoencoders

As indicated by its name, this generative model uses AE architecture. Given z the latent representation of a sample x via the decoder, each component of z is forced to follow the Normal distribution. Here is the cost optimized for VAE:

$$\mathcal{L}_{\text{VAE}} = \frac{1}{N}\sum_{x\in\mathcal{X}} \Big(d_r(x,\hat{x}) + d_{\text{KL}}\big(q(z|x), \mathcal{N}(\mathbb{0}, I)\big)\Big) \tag{IV.28}$$

where $\hat{x}$ is the reconstruction of x via AE, d_r is a dissimilarity metric used for reconstruction loss ($d_r = d_{BCE}$ by default) and $q(z|x)$ is a distribution of z given x which enforced to follow the Normal distribution $\mathcal{N}(\mathbb{0}, I)$.

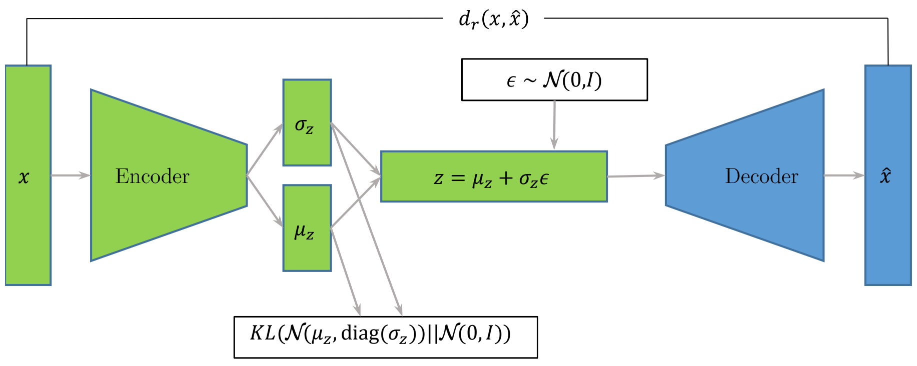

$d_r(x,\hat{x})$
$\epsilon \sim \mathcal{N}(0,I)$
x
Encoder
σ_z
μ_z
$z = \mu_z + \sigma_z\epsilon$
Decoder
$\hat{x}$
$KL(\mathcal{N}(\mu_z, \text{diag}(\sigma_z))||\mathcal{N}(0,I))$

Figure IV.10: Variational Autoencoder.

Figure IV.10 illustrates the VAE model. For the first step, the encoder is slightly modify to have two outputs μ_z and σ_z. Then $q(z|x)$ follows a Normal distribution if z is sampled from $\mathcal{N}\big(\mu_z, \text{diag}(\sigma_z)\big)$ in many epochs. For each training batch, ϵ is sampled from $\mathcal{N}(\mathbb{0}, I)$ in the forward pass and then fixed in the backward pass to calculate backward gradient.

In the second step, the KL divergence between $\mathcal{N}\big(\mu_z, \text{diag}(\sigma_z)\big)$ and $\mathcal{N}(\mathbb{0}, I)$ is calculated. Here is its explicit form:

$$d_{\text{KL}}\Big(\mathcal{N}\big(\mu_z, \text{diag}(\sigma_z)\big), \mathcal{N}(\mathbb{0}, I)\Big) = \sum_k \frac{1}{2}\left((\mu_z[k])^2 + (\sigma_z[k])^2 - 1 - \log(\sigma_z[k])^2\right) \tag{IV.29}$$

After training VAE model, for generating new data samples, we sample z from $\mathcal{N}(\mathbb{0}, I)$ then pass it via the decoder.

For semi-supervised learning by using VAE model, [Kingma *et al.* 2014] proposed two versions M1 and M2, which are detailed below.

Latent-feature discriminative model (M1). In this model, VAE is used as feature extractor and it consists of two steps. Firstly, the VAE model is trained with both labelled and unlabelled samples. Secondly, the samples are embedded into the latent representation space where a classifier is learnt in a supervised or semi-supervised manner.

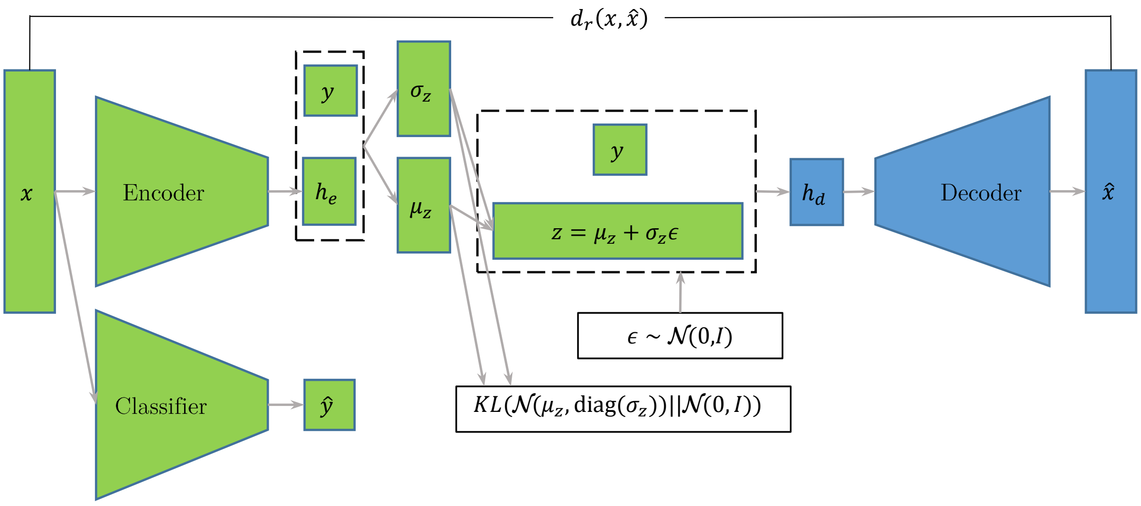

$d_r(x, \hat{x})$
x
Encoder
y
h_e
σ_z
μ_z
y
$z = \mu_z + \sigma_z \epsilon$
$\epsilon \sim \mathcal{N}(0, I)$
h_d
Decoder
$\hat{x}$
Classifier
$\hat{y}$
$KL(\mathcal{N}(\mu_z, \text{diag}(\sigma_z))||\mathcal{N}(0, I))$

Figure IV.11: An example of architecture for semi-supervised VAE, M2 model. Dashed line represents concatenation. Label information y here has intervened at the end of encoder and at the beginning of decoder, but it can be used elsewhere. Here $\hat{y} = q_\phi(y|x)$ is the output of classifier for a given sample x.

Generative semi-supervised model (M2). As opposed to M1 model, label information are used during the VAE training in this M2 model. In addition to the encoder and the decoder, there is a classifier (figure IV.11). Let ϕ denote parameters of encoder and classifier. This two models can eventually share some upstream layers. The classifier and the encoder are respectively represented by $q_\phi(y|x)$ and $q_\phi(z|x,y)$. For a labelled sample $x \in \mathcal{X}^l$ and its label y, the learning objective is given by:

$$\mathcal{L}_{\text{ssVAE}}(x,y) = d_r(x,\hat{x}) + d_{\text{KL}}\big(q_\phi(z|x,y), \mathcal{N}(\mathbb{0}, I)\big) + d_{\text{CE}}(P_y, y) \tag{IV.30}$$

where P_y is prior probability for y, usually $P_y = \left[\frac{1}{C}, .., \frac{1}{C}\right]$ if the same number of samples is used per classes for training. $\mathcal{L}_{\text{ssVAE}}$ is similar to equation IV.28's penalty, except for the fact that it uses labels information. For an unlabelled sample $x \in \mathcal{X}^u$ and its probability pseudo label $\hat{y} = q_\phi(y|x)$, the following auxiliary function is used:

$$\mathcal{U}_{\text{ssVAE}}(x) = \sum_y q_\phi(y|x)\mathcal{L}_{\text{ssVAE}}(x,y) + H(q_\phi(y|x)) = \sum_{k=1}^{C} \hat{y}[k]\mathcal{L}_{\text{ssVAE}}(x, y=k) + H(\hat{y}) \tag{IV.31}$$

Then the optimal configuration happens when classification loss is large and VAE loss is small or conversely, classification loss is small and VAE loss is large.

The second term $H(\hat{y})$ can be viewed as Entropy Minimization as mentioned in section IV.A.2.3. It is worth noting that $d_{\text{CE}}(P_y, y)$ is constant for labelled samples in $\mathcal{L}_{\text{ssVAE}}$ but it gives prior probability for unlabelled samples in $\mathcal{U}_{\text{ssVAE}}$. Integrating these cost functions with the supervised cost, the final objective function for semi-supervised VAE, M2 model is given by

$$\frac{1}{N_l}\sum_{i=1}^{N_l} d_{\text{CE}}\big(q_\phi(y|x_i^l), y_i\big) + \lambda\left(\frac{1}{N_l}\sum_{i=1}^{N_l}\mathcal{L}_{\text{ssVAE}}(x_i^l, y_i) + \frac{1}{N_u}\sum_{i=1}^{N_u}\mathcal{U}_{\text{ssVAE}}(x^u)\right) \tag{IV.32}$$

Stacked generative semi-supervised model (M1+M2). These two models can be combined, first using the generative model M1 to learn a new latent representation, then using this new latent representation as input for model M2.

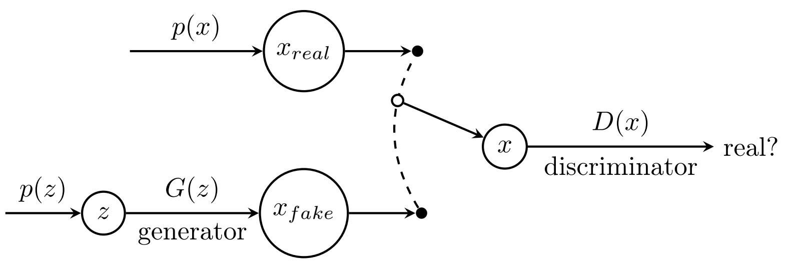


Figure IV.12: An illustration for generative adversarial networks. Source: [Veličković 2017].

A.4.2 Generative Adversarial Networks

This machine learning framework designed by [Goodfellow *et al.* 2014a] and consists of a discriminator D and a generator G. The generator G takes as input a latent representation z sampled from the prior distribution $p(z)$ and produces a synthetic sample x. On the other hand, real sample x is sampled from the real data distribution $p(x)$. With both fake and real samples x as input, the discriminator D tries to distinguish between fake and real. Figure IV.12 illustrates the architecture for generative adversarial networks (GAN) and $p(z)$ can be the Uniform distribution or the Normal distribution, depending on the specific task. The objective function for GAN model is described as:

$$\min_G \max_D \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[1 - \log D(G(z))] \tag{IV.33}$$

where the discriminator D outputs value between 1 and 0, which indicate respectively for real samples and for fake samples. In practice, the discriminator and the generator are optimized alternately:

$$\begin{aligned} \mathcal{L}_D &= \max_D \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[1 - \log D(G(z))] \\ \mathcal{L}_G &= \min_G -\mathbb{E}_{z\sim p(z)}[\log D(G(z))] \end{aligned} \tag{IV.34}$$

The generator and discriminator can be considered as two players that contest with each other in a zero-sum game.

Let turn to some works that use GAN for semi-supervised learning.

Categorical generative adversarial networks (CatGAN). This method proposed by [Springenberg 2016] is a variant of GAN, where the binary discriminator D (fake or real) is replaced by a K-classes discriminator. The objective function for CatGAN is constructed by maximizing or minimizing entropies. The five constraints on entropy used in CatGAN are described as follows:

(1) The output of the discriminator D, for a real sample, must have a low entropy.

(2) The output of the discriminator D, for a fake sample, must have a high entropy when optimizing D.

(3) The average of the outputs of the discriminator D, for all real samples, must have a high entropy.

(4) The output of the discriminator D, for a fake sample, must have a low entropy when optimizing G.

(5) The average of the outputs of the discriminator D, for all fake samples, must have a high entropy when optimizing G which translates the hypothesis that fake samples are uniformly sampled from their latent distribution.

The entropy of average of outputs in the constraint (*3*) and in the constraint (*5*) is represented respectively as:

$$\begin{aligned} H_{\mathcal{X}} &= H\left(\frac{1}{N}\sum_{i=1}^{N} D(x_i)\right) \\ H_G &= H\left(\frac{1}{M}\sum_{i=1}^{M} D(G(z_i))\right) \end{aligned} \tag{IV.35}$$

By taking constraints (*1*), (*2*) and (*3*) for the discriminator D and by taking constraints (*4*) and (*5*) for the generator G, we get the objective function of CatGAN:

$$\begin{aligned} \mathcal{L}_D &= \max_D -\mathbb{E}_{x\sim\mathcal{X}}[H(D(x))] + \mathbb{E}_{z\sim p(z)}[H(D(G(z)))] + H_{\mathcal{X}} \\ \mathcal{L}_G &= \min_G \mathbb{E}_{z\sim p(z)}[H(D(G(z)))] - H_G \end{aligned} \tag{IV.36}$$

It is worth noting that, until here, CatGAN is only unsupervised, and the number of components at the output K can be considered as a hyper-parameter. To incorporate labels information, one just have to set $K = C$, where C is the number of classes for labelled samples and then use the previous loss as an auxiliary loss in addition to a classification objective, which forms a semi-supervised learning method as follows:

$$\mathcal{L}_D + \frac{1}{\lambda}\frac{1}{N_l}\sum_{i=1}^{N_l} d_s\big(D(x_i^l), y_i\big) \tag{IV.37}$$

Deep convolutional generative adversarial networks (DCGAN) In this framework proposed by [Radford *et al.* 2016], one uses CNN architecture for both discriminator and generator. DCGAN is essentially used for classification task as a feature extraction.

A.5 Virtual Adversarial Training. In order to enforce outputs closeness for similar inputs, [Miyato *et al.* 2017] proposed a method called Virtual Adversarial Training (VAT), which has the following auxiliary loss:

$$\mathcal{L}_u = \frac{1}{N}\sum_{i=1}^{N} d_{\mathrm{KL}}\big(f(x_i), f(\tilde{x}_i)\big) \tag{IV.38}$$

where $\tilde{x}_i$ is an adversarial example related to x_i. $\tilde{x}_i$ can be generated using several methods mentioned in appendix B.2. The auxiliary loss $\mathcal{L}_u$ can be classified in the group of consistency constraints as mentioned in section IV.A.2.4. Using adversarial noise for generating $\tilde{x}_i$ helps to

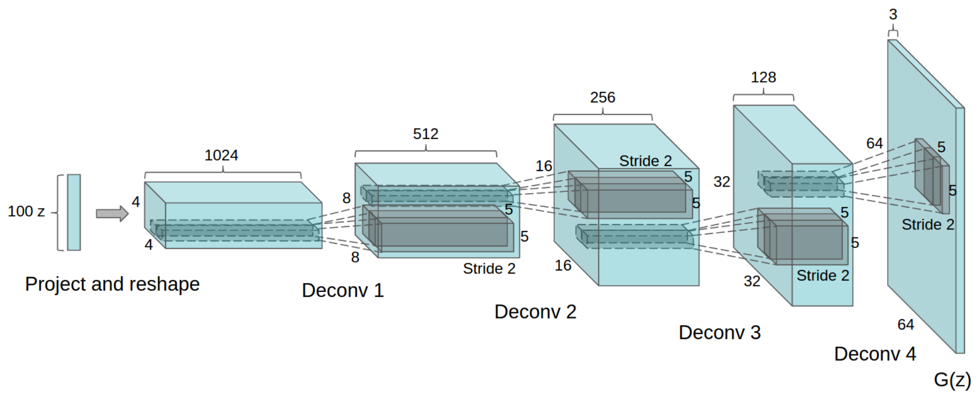


Figure IV.13: An illustration for generator used in DCGAN.

push efficiently decision boundaries far apart from a sample x_i.

In optimization process, one has on the one hand the attack stages, in which an adversarial examples is generated with a fixed model f. On the other hand, the model is updated using the adversarial examples by optimizing the total cost $\mathcal{L} = \mathcal{L}_s + \lambda\mathcal{L}_u$. The authors also found that adding Entropy Minimization loss to VAT loss (IV.38) can improve accuracy rate.

A.6 Holistic methods. A holistic method is an union of single methods and is expected to get better performances. We present for this subsection **MixMatch** [Berthelot *et al.* 2019b], which can be considered as a combination of ICT and Mix-up in section IV.A.2.7, through three main points:

1. **Data Augmentation**. A stochastic data augmentation is used for both labelled and unlabelled data. We note (x_i^l, y_i) where $i = 1, .., N_b$ a batch of labelled samples and (x_i^u) where $i = 1, .., N_b$ a batch of unlabelled samples. Each batch has N_b samples. Augment() denotes the stochastic data augmentation operator. For each labelled sample, the data augmentation is performed one time as $\tilde{x}_i^l = \text{Augment}(x_i^l)$. For each unlabelled sample, data augmentation is performed K times: $\tilde{x}_{i,k}^u = \text{Augment}(x_i^u)$ where $k = 1, .., K$.

2. **Label Guessing**. For each unlabelled sample, pseudo labels are computed and attributed to the associated K augmented data for each sample:

$$\bar{y}_{i,k} = \frac{1}{K}\sum_{j=1}^{K} f(\tilde{x}_{i,j}^u) \text{ with } k = 1, .., K \tag{IV.39}$$

Therefore, the batch of unlabelled samples now becomes $(\tilde{x}_{i,k}, y_{i,k})$ where there are $K \times N_b$ samples with their pseudo labels. The label guessing $\bar{y}_{i,k}$ is then sharpened as described in appendix A.8.2. Figure IV.14 shows an illustration for Data Augmentation followed by Label

Guessing.

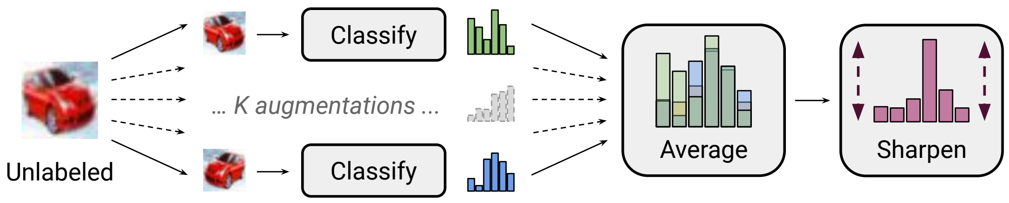


Figure IV.14: Data Augmentation followed by Label Guessing in MixMatch. Source: [Berthelot *et al.* 2019b].

3. **Mix-up**. Firstly, given two elements (x_a, y_a) and (x_b, y_b), the MixUp operator is defined as follows:

$$\begin{gathered}\text{MixUp}\big((x_a, y_a), (x_b, y_b)\big) = (\lambda x_a + (1-\lambda)x_b, \lambda y_a + (1-\lambda)y_b) \\ \text{where: } \lambda = \max(\lambda', 1-\lambda'). \\ \lambda' \sim \text{Beta}(\alpha, \alpha), \alpha > 0.\end{gathered} \tag{IV.40}$$

Let consider a set $\hat{\mathcal{X}}$ that contains labelled samples and their corresponding labels, a set $\hat{\mathcal{U}}$ that contains unlabelled samples (with K augmented samples) and their corresponding pseudo labels by Label Guessing, $\mathcal{W}$ a set that contains all samples after shuffling as follows:

$$\begin{aligned}\hat{\mathcal{X}} &= \{(\tilde{x}_i^l, y_i) | i = 1, .., N_b\} \\ \hat{\mathcal{U}} &= \{(\tilde{x}_i, y_i) | i = 1, .., N_b \times K\} \\ \mathcal{W} &= \text{Shuffle}\big(\text{Concat}(\hat{\mathcal{X}}, \hat{\mathcal{U}})\big)\end{aligned} \tag{IV.41}$$

Then the new labelled set $\mathcal{X}'$ and unlabelled set $\mathcal{U}'$ are defined as follows:

$$\begin{aligned}\mathcal{X}' &= \{\text{MixUp}(\hat{\mathcal{X}}_i, \mathcal{W}_i) | i = 1, .., |\hat{\mathcal{X}}|\} \\ \mathcal{U}' &= \{\text{MixUp}(\hat{\mathcal{U}}_i, \mathcal{W}_{|\hat{\mathcal{X}}|+i}) || i = 1, .., |\hat{\mathcal{U}}|\}\end{aligned} \tag{IV.42}$$

The complete MixMatch cost is given by:

$$\mathcal{L} = \mathcal{L}_s + \lambda \mathcal{L}_u = \sum_{x,y \in \mathcal{X}'} d_s(f(x), y) + \lambda \sum_{x,y \in \mathcal{U}'} d_{\text{MSE}}(f(x), y) \tag{IV.43}$$

Holistic methods give a high performance for classification task and are the state of art on several dataset such as CIFAR-10 or SVHN. Aside from MixMatch, we have also other holistic methods such as Fix-Match [Sohn *et al.* 2020], Real-Mix [Nair *et al.* 2019], Ensemble Auto-Encoding Transformations (EnAET) and [Wang *et al.* 2019].

A.7 Partial conclusion for semi-supervised neural networks. To construct a SSNN model, one combines one or more auxiliary losses with a standard classification objective. Various losses can be used including reconstruction, consistency constraint, classification objective for self-annotated samples, manifold preservation constraint, classification objective on pseudo-labels... Generative models can also be adapted for semi-supervised learning. Holistic methods are the obvious strategy to think about, in order to get a better performance than using each single method. Nevertheless, it is required combined methods to be compatible with one another. Otherwise, the performance might drop instead of improving. A survey of semi-supervised neural network methods can be found in [Ouali *et al.* 2020].

B Manifold attack

In this section, we introduce a new semi-supervised neural network learning method that uses an auxiliary loss based on adversarial learning and manifold learning. Hence the name "manifold attack". We have seen previously that an optimized perturbation of small amplitude can change significantly a neural network model output, leading to a misclassification (figure B.1 in appendix B.2). This can be explained from a manifold structure point of view by the fact that the model does not necessarily compute a locally isometric embedding of the original dataset. Hence, close by point in the original data space can yield far apart representations in the model derived latent space. Manifold attack introduced a new way of dealing with this problem while smoothing the decision boundaries. In the following $g()$ denotes a function that maps a sample $\mathbf{x} \in \mathbb{R}^n$ to its embedded representation $\mathbf{a} \in \mathbb{R}^p$.

B.1 Individual attack point versus data points. We define a *virtual point* is a synthetic sample generated in such a way to be likely on the observed samples underlying manifold. For example, a virtual point can be produced by a generative model or obtained as the result of a small shift of an observed sample. An *anchor point* is a sample used for generating a *virtual point.* An *attack point* is a virtual point that maximises locally a chosen measure of model distortion. For example, attack point can be a sample perturbed with an adversarial noise.

We use the same notation as in section II.D (Manifold learning). Given a dataset $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and the corresponding embedded set $\mathcal{A} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}$, $\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c)$ denotes the embedding loss, $\mathcal{A}_i^c$ being the complement of $\{\mathbf{a}_i\}$ in $\mathcal{A}$. We consider the manifold learning objective function defined as $\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c)$. Let's consider p anchor points $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_p \in \mathbb{R}^n$, a virtual point $\tilde{\mathbf{x}} \in \mathbb{R}^n$ is defined as:

$$\begin{aligned} \tilde{\mathbf{x}} &= \gamma_1 \mathbf{z}_1 + \gamma_2 \mathbf{z}_2 + ... + \gamma_p \mathbf{z}_p, \\ \text{subject to: } & \gamma_1, \gamma_2, .., \gamma_p \geq 0, \\ & \gamma_1 + \gamma_2 + ... + \gamma_p = 1. \end{aligned} \tag{IV.44}$$

The anchor points $\mathbf{z}_i$ define a region or feasible zone, in which a virtual point $\tilde{\mathbf{x}}$ must be located and $\gamma = [\gamma_1, \gamma_2, .., \gamma_p]$ is the coordinate of $\tilde{\mathbf{x}}$. In general, anchor points are sampled from the dataset $\mathcal{X}$ with different strategies, which are defined according to a user provided rule (see section IV.B.4 for several examples). Figure IV.15 shows an example for setting of anchor points and relations between points. For a given embedding $\mathbf{a}_i = g(\mathbf{x}_i)$, the embedding loss is defined as

$$\mathcal{L}_e(\mathbf{a}_i, \mathcal{A}_i^c) = \mathcal{L}_e\big(g(\mathbf{x}_i), \{g(\mathbf{x}_1), .., g(\mathbf{x}_N)\} \backslash \{g(\mathbf{x}_i)\}\big). \tag{IV.45}$$

Similarly, the embedding loss is defined for a virtual point $\tilde{\mathbf{a}} = g(\tilde{\mathbf{x}})$ as

$$\mathcal{L}_e(\tilde{\mathbf{a}}, \mathcal{A}) = \mathcal{L}_e\big(g(\tilde{\mathbf{x}}), \{g(\mathbf{x}_1), .., g(\mathbf{x}_N)\}\big). \tag{IV.46}$$

The algorithm 2 describes the computation of an attack point. It consists in finding the local coordinates γ that maximizes the embedding loss $\mathcal{L}_e(\tilde{\mathbf{a}}, \mathcal{A})$ for the current model $g()$ state. Hence, γ is estimated though a projected gradient ascent.

Algorithm 2 Individual manifold attack

```
Require: Anchor points $\{\mathbf{z}_1, .., \mathbf{z}_p\}$, data points $\{\mathbf{x}_1, .., \mathbf{x}_N\}$, embedding loss $\mathcal{L}_e()$, model $g()$,
  $\xi, n\_iters$.
  initialize: $\gamma \in \mathbb{R}^p, \gamma = [\gamma_1, .., \gamma_p]$ for constraints in eq. (IV.44)
  $\tilde{\mathbf{x}} = \gamma_1 \mathbf{z}_1 + \gamma_2 \mathbf{z}_2 + ... + \gamma_p \mathbf{z}_p$
  for $i = 1$ to $n\_iters$ do
    $L = \mathcal{L}_e\big(g(\tilde{\mathbf{x}}), \{g(\mathbf{x}_1), .., g(\mathbf{x}_N)\}\big)$
    $\gamma \leftarrow \gamma + \xi \nabla_\gamma L(\tilde{\mathbf{x}})$
    $\gamma \leftarrow \Pi_{ps}(\gamma)$
    $\tilde{\mathbf{x}} = \gamma_1 \mathbf{z}_1 + \gamma_2 \mathbf{z}_2 + ... + \gamma_p \mathbf{z}_p$
  end for
  Output: $\tilde{\mathbf{x}}$
```

In order to guarantee the constrains in IV.44, we use the projector Π_{ps} defined by the problem

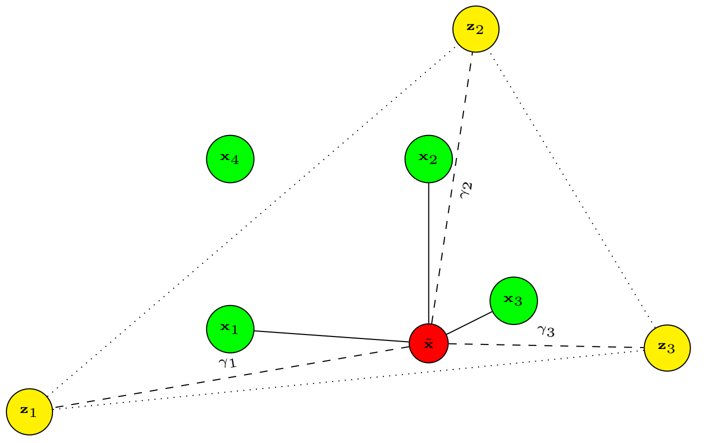


Figure IV.15: Virtual point and anchor points illustration. The three anchor points ($i \in \{1,2,3\}$) are computed as $\mathbf{z}_i = \mu([\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3]) + s(\mathbf{x}_i - \mu([\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3])$, where $\mu([\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3]) = \frac{\mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3}{3}$. The dotted lines represent the zone defined by anchor points $\mathbf{z}_i$ within which the virtual point $\tilde{\mathbf{x}}$ necessarily lies. The parameter s determines whether the anchor points lie strictly within $[\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3]$ convex hull ($1 > s > 0$) or are strictly outside ($s > 1$). The dashed lines represent the coordinates γ of $\tilde{\mathbf{x}}$. The solid lines illustrates the relatedness of $\tilde{\mathbf{x}}$ to the data points that the embedding will try to preserve minimizing $\mathcal{L}_e(g(\tilde{\mathbf{x}}), \{g(\mathbf{x}_1), g(\mathbf{x}_2), g(\mathbf{x}_3)\})$ in this case.

$$\begin{aligned} &\min_{\gamma\in\mathbb{R}^p} \frac{1}{2}\left\|\kappa-\gamma\right\|_2^2, \\ \text{subject to: } &\gamma_1,\gamma_2,..,\gamma_p \geq 0, \\ &\gamma_1+\gamma_2+...+\gamma_p = c, (c>0). \end{aligned} \tag{IV.47}$$

This convex problem with constraints can be solved quickly by a simple sequential projection that alternates between sum constraint and positive constraint (algorithm 3). The demonstration can be inspired by Lagrange multiplier method. A simple demonstration can be found in appendix B.1.

Algorithm 3 Projection for positive and sum constraint Π_{ps}

```
Require: $\kappa \in \mathbb{R}^p$, $c = 1$ (by default).
  $\delta = (c - \sum_{i=1}^{p} \kappa_i)/p$
  $\gamma_i \leftarrow \gamma_i + \delta, \forall i = 1,..,p$
  while $\exists i \in \{1,..,p\} : \gamma_i < 0$ do
    $\mathcal{P} = \{i|\gamma_i > 0\}$ and $\mathcal{N} = \{i|\gamma_i < 0\}$
    $\gamma_i \leftarrow 0, \forall i \in \mathcal{N}$
    $\delta = (c - \sum_{i\in\mathcal{P}} \gamma_i)/|\mathcal{P}|$
    $\gamma_i \leftarrow \gamma_i + \delta, \forall i \in \mathcal{P}$
  end while
  Output: $\gamma = [\gamma_1,\gamma_2,..,\gamma_p]$
```

By default, each set of anchor points has one attack point. However, we can generate more than one attack point for the same set of anchor points by using different initializations of γ, so as to find different local maxima. The double embedding constraint, on the observed samples on the one hand and on the attack points on the other hand is expected to enforce the the model $g()$ smoothness over the underlying manifold, including in low samples density areas. The general optimization scheme goes as follows: we optimize alternately between attack stages and model update stages until convergence. In attack stage, we optimize the attack points through γ while fixing the model $g()$ and in the model update stage, we optimize the model $g()$ while fixing attack points.

B.2 Attack points as data augmentation. In algorithm 2, an attack points only interacts with observed samples. In the general manifold attack (algorithm 4), attack points and observed samples are undifferentiated in the model update stage. This way, hence generating attack points can be considered a data augmentation technique. We denote $\mathcal{B}$ as a set that contains all embedded points (both attack points and observed samples). $\mathcal{B}_s$ is a random subset of $\mathcal{B}$, used batch for batch optimization. In each step, only attack points from the current batch are used to distort the manifold by maximizing the batch loss L.

Algorithm 4 Manifold attack

```
Require: Data points $\{\mathbf{x}_1, .., \mathbf{x}_N\}$, embedding loss $\mathcal{L}_e()$, model $g()$, $\xi$, $n\_iters$, an anchoring rule.
  initialize: $g()$
  Create $M$ sets of anchor points $\{\mathbf{z}_1^k, .., \mathbf{z}_p^k\}, \forall k = 1, .., M$ by the anchoring rule
  for $epoch = 1$ to $n\_epoch$ do
    Initialize $\gamma^k \in \mathbb{R}^p$ for constraints in eq. (IV.44)
    $\tilde{\mathbf{x}}^k = \gamma_1^k \mathbf{z}_1^k + \gamma_2^k \mathbf{z}_2^k + ... + \gamma_p^k \mathbf{z}_p^k, \forall k = 1, .., M$
    Set $\mathcal{B} = \{g(\tilde{\mathbf{x}}^1), .., g(\tilde{\mathbf{x}}^M)\} \cap \{g(\mathbf{x}_1), .., g(\mathbf{x}_N)\}$ and divide it into subsets $\mathcal{B}_s$
    for each $\mathcal{B}_s$ do
      $L = \sum_{\mathbf{a} \in \mathcal{B}_s} \mathcal{L}_e(\mathbf{a}, \mathcal{B}_s \backslash \{\mathbf{a}\})$
      Update $\{\tilde{\mathbf{x}}^i | g(\tilde{\mathbf{x}}^i) \in \mathcal{B}_s\}$ to maximize $L$ by algorithm 5
      Update $g()$ to minimize $L$
    end for
  end for
  Output: $g()$
```

Algorithm 5 represents the update step for multiple attack points. We assumed that the embedding loss $\mathcal{L}_e$ is smooth with respect to γ and used a gradient-based algorithm to estimate the latter. Nevertheless, in manifold learning field, there are several methods whose embedding loss is not even continuous. For example, in LLE (II.37), the embedding loss takes into account the k nearest neighbors of a point, which might change throughout the estimation of an attack point, producing a discontinuity. To circumvent this problem, we use several strategies to avoid singularities:

Algorithm 5 Virtual points update

```
Require: $m$ sets of anchor points $\{\mathbf{z}_1^k, .., \mathbf{z}_p^k\}$, $\gamma^k$, $\forall k = 1, .., m$, loss $L$, $\xi, n\_iters$.
  $\tilde{\mathbf{x}}^k = \gamma_1^k \mathbf{z}_1^k + \gamma_2^k \mathbf{z}_2^k + ... + \gamma_p^k \mathbf{z}_p^k, \forall k = 1, .., m$
  for $i = 1$ to $n\_iters$ do
    Calculate gradient $\nabla L$ (w.r.t $[\gamma^1, .., \gamma^m]$) of function $L(\tilde{\mathbf{x}}^1, .., \tilde{\mathbf{x}}^m)$
    $[\gamma^1, .., \gamma^m] \leftarrow [\gamma^1, .., \gamma^m] + \xi \nabla_\gamma L(\tilde{\mathbf{x}}^1, .., \tilde{\mathbf{x}}^m)$
    $\gamma^k \leftarrow \Pi_{ps}(\gamma^k), \forall k = 1, .., m$
    $\tilde{\mathbf{x}}^k = \gamma_1^k \mathbf{z}_1^k + \gamma_2^k \mathbf{z}_2^k + ... + \gamma_p^k \mathbf{z}_p^k, \forall k = 1, .., m$
  end for
  Output: $\tilde{\mathbf{x}}^1, .., \tilde{\mathbf{x}}^m$
```

- By reducing the gradient step ξ which limits the virtual point displacement.
- By taking a small number of attack points in each subset $\mathcal{B}_s$ or by using randomly a part of attack points to perform the attack while fixing other attack points, in an attack stage.
- By updating γ only if embedding loss increases.

Besides, some metrics might be approximated by smooth functionals. For instance, in the contrastive loss (II.D), we can replace the metric $d_x()$ which outputs only 0 or 1, with $d_x(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma_i^2}\right)$ to make embedding loss continuous.

B.3 Pairwise manifold learning. For some manifold learning methods as MDS or LE, the embedding loss $\mathcal{L}_e$ can be decomposed into the sum of elementary pairwise loss l_e:

$$\mathcal{L}_e(\mathbf{a}, \mathcal{B}) = \sum_{\mathbf{b} \in \mathcal{B}} l_e(\mathbf{a}, \mathbf{b}) \tag{IV.48}$$

Then the batch loss L (in algorithm 4) can be modified into:

$$L = \sum_{\mathbf{a} \in \mathcal{B}_s} \mathcal{L}_e\big(\mathbf{a}, \mathcal{B}_s \backslash \{\mathbf{a}\}\big) = \sum_{\mathbf{a} \in \mathcal{B}_s} \sum_{\mathbf{b} \in \mathcal{B}_s, \mathbf{b} \neq \mathbf{a}} l_e(\mathbf{a}, \mathbf{b}) \tag{IV.49}$$

Following this change, L can be decomposed into three parts:

$$\sum_{\mathbf{a} \in \mathcal{B}_s^d} \sum_{\mathbf{b} \in \mathcal{B}_s^d, \mathbf{b} \neq \mathbf{a}} l_e(\mathbf{a}, \mathbf{b}) \qquad \text{(data-data)}$$

$$\sum_{\mathbf{a} \in \mathcal{B}_s^d} \sum_{\mathbf{b} \in \mathcal{B}_s^v} l_e(\mathbf{a}, \mathbf{b}) \qquad \text{(data-virtual)}$$

$$\sum_{\mathbf{a} \in \mathcal{B}_s^v} \sum_{\mathbf{b} \in \mathcal{B}_s^v, \mathbf{b} \neq \mathbf{a}} l_e(\mathbf{a}, \mathbf{b}) \qquad \text{(virtual-virtual)},$$

where $\mathcal{B}_s^d$ and $\mathcal{B}_s^v$ are respectively set that contains all embedded data points (or observed samples) and all embedded virtual points of $\mathcal{B}_s$.

In some manifold learning losses which can be decomposed into the sum of elementary pairwise loss, we can balance the effect of the virtual points with respect to the observed samples, not only by tuning the ratio between the number of virtual points and the number of observed samples in B_s, but also by weighting each of three parts above which corresponds to the settings observed-observed, observed-virtual and virtual-virtual.

B.4 Settings of anchor points and initialization of virtual points. In this section, we provide two settings (or rules) for computing anchor points with the corresponding initializations. These settings need to be chosen carefully to guarantee that virtual points are on the sample underlying manifold.

Neighbor anchors: The first anchor point $\mathbf{z}_1$ is taken randomly from $\mathcal{X}$, then the next $(p-1)$ anchor points $\mathbf{z}_2, .., \mathbf{z}_p$ are taken as $(p-1)$ nearest neighbor points of $\mathbf{z}_1$ in $\mathcal{X}$ (Euclidean metric by default). Here, we assume that the convex hull of a sample and its neighbors is likely comprised in the samples manifold. The number of anchors p needs to be small compared to the number of data points N. The initialization for virtual points can be set by taking $\gamma_i \sim \mathcal{U}(0,1), \forall i = 1, .., p$ then normalize to have $\sum_{i=1}^{p} \gamma_i = 1$.

Random anchors: The second setting is inspired by Mix-up method [Zhang *et al.* 2017]. p anchors are taken randomly from $\mathcal{X}$ and we take $\gamma \sim \text{Dirichlet}(\alpha_1, .., \alpha_p)$. If $\alpha_i \ll 1, \forall i = 1, .., p$, the Dirichlet distribution returns γ where $\gamma_i \geq 0$, $\sum_{i=1}^{p} \gamma_i = 1$. In particular, there is a coefficient γ_k much greater than other ones with a strong probability, which implies that virtual points are more probably in the neighborhood of a data sample. Since the manifold attack tries to find only local maximum by gradient-based method, if ξ and n_iters are both small, we expect that attack points in the attack stage do not move too far from their initiated position, remaining on the manifold of data. Note that, in the case $\alpha_i = 1, \forall i = 1, .., p$, the Dirichlet distribution become the Uniform distribution.

To ensure that the coefficient γ_k is always much greater than other ones, we apply one more constraint: $\gamma_k \geq \tau$, and by taking τ close to 1. The constraints in IV.44 become:

$$\begin{aligned} &\gamma_1, \gamma_2, .., \gamma_p \geq 0, \\ &\gamma_1 + \gamma_2 + ... + \gamma_p = 1, \\ &\gamma_k \geq \tau, (\tau < 1). \end{aligned} \tag{IV.50}$$

Then the projection in algorithm 3 needs to be slightly modified to incorporate this new constraint. We define the projection $\gamma = \Pi'_{ps}(\kappa)$ as follow:

$$\begin{aligned}
&\kappa' \leftarrow \kappa \\
&\kappa'_k \leftarrow \kappa'_k - \tau \\
&\gamma \leftarrow \Pi_{ps}(\kappa', c = 1 - \tau) \\
&\gamma_k \leftarrow \gamma_k + \tau
\end{aligned} \tag{IV.51}$$

C Applications of manifold attack

We present several applications of manifold attack for NNMs that uses manifold structure constraints. Firstly, we show advantages of manifold attack for a manifold learning task when few training samples are available. Secondly, we show that manifold attack is compatible with several neural network learning methods, which it improves the accuracy rate and the robustness to adversarial examples.

C.1 Manifold learning on a small dataset. For this experiment, we use the S curve data and Digit data. The S curve data contains $N = 1000$ 3-dimensional samples, as shown in figure IV.16. The Digit data contains $N = 1797$ images, of size 8×8 of a digit. We want to compute 2-dimensional embeddings for these data. Each data is separated into two sets: N_{tr} samples are randomly taken for training set and the N_{te} remaining samples are used for testing. $\mathbf{x}^{tr}$ denotes a training sample and $\mathbf{x}^{te}$ a testing sample. We perform four training modes as described in table IV.1 with a neural network model $g()$. The evaluation loss, after optimizing model $g()$, is defined as:

$$L_{ev} = \frac{1}{N_{te}} \sum_{i=1}^{N_{te}} \mathcal{L}_e(g(\mathbf{x}_i^{te}), \{g(\mathbf{x}_j^{te}) | j \neq i\}) \tag{IV.52}$$

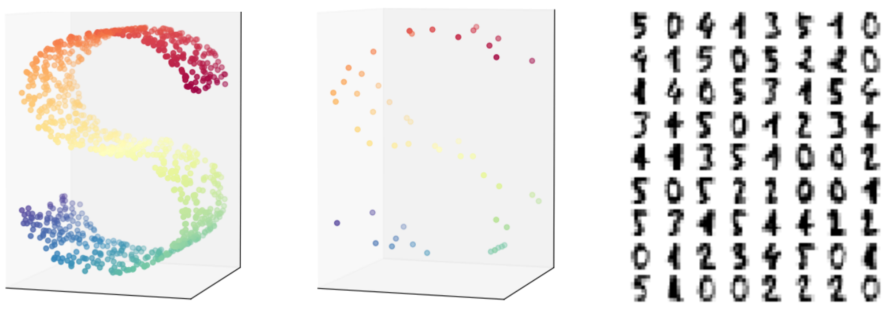


Figure IV.16: Left: S curve data 1000 samples. Center: S curve data 50 samples. Right: Digit data.

The anchoring rule, embedding loss $\mathcal{L}_e$ and the model $g()$ are precised in the following.

Mode	Description of objective function
1. REF	Manifold learning that takes into account both training and testing samples: $$L_{tr} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_e(g(\mathbf{x}_i), \{g(\mathbf{x}_j) \mid j \neq i\})$$ The result of this training mode is considered as "reference" in order to compare to other training modes.
2. DD	Manifold learning that takes only training data samples: $$L_{tr} = \frac{1}{N_{tr}}\sum_{i=1}^{N_{tr}} \mathcal{L}_e(g(\mathbf{x}_i^{tr}), \{g(\mathbf{x}_j^{tr}) \mid j \neq i\}).$$
3. RV	Using virtual points as supplement data, virtual points are only randomly initialized and without attack stage (by setting $n\_iters = 0$ in algorithm 5): $$L_{tr} = \frac{1}{\lvert\mathcal{B}\rvert}\sum_{\mathbf{a}_i \in \mathcal{B}} \mathcal{L}_e(\mathbf{a}_i, \mathcal{B}\backslash\{\mathbf{a}_i\}),$$ where $\mathcal{B} = \{g(\tilde{\mathbf{x}}^1), .., g(\tilde{\mathbf{x}}^M)\} \cap \{g(\mathbf{x}_1^{tr}), .., g(\mathbf{x}_{N_{tr}}^{tr})\}$.
4. MA	Using manifold attack, the same objective function as the previous case, except $n\_iters \neq 0$ (virtual points become attack points).

Table IV.1: Four training modes: REF (Reference), DD (Data-Data), RV (Random Virtual) and MA (Manifold Attack), and their corresponding objective function.

Anchoring rule. Two settings are considered:

- *Neighbor anchors* (NA) : A set of anchor points is composed by a sample with its 4 nearest neighbors. In this case, we have $M = N_{tr}$ sets of anchor points and $p = 5$ anchor points in each set. The coefficient γ is initialized by the Uniform distribution.

- *Random anchors* (RA) : We take randomly 2 points among N_{tr} training points to create a set of anchor points. In this case, we have $M = \binom{N_{tr}}{2}$ sets of anchor points and $p = 2$ anchor points in each set. The coefficient γ is initialized by the Dirichlet distribution with $\alpha_i = 0.5, \forall i = 1, .., p$.

The embedding loss. We use the loss from MDS and LE methods as described in section II.D, with the default metrics. For similarity metric $d_x()$ in LE method, we take $\sigma = 0.2$ for S curve data and $\sigma = 0.5$ for Digit data.

The model. A simple structure of CNN is used. Here are the detailed architectures for each CNN as the dimension of the inputs are different for the two datasets:

- S curve data: $\textit{Conv1d}[1,4,2] \rightarrow \textit{ReLu} \rightarrow \textit{Conv1d}[4,4,2] \rightarrow \textit{ReLu} \rightarrow \textit{Flatten} \rightarrow \textit{Fc}[4,2]$.

- Digit data: *Conv2d*[1, 8, 3] → *ReLu* → *Conv2d*[8, 16, 3] → *ReLu*→ *Flatten* → *Fc*[64, 2].

For LE method, two additional constraints are imposed to avoid trivial embeddings:

$$\begin{aligned} \mathrm{E}(\mathbf{A}^{tr}) = [\mathrm{E}(\mathbf{A}^{tr}[1,:]), .., \mathrm{E}(\mathbf{A}^{tr}[d,:])]^\top = \mathbb{0}_d \\ \Sigma(\mathbf{A}^{tr}, \mathbf{A}^{tr}) = I_d \end{aligned} \tag{IV.53}$$

where $d = 2$ is the number of output dimensions, $\mathbf{A}^{tr} = [\mathbf{a}_1^{tr}, .., \mathbf{a}_{N_{tr}}^{tr}] = [g(\mathbf{x}_1^{tr}), .., g(\mathbf{x}_{N_{tr}}^{tr})]$. To adapt these constraints, we add a normalization layer at the end of model $g()$: $(g(\mathbf{x}) - \mathrm{E}(\mathbf{A}^{tr}))\Sigma^{-1}(\mathbf{A}^{tr}, \mathbf{A}^{tr})$, where Σ^{-1} is performed by Cholesky decomposition.

To simulate the case of few training samples, we fix $N_{tr} = 100$ for MDS method and $N_{tr} = 50$ for LE method. The balance between virtual points and samples is controlled by the couple $\lambda =$ (number of virtual points in $\mathcal{B}_s$, number of samples in $\mathcal{B}_s$). We set $\lambda = (2, 5)$ for MDS method and $\lambda = (5, 10)$ for LE method. The gradient step ξ is selected from $\{0.1, 1, 10\}$ and the number of iterations is fixed at $n\_iters = 2$.

The initialization of model $g()$ is impactful, especially since there are few training data. Five different initialization of model for each method are performed. The mean and the standard deviation of the evaluation loss L_{ev} are represented in table IV.2. Firstly, we see that using random virtual (RV) points as additional data points gives a better loss than using only data points. Secondly, using manifold attack (MA) further improves the results which shows the benefit of the proposed approach to regularize the model.

For the S curve data, initialization by *Neighbors anchors* (NA) gives a better result compared to initialization by *Random anchors* (RA). However, for the Digit data, initialization by *Random anchors* gives a better result. This is due to the fact that in the S curve data, *Neighbor Anchors* covers better the manifold of data than *Random Anchor*. On the other hand, in Digit data, *Neighbor Anchors* (by using Euclidean metric to determine nearest neighbors) can generate, with greater probability, a virtual point that is not in the manifold of data. This leads to a greater evaluation loss compared to *Random Anchor*.

The five embedded representations, respectively with five different initialization of $g()$, for testing samples in S curve data are found in figure IV.17 for MDS method and in figure IV.18 for LE method.

C.2 Robustness to adversarial examples. In this subsection, we integrate manifold attack to Mix-up [Zhang *et al.* 2017], a supervised learning approach and study the robustness to adversarial

	S curve data		Digit data	
Mode / Method	MDS	LE	MDS	LE
REF	130.7 ± 24.74	0.399 ± 0.07	2015 ± 14	0.07 ± 0.002
DD	352.56 ± 119.19	1.21 ± 0.46	2409 ± 78	0.58 ± 0.07
RV (NA)	173.87 ± 9.38	0.59 ± 0.11	2395 ± 73	0.31 ± 0.03
MA (NA)	170.62 ± 5.89	0.55 ± 0.07	2362 ± 63	0.24 ± 0.03
RV (RA)	183.42 ± 18.13	0.65 ± 0.14	2342 ± 56	0.22 ± 0.03
MA (RA)	169.04 ± 5.30	0.63 ± 0.14	2331 ± 56	0.2 ± 0.02

Table IV.2: Evaluation loss L_{ev} of two manifold learning methods MDS and LE, in four modes: REF, DD, RV, MA (as described in table IV.1) and two initialization strategies: Neighbor Anchors (NA) and Random Anchors (RA).

examples. We remind the objective function of Mix-up:

$$\begin{aligned} \mathcal{L}_{mu} &= \frac{1}{N_l^2 N_\lambda} \sum_{k=1}^{N_\lambda} \sum_{i=1}^{N_l} \sum_{j=1}^{N_l} \left[d_s(\gamma_1 \mathbf{x}_i + \gamma_2 \mathbf{x}_j, \gamma_1 y_i + \gamma_2 y_j) \right] \\ &\text{where: } \gamma_1 + \gamma_2 = 1 \text{ and } \gamma_1 \sim \text{Beta}(\alpha, \alpha), \alpha > 0 \\ &N_\lambda \text{ is the number of samplings } \lambda \text{ from Beta}(\alpha, \alpha). \end{aligned} \tag{IV.54}$$

where $\mathbf{x}_i, \mathbf{x}_j$ are labelled samples and y_i, y_j are their corresponding labels. To apply manifold attack to Mix-up, we consider a set of anchor points consist of $\mathbf{x}_i, \mathbf{x}_j$ and an attack point is coordinated by $\gamma = [\gamma_1, \gamma_2]$. Then we add an attack stage to find γ that gives the greater loss $\mathcal{L}_{mu}$, before performing model update stage for $g()$. We repeat alternatively these two stages until convergence. The attack stage is performed by using algorithm 5 with loss L, here is a batch loss of $\mathcal{L}_{mu}$. As in Mix-up, we can take $\gamma 2 = 1 - \gamma_1$, so that we only need to deal with one variable γ_1 to maximize the batch loss L. The projection IV.47 for γ_1 is now just the clamping function, to make sure that γ_1 is between 0 and 1.

We compare four supervised training methods: ERM (Empirical Risk Minimization) which is thus traditional supervised learning as equation IV.6, Mix-up, Mix-up attack and Cut-Mix [Sangdoo *et al.* 2019] on ImageNet dataset with the model ResNet-50 [He *et al.* 2015] (see section II.C.3), which has about 25.8M trainable parameters. We use ImageNet dataset. We retrieve 948 classes consisting in 400 labelled training samples and 50 testing samples to evaluate the models.

We evaluate the error rate for testing set at the end of each epoch and report the best best error rate (top-1 and top-5) in table IV.3. For adversarial examples, we create them by Fast Gradient Sign Method (FGSM) [Goodfellow *et al.* 2014b] (see appendix B.2), on the trained model by ERM, with $\epsilon = 0.05$. In Mix-up Attack, n_iters is fixed at 1 and ξ is set up following two configurations, (1) ξ is reduced linearly from 0.1 to 0.01 and (2) ξ is fixed at 0.01. Following the original article, α is set at 0.2 for Mix-Up and Mix-up Attack and $\alpha = 1$ for Cut-Mix. More details

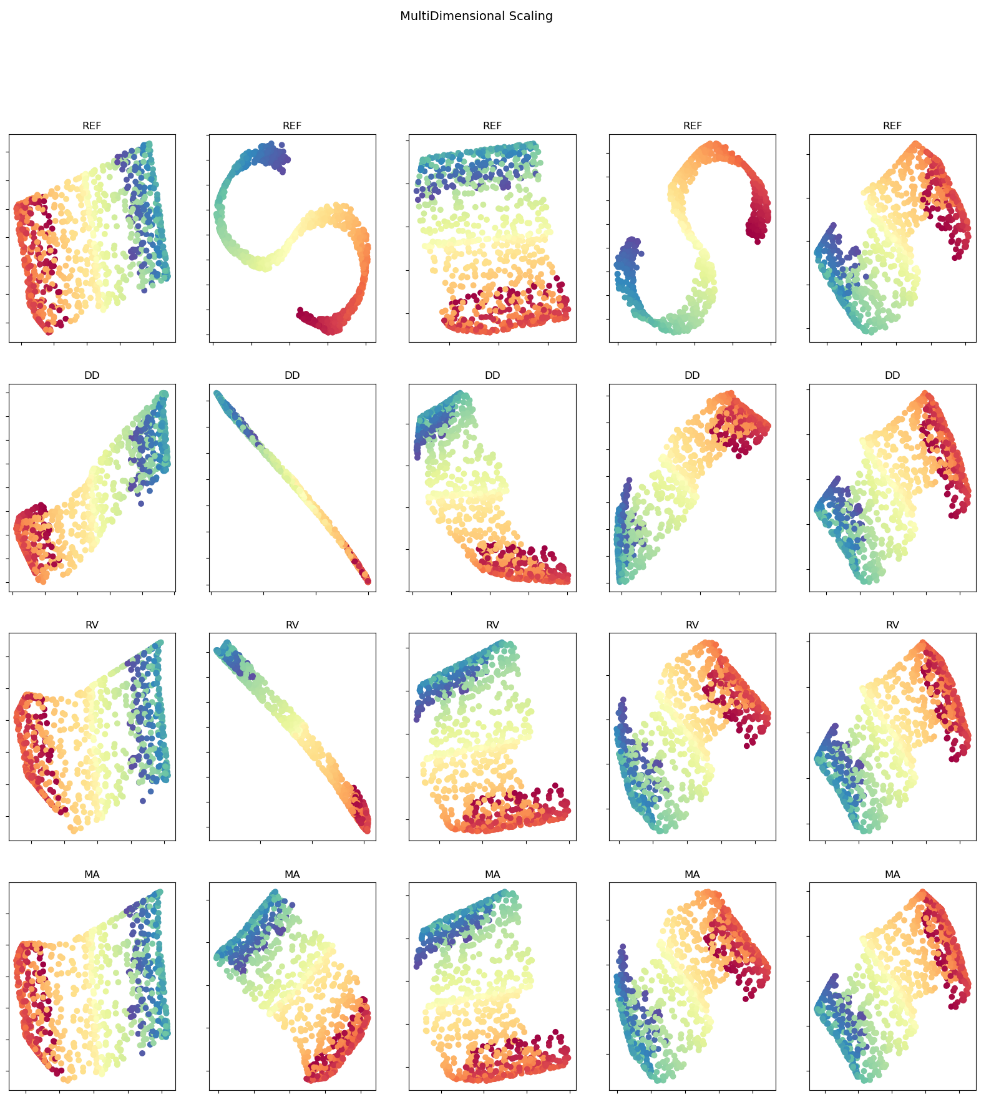


Figure IV.17: Five evaluations with different initialization of model for the S curve data, using manifold learning method MDS with four modes: REF, DD, RV (NA) and MA (NA). We see clearly the effect of Manifold Attack by the second column. Thus, the embedded representation of Manifold Attack is more spread compared to Random Virtual.

for hyper-parameters can be found in appendix B.3.1.

Firstly, in Mix-up Attack (1) and (2), we see clearly the trade-off between error rate for testing set and error rate for adversarial examples. If ξ takes a large value as in (1), the error rate for testing sample can be even worse than Mix-up (without using attack stage) about 0.5%, but it gains more than 10% for the robustness against adversarial examples. On the other hand, if ξ takes a smaller value as in (2), error rates for both testing sample and adversarial examples are smaller

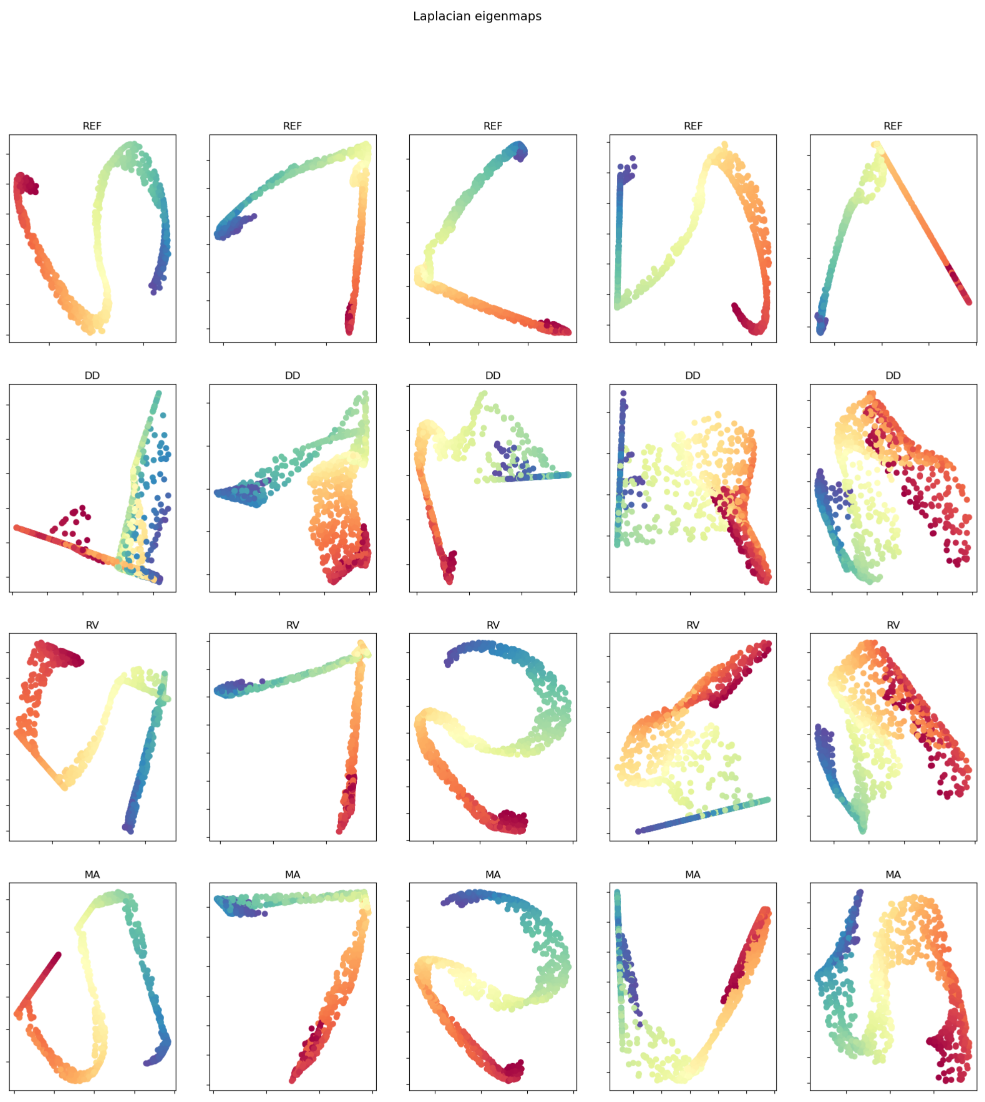


Figure IV.18: Five evaluations with different initialization of model for the S curve data, using manifold learning method LE with four modes: REF, DD, RV (NA) and MA (NA). We see clearly the effect of Manifold Attack by the fourth column, where the embedded representation shape of Manifold Attack is more similar to Reference than one of Random Virtual.

than those of Mix-Up, but it gains only about 5% for the robustness against adversarial examples. Secondly, Mix-up Attack (2) provides a worse error rate than Cut-Mix, about 0.4 % in the case of testing sample, but it gains about 10% in the case of adversarial examples. Therefore, we conclude that manifold attack not only improves accuracy rate on testing set but also significantly improve robustness of the model with respect to adversarial examples.

It is worth noting that in attack stage, the model $g()$ needs to be continuous. In the case of

Method / Data evaluation	Testing set		Adversarial examples	
	Top-1	Top-5	Top-1	Top-5
ERM	33.84	12.46	81.69	59.14
Mix-up [Zhang *et al.* 2017]	32.13	11.35	76.79	50.76
Mix-up Attack (1) ($\xi = 0.1 \to 0.01$)	32.57	10.98	65.60	37.57
Mix-up Attack (2) ($\xi = 0.01$)	31.35	10.81	71.31	44.5
Cut-Mix [Sangdoo *et al.* 2019]	30.94	10.41	81.24	58.72

Table IV.3: : ImageNet error rate (top-1 and top-5 in %) on testing set and on adversarial examples on different training modes: ERM, Mix-Up, Mix-Up Attack and Cut-Mix.

using NNMs with *Dropout* layer, the active connections need to be fixed in an attack stage to guarantee that $g()$ is continuous.

C.3 Semi-supervised manifold attack. We can apply manifold attack to all SSNN models that use manifold structure constraint (see section IV.A.2.7). In this experiment, we apply manifold attack on MixMatch [Berthelot *et al.* 2019b], one of best semi-supervised learning methods in the current state-of-the-art. The details of this approach can be found in section IV.A.6 (Holistic methods). As in Mix-up, we introduce attack stages to alternately optimize with model update stages. Note that, γ_1 in MixMatch is slightly different from Mix-up as:

$$\gamma_1 + \gamma_2 = 1, \gamma_1 \sim \text{Beta}(\alpha, \alpha) \text{ and } \gamma_1 \geq \gamma_2 \tag{IV.55}$$

Then the projection for γ_1 is now the clamping function between 0.5 and 1. When two separated variables γ_1 and γ_2 are defined, we can use the projector Π'_{ps} (IV.51) defined in section IV.B.4. We use the *Pytorch* implementation for MixMatch by Yui [Yui 2019] (with all hyper-parameters by default), then we introduce attack stages, with the number of iterations $n\_iters = 1$. In each experiment, for both CIFAR-10 and SVHN dataset, we divide the training set into three parts: labelled set, unlabelled set and validation set. The number of validation samples is fixed at 5000. The number of labelled samples is 250, and the remaining samples are part of the unlabelled set. We repeat the experiment four times, with different samplings of labelled samples, unlabelled samples, validation samples and different initialization of model Wide ResNet-28 [Zagoruyko & Komodakis 2016] which has about 1.47M trainable parameters. More details for hyper-parameters can be found in appendix B.3.2.

The error rate on testing set, which corresponds to the best validation error rate, is reported in table IV.4, for both MixMatch and MixMatch Attack. We see that MixMatch Attack improves the performance of MixMatch, about 1.5% less on error rate. There is a considerable difference

Data	Method / Test	1	2	3	4	Mean
CIFAR-10	MixMatch	10.62	12.72	12.02	15.26	12.65 ± 1.68
CIFAR-10	MixMatch Attack	8.84	10.46	10.09	12.89	10.57 ± 1.47
SVHN	MixMatch	6.0925	6.73	7.802	7.37	7.0 ± 0.65
SVHN	MixMatch Attack	5.07	5.93	5.42	5.47	5.47 ± 0.3

Table IV.4: : CIFAR-10 and SVHN Error rate in four different configurations, each configuration consists of data partitioning and initialization of model parameters. The number of labelled samples is fixed at 250 and the used model is Wide ResNet-28.

between the error rate of MixMatch reproduced by our experiments and the one reported from the official paper, which might come from the sampling, the initialization of model, the library used (Pytorch *vs* TensorFlow) and the computation of the error rate (error rate associated to best validation error *vs* the median error rate of the last 20 checkpoints).

Method / Data	CIFAR-10	SVHN
Pi Model $^\diamond$ [Laine & Aila 2016]	53.02 ± 2.05	17.56 ± 0.275
Pseudo Label $^\diamond$ [hyun Lee 2013]	49.98 ± 1.17	21.16 ± 0.88
VAT $^\diamond$ [Miyato *et al.* 2017]	36.03 ± 2.82	8.41 ± 1.01
SESEMI SSL [Tran 2019]		8.32 ± 0.13
Mean Teacher $^\diamond$ [Tarvainen & Valpola 2017]	47.32 ± 4.71	6.45 ± 2.43
Dual Student [Ke *et al.* 2019]		4.24 ± 0.10
MixMatch $^\diamond$ [Berthelot *et al.* 2019b]	11.08 ± 0.87	3.78 ± 0.26
MixMatch * [Berthelot *et al.* 2019b]	12.65 ± 1.68	7.0 ± 0.65
MixMatch Attack *	10.57 ± 1.47	5.47 ± 0.3
Real Mix [Nair *et al.* 2019]	9.79 ± 0.75	3.53 ± 0.38
EnAET [Wang *et al.* 2019]	7.6 ± 0.34	3.21 ± 0.21
ReMixMatch [Berthelot *et al.* 2019a]	6.27 ± 0.34	3.10 ± 0.50
Fix Match [Sohn *et al.* 2020]	5.07 ± 0.33	2.48 ± 0.38

Table IV.5: CIFAR-10 and SVHN error rate of different semi-supervised learning methods. The number of labelled sample is fixed at 250. ($^\diamond$) means that the results are reported from [Berthelot *et al.* 2019b]. (*) means that that the results are reported from our experiments. The resting results are reported from their corresponding official paper.

Table IV.5 shows error rates among semi-supervised methods based on DeL model, for both CIFAR-10 and SVHN dataset with only 250 labelled samples. We refer also readers to the site PapersWithCode that provides the lasted record for each dataset: CIFAR-10[1] and SVHN [2].

C.4 Conclusion about manifold attack. Manifold attack has several benefits. Firstly, it is generally compatible with NNMs that uses manifold structure constraint and improves their accuracy and robustness. Secondly, it is more general than adversarial noise (see for a comparison in table IV.6). By applying manifold attack instead of adversarial noise, we get a new type of

[1]CIFAR-10 `https://paperswithcode.com/sota/semi-supervised-image-classification-on-cifar-6`
[2]SVHN `https://paperswithcode.com/sota/semi-supervised-image-classification-on-svhn-1`

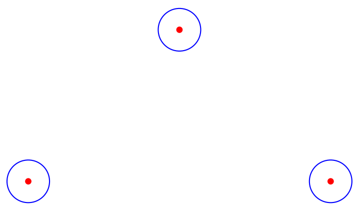
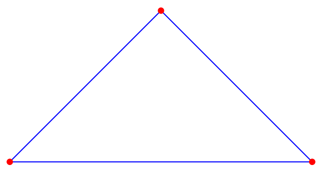
	Adversarial Noise	Manifold Attack
Illustration, Red point: a sample Blue line: border of feasible zone		
Feasible zone	Locality of each sample	Convex hull
Variable	Noise ϵ that has the same size as samples	γ has the size which equals to the number of anchor points
Manifold learning task	Points in the locality of a sample must have a similar embedded representation	Available for almost manifold learning tasks

Table IV.6: A simple comparison between adversarial noise and manifold attack in general.

Virtual Adversarial Training method (VAT, see section IV.A.5).

However, the anchoring points and initialization of attack points has to be carefully set so that attack points are close to the samples manifold. Besides, it yields an additional computational cost that can be important if the number of iterations in an attack stage is high. We provide a detailed analyse and some standard parallelizations methods to accelerate the training stage in appendix B.4. We also provide in appendix B.5 a method to mitigate the additional memory cost.

Chapter V

General Conclusion

In this chapter, we present a summary about our contributions to semi-supervised dictionary learning and semi-supervised neural network learning. Then, we highlight some links between these two approaches. Finally, we present the perspectives for each approach.

Summary of contributions

Semi-supervised dictionary learning. In chapter III, we presented three ways of extending a supervised dictionary learning method to a semi-supervised learning setting. The first straightforward approach consists in integrating unlabelled data into the reconstruction error and the sparse codes penalty. Another approach relies on manifold structure preservation. Finally, one can learn a semi-supervised internalw classifier. We propose our approach called SSDL-GA with two novelties. Firstly, we used the locally linear embedding (LLE) structure to constraint the sparse codes by preserving the manifold structure of the original representation into the sparse code space. Secondly, the manifold structure is taken into account when sparse coding out-of-sample data. When there is both a low number of labelled and unlabelled samples available during the training, our approach outperforms state-of-the-art SSDL methods on MNIST and USPS datasets. An implementation of this method can be found on Github[1].

Semi-supervised neural network learning. In chapter IV, we present our approach called manifold attack which is based on adversarial learning. Given a manifold structure preservation loss, we generate attack points that maximize that loss, and then use these attack points as additional samples to train the model. Our approach provides a significant improvement on the

[1]`https://github.com/ktran1/SSDL`

accuracy rate and especially on the robustness to adversarial examples on ImageNet, CIFAR-10 and SVHN datasets. In addition, by using attack points as additional training samples in a semi-supervised setting, our approach can deal with datasets that originally have a low number of training samples. We provide an implementation of this approach on Github [2].

Similarities

In general, semi-supervised dictionary learning models and semi-supervised neural network models share several points:

- Both of them use hidden representations and the classifier work on hidden representation instead of original representation. Dictionary learning models have only one hidden representation (sparse code) while neural network models can have more than one hidden representations.
- In both cases, manifold learning losses can be used to regularize hidden representations. In dictionary learning models, the manifold learning loss variables are the sparse codes, and indirectly the dictionary, while in neural network models, the manifold learning loss variables are the model's parameters.
- In order to further improve performance, both of approaches can benefit popular semi-supervised learning techniques such as pseudo labeling, sharpening ...

Perspectives

Here are some perspectives for semi-supervised dictionary learning:

+ Computing a non-linear internal classifier, for instance using a cross-entropy classification loss:

$$\left\|\mathbf{Y} - \mathbf{W}\mathbf{A}^l\right\|_F^2 \rightarrow -\sum_{i=1}^{N_l} \mathbf{y}_i^\top \log\left(softmax(\mathbf{W}\mathbf{a}_i^l)\right) = -\sum_{i=1}^{N_l}\sum_{k=1}^{C} \mathbf{y}_i[k] \log\left(\frac{\exp\left(\mathbf{w}_k^\top \mathbf{a}_i^l\right)}{\sum_{j=1}^{C} \exp\left(\mathbf{w}_j^\top \mathbf{a}_i^l\right)}\right) \tag{V.1}$$

  By allowing a more complex decision boundary for the internal classifier, the sparse code might end up more discriminative.

+ The Mix-up trick can be applied to dictionary learning, in the same fashion as for neural networks, for both labelled and unlabelled data.

[2]`https://github.com/ktran1/Manifold-attack`

+ Updating dictionary $\mathbf{D}$ by EMA process: $\mathbf{D} \leftarrow \alpha\mathbf{D} + (1-\alpha)\mathbf{D}_t$, where $\mathbf{D}_t$ is the optimal dictionary after iteration t.

We suggest following perspectives for the semi-supervised neural network learning:

+ In section IV.B.4 (Settings of anchor points and initialization of virtual points), for Neighbor anchors setting, we can go a little further by introducing a parameter s that controls expansion ($s > 1$) or contraction ($0 > s > 1$) of the convex hull as shown in figure IV.15. We can also think of optimizing the settings of anchor points and initialization for attack points.

+ Optimization of the layers between those the PGS needs to be applied. Indeed, we could also implement PGS between one latent representation and another one in NNMs as in [Verma *et al.* 2019a].

+ A shortcoming of manifold attack in the presented setting, is that it is not obvious to guarantee that attack points actually belong to the underlying samples manifold. One can consider using this methodology for generative models. Since latent representation follows a known distributions in this case, *e.g.* Normal distribution in VAE models or Uniform distribution in GAN models, anchor points are no longer required. The absolute coordinates of an attack point are the variables in maximizing manifold learning loss and only the generator is involved in the attack phase. Concretely, *Mode collapse* is a popular problem while training GAN models. GAN with samples that are well balanced among classes, generated samples by the generator are biased on only a few classes (as showed in figure V.1). This is because the latent representation is not well regularized. We expect to overcome the problem Mode collapse by introducing manifold attack from the latent representation back to the original representation as show in figure V.2 and by optimizing problem V.2, where $\mathcal{L}_t$ is a PGS task as showed in section II.D.

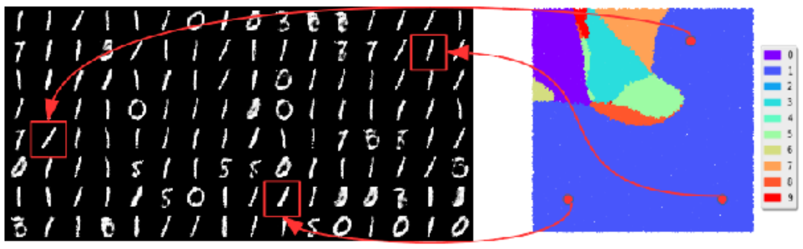


Figure V.1: Mode collapse observed in a GAN with the MNIST dataset. Left: Samples generated by the generator in GAN. Right: The latent representation of GAN that follows uniform distribution. In this example, the class 1 dominates in the samples generated. Credit: [Tran *et al.* 2018].

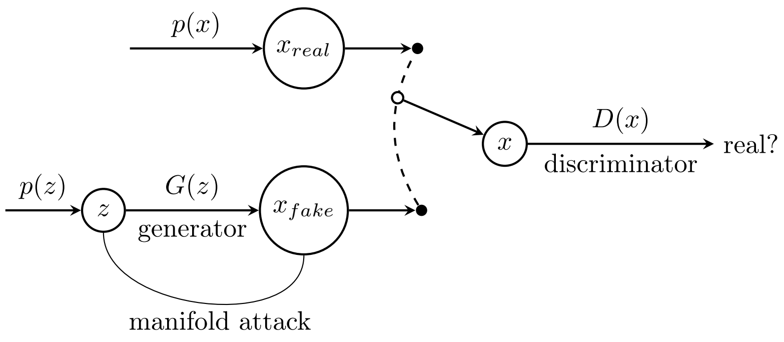


Figure V.2: An GAN that we apply PGS from the latent representation back to original representation.

$$\min_G \max_D \max_{z_1,..,z_M \in [0,1]^p} \Big(\mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[1 - \log D(G(z))] + \lambda \mathcal{L}_t\big(G(z_1),..,G(z_M)\big)\Big) \tag{V.2}$$

Research activities

Publications.

- Khanh-Hung Tran, Fred-Maurice Ngole-Mboula and Jean-Luc Starck. (2018). Semi-supervised dual graph regularized dictionary learning. In *Conference iTWIST'18.* [3]
- Khanh-Hung Tran, Fred-Maurice Ngole-Mboula, Jean-Luc Starck and Vincent Prost. (2020). Semisupervised Dictionary Learning with Graph Regularized and Active Points. In *SIAM Journal on Imaging Sciences.* [4]
- Khanh-Hung Tran, Fred-Maurice Ngole-Mboula and Jean-Luc Starck. (2020). Manifold attack. Submitted in *SIAM Journal on Mathematics of Data Science.* [5]

Conferences and schools.

- ADA9 summer School, 20-22 Mai 2018, Valencia: *Signal processing for astrophysics data sets.*
- Peyresq summer School, 01-07 July 2018, Castellane: *Signals, images and data science.*
- Itwist18 conference, 19-23 November 2018, Marseille: *Sparse or low-rank data models* (talk).
- DS3 summer school, 24-28 June 2019, Palaiseau: *data science* (Poster).

[3]`https://arxiv.org/html/1812.00648`
[4]`https://epubs.siam.org/doi/abs/10.1137/19M1285469?mobileUi=0`
[5]`https://arxiv.org/abs/2009.05965`

- Seminar *Wavelets and Beyond - A celebration for Alexandre Grossmann and Yves Meyer*, 12-14 June 2019, Orsay (Poster).

Teaching activity.

- Polytech Paris Saclay, Maths S3, 1st semester in 2018-2019 and 2019 - 2020.

Appendix A

1 Optimization for Laplacian Learning

$$\begin{aligned}
\min_{\mathbf{L}\in\mathbb{R}^{N\times N}} \quad & \operatorname{tr}\left(\mathbf{X}\mathbf{L}\mathbf{X}^\top\right) + \theta \left\|\mathbf{L}\right\|_F^2 \\
\text{subject to} \quad & \operatorname{tr}(\mathbf{L}) = N, \\
& \mathbf{L}_{ij} = \mathbf{L}_{ji} < 0 (i \neq j), \\
& \sum_j \mathbf{L}_{ij} = 0.
\end{aligned} \tag{A.1}$$

Beside the proposed solution in [Dong *et al.* 2016], we propose an other solution to solve quickly the problem A.1. The first two constraints can be used to reduce the number of variables. We express each diagonal element by all other ones in the same row (as $\sum_j \mathbf{L}_{ij} = 0$) and all strictly upper elements of $\mathbf{L}$ by strictly lower one (as $\mathbf{L} = \mathbf{L}^\top$). We denote vec() an operator that concatenate vertically all columns of a matrix and vlow an operator that concatenate vertically all strictly column lower parts. Let's vertorize matrix $\mathbf{L}$ by vector $\operatorname{vec}(\mathbf{L}) \in \mathbb{R}^{N^2}$ and let $\operatorname{vlow}(\mathbf{L}) \in \mathbb{R}^{\frac{N(N-1)}{2}}$ be the vector that contains all strictly lower elements of $\mathbf{L}$. We define $\mathcal{M} \in \mathbb{R}^{N^2\times\frac{N(N-1)}{2}}$ the matrix whose i^{th} row stocks the linear relation between i^{th} element of $\operatorname{vec}(\mathbf{L})$ and all elements of $\operatorname{vlow}(\mathbf{L})$. Then we have:

$$\mathcal{M}\operatorname{vlow}(\mathbf{L}) = \operatorname{vec}(\mathbf{L}) \tag{A.2}$$

Using the priorities of trace and (A.2), we have:

$$\begin{aligned}
& \operatorname{tr}\left(\mathbf{X}\mathbf{L}\mathbf{X}^\top\right) + \theta \left\|\mathbf{L}\right\|_F^2 \\
=& \operatorname{tr}\left(\mathbf{X}^\top\mathbf{X}\mathbf{L}\right) + \theta \operatorname{vec}(\mathbf{L})^\top \operatorname{vec}(\mathbf{L}) \\
=& \operatorname{vec}(\mathbf{X}\mathbf{X}^\top)^\top \operatorname{vec}(\mathbf{L}) + \theta (\mathcal{M}\operatorname{vlow}(\mathbf{L}))^\top \mathcal{M}\operatorname{vlow}(\mathbf{L}) \\
=& \operatorname{vec}(\mathbf{X}\mathbf{X}^\top)^\top \mathcal{M}\operatorname{vlow}(\mathbf{L}) + \theta \operatorname{vlow}(\mathbf{L})^\top \mathcal{M}^\top \mathcal{M}\operatorname{vlow}(\mathbf{L})
\end{aligned} \tag{A.3}$$

We rewrite the problem (A.1):

$$\begin{aligned} \min_{\text{vlow}(\mathbf{L})\in\mathbb{R}^{\frac{N(N-1)}{2}}} \quad & \text{vec}(\mathbf{X}\mathbf{X}^\top)^\top \mathcal{M}\text{vlow}(\mathbf{L}) + \theta \text{vlow}(\mathbf{L})^\top \mathcal{M}^\top \mathcal{M}\text{vlow}(\mathbf{L}) \\ \text{subject to} \quad & \sum_i \text{vlow}(\mathbf{L})[i] = -N/2, \\ & \text{vlow}(\mathbf{L})[i] \leq 0 \end{aligned} \tag{A.4}$$

This is a convex quadratic problem with convex constraints, then it can be solved by using Proximal Splitting Method [Combettes & Pesquet 2009]. The projection for the constraints is exactly problem in appendix B.1. We provide an implementation for this approach on Github [1].

(*) An important note, in practice, since $\mathcal{M} \in \mathbb{R}^{N^2 \times \frac{N(N-1)}{2}}$, it is not enough memory to stock this matrix for large value N, *e.g.*, $N \approx 1000$. Then we need to find a pertinent strategy to calculate $\text{vec}(\mathbf{X}\mathbf{X}^\top)^\top \mathcal{M}$ for premier order and $\mathcal{M}^\top \mathcal{M}$ for second order. A simple way is to reduce problem A.1 first for diagonal and strictly lower elements of $\mathbf{L}$ (by the symmetry) and then express diagonal elements in term of strictly lower elements.

2 Probability update

$$\begin{aligned} &\min_{\mathbf{P}\geq \mathbb{0}} \sum_{j=1}^{N_u} \sum_{k=1}^{C} (\mathbf{P}_{kj})^r \sum_{c=1}^{C} \mathbf{Q}_k^u[c,j] \left\| y_j^c(k)(\mathbf{w}_c^\top \mathbf{a}_j^u + b_c) - 1 \right\|_2^2, \\ &\text{subject to } \sum_{k=1}^{C} \mathbf{P}_{kj} = 1, \forall j. \end{aligned} \tag{A.5}$$

The solution is proposed by [Wang *et al.* 2014]. Obviously, this problem can be decoupled for each independent sample. So it is equivalent to solving:

$$\begin{aligned} &\min_{\mathbf{P}[:,k]\geq \mathbb{0}} \sum_{k=1}^{C} (\mathbf{P}[k,j])^r \mathbf{B}[k,j], \\ &\text{subject to } \sum_{k=1}^{C} \mathbf{P}[k,j] = 1. \end{aligned} \tag{A.6}$$

where $\mathbf{B}[k,j] = \sum_{c=1}^{C} \mathbf{Q}_k^u[c,j] \left\| y_j^c(k)(\mathbf{w}_c^\top \mathbf{a}_j^u + b_c) - 1 \right\|_2^2, \mathbf{B}[k,j] \geq 0$. If $r = 1$, then the solution is obvious:

[1] `https://github.com/ktran1/Leant_Laplacian`

$$\begin{cases} \mathbf{P}[j,k] = 1 \text{ if } k = \underset{i}{\operatorname{argmin}}\, \mathbf{B}[i,j] \\ \mathbf{P}[j,k] = 0 \text{ otherwise.} \end{cases}$$

If $r > 1$, we employ Lagrange multiplier to solve problem A.6:

$$\sum_{k=1}^{C} (\mathbf{P}[k,j])^r \mathbf{B}[k,j] + \beta\Big(\sum_{k=1}^{C} \mathbf{P}[k,j] - 1\Big) + \sum_{k=1}^{C} \mu_k \mathbf{P}[k,j]$$
$$\text{subject to: } \mu_1, \mu_2, .., \mu_C \leq 0$$

We solve the following system of equations:

$$\begin{cases} r(\mathbf{P}[k,j])^{r-1}\mathbf{B}[k,j] + \beta = 0 \\ \sum_{k=1}^{C} \mathbf{P}[k,j] = 1 \\ \mu_k \mathbf{P}[k,j] = 0 \\ \mu_k \leq 0 \\ \mathbf{P}[k,j] \geq 0 \end{cases} \Leftrightarrow \begin{cases} \mathbf{P}[k,j] = \big(\frac{-\beta}{r\mathbf{B}[k,j]}\big)^{\frac{1}{r-1}} \\ \sum_{k=1}^{C} \mathbf{P}[k,j] = 1 \\ \beta < 0 \\ \mu_k = 0 \end{cases} \Leftrightarrow \mathbf{P}[k,j] = \frac{(\mathbf{B}[k,j])^{\frac{-1}{r-1}}}{\sum_{k=1}^{C} (\mathbf{B}[k,j])^{\frac{-1}{r-1}}}$$

3 Sparse coding

$$\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \lambda \|\mathbf{A}\|_1 + \beta \operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) + \varphi \operatorname{tr}(\mathbf{D}^\top \mathbf{L}_D \mathbf{D})$$
$$+\gamma\Big(\big\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\big\|_F^2 + \sum_{k=1}^{C} \Big\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\Big\|_F^2 \Big) \tag{A.7}$$

By applying FISTA, we separate this problem into two functions f_1^{sp} and f_2^{sp}:

$$f_1^{sp}(\mathbf{A}) = \lambda \|\mathbf{A}\|_1$$
$$f_2^{sp}(\mathbf{A}) = \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \beta \operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) + \gamma \big\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\big\|_F^2$$
$$+ \gamma \sum_{k=1}^{C} \Big\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\Big\|_F^2$$

The gradient of f_2^{sp} is given by:

$$\nabla f_2^{sp}(\mathbf{A}) = -2\mathbf{D}^\top(\mathbf{X}-\mathbf{DA}) + 2\beta(\mathbf{AL}_A) + 2\gamma\Big[\mathbf{W}^\top\Big((\mathbf{Q}^l)^2 \circ (\mathbf{WA}^l - \mathbf{b1}_{N_l} - \mathbf{Y})\Big),$$
$$\sum_{k=1}^{C}\mathbf{W}^\top\Big((\mathbf{Q}_k^u)^2 \circ (\mathbf{1}_{N_u}\mathbf{P}[k,:])^r \circ (\mathbf{WA}^u - \mathbf{b1}_{N_u} - \mathbf{Y}_k)\Big)\Big],$$

where $(\mathbf{Q}^l)^2 = \mathbf{Q}^l \circ \mathbf{Q}^l$, $(\mathbf{Q}_k^u)^2 = \mathbf{Q}_k^u \circ \mathbf{Q}_k^u$. Algorithm 6 is used to solve for problem A.7.

Algorithm 6 Sparse coding by FISTA with backtracking.

```
Require: $\mathbf{X}, \mathbf{A}_0, \mathbf{D}, \mathbf{W}, \mathbf{b}, \mathbf{Y}, \mathbf{Y}_k, \mathbf{L}_A, \beta, \gamma, \lambda, \mathbf{Q}^l, \mathbf{Q}_k^u$.
  Initialize: $\mathbf{Z}_0 \leftarrow \mathbf{A}_0, t_0 \leftarrow 1, \tau > 0, \eta > 1$.
  for $n = 0, 1, ...$ do
    while True do
      $\mathbf{H} \leftarrow \mathbf{A}_n - \tau^{-1}\nabla f_2^{sp}(\mathbf{A}_n)$
      $\mathbf{Z}_{n+1} \leftarrow \text{sign}(\mathbf{H}) \circ \max(|\mathbf{H}| - \tau^{-1}\lambda, 0)$
      if $f_2^{sp}(\mathbf{Z}_{n+1}) \le f_2^{sp}(\mathbf{A}_n) + \langle \mathbf{Z}_{n+1} - \mathbf{A}_n, \nabla f_2^{sp}(\mathbf{A}_n)\rangle + \frac{\tau}{2}\|\mathbf{Z}_{n+1} - \mathbf{A}_n\|^2$ then
        break
      end if
      $\tau \leftarrow \eta\tau$
    end while
    $t_{n+1} \leftarrow \frac{1+\sqrt{4t_n^2+1}}{2}$
    $\upsilon \leftarrow 1 + \frac{t_n - 1}{t_{n+1}}$
    $\mathbf{A}_{n+1} \leftarrow \mathbf{Z}_n + \upsilon(\mathbf{Z}_{n+1} - \mathbf{Z}_n)$
  end for
```

FISTA with backtracking does not require a Lipschitz coefficient for ∇f_2^{sp}, but anyway, we try to calculate it to know relatively which factors that step descent depends on. We consider that $\|.\|$ is the Euclidean norm. By using these following inequalities that apply for two matrices E and F:

- $\|EF\| \le \|E\|\,\|F\|$,
- $\|E+F\| \le \|E\| + \|F\|$,
- $\|E \circ F\| \le \|F\|$ (if $\mathbb{0} \le E \le \mathbb{1}$),

and note that $\mathbb{0} \le \mathbf{Q}^l, \mathbf{Q}_k^u, (\mathbf{1}_{N_u}^\top\mathbf{P}[k,:])^r \le \mathbb{1}$, we prove that:

$$\begin{aligned}
&\|\nabla f_2^{sp}(\mathbf{A}_1) - \nabla f_2^{sp}(\mathbf{A}_2)\| \\
&\qquad \le 2\|\mathbf{D}^\top\mathbf{D}(\mathbf{A}_1 - \mathbf{A}_2)\| + 2\beta\|(\mathbf{A}_1 - \mathbf{A}_2)\mathbf{L}_A\| \\
&\qquad + 2\gamma\|\mathbf{W}^\top\|\left\|\left[\mathbf{W}(\mathbf{A}_1^l - \mathbf{A}_2^l), \sum_{k=1}^{C}\mathbf{W}(\mathbf{A}_1^u - \mathbf{A}_2^u)\right]\right\| \\
&\qquad \le 2\|\mathbf{D}^\top\mathbf{D}\|\|\mathbf{A}_1 - \mathbf{A}_2\| + 2\beta\|\mathbf{A}_1 - \mathbf{A}_2\|\|\mathbf{L}_A\| \\
&\qquad + 2\gamma\|\mathbf{W}^\top\|\|\mathbf{W}\|\sum_{k=1}^{C}\left\|\left[(\mathbf{A}_1^l - \mathbf{A}_2^l), (\mathbf{A}_1^u - \mathbf{A}_2^u)\right]\right\|
\end{aligned}$$

$$\leq \underbrace{2\left(\left\|\mathbf{D}^\top\mathbf{D}\right\| + \beta\left\|\mathbf{L}_A\right\| + \gamma C\left\|\mathbf{W}\right\|^2\right)}_{K}\left\|\mathbf{A}_1 - \mathbf{A}_2\right\|$$

The step size τ in the algorithm 6 is relatively proportional to $K^{-1} = \frac{1}{2}\left(\left\|\mathbf{D}^\top\mathbf{D}\right\| + \beta\left\|\mathbf{L}_A\right\| + \gamma C\left\|\mathbf{W}\right\|^2\right)^{-1}$. As we expect to find the maximum possible τ to converge quickly to optimal solution, τ depends on following factors:

- $\left\|\mathbf{D}^\top\mathbf{D}\right\|$, which is good if atoms in dictionary are orthogonal and $\|\mathbf{D}\|$ is small.

- $\beta\left\|\mathbf{L}_A\right\|$, which is good if energy of Laplacian matrix $\mathbf{L}_A$ is small.

- $\gamma C\left\|\mathbf{W}\right\|^2$, which is good if number of classes C is small or energy of classifier $\|\mathbf{W}\|$ is small.

In case of using Laplacian-based matrix by LLE, if the number of nearest samples k is too large or training data point are not regularly sampled from the data distribution, it happens that a sample j (in the center of data distribution) are always in the neighborhood of other samples i, therefore $\mathbf{V}[i,j] = \lambda_{ij}$ is not zero for all i, which means column j of $\mathbf{V}$ are not sparse and $\|\mathbf{L}_A\| \approx \left\|\mathbf{V}^\top\mathbf{V}\right\|$ becomes large. Then step size τ can be small.

4 Dictionary update

$$\min_{\mathbf{D}\in\mathcal{C}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \varphi\,\mathrm{tr}(\mathbf{D}^\top\mathbf{L}_D\mathbf{D}) \tag{A.8}$$

Let note $f^{du}(\mathbf{D}) = \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \varphi\,\mathrm{tr}(\mathbf{D}^\top\mathbf{L}_D\mathbf{D})$. The gradient of f^{du} is given by:

$$\nabla f^{du}(\mathbf{D}) = -2(\mathbf{X} - \mathbf{D}\mathbf{A})\mathbf{A}^\top + 2\varphi\mathbf{L}_D^\top\mathbf{D}$$

As in sparse coding, we try to find a Lipschitz coefficient:

$$\begin{aligned}
&\|\nabla f^{du}(\mathbf{D}_1) - \nabla f^{du}(\mathbf{D}_2)\| \\
=&2\|(\mathbf{D}_1 - \mathbf{D}_2)\mathbf{A}\mathbf{A}^\top + \varphi\mathbf{L}_D^\top(\mathbf{D}_1 - \mathbf{D}_2)\| \\
\leq&2\|\mathbf{D}_1 - \mathbf{D}_2\|\|\mathbf{A}\mathbf{A}^\top\| + 2\varphi\|\mathbf{L}_D^\top\|\|\mathbf{D}_1 - \mathbf{D}_2\| \\
\leq&\underbrace{2\left(\|\mathbf{A}\mathbf{A}^\top\| + \varphi\|\mathbf{L}_D^\top\|\right)}_{K}\|\mathbf{D}_1 - \mathbf{D}_2\|
\end{aligned}$$

Algorithm 7 is used to solve for problem A.8. The step size τ in this algorithm is relatively proportional to $K^{-1} = \frac{1}{2}\left(\|\mathbf{A}\mathbf{A}^\top\| + \varphi\|\mathbf{L}_D^\top\|\right)^{-1}$. Therefore, the step size τ relates to $\left\|\mathbf{A}\mathbf{A}^\top\right\|$,

Algorithm 7 Dictionary update by FISTA with backtracking.

```
Require: $\mathbf{X}, \mathbf{A}, \mathbf{D}_0, \varphi, \mathbf{L}_D$.
  Initialize: $\mathbf{Z}_0 \leftarrow \mathbf{D}_0, t_0 \leftarrow 1, \tau > 0, \eta > 1$.
  for $n = 0, 1, ...$ do
    while True do
      $\mathbf{H} \leftarrow \mathbf{D}_n - \tau^{-1}\nabla f^{du}(\mathbf{D}_n)$
      $\mathbf{Z}_{n+1} \leftarrow \underset{\mathbf{D}\in\mathcal{C}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{H} - \mathbf{D}\|_2^2$
      if $f^{du}(\mathbf{Z}_{n+1}) \leq f^{du}(\mathbf{D}_n) + \langle \mathbf{Z}_{n+1} - \mathbf{D}_n, \nabla f^{du}(\mathbf{D}_n)\rangle + \frac{\tau}{2}\|\mathbf{Z}_{n+1} - \mathbf{D}_n\|^2$ then
        break
      end if
      $\tau \leftarrow \eta\tau$
    end while
    $t_{n+1} \leftarrow \frac{1+\sqrt{4t_n^2+1}}{2}$
    $\upsilon \leftarrow 1 + \frac{t_n - 1}{t_{n+1}}$
    $\mathbf{D}_{n+1} \leftarrow \mathbf{Z}_n + \upsilon(\mathbf{Z}_{n+1} - \mathbf{Z}_n)$
  end for
```

which implies that $\mathbf{A}$ needs to be sparse, so λ need to be large enough. In our work, if λ is too small, we use the Dictionary Update algorithm in [Lee *et al.* 2007] instead of FISTA with backtracking.

5 Classifier update

$$\min_{\mathbf{W},\mathbf{b}} \gamma\Big(\left\|\mathbf{Q}^l \circ (\mathbf{W}\mathbf{A}^l + \mathbf{b}\mathbf{1}_{N_l} - \mathbf{Y})\right\|_F^2 + \sum_{k=1}^{C} \left\|\mathbf{Q}_k^u \circ (\mathbf{1}_{N_u}^\top \mathbf{P}[k,:])^{r/2} \circ (\mathbf{W}\mathbf{A}^u + \mathbf{b}\mathbf{1}_{N_u} - \mathbf{Y}_k)\right\|_F^2 \Big) + \mu(\|\mathbf{W}\|_F^2 + \|\mathbf{b}\|_2^2) \tag{A.9}$$

We rewrite this problem with more details:

$$\begin{aligned} f_{cu}(\mathbf{W}) = {} & \gamma \sum_{i=1}^{N_l}\sum_{c=1}^{C} \mathbf{Q}^l[c,i] \left\|\mathbf{w}_c^\top \mathbf{a}_i^l + b_c - y_i^c\right\|_2^2 \\ & + \gamma \sum_{j=1}^{N_u}\sum_{k=1}^{C} (\mathbf{P}_{kj})^r \sum_{c=1}^{C} \mathbf{Q}_k^u[c,j] \left\|\mathbf{w}_c^\top \mathbf{a}_j^u + b_c - y_j^c(k)\right\|_2^2 + \mu \sum_{c=1}^{C} \left(\|\mathbf{w}_c\|_2^2 + b_c^2\right) \end{aligned}$$

Then, as each binary classifier c is independent, we calculate gradient for each one:

$$\begin{aligned} \nabla f_{cu}(\hat{\mathbf{w}}_c) = {} & 2\gamma \sum_{\mathbf{Q}^l[c,i]=1} \hat{\mathbf{a}}_i^l \left(\hat{\mathbf{a}}_i^{l\top}\hat{\mathbf{w}}_c - y_i^c\right) \\ & + 2\gamma \sum_{\mathbf{Q}_k^u[c,j]=1} \sum_{k=1}^{C} (\mathbf{P}_{kj})^r \hat{\mathbf{a}}_j^u \left(\hat{\mathbf{a}}_j^{u\top}\hat{\mathbf{w}}_c - y_j^c(k)\right) + 2\mu\hat{\mathbf{w}}_c, \end{aligned}$$

where $\hat{\mathbf{w}}_c = \begin{bmatrix} \mathbf{w}_c \\ b_c \end{bmatrix}$, $\hat{\mathbf{a}}_i^l = \begin{bmatrix} \mathbf{a}_i^l \\ 1 \end{bmatrix}$ and $\hat{\mathbf{a}}_j^u = \begin{bmatrix} \mathbf{a}_j^u \\ 1 \end{bmatrix}$. The optimal binary classifier $\hat{\mathbf{w}}_c$ is obtained by solving the first order optimality:

$$\hat{\mathbf{w}}_c = \Bigg(\sum_{\mathbf{Q}^l[c,i]=1} \hat{\mathbf{a}}_i^l \hat{\mathbf{a}}_i^{l\top} + \sum_{\mathbf{Q}_k^u[c,j]=1} \sum_{k=1}^{C} (\mathbf{P}_{kj})^r \hat{\mathbf{a}}_j^u \hat{\mathbf{a}}_j^{u\top} + \frac{\mu}{\gamma} I \Bigg)^{-1} \Bigg(\sum_{\mathbf{Q}^l[c,i]=1} \hat{\mathbf{a}}_i^l y_i^c + \sum_{\mathbf{Q}_k^u[c,j]=1} \sum_{k=1}^{C} (\mathbf{P}_{kj})^r \hat{\mathbf{a}}_j^u y_j^c(k) \Bigg)$$

We employ function *einsum* (numpy) to parallelize calculations for all binary classifiers $\hat{\mathbf{w}}_c$ at the same time.

6 Sparse coding with epoch and batch

In case of large number of training samples (labelled and unlabelled), we can employ the epoch and batch strategy to minimize the Sparse Coding problem with single device. For example, we repeat n epochs for m batches divided from $\mathbf{A}$. Note that only manifold learning loss needs to be slightly modified since it has the interaction between sparse codes via the matrix $\mathbf{L}_A$. We define $\mathbb{U}$ as the set that contains all indices of the training samples, thus $\mathbb{U} = \{1, 2, .., N\}$. In an epoch, for each batch i, we select randomly n_i training samples ($n_i \approx N/m$) whose indices are stored in the set M_i. M_i^C is the complement of M_i in $\mathbb{U}$. We rewrite:

$$\begin{aligned}
&\operatorname{tr}(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top) \\
&= \operatorname{tr}\left([\mathbf{A}[:, M_i], \mathbf{A}[:, M_i^C]] \begin{bmatrix} \mathbf{L}_A[M_i, M_i] & \mathbf{L}_A[M_i, M_i^C] \\ \mathbf{L}_A[M_i, M_i^C]^\top & \mathbf{L}_A[M_i^C, M_i^C] \end{bmatrix} [\mathbf{A}[:, M_i], \mathbf{A}[:, M_i^C]]^\top \right) \\
&= \operatorname{tr}\left([\mathbf{A}_i, \mathbf{A}_i^C] \begin{bmatrix} \mathbf{L}_{A[ii]} & \mathbf{L}_{A[iC]} \\ \mathbf{L}_{A[iC]}^\top & \mathbf{L}_{A[CC]} \end{bmatrix} [\mathbf{A}_i, \mathbf{A}_i^C]^\top \right) \\
&= \operatorname{tr}(\mathbf{A}_i \mathbf{L}_{A[ii]} \mathbf{A}_i^\top) + 2 \operatorname{tr}(\mathbf{A}_i \mathbf{L}_{A[iC]} \mathbf{A}_i^{C\top}) + \operatorname{tr}(\mathbf{A}_i^C \mathbf{L}_{A[CC]} \mathbf{A}_i^{C\top}),
\end{aligned}$$

where:

- $\mathbf{L}_{A[ii]} = \mathbf{L}_A[M_i, M_i]$ is a $\mathbb{R}^{n_i \times n_i}$ matrix formed by extracting from $\mathbf{L}_A$, rows M_i and columns M_i.

- $\mathbf{L}_{A[CC]}$ is a $\mathbb{R}^{(N-n_i)\times(N-n_i)}$ matrix formed by extracting from $\mathbf{L}_A$ rows M_i^C and columns M_i^C.

- $\mathbf{L}_{A[iC]}$ is a $\mathbb{R}^{(n_i)\times(N-n_i)}$ matrix formed by extracting from $\mathbf{L}_A$ rows M_i and columns M_i^C.

Then we optimize for each $\mathbf{A}_i$ while fixing other batches ($\mathbf{A}_i^C$) as in the FISTA method for Sparse Coding mentioned before but this time we need to adjust the gradient of $2\,\mathrm{tr}\,(\mathbf{A}_i\mathbf{L}_{A[iC]}\mathbf{A}_i^{C\top})$ in the new ∇f_2^{sp} (in A.3). At the end of a batch, we update the sparse code before performing next batch. We suggest also evaluating the objective function after each step, as $\mathbf{Q}^l$, $\mathbf{Q}_k^u$ are fixed in an iteration, the value of objective function must decrease. Since $\mathbf{Q}^l$, $\mathbf{Q}_k^u$ are updated at the beginning of the next iteration, the objective function can increase slightly, but in general, we must see something decreases and converges.

7 Parallel training for dictionary learning model

In this section, we provide two strategies to parallelize training stage for DiL model with multiple devices. We consider here the proposed SSDL method (section III.B with objective function III.8). The model contains $\mathbf{D}, \mathbf{W}, \mathbf{b}, \mathbf{P}, \mathbf{Q}^l, \mathbf{Q}_k^u$. We try to adapt each optimization stage from A.2 to A.5 and the probability update for parallel training.

We split the whole dataset $\mathbf{X}$, sparse codes $\mathbf{A}$ and labels $\mathbf{Y}, \mathbf{Y}_k$ into N_{split} batches, which means that a batch contains samples, theirs corresponding sparse codes and theirs corresponding label.

Data parallel: if the memory of each device is enough to stock the model, we clone the model for each device and associate a batch to a device. The active elements indexed by $\mathbf{Q}^l, \mathbf{Q}_k^u$ and the probability for unlabelled samples $\mathbf{P}$ can be updated independently for each batch. In one iteration of dictionary update (algorithm 7) and in classifier update, we compute independently each gradient for each batch, then aggregate all gradients before updating dictionary and classifier. Paralleling sparse coding stage is difficult because of the dependencies between samples, which is represented by the manifold learning loss $2\beta\,\mathrm{tr}\,(\mathbf{A}\mathbf{L}_A\mathbf{A}^\top)$. We do not really have an efficient solution for this stage. Our proposition is as follows. For one iteration in algorithm 6, we need to compute first its gradient: $\nabla_{\mathbf{A}} = 2\beta\mathbf{A}\mathbf{L}_A$. Then we split $\nabla_{\mathbf{A}}$ to N_{split} parts in the same manner as with the whole dataset and distribute each part to the corresponding device. Now, we can compute independently each gradient for each batch and update separately sparse codes.

Split model: Now, if the memory of each device is not enough to stock the model, we split the model into N_m parts, which means $\mathbf{D}, \mathbf{W}, \mathbf{b}, \mathbf{P}, \mathbf{Q}^l, \mathbf{Q}_k^u$ are horizontally split such as:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_m} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_{N_m} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{N_m} \end{bmatrix} \quad \mathbf{Q}^l = \begin{bmatrix} \mathbf{Q}_1^l \\ \vdots \\ \mathbf{Q}_{N_m}^l \end{bmatrix} \quad \mathbf{Q}_k^u = \begin{bmatrix} (\mathbf{Q}_k^u)_1 \\ \vdots \\ (\mathbf{Q}_k^u)_{N_m} \end{bmatrix} \tag{A.10}$$

Each device now associated to $\mathbf{D}_i, \mathbf{W}_i, \mathbf{b}_i, \mathbf{P}_i, \mathbf{Q}_i^l, (\mathbf{Q}_k^u)_i$. We split also the whole dataset, sparse codes and labels into N_{split} batches as in the case of Data parallel. The batches are now processed one by one, from 1 to N_{split} instead of at the same time in Data parallel. The five optimization stages are the same as in Data parallel but we reduce:

- The memory required for a device to stock a part of model is divided by N_m.
- Regarding to the memory required to load a batch, in the five stages, the memory required to load a batch j is also reduced. For the reconstruction loss as described in A.11, only $\mathbf{X}_{ij}$ (instead of $\mathbf{X}_j$) is loaded in device i. In the same manner, for classification loss, only a part of $\mathbf{Y}$ and $\mathbf{Y}_k$ is loaded in a device.

$$\begin{aligned} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 &= \left\| \begin{bmatrix} \mathbf{X}_1 & \cdots & \mathbf{X}_{N_{split}} \end{bmatrix} - \begin{bmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N_m} \end{bmatrix} \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_{N_{split}} \end{bmatrix} \right\|_F^2 \\ &= \left\| \begin{bmatrix} \mathbf{X}_{11} & \cdots & \mathbf{X}_{1N_{split}} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{N_m N_{split}} & \cdots & \mathbf{X}_{N_m N_{split}} \end{bmatrix} - \begin{bmatrix} \mathbf{D}_1\mathbf{A}_1 & \cdots & \mathbf{D}_1\mathbf{A}_{N_{split}} \\ \vdots & \ddots & \vdots \\ \mathbf{D}_{N_m}\mathbf{A}_1 & \cdots & \mathbf{D}_{N_m}\mathbf{A}_{N_{split}} \end{bmatrix} \right\|_F^2 \\ &= \sum_{i=1}^{N_m} \sum_{j=1}^{N_{split}} \|\mathbf{X}_{ij} - \mathbf{D}_i\mathbf{A}_j\|_F^2 \end{aligned} \tag{A.11}$$

8 Pseudo labeling

In short, pseudo labeling consists of methods that create pseudo labels for each unlabelled sample then use those unlabelled samples as labelled samples to train the model. As opposed to "real" labels, pseudo labels are updated during the training stage. In this subsection, we first revisit two methods of Label Propagation (LP), then we present a trick called *sharpening* which is widely applied in Pseudo-labeling.

8.1 Label Propagation. Label Propagation is a semi-supervised learning method, that can be used for creating pseudo labels.

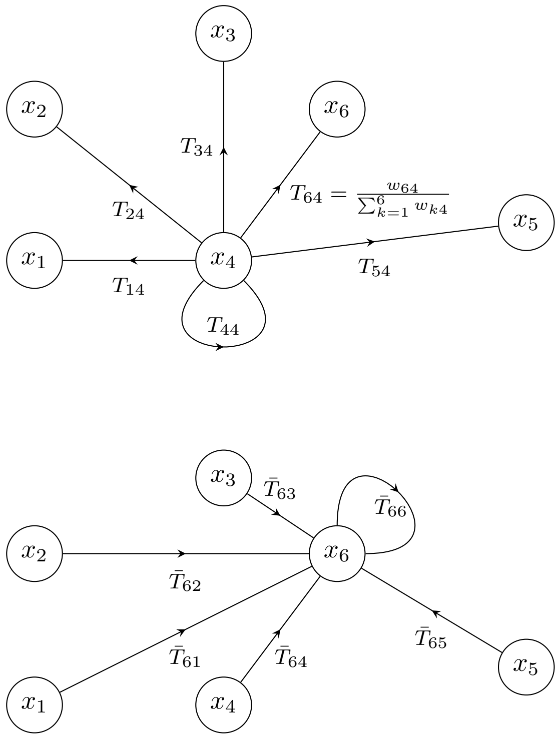


Figure A.1: Above: sample x_4 as transmitter, weight w is normalized with respect to each receiver sample. Below: sample x_6 as receiver which takes label information from all samples.

The first Label Propagation method was proposed by [Zhou *et al.* 2004]. $Y_L \in \mathbb{R}^{N_l \times C}$ and $Y_U \in \mathbb{R}^{N_u \times C}$ denote label matrix for labelled samples and guessed label matrix for unlabelled samples respectively. $Y_L[i,j] = 1$ if sample x_i belongs to class j, otherwise $Y_L[i,j] = 0$. Y_U is randomly initialized between 0 and 1, then row-normalized. The complete label matrix is defined as $Y = \begin{bmatrix} Y_L \\ Y_U \end{bmatrix}$. Since label information is propagated based on the locality, we must define a matrix W that measures pairwise similarity between samples:

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|_2^2}{\sigma^2}\right) \tag{A.12}$$

Then the probabilistic transition matrix T of size $N \times N$ is defined as:

$$T_{ij} = P(j \to i) = \frac{W_{ij}}{\sum_{k=1}^{N} W_{kj}} \tag{A.13}$$

T can be written as $T = WD^{-1}$, where D is diagonal matrix and $D_{ii} = \sum_{k=1}^{N} W_{ki}$. The column-normalization helps a transmitter sample to normalize its influence versus all samples, including itself, as illustrated in figure A.1 (above). Finally, the Label Propagation algorithm is described as follows:

1. Propagate $Y \leftarrow TY$
2. Row-normalize Y
3. Reset Y_L to its initial value.
4. Repeat from step 1 until Y converges.

The step 1 and step 2 can be combined into:

$$Y \leftarrow \bar{T}Y \tag{A.14}$$

with $\bar{T}$ is row-normalization of T such as $\bar{T} = D^{-1}T$, where D is diagonal matrix and $D_{ii} = \sum_{j=1}^{N} T_{ij}$. The row-normalization helps a receiver to normalize label information sent from all samples, including itself, as illustrated in figure A.1 (below).

The initialization of rows of Y_U is not important since it is proven in [Zhou *et al.* 2004] that Y_U always converges to Y_U^* regardless of initialization for Y_U. Here is the explicit form for Y_U^*:

$$Y_U^* = (I - \bar{T}_{uu})^{-1}\bar{T}_{ul}Y_L \tag{A.15}$$

where I is an identity matrix and $\bar{T} = \begin{bmatrix} \bar{T}_{ll} & \bar{T}_{lu} \\ \bar{T}_{ul} & \bar{T}_{uu} \end{bmatrix}$. Note that, we use an iterative algorithm instead of calculating directly Y_U^* because the direct matrix inversion is computationally heavy for large value of N.

The second Label Propagation was proposed by [Zhou *et al.* 2004] and is described as follows:

1. Define $W_{ij} = \exp(-\|x_i - x_j\|_2^2/2\sigma^2)$ if $i \neq j$ and $W_{ii} = 0$.
2. Construct the matrix $\bar{T} = D^{-1/2}WD^{-1/2}$ where D is diagonal matrix and $D_{ii} = \sum_{j=1}^{N} W_{ij}$.
3. Iterate $F \leftarrow \alpha\bar{T}F + (1-\alpha)Y$ until convergence, where $\alpha \in (0, 1)$.

F converges to F^* given explicitly by

$$F^* = (I - \alpha\bar{T})^{-1}Y \tag{A.16}$$

In this second version, there are three main points which are different from the first Label Propagation. Firstly, for probabilistic transition matrix $\bar{T}$, instead of using column-normalization followed by row-normalization from W, $\bar{T}$ is created by proceeding to a symmetric normalization of W, which is necessary for the convergence of step 3. This symmetric normalization can be considered as a mix of column-normalization (by multiplying with $D^{-1/2}$ on the right) for transmitters and of row-normalization (by multiplying with $D^{-1/2}$ on the left) for receivers. Secondly, the labels of the labeled samples might change to fit better the propagation. Finally, F^* depends on the initialization of Y_U as in equation A.16. In the original paper, authors initialize Y_U by a zero matrix.

It is also interesting to observe that matrix F^* is the optimal solution for minimizing the following cost function:

$$J(F) = \frac{\alpha}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \bar{T}_{ij} \left\| \frac{F[i,:]}{D_{ii}} - \frac{F[j,:]}{D_{jj}} \right\|_2^2 + (1-\alpha) \|F - Y\|_F^2 \tag{A.17}$$

8.2 Sharpening. This is a trick that helps to sharpen the output distribution of probability $[p_1, p_2, .., p_C]$. This distribution is controlled by the activation hyper-parameter $r > 1$ as follows:

$$p_k = \frac{p_k^r}{\sum_{i=1}^{C} p_i^r} \tag{A.18}$$

In some approaches for classification, if soft-max layer is used, sharpening by equation A.18 is equivalent to

$$p_k = \frac{\exp(z[k])^{\frac{1}{\tau}}}{\sum_{i=1}^{C} \exp(z[i])^{\frac{1}{\tau}}} \tag{A.19}$$

where z is normally the hidden representation just before soft-max layer and τ is called temperature and equivalent to $1/r$. Figure A.2 shows the effect of sharpening, which increases strong probabilities and decreases weak probabilities. Note that, sharpening is a double-edged sword, since it can increase probabilities for wrong pseudo labels.

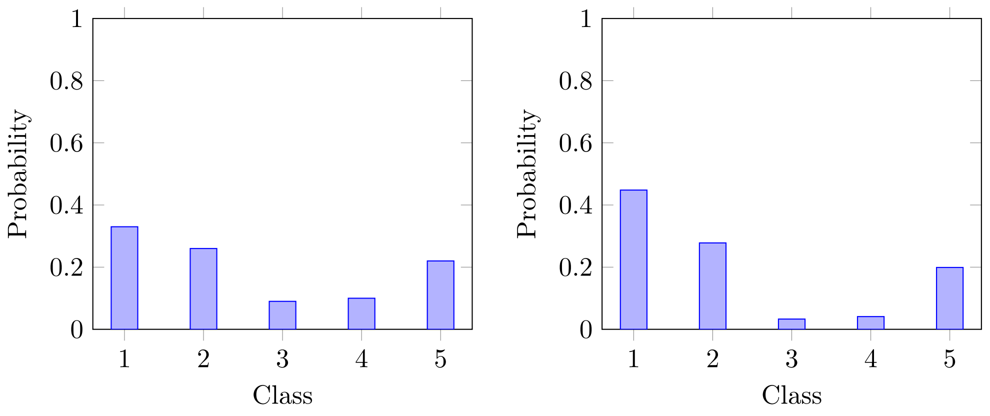


Figure A.2: Left: probability before sharpening. Right: probability after sharpening, the dominant probability (class 1) is significantly increased. In this case the sharpening factor $r = 2$ which is equivalent to $\tau = 0.5$.

Appendix B

1 Projection for sum and positive constraint

$$\begin{aligned} &\min_{\gamma\in\mathbb{R}^p} \frac{1}{2}\left\|\kappa-\gamma\right\|_2^2, \\ \text{subject to : } &\gamma_1,\gamma_2,..,\gamma_p \geq 0, \\ &\gamma_1+\gamma_2+...+\gamma_p = c, (c>0). \end{aligned} \tag{B.1}$$

By using Lagrange multiplier, the above problem becomes:

$$\min_{\gamma,\mu\in\mathbb{R}^p,\lambda} \frac{1}{2}\sum_{i=1}^{p}\left\|\kappa_i-\gamma_i\right\|_2^2 + \lambda(\sum_{i=1}^{p}\gamma_i - c) + \sum_{i=1}^{p}\mu_i\gamma_i$$
$$\text{subject to : } \mu_1,\mu_2,..,\mu_p \leq 0$$

We solve the following system of equations:

$$\begin{cases} \gamma_i-\kappa_i+\lambda+\mu_i=0 \\ \sum_{i=1}^{p}\gamma_i = c \\ \mu_i\gamma_i=0 \\ \mu_i\leq 0 \\ \gamma_i\geq 0 \end{cases} \Leftrightarrow \begin{cases} \lambda = \frac{1}{p}(\sum_{i=1}^{p}\kappa_i - \sum_{i=1}^{p}\mu_i - c) \\ \gamma_i = \kappa_i - \frac{1}{p}(\sum_{i=1}^{p}\kappa_i - c) - \frac{p-1}{p}\mu_i + \frac{1}{p}\sum_{j\neq i}\mu_j \\ \sum_{i=1}^{p}\gamma_i = c \\ \mu_i\gamma_i=0 \\ \mu_i\leq 0 \\ \gamma_i\geq 0 \end{cases}$$

In the case that $\kappa_i - \frac{1}{p}(\sum_{i=1}^{p}\kappa_i - c) < 0$. From the second equation, we infer that $\mu_i \neq 0$ (because if $\mu_i = 0$ and $\mu_j \leq 0$ as in inequality 5, then $\gamma_i < 0$, in contradiction to inequality 6). From $\mu_i \neq 0$, we infer that $\gamma_i = 0$ with equation 4.

In the case that $\kappa_i - \frac{1}{p}(\sum_{i=1}^p \kappa_i - c) = 0$. From the second equation, first if $\mu_i = 0$ then $\gamma_i \leq 0$ since $\mu_j \leq 0$ as in equation 5. With inequality 6, we infer $\gamma_i = 0$. Second, if $\mu_i \neq 0$, then we infer that $\gamma_i = 0$ with equation 4.

Let's $\mathcal{P} = \{i | \kappa_i - \frac{1}{p}(\sum_{i=1}^p \kappa_i - c) > 0\}$ and $\mathcal{N} = \{i | \kappa_i - \frac{1}{p}(\sum_{i=1}^p \kappa_i - c) \leq 0\}$. We find exactly the same problem as before, but with only active index in the set $\mathcal{P}$.

$$\begin{cases} \gamma_i - \kappa_i + \lambda + \mu_i = 0, \forall i \in \mathcal{P} \\ \sum_{i \in \mathcal{P}} \gamma_i = c \\ \mu_i \gamma_i = 0, \forall i \in \mathcal{P} \\ \mu_i \leq 0, \forall i \in \mathcal{P} \\ \gamma_i \geq 0, \forall i \in \mathcal{P} \end{cases}$$

Then we repeat until the constraint satisfaction for γ. For a proof of convergence, as γ has exactly p elements γ_i, then each time we project to get a new active set $\mathcal{P}$, we reduce the number of active elements γ_i. As the number of active elements is something positive and it decreases, so it converges. Here is an implementation for multiple κ ($\kappa \in \mathbb{R}^{M \times p}$) in *pytorch*.

```
def prox_positive_and_sum_constraint(x,c):
    """ x is 2-dimensional array (M \times p) """
    n = x.size()[1]
    k = (c - torch.sum(x,dim=1))/float(n)
    x_0 = x + k[:,None]
    while len(torch.where(x_0 < 0)[0]) != 0:
        idx_negative = torch.where(x_0 < 0)
        x_0[idx_negative] = 0.
        one = x_0 > 0
        n_0 = one.sum(dim=1)
        k_0 =(c - torch.sum(x_0,dim =1))/ n_0
        x_0 = x_0 + k_0[:,None] * one
    return x_0
```

2 Adversarial examples

An adversarial example for a given ML model is a slightly modified sample which induces a significantly different model output. For example, in classification problem, despite of high

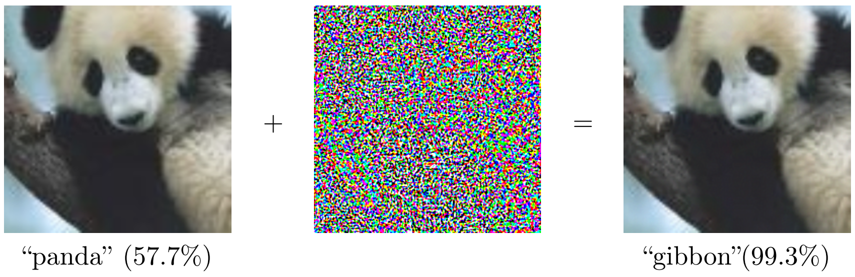


Figure B.1: A demonstration for adversarial example. Left: initial sample. Middle: adversarial noise. Right: adversarial example. We see that by adding an imperceptible noise, we can change significantly output compared to the one of initial sample. Source: [Goodfellow *et al.* 2014b].

accuracy given by neural network models, it has been shown that these models are very sensitive to a small amplitude calibrated noise, as illustrated in figure B.1. Here is the general objective function for generating an adversarial example $\tilde{x}$ from a given sample x and model f:

$$\begin{aligned} &\min_{\tilde{x}} \|\tilde{x} - x\| \\ \text{subject to: } &\operatorname*{argmax}_l f(\tilde{x})[l] \neq \operatorname*{argmax}_l f(x)[l] \\ &\tilde{x} \in [0,1]^n \text{ if } x \text{ is normalized between 0 and 1.} \end{aligned} \tag{B.2}$$

In the following, we present popular methods for generating adversarial examples.

L-BFGS Attack. This method was introduced in [Szegedy *et al.* 2014b]. We name it L-BFGS Attack because it use L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) algorithm to solve for the following problem, in order to generate an adversarial example:

$$\begin{aligned} &\min_{\tilde{x}} c\,\|\tilde{x} - x\|_2^2 + d_{\mathrm{CE}}(f(\tilde{x}), l') \\ \text{subject to: } &\tilde{x} \in [0,1]^n \\ &\operatorname*{argmax}_l f(\tilde{x})[l] = l' \\ \text{where: } &l' \neq \operatorname*{argmax}_l f(x)[l] \text{ and } c > 0 \\ &d_{\mathrm{CE}} \text{ means cross entropy metric.} \end{aligned} \tag{B.3}$$

It is worth noting that, first problem B.3 is not convex because of the nature of neural network model f, hence we obtain a local optima. Secondly, the parameter c needs to be tuned in order to obtain the $\tilde{x}$ that satisfies $\operatorname*{argmax}_l f(\tilde{x})[l] = l'$ and yields a minimum distance from x.

Fast gradient sign method (FGSM). [Goodfellow *et al.* 2014b] proposed to find noise or perturbation η_i for adding into sample x_i as follows:

$$\eta_i = \epsilon \text{sign}(\nabla_x f(x_i)[y_i]), \tag{B.4}$$

where ϵ control the amplitude of noise. This method is fast since it perform only one backward gradient regarding to x_i. In order to make the model f robust to adversarial examples, an obvious way is to train model f with both labelled samples x_i and their adversarial examples $x_i + \eta_i$.

Iterative-Fast gradient sign method (I-FGSM) As its name, this is an iterative version of FGSM proposed by [Kurakin *et al.* 2017] coupled with a clip function:

$$\tilde{x}_i^{t+1} = Clip_{x,\epsilon}\left\{\tilde{x}_i^t + \alpha \text{sign}(\nabla_x f(\tilde{x}_i^t)[y_i])\right\} \tag{B.5}$$

where $\tilde{x}_i^0 = x_i$ and t goes from 0 to n_iters (number of iterations). The clip function is used to guarantee that $\tilde{x}$ is always in a ball of radius of ϵ centered on $\tilde{x}$ and $\tilde{x}$ takes value between 0 and 1. Here is the explicit form of clip function:

$$Clip_{x,\epsilon}\left\{\tilde{x}\right\}(a,b,c) = \min\left\{1, x(a,b,c) + \epsilon, \max\{0, x(a,b,c) - \epsilon, \tilde{x}(a,b,c)\}\right\}, \tag{B.6}$$

where (a, b, c) means for pixel position.

DeepFool [Moosavi-Dezfooli *et al.* 2015] creates an adversarial example of sample x_0 by adding into x_0 a displacement (noise) $\mathbf{r}$. We presented DeepFool in three different settings.

For a binary linear classifier f, where $f(x) = w^\top x + b$, the adversarial example of sample x_0 is created by first projecting x_0 to the hyperplane $f(x) = 0$ and then continuing a little further to change strictly the side (class). In this case, the lowest additional noise or the shortest displacement $\mathbf{r}$ is calculated as follows:

$$\hat{\mathbf{r}}(x_0) = \underset{\|\mathbf{r}\|_2}{\text{argmin}}\ [\text{sign}(f(x_0))\text{sign}(f(x_0 + r)) = -1] \tag{B.7}$$

As a reminder, the distance from a point x_0 to the hyperplane $f(x) = 0$ is calculated by $\frac{|f(x_0)|}{\|w\|_2}$. We denote $P_f(x_0)$ the projection of x_0 on $f(x) = 0$, then $\hat{\mathbf{r}}(x_0)$ is calculated as

$$\hat{\mathbf{r}}(x_0) = P_f(x_0) - x_0 = -\frac{|f(x_0)|}{\|w\|_2}\frac{w}{\|w\|_2}\text{sign}(f(x_0)) = -\frac{f(x_0)}{\|w\|_2^2}w \tag{B.8}$$

To make sure that adversarial sample is strictly on the other side of hyperplane, $\hat{\mathbf{r}}(x_0)$ is multiplied by $(1 + \eta)$ where η is small positive real number.

For a binary non-linear classifier f, an iterative process is performed as depicted in algorithm 8,

Algorithm 8 DeepFool for binary classifiers

```
1: input: Sample $x_0$, classifier $f$.
2: output: Perturbation $\hat{\mathbf{r}}$.
3: Initialize $x_0^0 \leftarrow x_0$ and $i \leftarrow 0$.
4: while $\mathrm{sign}(f(x_0^i)) = \mathrm{sign}(f(x_0))$ do
5:     $\mathbf{r}_i \leftarrow -\frac{f(x_0^i)}{\|\nabla f(x_0^i)\|_2^2} \nabla f(x_0^i)$,
6:     $x_0^{i+1} \leftarrow x_0^i + \mathbf{r}_i$,
7:     $i \leftarrow i + 1$.
8: end while
9: return $\hat{\mathbf{r}} = \sum_i \mathbf{r}_i$.
```

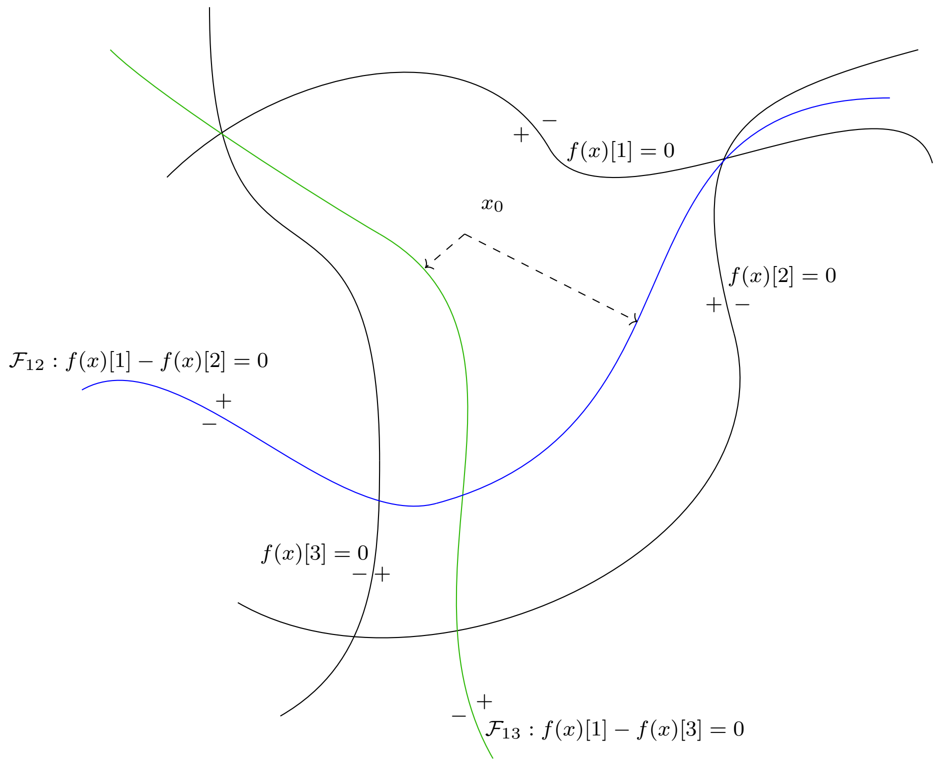
Algorithm 9 DeepFool: multi-class case

```
1: input: Image $x$, classifier $f$.
2: output: Perturbation $\hat{\mathbf{r}}$.
3: Initialize $x_0^0 \leftarrow x_0$ and $i \leftarrow 0$.
4: Set $\hat{k}(a) = \underset{k}{\mathrm{argmax}}\, f(a)[k]$
5: while $\hat{k}(x_0^i) = \hat{k}(x_0)$ do
6:     for $k \neq \hat{k}(x_0)$ do
7:         $w'_k \leftarrow \nabla f(x_0^i)[k] - \nabla f(x_0^i)[\hat{k}(x_0)]$
8:         $f'_k \leftarrow f(x_0^i)[k] - f(x_0^i)[\hat{k}(x_0)]$
9:     end for
10:    $\hat{l} \leftarrow \underset{k \neq \hat{k}(x_0)}{\mathrm{argmin}} \frac{|f'_k|}{\|w'_k\|_2}$
11:    $\mathbf{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$
12:    $x_0^{i+1} \leftarrow x_0^i + \mathbf{r}_i$
13:    $i \leftarrow i + 1$
14: end while
15: return $\hat{\mathbf{r}} = \sum_i \mathbf{r}_i$
```

by replacing w with gradient $\nabla f(x_0^i)$ (line number 5). This algorithm can be seen as Newton's iterative algorithm for finding the roots of a nonlinear function. Since Newton's method depends on the initial guess x_0, theoretically, we may never have a change of $\mathrm{sign}(f(x_0))$ (at line 4). However, in practice, algorithm 8 often converges to a root of f.

For multi-class non-linear classifier f, we explain it through an example with three classes in figure B.2.

Adversarial attacks can be categorized as targeted and untargeted. For example, L-BFGS is targeted attack because the class of adversarial example has to be predefined. On the contrary, FGSM, Deepfool are untargeted attack because adversarial sample $\hat{x}$ is only required to have a different label compared to the one of x. Besides, the robustness of a model to adversarial examples can be assessed using two approaches. On the one hand, in white box attack, the model itself is used to generate adversarial examples. On the contrary, in black box attack, two models are needed, one model of which is used for generating adversarial examples to evaluate robustness of the other model. More methods for generating adversarial examples can be found in two reviews [Yuan *et al.* 2017, Wiyatno *et al.* 2019] and some results about the robustness of model can be found in [Carlini & Wagner 2017]. Besides adversarial examples, adversarial dropout [Park *et al.* 2017] intervenes for model parameters.



$f(x)[1] = 0$
x_0
$f(x)[2] = 0$
$\mathcal{F}_{12} : f(x)[1] - f(x)[2] = 0$
$f(x)[3] = 0$
$\mathcal{F}_{13} : f(x)[1] - f(x)[3] = 0$

Figure B.2: An illustration for Deepfool, for $C = 3$ classes. $f()[1], f()[2]$ and $f()[3]$ are three outputs (before soft-max layer) of classifier. Three decision boundaries (one vs all) $f(x)[1] = 0, f(x)[2] = 0$ and $f(x)[3] = 0$ are represented by black solid lines. The signs $(+)$ and $(-)$ mean positive side and negative side of a decision boundary. $\mathcal{F}_{12}$ (blue line) and $\mathcal{F}_{13}$ (green line) represent respectively the decision boundary (one vs one) $f(x)[1] - f(x)[2] = 0$ and the decision boundary (one vs one) $f(x)[1] - f(x)[3] = 0$. x_0 is assumed to belong to class 1 with $f(x_0)[1] > f(x_0)[2]$ and $f(x_0)[1] > f(x_0)[3]$. For generating adversarial example of x_0, in principle, Deepfool projects x_0 to decision boundaries $\mathcal{F}_{12}$ and $\mathcal{F}_{13}$, then retains the projection of x_0 that gives the shorter distance. However, in practice, since $f()$ is non-linear, finding these projections are not trivial. A workaround, would be to use algorithm 8 (DeepFool for binary classifiers) to find displacement $\hat{\mathbf{r}}(x_0)$ w.r.t each decision boundary (one vs one) $\mathcal{F}_{12}$ and $\mathcal{F}_{13}$ and then use the shortest $\hat{\mathbf{r}}(x_0)$. Algorithm 9 (DeepFool: multi-class case) follows this idea with little twist: at the end of each iteration, x_0^i is updated by the shortest distance $\frac{|f'_k|}{\|w'_k\|_2}$ (line 10 to 12), in order to reduce the computational cost.

3 Hyper-parameters

3.1 Robustness to adversarial examples. Hyper-parameters : optimizer = Momentum (torch.optim.SGD), number of epochs = 300, learning rate = 0.1, momentum = 0.9, learning rate is reduce by $lr = 0.1lr$ after each 75 epochs, batch size = 200, weight decay = 0.0001.

3.2 Semi-supervised manifold attack. Hyper-parameters : optimizer = Adam, number of epochs = 1024, learning rate = 0.002, $\alpha = 0.75$, batch size labelled = batch size unlabelled = 64, T = 0.5 (in sharpening), $\lambda = 75$ (linearly ramp up from 0), EMA = 0.999 , error validation after 1024 batchs.

To reproduce an experiment, we define function seed_ as:

```
def seed_(p):
    """ for reproductive """
    torch.manual_seed(p)
    np.random.seed(p)
    random.seed(p)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(p)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

    return 0
```

The four experiment 1,2,3,4 in table IV.4 are launched with respectively seed_(0), seed_(1), seed_(2), seed_(3).

In Mix-up Attack, n_iters is fixed at 1. In dataset CIFAR-10, ξ starts at 0.1 and decreases linearly to 0.01 after 1024 epochs. In dataset SVHN, ξ starts at 0.1 and decreases linearly to 0.01 after 1024 epochs for seed_(0) and seed_(3); ξ starts at 0.1 and decreases linearly to 0.001 after 1024 epochs for seed_(2); ξ starts at 0.01 and decreases linearly to 0.001 after 1024 epochs for seed_(1).

4 Parallel programming

By introducing attack stages, we analyse the additional cost via execution time. In a batch training (without attack stage), execution time can be decomposed into: *loading time* (loading data into a batch + pre-processing . . .) and *working time* (loss forward calculating + gradient back propagation). When introducing attack stage, via our experiments, execution time becomes: *loading time* + *working time* × ($n\_iters + 1$). Since *working time* is usually much greater than *loading time* and it is multiplied by ($n\_iters + 1$), we need to consider to parallel programming in case of using large value n_iters. We present in this section some basic strategies that can be applied for almost deep NNMs to reduce the *working time*: **Data Parallel** and **Model Parallel**.

4.1 Data Parallel. We clone the whole model for each device (GPU), then we split the input batch into N_m sub-batches, where N_m is the number of devices. Each sub-batch is treated by a device and the model parameters are updated by aggregating all backward gradients. The algorithm is simple and the memory balance among devices are the advantage of this method. But it is not useful for large models (in term of parameters) because we lose in each device a lot of memory to stock a cloned model. This implies maybe a limit for the size of a sub-batch, which is loaded in each device. Moreover, if devices are not in the same node, it requires a communication protocol between nodes. The multi-process mode is eventually required if we want to customize actions (transfer, receive, wait ...) between devices.

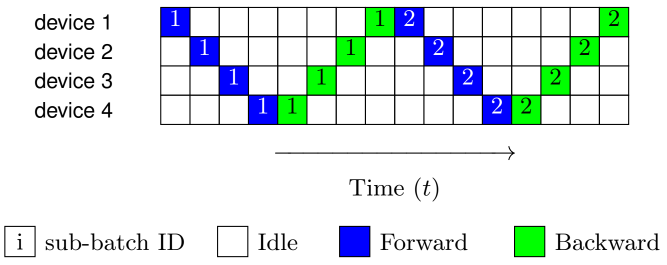


Figure B.3: Naive sequential. Source: [Harlap *et al.* 2018]

4.2 Model Parallel. In this method, we split both the model and the input batch. The model is split into N_m pipelining model parts and the input batch is split into N_{split} sub-batches. Each part of model is associated to a device. By splitting the model, this method can deal with a large model, specially in the case that the required memory for the model is greater than the own memory of each device. Figure B.3 shows a naive sequential for split model training with 4 devices. In forward pass, at $t = 1$, the first sub-batch is loaded and processed in device 1 while other devices are idle. Then at $t = 2$, output of device 1 is passed to device 2 where it is processed while all devices are idle except device 2, and so on. In backward pass, it is exactly the same thing as in forward pass, but we start from device 4. Next, the process continues with the second sub-batch and so on.

Until here, there is nothing parallel and each device is idle in the most of training time. [Harlap

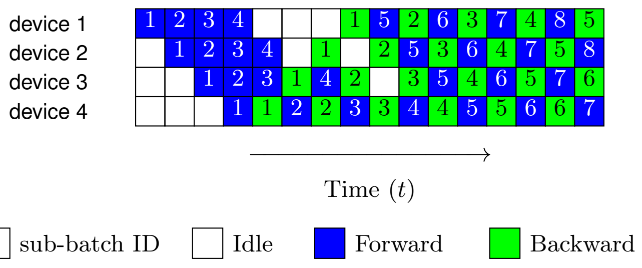


Figure B.4: PipeDream. The stability is established from time = 10, where there is no more idle device. Source: [Harlap *et al.* 2018].

et al. 2018] propose a model parallel method called PipeDream to make use of all devices at the same time (an illustration is showed in figure B.4). Now, sub-batches are loaded in a streaming way. After the startup state until $t = 9$, we pass in the steady stage where all devices are employed. A device performs alternatively between forward and backward pass. In spite of a good working strategy, PipeDream and other methods of Model Parallel must overcome two main issues to attain the optimal configuration: computation balance and memory balance. Since model is split into many parts and these parts are not the similar (different types of layer, different number of layers, different number of parameters...), then computational time and used memory is not the same across devices. No computation balance leads to additional waiting times, maybe all devices need to wait for only one device.

Furthermore, the optimal time for Model Parallel is known as being sensitive sensible to the number of sub-batches N_{split}. Using a small N_{split} leads to many time steps, while using a large N_{split} results intuitively to a long waiting time among devices, because absolute computation balance is impossible. In a short conclusion, user must understand deeply about the used model's architecture to perform pertinent splittings on the model and on a batch.

4.3 Discussion about parallel training. In general, there are two popular types for parallel programming in DeL field: Data Parallel and Model Parallel. For both two types, we need to consider the following issue in parallel training, the *dependencies between samples.* This means that in an approach, there are interactions between samples, *e.g*, the batch normalization layer [Ioffe & Szegedy 2015]. The latter requires mean and variance of all samples in a batch. For Data Parallel, they can be inferred from all sub-batch means and variances, but the process in each device needs to be synchronized in this layer (wait for all arrivals and communicate among them for the batch mean and variance), before continuing with the the next layer. Note that, in a batch normalization layer, the mean and variance need to be stocked in each device to use for backward propagation. On the contrary, for Model Parallel and particularly PipeDream, having mean and variance of all samples in a batch is really impossible since a sub-batch may achieve its mission before the arrival of other one, *e.g.* sub-batches 1 and 5 in figure B.4. If we do not pay attention, the dependencies between samples is one of the causes that imply the difference between the normal training and parallel training.

In conclusion, Data Parallel is much less complicated and faster than Model Parallel. In addition, it can deal with the dependencies between samples. However, Data Parallel is not optimal in term of memory using, because the whole model needs to be cloned in each device. On the other hand, Model Parallel can give a better use of memory, but it is much more complicated and requires an in-depth knowledge about model architecture to attain the optimal configuration (computation balance and memory balance). Inspired by parallel programming for DeL models, we try to propose the same thing for DiL model in A.7.

5 System of index for anchor points

Let consider the following problem for calculating the gradient backward in attack stage, to update the coefficient γ :

$$\nabla_\gamma \mathcal{L}_t\big(\{g(\gamma_1^k \mathbf{z}_1^k + \gamma_2^k \mathbf{z}_2^k + .. + \gamma_p^p \mathbf{z}_p^k) | k = 1, .., M\}\big) \tag{B.9}$$

Then the memory needed to reserve anchor points is equivalent to $M \times p$ samples. If anchor points are sampled from data samples, we denote :

- $\mathbf{X}^b \in \mathbb{R}^{N_b \times K \times H \times W}$, input batch (images), where N_b is number of inputs in a batch, K, H, W are respectively number of channel, height and width.

- $\mathbf{M} \in \mathbb{R}^{N_b \times M}$, index matrix (or mask matrix) for M sets of anchor points. $\mathbf{M}[i,k] = \begin{cases} 1 \text{ if sample } i \text{ of } \mathbf{X}^b \text{ is an anchor point in set } k, \\ 0 \text{ otherwise.} \end{cases}$

- $\Gamma \in \mathbb{R}^{N_b \times M}$, coefficients associated to each input (in a column), and M is the number of sets of anchor points. Γ is initialized as follows : $\Gamma = \mathbb{0}_{N_b \times M}$, $\Gamma[\mathbf{M}[:,k],k] = \gamma^k, k = 1, .., M$.

The system of index for anchor points is performed by *einsum* function and by a mask matrix $\mathbf{M}$. Then all virtual points are inferred by :

$$\text{einsum}(\text{``}N_bKHW, N_bM \to MKHW\text{''}, \mathbf{X}^b, \Gamma) \tag{B.10}$$

Then we update Γ by the corresponding gradient by mask matrix $\mathbf{M}$:

$$\begin{aligned} \Gamma &\leftarrow \Gamma + \nabla_{\Gamma}\mathcal{L}_t(\Gamma) \circ \mathbf{M} \\ \Gamma[\mathbf{M}[:,k],k] &\leftarrow \Pi_{ps}(\Gamma[\mathbf{M}[:,k],k]), k = 1,..,M \end{aligned} \tag{B.11}$$

In short, by using *einsum* function, the memory needed is reduced from $M \times p$ samples equivalent to N_b samples equivalent (two supplement matrices $\Gamma, \mathbf{M} \in \mathbb{R}^{N_b \times M}$ are negligible compared to N_b samples equivalent). Not that, usually M is approximated to N_b then, we reduce p times the memory needed.

Bibliography

[Aharon *et al.* 2006] Aharon, M., Elad, M., and Bruckstein, A. (2006). K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Trans. Sig. Proc.*, 54(11):4311–4322.

[Ahmed *et al.* 1974] Ahmed, N., Natarajan, T., and Rao, K. R. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93.

[Alemi 2016] Alemi, A. (2016). *Improving Inception and Image Classification in TensorFlow.* `https://ai.googleblog.com/2016/08/improving-inception-and-image.html`.

[Ba *et al.* 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

[Babagholami-Mohamadabadi *et al.* 2013] Babagholami-Mohamadabadi, B., Zarghami, A., Zolfaghari, M., and Baghshah, M. S. (2013). Pssdl: Probabilistic semi-supervised dictionary learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 192–207, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Bahdanau *et al.* 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.

[Beck & Teboulle 2009] Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202.

[Belkin & Niyogi 2003] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396.

[Berthelot *et al.* 2019a] Berthelot, D., Carlini, N., Cubuk, E. D., Kurakin, A., Sohn, K., Zhang, H., and Raffel, C. (2019a). Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring.

[Berthelot *et al.* 2019b] Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., and Raffel, C. (2019b). Mixmatch: A holistic approach to semi-supervised learning.

[Blumensath & Davies 2008] Blumensath, T. and Davies, M. E. (2008). Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14(5):629–654.

[Boyd *et al.* 2011] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122.

[Boyd & Vandenberghe 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press.

[Bühlmann 2004] Bühlmann, P. (2004). Bagging, boosting and ensemble methods. Papers 2004,31, Humboldt University of Berlin, Center for Applied Statistics and Economics (CASE).

[Candès 1998] Candès, E. J. (1998). Ridgelets: Theory and applications. In *PhD thesis.*

[Candès & Donoho 2000] Candès, E. J. and Donoho, D. (2000). Curvelets - a surprisingly effective nonadaptive representation for objects with edges. *Curves and Surfaces.*

[Carlini & Wagner 2017] Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks.

[Chapelle *et al.* 2006] Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning (Adaptive Computation and Machine Learning).* The MIT Press.

[Chen & Yang 2017] Chen, L. and Yang, M. (2017). Semi-supervised dictionary learning with label propagation for image classification. *Computational Visual Media*, 3:83–94.

[Chen & Guestrin 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

[Chen *et al.* 2020] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations.

[Chollet 2016] Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions.

[Coates & Ng 2011] Coates, A. and Ng, A. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *ICML.*

[Combettes & Pesquet 2009] Combettes, P. L. and Pesquet, J.-C. (2009). Proximal Splitting Methods in Signal Processing. *ArXiv e-prints.*

[Dean *et al.* 2018] Dean, J., Patterson, D., and Young, C. (2018). A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 38(2):21–29.

[Deng *et al.* 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Doersch *et al.* 2015] Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction.

[Dong *et al.* 2016] Dong, X., Thanou, D., Frossard, P., and Vandergheynst, P. (2016). Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23):6160–6173.

[Dosovitskiy *et al.* 2014] Dosovitskiy, A., Fischer, P., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with exemplar convolutional neural networks.

[Dozat 2016] Dozat, T. (2016). Incorporating nesterov momentum into adam.

[Duchi *et al.* 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159.

[Efron *et al.* 2004] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32:407–499.

[Elad & Aharon 2006] Elad, M. and Aharon, M. (2006). Image denoising via learned dictionaries and sparse representation. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 895–900. IEEE.

[Elhamifar & Vidal 2012] Elhamifar, E. and Vidal, R. (2012). Sparse subspace clustering: Algorithm, theory, and applications. *CoRR*, abs/1203.1005.

[Engan *et al.* 1999] Engan, K., Aase, S. O., and Hakon Husoy, J. (1999). Method of optimal directions for frame design. In *Proceedings of the Acoustics, Speech, and Signal Processing, 1999. On 1999 IEEE International Conference - Volume 05*, ICASSP '99, pages 2443–2446, Washington, DC, USA. IEEE Computer Society.

[Ester *et al.* 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press.

[Fei-Fei & Perona 2005] Fei-Fei, L. and Perona, P. (2005). A bayesian hierarchical model for learning natural scene categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 524–531 vol. 2.

[Figueiredo *et al.* 2007] Figueiredo, M. A. T., Nowak, R. D., and Wright, S. J. (2007). Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):586–597.

[Fukushima 1980] Fukushima, K. (1980). Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193—202.

[Gangeh *et al.* 2015] Gangeh, M. J., Farahat, A. K., Ghodsi, A., and Kamel, M. S. (2015). Supervised dictionary learning and sparse representation-a review. *arXiv preprint arXiv:1502.05928*.

[Gangeh *et al.* 2010] Gangeh, M. J., Sørensen, L., Shaker, S. B., Kamel, M. S., de Bruijne, M., and Loog, M. (2010). A texton-based approach for the classification of lung parenchyma in ct images. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010*, pages 595–602, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Georghiades *et al.* 2001] Georghiades, A., Belhumeur, P., and Kriegman, D. (2001). From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 23(6):643–660.

[Ghiasi *et al.* 2018] Ghiasi, G., Lin, T., and Le, Q. V. (2018). Dropblock: A regularization method for convolutional networks. *CoRR*, abs/1810.12890.

[Gidaris *et al.* 2018] Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations.

[Goodfellow *et al.* 2014a] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA. MIT Press.

[Goodfellow *et al.* 2014b] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples.

[Grandvalet & Bengio 2004] Grandvalet, Y. and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. volume 17.

[Hale *et al.* 2007] Hale, E. T., Yin, W., and Zhang, Y. (2007). A Fixed-Point Continuation Method for l1 -Regularized Minimization with Applications to Compressed Sensing. Technical report, Rice University.

[Harlap *et al.* 2018] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., and Gibbons, P. B. (2018). Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377.

[He *et al.* 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[He *et al.* 2005] He, X., Yan, S., Hu, Y., Niyogi, P., and Zhang, H.-J. (2005). Face recognition using laplacianfaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(3):328–340.

[Hinton & Roweis 2003] Hinton, G. and Roweis, S. (2003). Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840.

[Hinton *et al.*] Hinton, G., Srivastava, N., and Swersky, K. *Lecture 6a Overview of mini-batch gradient descent*. `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[Hjort 1990] Hjort, N. L. (1990). Nonparametric bayes estimators based on beta processes in models for life history data. *Ann. Statist.*, 18(3):1259–1294.

[Hochreiter & Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Howard *et al.* 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[Hu *et al.* 2017] Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2017). Squeeze-and-excitation networks.

[Huang *et al.* 2016] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2016). Densely connected convolutional networks.

[hyun Lee 2013] hyun Lee, D. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

[Ioffe & Szegedy 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Iscen *et al.* 2019] Iscen, A., Tolias, G., Avrithis, Y., and Chum, O. (2019). Label propagation for deep semi-supervised learning.

[Jenatton *et al.* 2010] Jenatton, R., Mairal, J., Obozinski, G., and Bach, F. (2010). Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 487–494, Madison, WI, USA. Omnipress.

[Jianchao Yang *et al.* 2009] Jianchao Yang, Kai Yu, Yihong Gong, and Huang, T. (2009). Linear spatial pyramid matching using sparse coding for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801.

[Jiang *et al.* 2013] Jiang, Z., Lin, Z., and Davis, L. S. (2013). Label consistent k-svd: Learning a discriminative dictionary for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2651–2664.

[Jutten & Herault 1991] Jutten, C. and Herault, J. (1991). Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, 24(1):1–10.

[Ke *et al.* 2019] Ke, Z., Wang, D., Yan, Q., Ren, J., and Lau, R. W. H. (2019). Dual student: Breaking the limits of the teacher in semi-supervised learning.

[Kim *et al.* 2007] Kim, S. J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D. (2007). An interior-point method for large-scale ell_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617.

[Kingma & Ba 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[Kingma *et al.* 2014] Kingma, D. P., Rezende, D. J., Mohamed, S., and Welling, M. (2014). Semi-supervised learning with deep generative models.

[Kingma & Welling 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes.

[Kolesnikov *et al.* 2019] Kolesnikov, A., Zhai, X., and Beyer, L. (2019). Revisiting self-supervised visual representation learning.

[Kong & Wang 2012] Kong, S. and Wang, D. (2012). A dictionary learning approach for classification: Separating the particularity and the commonality. In *Computer Vision – ECCV 2012*, pages 186–199, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Krizhevsky *et al.* 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Krogh & Hertz 1992] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann.

[Kruskal & Wish 1978] Kruskal, J. and Wish, M. (1978). *Multidimensional Scaling.* Sage Publications.

[Kurakin *et al.* 2017] Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world.

[Kurita 2018] Kurita, K. (2018). *An Overview of Normalization Methods in Deep Learning.* `https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/#:~:text=Instance`.

[Laine & Aila 2016] Laine, S. and Aila, T. (2016). Temporal ensembling for semi-supervised learning. *CoRR*, abs/1610.02242.

[Lanusse *et al.* 2017] Lanusse, F., Ma, Q., Li, N., Collett, T. E., Li, C.-L., Ravanbakhsh, S., Mandelbaum, R., and Póczos, B. (2017). Cmu deeplens: deep learning for automatic image-based galaxy–galaxy strong lens finding. *Monthly Notices of the Royal Astronomical Society*, 473(3):3895–3906.

[Lecun *et al.* 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun *et al.* 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[Lee *et al.* 2007] Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19.* MIT Press, Cambridge, MA.

[Long *et al.* 2014] Long, J., Shelhamer, E., and Darrell, T. (2014). Fully convolutional networks for semantic segmentation.

[MacQueen 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics* *and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif. University of California Press.

[Mairal *et al.* 2014] Mairal, J., Bach, F., and Ponce, J. (2014). Sparse modeling for image and vision processing. *Found. Trends. Comput. Graph. Vis.*, 8(2-3):85–283.

[Mairal *et al.* 2009a] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009a). Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 689–696, New York, NY, USA. ACM.

[Mairal *et al.* 2008] Mairal, J., Elad, M., and Sapiro, G. (2008). Sparse representation for color image restoration. *IEEE Transactions on Image Processing*, 17(1):53–69.

[Mairal *et al.* 2009b] Mairal, J., Ponce, J., Sapiro, G., Zisserman, A., and Bach, F. R. (2009b). Supervised dictionary learning. In *Advances in Neural Information Processing Systems 21*, pages 1033–1040. Curran Associates, Inc.

[Mallat 2008] Mallat, S. (2008). *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, Inc., USA, 3rd edition.

[Mallat & Zhang 1993] Mallat, S. G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, 41:3397–3415.

[Martinez & Benavente 1998] Martinez, A. M. and Benavente, R. (1998). The ar face database. *CVC Technical Report No. 24*.

[Matiz & Barner 2016] Matiz, S. and Barner, K. E. (2016). Label consistent recursive least squares dictionary learning for image classification. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1888–1892.

[Meyer 1993] Meyer, Y. (1993). *Wavelets and Operators*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press.

[Milletari *et al.* 2016] Milletari, F., Navab, N., and Ahmadi, S.-A. (2016). V-net: Fully convolutional neural networks for volumetric medical image segmentation.

[Miyato *et al.* 2017] Miyato, T., ichi Maeda, S., Koyama, M., and Ishii, S. (2017). Virtual adversarial training: A regularization method for supervised and semi-supervised learning.

[Moosavi-Dezfooli *et al.* 2015] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2015). Deepfool: a simple and accurate method to fool deep neural networks.

[Morgan & Bourlard 1990] Morgan, N. and Bourlard, H. (1990). Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in Neural Information Processing Systems 2*, pages 630–637. Morgan-Kaufmann.

[Nair *et al.* 2019] Nair, V., Alonso, J. F., and Beltramelli, T. (2019). Realmix: Towards realistic semi-supervised deep learning algorithms.

[Nair & Hinton 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

[Needell & Vershynin 2007] Needell, D. and Vershynin, R. (2007). Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit.

[Nesterov 1983] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o$(1/k^2)$.

[Ngolè Mboula *et al.* 2014] Ngolè Mboula, F. M., Starck, J. L., Ronayette, S., Okumura, K., and Amiaux, J. (2014). Super-resolution method using sparse regularization for point-spread function recovery. *CoRR*, abs/1410.7679.

[Noroozi & Favaro 2016] Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles.

[Oliver *et al.* 2018] Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., and Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms.

[Olshausen & Field 1996] Olshausen, B. and Field, D. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–9.

[Ouali *et al.* 2020] Ouali, Y., Hudelot, C., and Tami, M. (2020). An overview of deep semi-supervised learning.

[Oudeyer *et al.* 2007] Oudeyer, P., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

[Park *et al.* 2017] Park, S., Park, J.-K., Shin, S.-J., and Moon, I.-C. (2017). Adversarial dropout for supervised and semi-supervised learning.

[Pati *et al.* 1993] Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *in Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 1–3.

[Pearson 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Pereyra *et al.* 2017] Pereyra, G., Tucker, G., Chorowski, J., Łukasz Kaiser, and Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions.

[Pham & Venkatesh 2008] Pham, D. and Venkatesh, S. (2008). Joint learning and dictionary construction for pattern recognition. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[Prechelt 1997] Prechelt, L. (1997). Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, 11:761–767.

[Qian 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1):145–151.

[Radford *et al.* 2016] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

[Ramirez *et al.* 2010] Ramirez, I., Sprechmann, P., and Sapiro, G. (2010). Classification and clustering via dictionary learning with structured incoherence and shared features. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR)*, volume 00, pages 3501–3508.

[Rasmus *et al.* 2015] Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder networks.

[Ratner *et al.* 2019] Ratner, A., Varma, P., Hancock, B., Ré, C., and other members of Hazy Lab (2019). *Weak Supervision: A New Programming Paradigm for Machine Learning.* `http://ai.stanford.edu/blog/weak-supervision/`.

[Reddi *et al.* 2019] Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond.

[Reinhold 2019] Reinhold, J. (2019). *Dropout on convolutional layers is weird.* `https://towardsdatascience.com/dropout-on-convolutional-layers-is-weird-5c6ab14f19b2`.

[Robert *et al.* 2018] Robert, T., Thome, N., and Cord, M. (2018). Hybridnet: Classification and reconstruction cooperation for semi-supervised learning.

[Ronneberger *et al.* 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

[Roweis & Saul 2000] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326.

[Rubinstein *et al.* 2013] Rubinstein, R., Peleg, T., and Elad, M. (2013). Analysis k-svd: A dictionary-learning algorithm for the analysis sparse model. *IEEE Transactions on Signal Processing*, 61(3):661–677.

[Ruder 2017] Ruder, S. (2017). An overview of gradient descent optimization algorithms.

[Salimans & Kingma 2016] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks.

[Sandler *et al.* 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks.

[Sangdoo *et al.* 2019] Sangdoo, Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899.

[Santurkar *et al.* 2019] Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2019). How does batch normalization help optimization?

[Schapire 1999] Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, page 1401–1406, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Shorten & Khoshgoftaar 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48.

[Shrivastava *et al.* 2012] Shrivastava, A., Pillai, J. K., Patel, V. M., and Chellappa, R. (2012). Learning discriminative dictionaries with partially labeled data. In *2012 19th IEEE International Conference on Image Processing*, pages 3113–3116.

[Simonyan & Zisserman 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[Skretting & Engan 2010] Skretting, K. and Engan, K. (2010). Recursive least squares dictionary learning algorithm. *IEEE Transactions on Signal Processing*, 58(4):2121–2130.

[Sohn *et al.* 2020] Sohn, K., Berthelot, D., Li, C.-L., Zhang, Z., Carlini, N., Cubuk, E. D., Kurakin, A., Zhang, H., and Raffel, C. (2020). Fixmatch: Simplifying semi-supervised learning with consistency and confidence.

[Springenberg 2016] Springenberg, J. T. (2016). Unsupervised and semi-supervised learning with categorical generative adversarial networks.

[Srivastava *et al.* 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Starck *et al.* 2007] Starck, J., Fadili, J., and Murtagh, F. (2007). The undecimated wavelet decomposition and its reconstruction. *IEEE Transactions on Image Processing*, 16(2):297–309.

[Starck *et al.* 2002] Starck, J.-L., Candès, E. J., and Donoho, D. L. (2002). The curvelet transform for image denoising. *IEEE Transactions on Image Processing*, 11(6):670–684.

[Stutz 2016] Stutz, D. (2016). *Tikz CNN*. `https://github.com/davidstutz/latex-resources/blob/master/tikz-cnn/cnn.tex`.

[Sugiyama 2007] Sugiyama, M. (2007). Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *J. Mach. Learn. Res.*, 8:1027–1061.

[Szegedy *et al.* 2016] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning.

[Szegedy *et al.* 2014a] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014a). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[Szegedy *et al.* 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision.

[Szegedy *et al.* 2014b] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014b). Intriguing properties of neural networks.

[Tan *et al.* 2018] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2018). Mnasnet: Platform-aware neural architecture search for mobile.

[Tan & Le 2019] Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks.

[Tang *et al.* 2019] Tang, W., Panahi, A., Krim, H., and Dai, L. (2019). Analysis dictionary learning based classification: Structure for robustness. *IEEE Transactions on Image Processing*, 28(12):6035–6046.

[Tarvainen & Valpola 2017] Tarvainen, A. and Valpola, H. (2017). Weight-averaged consistency targets improve semi-supervised deep learning results. *CoRR*, abs/1703.01780.

[Tibshirani 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

[Tran *et al.* 2018] Tran, N.-T., Bui, T.-A., and Cheung, N.-M. (2018). Dist-gan: An improved gan using distance constraints.

[Tran 2019] Tran, P. V. (2019). Exploring self-supervised regularization for supervised and semi-supervised learning.

[Ulyanov *et al.* 2016] Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. (2016). Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022.

[Valpola 2014] Valpola, H. (2014). From neural pca to deep unsupervised learning.

[van der Maaten & Hinton 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9(nov):2579–2605. Pagination: 27.

[Vapnik *et al.* 1992] Vapnik, V. N., Boser, B. E., and Guyon, I. M. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA. Association for Computing Machinery.

[Veličković 2017] Veličković, P. (2017). *Unsupervised methods - Diving deep into autoencoders.* `https://petar-v.com/talks/UCLSlides.pdf`.

[Verma *et al.* 2019a] Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Courville, A., Lopez-Paz, D., and Bengio, Y. (2019a). Manifold mixup: Better representations by interpolating hidden states.

[Verma *et al.* 2019b] Verma, V., Lamb, A., Kannala, J., Bengio, Y., and Lopez-Paz, D. (2019b). Interpolation consistency training for semi-supervised learning.

[Wang *et al.* 2014] Wang, D., Nie, F., and Huang, H. (2014). Large-scale adaptive semi-supervised learning via unified inductive and transductive model. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 482–491, New York, NY, USA. ACM.

[Wang *et al.* 2016] Wang, D., Zhang, X., Fan, M., and Ye, X. (2016). Semi-supervised dictionary learning via structural sparse preserving. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2137–2144. AAAI Press.

[Wang *et al.* 2013] Wang, H., Nie, F., Cai, W., and Huang, H. (2013). Semi-supervised robust dictionary learning via efficient l-norms minimization. In *2013 IEEE International Conference on Computer Vision*, pages 1145–1152.

[Wang *et al.* 2015] Wang, X., Guo, X., and Li, S. Z. (2015). Adaptively unified semi-supervised dictionary learning with active points. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1787–1795.

[Wang *et al.* 2019] Wang, X., Kihara, D., Luo, J., and Qi, G.-J. (2019). Enaet: Self-trained ensemble autoencoding transformations for semi-supervised learning.

[Weng 2019] Weng, L. (2019). *Self-Supervised Representation Learning*. `https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html`.

[Weston & Ratle 2008] Weston, J. and Ratle, F. (2008). Deep learning via semi-supervised embedding. In *International Conference on Machine Learning*.

[Wiyatno *et al.* 2019] Wiyatno, R. R., Xu, A., Dia, O., and de Berker, A. (2019). Adversarial examples in modern machine learning: A review.

[Wright *et al.* 2009] Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., and Ma, Y. (2009). Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227.

[Wright 2015] Wright, S. J. (2015). Coordinate descent algorithms.

[Wu & He 2018] Wu, Y. and He, K. (2018). Group normalization. *CoRR*, abs/1803.08494.

[Yang *et al.* 2010a] Yang, J., Wright, J., Huang, T. S., and Ma, Y. (2010a). Image super-resolution via sparse representation. *Trans. Img. Proc.*, 19(11):2861–2873.

[Yang *et al.* 2014] Yang, M., Zhang, L., Feng, X., and Zhang, D. (2014). Sparse representation based fisher discrimination dictionary learning for image classification. *Int. J. Comput. Vision*, 109(3):209–232.

[Yang *et al.* 2010b] Yang, M., Zhang, L., Yang, J., and Zhang, D. (2010b). Metaface learning for sparse representation based face recognition. In *2010 IEEE International Conference on Image Processing*, pages 1601–1604.

[Yankelevsky & Elad 2017] Yankelevsky, Y. and Elad, M. (2017). Structure-aware classification using supervised dictionary learning. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4421–4425.

[Yuan *et al.* 2017] Yuan, X., He, P., Zhu, Q., and Li, X. (2017). Adversarial examples: Attacks and defenses for deep learning.

[Yui 2019] Yui (2019). *Pytorch Implementation for Mix Match.* imikushana@gmail.com. `https://github.com/YU1ut/MixMatch-pytorch`.

[Zagoruyko & Komodakis 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks.

[Zeiler 2012] Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method.

[Zhai *et al.* 2019] Zhai, X., Oliver, A., Kolesnikov, A., and Beyer, L. (2019). S4l: Self-supervised semi-supervised learning.

[Zhang *et al.* 2013] Zhang, G., Jiang, Z., and Davis, L. S. (2013). Online semi-supervised discriminative dictionary learning for sparse representation. In *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part I*, ACCV'12, pages 259–273, Berlin, Heidelberg. Springer-Verlag.

[Zhang *et al.* 2017] Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412.

[Zhang & Li 2010] Zhang, Q. and Li, B. (2010). Discriminative k-svd for dictionary learning in face recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2691–2698.

[Zhang *et al.* 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization.

[Zhang *et al.* 2015] Zhang, Z., Xu, Y., Yang, J., Li, X., and Zhang, D. (2015). A survey of sparse representation: Algorithms and applications. *IEEE Access*, 3:490–530.

[Zheng *et al.* 2011] Zheng, M., Bu, J., Chen, C., Wang, C., Zhang, L., Qiu, G., and Cai, D. (2011). Graph regularized sparse coding for image representation. *IEEE Transactions on Image Processing*, 20(5):1327–1336.

[Zhili Wu *et al.* 2006] Zhili Wu, Chun-hung Li, Ji Zhu, and Jian Huang (2006). A semi-supervised svm for manifold learning. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 490–493.

[Zhou *et al.* 2004] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004). Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328. MIT Press.

[Zhou *et al.* 2019] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2019). Graph neural networks: A review of methods and applications.

[Zhou *et al.* 2009] Zhou, M., Chen, H., Ren, L., Sapiro, G., Carin, L., and Paisley, J. W. (2009). Non-parametric bayesian dictionary learning for sparse image representations. In *Advances in Neural Information Processing Systems 22*, pages 2295–2303. Curran Associates, Inc.

[Zhu & Ghahramani 2002] Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Technical Report.* CMU-CALD-02–107.

[Zisserman 2018] Zisserman, A. (2018). *Self-Supervised Learning.* `https://project.inria.fr/paiss/files/2018/07/zisserman-self-supervised.pdf`.

Titre: Apprentissage semi-supervisé de dictionnaire et de réseaux de neurones profonds

Mots clés: Apprentissage de dictionnaire, Apprentissage de variété, Apprentissage profond, Apprentissage antagoniste

Résumé: Notre travail est divisé en deux parties principales: l'apprentissage de dictionnaire et l'apprentissage profond. Dans un premier temps, nous proposons une nouvelle méthode d'apprentissage de dictionnaire semi-supervisée basée sur deux axes: d'une part, nous renforçons la préservation de l'organisation originale des données dans l'espace de représentation parcimonieuse associé au dictionnaire ; d'autre part, nous entraînons un classifieur semi-supervisé dans l'espace de représentation parcimonieuse. Cette méthode apporte une amélioration sur le taux de précision pour des jeux de données de petites tailles. Afin de traiter des données dont les échantillons sont de grandes dimension, dans un deuxième temps, nous utilisons le modèle d'apprentissage profond. Nous renforçons la préservation de l'organisation originale des données dans un espace latent induit par une couche interne du réseau de neurones. Nous inspirons pour cela de l'apprentissage antagoniste : nous déterminons des points virtuels pour lesquels l'organisation originale des données est la moins préservée dans l'espace laten, et exploitons ces derniers dans l'entraînement du modèle. Cette approche apporte une amélioration sur le taux de précision et aussi sur la robustesse contre des exemples antagonistes.

Title: Semi-supervised dictionary learning and Semi-supervised deep neural network

Keywords: Dictionary learning, Manifold learning, Deep learning, Adversarial learning

Abstract: Our work is divided into two main parts: dictionary learning and deep learning. First, we propose a new semi-supervised dictionary learning method based on two axes: on the one hand, we strengthen the preservation of the original data organization in the sparse representation space associated with the dictionary; on the other hand, we train a semi-supervised classifier in the sparse representation space. This method brings an improvement on the accuracy rate for small datasets. In order to process data with high dimensionality, in a second step, we use the deep learning model. We reinforce the preservation of the original organization of the data in a latent space induced by a hidden layer of the neural network. For this, we draw inspiration from adversarial learning: we determine virtual points for which the original data organization is least preserved in the latent space, and exploit these in the training of the model. This approach improves the accuracy rate and also the robustness against adversarial examples.