# Pondération dynamique dans un cadre multi-tâche pour réseaux de neurones profonds

Soufiane Belharbi[1] Romain Hérault[1] Clément Chatelain[1] Sébastien Adam[2]

[1] INSA de Rouen - LITIS EA 4108 Saint Étienne du Rouvray 76800 - France
[2] Université de Rouen, UFR des Sciences - LITIS EA 4108 Saint Étienne du Rouvray 76800 - France

soufiane.belharbi@insa-rouen.fr

## Résumé

*La quantité de données non-étiquetées que nous générons ne cesse de croître. Elles constituent un type de données moins coûteux à obtenir que les données étiquetées, ce qui a motivé son exploitation. Elles ont notamment été utilisées avec succès par les modèles de type réseaux de neurones profonds grâce au concept du pré-apprentissage, ainsi que dans les méthodes d'apprentissage semi-supervisé. Cependant, ces approches présentent des limitations. Le concept de pré-apprentissage souffre du nombre important d'hyper-paramètres à régler, alors que l'apprentissage semi supervisé comporte certaine restrictions. Nous proposons dans ce travail un cadre permettant de résoudre ces problèmes en combinant les deux approches. Notre méthode est basée sur un cadre d'apprentissage multi-tâche avec une tâche principale supervisée et une tâche secondaire non-supervisée, chacune étant pondérée selon son importance. Pour cela, nous proposons de faire évoluer les poids de ces tâches au cours de l'apprentissage selon différents schémas. Des expérimentations sur la base MNIST ont montré des résultats intéressants.*

## Mots Clef

Apprentissage mutli-tâche, réseaux de neurones profonds, apprentissage semi-supervisé

## Abstract

*In the last years, a large amount of unlabeled data has emerged. An easy and cheap data to collect in the opposite of labeled data which motivates their use. Deep neural networks is one of the models that has benefited from this data by using the pre-training approach or other adapted methods from the semi-supervised learning field with success. However, these methods still present some limitations such as the large number of hyper-parameters to tune and the restriction of the semi-supervised learning approaches. We present in this work a framework that combines both approaches to solve their limitations. Our approach is a multi-task learning framework with a main supervised task and a secondary unsupervised task. We propose to weight the importance of each task. We evolve the importance weights through the learning epochs using different schemes. Extensive experiments on MNIST showed interesting results.*

## Keywords

Multi-task learning, deep neural networks, semi-supervised learning

# 1 Introduction

Dans de nombreux domaines d'application de l'apprentissage, l'obtention de données non-étiquetées est plus rapide, plus facile et moins coûteuse que l'acquisition de données étiquetées. On dispose alors de ces deux types de données (étiquetées et non-étiquetées) que l'on souhaite utiliser conjointement pendant l'apprentissage. L'objectif de cette combinaison est l'amélioration de la généralisation. Ce problème est connu sous le nom d'apprentissage semi-supervisé [4, 5, 16, 1].

Grâce à leurs très bonnes performances, les réseaux de neurones profonds (RNP) jouissent d'une popularité importante dans le domaine de l'apprentissage statistique. Principalement conçus pour l'apprentissage supervisé, différentes adaptations ont été proposées pour l'apprentissage semi-supervisé avec de nombreux succès, mais aussi des limitations. Le but de ce travail est de proposer un nouveau cadre pour l'apprentissage semi-supervisé de RNP améliorant les méthodes existantes.

Une première solution a été proposée dans [8, 7, 2, 11] sous le nom du pré-apprentissage. Motivée par l'idée de réaliser plusieurs tâches au lieu d'une seule au sein d'une couche du RNP, une telle solution se rapproche beaucoup d'un transfert séquentiel d'apprentissage, dans lequel deux tâches d'apprentissage séparées sont réalisées successivement. Un premier apprentissage non-supervisé exploite uniquement les données d'entrée $\mathbf{x}$. Les exemples non-étiquetés peuvent donc être utilisés. Un second apprentissage supervisé classique est ensuite effectué uniquement sur les exemples étiquetés $(\mathbf{x}, \mathbf{y})$.

L'apprentissage non-supervisé, qui se fait couche après

couche, consiste à reconstruire l'entrée de chaque couche en utilisant un auto-encodeur [2]. Cet auto-encodeur partage une partie de ses paramètres avec la couche en question du RNP. L'intérêt du premier apprentissage (non-supervisé) est de fournir une initialisation des paramètres du réseau qui soit meilleure qu'une initialisation aléatoire. Cette approche a ouvert, de nouveau, les portes de succès pour les RNP. Cependant, deux inconvénients majeurs se présentent dans cette approche. Le premier est la difficulté à contrôler le sur-apprentissage qui apparaît lors de la tâche non-supervisée et qui, par conséquent nuit aux paramètres qui seront utilisés plus tard dans la tâche supervisée. Le deuxième inconvénient réside dans le nombre conséquent des hyper-paramètres à optimiser [14].

Une autre solution [14] a été proposée dans le cadre particulier de la classification semi-supervisée basée sur les réseaux de neurones profonds avec une stratégie d'apprentissage multi-tâche (AMT). L'idée de ce travail est de classer les données non-étiquetées en s'appuyant sur les données étiquetées. La classification est basée sur le concept de l'hypothèse de variété (*manifold assumption*) qui considère que les exemples qui sont dans la même structure ont tendance à avoir la même classe. Les auteurs proposent de coder une tâche auxiliaire dans chaque couche basée sur la préservation de similarité et l'hypothèse de variété. Un avantage majeur de cette méthode est le fait que les deux tâches supervisée et non-supervisée sont optimisées en parallèle.

Ce travail a donné de bons résultats. Cependant, il présente de nombreuses limitations. Mentionnons par exemple la nécessité d'estimer une large matrice de similarité composée des distances entre chaque paire d'exemples, ce qui exige de mesurer leurs distances avec la méthode de K-plus-proches-voisins ; cela rend la méthode difficile à étendre pour un grand ensemble de données. De plus, quand un nouvel exemple arrive, il faut ré-apprendre le modèle pour estimer sa similarité aux autres exemples. Pour finir, l'apprentissage se fait en mode en-ligne (un exemple à la fois) pour estimer la similarité de chaque paire, ce qui rend l'apprentissage lent.

Nous proposons dans ce travail un cadre permettant de résoudre ces problèmes en combinant ces deux travaux. Notre approche consiste à utiliser un cadre d'apprentissage multi-tâche similaire à [14] avec une tâche supervisée et une tâche non-supervisée. La tâche non-supervisée est une tâche de reconstruction codée au niveau de chaque couche basée sur l'auto-encodeur [2]. En outre, nous proposons d'équilibrer l'importance des deux tâches en utilisant des poids dynamiques qui évoluent au cours de l'apprentissage. Ainsi, le schéma du pré-apprentissage traditionnel est un cas particulier de notre cadre lorsque les poids sont fixés d'une certaine manière. Nous utilisons, dans cet article, un type particulier d'auto-encodeur qui est l'auto-encodeur débruitant [13], basé sur une idée de corruption de l'entrée [9, 12].

Notre méthode permet d'exploiter les données non-étiquetées avec très peu d'hyper-paramètres à régler, tout en permettant un transfert parallèle d'apprentissage qui évite d'endommager les paramètres partagés. Notre méthode peut ainsi réaliser les tâches classiques d'un RNP : la classification et la régression en l'occurrence. Par ailleurs, l'apprentissage peut être fait en mini-batch comme online. Nous avons évalué notre méthode sur la base MNIST pour la tâche de classification. Signalons que l'objectif de ce travail n'est pas de dépasser l'état-de-l'art sur MNIST, mais de montrer que nous pouvons facilement incorporer les données non-étiquetées dans l'apprentissage des RNP avec peu d'hyper-paramètres. Nos expériences montrent qu'une meilleure généralisation et qu'une convergence plus rapide de l'apprentissage des RNP peuvent être obtenues.

Le reste du papier est divisé en 4 sections. Dans la première, nous présenterons notre approche. Ensuite, nous détaillerons sa mise en œuvre, dans laquelle nous expliquerons en détail la tâche de reconstruction, l'évolution des poids dynamiques et l'optimisation du modèle. Nous enchaînerons avec une section d'expérimentations. La dernière section est dédiée à la conclusion et aux perspectives de ce travail.

# 2 Modèle proposé

Nous formalisons notre approche comme un problème d'apprentissage multi-tâche (AMT) [3] avec une tâche principale notée $\mathcal{M}(.)$ et une tâche secondaire notée $\mathcal{R}(.)$. Soit un ensemble d'apprentissage

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_l, y_l), (x_{l+1}, —), \ldots, (x_{l+u}, —)\} \ ,$$

tel que les $l$ premiers exemples sont étiquetés et les $u$ derniers exemples ne le sont pas.

$\mathcal{M}$ et $\mathcal{R}$ sont respectivement les fonctions de prédiction de la tâche principale et de la tâche secondaire ; $\mathcal{C}_s$ et $\mathcal{C}_r$ sont les fonctions de coût liées à ces deux tâches. Les deux tâches partagent un ensemble de paramètres $\mathbf{w}_p$.

La tâche principale est une tâche supervisée paramétrée par $\mathbf{w} = \{\mathbf{w}_p, \mathbf{w}_s\}$ où $\mathbf{w}_s$ est un ensemble de paramètres propre à la tâche principale. Son critère d'apprentissage est défini par $\mathcal{J}_s$,

$$\mathcal{J}_s(\mathcal{D}; \mathbf{w} = \{\mathbf{w}_p, \mathbf{w}_s\}) = \sum_{i=1}^{l} \mathcal{C}_s(\mathcal{M}(x_i; \mathbf{w}), y_i). \quad (1)$$

La tâche secondaire est une tâche de reconstruction paramétrée par $\mathbf{w}' = \{\mathbf{w}_p, \mathbf{w}_r\}$ où $\mathbf{w}_r$ est un ensemble de paramètres propre à la tâche de reconstruction. Son critère d'apprentissage est défini par $\mathcal{J}_r$,

$$\mathcal{J}_r(\mathcal{D}; \mathbf{w}' = \{\mathbf{w}_p, \mathbf{w}_r\}) = \sum_{i=1}^{l+u} \mathcal{C}_r(\mathcal{R}(x_i; \mathbf{w}'), x_i). \quad (2)$$

En combinant les deux tâches au sein du même cadre, nous espérons que la tâche secondaire améliore la tâche principale. Nous pondérons l'importance des deux tâches au

cours de l'apprentissage via deux poids d'importance $\lambda_s$ pour la tâche principale et $\lambda_r$ pour la tâche secondaire.
Ainsi, la fonction objective totale de notre modèle peut être ré-écrite comme suit,

$$\begin{aligned}\mathcal{J}(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_s,\mathbf{w}_r\}) &= \lambda_s \cdot \mathcal{J}_s(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_s\}) \\ &+\lambda_r \cdot \mathcal{J}_r(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_r\}). \quad (3)\end{aligned}$$

Dans cet article, nous proposons de faire évoluer les valeurs de $\lambda_s$ et $\lambda_r$ au cours des itérations d'apprentissage. L'objectif d'une telle stratégie est de donner une plus grande importance à la tâche secondaire dans les premières époques d'apprentissage, en gardant la tâche principale présente pour éviter de trop détériorer les paramètres partagés $\mathbf{w}_p$. L'amélioration de la tâche principale étant le but final, son poids est augmenté au cours des époques d'apprentissage et celui de la tâche secondaire est diminué (Fig.1).
Ainsi, la fonction objective totale (Eq.3) s'exprime en fonction de $t$ par :

$$\begin{aligned}\mathcal{J}(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_s,\mathbf{w}_r\}) &= \lambda_s(t) \cdot \mathcal{J}_s(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_s\}) \\ &+\lambda_r(t) \cdot \mathcal{J}_r(\mathcal{D};\{\mathbf{w}_p,\mathbf{w}_r\}), \quad (4)\end{aligned}$$

où $t \geq 0$ indique les époques d'apprentissage. Dans l'optique d'une comparaison équitable entre les différents modèles utilisés dans nos expériences, nous contraignons les valeurs des poids de telle sorte que $\forall t \geq 0$ :

$$\left\{\begin{array}{rcl} 0 \leq \lambda_s(t) & \leq & 1 \\ 0 \leq \lambda_r(t) & \leq & 1 \\ \lambda_s(t)+\lambda_r(t) & = & 1 \end{array}\right.$$

Ces contraintes ne sont pas indispensables dans le cadre théorique de l'AMT [3, 15] mais assurent que les différences de performances entre les différents modèles ne sont pas dues à une augmentation artificielle du pas d'apprentissage par le produit par $\lambda_\cdot(t)$ dans l'équation 4.

# 3 Détails de mise en œuvre

## 3.1 La tâche de reconstruction

Dans la pratique, la tâche principale est réalisée par un réseau de neurones $RNP$ avec $K$ couches cachées. La tâche secondaire est modélisée par un ensemble de $K$ fonctions de reconstruction réalisées par autant d'auto-encodeurs débruitant (AED). Nous rappelons qu'un AED est un réseau de neurones à deux couches dans lequel une couche de codage est suivie par une couche de décodage. Le $k^{ème}$ AED possède un ensemble de paramètres $\mathbf{w}_{a,k} = \{\mathbf{w}_{c,k}, \mathbf{w}_{d,k}\}$ où $\mathbf{w}_{c,k}$ et $\mathbf{w}_{d,k}$ sont respectivement les paramètres de la couche de codage et de la couche de décodage et une fonction $f$ (une fonction binomiale, par exemple) qui est utilisée pour bruiter l'entrée. Tous les paramètres $\mathbf{w}_{c,k},\ 1 \leq k \leq K$ sont partagés avec la tâche supervisée, ce qui permet un transfert d'apprentissage parallèle. On a ainsi $\mathbf{w}_r = \{\mathbf{w}_{a,1},\ldots,\mathbf{w}_{a,K}\}$. Chaque auto-encodeur possède sa propre fonction de coût $\mathcal{C}_{a,k}$ pour que plusieurs types d'auto-encodeur puissent être appliqués. Ainsi, le critère d'apprentissage de la second tâche (Eq.2) s'écrit de la manière suivante,

$$\mathcal{J}_r(\mathcal{D};\mathbf{w}_r) = \sum_{i=1}^{l+u}\sum_{k=1}^{K} \mathcal{C}_{a,k}(\mathcal{R}(f(x_{i,k-1});\mathbf{w}_{a,k}),x_{i,k-1}), \quad (5)$$

où $x_{*,k}$ est la représentation au sein de la $k^{ème}$ couche du $RNP$ ($x_{*,0}$ est l'entrée originale). Dans la suite, nous considérons que toutes les fonctions de reconstruction sont associées au même poids $\lambda_r(t)$, même si notre proposition n'exige pas une telle configuration et permet l'utilisation de poids différents.

## 3.2 Évolution des poids d'importance au cours de l'apprentissage

Nous comparons dans cet article quatre stratégies d'évolution des poids d'importance au cours des époques d'apprentissage (Fig.1) :

- ***Schéma escalier*** : c'est la stratégie traditionnelle du pré-apprentissage de réseaux de neurones profonds. On a ainsi $\lambda_s(t) = 0$ et $\lambda_r(t) = 1$ avant l'époque $t_1$ et $\lambda_s(t) = 1$ et $\lambda_r(t) = 0$ après l'époque $t_1$. Cette stratégie est notée $escalier_{t_1}$.
- ***Schéma linéaire*** : les poids évoluent linéairement à partir de l'époque 0 jusqu'à la dernière époque d'apprentissage. Ce cas est référencé par $lin$.
- ***Schéma linéaire abrégé*** : La tendance linéaire peut être arrêtée à l'époque $t_1$, ensuite $\lambda_s(t)$ et $\lambda_r(t)$ sont saturés à 1 et 0, respectivement. Ce cas est référencé par $lin_{t_1}$.
- ***Schéma exponentiel*** : les poids évoluent de manière exponentielle proportionnellement à $exp^{\frac{-t}{\sigma}}$ où $t$ est le nombre d'époques et $\sigma$ désigne la pente. Ce cas est référencé par $exp_\sigma$.

## 3.3 Optimisation

Nous proposons d'utiliser la méthode de descente de gradient (DG) pour minimiser l'équation 4. Généralement, dans le cas où il y a plusieurs tâches qui se présentent au sein d'une seule fonction objectif, une alternance entre les tâches donne de bons résultats [3, 14, 6, 15]. La méthode d'optimisation est illustrée dans l'algorithme 1.

**Algorithm 1** Une époque d'apprentissage de notre modèle

1: $\mathcal{D}$ ensemble d'apprentissage *mélangé*. $B$ un mini-batch. $\mathbf{w} = \{\mathbf{w}_p, \mathbf{w}_s\}$, $\mathbf{w}' = \{\mathbf{w}_p, \mathbf{w}_r\}$.
2: **for** $B$ in $\mathcal{D}$ **do**
3: $\quad B_s \Leftarrow$ exemples étiquetés de $B$,
4: $\quad$ Faire un pas de gradient vers $\mathcal{J}_r$ en utilisant $B$ (mettre à jour $\mathbf{w}'$)
5: $\quad$ Faire un pas de gradient vers $\mathcal{J}_s$ en utilisant $B_s$ (mettre à jour $\mathbf{w}$)
6: **end for**

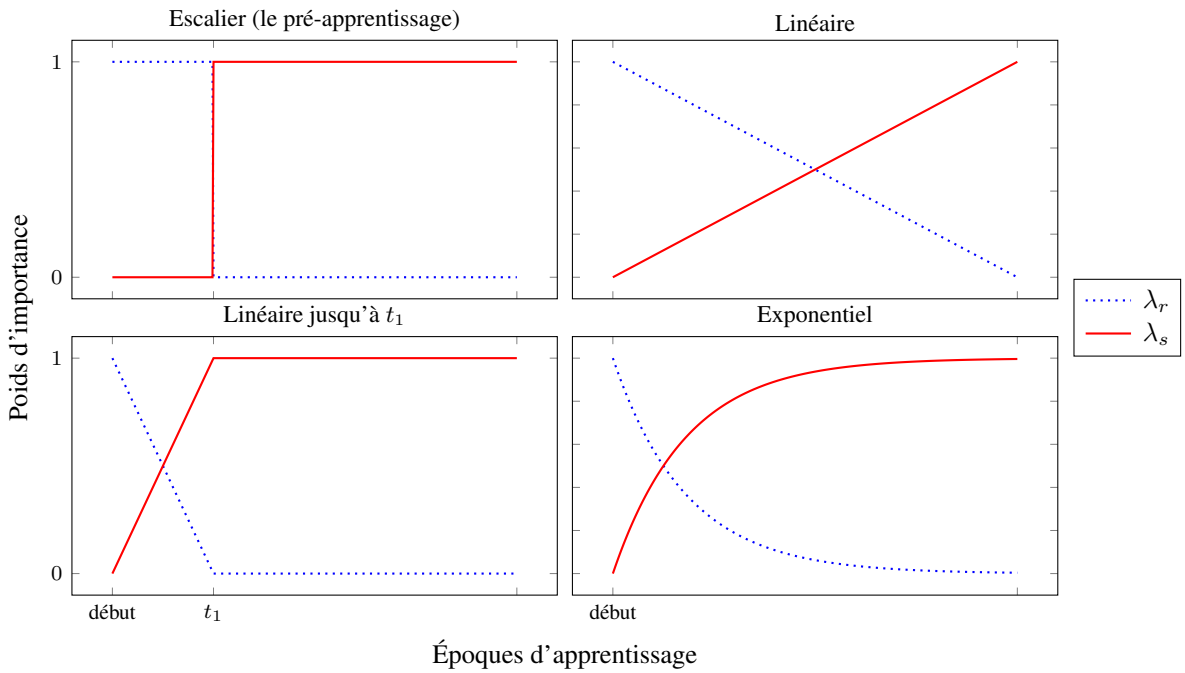


FIGURE 1 – Schémas d'évolution des poids d'importance des tâches au cours des époques d'apprentissage

# 4 Expériences

Nous évaluons notre approche dans le cadre d'un problème de classification sur la base MNIST. Le protocole expérimental est similaire à celui de [14]. Pour réaliser ces expériences, nous nous appuyons sur une version adaptée de la bibliothèque Crino [1] basée sur les travaux de [10].

Nous testons ce problème sur différentes architectures. Quelle que soit l'architecture choisie, les dimensions d'entrée ($28^2$) et de sortie (10) sont fixes. Les architectures diffèrent uniquement par le nombre de couches cachées et par la taille de chacune des couches. Nous les référençons par $RNP_K$ où $K$ indique le nombre de couches cachées :

- $RNP_1$ : avec une taille de 50.
- $RNP_2$ : avec les tailles de : 60, 50.
- $RNP_3$ : avec les tailles de : 70, 60, 50.
- $RNP_4$ : avec les tailles de : 80, 70, 60, 50.

Une initialisation aléatoire des poids est effectuée. Afin d'avoir une comparaison juste entre les méthodes, cet état est enregistré comme point de départ de toutes les approches présentées. Chaque réseau $RNP_K$ est optimisé une première fois en utilisant seulement le critère supervisé $\mathcal{J}_s$ (Eq.1). L'erreur de classification sur l'ensemble de test est considérée comme l'*erreur de base*.

En partant à nouveau de l'état initial enregistré, chaque $RNP_K$ est optimisé avec le critère multi-tâche (Eq.4) en utilisant les différents schémas d'évolution des poids décrits précédemment :

— $escalier_{100}$, le pré-apprentissage traditionnel ;

— $lin_{100}$, le schéma linéaire abrégé ;

— $lin$, le schéma linéaire ; et

— $exp_{40}$, le schéma linéaire exponentiel.

Tous les apprentissages se terminent après 5000 époques en utilisant un mini-batch de taille 600 et un pas d'apprentissage de 0.01. Afin de ne pas entraîner de biais entre l'évolution des poids du cadre multi-tâche et l'évolution du pas d'apprentissage, ce dernier est maintenu constant jusqu'aux 500 dernières époques puis est atténué. Pour apprendre les auto-encodeurs dans le cas du schéma $escalier_{100}$, nous utilisons un pas d'apprentissage optimisé sur une base de validation comme il est d'usage dans le pré-apprentissage traditionnel.

Dans les tableaux de la Table 1, nous présentons en entête l'erreur de base en pourcentage sur l'ensemble de test pour différentes tailles $l$ d'ensemble d'apprentissage étiqueté. En dessous, nous indiquons les erreurs des méthodes AMT pour différentes tailles $l$ et $u$ des ensembles étiquetés et non-étiquetés, et pour les différent schémas d'évolution des poids. Dans cette partie, les erreurs sont indiquées en tant que différences relatives par rapport à l'erreur de base correspondante. Ainsi, dans le tableau Tab.1a, à la ligne $u = 10^3$ et à la colonne $(l = 100, \boldsymbol{lin_{100}})$, on trouve le chiffre $-0.87$. Ce qui veut dire que le réseau de neurone $\boldsymbol{RNP_1}$ appris avec une méthode AMT sur 100 exemples étiquetés et $10^3$ exemples non-étiquetés suivant un schéma d'évolution des poids $\boldsymbol{lin_{100}}$ propose une erreur inférieure de $0.87\%$ par rapport au réseau équivalent appris de manière standard, qui lui affiche $31\%$ d'erreurs.

Tous les schémas d'évolution des poids présentés sont testés pour les réseaux à une couche cachée (Tab.1a). Pour

1. https://github.com/jlerouge/crino

les réseaux plus profonds (Tab.1b), nous présentons uniquement les résultats du schéma $exp_{40}$, qui est celui ayant donné les meilleurs résultats dans le cas des architectures à une couche cachée.
Dans le cas des réseaux à une couche cachée (Tab.1a), nous constatons que notre approche améliore les résultats dans les différents schémas avec une meilleure performance globale du schéma $exp_{40}$. Nous observons aussi que plus on ajoute de données étiquetées ($l$ augmente), plus l'écart entre l'erreur de base et les erreurs en AMT se rapprochent. Dans ce cas, nous notons en parallèle une augmentation des poids $\mathbf{w}_p$ ; ce problème peut être allégé avec une régularisation de type $l_1, l_2$, ce que nous envisageons de faire dans des travaux futurs. Nous constatons aussi que notre approche n'améliore pas les performances lorsque ce sont uniquement les données étiquetées qui sont utilisées, i.e lorsque $u = 0$. Nous pouvons expliquer ce comportement par le fait que l'amélioration observée dans notre approche est principalement due à l'information capturée à partir des données non-étiquetées.
Dans le cas des réseaux de neurones profonds (Tab.1b), nous observons une amélioration globale en utilisant notre approche. Lorsque la quantité de données étiquetées croît, l'écart se réduit, comme pour les réseaux à une couche cachée.

# 5 Conclusion

Nous avons présenté dans cet article un nouveau schéma d'apprentissage pour les réseaux de neurones profonds, dans lequel nous avons utilisé un cadre d'apprentissage multi-tâche avec une tâche supervisée et une tâche non-supervisée. Chacune des tâches possède un poids d'importance. Nous avons proposé de faire évoluer ces poids au cours de l'apprentissage de manière dynamique suivant différents schémas. Nous étendons ainsi la méthode de pré-apprentissage des couches d'entrée par auto-encodeurs qui n'est qu'un cas spécifique d'évolution des poids dans ce cadre. En utilisant peu d'hyper-paramètres, nous avons réussi à améliorer la performance des réseaux de neurones en exploitant les données non étiquetées.
Dans nos travaux futurs, nous envisageons l'utilisation d'une régularisation de type $l_1, l_2$ pour mieux contrôler les paramètres partagés. Dans l'objectif de construire un cadre AMT automatique, nous envisageons d'arrêter l'apprentissage de la tâche de reconstruction *tôt* (early stopping) en nous basant sur l'erreur sur les ensembles d'apprentissage et de validation.

# Références


[1] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization : A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning*, 7 :2399–2434, 2006.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural information Processing Systems 19, NIPS 2006*, pages 153–160, 2006.

[3] R. Caruana. Multitask learning. *Machine Learning*, 28(1) :41–75, 1997.

[4] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. Adaptive computation and machine learning. MIT Press, 2006.

[5] O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In *Advances in Neural Information Processing 15, NIPS 2002*, pages 585–592, 2002.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing : deep neural networks with multitask learning. In *Machine Learning, Proceedings of he 25th International Conference, ICML 2008*, pages 160–167, 2008.

[7] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7) :1527–1554, 2006.

[8] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, 2006.

[9] L. Holmstrom and P. Koistinen. Using additive noise in back-propagation training. *Neural Networks, IEEE Transaction*, 3(1) :24–38, 1992.

[10] J. Lerouge, R. Herault, C. Chatelain, F. Jardin, and R. Modzelewski. Ioda : An input/output deep architecture for image labeling. *Pattern Recognition*, 48(9) :2847 – 2858, 2015.

[11] M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. unsupervised learning of invariant feature hierarchies with applications to object recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2007*, 2007.

[12] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1) :67–79, 1991.

[13] P. Vincent, H. Larochelle, I. Lajoie, and Y. Bengio. Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11 :3371–3408, 2010.

[14] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Machine Learning, Proceedings of 25th International Conference, ICML 2008*, pages 1168–1175, 2008.

[15] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *Computer Vision, ECCV 2014, 13th European Conference*, pages 94–108, 2014.

[16] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, CMU-CALD-02-107, Carnegie Mellon university, 2002.

TABLE 1 – Erreur de classification sur la base de test de MNIST.

Les valeurs indiquées en en-tête sont les erreurs de test en pourcentage absolu ;
et sont les différences relatives aux erreurs de base correspondantes pour les valeurs dans le corps

(a) Réseau de neurones *à une couche cachée*

| Étiquetées | $l = 100$ | | | | $l = 10^3$ | | | | $l = 10^4$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Architecture | $\boldsymbol{RNP_1}$ | | | | $\boldsymbol{RNP_1}$ | | | | $\boldsymbol{RNP_1}$ | | | |
| Erreur de base | 31% | | | | 12.85% | | | | 9.08% | | | |
| Schéma / Non-étiquetées | $\boldsymbol{escalier_{100}}$ | $\boldsymbol{lin_{100}}$ | $\boldsymbol{lin}$ | $\boldsymbol{exp_{40}}$ | $\boldsymbol{escalier_{100}}$ | $\boldsymbol{lin_{100}}$ | $\boldsymbol{lin}$ | $\boldsymbol{exp_{40}}$ | $\boldsymbol{escalier_{100}}$ | $\boldsymbol{lin_{100}}$ | $\boldsymbol{lin}$ | $\boldsymbol{exp_{40}}$ |
| $u = 0$ | +0.03 | +0.03 | +1.32 | +0.03 | +0.03 | 0.0 | +1.5 | +0.05 | +0.03 | +0.01 | +0.95 | -0.04 |
| $u = 10^3$ | -0.1 | -0.87 | -0.19 | -0.85 | +0.09 | -0.71 | +0.33 | -0.73 | +0.03 | -0.08 | +0.85 | -0.08 |
| $u = 2 * 10^3$ | -0.17 | -1.35 | -1.03 | -1.46 | +0.009 | -0.98 | -0.23 | -0.95 | +0.03 | -0.2 | +0.74 | -0.21 |
| $u = 5 * 10^3$ | -0.16 | -1.69 | -1.78 | -1.91 | +0.05 | -1.43 | -0.95 | -1.44 | +0.03 | -0.26 | +0.31 | -0.3 |
| $u = 10^4$ | -0.28 | -2.21 | -2.03 | -2.21 | 0.0 | -1.6 | -1.12 | -1.44 | +0.01 | -0.5 | -0.1 | -0.51 |
| $u = 2 * 10^4$ | -0.28 | -1.82 | -2.26 | -2.42 | -0.02 | -1.83 | -1.46 | -1.78 | +0.01 | -0.63 | -0.44 | -0.75 |
| $u = 4 * 10^4$ | -0.46 | -2.39 | -2.62 | -2.44 | -0.049 | -1.63 | -1.37 | -1.7 | +0.01 | -0.63 | -0.63 | -0.56 |
| $u = 5 * 10^4 - l$ | -0.19 | -1.89 | -2.03 | -2.44 | -0.06 | -1.68 | -1.56 | -1.75 | +0.01 | -0.63 | -0.63 | -0.56 |

(b) Réseaux de neurones *profonds*

| Étiquetées | $l = 100$ | | | $l = 10^3$ | | | $l = 10^4$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Architecture | $\boldsymbol{RNP_2}$ | $\boldsymbol{RNP_3}$ | $\boldsymbol{RNP_4}$ | $\boldsymbol{RNP_2}$ | $\boldsymbol{RNP_3}$ | $\boldsymbol{RNP_4}$ | $\boldsymbol{RNP_2}$ | $\boldsymbol{RNP_3}$ | $\boldsymbol{RNP_4}$ |
| Erreur de base | 31.48% | 33.58% | 30.53% | 12.32% | 11.96% | 12.54% | 5.62% | 5.69% | 5.5% |
| Schéma / non-étiquetées | $\boldsymbol{exp_{40}}$ | | | $\boldsymbol{exp_{40}}$ | | | $\boldsymbol{exp_{40}}$ | | |
| $u = 0$ | +0.01 | -0.01 | +0.03 | +0.02 | -0.08 | -0.03 | -0.11 | -0.03 | +0.08 |
| $u = 10^3$ | -1.05 | -0.83 | -0.92 | -0.55 | -0.58 | -0.8 | -0.2 | -0.07 | +0.12 |
| $u = 2 * 10^3$ | -2.03 | -1.35 | -1.17 | -0.78 | -0.77 | -1.18 | -0.4 | -0.01 | +0.04 |
| $u = 5 * 10^3$ | -2.32 | -1.71 | -1.31 | -1.15 | -1.23 | -1.56 | -0.54 | -0.18 | -0.04 |
| $u = 10^4$ | -2.26 | -1.29 | -0.72 | -1.14 | -1.29 | -1.76 | -0.75 | -0.22 | -0.24 |
| $u = 2 * 10^4$ | -2.4 | -0.78 | -0.82 | -1.25 | -1.48 | -1.88 | -0.81 | -0.41 | -0.08 |
| $u = 4 * 10^4$ | -2.12 | -1.17 | -1.65 | -1.08 | -1.3 | -1.92 | -0.81 | -0.42 | -0.55 |
| $u = 5 * 10^4 - l$ | -1.99 | -1.23 | -0.88 | -1.23 | -1.3 | -1.85 | -0.81 | -0.42 | -0.55 |