Université Claude Bernard Lyon 1

Rapport de Travaux Pratiques

Bio-Inspired Intelligence
M. Lefort

Introduction à l'apprentissage profond - Partie I

Otman AZZIZ, 11709456
otman.azziz@etu.univ-lyon1.fr

Swann RUYTER, 12105817
swann.ruyter@etu.univ-lyon1.fr

Partie I : Perceptron

1.1 Lecture du premier chapitre

Tenseur : structure de données spécialisée qui est très similaire aux tableaux et aux matrices. Permet d'encoder les entrées et les sorties d'un modèle, ainsi que les paramètres du modèle.

1.1.1 Initialisation du tenseur

A partir des données : `torch.tensor(data)`

A partir d'un tableau numpy : `torch.from numpy(npArray)`

A partir d'un autre tenseur : `torch.ones like(xData)`

Si l'on veut changer les propriétés : `torch.rand like(xData, dtype=torch.float)`

Il est également possible de le faire avec des valeurs aléatoire ou des constantes :

```
shape = (2, 3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

1.1.2 Attributs du Tensor

Permettent de décrire leur forme, leur type de données et l'appareil sur lequel ils sont stockés.

1.1.3 Opérations Tensorielles

Il existe plus de 100 opérations possibles, au niveau des opérations mathématiques jusqu'à la création d'OPS. Pour une liste exhaustive : `https://pytorch.org/docs/stable/torch.html`

1.1.4 Tensor et Numpy

Il est également possible de passer d'un tableau Tensor vers Numpy et inversement.

1.2 Tenseur

Indiquer la taille de chaque tenseur dans le premier fichier. Expliquer.

Avant tout, le jeu de données considéré dans ce TP est le *MNIST*, composé de 70000 images de chiffres manuscrits (labellisées) où chacune comporte 28 x 28 pixels, soit au 784 pixels. En vectorisant ces images, les 784 pixels sont chacun attribués à un neurone de la couche d'entrée, qui seront associés à 10 neurones de sortie, correspondant aux labels, dont l'objectif est de classifier correctement le chiffre présent sur l'image. Ainsi, les données de sortie sont un vecteur de 10 probabilité, la plus importante sera celle retenue pour la classification.

```
Shape of data_train: torch.Size([63000, 784])
Shape of label_train: torch.Size([63000, 10])
Shape of data_test: torch.Size([7000, 784])
Shape of label_test: torch.Size([7000, 10])
Shape of w: torch.Size([784, 10])
Shape of b: torch.Size([1, 10])
Shape of x: torch.Size([5, 784])
Shape of y: torch.Size([5, 10])
Shape of t: torch.Size([5, 10])
Shape of grad(t-y): torch.Size([5, 10])
```

- Les `data_train` et `data_test`, de tailles respectives *63 000 × 784* et *63 000 × 10* correspondent aux images (et permettent respectivement de faire apprendre le Perceptron et de le tester), au nombre de 63000 et de taille 784 pixels, ou 10 sorties possibles.

- Les `label_train` et `label_test` correspondent à la véritable valeur de l'image (e.g., $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$ associé à image "2").

- Un poids `w` st associé à chaque entrée du Perceptron (i.e. à chaque pixel de l'image) qui est reliée à toutes les sorties (e.g., la véritable valeur de l'image de 0 à 9). Finalement, `w` st cette association entre les deux : c'est une matrice de taille 728 x 10 représentant le poids des connexions mis à jour au fur et à mesure de l'apprentissage.

- Le biais `b` st le biais associé à chaque sortie, de taille 1 x 10 (i.e., un seul biais par sortie).

- Concernant l'apprentissage pour `x` `y` `t` t `grad` sur 10 itérations, l'algorithme va mélanger `data_train` et va récupérer 5 images `x` prendre toutes les images serait beaucoup trop long pour le calcul d'une sortie `y` et prendre une seule image ne permettrait pas à l'algorithme de discriminer le chiffre présent sur l'image (variation du noir au blanc). Ensuite, l'algorithme récupère leur vraie valeur numérique (label) `t` t l'utilise pour calculer `grad` ui est l'erreur (*loss function*).

- L'algorithme fait ensuite une `back-propagation` sur le `poids w` et le `biais b`

- Enfin, pour la phase de test, `x` `y` `t` on récupère non pas 5 images mais 1 image pour vérifier que l'algorithme à bien appris. Les dimensions associées sont :

```
Shape of x: torch.Size([5, 784])
Shape of y: torch.Size([5, 10])
Shape of t: torch.Size([5, 10])
```

1.3 Impact des hyperparamètres

Ajuster **le taux d'apprentissage** η est une étape sensible car un taux d'apprentissage faible (ou trop petit) conduit à une convergence très lente (très petits pas vers les valeurs de paramètres optimaux) ayant pour conséquence un temps d'entraînement trop grand. Néanmoins, un trop fort taux d'apprentissage aura pour conséquence l'absence de convergence en faisant fluctuer la fonction de perte autour du minimum. Dans certains cas, il peut aussi diverger. Il y a donc un juste milieu à trouver dans le taux d'apprentissage, qui est généralement fixé à $1e^{-4}$.

Vis-à-vis des poids initiaux, si les poids sont importants, par exemple $de - 0.1$ à 0.1 l'algorithme, dans ce contexte-ci,va partir d'une performance relativement basse et augmenter rapidement avant de se stabiliser, sans atteindre de bonnes performances. Ceci s'explique car un pas d'apprentissage trop grand ne permet pas au modèle de converger vers le minimum.

```
tensor([0.4634])
tensor([0.5927])
tensor([0.6544])
tensor([0.6870])
tensor([0.7091])
tensor([0.7263])
tensor([0.7377])
tensor([0.7474])
tensor([0.7559])
tensor([0.7634])
```

En comparaison avec de -0.01 à 0.01, où ici, le réseau fait tout de suite de bonnes prédictions mais atteint un plafond relativement vite :

```
tensor([0.7949])
```

```
tensor([0.8217])
tensor([0.8326])
tensor([0.8373])
tensor([0.8409])
tensor([0.8440])
tensor([0.8457])
tensor([0.8483])
tensor([0.8503])
tensor([0.8524])
```

Enfin, dans des cas plus extrêmes, lorsque les poids initiaux sont très petits par exemple, le modèle convergera très lentement ; à l'opposé d'un pas beaucoup trop grand (par exemple, 1), nous nous retrouverons dans un cas similaire à ceux présentés précédemment : l'algorithme tombera dans des minima locaux et ne pourra pas converger.

Partie II : Shallow network

Cette partie tente de quantifier l'influence des hyperparamètres η, des poids initiaux et du nombre de neurones pour la couche cachée. Ici, le nombre d'époques et de mini-batch sera fixé à `nb_epoch = 10` et `batch_size = 5`

Le premier paramètre investigué est le **taux d'apprentissage** η. D'après la littérature, c'est le paramètre le plus important et sa valeur optimale est d'environ la moitié du taux d'apprentissage maximal, i.e. le taux d'apprentissage au dessus duquel l'algorithme d'entraînement diverge.

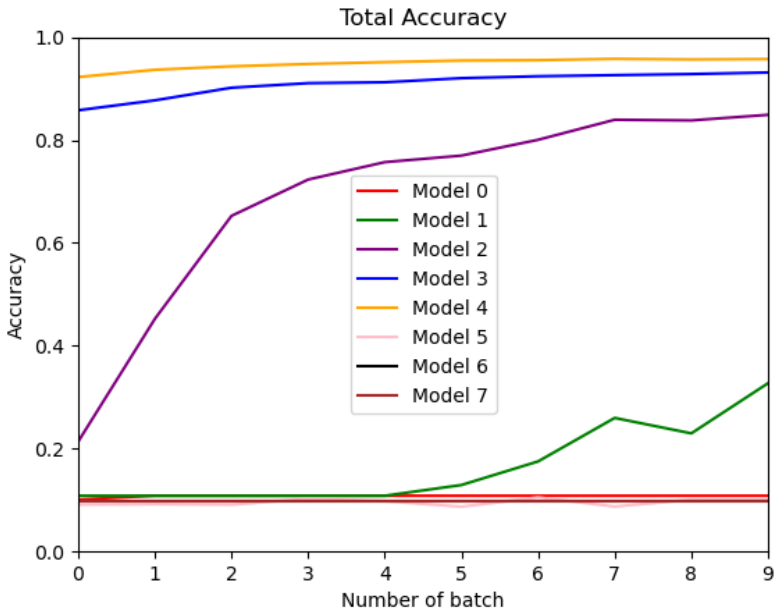

FIGURE 1 – Variation des performances pour η variant de $1e^{-5}$ à 10

Ici, chaque modèle est associé à un taux d'apprentissage de plus en plus grand à partir de $1e^{-5}$ (augmentation par $log(10)$). Les résultats indiquent que pour un taux d'apprentissage faible, i.e $1e^{-5}$ et $1e^{-4}$, le réseau ne dépasse pas 0.4 de performance, comme c'est le cas pour un taux d'apprentissage important, i.e. 0.1, 1 et 10. En outre, un taux compris entre 0.0001, 0.001, 0.01 offre de belles performances, particulièrement pour 0.01. Le modèle offrant le plus bel apprentissage est le numéro 2, même si ses performances finales sont inférieures. Cette valeur sera utilisée pour la suite de cette partie afin de voir si l'ajustement d'autres paramètres influencent la performance du réseau.

L'initialisation des **poids initiaux** est le deuxième paramètre dont l'influence tient à être quantifiée. Rappelons que le neurone reçoit des signaux en provenance de différentes sources (entrées) avec une certaine force associée (le poids), et plus cette valeur a de l'influence, plus l'intensité est forte, donc plus l'entrée correspondante l'est aussi. En ce qui concerne l'initialisation des poids, la littérature conseille de petites valeurs pour que le réseau s'ajuste au fur et à mesure des itérations.

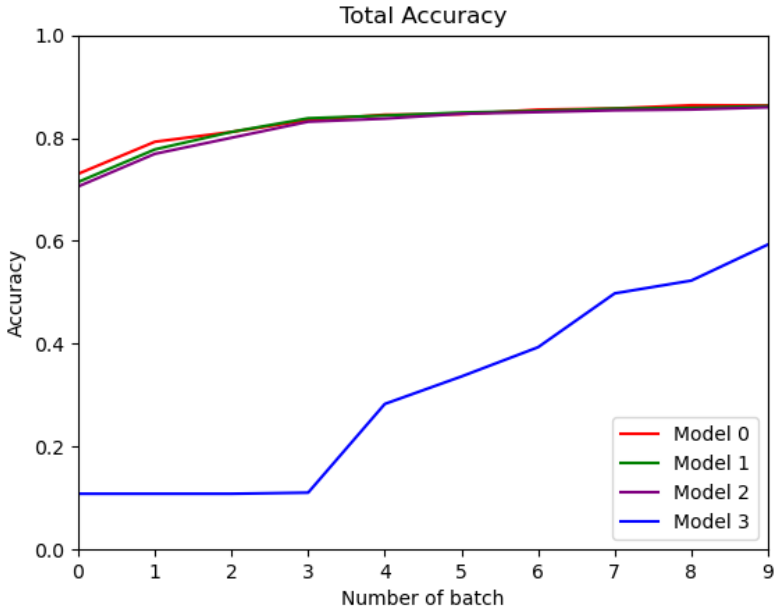

FIGURE 2 – Variation des performances pour différentes combinaisons de poids initiaux

Ici, chaque modèle a reçu une initialisation de poids comme suit (c1 pour couche une et c2 pour couche 2) :

1. Modèle 0 - c1 = 0.1, c2 = 0.1
2. Modèle 1 - c1 = 0.00001, c2 = 0.1
3. Modèle 2 - c1 = 0.1, c2 = 0.00001
4. Modèle 3 - c1 = 0.00001, c2 = 0.00001

La seule remarque importante à fournir est que des poids initialisés trop faibles ne permettront pas d'atteindre de bonnes performances, en tout cas rapidement. Dans le cadre d'un réseau de neurones à une couche cachée, il suffit qu'une partie des poids soit initialement suffisamment élevée pour que l'algorithme converge rapidement vers de bonnes performances. Ici, les poids sont assignés en suivant la loi Uniforme sur l'intervalle spécifié. Il serait par la suite intéressant d'investiguer ces résultats pour des valeurs plus extrêmes et uniquement positives ou négatives.

Le dernier paramètre étudié dans cette partie est **le nombre de neurones** de la couche cachée. Les résultats sont les suivants ($\eta = 1e^{-3}$) :

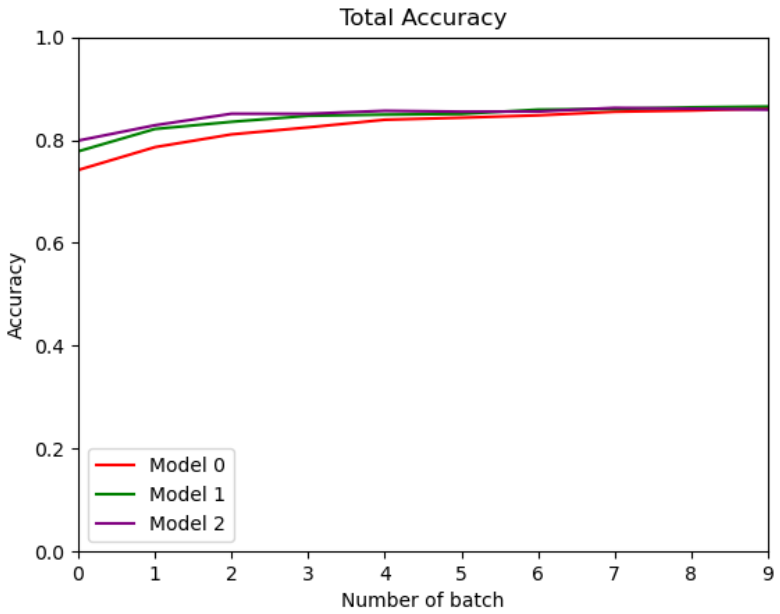

FIGURE 3 – Variation des performances pour le nombre de neurones pour la couchée cachée (respectivement, 64, 128, 256, $\eta = 1e^{-3}$)

Il est difficile ici de quantifier l'importance du nombre de neurones. Nous avons donc baisser le taux d'apprentissage, et les résultats sont plus saillants : un nombre élevé de neurone offre de meilleures performances. Il aurait été intéressant de fournir une visualisation pour un nombre de neurones beaucoup plus importants, mais faute de capacités suffisantes et du temps de traitement de l'algorithme, nous ne sommes pas aller au delà de 256.

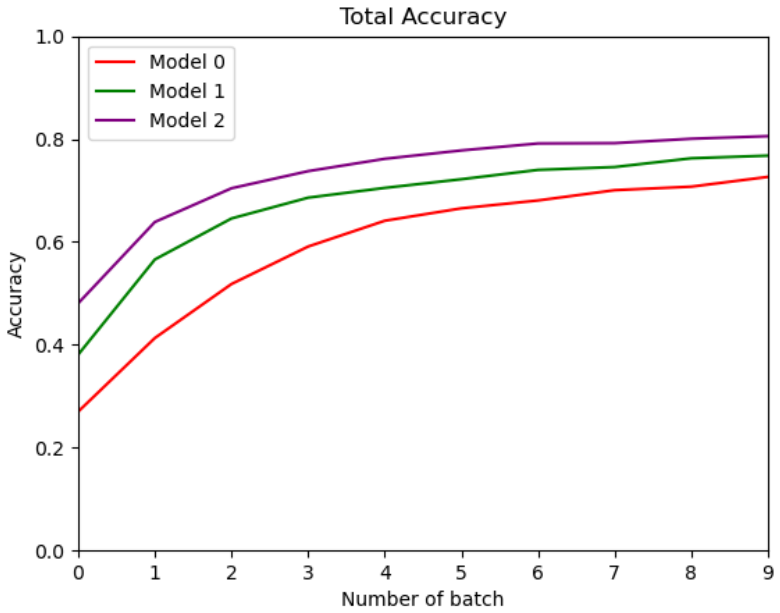

FIGURE 4 – Variation des performances pour le nombre de neurones pour la couchée cachée (respectivement, 64, 128, 256, $\eta = 1e^{-4}$)

Partie III : Deep network

La partie II nous a permis d'identifier l'importance du **taux d'apprentissage**, spécifique au problème concerné (i.e. il n'y a pas de méthodes universelles permettant de déterminer le taux optimal). Dans le cadre d'un réseau avec une couche cachée, les performances du modèle peuvent varier énormément. Vis-à-vis des poids initiaux, l'influence semble moins importante. Enfin, un nombre important de neurone dans la couche cachée offre de meilleures performances.

L'objectif de cette partie est tout d'abord d'observer l'influence de ces paramètres lors de **2 couches cachées**. Ensuite, de déterminer l'influence de `batch size` de l'ajout d'une troisième couche cachée, et de la variation du nombre de neurones.

Cette partie est accompagnée de la somme cumulée de la fonction de coût. Un bon modèle d'apprentissage devrait être associé à une fonction de coût dont la forme est logarithmique (augmente beaucoup au début puis se stabilise lors de l'arrivée à l'optimum).

De la même manière que la partie II, cette première figure représente la performance du modèle en faisant varier le **taux d'apprentissage** η. Les résultats sont similairement les mêmes à la différence que les modèles les moins performants sont ceux situés aux valeurs extrêmes de η. La fonction de coût associée est cohérente avec la performance : les modèles 0 et 6 sont associés à une fonction de coût qui ne cesse d'augmenter, à l'inverse du modèle 4 et 5 qui se stabilisent.

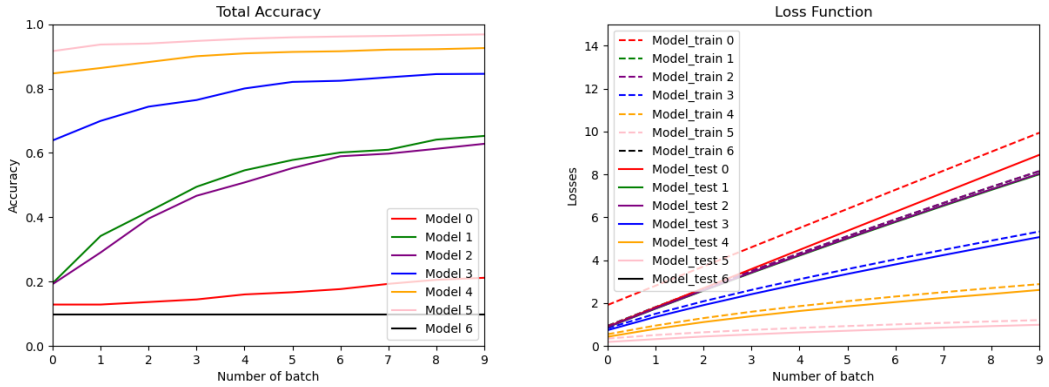

FIGURE 5 – Variation des performances pour différentes valeurs de η

Pour les analyses suivantes, le modèle 3 associé à $\eta = e^{-3}$ a été sélectionné. Nous avons fait varier **les poids initiaux** des couches cachées de la façon suivante : les poids initialisés à 0.1 pour toutes les couches, à $1e^{-4}$ pour toutes les couches, puis $1e^{-4}$ pour la première et troisième couche cachée avec la deuxième à 0.1, et inversement pour le dernier modèle. Les résultats sont cohérents avec ce qui a été observé précédemment : les poids initiaux ne doivent pas être trop petits. Ici, nous pouvons également émettre l'hypothèse que le plus important est l'initialisation des poids pour la première et troisième couche, la couche intermédiaire ne détermine pas, ou peu, les performances du modèle.

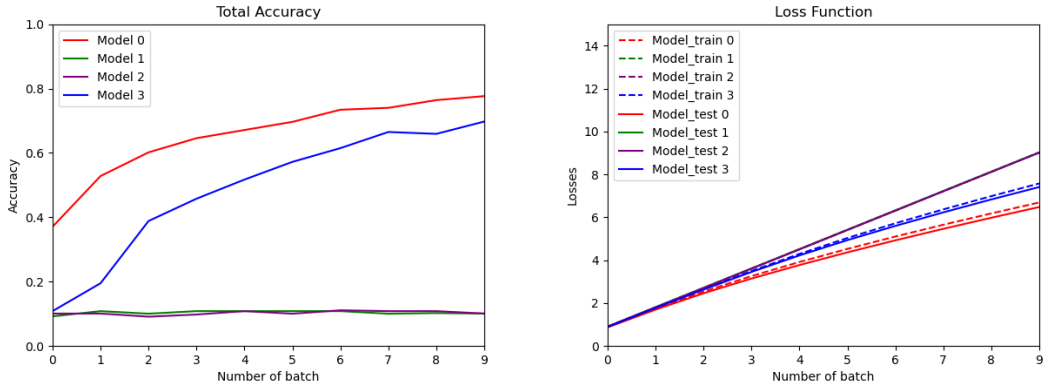

FIGURE 6 – Variation des performances par rapport aux poids initiaux

A présent, il est important de comprendre l'importance de **la taille du jeu de données d'apprentissage**. En ce sens, nous pourrions nous attendre à de meilleures performances des modèles ayant des jeux d'apprentissage plus élevés. Or, en faisant varier le nombre de `Batch Size` e 5 à 25, les différences frôlent le néant. La littérature souligne qu'une taille de lot élevée conduit souvent à des instabilités d'entraînement, au particulier au début, ce que nous ne notons pas particulièrement ici. La généralisation du modèle est également plus difficile, est Yann Lecun a même dit *"Friends don't let friends use mini-batches larger than 32"*

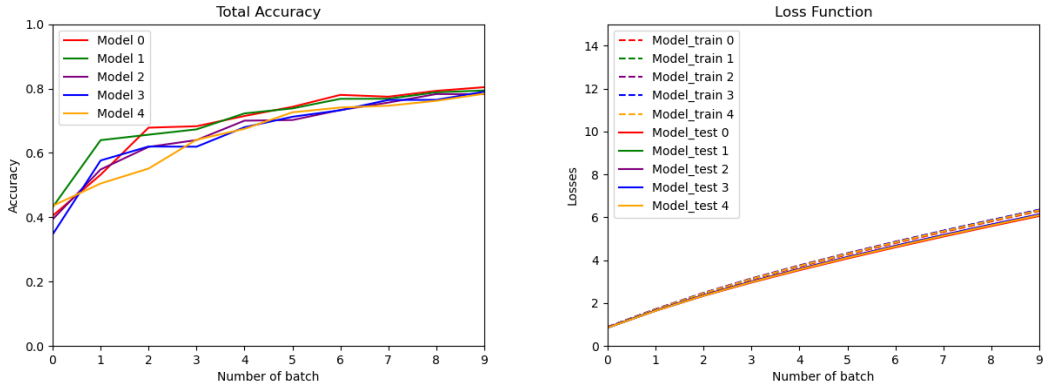

FIGURE 7 – Variation des performances selon la taille de lot)

Les figures suivantes sont associées aux performances des modèles en faisant variant **le nombre de neurones par couche cachée**, respectivement : *64 - 64, 256 - 256, 64 - 256* et inversement. D'après ces résultats, la deuxième couche cachée semble avoir le plus de poids parce qu'elle atteint les mêmes performance que le modèle 1, associé à un nombre de neurones élevés. Ces résultats sont intéressants mais il est nécessaire de comprendre l'implication de la première et dernière couche cachée lors de l'ajout d'une troisième couche cachée, intermédiaire.

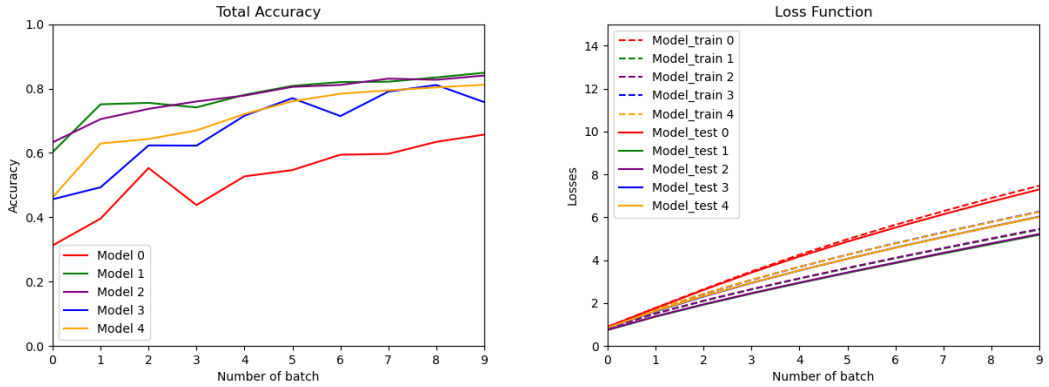

FIGURE 8 – Variation des performances selon le nombre de neurones par couche cachée

Afin de quantifier l'influence du nombre de neurones par couches, nous avons ajouté une **troisième couche cachée** au modèle. La seule conclusion pouvant être émise ici est la nécessité d'un taux d'apprentissage supérieur lors de l'ajout de couches de neurones, en notant également que l'apprentissage semble très instable et sans convergence.

Caractéristiques : $\eta = 1e^{-}3$, pour chaque modèle : [256, 64, 64], [256, 128, 64], [256, 256, 64], [64, 64, 256], [64, 128, 156], [64, 256, 256].

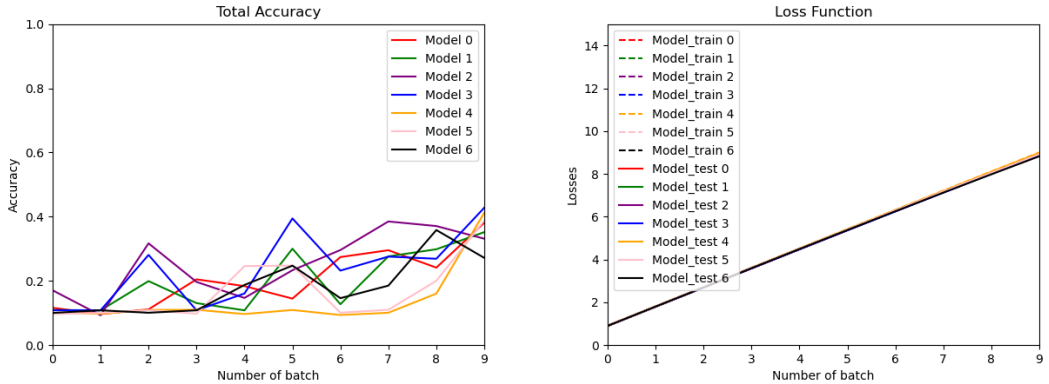

FIGURE 9 – Variation des performances selon le nombre de neurones pour la couchée cachée, $\eta = 1e^{-}3$

Nous avons donc augmenté le taux d'apprentissage, et la performance est plus importante. Néanmoins les modèles convergent tous vers la même performance, limitant encore la possibilité d'interpréter le comportement.

Caractéristiques : $\eta = 1e^{-}2$, pour chaque modèle : [256, 64, 64], [256, 128, 64], [256, 256, 64], [64, 64, 256], [64, 128, 156], [64, 256, 256].

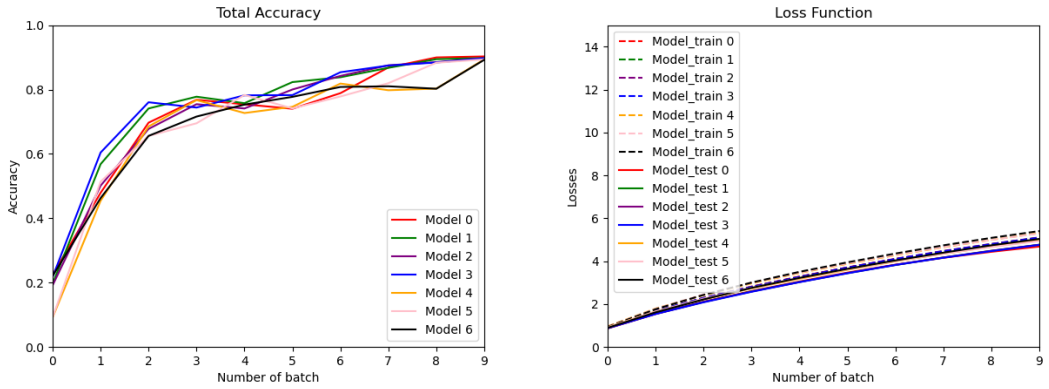

FIGURE 10 – Variation des performances selon le nombre de neurones pour la couchée cachée, $\eta = 1e^{-}3$

Nous avons donc **changé les combinaisons de nombre de neurones par couche** selon la logique suivante : un nombre élevé, intermédiaire et faible pour chaque couche, puis soit couche 1 et 3 élevées et 2 faible ou moyenne, soit des couches 1 et 3 faibles et une couche intermédiaire élevée.

Caractéristiques : $\eta = 1e^{-}2$, pour chaque modèle : [256, 256, 256], [128, 128, 128], [64, 64, 64], [256, 64, 256], [256, 128, 256], [64, 256, 64].

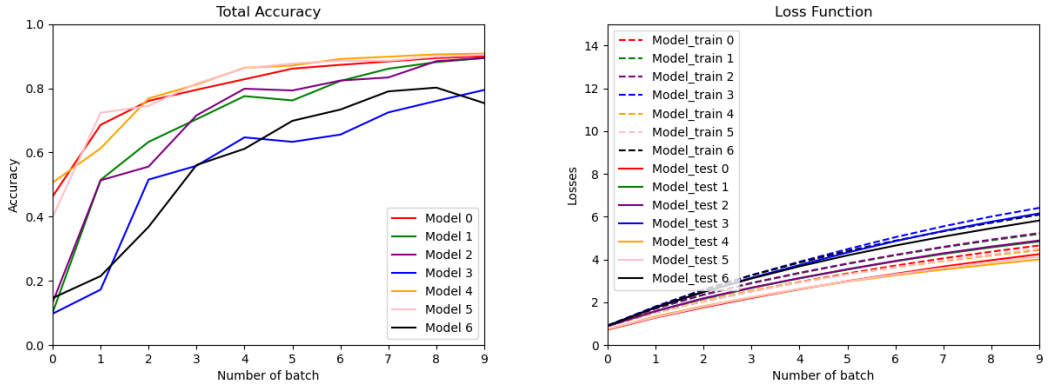

FIGURE 11 – Variation des performances selon le nombre de neurones pour la couchée cachée, $\eta = 1e^{-}2$

Les résultats sont clairs et semblent confirmer notre hypothèse de la partie 2 : un nombre de neurones élevé sur les couches périphériques offre de meilleurs résultats que pour un nombre élevé dans la couche intermédiaire. En effet, lors d'un nombre de neurones des couches cachées faibles, le réseau compresse l'information, et indéniablement, perd de l'information. A l'opposé, un nombre important de neurones dans les couches cachées améliore ses performances puisque cela augmente son nombre de paramètres, et donc ses capacités de généralisation. Néanmoins, un nombre trop important de neurones n'offre pas que des avantages, puisque cela augmente considérablement le temps de calcul, et parfois, amène au même résultat qu'un faible nombre de neurones, certainement par une redondance des paramètres.

Partie IV : Pour aller plus loin

Nous avons décidé de nous intéresser à l'influence des **fonctions d'activation** dans le cadre d'un réseau à deux couches cachées. Nous avons fait varier le taux d'apprentissage pour chaque modèle. Les figures suivantes indiquent de meilleures performances pour la fonction d'activation ReLU pour $\eta = 0.01$. Néanmoins, les modèles semblent sur-ajustés, et perdent potentiellement en capacité de généralisation.

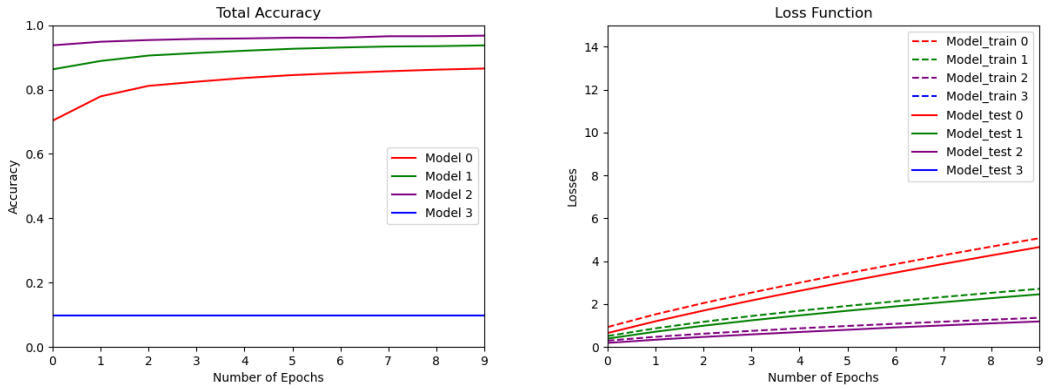

FIGURE 12 – Variation des performances selon la fonction d'activation Tanh

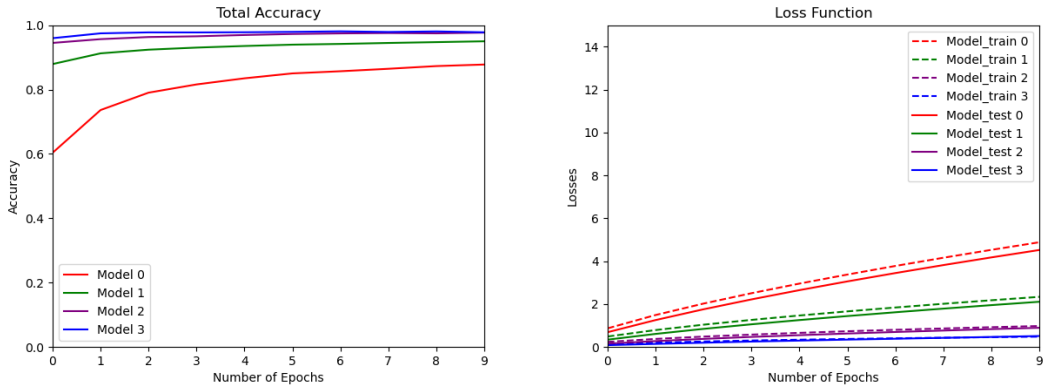

FIGURE 13 – Variation des performances selon la fonction d'activation ReLU

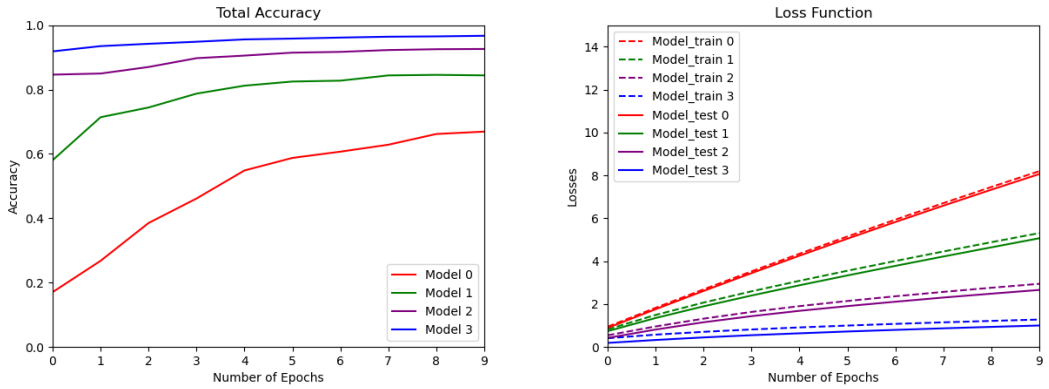

FIGURE 14 – Variation des performances selon la fonction d'activation Sigmoid

Pour finir, nous avons testé **une initialisation des poids selon une loi normale**. Les résultats sont les suivants :

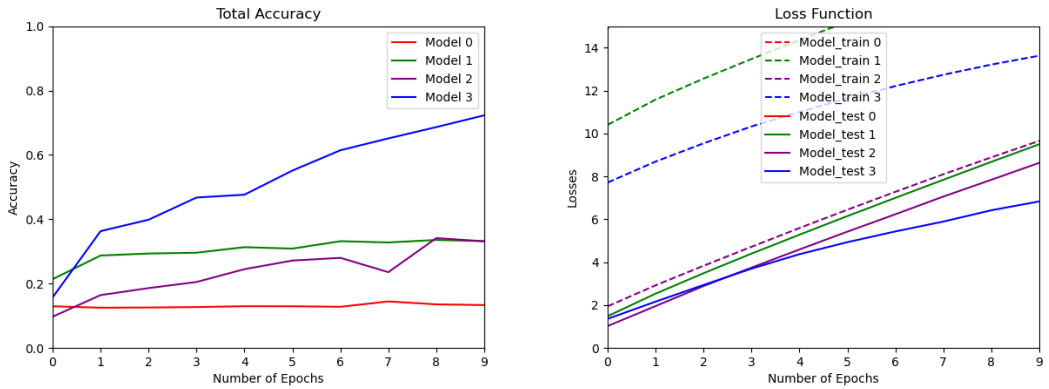

FIGURE 15 – Variation des performances selon une initialisation gaussienne des poids

C'est la première fois, durant l'ensemble des variations des paramètres, que nous observons une telle différence entre la fonction de coût d'apprentissage et de test. Vis-à-vis de la performance, nous arrivons à la même conclusion que précédemment : le taux d'apprentissage est le paramètre le plus important.