FACULTÉ DES SCIENCES ET INGÉNIERIE

SORBONNE UNIVERSITÉ

RECONNAISSANCE DES FORMES POUR L'ANALYSE ET L'INTERPRÉTATION D'IMAGES

Rapport de TMEs 4 à 7

Auteur :
Ahmed Tidiane BALDÉ

Encadrants :
Arthur DOUILLARD
Yifu CHEN

17 novembre 2019

TABLE DES MATIÈRES

Introduction aux réseaux de neurones **3**

Partie 1 - Formalisation mathématique **3**

1.1 Jeu de données . . . 3
Q1-1.1 . . . 3
Q1-1.2 . . . 3
1.2 Architecture du réseau (phase *Forward*) . . . 3
Q1-2.3 . . . 3
Q1-2.4 . . . 3
Q1-2.5 . . . 3
Q1-2.6 . . . 4
Q1-2.7 . . . 4
1-3 Fonction coût . . . 4
Q1-3.8 . . . 4
Q1-3.9 . . . 5
1.4 Méthode d'apprentissage . . . 5
Q1-4.10 . . . 5
Q1-4.11 . . . 5
Q1-4.12 . . . 6
Q1-4.13 . . . 6
Q1-4.14 . . . 6
Q1-4.15 . . . 6
Q1-4.16 . . . 7
Q1-4.17 . . . 7

Partie 2 - Implémentions **7**

2.1 Application à MNIST . . . 7

Annexe : Résultats CirclesData **8**

Réseaux convolutionnels pour l'image **9**

Partie 1 - Introduction aux réseaux convolutionnels **9**

Q1.1 . . . 9
Q1.2 . . . 9
Q1.3 . . . 9
Q1.4 . . . 9
Q1.5 . . . 10
Q1.6 . . . 10
Q1.7 . . . 10

Partie 2 - Apprentissage from *scratch* du modèle **10**
2.1 Architecture du réseau . . . 10
Q2-1.8 . . . 10
Q2-1.9 . . . 10
Q2-1.10 . . . 10
Q2-1.11 . . . 11
Q2-1.12 . . . 11
2.2 Apprentissage du réseau . . . 11
Q2-2.14 . . . 11
Q2-2.15 . . . 11
Q2-2.16 . . . 12
Q2-2.17 . . . 12
Q2-2.18 . . . 13

Partie 3 - Amélioration des résultats **13**
3.1 Normalisation des exemples . . . 13
Q3-1.19 . . . 13
Q3-1.20 . . . 13
Q3-1.21 **Bonus** . . . 13
3.2 Augmentation du nombre d'exemples d'apprentissage par *data augmentation* . . . 14
Q3-2.22 . . . 14
Q3-2.23 . . . 14
Q3-2.24 . . . 14
Q3-2.25 **Bonus** . . . 14
3.3 Variantes sur l'algorithme d'optimisation . . . 15
Q3-3.26 . . . 15
Q3-3.27 . . . 15
Q3-2.28 **Bonus** . . . 15
3.4 Régularisation du réseau par *dropout* . . . 16
Q3-.29 . . . 16
Q3-4.30 . . . 17
Q3-4.31 . . . 17
Q3-4.32 . . . 17
Q3-4.33 . . . 17
3.5 Utilisation de *batch normalization* . . . 18
Q3-4.34 . . . 18

Annexe **18**

INTRODUCTION AUX RÉSEAUX DE NEURONES

PARTIE 1 - FORMALISATION MATHÉMATIQUE

1.1 JEU DE DONNÉES

Q1-1.1

Nous séparons nos données en trois sous-ensembles :
- Un ensemble d'**apprentissage**, qui est l'ensemble qui permet l'apprentissage des paramètres du modèle.
- Un ensemble de **validation** est l'ensemble sur lequel les hyper-paramètres sont réglés.
- Un ensemble de **test** sur lequel notre modèle est testé.

Q1-1.2

Il est important que le nombre d'exemples N soit grand car il permet une meilleure généralisation et une approximation de la loi générant nos données et nous évite un sur-apprentissage.

1.2 ARCHITECTURE DU RÉSEAU (PHASE *Forward*)

Q1-2.3

Ajouter des fonctions d'activation entre des transformations linéaires est nécessaire afin d'apporter de la **non-linéarité** à notre réseau. Dans le cas contraire, notre réseau ne sera pas performant sur des données non-linéairement séparables.

Q1-2.4

En pratique, la dimension des entrées dépend de la dimension des données d'entrée, la dimension des sorties dépend de la dimension que nous souhaitons obtenir, le nombre de neurones est un hyper-paramètre que nous fixons en fonction du score obtenu sur le sous-ensemble de validation.
Nous considérons l'architecture du réseau de neurone de la figure 1 :
n_x : Dimension des entrées, $n_x = 2$.
n_h : Nombre de neurones dans la couche cachée, $n_h = 4$.
n_y : Dimension des sorties, $n_y = 2$.

Q1-2.5

Le vecteur y représente le vecteur de valeurs attendues (labels) alors que le vecteur $\hat{y}$ représente le vecteur des valeurs prédites par notre modèle. Ainsi, la différence entre ces deux vecteur est représentée par **la fonction coût**.

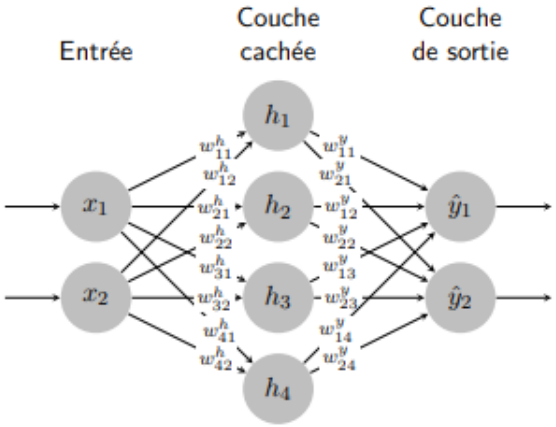


FIGURE 1 – Architecture d'un réseau de neurones à une couche cachée.

Q1-2.6

L'utilité de la fonction **Softmax** permet d'avoir la probabilité d'appartenance à chaque classe lors du problème multi-classes, et non un label de classe le plus probable, ce qui rend les résultats détaillés.

Q1-2.7

Les équations mathématiques permettant d'effectuer la passe *forward* du réseau de neurones sont les suivantes :

$$\tilde{h} = W_h x + b_h \tag{1}$$

$$h = \tanh(\tilde{h}) \tag{2}$$

$$\tilde{y} = W_y h + b_y \tag{3}$$

$$\hat{y}_i = SoftMax(\tilde{y}) \tag{4}$$

1-3 FONCTION COÛT

Q1-3.8

Afin de faire diminuer la loss : $\hat{y}_i$ doit être le plus proche possible de y_i, Ce qui nous amène à vérifier cela grâce à la dérivée de la fonction coût.
- Pour la **cross-entropy**, la dérivée est

$$\frac{\partial l}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \tag{5}$$

- Pour le coût **MSE**, la dérivée est :

$$\frac{\partial l}{\partial \hat{y}} = 2(y - \hat{y}) \tag{6}$$

Q1-3.9

Le coût *cross-entropy* va modifier le réseau de manière à faire augmenter le poids de la sortie correspondant au vrai label car c'est le seul neurone de sortie qui aura une dérivée non nulle. À l'inverse, le coût *MSE* va essayer d'avoir exactement la même distribution que l'entrée, ce qui fais que chaque sortie aura une dérivée non nulle, ce qui conduit à d'avantage tenir en compte les erreurs que la valeur associée au vrai label. Ce désavantage est très problématique pour des données en grande dimension.

1.4 MÉTHODE D'APPRENTISSAGE

Q1-4.10

Afin de minimiser la fonction de perte, nous minimisons les paramètres de notre modèle grâce à la descente de gradient, dont il existe plusieurs variantes :

- **Batch gradient descent** Consiste simplement à calculer le gradient sur la totalité des données. Certains problèmes qui se posent en ce faisant sont la redondance des calculs pour des datasets plus larges et dans le même ordre, il faut prendre en compte le temps de calcul très conséquent avant la visualisation des résultats. Cependant, cette technique nous permet d'atteindre le point de convergence optimal pour les problèmes convexes et nous garantit tout autant une convergence local, le cas échéant.
- **Stochastic Gradient descent** A contrario de la *vanilla gradient descent* (autre appellation du batch descent gradient), nous considérons ici un seul exemple à la fois pour le calcul du gradient. Cette méthode dispose de ses avantages, tels que la rapidité et la facilité de déploiement pour un apprentissage online. Toutefois, mettre à jour les paramètres pour un seul exemple à la fois cause des oscillations qui potentiellement peuvent mener l'algorithme a un meilleur minimum local mais aussi sous un angle moins enchantant, cela compliquerait naturellement la convergence.
- **Mini-batch gradient descent** Puis nous avons le *minibatch gradient descent* qui est quand à lui hybride car réunit le meilleur des deux dernières techniques en mettant à jour les paramètres θ pour chaque *minibatch* de n exemples d'apprentissage, plutôt que pour un (1) seul exemple ou la totalité des exemples d'apprentissage. Supposons n le nombre d'exemples d'apprentissage du *mini-batch*, alors si n est égal à 1 ou d, la totalité du nombre d'exemples de notre base d'apprentissage, nous sommes respectivement dans le cas du *stochastic gradient descent* et du *vanilla gradient descent*. Notons que cet algorithme est parfois appelé comme le précédent SGD, pour *stochastic gradient descent*.

Toujours est-il que nous avons certains *problèmes* ou plutôt challenges à surmonter, dont un qui est la déterminaison du taux d'apprentissage (*learning rate*) idéal.

Q1-4.11

Le choix de l'hyper-paramètre (*learning rate*) est important car une petite valeur de ce dernier ralentirait la convergence de la descente de gradient, cependant pour une valeur trop grande, nous nous éloignerons du minimum de la fonction de coût, ce qui risques de faire diverger nos résultats.

Q1-4.12

L'implémentation classique stocke les gradients en mémoire ce qui permet d'effectuer une passe en $O(n)$ car si nous ne les stockions pas (naïf) la complexité algorithmique serait en $O(n^2)$, $(\frac{n(n-1)}{2})$ plus précisément.

Q1-4.13

Le critère que doit respecter l'architecture du réseau pour permettre la *backpropagation* est que chaque coût et fonction d'activation doit être dérivable.

Q1-4.14

$$\begin{aligned} l(y,\tilde{y}) &= -\sum_i y_i \log(\tilde{y}_i) \\ &= -\sum_i y_i \log(\frac{\exp(\tilde{y}_i)}{\sum_i \exp(\tilde{y}_i)}) \\ &= -\sum_i y_i (\log(\exp\tilde{y}_i) - \log(\sum_i \exp(\tilde{y}_i))) \\ &= -\sum_i y_i (\tilde{y}_i - \log(\sum_i \exp(\tilde{y}_i))) \\ &= -\sum_i y_i \tilde{y}_i + \sum_i y_i \log(\sum_i \exp(\tilde{y}_i)) \end{aligned}$$

Sachant que $\sum_i y_i = 1$, alors :

$$l(y,\tilde{y}) = -\sum_i y_i \tilde{y}_i + \log(\sum_i \exp(\tilde{y}_i)) \tag{7}$$

Q1-4.15

$$\begin{aligned} \frac{\partial l(y,\tilde{y})}{\partial \tilde{y}_k} &= \frac{\partial(-\sum_i y_i \tilde{y}_i + \log(\sum_i \exp(\tilde{y}_i)))}{\partial \tilde{y}_k} \\ &= \frac{-\partial \sum_i y_i \tilde{y}_i}{\partial \tilde{y}_k} + \frac{\partial \log(\sum_i \exp(\tilde{y}_i))}{\partial \tilde{y}_k} \\ &= -y_k + \frac{\exp(\tilde{y}_k)}{\sum_i \exp(\tilde{y}_i)} \\ &= -y_k + \tilde{y}_k \end{aligned}$$

Q1-4.16

$$\frac{\partial l(y,\tilde{y})}{\partial W_{i,j}} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y_k}} \tag{8}$$

$$\frac{\partial \tilde{y_k}}{\partial W_{i,j}} = (-y_k + \tilde{y_k})h_i \tag{9}$$

$$\frac{\partial \tilde{y_k}}{\partial W_{i,j}} = \frac{\partial \sum_m W_{i,m}h_m}{\partial W_{i,j}} = h_j \tag{10}$$

Q1-4.17

$$\frac{\partial l(y,\tilde{y})}{\partial b_{y,j}} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y_k}}$$

$$\frac{\partial \tilde{y_k}}{\partial b_{y,j}} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y_i}}$$

$$\frac{\partial \tilde{y_i}}{\partial b_i} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y_i}}\mathbf{1}$$

$$\frac{\partial \tilde{y_k}}{\partial \tilde{h_i}} = \sum_j \frac{\partial \tilde{y_k}}{\partial \tilde{h_j}}$$

$$\frac{\partial h_j}{\partial \tilde{h_i}} = \frac{\partial \tilde{y_k}}{\partial h_i}\frac{\partial h_i}{\partial \tilde{h_i}} = W_{k,i}(1-h_i^2)$$

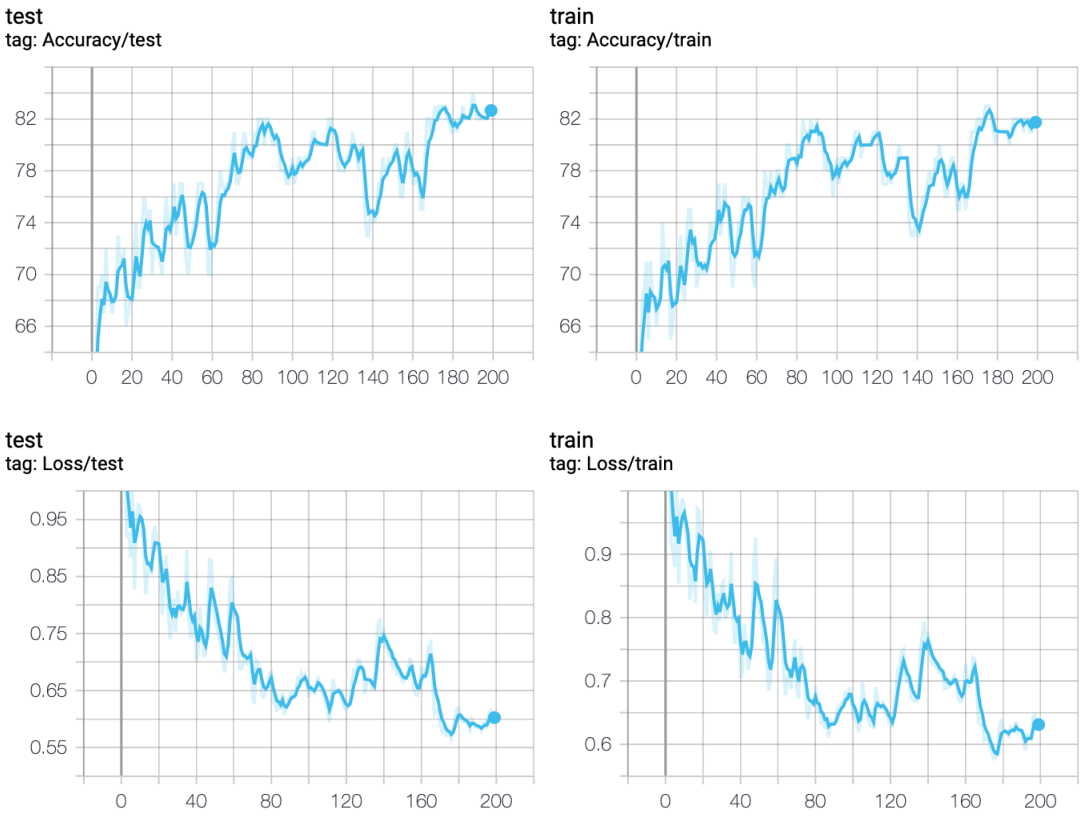
$$\frac{\partial h_i}{\partial \tilde{h_i}} = \frac{\partial \tanh(\tilde{h_1})\partial \tilde{h_i}}{} = \mathbf{1} - \tanh(\tilde{h_1^2}) = \mathbf{1} - h_i^2$$

$$\frac{\partial \tilde{y_k}}{\partial h_i} = \frac{\partial \sum_j W_{k,j}hj}{\partial h_i} = W_{k,i}$$

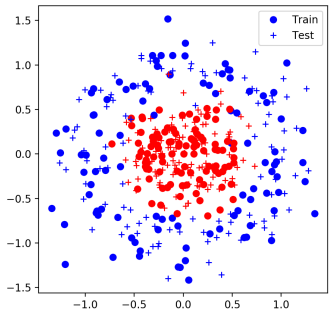
PARTIE 2 - IMPLÉMENTIONS

2.1 APPLICATION À MNIST

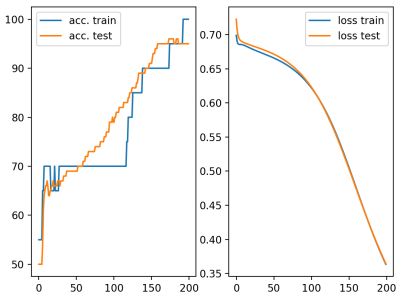
Après avoir implémenté toutes les différentes approches d'implémentations d'un réseau de neurones énoncées dans le sujet, nous avons appliqué notre code au jeu de données MNIST et avons obtenu les résultats qui suivent. Rappelons que ce jeu de données est un jeu d'images de chiffres manuscrits, disposant de 10 classes (10 chiffres) dont chaque image fait 28×28 pixels. Cela dit, elles sont représentées ici chacune en un vecteur de 784 valeurs.



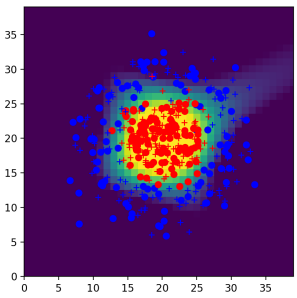
ANNEXE : RÉSULTATS CIRCLESDATA



Données



Accuracy and Loss



Données classées

RÉSEAUX CONVOLUTIONNELS POUR L'IMAGE

PARTIE 1 - INTRODUCTION AUX RÉSEAUX CONVOLUTIONNELS

Q1.1

En considérant un seul filtre de convolution de *padding* p, de *stride* s et de *taille de kernel* k, pour une entrée de dimension $x \times y \times z$, la taille de sortie serait :

$$\left(\left[\frac{x-k+2p}{s}\right]+1\right) \times \left(\left[\frac{y-k+2p}{s}\right]+1\right)$$

Ainsi, nous aurions alors $(k \times k \times z)$ poids à apprendre. Avec une couche *fully-connected*, nous aurions eu, $(x \times y \times z) \times (nb\_channel\_out \times x \times y)$, avec en l'occurrence *nb_channel_out* qui est égale à 1.

Q1.2

Au vu de ce qui précède, commençons par souligner la différence importante qui réside entre le nombre de poids en *fully-connected* et celui en *convolution*. Cette dernière en compte beaucoup moins, c'est d'ailleurs l'un de ses plus grands atouts par rapport à son homologue. De plus, la convolution peut se vanter d'avoir un nombre de paramètres invariant par rapport à la taille de l'image mais également d'une habilité à pouvoir reconnaître toute sorte de motif sur une image indépendamment de son emplacement. Toutefois, l'un des inconvénients majeurs est qu'elle ne prend en compte que des informations locales à l'image au fur et à mesure, mais aussi le temps d'apprentissage pouvant être assez conséquent sur de grosses bases de données.

Q1.3

Après chaque couche de *convolution*, il est habituel d'appliquer une couche de *pooling*, non seulement cela réduit la dimension de la couche de sortie radicalement, par conséquent réduit la complexité des calculs, mais il permet également d'être robuste à certaines translations.

Q1.4

Traiter une image de plus grande taille que celle initialement prévue est possible, d'autant plus que seuls le nombre de *channels* en entrée et en sortie ainsi qu'éventuellement le *padding*, le *stride* et la *taille du kernel* sont nécessaires pour appliquer une convolution. Cela dit, un problème se poserait au niveau des couches *fully-connected* car nous aurions des tailles variantes d'une image à l'autre. Pour palier à ce problème, la technique du *Global Average Pooling* pourrait être utilisée à la place des *fully-connected* layers.

Q1.5

Une couche de *fully-connected* peut en effet être vue comme une convolution de *taille de kernel k* égale à la dimension spatiale de la *feature map* en entrée, ainsi qu'un nombre de filtre équivalent au nombre de neurones en sortie de la couche *fully-connected*.

Q1.6

Dès lors, en procédant au remplacement des couches *fully-connected* par des convolutions, nous pouvons accepter des images de taille supérieure, dont la forme est sa dimension, soit un tenseur, et l'intérêt, de pouvoir justement avoir un réseau dynamique, capable de traiter des images de taille variable.

Q1.7

Rappelons que le principe de la convolution est l'application de différents filtres sur différentes parties d'une image, dont les sorties, aussi appelées les *features maps*, correspondent pour chacune, un ensemble de neurones, à savoir un pour chaque case. Supposons un *kernel* de taille 3, lors de la première couche de convolution, la taille des *receptive fields* est donc 3×3, le nombre de pixels impliqués. Les neurones de la seconde couche de convolution qui s'en suit auront alors 5×5 comme taille de *receptive fields*. Plus nous avançons dans les couches profondes plus cette taille augmente. L'idée est que les premières couches de notre réseau vont détecter des formes simples telles que des contours, puis des formes plus complexes telles que des objets dans des couches plus profondes.

PARTIE 2 - APPRENTISSAGE FROM *scratch* DU MODÈLE

2.1 ARCHITECTURE DU RÉSEAU

Q2-1.8

Pour conserver les mêmes dimensions spatiales en sortie, nous choisirons un*padding* et un *stride* respectivement égal à 2 et 1.

Q2-1.9

Puis, pour réduire les dimensions spatiales d'un facteur 2, nous paramétrons le *max pooling* d'un *padding* de 0 et d'un *stride* de 2.

Q2-1.10

Plus nous avançons dans le réseau, plus le nombre de poids à apprendre est conséquent. Et comme nous l'avons mentionné auparavant, la différence entre le nombre de poids à apprendre dans une couche *fully-connected* et une couche de convolution tape à l'oeil.

Layers	Output size	Number of weights	Total	Total cumul
conv1	$32 \times 32 \times 32$	$3 \times 5 \times 5 \times 32$	2400	2400
pool1	$16 \times 16 \times 32$	0	0	2400
conv2	$16 \times 16 \times 64$	$32 \times 5 \times 5 \times 64$	51200	53600
pool2	$8 \times 8 \times 64$	0	0	53600
conv3	$8 \times 8 \times 64$	$64 \times 5 \times 5 \times 64$	102400	156000
pool3	$4 \times 4 \times 64$	0	0	156000
fc4	1000	$64 \times 4 \times 4 \times 1000$	1024000	1180000
fc5	10	1000×10	10000	1190000

Q2-1.11

Nous avons en tout 1900000 poids à apprendre, tout en disposant d'un total de 50000 images en *Train*, ce qui est relativement petit par rapport au nombre de poids. Nous risquons dès lors clairement le *surapprentissage*.

Q2-1.12

Rappelons que pour l'approche *BoW + SVM*, nous avions autant de paramètres que de mots visuels fois le nombre de classes. Soit 1000 le nombre de mots visuels comme nous l'avions choisi dans l'exercice correspondant, nous aurions alors eu $1000 * 10$, soit 10000 paramètres. Les réseaux de neurones convolutifs ont alors bien plus de paramètres à apprendre.

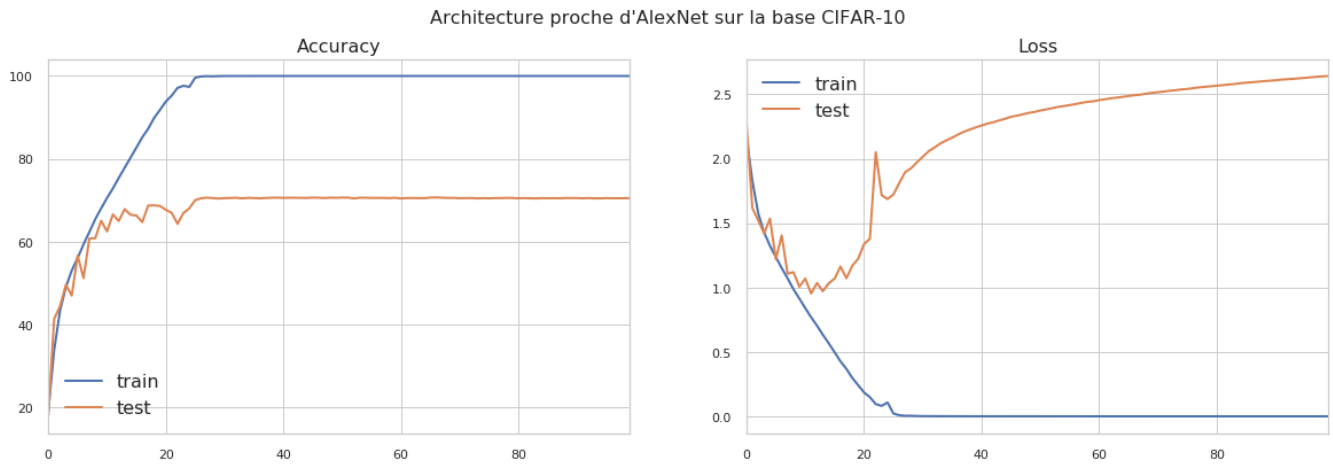
2.2 APPRENTISSAGE DU RÉSEAU

Q2-2.14

En période d'*apprentissage*, nous devons appliquer la *backpropagation* afin de corriger l'erreur. Cela implique de faire appel à la méthode de *backward()* sur la *loss*, puis de mettre à jour les paramètres en appelant la fonction *step()*. En période d'*évaluation* ou de *test*, nous sommes dispensés de toutes ces pratiques. Telle est la différence.

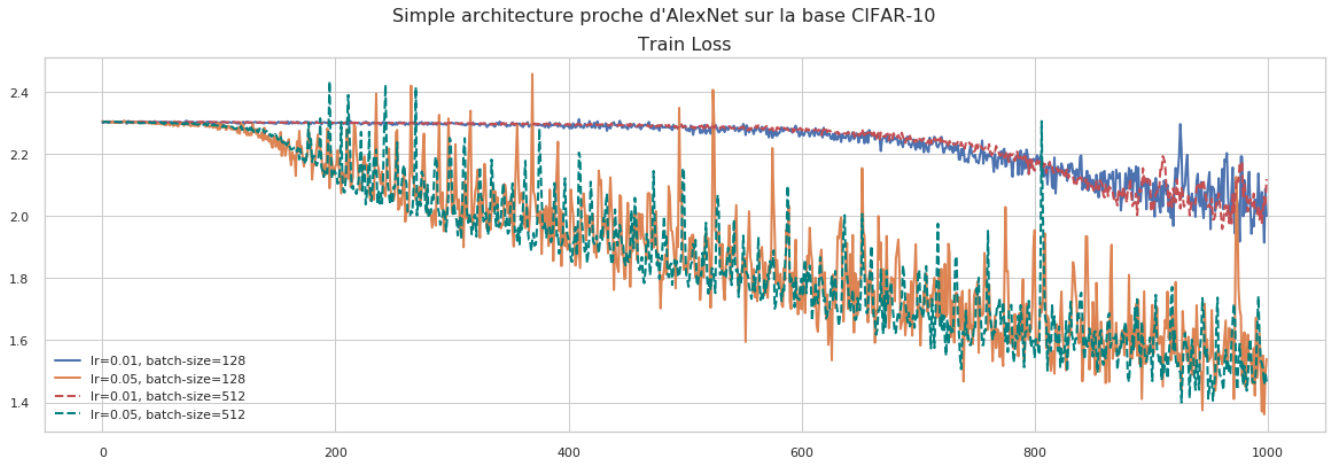
Q2-2.15

Pour toutes les expériences qui suivent, nous avons choisi d'entraîner nos différents modèles et améliorations sur 100 *epochs* pour ainsi s'assurer de leur convergence.
Ci-dessous, nous avons implémenté l'architecture demandée (proche d'*AlexNet*) et avons également utlisé la base CIFAR-10. Notre réseau atteint bien la convergence.



Q2-2.16

L'effet du *learning rate* et de la taille du *mini-batch* ont un impact extrêmement important sur les résultats de notre réseau. En effet, un *learning rate*, pourrait causer la divergence de notre réseau, notamment nous pouvons nous retrouver avec des *NaNs* au bout d'un certain nombre d'*epochs*. Ce dernier apprendrait alors pas. Inversement, avec un *learning rate* plus petit, notre réseau tarderait à converger et pourrait tout aussi se coincer sur un minima local dans le cadre d'un problème convexe. Nous remarquons sur l'expérience menée ci-dessous la différence de signal et de convergence quand le *learning rate* est paramétré à 0.01, 0.1, respectivement.
Quant à la taille du *mini-batch*, en définir une grande, stabilise la *loss* dès le début de l'*apprentissage* mais demande beaucoup de ressources mémoire. Tandis qu'une petite taille rendrait alors la *loss* beaucoup plus instable. Nous pouvons observer les différences sur la figure ci-bas. Avec des tailles de *mini-batch* définies respectivement à 512 et 128.



Q2-2.17

Notons que l'erreur au début de l'apprentissage est celle engendrée par une première initialisation aléatoire des poids de notre réseau.

Q2-2.18

En observant les courbes obtenues à la *Q*2. – 2.15, nous constatons immédiatement que la *loss* en *test* augmente au bout d'une vingtaine (20) d'*epochs*. Nous nous heurtons alors en l'occurrence à un phénomène de *sur-apprentissage*.

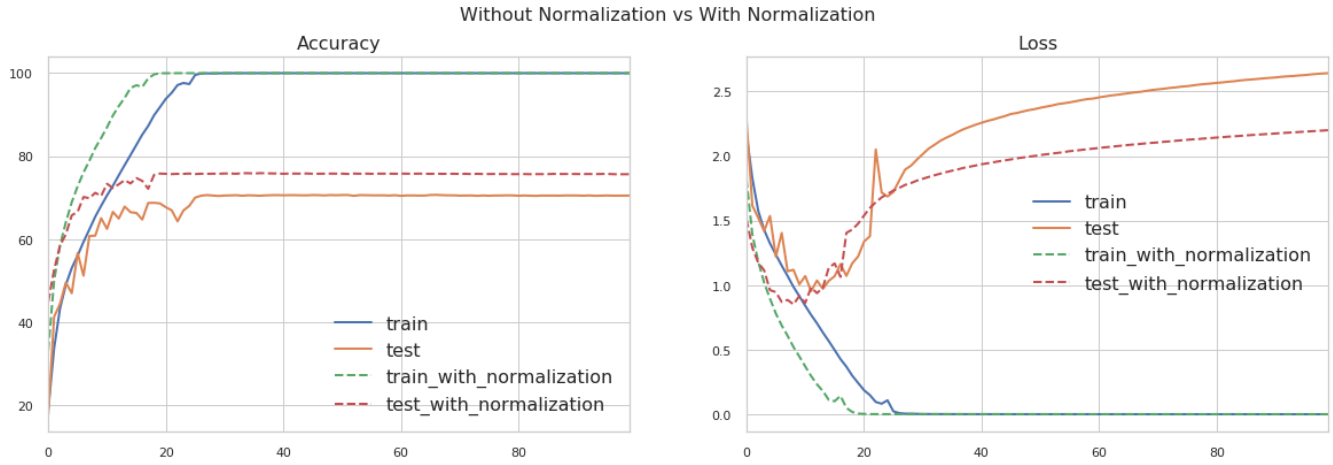
PARTIE 3 - AMÉLIORATION DES RÉSULTATS

Pour venir à bout du phénomène énoncé ci-dessus, nous allons testé plusieurs techniques usuelles dans le but d'améliorer les performances de notre réseau.

3.1 NORMALISATION DES EXEMPLES

Q3-1.19

En ajoutant la normalisation, non seulement nous avons une meilleure *accuracy* en *Train* et en *Test*, la *loss* en *Train* diminue plus vite mais notre modèle sur-apprend moins également en observant la *loss* en *Test*.



Q3-1.20

Il convient en effet de souligner qu'il est important de calculer l'image moyenne que sur l'ensemble de *Train*, à défaut d'obtenir des résultats biaisés par nos données de *Test*. Il est crucial de retenir que les données en *Test* ne doivent servir qu'exclusivement au *Test*, ni à un *hyper-paramétrage*, chose faite sur les données de *validation*, ni à un apprentissage quelconque.

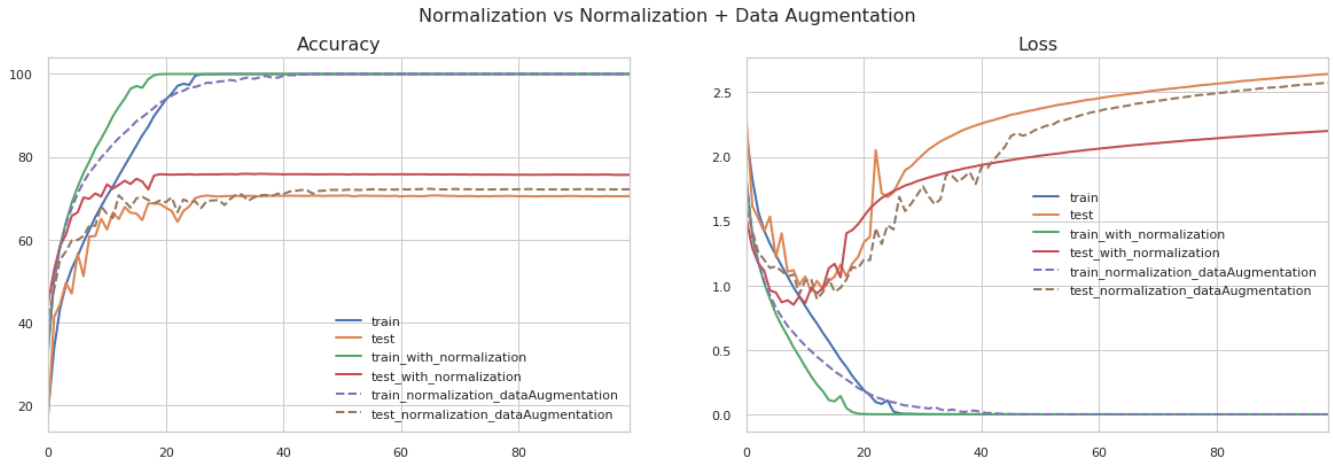
Q3-1.21 **BONUS**

La normalisation *ZCA* consiste à décorréler les données de manière à ce que leur matrice de covariance $\sum$ soit la matrice d'identité. Cela permet de réduire la corrélation entre les différentes channels *RGB* et aussi d'avoir que des *zéros* (0) à l'endroit où nous avions auparavant des pixels très corrélés entre eux.

3.2 Augmentation du nombre d'exemples d'apprentissage par *data augmentation*

Q3-2.22

En appliquant la technique de *data augmentation*, nous remarquons que la *loss* en *Test* remonte toujours, toutefois moins vite par rapport aux résultats précédents. Outre cela, nous ne remarquons de nette amélioration.



Q3-2.23

Cette approche par symétrie horizontale n'est pas utilisable sur toute sorte d'images. Il faut que la symétrie horizontale de l'image soit assez représentative de l'image en question, c'est à dire que pour certains types d'images, notamment des chiffres, nous en perdons complètement la sémantique, tandis que pour certaines, ce choix peut s'avérer judicieux, tels que des objets (une bouteille, un carton).

Q3-2.24

Comme nous l'avons indiqué ci-haut, l'application de la symétrie horizontale sur certains types d'images peut générer des résultats incohérents, telle est alors la limite de cet usage. Quant au *cropping*, cette technique peut s'avérer non suffisante à cause des valeurs de *padding* représentant le vide. Ce n'est pas toujours satisfaisant. D'autres types de complétions existent, telle que la symétrie.

Q3-2.25 **Bonus**

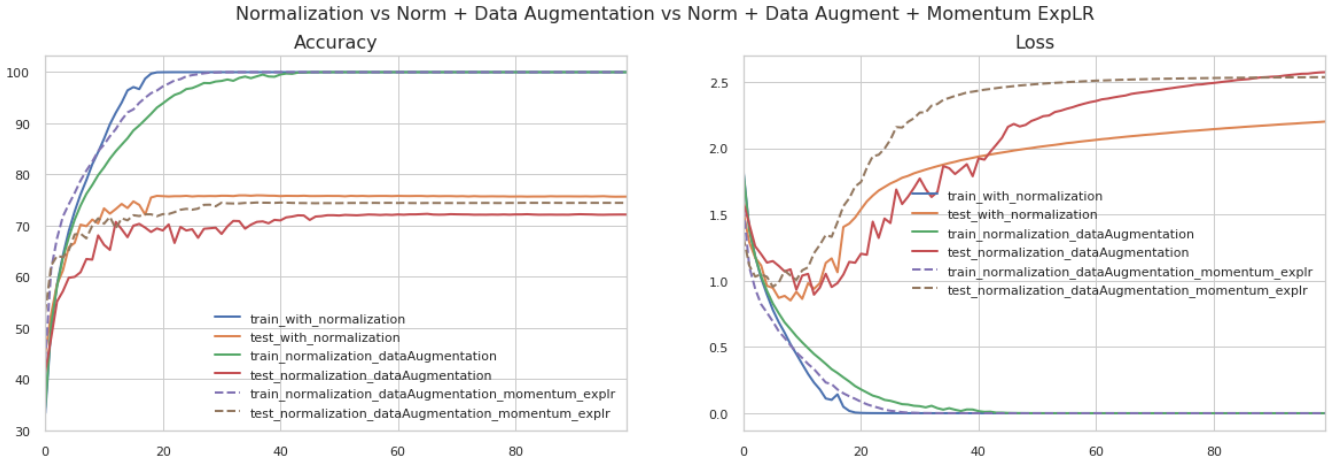
Il existe alors d'autres types de techniques de *data augmentaion*, tels que :

- Ajout de différents niveaux de changement d'intensités à l'image, éclairage, saturation, bruit, etc... ;
- Ajout de différentes rotations à l'image pour rendre notre réseau robuste à la rotation ;
- Suppression partielle d'une image.

3.3 VARIANTES SUR L'ALGORITHME D'OPTIMISATION

Q3-3.26

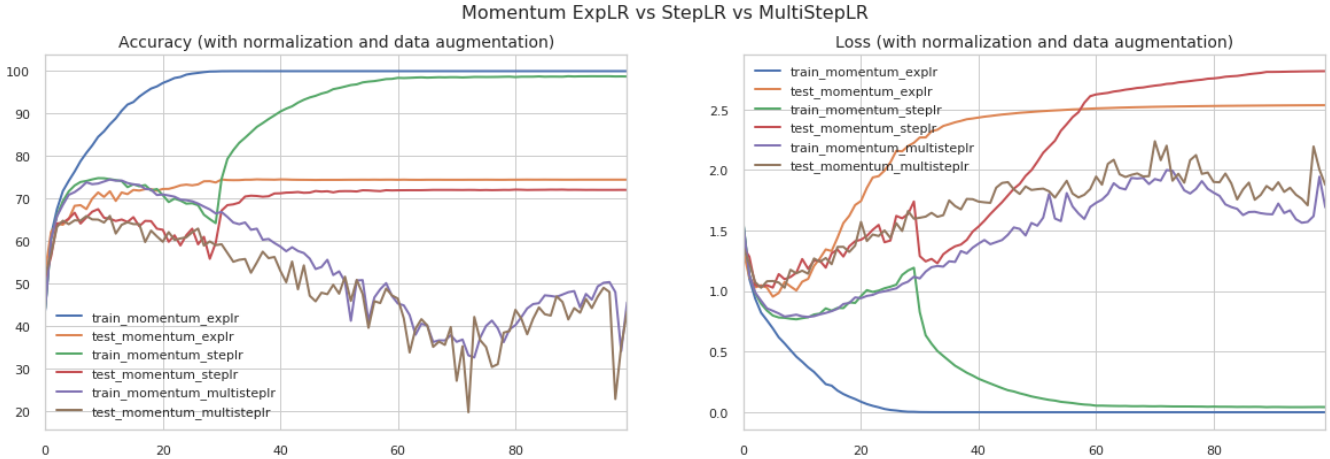
L'*accuracy* en *Train* et en *Test* augmente légèrement plus vite durant les premières *epochs*. Cependant, nous ne remarquons pas d'améliorations quant à sa *loss* par rapport aux résultats précédents. La vraie différence réside en effet dans la stabilité de l'*accuracy* et de la *loss* tout au long des *100 epochs*.



Q3-3.27

Rappelons que l'un des problèmes de la descente de gradient *mini-batch* était la faible progression vers le minimum local. Ł'ajout du *momentum* permet alors d'accélérer la descente de gradient vers la bonne direction en ajoutant un paramètre γ au vecteur de la mise à jour du pas de temps précédent au vecteur de mise à jour actuel. De plus, le *learning rate scheduler* réduit la taille du *learning rate* après chaque *epoch*, ce qui nous permet d'approcher un *optimum* le plus vite possible, puis stabiliser les oscillations de la *loss* tout au long de l'*apprentissage*.

Q3-2.28 **BONUS**



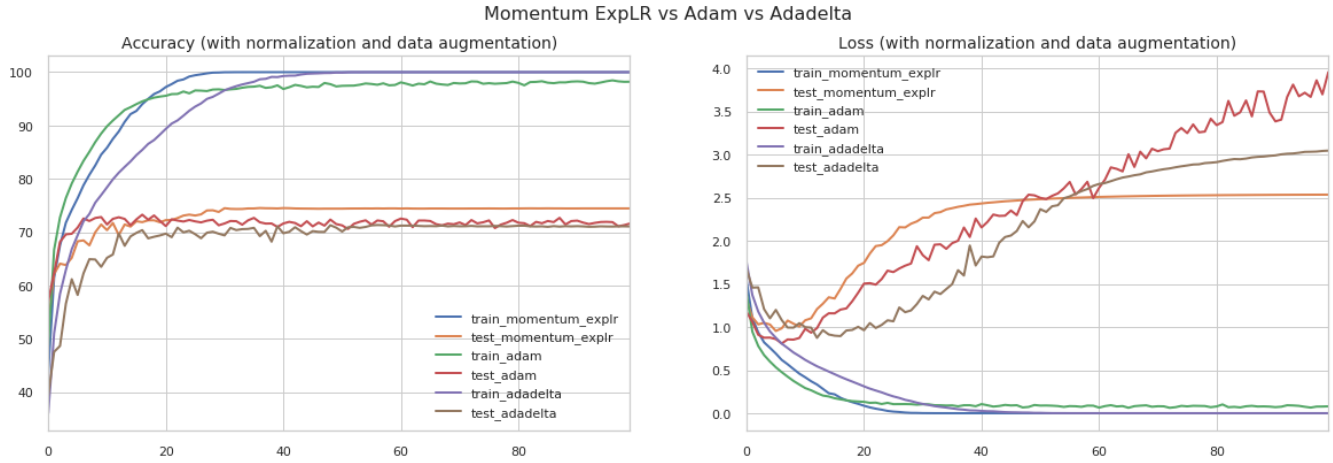
Plusieurs autres stratégies de planification du *learning rate* existent, nous en avons testé deux autres le *StepLR* puis le *MultiStepLR* consistant toutes les deux à réduire le leanring rate après un certain temps ou *epochs*. Puis nous les avons comparé à la technique *ExpLR*, utilisée ci-dessus.

Nous pouvons remarqué sur l'image ci-dessus que la meilleure méthode reste le *learning rate scheduler ExpLR*. Les deux autres, plus particulièrement le *MultiStepLR* tend d'ailleurs à *sous-apprendre* d'emblée. Nous concluons alors qu'elles ne sont pas adaptées à ce problème.

Par ailleurs, nous avons aussi testé d'autres techniques qui ne requièrent pas que l'on définisse manuellement le *learning rate*, ce sont *Adam* et *Adadelta*.

— **Adaptive Moment Estimation** (*Adam*) : est une technique similaire au *Adagrad*, *Adadelta* ainsi qu'au *RMSprop*. Elle est une combinaison du *RMSprop* et le *momentum*. En plus de stocker la décadence exponentielle de la moyenne des carrés des gradients passés, elle stocke également celle de la moyenne des gradients passés, respectivement m_t et v_t.

— **Adadelta** : est une extension d'*Adagrad* qui réduit la décroissance exponentielle du *taux d'apprentissage* due à l'accumulation de la somme des carrés des gradients. Elle restreint simplement la fenêtre de cette accumulation des gradients antérieurs à un paramètre fixe *w*.

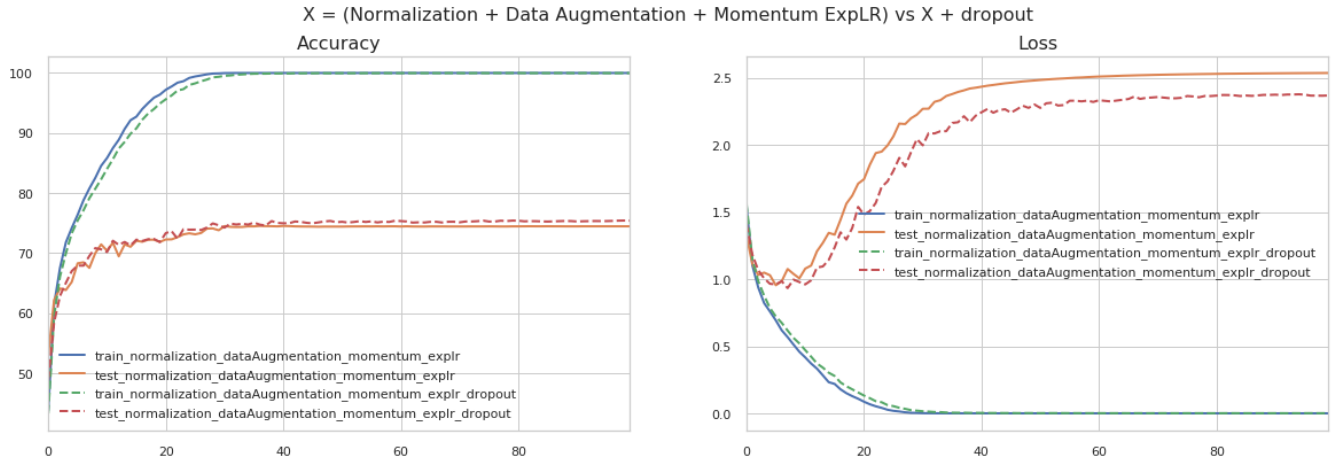


En les comparant aux résultats précédents notamment au *momentum* avec *learning rate scheduler ExpLR*, nous remarquons qu'*Adam* a une meilleure *accuracy* dès les premières *epochs*, cependant, en *Train*, il n'atteint pas les 100%, raison pour laquelle il sur-apprend peut être moins que le *momentum* avec *expLR*. Plus ou moins pareillement, *Adadelta* converge moins lentement en *accuracy* que ses deux (2) homologues et dispose des meilleures résultats quant au *sur-apprentissage* de la *loss*.

3.4 RÉGULARISATION DU RÉSEAU PAR *dropout*

Q3-.29

Le DropOut, tout comme les techniques précédentes, présente également des résultas qui semblent améliorer la loss en *Test*, pour mieux combattre le *sur-apprentissage*.



Q3-4.30

La régularisation est une méthode visant à lutter contre le *sur-apprentissage*. Donc à mieux généraliser nos modèles. Elle consiste le plus souvent, à la mise à néant de certains poids, ce qui permet donc de supprimer certains features, réduire la dimensionalité, c'est le cas notamment de la régularisation L_1, encore appelée *Lasso*. Ou encore plutôt que de complètement mettre à annuler certains poids, d'autres types de régularisation se contentent de les réduire au plus bas, de sorte à quand même utiliser ces *features* aussi peu soit-il. C'est le cas de la régularisation L_2, ou *Ridge*.

Q3-4.31

L'intuition est que cette technique permet de généraliser au mieux la performance des différents neurones. Eteindre certains neurones selon une probabilité p aléatoirement, pousse ceux activés à travailler beaucoup plus et à être indépendant de leurs voisins. Ainsi ils deviennent chacun plus robuste et ont un plus fort impact à la décision.

Q3-4.32

L'hyper-paramètre de la couche *dropout* est tout simplement la probabilité de désactivation d'un neurone durant un tour d'apprentissage. S'il est trop grand, alors nous risquons de sous-apprendre, tandisque s'il est trop petit, nous n'observerons pratiquement pas de valeur ajoutée.

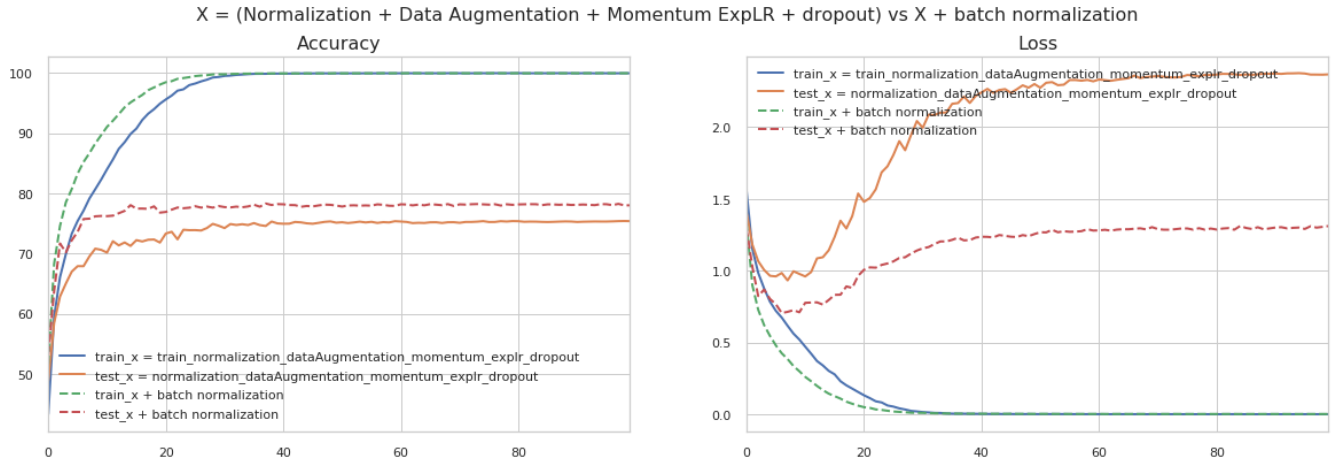
Q3-4.33

La différence de comportement de la couche *dropout* entre l'*apprentissage* et l'*évaluation* réside dans le fait que contrairement à la période d'*apprentissage*, le *dropout* est désactivé en évaluation. Toutefois, notons qu'en ce faisant, il nous incombe de réduire les sorties d'un facteur p. Le réseau étant entraîné à avoir des entrées plus faibles, réactiver le *dropout* engendrerait un signal p fois plus grand.

3.5 UTILISATION DE *batch normalization*

Q3-4.34

Cette technique qui est la *batch normalization* est certainement celle qui empiriquement concède le plus gros gain, en *accuracy* et en *loss*.
La performance augmente plus vite en *Train* ainsi qu'en *Test* et on observe d'ailleurs une légère amélioration en *Test*. La *loss* en *Train* diminue plus vite et est plus stable tout comme celle en *Test* où nous observons une diminution importante du *sur-apprentissage*.



Cependant, il est judicieux de préciser qu'en utilisant cette *technique*, nous augmentons la complexité des calculs.

ANNEXE