Université de Sorbonne

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

Rapport TMEs 3-6

Réalisé par :

Ghada Ferjani
Nada Haj Salah

M2 DAC

Année universitaire : 2020/2021

Chapter 1

Introduction aux réseaux de neurones

1.1 Formalisation mathématique

1.1.1 Jeu de données

Question 1

Afin d'éviter l'overfitting tout en convergeant vers le meilleur modèle possible, il est préférable d'utiliser non pas deux jeux de données seulement (apprentissage et évaluation) mais trois (apprentissage, validation et évaluation). En effet ,l'ensemble d'apprentissage est utilisé pour ajuster les paramètres d'un modèle. Un modèle ajusté est par la suite utilisé pour prédire les réponses aux observations dans un second ensemble de données appelé ensemble de données de validation. L'ensemble de validation est utilisé d'une part pour évaluer de façon non biaisé le modèle ajusté , ce qui permettera d'avoir une estimation non optimiste de l'erreur, et d'autre part pour réajuster les hyperparamètres du modèle. On choisi par la suite le modèle qui a obtenu les meilleurs résultats avec l'ensemble de validation. L'ensemble de test permet à la fin d'évaluer l'ajustement final du meilleur modèle choisi.

Question 2

Plus le nombre d'exemples est petit , plus l'analyse est difficile, mais plus il y en a, plus le besoin de mémoire informatique est élevé et plus longue est l'analyse des données.

1.1.2 Architecture du réseau (phase forward)

Question 3

Les fonctions d'activation présentent plusieurs intêrets . D'abord, elles permettent la non linéarité des combinaisons de transformations . La conséquence directe de leur ab-

sence est que le réseau ne peut représenter efficacement que des données linéairement dépendantes (et donc on limite l'utilisation de notre intelligence artificielle) . Ensuite,elles permettent d'éviter les grands calculs et de réduire la consommation excessive de mémoire vu qu'elles sont dans la plupart du temps bornées (tanh, softmax ..) Enfin, elles sont partout différentiables . Cette propriété permet de créer des optimisations basées sur les gradients.

Question 4

$n_x = 2$, $n_y = 2$, $n_h = 4$. $n_x = 2$ est la dimension des observations de la dataset. Dans le cas d'une régression linéaire par exemple , on aura autant de neurones d'entrées que de features . $n_y = 2$ est la dimension de la valeur prédite , elle est égale au nombre de classes , dans le cas d'une multi-class multilabel classification (logistic regression avec softmax), et est égale à 1 s'il s'agit une multi-class classification monolabel. $n_h = 4$ la dimension de la couche cachée . Chaque neuronne supplémentaire permet de prendre en compte des profils spécifiques des neurones d'entrée. Un nombre plus important permet donc de mieux coller aux données présentées mais diminue la capacité de généralisation du réseau. Il serait donc préferable d'avoir un nombre de neurones inférieur ou égale à la dimension de l'input layer.

Question 5

$\hat{y}$ est y prédite par le modèle , y et la vrai valeur du target correspondant à une ligne d'observations

Question 6

On utilise une fonction softmax d'activation dans le but de mettre en compétition toutes les classes possibles en sortie dans le cas d'une classification multiclass multi label et de rapprocher la valeur de y_k prédite de $P(K|x, w)$. sum=1

Question 7

- $\tilde{h_i} = \sum_{j=1}^{2} W_h ij * x_j + b_h i \; \forall i \in 1..4$
- $h_i = tanh(\tilde{h_i}) \; \forall i \in 1..4$
- $\tilde{y_i} = \sum_{j=1}^{4} W_y ij * hj + b_y i$
- $\hat{y}_i = Softmax(\tilde{y_i})$

1.1.3 Fonction de coût

Question 8

- La perte d'entropie croisée augmente à mesure que la probabilité prédite diverge de l'étiquette réelle. Ainsi, à mesure que la probabilité prédite du vrai target approche de 1 , la perte logarithmique diminue lentement.

Question posée lors du TP : N'est-ce pas un problème que $\hat{y}_i$ pourrait être 0 pour un $y_i = 1$? Cela signifierait que nous avons un très mauvais classificateur, bien sûr mais aussi un loss infini . Est-ce que ça s'écraserait simplement? Le modèle que nous avons choisi (activation softmax à la fin) ne donne-t-il jamais la probabilité 0 pour la classe correcte?

Réponse
Oui. C'est un problème. Cela correspond à la situation où votre modèle pense qu'une classe a une probabilité d'occurrence nulle, et pourtant la classe apparaît dans la réalité. En conséquence, la "surprise" de votre modèle est infiniment grande: votre modèle n'a pas tenu compte de cet événement et a maintenant besoin d'une infinité de bits pour l'encoder. C'est pourquoi vous obtenez l'infini comme entropie croisée. Pour éviter ce problème, vous devez vous assurer que votre modèle ne fait pas d'hypothèses irréfléchies sur le fait que quelque chose est impossible alors que cela peut arriver. En réalité, les gens ont tendance à utiliser des fonctions sigmoïdes ou softmax comme modèles d'hypothèse, qui sont suffisamment conservateurs pour laisser au moins une chance pour chaque option. Si vous utilisez un autre modèle d'hypothèse, c'est à vous de le régulariser (c'est-à-dire "lisser") afin qu'il ne fasse pas l'hypothèse de zéro là où il ne devrait pas.

- Pour l'erreur quadratique , plus $\hat{y}_i$ est proche de y_i $\forall i$ plus la MSE est faible

Question 9

Pour le problème de classification, la cross entropy est la mesure de performance la plus adaptée et c'est pour plusieurs raisons :

- **Justification théorique**
  Dans le cadre de l'estimation du maximum de vraisemblance, le but de l'apprentissage automatique est de maximiser la vraisemblance de nos paramètres compte tenu de nos données, ce qui équivaut à la probabilité de nos données compte tenu de nos paramètres. Or maximiser la vraissemblance des données compte tenu de nos paramètres revient à minimiser la cross entropy:

  Soit Ł$(\theta|D) = P(D|\theta)$ où θ est le vecteur des paramètres à optimiser et D est le jeu de données . comme le logarithme est une fonction monotone, maximiser la vraisemblance équivaut à minimiser la log-vraisemblance négative de nos paramètres compte tenu de nos données.
  Donc $argmax_\theta L(\theta|D) = argmin_\theta(-log(P(D|\theta)))$
  $argmax_\theta L(\theta|D) = argmax_\theta P(D|\theta)$
  $= argmax_\theta \prod_{(x,y)\in D} P(x,y|\theta)$
  $= argmin_\theta \sum_{(x,y)\in D} -log(P(x,y|\theta))$
  $= argmin_\theta \sum_{(x,y)\in} -log(P(x,y|\theta)$, where i où i est l'indice de la catégorie correcte
  $= argmin_\theta \sum_{(x,y)\in D} -\sum_{j=1}^{n} y_j log(P(x,y|\theta))$
  $= argmin_\theta \sum_{(x,y)\in D} -\sum_{j=1}^{n} y_j log(\hat{y}_j)$

- **L'utilisation du log est pratique quand il s'agit de probabilité**

  Comme démontré dans le point ci-dessus, afin de calculer la probabilité du modèle, nous devons calculer une probabilité conjointe sur chaque exemple de l'ensemble de données. Cela implique de multiplier toutes les probabilités . Ce produit π devient très petit et peut converger rapidement vers 0 ce qui pose problème au niveau de la mise à jour des paramètre. Cependant, ce problème peut être évité avec les log probabilités car, par la règle de produit des logarithmes, nous pouvons transformer le produit π des probabilités à l'intérieur du logarithme en une somme de logarithmes.

  Les avantages mathématiques de l'erreur quadratique moyenne sont particulièrement évidents dans son utilisation pour analyser les performances de la régression linéaire , car elle permet de partitionner la variation d'un ensemble de données en variation expliquée par le modèle et en variation expliquée par le hasard. Erreur quadratique moyenne. Sa puissance au carré permet également d'amplifier l'erreur.

1.1.4 Méthodes d'apprentissage

Question 10

- Batch gradient descent

  Idéale pour les problèmes convexes. Rapide et stable car il descend directement vers le minimum de la fonction coût. Cependant, cet algorithme utilise l'ensemble des données d'entrainement à chaque étape de mise à jour des paramètres du problème . De ce fait, il est extrement lent sur les jeux d'entrainement très volumineux

- Descente de gradient stochastique
  Comme la SGD ne choisit à chaque étape qu'une observation prise au hasard dans l'ensemble d'entrainement et calcule les gradients en se basant uniquement sur cette seule observation, cette algorithme est beaucoup plus rapide que les deux qui précedent puisqu'il n'a que très peu de données à manipuler à chaque itération . Cela permet donc de l'entraîner sur des jeux de données de grande taille. Cette algorithme peut être implementé comme un algorithme hors-mémoire. Par contre, étant donné sa nature stochastique , cet algorithme est beaucoup moins régulier qu'une descente de gradient ordinaire : au lieu de décroitre doucement jusqu'à atteindre le minimum , la fonction coût va effectuer des sauts vers le haut et vers le bas et ne décroîtra qu'en moyenne. Au fil du temps, elle arrivera très près du minimum , mais une fois là elle continuera à effectuer des sauts aux alentours sans jamais s'arrêter . Par conséquent lorsque l'algorithme est stoppé , les valeurs finales des paramètres sont bonnes mais pas optimales.

- Mini Batch gradient descent
  Le prinicpal avantage par rapport à la descente de gradient stochastique, c'est qu'on améliore les performances du fait de l'optimisation matérielle des opérations

matricielles . La progression de l'algorithme dans l'espace des paramètres est moins désordonnée que dans le cas de la descente de gradient stochastique , tout particulièrement lorsque les mini-lots sont relativement grands : par conséquent, la descente de gradient par mini-lots aboutira un peu plus près du minimum qu'une descente de gradient stochastique. Par contre, elle aura plus de mal à sortir d'un minimum local dans le cas des problèmes où il existe des minimums locaux .

Question 11

Le learning rate est un hyperparamètre. Si le taux d'apprentissage est trop petit, l'algorithme devra effectuer un grand nombre d'iterations pour converger et prendra beaucoup de temps . Inversement, si le taux d'apprentissage est trop élevé, on risque de dépenser le point le plus bas et de se trouver de l'autre côté , peut etre même plus haut qu'avant . Ceci pourrait faire diverger l'algorithme.

Question 12

L'algorithme de la backpropagation utilise les éléments de la couche précédente pour calculer les gradients. Pour l'approche naïve, on calcule l'ensemble des gradients de toutes les couches de manière indépendante. L'algorithme de la backpropagation est moins coûteux en compléxité.

Question 13

Les fonctions appliquées au niveau de chaque couche du réseau de neuronne doivent être différentiables

Question 14

$l = -\sum_i^n y_i * log(\frac{e^{\tilde{y}_i}}{\sum_j^n e^{\tilde{y}_j}}) = -\sum_i^n y_i * (log(e^{\tilde{y}_i}) - log(\sum_j^n e^{\tilde{y}_j})$
$= y_i * \tilde{y}_i - y_i * log(\sum_j^n e^{\tilde{y}_j}) = y_i * \tilde{y}_i - *log(\sum_j^n e^{\tilde{y}_j})$

Question 15

$\frac{\partial l}{\partial \tilde{y}_i} = \frac{\partial -\sum_i^n y_i * log(\hat{y}_i)}{\partial \tilde{y}_i} = \frac{\partial -\sum_i^n y_i * log(\sum_j^n e^{\tilde{y}_j}}{\partial \tilde{y}_i} = -y_i + \frac{e^{\tilde{y}_i}}{\sum i^n e^{\tilde{y}_j}} = -y_j + \hat{y}_i$
Donc $\nabla_y l = \begin{pmatrix} \hat{\mathrm{y}}_1 - y_1 \\ ... \\ \hat{\mathrm{y}}_n - y_n \end{pmatrix}$

Question 16

$\frac{\partial l}{\partial W_{\tilde{y}}, ij} = \sum_k \frac{\partial}{\partial \tilde{y}_k} * \frac{\partial \tilde{y}_k}{\partial W_y, ij} = \sum_k \nabla_y l * \frac{\partial \sum_j W_{y,kj} * h_j + b_k}{\partial W_y, ij} = \sum_k (\hat{y}_k - y_k) * h_j$
$\frac{\partial l}{\partial b_i^{\tilde{y}}} = \sum_k \nabla_y l * \frac{\partial \tilde{y}_k}{\partial b_i^{\tilde{y}}} = \sum_k \nabla_y l * 1$

1.1.5 Question 17

$$\frac{\partial l(y,\tilde{y})}{\partial b_{y,j}} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y}_k}$$
$$\frac{\partial \tilde{y}_k}{\partial b_{y,j}} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y}_i}$$
$$\frac{\partial \tilde{y}_i}{\partial b_i} = \frac{\partial l(y,\tilde{y})}{\partial \tilde{y}_i}\mathbf{1}$$
$$\frac{\partial \tilde{y}_i}{\partial \tilde{h}_i} = \sum_j \frac{\partial \tilde{y}_k}{\partial \tilde{h}_j}$$
$$\frac{\partial h_j}{\partial \tilde{h}_i} = \frac{\partial \tilde{y}_k}{\partial h_i}\frac{\partial h_i}{\partial \tilde{h}_i} = W_{k,i}\left(1 - h_i^2\right)$$
$$\frac{\partial h_i}{\partial \tilde{h}_i} = \mathbf{1} - \tanh\left(\tilde{h}_1^2\right) = \mathbf{1} - h_i^2$$

1.2 Contextualisation du TP

Dans cette partie,nous étudions le cas d'une classification binaire sur le jeu de données CirclesData. Comme le montre la figure 1,il est clair que cet ensemble de données n'est pas linéairement séparable. Le modèle de classification qui sera abordé sera un réseau de neuronnes dont l'architecture est la suivante :
Une couche cachée: constituée d'une transformation affine avec biais et d'une fonction d'activation tanh
Une couche de sortie:constituée d'une transformation affine avec biais et d'une fonction d'activation softmax

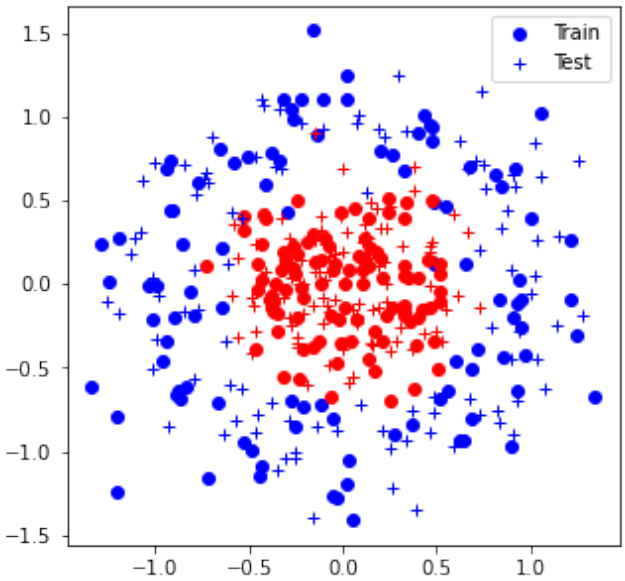

Figure 1: Circles Dataset

L'implementation du réseau de neuronnes a été réalisée à travers 4 versions différentes présentées ci dessous.

1.2.1 Implémentation manuelle

On effectue un split (0.5,0.5) du dataset de taille 400 . L'intialisation des paramètres du réseau de neuronnes a été effectuée avec une distribution normale de moyenne 0 et d'écart type 0.3 . Avec un nombre de batchs égale à 10 , une couche cachée de taille 10 et un learning rate égale à 0.03 , on aboutit au bout de 200 epochs à une accuracy de 95% en train et en test avec un loss = 0.18 . On pourra également utiliser la méthode du early stopping pour éviter le surajustement lors de l'apprentissage avec $\epsilon = 0.05$. On remarque que l'accuracy maximale et atteinte à partir de la 130ème itération. Les petites fluctuations de l'accuracy sont dûes au caractère stochastique de la descente de gradient. Ci dessous les figure 2,3 & 4 qui illustrent respectivement l'évolution de l'accuracy en train et en test l'évolution du cross entropy loss en train et en test et la visualisation du plan séparateur des données binaires

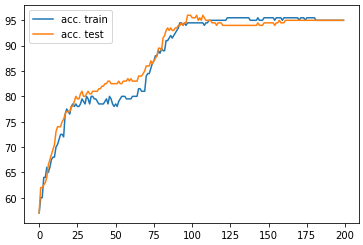

Figure 2: L'évolution de l'accuracy au cours du temps

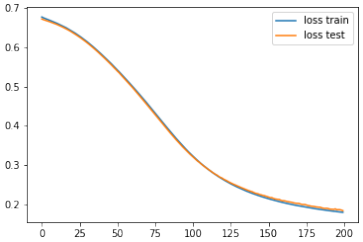

Figure 3: L'évolution du cross entropy loss au cours du temps

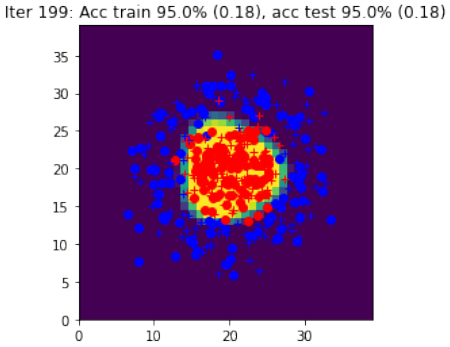

Figure 4: Visualisation de l'hyperplan séparateur

1.2.2 Simplification du backward avec torch.autograd

Dans cette partie la fonction forward est inchangée par rapport à la partie précédente et la fonction backward n'est plus utilisé grâce à l'autograd. En autograd,il suffit de déclarer les Tensors pour lesquels les gradients doivent être calculés avec le requires grad=True puis d'appeler la fontion backward sur la loss. En conservant les mêmes hyperparamètres de la première partie, on aboutit au bout de 200 itérations à une accuracy de 96.5% en train et de 95.5 % en test et un cross entropy loss de 0.17 en train et de 0.19 en test .

Ci dessous les figures 4,5 & 6 qui illustrent respectivent les accuracy , les courbes d'apprentissage et de test ainsi qu'une visualisation de l'hyperplan séparateur des données binaires .

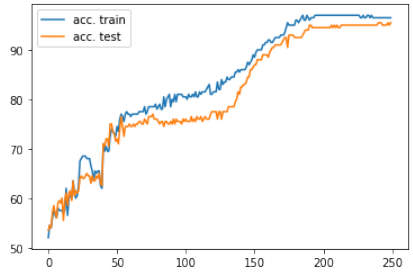

Figure 5: L'évolution de l'accuracy au cours du temps

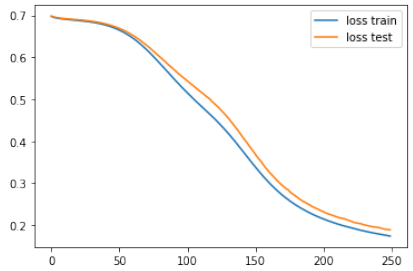

Figure 6: L'évolution du cross entropy loss au cours du temps

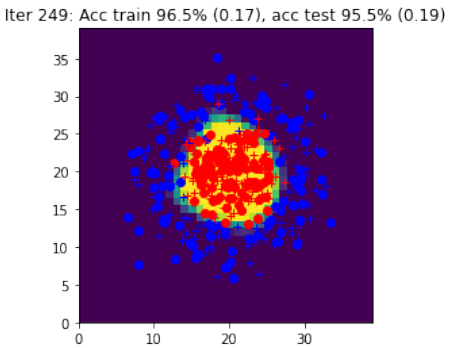

Figure 7: Visualisation de l'hyperplan séparateur

1.2.3 Simplification du forward avec torch.nn

Dans nn.Sequential de la méthode init params, les nn.Module stockés à l'intérieur sont connectés en cascade . Dans le cadre de ce tp, le réseau de neuronnes est composé des blocs suivants, dans l'ordre suivant: Linear $->$ tanh $->$ Linear $->$ Softmax .
Avec torch.nn, on est monté un cran au niveau de l'automatisation utilisée. Les classes de conteneur de ce module peuvent déterminer par eux-mêmes sur quels tenseurs doivent on appliquer la différentiation. l'approche séquentielle semble très attrayante. Cependant, pour les réseaux de neurones non triviaux tels qu'un autoencodeur variationnel, l'approche Module est beaucoup plus facile à utiliser. En effet, avec Module () on doit définir une méthode forward () mais avec Sequential () une méthode forward () implicite est définie.
nn.Sequential utilise le mécanisme par défaut de PyTorch pour initialiser les pondérations et les biais. Ce n'est presque jamais une bonne approche.Donc à mesure que les réseaux deviennent plus complexes, l'approche séquentielle perd rapidement son avantage de simplicité par rapport à l'approche module().

Ci dessous les les figures 7,8 & 9 qui illustrent respectivent les accuracy , les courbes d'apprentissage et de test ainsi qu'une visualisation de l'hyperplan séparateur des données binaires .

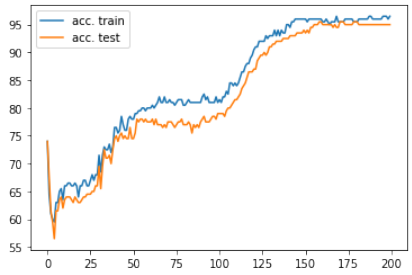

Figure 7: L'évolution de l'accuracy au cours du temps

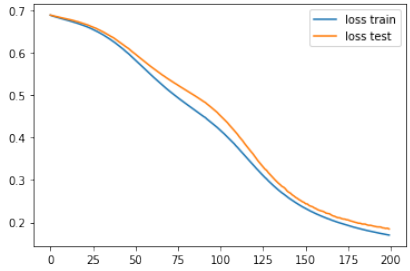

Figure 8: L'évolution du cross entropy loss au cours du temps

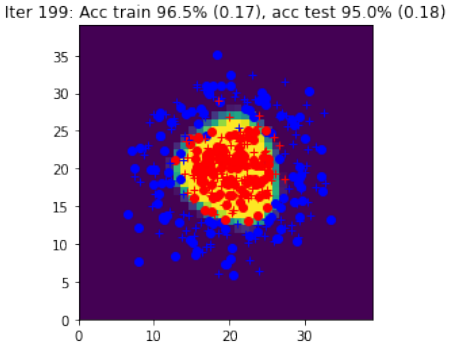

Figure 9: Visualisation de l'hyperplan séparateur

Chapter 2

Réseaux convolutionnels pour l'image

2.1 Introduction aux réseaux convolutionnels

Question 1
La taille de sortie:

$n'_x = \left\lfloor \frac{x-k+2p}{s} + 1 \right\rfloor$ $n'_y = \left| \frac{y-k+2p}{s} + 1 \right\rceil$ $n'_z = 1$ Le nombre de poids :$(k \times k \times 1 \times z)+1$

Le nombre de poids à apprendre en Fully Connected : $x \times y \times z \times n'_x \times n'_y$
Question 2
L'avantage de la convolution par rapport au Fully connected est que le nombre de poids à apprendre est beaucoup plus inférieur qu'avec des couches fully-connected. Ceci est dû au fait que les CNNs utilise une réplication des poids : Un détecteur de feature qui est utile dans une partie de l'image est probabluement utile dans une autre partie de l'image. Ainsi que la sparsity of connections . L'inconvenient des CNN est que les dépendances sont restreintes aux informations locales , extraction locale des features
Question 3
L'intérêt du pooling spatial est de réduire la dimension de l'image afin de réduire la complexité du réseau et le coût de calcul tout en conservant l'information la plus importante et fournit une invariance par petites translations.
Question 4
Si on prend une image plus grande, on peut toujours appliquer les couches de convolution car le nombre de poids à apprendre ne change pas. Par contre les couches Fully connected ne peuvent pas s'appliquer correctement car le nombre de poids à apprendre dépend des dimensions de l'entrée. Ainsi on peut appliquer la partie du réseau où on n'utilise pas des couches fully-connected.

Question 5
Les couches Fully Connected peucent être remplacées par des convolutions de deux manières différentes:

- la taille du kernel correspond à la taille des entrés et le nombre de filtres correspond aux dimensions de sortie.
- Nous pouvons concaténer les entrées en images 1*1 avec n_x channels où n_x la taille de l'input size et faire n_y convolutions où n_y est la taille de l'ouput size du fully connected layer . On rappelle que chaque noyau a également n_x channels

Question 6
Si toutes les couches sont des convolutions, on peut tout calculer en ayant une sortie de plus grande dimension. Le résultat sera équivalent au réseau avec une couche Fully connectect. L'interet est de pouvoir utiliser des images de taille quelconque.
Question 7
Pour la sortie de la première couche de convolution, la taille du receiptive field = la taille du filtre de la convolution de la première couche= k1.
Pour la sortie de la deuxième couche de convolution, la taille du receiptive field = (k2-1) * s1 + k1 , avec s1= sortie de la couche 1.
La taille du champ récepteur augmente avec la profondeur .

2.2 Apprentissage from scratch du modèle

Question 8
D'aprés la question 1, conserver en sortie les mêmes dimensions spatiales,on trouve:

$$s = p \times \frac{2}{n-1} + \frac{n-k}{n-1}$$
$$p = s \times \frac{n+1}{2} - \frac{n-k}{2}$$

On prend
$s = 1 => p = \frac{k-1}{2}$
Question 9
Pour qu'un max pooling, pour réduire les dimensions spatiales d'un facteur 2 , on utilise un stride de 2, un kernel de taille 2 et aucun padding.
Question 10
Taille de sortie | Nombre de poids
conv1 : (32,32,32) | (5*5*3+1)*32=2432
pool1 : (16,16,32) | 0
conv2 : (16,16,64) | (5*5 × 32 + 1) × 64=51264
pool2 : (8,8,64) | 0
conv3 : (8,8,64) |(5*5 ×64+ 1) × 64=102464
pool3 : (4,4,64) | 0
fc4 : 1000 | 1024*1000=1024000
fc5 : 10 | 1000*10=10000

Question 11
Le nombre total de poids à apprendre est : 2432+51264+102464+ 1024000 + 10000 = 1 190 160. Alors que on a 50 000 images en apprentissage de taile 32×32×3 ce qui est

petit par rapport au nombre de poids. Il y a un risque de surapprentissage.

Question 12

Avec l'approche BoW + SVM, le nombre de paramètres est égale au nombre de mots visuels fois le nombre de classes. Soit 1000 mots visuels , on a donc alors 1000 10= 10000 paramètres. Avec les réseaux de neurones convolutifs , il y plus de paramètres à apprendre.

Question 14

La différence entre la façon de calculer la loss et l'accuracy en train et en test est qu'on utilise Module.eval() pendant l'évaluation afin de désactiver plusieurs couches comme torch.nn.BatchNormNd et torch.nn.DropoutNd .

On n'utilise le loss.backward() que pendant l'apprentissage afin de calculer le gradient de la loss par rapport à l'ensemble de l'apprentissage et de mettre à jour les hyper-parmètres .

En phase de test on utilise le processeur graphique (GPU) pour exécuter des calculs généraux à la place du processeur (CPU) , ceci en mettant cuda=cuda .

Question 15

on a utilisé 30 epochs.

Question 16

Batch size est l'un des hyperparamètres les plus importants à régler . Plusieurs souhaitent souvent utiliser une taille de lot plus importante pour entraîner leur modèle car ça permet d'accélérer les calculs grâce au parallélisme des GPU. Cependant,il est bien connu qu'une taille de lot trop importante entraîne une mauvaise généralisation.D'autre part, l'utilisation de lots plus petits permettait une convergence plus rapide vers de "bonnes" solutions.

Learning rate: une valeur trop faible peut entraîner un long processus d'apprentissage qui pourrait se bloquer, tandis qu'une valeur trop élevée peut entraîner un apprentissage trop rapide d'un ensemble de poids sous-optimal ou un processus d'apprentissage instable.

Il y a une règle "Linear scaling rule" : lorsque la taille du mini lot est multipliée par k, multiplier le taux d'apprentissage par k. Bien que que les lots de grande taille sont moins performants, on peut combler ce défaut en augmentant le taux d'apprentissage.

Question 17

L'erreur au début de la première époque correspond à l'erreur avec les poids initialisés aléatoirement. Au cours de l'apprentissage, le modéle fait la mise à jour de ces poids afin de minimiser l'erreur.

Question 18

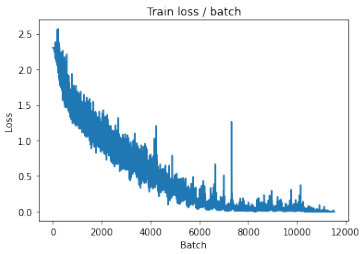

Figure 1: loss de chaque itération en train

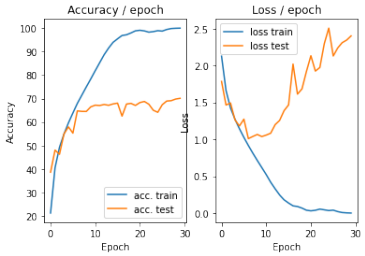

Figure 2: Courbe de loss et d'accuracy avec les données CIFAR.

D'apres les figures ci dessus, on remarque que le modele en apprentissage, la loss diminue vers 0,le réseau a correctement appris. Par contre, pendant le test, le loss a commencé de se diminuer mais aprés certains epochs, le loss a augmenté : overfitting . Sur la figure 1, on remarque qui'il y a des oscillations à cause de la variance des batchs. Parfois, des batchs sont faciles à traiter alors que d'autres sont plus difficiles .

2.3 Améliorations des résultats

2.3.1 Normalisation des exemples

Question 19
En ajoutant la normalisation, on remarque que nous avons une meilleure accuracy en Train et en Test, la loss en Train diminue plus vite mais nous sommes encore dans un cas de sur-apprentissage en observant la loss en Test.

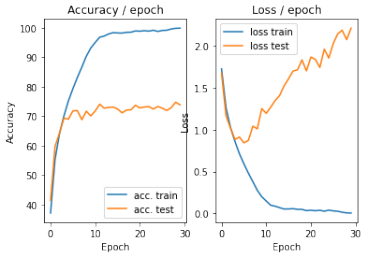

Figure 4: Courbe de loss et d'accuracy en utilisant la normalisation.

Question 20
On calcule la normalisation uniquement sur l'ensemble d'apprentissage pour ne pas avoir des résultats biaisés par l'ensemble de Test. En effet, les données dans l'ensemble Test ne doivent pas etre exploitées.
Question 21
La normalisation ZCA permet de réduire la corrélation entre les différentes channels RGB.

2.3.2 Data augmentation

Question 22
On remarque que avec le data augmentation, le résultat de notre modèle est amelioré: le modéle sur-apprend moins que précedement.

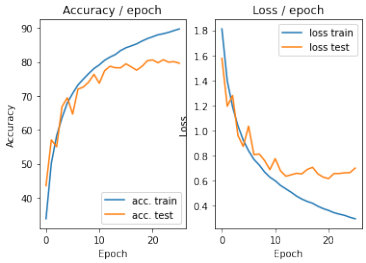

Figure 3: Courbe de loss et d'accuracy en utilisant Data Augmentation.

Question 23
Cette approche par symétrie horizontale n'est pas utilisable sur toute sorte d'images. Il y a des images qui perdent la sémantique comme : les chiffres , les lettres, tandis que pour certaines, on peut utiliser cet approche sans probléme comme les images des objets , des animaux..
Question 24 Il y a plusieurs limites pour le data augmentation:
-Le jeu de données n'a pas gagné en diversité car le sujet de la photo n'a pas changé
- Changer un aspect sémantique de la donnée : Il faut que le résultat désiré de la prédiction ne change pas malgré la transformation appliquée.
Question 25
La data augmentation peut consister en: des rotations, déformations, recadrages, changements de couleurs, ajout de bruit ou autres.

2.3.3 Variantes sur l'algorithme d'optimisation

Question 26
On remarque que la stabilité de l'apprentissage est ameliorée. Aussi, l'entrainment est plus lent que le modéle précedent, ce qui est expliqué par le learning rate scheduler.

Figure 3: Courbe de loss et d'accuracy avec momentum et learning rate scheduler .

Question 27
Le momentum ou SGD avec momentum est une méthode qui permet d'accélérer les vecteurs de gradient dans les bonnes directions, ce qui conduit à une convergence plus rapide: pour chaque étape, pour calculer le nouveau gradient avant la mise à jour, on ajoute le gradient calculé à l'étape précédente pondéré par une valeur autour de 0.9 .
Aussi, le learning rate scheduler réduit la taille du learning rate durant l'apprentissage, ce qui permet de converger vers un optimum plus vite et ensuite continuer l'entraînement en stabilisant les oscillations de la Loss.
Question 28
Il existe plusieurs stratégies de planification du learning rate comme le StepLR et le MultiStepLR qui permettent de réduire le leanring rate après un certain temps ou epochs.
Plusieurs variantes de SGD existent:
Adam : le principe est : tant que le gradient est dans la même direction que ceux précédents, on va accélérer la vitesse (dite d'apprentissage) à laquelle on descend la courbe (on met à jour les paramètres).
AdaMax : Une variante d'Adam, le principe est d'utiliser à la norme l2 de la variance, et on prend le max entre la variance estimée et la norme des gradients (moins sensible aux hyper-paramètres) .
AMSGrad : une amélioration de ADAM , le principe est de calculer le max entre la variance estimée à cette étape et le maximum des variances précédentes.

2.3.4 Régularisation du réseau par dropout

Question 29
On remarque que le modèle sur-apprend beaucoup moins, le dropout est efficace pour la régularisation. On remarque aussi, lorqu'on augmenté le paramétre de dropout, les performances obtenues en test sont meilleures que celles obtenues en train.
En 26 epochs, la valeur de l'accuracy est diminuée par rapport au modéle précedent. En effet, un réseau avec du dropout nécessite plus d'epochs pour s'entraîner , mais les epochs sont plus rapides à calculer parcequ' il y a moins de poids.

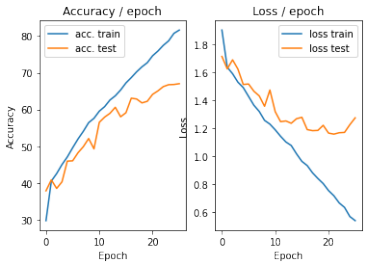

Figure 3: Courbe de loss et d'accuracy en utilisant Dropout=O.1.

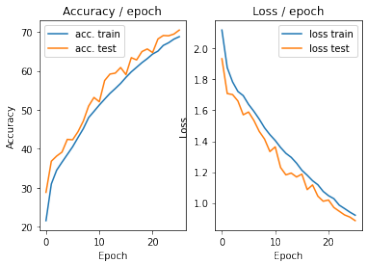

Figure 3: Courbe de loss et d'accuracy en utilisant Dropout=O.5..

Question 30
C'est une technique clé de machine learning qui vise à limiter le surapprentissage (overfitting) et à contrôler l'erreur de type variance pour aboutir à de meilleures performances.
Lors de l'apprentissage d'un modèle, la régularisation permet d'imposer une contrainte pour favoriser les modèles simples au détriment des modèles complexes. Cela permet de réduire l'erreur de type variance et d'améliorer la généralisation de la solution.
Question 31 Dropout consiste à désactiver certains neurones du modèle, et ce de façon aléatoire d'une même couche, qui ne contribuera donc ni à la phase de feedforward, ni à la phase de backpropagation. D'un point de vue du réseau, cela revient à instancier la valeur en sortie d'une fonction d'activation à 0.
Si des neurones sont retirés du réseau de manière aléatoire pendant la formation, d'autres neurones devront intervenir et gérer la représentation nécessaire pour faire des prédictions sur les neurones manquants. Le réseau apprend ainsi de multiples représentations internes indépendantes.

L'effet est que le réseau devient moins sensible aux poids spécifiques des neurones. Il en résulte un réseau capable de mieux généraliser et moins susceptible de faire un "overfiting" .

Question 32
L'hyper-paramètre de la couche dropoutp est la probabilité d'un neurone d'être désactivé. En général, il faut utiliser une petite valeur d'abandon de 20 à 50 % des neurones, 20 % constituant un bon point de départ. Une probabilité trop faible a un effet minimal et une valeur trop élevée entraîne un sous-apprentissage par le réseau.
Question 33
En apprentissage, la couche de dropout désactive aléatoirement une partie des neurones du réseau. Par contre , pendant la validation, la couche de dropout est desactivée, elle n'aura aucun effet sur le réseau.

2.3.5 Utilisation du batch normalization

Question 34
On remarque que le batch normalization permet d'accélèrer l'entraînement du modèle

: en 26 epochs, l'accuracy passe de 70% à 90%. Aussi, la loss en Train et en Test diminue plus vite et est plus stable. En effet, on ne normalise plus sur l'ensemble des données, mais uniquement sur les batchs. Ce qui permet à chaque couche d'apprendre indépendament des autres couches et éviter par la suite le sur-apprentissage.

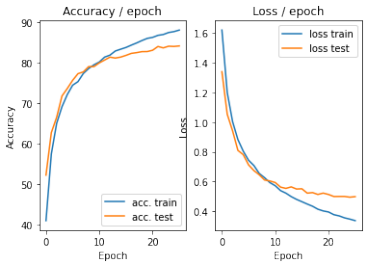

Figure 3: Courbe de loss et d'accuracy en utilisant Batch normalization

2.4 Resultats

Dans ce TP, on a amelioré l'architecure d'un réseau de neurones convolutionnel afin de lutter contre le problème de sur-apprentissage avec plusieurs méthodes:

- En normalisant les données, on obtient une meilleur accuracy mais le probléme de sur apprentissage persiste.
-On a utilisé la data augmentation en faisant un crop aléatoire et une symétrie horizontale en train et un crop centré en test . En effet, le nombre d'images est très faible par rapport au nombre de poids à apprendre, on a essayé de génerer de nouveux exemples en appliquant des transformations sur les images existants. Par la suite, le sur-apprentissage est beaucoup moins marqué. D'autres méthodes de data augmentation sont possible tel que la translation, la modification de l'éclairage , la rotation et les GANs conditionnels .
- Avec SGD et un learning rate scheduler, l'apprentissage est stabilisé, la convergence est plus rapide ainsi que l'accuracy.
- Le dropout est une approche de régularisation dans les réseaux de neurones qui aide à réduire l'apprentissage interdépendant entre les neurones. Le dropout force un réseau de neurones à apprendre des fonctionnalités plus robustes qui sont utiles en conjonction avec de nombreux sous-ensembles aléatoires différents des autres neurones. Il fait diminuer le temps d'apprentissage. Avec dropout (p=0.5), le sur-apprentissage est beaucoup moins marqué dans les exemples ci dessus.
- Avec le batch normalization, l'entraînement du réseau est accélèré.
Au début, 26 epochs sont suffisants pour que le modéle converge, mais au final, il faut plus que 50 epochs. La valeur d'accuracy est augmentée (environ 80% en test et environ 94% en train pendant 50 epochs) .

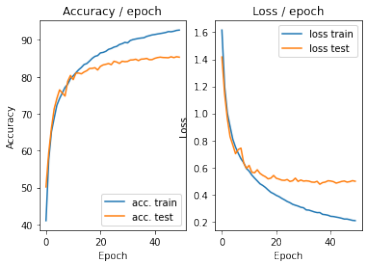

Figure 3: Courbe de loss et d'accuracy avec 50 epochs.

On remarque que tous ces améliorations q'on a fait, les résultats obtenues ne sont pas les meilleures. Le nombre d'éxemples en apprentissage reste toujours faible par rapport au nombre de poids à apprendre donc pour obtenir des meilleures performances, il faut travailler sur un dataset plus grand comme ImageNet