Polytechnique Montréal
Département de Génie Informatique
INF8225 I.A.:tech. probabilistes et d'apprentissage

TP2

ELAKHRASS, Mohamed Ali 1847744
REDDY, Philippe 1746145

Travail remis à:
Christopher J. Pal

27 février 2020

Table of Contents

1. Partie 1

1.1 Pseudocode

Dans le pseudocode, on suppose un réseau de neurones fully connected avec D=300 neurones pour chaque hidden layer.

Algorithm 1 Backpropagation Algorithm

procedure BACKPROPAGATION($state$)
  $X \leftarrow vecteurdetailleDx1$
  $\theta \leftarrow matricedetailleDxDpluslebiaisdetailleDx1$
  $L \leftarrow Nombredehiddenlayers + 1(output)$
  for $i \leftarrow 1, NbEpochs$ **do**
    $A[0] \leftarrow X.T$
    for $l \leftarrow 1, L-1$ **do** ▷ Hidden Layers
      $Z[l] \leftarrow \theta[l] \cdot A[l-1] + b[l]$
      $A[l] \leftarrow Relu(Z[l])$
    $Z[L] \leftarrow \theta[L] \cdot A[L-1]$
    $A[L] \leftarrow Softmax(Z[L])$
    $\nabla_Z \leftarrow A[L] - Y.T$ ▷ Debut du backpropagation
    $\nabla_\theta[L] \leftarrow \nabla_Z \cdot A[L-1].T$
    $\nabla_{A_{prev}} \leftarrow \nabla_\theta[L].T \cdot \nabla_Z$
    for $l \leftarrow L-1, 1$ **do** ▷ Hidden Layers
      $\nabla_Z \leftarrow \nabla_{A_{prev}} * relu - prime(Z[l])$
      $\nabla_\theta[l] = \nabla_Z \cdot A[l-1].T$
      $\nabla_{A_{prev}} \leftarrow \nabla_\theta[l].T \cdot \nabla_Z$
    for $l \leftarrow 1, L+1$ **do** ▷ Update des poids
      $\theta[l] \leftarrow \theta[l] - \alpha * \nabla_\theta[l]$
  return θ ▷ Les poids finaux

1.2 Optimisation

Dans cette partie, il est demandé de coder l'algorithme de la question A. Une fois que ce dernier est développé, il faut le comparer a une implémentation avec le logiciel PyTorch.

La première étape a donc été de faire le pré-traitement des données. Il y a donc eu standardisation des données. Un encodage "one hot" a aussi été appliqué sur les labels.

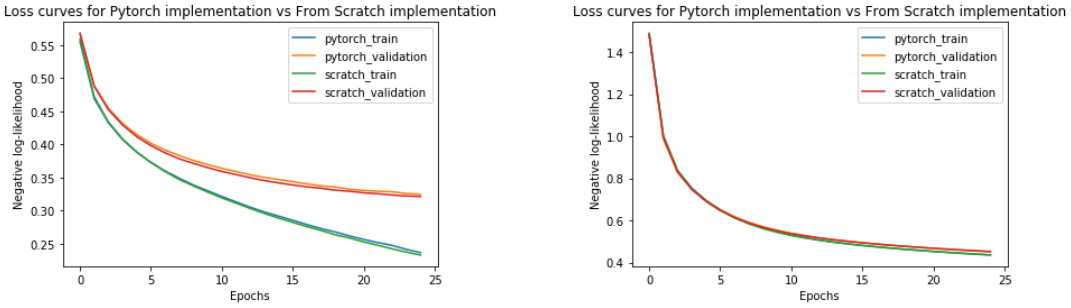
Par la suite, les deux codes ont été testés avec différents paramètres. Le tableau 1 compare l'accuracy sur les données de test et les données de validation.

Table 1.1: Comparaison de l'accuracy pour l'implémentation en python pur, par rapport à l'implémentation en pytorch

Implémentation from scratch					
Model	Hidden Layer Number	Layer nodes	Lr	Val Accuracy	Test Accuracy
python pure	2	300	0.01	0.877	0.885
python pure	3	300	0.01	0.871	0.882
python pure	4	300	0.01	0.876	0.885
python pure	2	300	0.001	0.837	0.839
PyTorch	2	300	0.01	0.886	0.796
PyTorch	3	300	0.01	0.883	0.782
PyTorch	4	300	0.01	0.88	0.781
PyTorch	2	300	0.001	0.84	0.803

On observe que les valeurs de l'accuracy sont quasiment identiques. Les différences peuvent être expliquées par les matrices de poids et le biais. En effet, il n'y a pas les même matrices dans les deux implémentations. Afin de réduire le plus possible ce problème, les valeurs de ces matrices ont été générées à l'aide du même intervalle.

De plus, pour vérifier que les deux implémentations aient le même comportement lors de l'entrainement, les graphiques de pertes (loss) en fonction des époques ont été tracé. Comme il est possible de voir dans la figure 1, on peut voir que l'entrainement a des tendances très similaires entre les deux implémentations. L'architecture de ces deux implémentaitons est une architecture avec deux hidden layers de 300 noeuds. Le learning rate est de 0.01 pour la figure a et de 0.001 pour la figure b. On remarque que même lorsque nous avons des hyperparamètres différents (le learning rate dans ce cas), les deux implémentations ont le même comportement. Fait intéressant, les courbes de validation du graphique a divergent plus des courbes d'entrainement que les courbes de validation du graphique b. Cela est simplement causé par le learning rate. En effet, on remarque que l'échelle de l'axe des y n'est pas la même pour les deux graphiques. Un learning rate plus grand fait en sorte que les pertes observées pour les premières époques d'entrainement sont sensiblement plus petites que pour un learning rate plus petit.



(a) learning rate= 0.01 **(b)** learning rate= 0.001

Figure 1.1: Graphiques de pertes pour les deux implémentations

2. Partie 2

Dans cette partie, plusieurs combinaisons de paramêtres et architectures différentes ont été testées, dans le but de se familiariser avec le module Pytorch. Plus précisément, deux types de réseaux de neurones ont été utilisés, soit des réseaux de neurones entièrement connectés et des réseaux de neurones convolutifs.

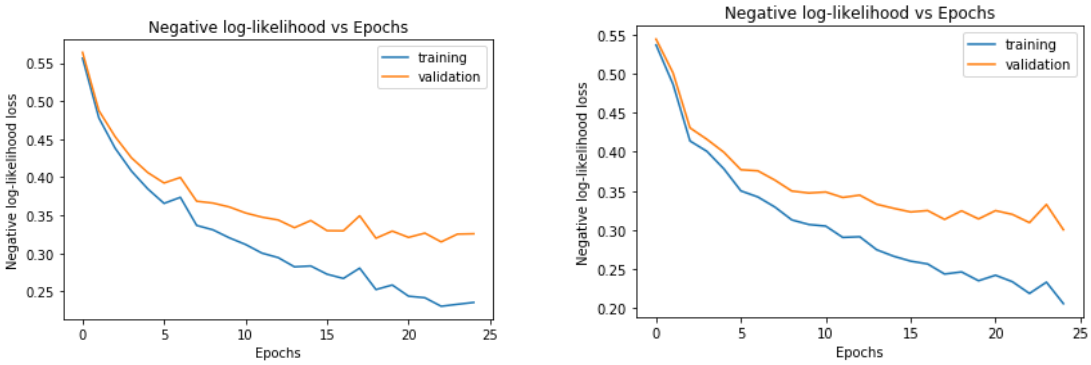
Pour les deux types d'architecture, différentes valeurs de learning rate (0.1,0.01,0.001) ainsi que de nombres d'époques (10,25,50) ont été testés. Cependant, pour les réseaux enitièrement connectés, 25 époques étaient suffisantes et continuer l'entrainement à 50 époques n'apportait pas de précision plus grande. Pour les réseaux convolutifs, c'est plutôt 10 époques qui était suffisantes. La taille de batch est également restée fixe à 64 exemples pour les deux types de modèles. Aussi, la fonction de perte utilisée dans les deux cas est la log-vraissemblance négative. Les autres paramètres testés diffèrent selon le type d'architecture.

Pour ce qui est du réseau de neurones entièrement connecté, les nombre de couches cachées qui a été exploré est de 1, 2 et 3 couches. De plus, la taille des couches testées étaient de 100, 300 et 800 noeuds. Enfin, différentes fonction d'activations ont été utilisée, soient "relu", "sigmoid" et "tanh". La table suivante décrit résultats obtenus pour les modèles testés:

Table 2.1: Accuracy obtenues pour les différentes combinaisons de paramètres et d'architecture testées dans le cas d'un réseau de neurones entièrement connecté

Fully Connected Neural Network						
Model	Hidden Layers	Nodes	Lr	Activation	Val Accuracy	TestAccuracy
1	2	300	0.01	relu	0.886	0.883
2	2	300	0.01	tanh	0.883	0.8747
3	2	300	0.01	sigmoid	0.831	0.821
4	1	300	0.01	relu	0.882	0.871
5	2	100	0.01	relu	0.881	0.878
6	2	800	0.01	relu	0.890	0.883
7	3	300	0.01	relu	0.886	0.876

On peut voir dans la table 2 que les deux meilleurs modèles obtenus sont les modèles 1 et 6, soit des modèles avec 2 couches cachées et avec respectivement 300 et 800 unités par couche. De plus, la fonction d'activation des deux modèles est "relu" et le learning rate est de 0.01. Il est intéressant de constater qu'augmenter le nombre de noeuds ne produit pas de meilleurs résultats sur notre ensembles de données, mais n'est pas non plsu nuisible à sa performance. Cependant, ce plus grand nombre de neurones implique un plus grand coût d'entrainement. Avec moins d'unités par couche, comme au modèle 5 (100 unités), on observe une petite baisse de précision. Enfin, le pire modèle est le modèle 3, qui utiliser la fonction d'activation "sigmoid".



(a) Courbes d'apprentissage du modèle 1 **(b)** Courbes d'apprentissage du modèle 6

Figure 2.1: Graphiques de pertes pour les modèle entièrement connectés 1 et 6

La figure 4 démontre les courbes de pertes pour les deux meilleurs modèles entièrement connectés obtenus, soient les modèles 1 et 6. On constate que dans les deux cas la perte commence aux alentour de 0.55, pour terminer autour de 0.25 après 25 époques d'entrainement. Il semble y avoir légèrement plus de divergence entre les courbes de validation et d'entrainement du modèle 6 par rapport au modèle 2, ce qui impliquerait peut être qu'il y a un peu plus tendance au surapprentissage lorqu'on augment la taille des couches. Cependant, cette différence est minime et les deux modèle ont des performances très similaires.

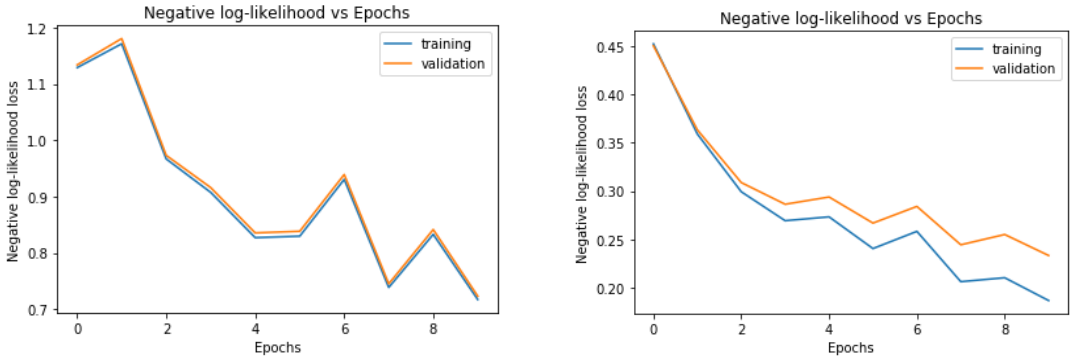
Dans le cas des réseaux de neurones convolutifs, les réseaux testés ont tous 4 couches de convolution en plus d'une couche de pooling par couche de convolution. Les paramètres explorés sont les fonctions d'activation ("relu","tanh","sigmoid"), la probabilité de dropout d'un noeud (0.1,0.5,0.9) ainsi que le learning rate (0.1, 0.01, 0.001). Le nombre d'époques d'entrainement est fixé à 10.

Table 2.2: Accuracy obtenues pour les différentes combinaisons de paramètres et d'architecture testées dans le cas d'un réseau de neurones convolutif

Convolutional Neural Network						
Model	Pooling type	Drop out	Lr	Activation	Val Accuracy	Test Accuracy
1	average	0.5	0.1	relu	0.862	0.8521
2	max	0.5	0.1	relu	0.869	0.8524
3	max	0.5	0.1	tanh	0.753	0.743
4	max	0.5	0.1	sigmoid	0.822	0.813
5	max	0.9	0.1	relu	0.101	0.100
6	max	0.1	0.1	relu	0.915	0.9034

Le tableau ci-dessus montre les résultats obtenus pour les réseaux convolutifs. On remarque que le modèle 6 est nettement supérieur aux autres, avec une précision de validation de 0.915. Ce résultat est supérieur à tous les modèles entièrement connectés essayé plus haut. De plus, ce résultat s'obtient avec 10 époques comparativement à 25 époques pour les réseaux entièrement connectés. Ceci indique que les réseaux convolutionnels sont plus adaptés que les réseaux entièrement connectés pour la tâche de classification d'image. De plus, il est intéressant de noter le résultat du modèle 5. Ce modèle a un dropout=0.9, ce qui signifie que le modèle a une très grande probabilité de ne pas considérer un noeud du réseau. De ce fait, le modèle apprend très peu et la précision obtenue est de seulement 0.101.

La figure suivante contient les courbes de pertes pour les deux meilleurs convolutionnels obtenus, soient les modèles 2 et 6. :



(a) Courbes d'apprentissage du modèle 2 **(b)** Courbes d'apprentissage du modèle 6

Figure 2.2: Graphiques de pertes pour les modèles convolutifs 2 et 6

Sur la graphique de gauche, on voit que les courbes sont très sacadées et oscillent beaucoup. Ceci est dû au fait que le taux d'apprentissage est assez élevé et que la probabilité de dropout est de 0.5. Pour le graphe de droite, on voit des coubres beaucoup plus lisses, ce qui est surement dû au droupout de 0.1, qui est en fait la seule différence architecturale entre les deux modèles. Ces deux graphiques démontrent donc bien l'importance du dropout sur l'entraînement. Enfin, on remarque que les pertes sont beaucoup plus élevées pour le modèle 2 que le modèle 6, à cause du drop-out qui ne permet pas au modèle 2 d'apprendre aussi vite que le modèle 6.

Pour conclure, suite à tous les modèles observés, on conlut que le meilleur modèle est le modèle convolutif 6, parce qu'il obtient des meilleurs performances que le modèle entièrement connecté 6, en plus de demander plus de deux fois moins d'époques d'entrainement. Cette conclusion vient valider les notions apprises en classes au sujet des réseaux convolutifs, qui sont particulièremenet bien adaptés à la classification d'images.