INF8225 TP2 H20
Date limite : 12h le 28 février 2019

1 Partie I (20 points)

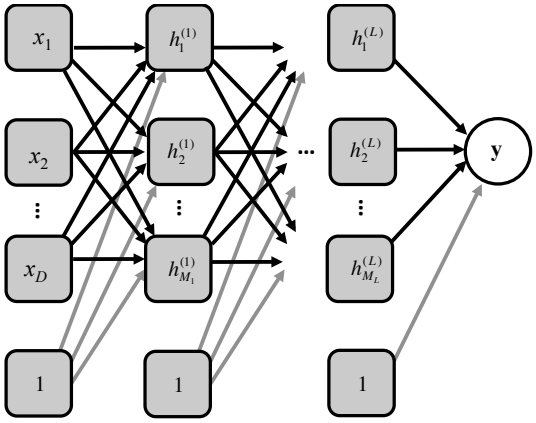

Figure 1: Un réseau de neurones.

Pour cette partie, vous pouvez travailler en groupes de 2, mais il faut écrire sa propre dérivation et soumettre son propre rapport. Si vous travaillez avec un partenaire, il faut indiquer leur nom dans votre rapport.

Considérons un réseau de neurones avec une couche d'entrée avec $D = 784$ unités, L couches cachées, chacune avec 300 unités et un vecteur de sortie $\mathbf{y}$ de dimension K. Vous avez $i = 1, .., N$ exemples. $\mathbf{y}$ est un vecteur du type *one-hot* – un vecteur de zéros avec un seul 1 pour indiquer quand la classe $C = k$ dans la dimension k. Par exemple, le vecteur $\mathbf{y} = [0, 1, 0, \cdots, 0]^T$ pour représenter la deuxième classe, et chaque vecteur continus $\mathbf{x}_i \in \mathbb{R}^{784}$ dans un ensemble d'apprentissage. La fonction de perte est donnée par

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{k,i} \log(f_k(\mathbf{x}_i)) \tag{1}$$

La fonction d'activation de la couche finale a la forme $\mathbf{f} = [f_1, ..., f_K]$ donné par la fonction d'activation *softmax*:

$$f_k(\mathbf{a}^{(L+1)}(\mathbf{x}_i)) = \frac{\exp(a_k^{(L+1)})}{\sum_{c=1}^{K}\exp(a_c^{(L+1)})}, \quad \mathbf{h}^{(l)}(\mathbf{a}^{(l)}(\mathbf{x}_i)) = \text{ReLU}(\mathbf{a}^{(l)}(\mathbf{x}_i)),$$

où $\mathbf{a}^{(l)}$ est le vecteur résultant du calcul de la préactivation habituelle $\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$, qui pourrait être simplifiée à $\boldsymbol{\theta}^{(l)}\hat{\mathbf{h}}^{(l-1)}$ en utilisant l'astuce de définir $\hat{\mathbf{h}}$ comme $\mathbf{h}$ avec un 1 concaténé à la fin du vecteur.

a) (10 points) Donnez le pseudocode incluant des *calculs matriciels—vectoriels* détaillés pour l'algorithme de rétro-propagation pour calculer le gradient pour les paramètres de chaque couche étant donné un exemple d'entraînement.

b) (10 points) Implémentez l'optimisation basée sur le gradient de ce réseau en Python, *sans utiliser une logiciel comme Pytorch pour obtenir vos gradients* et comparez-la avec le même réseau implémenté en utilisant une logiciel de haut niveau. Utilisez **Fashion MNIST** pour comparer les deux modèles. Une partie de votre note sera basée sur la façon dont vous comparez ces deux implémentations.

Part I – English version (20 points)

For this part, you may work in groups of 2, but you must write your own derivation and hand in your own report. If you work in a group, indicate the name of your partner in your report.

Consider a neural network with a single input layer having $D = 784$ units, L hidden layers, each having 300 units and a K dimensional output vector $\mathbf{y}$. You have $i = 1, .., N$ examples. $\mathbf{y}$ is a *one-hot* vector, or an all zero vector except for a single 1 in dimension k which indicates when the class $C = k$. For example, the vector $\mathbf{y} = [0, 1, 0, \cdots, 0]^T$ represents the second class. The input is a continuous vector $\mathbf{x}_i \in \mathbb{R}^{784}$. The loss function is given by

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{k,i} \log(f_k(\mathbf{x}_i)) \tag{2}$$

The final layer has a set of activation functions encoded by the vector $\mathbf{f} = [f_1, ..., f_K]$ given by the softmax function

$$f_k(\mathbf{a}^{(L+1)}(\mathbf{x}_i)) = \frac{\exp(a_k^{(L+1)})}{\sum_{c=1}^{K} \exp(a_c^{(L+1)})}, \quad \mathbf{h}^{(l)}(\mathbf{a}^{(l)}(\mathbf{x}_i)) = \mathrm{ReLU}(\mathbf{a}^{(l)}(\mathbf{x}_i)),$$

where $\mathbf{a}^{(l)}$ is the vector resulting from the calculation of the usual preactivation function $\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$, which could be simplified to $\boldsymbol{\theta}^{(l)}\hat{\mathbf{h}}^{(l-1)}$ using the trick of defining $\hat{\mathbf{h}}$ as $\mathbf{h}$ with a 1 concatenated at the end.

a) (10 points) Give pseudocode including detailed *matrix-vector calculations* for the backpropagation algorithm to calculate the gradient for the parameters of each layer given a training example.

b) (10 points) Implement the gradient based optimization of this network in Python, *without using a library like Pytorch to obtain your gradients* and compare it with the same network implemented using a high level library. Use **Fashion MNIST** to compare the two models. Part of your grade will be based on how you compare these two implementations.

Partie II (20 points)

Pour cette partie, vous pouvez travailler en groupes de 2. Chacun doit soumettre son propre rapport, mais pour cette partie les rapports doivent être identiques si vous avez travaillé en groupe. L'objectif de la partie II est de se familiariser avec la librairie Pytorch (ou Tensorflow + Keras, ou votre choix de logiciel). Vous allez expérimenter avec les données **Fashion MNIST**. Les données sont des images similaires à MNIST et consiste en 10 catégories de vêtements.

Exécutez des expériences avec différentes architectures, e.g. différents types et nombres de couches. Utilisez au moins un réseau de convolution et un réseau "fully connected". Utilisez un ensemble de validation pour sélectionner la meilleure architecture et utiliser l'ensemble de test seulement à la fin. Comparez vos résultats à l'aide de courbes d'apprentissage et de la log-vraisemblance. Vous pouvez vous référer au code disponible ici: `https://github.com/AlexPiche/INF8225/tree/master/tp2`, et aussi aux tutoriels pytorch ici: `https://github.com/MaximumEntropy/pytorch_notebooks`

Soumettez un rapport incluant les courbes d'apprentissage et le code de vos modèles.

Part II - English version (20 points)

For this part, you may work in groups of 2. Each person submits their own report, but for this part the reports have to be identical if you work in a group. The objective of Part II is to familiarize yourself with the Pytorch library (or Tensorflow + Keras, or your choice of other software). You will experiment with **Fashion MNIST**. This is a dataset of images similar to MNIST, but consisting of 10 different categories of clothing.

Perform experiments with different architectures; e.g. use different types and numbers of layers. Use at least one convolutional network and one fully connected network. Use a validation set to select the best architecture, and use the test set only at the end. Compare your results based on learning curves and log-probability. You can refer to code available here: `https://github.com/AlexPiche/INF8225/tree/master/tp2`, and also to pytorch tutorials here: `https://github.com/MaximumEntropy/pytorch_notebooks`

Submit a report including the learning curves and code of your models.