
**Département de Génie Informatique**

**et Génie Logiciel**

# INF8225

# I.A. : techniques probabilistes et d'apprentissage

## Laboratoire 3

## TP 3

# Objectif du laboratoire

La principale tâche du laboratoire est d'implémenter un réseau de neurones de type Perceptron multicouche en utilisant la librairie Theano. L'objectif du laboratoire est de se familiariser avec les librairies utilisant la différentiation symbolique pour le calcul du gradient des paramètres d'un réseau profond.

# Description des tâches

## Implémentation d'un réseau de neurones de type Perceptron Multicouche :

Un Perceptron multicouche est un type de réseaux de neurones dont l'information circule de la couche d'entrée vers la couche de sortie à travers le passage par les couches cachées. L'unité élémentaire d'une couche est appelée neurone. Dans un problème de classification, La couche de sortie est constituée de *C* neurones, tous connectés à la dernière couche cachée. Les neurones de la couche de sortie sont activés par la fonction softmax. La sortie du $j^{ème}$ neurone de la couche de sortie donnera un score pour la classe *j*. Ce score est interprété comme la probabilité que l'entrée ***x*** soit de cette classe *j*.

Comme on a vu au TP2 la fonction softmax est définie comme :

$$\begin{aligned} \boldsymbol{o}^s &= \text{softmax}(\boldsymbol{o}^a) \\ \boldsymbol{o}^s_k &= \frac{\exp(\boldsymbol{o}^a_k)}{\sum_{k'=1}^{m} \exp(\boldsymbol{o}^a_{k'})} \end{aligned}$$

Le réseau de neurones associe pour chaque entrée représentée par un vecteur ***x***, un vecteur de scores exprimant les probabilités d'appartenance de l'entrée ***x*** à chacune des classes. La probabilité, calculée par le réseau de neurones, qu'une observation ***x*** appartienne à la classe *y* est donc donnée par la $k^{ième}$ sortie. Ceci suggère d'utiliser la fonction de perte d'entropie croisée qui constitue le risque empirique. Elle est donnée par :

$$-\frac{1}{L}\sum_{i=l}^{L-1} \log(P(Y = y^i|x^i, \mathbf{W}, \boldsymbol{b}))$$

Dans ce TP vous n'êtes pas demandé d'implémenter le Perceptron multicouches en Matlab. Vous aurez à utiliser une librairie comme Theano[1] permettant d'évaluer des expressions mathématiques utilisant des variables multidimensionnelles appelées Tenseurs. Dans notre cas,

[1] http://www.deeplearning.net/software/theano/

Theano nous aidera à évaluer les expressions des paramètres du Perceptron multicouche en utilisant le mécanisme de dérivation symbolique permettant d'inférer un graphe computationnel exprimant les dérivés. Nous n'aurons pas alors à exprimer les gradients par nous-mêmes puisque c'est le graphe computationnel qui s'en charge.

Pour pouvoir implémenter le réseau de neurones multicouche vous aurez à implémenter, tout d'abord, la régression logistique qui permet de calculer un vecteur de scores exprimant les probabilités d'appartenance de l'entrée ***x*** à chacune des classes. Pour cela, vous aurez à télécharger le code pour la régression logistique se trouvant sur le tutoriel Theano suivant http://www.deeplearning.net/tutorial/logreg.html#logreg .

Pour pouvoir se familiariser avec Theano, il vous est fortement recommandé de suivre le tutoriel de base de la librairie
http://www.deeplearning.net/software/theano/tutorial/index.html#tutorial .

Pour pouvoir installer Theano il est recommandé de suivre les instructions qui se trouve sur la page suivante http://www.deeplearning.net/software/theano/install.html#anaconda .

Minimiser la fonction de perte est équivalent à maximiser (-Risque). Pour cela vous êtes demandés d'implémenter les mises à jour de l'algorithme de descente de gradient stochastique que vous avez dû implémenter au TP2. Pour pouvoir effectuer la minimisation du risque empirique en utilisant la descente de gradient stochastique, consultez la section suivante: http://www.deeplearning.net/tutorial/logreg.html#defining-a-loss-function .

Maintenant, pour pouvoir concrétiser cela et utiliser la régression logistique pour la classification des données MNIST, vous devez vous rendre à la section suivante :
http://www.deeplearning.net/tutorial/logreg.html#putting-it-all-together .

Pour faire de la prédiction en utilisant le modèle de régression logistique, rendez-vous à la section suivante : http://www.deeplearning.net/tutorial/logreg.html#prediction-using-a-trained-model .

Après avoir implémenter la régression logistique avec Theano, vous allez devoir implémenter la classe qui instancie une couche cachée d'un réseau de neurones de type multicouche. La sortie de la couche cachée est donnée par l'équation suivante :

$$\boldsymbol{h_i} = \mathrm{Relu}(\mathbf{W}_{i-1}\,\boldsymbol{h}_{i-1} + \boldsymbol{b_i})$$

Dans le cas où la couche précédente est la couche d'entrée, la couche cachée est remplacée par ***x***.

Pour l'implémentation d'une couche cachée, vous pouvez vous inspirer de l'implémentation se trouvant ici http://www.deeplearning.net/tutorial/mlp.html#going-from-logistic-regression-to-mlp .

Comme on utilise une fonction d'activation de type ReLU pour notre réseau, vous devez initialiser les poids pour les couches selon le critère de (He, 2015). Selon ce critère, les poids dans les couches devrait être initialisés uniformément ou normalement entre **[- 2 / fan_in, 2 / fan_in].**

En général, pour contrôler la capacité des réseaux de neurones, on devrait avoir des poids normalement distribués et centrés à l'origine. Une grande variance des poids induit à une grande capacité du modèle et donc à un problème de surapprentissage. Une variance très petite induit à un problème de sous-apprentissage. Une variance adéquate permet d'éviter ces deux problèmes et permet donc de généraliser sur les données non observées lors de l'entraînement. Pour pouvoir contrôler la variance des paramètres dans le réseau, une fonction pénalisant les grands poids est ajoutée au risque empirique afin de prendre en compte lors de l'optimisation l'effet des grands poids sur la généralisation. Une telle pénalité est appelée pénalité *L2* et permet de pénaliser la norme des poids. Vous aurez alors à implémenter la pénalité L2 pour les paramètres de votre réseau. Pour cela veuillez consulter le lien suivant http://www.iro.umontreal.ca/~bengioy/ift6266/H12/html/mlp_fr.html#controle-de-la-capacite-effective .

Vous pouvez implémenter votre Perceptron multicouche en vous basant sur l'implémentation se trouvant ici http://www.deeplearning.net/tutorial/mlp.html#putting-it-all-together .

L'augmentation des données est aussi requise lorsque le jeu de données est de petite taille. L'augmentation consiste à effectuer des transformations affines sur les données originales. À chaque itération, vous êtes demandés d'effectuer pour chaque exemple de données rencontré, une rotation aléatoire, une translation aléatoire ou bien une permutation aléatoire des axes de l'image (random flip). La transformation affine est choisie aléatoirement. Au lieu d'entrer l'image originale au réseau, c'est l'image transformée qui doit être passée. Pour mieux comprendre l'effet de ces transformations, veuillez consulter cette page http://machinelearningmastery.com/image-augmentation-deep-learning-keras/ .

Vous pouvez vous inspirer du lien suivant pour pouvoir implémenter les transformations requises https://github.com/fchollet/keras/blob/master/keras/preprocessing/image.py .

## Questions

1 – Implémentez tout d'abord la régression logistique avec Theano et donnez les résultats de de la classification pour les données d'entraînement, de validation et de test pour la meilleure configuration des hyperparamètres que vous avez trouvé;

2 – Implémentez la classe se rapportant à la couche cachée;

3 – Ajoutez au critère d'optimisation, la pénalité *L2* des poids. (N'ajoutez pas la pénalité au biais);

4 – Augmentez les instances du jeu de données en réalisant des transformations affines sur les entrées. Tracez les courbes d'entraînement et de validation dans le cas où vous n'utilisez pas

d’augmentation et comparez les dans le cas où vous effectuez des transformations affines des données. Rapportez l’erreur de test pour les deux cas une fois l’entraînement est terminé.

5 – Réalisez une recherche des paramètres optimaux pour le réseau en variant le nombre de neurones dans chaque couche cachée, le nombre de couches (entre 1 et 5 couches(maximum)), la constante de pénalisation et le pas d’apprentissage. La recherche des paramètres doit s’effectuer tout en réalisant les transformations sur les entrées. Une fois les paramètres optimaux obtenus, entraînez le modèle sur l’ensemble d’entraînement et de validation réunis. Donnez les courbes des erreurs d’entraînement et de validation lors de l’entraînement et rapportez l’erreur de test obtenu après l’entraînement.

## Références

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*. 2015.

# Barème

| Barème de correction Laboratoire 1, TP 1<br>(À remettre au début de notre prochain laboratoire) | |
|---|---|
| **Description des requis** | **Points alloués** |
| Question 1 | **5** |
| Question 2 | **5** |
| Question 3 | **5** |
| Question 4 | **5** |
| Question 5 | **10** |
| **Total** | **30** |