AS - TP1 - Prise en main de Torch et régression linéaire

Ludovic Denoyer et Nicolas Baskiotis

1 Tutorial LUA

Lire le tutorial LUA : `http://tylerneylon.com/a/learn-lua/`

2 Les tenseurs Torch

Torch est une bibliothèque LUA permettant la manipulation **de tenseurs**. Les tenseurs correspondent à des tableaux multi-dimensionnels (i.e vecteurs, matrices, etc...). Les tenseurs torch sont décrits ici :

1. `https://github.com/torch/torch7/blob/master/doc/tensor.md`
2. `https://github.com/torch/torch7/blob/master/doc/maths.md`

Un tout nouveau tutorial torch (en vidéo) : `https://github.com/Atcold/torch-Video-Tutorials/blob/master/README.md` (je n'en connais pas la qualité)

2.1 Exécution de Torch

Sur les machines de l'ARI, *torch* est installé dans le compte enseignant. La commande d'exécution est :

```
/home/Enseignants/denoyer/torch/install/bin/th
```

Torch possède une console interactive, mais peut aussi être appelé directement sur un fichier LUA :

```
/home/Enseignants/denoyer/torch/install/bin/th monfichier.lua
```

Pour des histoires de multi-threading, souvent, la commande suivante est beaucoup plus rapide :

```
OMP_NUM_THREADS=1 /home/Enseignants/denoyer/torch/install/bin/th monfichier.lua
```

3 Descente de gradient et Modèle linéaire

3.1 Rappel sur la descente de gradient

- Soit $(x_1, ..., x_n)$ les données d'apprentissage dans $\mathbb{R}^N$ associées aux étiquettes $(y_1, ..., y_n)$ dans $[-1; +1]$.
- Soit un modèle $f_\theta : \mathbb{R}^N \to \mathbb{R}$ de paramètres θ que l'on souhaite apprendre
- Soit $\Delta : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ la fonction de coût (ou loss) permettant de mesurer l'erreur de prédiction $\Delta(\hat{y}, y)$ entre la prédiction $\hat{y}$ et l'étiquette réelle y.

Nous cherchons à minimiser l'erreur de prédiction sur l'ensemble d'apprentissage. Cette erreur notée $\mathcal{L}(\theta)$ est :

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_i \Delta(f_\theta(x_i), y_i) \tag{1}$$

Afin de minimiser $\mathcal{L}(\theta)$, nous allons nous concentrer sur l'algorithme de descente de gradient. C'est un algorithme itératif dont il existe deux variantes principales : le **gradient batch** (ou gradient normal) et le **gradient stochastique** (ou SGD).

3.1.1 Gradient Batch

L'algorithme de gradient batch est itératif et consiste, à chaque itération, en la mise à jour suivante :

$$\theta \leftarrow \theta - \epsilon \nabla_\theta \mathcal{L}(\theta) \tag{2}$$

3.1.2 Gradient Stochastique

L'algorithme de gradient stochastique est itératif et consiste, à chaque itération, en la mise à jour suivante :

1. Tirage de i au hasard dans $[1; n]$
2. Mise à jour des paramètres :

$$\theta \leftarrow \theta - \epsilon \nabla_\theta \Delta(f_\theta(x_i), y_i) \tag{3}$$

3.2 Modèle linéaire

Un modèle linéaire $f_\theta(x)$ est défini par $f_\theta(x) =< \theta; x >$.

Exercice 1 : Calculez le gradient $\nabla_\theta \Delta(f_\theta(x_i), y_i)$ pour $\Delta(\hat{y}, y) = (\hat{y} - y)^2$ (coût des moindres carrés)

Exercice 2 : Calculez le gradient $\nabla_\theta \Delta(f_\theta(x_i), y_i)$ pour $\Delta(\hat{y}, y) = max(0, 1 - \hat{y}.y)$ (hinge loss)

Exercice 3 : Commentez sur la différence entre les deux fonctions de coûts précédentes

4 Implémentation en Torch

L'intérêt de Torch est qu'il permet le calcul automatique de fonction de gradients pour des modèles complexes (là où vous avez calculé le gradient manuellement dans l'exercice précédent). Il existe deux manières de calculer un gradient :

- Le **calcul algorithmique** (où backpropagation) (cf : Torch)
- Le **calcul symbolique** (où auto-dérivation) (cf : TensorFlow)

Nous allons pour l'instant nous focaliser sur la première technique.

4.1 Grands principes

Torch dispose de multiples *modules* (voir `https://github.com/torch/nn/blob/master/README.md`) permettant l'implémentation de méthodes dérivables de façon modulaires (voir "petit guide des réseaux de neurones profonds"). Un module correspond *in fine* à un modèle (i.e f_θ dans l'exemple précédent). Aujourd'hui, le module qui nous intéresse est le module *nn.Linear* qui correspond à une transformation linéaire. Comme tous les modules, il est composé de 4 méthodes importantes :

- **zeroGradParameters()** qui permet de remettre à 0 la valeur du gradient (stocké en mémoire) $Grad \leftarrow 0$
- **forward(x)** qui permet de calculer la valeur $f_\theta(x)$
- **backward(x,delta)** qui permet de calculer le gradient de l'erreur en fonction du gradient calculé sur les sorties du module : $Grad \leftarrow Grad + \nabla_\theta \Delta(f_\theta(x), y)$
- **updateParameters(l)** qui permet de mettre à jour les paramètres du module $\theta \leftarrow \theta - l.Grad$

Torch dispose aussi de *criterions* permettant le calcul d'une fonction coût (i.e Δ dans notre exemple précédent). Nous allons utiliser le critère des moindres carrés *nn.MSECriterion*. Tout comme les modules, un critère possède les méthodes suivantes :

- **forward(y',y)** qui permet de calculer la valeur $\Delta(y', y)$
- **backward(y',y)** qui permet renvoie la valeur de $\nabla_{y'} \Delta(y', y)$

Faire un "pas" de gradient sur une donnée (x, y) revient ainsi à effectuer les opérations suivantes :

```
module:zeroGradParameters()
output=module:forward(x)
loss=criterion:forward(output,y)
delta=criterion:backward(output,y)
module:backward(x,delta)
module:updateParameters(learning_rate)
```

Nous reviendrons sur ces différentes étapes dans le prochain TP suivants.

Exercice 4 : (à rendre par email)

- A partir du fichier `exercice_1_etu.lua`, implémentez la descente de gradient stochastique et la descente de gradient batch. Vous pouvez visualiser l'influence des différents paramètres.
- Lisez le fichier `exercice_1_optim.lua` qui propose une implémentation différente du gradient stochastique (et qui sera plus efficace par la suite).

Exercice 5 : Tester la méthode avec d'autres dataset issu de `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` en utilisant le package 'svm' de torch `https://github.com/koraykv/torch-svm` permettant de charger des données plus complexes.

Exercice 6 : (travail à rendre au prochain TP) : Lisez l'article suivant : `http://sebastianruder.com/optimizing-gradient-descent/`. Vous rendrez au prochain TP une page écrite donnant une description intuitive et succinte des différents algorithmes de descente de gradient disponibles et de leur grands principes (i.e batch, sgd, adagrad, adadelta, rmsprop, nag et adam). N.B : Ces différents algorithmes sont implémentés dans le package *optim* de torch.