**Projet de TAL**

Réseau de neurones pour   
l’étiquetage en parties du discours

**Bingzhi LI**

**Tian LAN**

M1 LI

2018-2019

**I. Introduction**

**présentation du sujet**

description de la tâche

définitions, concepts théoriques

questions et problématiques abordées(Durand\_M)

**II. implémentation informatique**

1.fonctionnement du programme

2.difficulté rencontrées et solution apportés

**III. Résultats et interprétations**

évaluation des performances de notre système  
(qualité des prédictions faites)

**conclusions/perspectives**

1. description de la tâche

**-**définition de réseau de neurones

L’objectif de ce projet est d’implémenter un réseau de neurones profond avec une architecture spécifiée (nombre, taille de couches et fonction d’activation appropriée) utilisant seulement NumPy afin de réaliser finalement une tâche d’étiquetage morphosyntaxique. Dans un premier temps, on se concentre sur notre sous-tâche : résoudre le problème XOR. On a implémenté un réseau de neurones à propagation avant basé sur l’abstraction du graphe de calcule (Goldberg, 2015) et une méthode d’entraînement par descente de gradient stochastique. Dans un second temps, on a testé le réseau sur la tâche d’étiquetage morphosyntaxique de type fenêtre de mots et a comparé les résultats avec un perceptron.

2. concepts théoriques et techniques

Above all, we must be able to train our network and make predictions using it.

what operations will have to be performed during the training of our neural network. It also shows how many parameters we will have to update and read at different stages of a single iteration. Building the right data structure and skillfully managing its state is the most difficult part of our task.

**Initiation of neural network layers**

-convention de notation

z valeur de pré-activation, a valeur d’activation, u = x•w+b est combinaision linaire

fonction d’activation :

relu ① ReLU = max(0,z)

② ReLU\_backward =

**fonction d’activation de la couche de sortie et fonction de perte**

La fonction softmax est utilisée comme fonction d’activation de la couche de sortie dans notre réseau. Elle est appliquée sur le vecteur de scores (un score pour chaque classe) de manière que les scores de classe forme une distribution de probabilité sommant à 1.

Soit l’exemple et le one hot vecteur de sa gold classe , u est vecteur de scores et p le vecteur de probabilités qu’on associe à chaque classe Ci sachant l’exemple  , la classe prédite.

softmax ① forward : pi = P(Ci **|** ) = ( softmax(u))i =

② backward =

Le réseau de neurones voudrait maximiser la prédiction : p( = **|**) . On peut minimiser le negative log-likelihood :

avec g = gold class

Pour un mini-batch de taille m, la perte total est la somme des pertes de chaque observation du mini-batch:

où est la probabilité que la ième exemple est classé dans sa gold classe en utilisant θ comme paramètre de notre modèle.

**algorithme du gradient stochastique mini-batch**

On cherche à minimiser la somme de la fonction de perte des exemples. Une façon naturelle de minimiser la fonction objective par rapport à son paramètre θ est d’utiliser l’algorithme du gradient. Le gradient est utilisé afin de trouver la direction pour laquelle la fonction J(θ) décroît le plus rapidement. Lorsque θ se situe au point θt, cette direction correspond à la direction négative du gradient de la fonction J, c’est à dire . Pour un taux d’apprentissage assez petit, si

= , alors J() . On part d’un point de depart θ0 initialisé aléatoirement pour la fonction de perte afin de diriger à petits pas vers un minimum en espérant que celui-ci soit un minimum global. Si la fonction objective est convexe, cet algorithme mène à une solution optimale.

Lorsque le nombre d’exemples T est grand pour certaines tâches, calculer le gradient de la somme de fonctions de pertes des tous les exemples peut être très difficile au niveau computationnel. C’est pourquoi, on utilise dans notre réseau l’algorithme du gradient stochastique Mini-Batch qui permet de paralléliser les traitements des mini-batchs. On regroupe les exemples en mini-batch de taille m, on somme la perte sur chaque exemple du min-batch et met à jour les paramètres sur chaque mini-batch.

1. initialiser θ (W,b)
2. pour N itération :

pour chaque mini-batch du corpus train :

=

Pour appliquer cet algorithme à l’apprentissage du réseau de neurones, il nous faut :

la fonction de perte

calculer le gradient de la fonction de perte par rapport aux paramètres

méthodes d’initialisation

**Calculer le gradient par rétro-propagation :**

gradient de la couche de sortie :

On commence par calculer le gradient de la fonction de perte par rapport à la sortie de la dernière couche du réseau. Soit n(0) n(1) n(2) …n(L) nombre de neurones respectif dans chacune des L+1 couche du réseau allant de la couche d’entrée (couche 0) jusqu’à la couche de sortie (la couche L+1). = (Wt,bt), t = nb d’itération, l = 0,…L sont des indices des couches, à l’intérieur desquelles, les paramètre à estimer sont et . Avec cette notation, la function de perte entropie croisée s’écrira :

Soit , la dérivée partielle de la fonction de perte par rapport au ième composante du vecteur de sortie a :

=

Soit ui est noté zi (z pour valeur de pré-activation), pi est noté ai (a pour valeur d’activation) et S = . Alors la ième composante de la sortie de la fonction de softmax ai par rapport à son entrée zi peut être calculé de cette manière :

Si classe i = j:

= = = =

Si classe i ≠ j:

= = =

Selon la règle de la dérivation en chaîne, la dérivée partielle de la fonction de perte par rapport au ième composant du vecteur de pré-activation z

Ainsi le gradient stocker dans la couche L est :

  est one hot vecteur de

gradient sur couches cachés

les dérivées partielles de la fonction de perte par rapport aux paramètres W et b :

fonction d’activation utilisé dans les couches caché est fonction ReLU :

g(z) = max(0,z)

(z) =

qui permet de résoudre le problème de fuite du gradient « vanishing gradient problème » (Pascanu et al., 2012), par conséquent d’arriver à une convergence plus rapide.

**IV. Bibliographie**

Pascanu, R. , Mikolov, T. et Bengio, Y. (2012) . Understanding the exploding gradient problem. CoRR, abs/ 1211 .5063. Récupéré de http: 1 /arxiv. org/ abs/1211.5063