Projet De Fin d'Etudes : Reseaux de Neurones et Application à la Compréhension du langage avec Tensorflow

Clément COLLET

Contents

Part I
Introduction

Le thème de ce projet est les Réseaux de Neurones. Les différents objectifs sont d'en comprendre la théorie, les différents types et d'utiliser un environnement adapté (TensorFlow) pour obtenir des résultats sur différents problèmes de compréhension du langage.

Les réseaux de neurones sont une forme d'intelligence artificielle largement inspirés du fonctionnement du cerveau humain. Ils se caractérisent par leur capacité d'apprentissage. Les réseaux de neurones sont particulièrement adaptés pour les problèmes de classification. Nous détaillerons dans la première partie les concepts essentiels des réseaux de neurones.

TensorFlow est une librairie en open-source destinée au Machine Learning. Elle est développée par Google Brain et écrite en Python et C++. Elle est utilisée par de nombreuses branches de Google, de Gmail à Youtube. Nous présenterons les avantages de TensorFlow dans la seconde partie. Cette environnement est assez complexe à prendre en main. Nous avons donc réaliser des implémentations simples sur un problème classique du Machine Learning : la classification de MNIST.

Notre application principale concerne la compréhension du langage, Natural Langage Processing (NLP) en anglais. Nous aurons besoin de plusieurs concepts comme la représentation des mots en vecteur (Word-Embedding), que nous présenterons. Nous étudierons un modèle de réseau convolutif qui permet de traiter des problèmes de NLP. Nous l'appliquerons sur deux corpus de texte.

Part II
Théorie des réseaux de neurones

Nous allons commencer par la théorie derrière les réseaux de neurones.

1 Neurone Formel

Un neurone est un automate. Il dispose d'entrées et d'une sortie. Son processus de décision est simple. Chaque entrée est multipliée par un "poids synaptique" et ces produits sont sommés. Si la somme est supérieure à la valeur seuil du neurone, il émet un signal. Les poids des différentes entrées déterminent le comportement du neurone face à une entrée donnée. Ils sont modifiés lors de l'apprentissage.

C'est l'assemblage en couche organisée de ces neurones qui permet de traiter des problèmes complexes. Un réseau de neurones est caractérisé par la structure de ses neurones et les poids les reliant.

Le principe général est de présenter à notre réseau des exemples labélisés du problème à résoudre, c'est à dire des cas où nous connaissons à l'avance la sortie que nous attendons de notre réseau. En comparant sa propre sortie avec ce qui était attendu, le réseau modifie ses poids pour progresser.

Durant la présentation de la théorie, nous allons souvent utiliser comme exemple le traitement d'image. C'est l'application la plus répandue des réseaux de neurones. Cela nous permettra de visualiser plus facilement des méchanismes parfois complexes.

Nous n'aborderons en dernier le traitement du langage par la machine.

2 Le perceptron

2.1 Modèle

Le premier modèle de perceptron provient de Frank Rosenblatt et date de 1957. Il ne contient qu'un unique neurone. Il est capable d'apprendre par expérience. C'est un classifieur linéaire.

La sortie Y du neurone est calculée en deux temps. Il réalise la somme pondérée des différentes entrées et il passe cette somme dans sa fonction d'activation. Si le résultat obtenu dépasse le seuil d'activation, le neurone émet. Il existe de nombreuses fonctions d'activation possibles différentes: la fonction tangente hyperbolique, la fonction sinusoide... Chacune peut avoir ses avantages et inconvénients, il convient de choisir la plus adaptée à notre modèle à chaque fois. Ici, nous allons utiliser la fonction de Heaviside avec un seuil valant 0 :

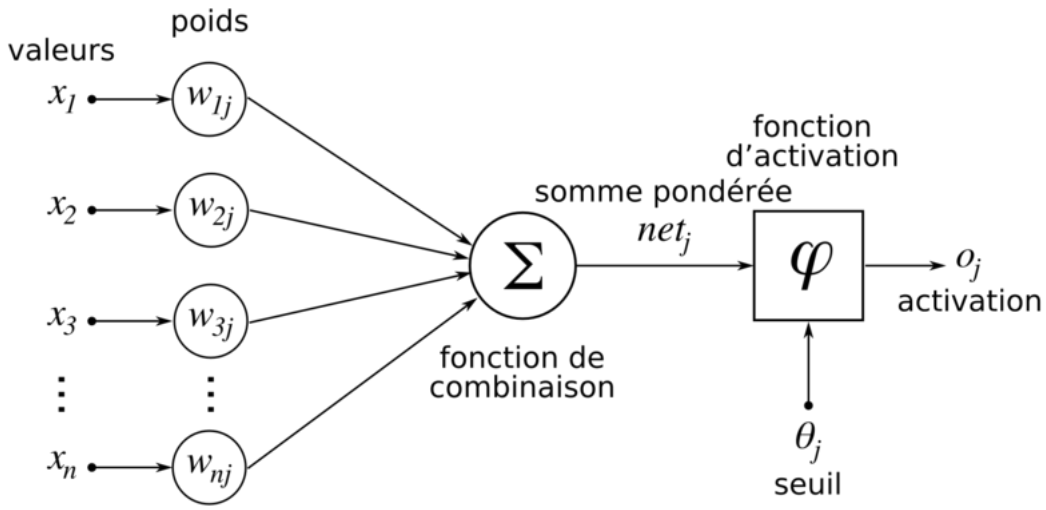


Figure 1: Schéma du Perceptron

$$Y = H(Z) = \begin{cases} 0 & \text{si} \quad Z < 0 \\ 1 & \text{si} \quad Z \geq 0 \end{cases}$$

2.2 Apprentissage

A l'initialisation, les différents poids sont choisis aléatoirement. Nous pouvons les initialiser à nuls mais il est admis qu'il est préférable qu'ils soient des réalisations d'une loi uniforme centrée réduite. Il n'existe pas de théorème pouvant démontrer que ce choix est le meilleur. Il nous faut expérimenter et obtenir nos réponses de manière empirique. Cela est valable pour l'ensemble des choix que nous aurons à faire durant notre implémentation.

Durant l'apprentissage, pour chaque exemple X présenté, nous calculons la sortie

$$Y = f(\sum W_i * X_i)$$

Avec f la fonction d'activation et W les poids du neurones.

Les poids sont mis à jour :

$$W_i(t+1) = W_i(t) + \alpha(D - Y)X_i$$

Avec D la sortie attendue, X_i les différentes entrées du neurone i, W_i les poids et α le taux d'apprentissage. Le taux d'apprentissage est une variable qui pondère les modifications de poids provoquées par chaque exemple. Nous reviendrons plus tard sur cette variable et comment optimiser l'apprentissage grâce à elle.

2.3 Biais

En plus des poids W_i multipliés par les entrées Xi, le neurone ajoute un biais b avant d'appliquer sa fonction d'activation f.

$$Sortie = f(X \times W + b)$$

3 Le perceptron Multicouche

3.1 Modèle

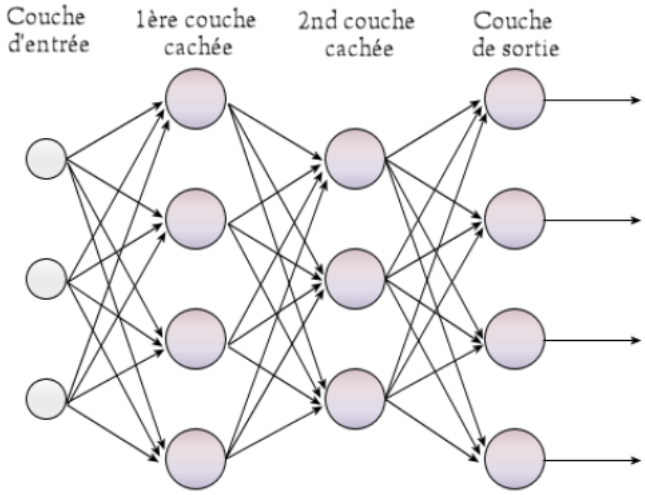


Figure 2: Modèle d'un perceptron multicouche [4-3-4]

Chaque neurone possède en entrée tous les neurones de la couche précédente et envoie sa sortie à tous les neurones de la couche suivante. Il y autant de neurones sur la dernière couche qu'il y a de valeur de sortie à notre réseau, chacune étant fournie par un neurone. Dans le cas où nous voulons classifier des exemples entre n classes, il y a n neurones sur la dernière couche. Chacune revoyant une valeur correspondant à la "force" avec laquelle le réseau associe cet exemple et cette classe donnée.

Nous pouvons choisir différentes fonctions d'activation pour notre réseau : la fonction sigmoïde, la tangente hyperbolique... Il est important que cette fonction soit facilement dérivable pour l'apprentissage.

3.2 Apprentissage - Algorithme de rétropropagation

De manière identique avec le perceptron simple, nous présentons des exemples à notre réseau qui met à jour ses poids en fonction de l'erreur obtenue.

Soit $\epsilon_i = d_i - y_i$ l'erreur obtenue pour le neurone de sortie i. L'erreur totale est

$$\xi = (1/2) \sum \epsilon_i^2$$

Les poids des neurones de la dernière couche sont les premiers mis à jour, la formule étant pour le neurone j :

$$W_{ji}(t+1) = W_{ji}(t) - \alpha * \frac{\partial \xi}{\partial v_j} * y_i$$

Avec y_i l'entrée de j provenant du neurone i, W_{ji} le poids entre le neurone i et j . Il faut calculer la dérivée :

$$\frac{\partial \xi}{\partial v_j} = \epsilon_j \phi'(v_j)$$

Avec ϕ' la dérivée de la fonction d'activation du neurone j. Une fois les poids de la dernière couche mis à jour, nous modifions ceux de la couche précédente avec :

$$\frac{\partial \xi}{\partial v_l} = \phi'(v_l) \sum_k -\frac{\partial \xi}{\partial v_k} W_{kl}$$

et k prenant tous les valeurs de la couche suivante, c'est à dire ceux de la dernière couche. Nous réutilisons cette formule autant de fois que nécessaire jusqu'à la première couche. Ce processus itératif est appelé rétropropagation. Ici, nous utilisons comme méthode de rétropropagation : la descente de gradient. C'est la version la plus simple. Nous pouvons noter qu' α y est constant. Nous constatons l'importance d'avoir une fonction d'activation facilement dérivable étant donnée que nous allons être amené à la calculer autant de fois qu'il y a de neurones dans notre réseau.

4 Différentes méthodes de rétropropagation

Cette partie décrit différentes méthodes pour optimiser la rétropropagation.

4.1 Descente de gradient

Cela correspond à la méthode décrite plus haut. Nous allons changer de notation. Ainsi, la formule générale de mis à jour des poids est :

$$\theta = \theta - \eta \bigtriangledown_\theta J(\theta)$$

avec η le taux d'apprentissage. Nous avançons à chaque instant dans la direction de la dérivé de la fonction d'activation. Si nous utilisons comme parallèle une boule qui glisserait le long d'une colline, à chaque instant la boule se déplacerait dans la direction de la pente.

4.2 Momentum

Cette méthode trouve sa limite lorsque la surface de solution est trop inclinée dans une direction par rapport aux autres. Pour palier la faiblesse de la méthode de la descente de gradient, nous ajoutons un terme γ généralement fixé à 0,9 que nous appelons momentum. Si nous reprenons notre analogie avec la boule et la pente, le momentum représente la vitesse qu'à la boule à ce moment donné, c'est à dire une force qui la pousse dans la direction qui est celle de la pente à l'itération précédente. La nouvelle direction sera celle de la pente plus une partie de la direction de la pente précédente.

$$v_t = \gamma v_{t-1} + \eta \bigtriangledown_\theta J(\theta)$$
$$\theta = \theta - v_i$$

4.3 Nesterov Accelerated Gradient

Nesterov va donner au terme momentum une connaissance sur la futur position des poids, de manière à la corriger en avance.

$$v_t = \gamma v_{t-1} + \eta \bigtriangledown_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_i$$

Cette méthode est plus rapide que les précedentes et gère bien les topographies irrégulières.

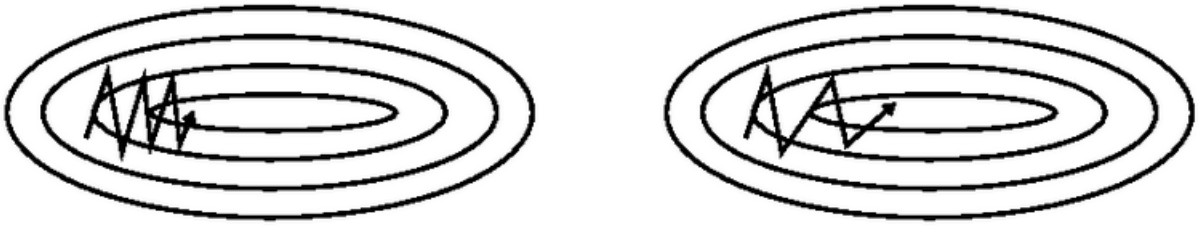


Figure 3: Différence entre descente de gradient avec ou sans momentum, nous constatons qu'une partie du déplacement vers la droite est conservée et donc "accélérée"

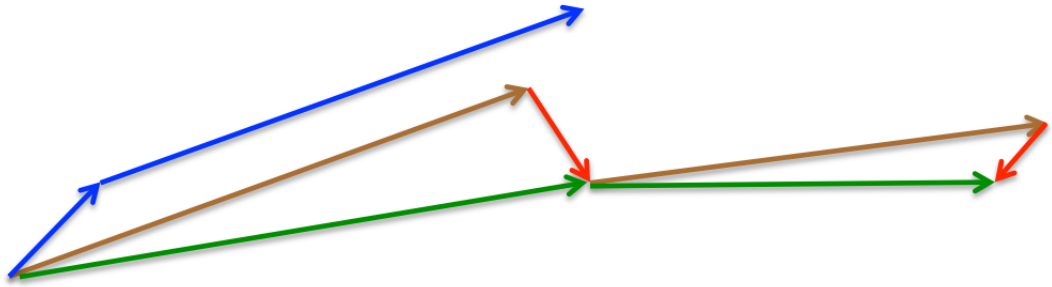


Figure 4: Le trait bleu correspond au Momentum, le trait brun est la part identique au Momentum dans Nesterov qui est corrigé directement par le trait rouge, ainsi Nesterov suit le vecteur vert

4.4 Adagrad

Agradad se concentre sur une chose : lier le taux d'apprentissage α à la fréquence des paramètres. Si le réseau rencontre un exemple nouveau et inhabituel, il va chercher à apprendre au maximum de celui ci. Les mises à jour des poids seront très importantes avec des paramètres d'entrée rares. A contrario, les mises à jours des poids seront légères lorsque l'exemple en entrée est commun. C'est très adapté pour les bases de données éparses où nous pouvons avoir peu d'exemples pour une classe donnée.

4.5 Adelta

Adelta s'inspire de Agradad et en corrige la tendance à réduire trop rapidement le taux d'apprentissage.

4.6 Adam

Adaptive Moment Estimation est une méthode qui fait aussi varier le taux d'apprentissage en fonction des paramètres. Elle conserve la moyenne du carré des gradients passés v_t, mais de manière à ce que l'importance des gradients passés dans cette moyenne diminue exponentiellement. Elle conserve aussi la moyenne des gradients passés m_t en mettant de la même manière l'accent sur les gradients passés les plus proches :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) v_t$$

Nous corrigeons les estimateurs qui sont en l'état biaisés.

$$\underline{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$\underline{v_t} = \frac{v_t}{1-\beta_2^t}$$

Nous mettons à jour nos paramètres :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\underline{v_t}+\epsilon}}\underline{m_t}$$

4.7 Comparatif

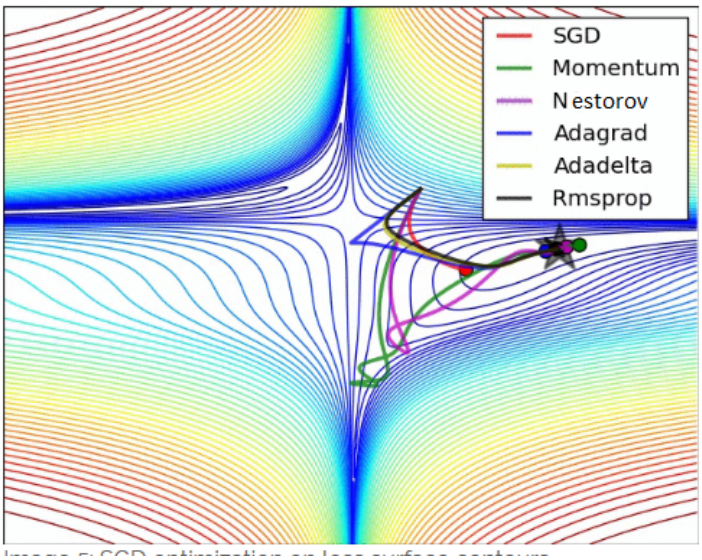


Figure 5: parcours des différents algorithmes

Il n'y a pas de méthode qui s'impose complétement par rapport aux autres. Cela dépend de nos données. Dans le cas où celles-ci sont mal réparties, il est important de choisir une méthode avec un taux d'apprentissage variable (Adelta, Adam, Adagrad). De plus, ces méthodes sont plus rapides que celles basées sur la descente de gradient. Cependant, si le temps n'est pas une contrainte et que le réseau n'est pas trop profond (plus il y a de couche, plus les calculs sont nombreux), la méthode simple de gradient de descente est la plus robuste.

5 Les différentes fonctions pour calculer l'erreur

Le but des fonctions "coût" est de calculer la différence entre la sortie obtenue y_o et celle attendue y_a.

5.1 Le coût quadratique

La fonction la plus commune est celle de l'erreur quadratique, c'est simplement la somme de l'écart entre notre solution et celle attendue au carré.

$$C = (1/n)\sum(y_a - y_O)^2$$

5.2 La Cross-Entropy

La fonction Cross-Entropy utilise le logarithme de la solution attendue.

$$C = (1/n) \sum (y_a log(y_O))^2 + (1 - y_a) log(1 - y_O)$$

5.3 Choix de la Cross-Entropy pour la classification

Nous allons utiliser un exemple pour nous permettre de faire le choix de la fonction Cross-Entropy pour les problèmes de classification.

Considérons 3 classes : A, B, C. Nous avons deux réseaux qui nous renvoient pour chaque exemple la probabilité selon eux que cette exemple fasse partie d'une classe donnée.

Resultat fourni par le reseau 2			Affectation			Correct ?
A	B	C	A	B	C	
0.3	0.3	0.4	0	0	1	Oui
0.3	0.4	0.3	0	1	0	oui
0.1	0.2	0.7	1	0	0	Non
Resultat fourni par le reseau 1			Affectation			Correct ?
A	B	C	A	B	C	
0.1	0.2	0.7	0	0	1	Oui
0.1	0.7	0.2	0	1	0	oui
0.3	0.4	0.3	1	0	0	Non

Figure 6: Exemple de résultats avec deux réseaux de neurones soumis aux mêmes entrées

Les deux réseaux ont une précision de 67%, c'est à dire qu'ils ont correctement affecté deux exemples sur trois en moyenne. Pour le premier réseau, l'erreur quadratique obtenue est 0.81. Le second a une erreur quadratique de 0,34. Quand nous regardons en détail les résultats obtenus, nous constatons que le premier réseau a identifié de justesse les deux premiers exemples (seulement 0,1 de différence) et est complètement passé à coté du dernier exemple. A contrario, le deuxième réseau a bien identifier les deux premiers exemples et a loupé le dernier de seulement 0,1. L'erreur cross-entropique du premier réseau est de 1,38 tandis que celle du deuxième est 0,64. Nous utiliserons donc cette fonction qui permet une erreur plus grande.

6 Gestion des données

Nous allons dans cette partie étudier la gestion de notre base de données : son découpage en base d'apprentissage et d'évaluation, la quantité de données que nous fournisssons à notre réseau à chaque étape du processus...

6.1 Base d'apprentissage et d'évaluation

Nous disposons de notre base de données d'exemples labélisés. Une partie de ces exemples seront utilisés pour l'apprentissage et modifieront les poids du réseau (90% des données en

général). Le reste servira de base d'évaluation et ne modifiera pas les poids. Cette base nous sert à vérifier que notre réseau est capable de généraliser, capable de traiter des cas qu'il n'a pas vu durant l'apprentissage. Nous pouvons tester notre réseau tout au long de l'apprentissage afin de suivre sa performance.

6.2 Quantité de données fournies à un même instant

6.2.1 Stochastique descente de gradient

Dans cette méthode, nous fournissons les exemples un par un et les poids sont modifiés à chaque fois. Le problème est la lenteur du processus s'il y a un nombre important de données puisque c'est la rétropropagation, la correction des poids, qui est couteuse en calcul. De plus, chaque exemple modifie pleinement les poids, cela signifie que les exemples extrêmes vont pouvoir influencer énormément nos poids ce qui peut être un problème.

6.2.2 Le Batch-descente gradient

La batch-descente gradient est l'opposé, nous fournissons à notre réseau l'ensemble des données à notre disposition; nous calculons l'erreur moyenne totale et nous mettons à jour nos poids avec celle-ci. Il est normalement déconseillé de soumettre plusieurs fois au réseau le même exemple car il risque d'apprendre "trop" sur celui-ci et de se spécialiser, mettant ainsi en danger le pouvoir "généralisateur" des réseaux de neurones. Néanmoins, ici le batch étant lui même un agrégat de nombreuses données, nous pouvons le soumettre plusieurs fois, modifiant les poids à chaque fois. Nous effectuons en général plusieurs itérations sur notre base d'apprentissage, jusqu'a ce que les résultats sur la base d'évaluation soient stabilisés.

6.2.3 Mini-Batch

La méthode que nous utiliserons est celle des mini-batch. Nous fournissons au réseau les données par petit groupe. Nous calculons de même l'erreur moyenne pour ces batchs et mettons à jour les poids. La taille des batchs est un hyper paramètre à déterminer au cours des tests. Entre deux itérations sur la base d'apprentissage, les données sont mélangées. Les batchs soumis ne contiennent pas exactement les mêmes exemples. Cela permet théoriquement de pouvoir soumettre un nombre très important de batchs différents alors que nous utilisons au final les mêmes données.

6.3 10 Fold Cross-Validation

Lors des différents tests, il est important prendre à chaque fois une section différente de données comme base d'évaluation. La technique la plus commune consiste à diviser la base de données en 10 parties, apprendre sur 9/10 et évaluer sur le dernier 10ième. Il faut recommencer 10 fois en changeant la base d'évaluation et prendre les moyennes des valeurs obtenues.

7 Evaluation de la performance

7.1 Précision et Recall

Il y a de multiples valeurs à suivre lors de la présentation d'exemples afin de savoir si notre réseau traite bien le problème.

Nous traquons la précision globale : le % d'exemple ayant été réussi, où notre réseau a fourni la réponse désirée. Dans le cas ou notre réseau est un classificateur, c'est à dire qu'il permet de déterminer à quelle classe appartient l'exemple soumis, il existe d'autres valeurs : les vrais positifs et les faux positifs.

Imaginons que tous nos exemples soient repartis en 2 classes: A et B.
1) Nous pouvons calculer le pourcentage d'exemple appartenant à A ayant été correctement identifié comme tel (Nombre d'exemples appartenant à A ayant étés identifié comme tel / Nombre d'exemples appartenant à A fournis au réseau). C'est simplement la précision limité à la classe A.
2) Nous pouvons aussi calculer le pourcentage d'exemples ayant été identifiés comme appartenant à A et qui l'était vraiment (Nombre d'exemples ayant étés identifiés comme appartenant à A et qui appartenaient vraiment à A / Nombre d'exemples ayant étés identifiés comme appartenant à A). Cette précision "inversée" nous indique si notre réseau n'associe pas à tort trop d'exemple à la classe A.
Ces deux valeurs sont à calculer pour toutes les classes. Elles permettent de mieux comprendre comment notre réseau réagît et traite nos données. La précision globale lisse les résultats par classe, elle peut nous masquer les différences de résultat entre les classes. Une classe peut être très bien identifiée et une autre ne pas l'être. Il faut alors chercher la cause de cet écart.

7.2 Sur-Apprentissage (Overfitting)

Ces valeurs sont à suivre aussi bien durant la phase l'apprentissage que lors des évaluations. Durant l'apprentissage, elles nous permettent de constater que notre réseau s'améliore au fur et à mesure que des données lui sont soumises. Lors de l'évaluation, c'est la capacité à traiter des exemples nouveaux qui est interrogée.

Il est normal de constater une différence entre les résultats sur la base d'apprentissage et ceux de la base d'évaluation qui sont souvent moins bons. Il existe cependant le problème du sur-apprentissage. C'est un phénomène où notre réseau se spécialise trop sur les données d'apprentissage et perd sa capacité à traiter des cas nouveaux. Nous pouvons déterminer que notre réseau est en sur-apprentissage quand ses résultats sur la base d'apprentissage continuent à progresser tandis que ceux sur la base d'évaluation régressent.

Plus un réseau est complexe et possède de neurones, plus il aura tendance à faire du sur-apprentissage. Cela est lié à l'augmentation du nombre de paramètres. Par exemple, un réseau de neurone avec 30 couches fully connected pour classifier MNIST possède 24 000 paramètres. Une solution peut être d'arrêter l'apprentissage avant cette inflexion des résultats sur la base d'évaluation mais cela ne corrige pas le problème structurel de notre réseau. Les deux principales solutions utilisées sont le drop-out et la L2/L1 Régularisation.

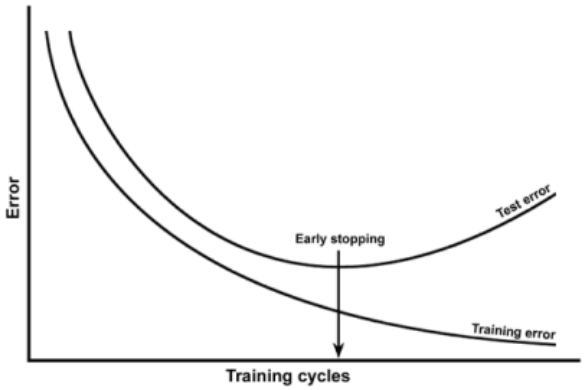


Figure 7: Erreur sur la base d'évaluation augmente alors que celle sur la base d'apprentissage diminue

8 Le Drop-Out

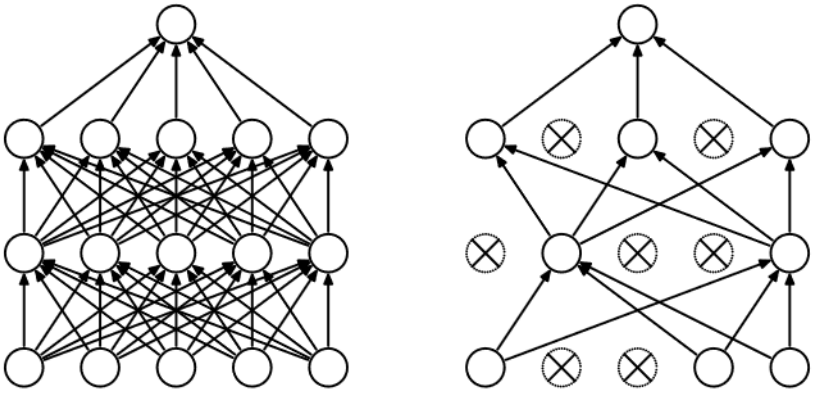


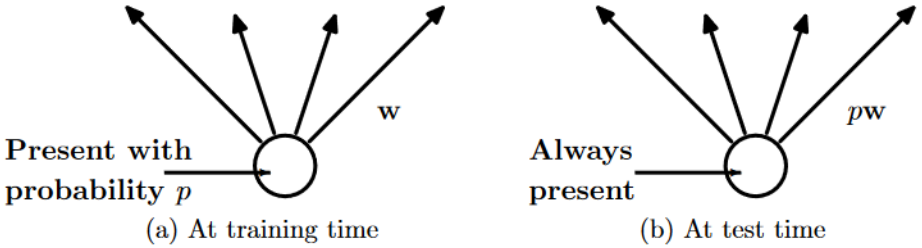
Figure 8: Exemple de Drop-Out sur un perceptron multi couche

L'objectif du Drop-Out est de réduire le sur-apprentissage. En effet, lorsque la structure d'un réseau de neurone est complexe ou lorsque le nombre de neurones présents est important, nous nous trouvons souvent dans un cas de sur-spécialisation du fait du nombre élevé de paramètres. Le principe du Drop-Out est de désactiver des neurones choisis aléatoirement. Nous définissons une probabilité p de "désactiver" un neurone et toutes ses connexions. Nous pouvons définir une probabilité de Drop-Out différentes sur chaque couche du réseau. Dans la majorité des cas, p vaut 0,5 à l'intérieur du réseau. Pour les couches externes, désactiver un neurone signifie se passer d'une entrée ou modifier notre sortie, c'est pourquoi nous préférons y fixer p à 0.

Lors des phases d'évaluation, tous les neurones sont activés et les poids sont pondérés par la probabilité p qu'avaient les neurones d'être désactivés.

9 L2/L1 Régularisation

L'idée est de pénaliser le réseau lorsque les poids de celui-ci augmentent trop. Nous rajoutons un terme à la fonction coût. Voici un exemple en utilisant la fonction Cross Entropy vue plus haut.



(a) At training time (b) At test time

Figure 9: Probabilité de drop-out de chaque neurone diffère entre entrainement et phase d'évaluation

L2 Régularisation :

$$C = -(1/n)\sum (y_a log(y_O))^2 + (1-y_a)log(1-y_O) + \frac{\lambda}{2n}\sum_w w^2$$

Avec λ le paramètre de régularisation et n la taille de la base de données.
Nous remarquons que nous ne prenons pas en compte les biais. Si nous choisissons une valeur de λ faible, cela signifie que nous préférons minimiser la fonction coût original, sinon nous mettons l'accent sur l'importance d'avoir des poids faibles.

Nous pouvons aussi utiliser la L1 Régularisation :

$$C = -(1/n)\sum (y_a log(y_O))^2 + (1-y_a)log(1-y_O) + \frac{\lambda}{2n}\sum_w |w|$$

Voir dans les références pour l'article comparant en profondeur les deux méthodes et les justifiants théoriquement.

10 Différentes couches possibles

10.1 Couche complètement connectée ou "Fully Connected"

Cette couche correspond à une couche de perceptron multicouche, avec tous les neurones reliés aux neurones de la couche précédente. C'est la couche la plus classique. Si nos entrées sont un vecteur v, avec w les poids et b le biais et f la fonction d'activation de cette couche , nous pouvons assimiler l'application de cette couche avec :

$$Sortie_i = f(\sum_j v_j w_{ji} + b_i)$$

10.2 Couche de Convolution

10.2.1 Principe

L'intérêt des couches de Convolution est la reconnaissance de motifs dans nos entrées. Contrairement aux couches fully connected, un neurone n'est pas toujours relié à l'ensemble des neurones de la couche précédente. Chaque neurone n'est relié qu'à un sous ensemble des neurones de la couche précédente qui forment une tuile. L'intérêt est de conserver dans l'analyse la notion de proximité spatiale (ou temporel) des entrées.

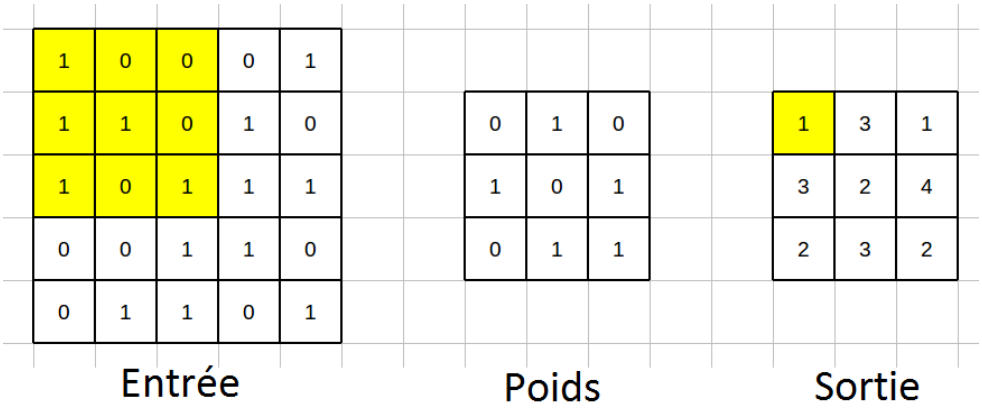


Figure 10: Exemple de convolution appliquée à une image avec : une tuile 3x3 sélectionnée, multipliée par les poids pour fournir la sortie

Nous appliquons souvent plusieurs convolutions en parallèle, chacune se spécialisant à termes dans la recherche d'un motif particulier. Cela signifie que ces poids vont s'organiser de manière à avoir une sortie maximale lorsque l'entrée correspond au motif recherché.

En sortie d'une couche de convolution, nous trouvons autant de matrices que nous cherchions de motifs.

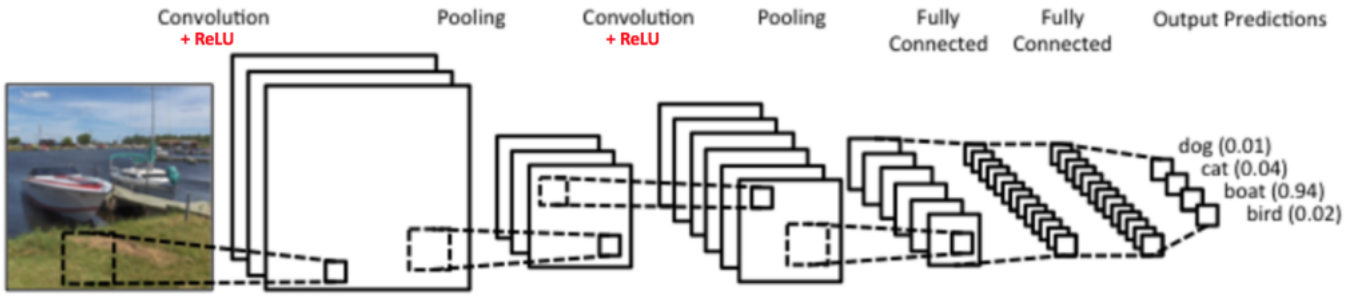


Figure 11: Schéma classique d'utilisation de la convolution au sein d'un réseau

Nous utilisons souvent la fonction d'activation Relu au sein des couches de convolutions. Elle renvoie le maximum entre zéro et notre somme d'entrées pondérées.

10.2.2 Gestion des bords

Il y a deux possibilités quant à la gestion des bords :
- Soit nous décidons de ne passer notre tuile que là où elle est remplie, dans ce cas, notre sortie est plus petite que notre entrée. C'est le cas dans l'exemple ci dessus, nous parlons de "Narrow convolution".
- Soit nous décidons de prolonger artificiellement notre entrée en rajoutant des zéros là où cela est nécessaire. Nous parlons de "Zéro Padding".

10.3 Couche de Pooling

Notre entrée est séparée en tuiles qui ne se chevauchent pas. La sortie dépend des valeurs en entrée

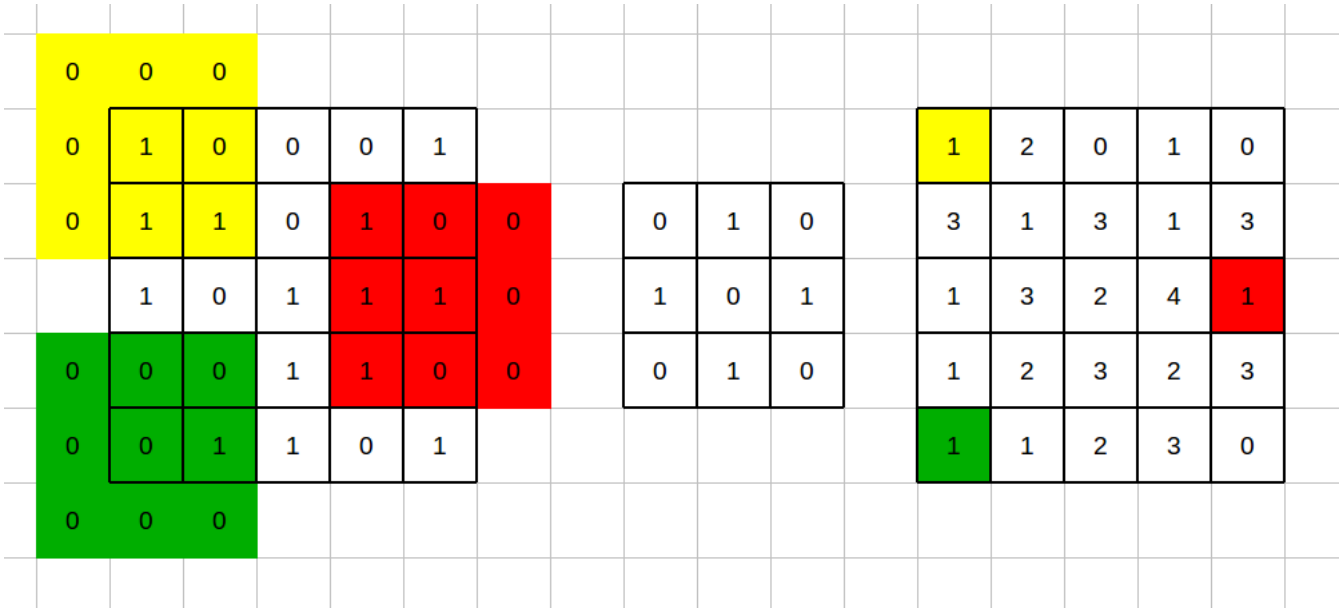


Figure 12: Exemple de Zero padding avec des filtres 3x3

1. Pour un Max-Pooling, il s'agit de la valeur maximal en entrée
2. Pour un Average-Pooling, il s'agit de la moyenne des entrées

Un des intérêts est de réduire les données et les calculs dans notre réseau. Cela permet aussi de réduire le sur-apprentissage en réduisant les paramètres entre deux couches de convolution. La convolution servant à détecter des motifs, seules les valeurs importantes nous intéressent puisqu'elles signifient qu'un motif a bien été détecté. Ainsi, les couches de pooling suivent souvent les couches de convolution. Le max-pooling est souvent utilisé pour relever des motifs détectés tandis que l'average pooling est utilisé pour réduire le nombre de données en conservant au maximum l'information.

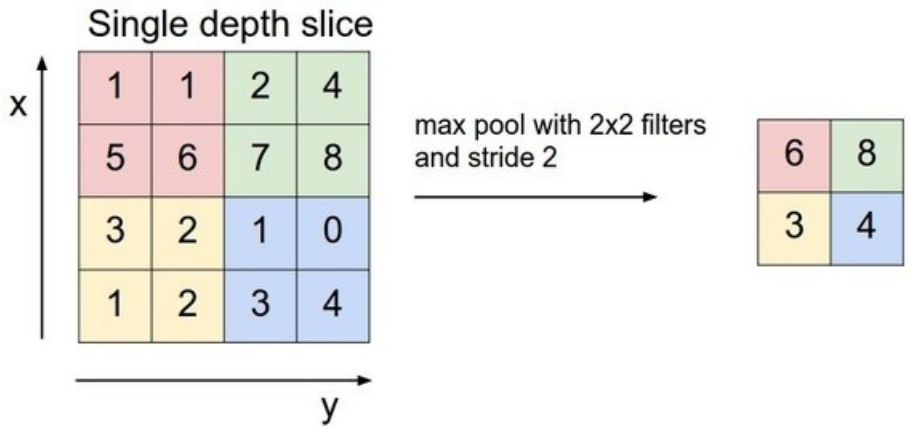


Figure 13: Couche de Max-Pooling avec tuile 2X2

Afin d'éviter de perdre trop de données, il faut privilégier les petites tailles de filtre (2x2).

10.4 Softmax

La sortie d'un classificateur "brut" est un vecteur de valeurs indiquant la force avec laquelle notre réseau associe l'exemple fourni à chacune des classes. La couche de softmax permet de passer du valeur de valeur "résultat" à des probabilités. Pour chaque valeur de sortie de la couche précédente, nous calculons l'exponentiel de celle-ci et nous normalisons les résultats obtenus.

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Figure 14: Mis à l'exponentielle et normalisation

Cela met en avant les valeurs fortes (grâce à l'exponentielle) et que la somme des sorties donne un. Nous pouvons ainsi assimiler la sortie du réseau pour une classe i à la probabilité que l'exemple fourni appartienne à la classe i.

11 Le "Word Embedding" ou représentation des mots

Comme nous l'avons montré plus haut, nos réseaux de neurones ne traite que des réels. Si nous voulons traiter des phrases, il va falloir transformer nos phrases en réels pour les rendre étudiables.

11.1 Généralités

Le Word Embedding est une méthode pour représenter les mots en vecteur. Il s'agît de transformer chaque mot en vecteur de taille définie n. Tous les mots sont donc associé à un vecteur de taille n. Cette taille est un hyper paramètre à étudier lors de nos différents essais afin de déterminer la valeur optimale en termes de résultat/temps de calcul. Nous parlerons par la suite de taille "d'Embedding".
Au début d'un test, nous créons un dictionnaire qui est simplement une matrice de taille [nombre de mot différents dans notre corpus , taille de l'Embedding]. Les valeurs initiales sont aléatoires et sont modifiées durant l'apprentissage au même titre que tous les poids du réseau. Chaque mot correspond à une ligne de notre matrice.

11.2 Word2Vec

Word2vec est un dictionnaire déjà entrainé issu de la mise en commun de deux algorithmes : Continuous Bag Of Word (CBOW) et Skip-Gram, ainsi qu'un important prétraitement. Il a été construit avec un corpus très important. La taille d'Embedding est de 300.
Soit deux phrases :
(1) Arthur va à l'école. Pierre va aussi à l'école.
(2) Arthur va parfois au cinéma.
Le dictionnaire est ['Arthur','va','à','l','école','Pierre','aussi','parfois','cinéma']. Avec les "sac de mots", chaque phrase devient un vecteur de la même taille que le dictionnaire où chaque

valeur indique combien de fois ce mot est présent dans notre phrase. Nous obtenons donc :
(1) → [1,2,2,2,2,1,1,0,0] et (2) → [1,1,0,0,0,0,0,1,1]
Pour un N-gram, nous fournissons N mots consécutifs d'une phrase et l'objectif pour l'algorithme est de déterminer le suivant en se servant du corpus. Un Skip-Gram est une généralisation d'un N-gram avec une séquence de mot non prise en compte. Un k-skip-n-gram a pour objectif de déterminer le mot placé k mots après notre séquence de n mots.
Ces algorithmes sont utilisés afin de rapprocher dans l'espace de projection les mots avec un sens commun ainsi que les mots qui sont souvent proches l'un de l'autre dans une phrase. De nombreuses propriétés apparaissent dans notre espace à 300 dimensions notamment la conservation du lien entre les mots dans l'espace. Par exemple, le vecteur entre "homme" et "femme" est le même que celui entre "roi" et "reine". La force de Word2Vec est d'avoir été lourdement entrainé en conservant à chaque fois les poids. Ainsi, les poids obtenus ont véritablement pu se placer correctement dans cette espace à 300 dimensions.

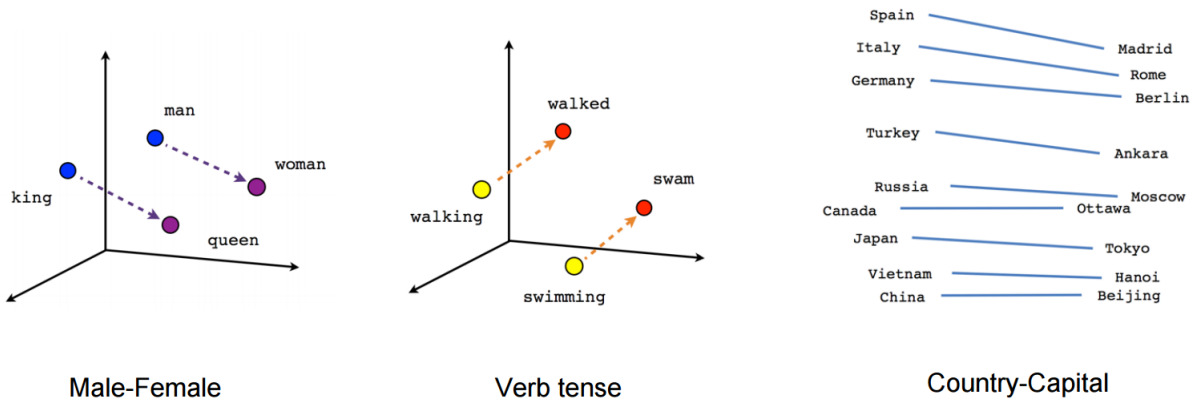


Figure 15: Exemple de correspondance dans les spatialisations sur Word2Vec

Dans nos tests, nous allons utiliser soit un dictionnaire nouveau que nous allons entrainer avec le reste du réseau, soit Word2Vec qui va directement nous fournir les vecteurs correspondants aux mots en entrée.

Part III
Tensorflow et premières applications

Le but de cette partie est de présenter Tensorflow, ses caractéristiques et d'y associer deux exemples d'applications. Ces deux exemples d'utilisation nous ont permis de prendre en main Tensorflow.

12 Tensorflow

Pour la partie programmation de ce projet, j'ai décidé de me servir d'une librairie de Machine Learning : Tensorflow. Développé par Google Brain et totalement Open-Source, Tensorflow est implémenté en Python & C. Il permet de distribuer facilement les calculs entre un ou plusieurs processeurs, cartes graphiques sur un ordinateur ou un réseau. Il est aujourd'hui largement répandu et est utilisé par de nombreuses entreprises comme Google, Safran, AirBNB ou Twitter.

L'avantage le plus évident de TensorFlow est de ne pas avoir à tout implémenter nous même. Nous devons prendre en main l'environnement et l'utiliser au maximum.

TensorBoard

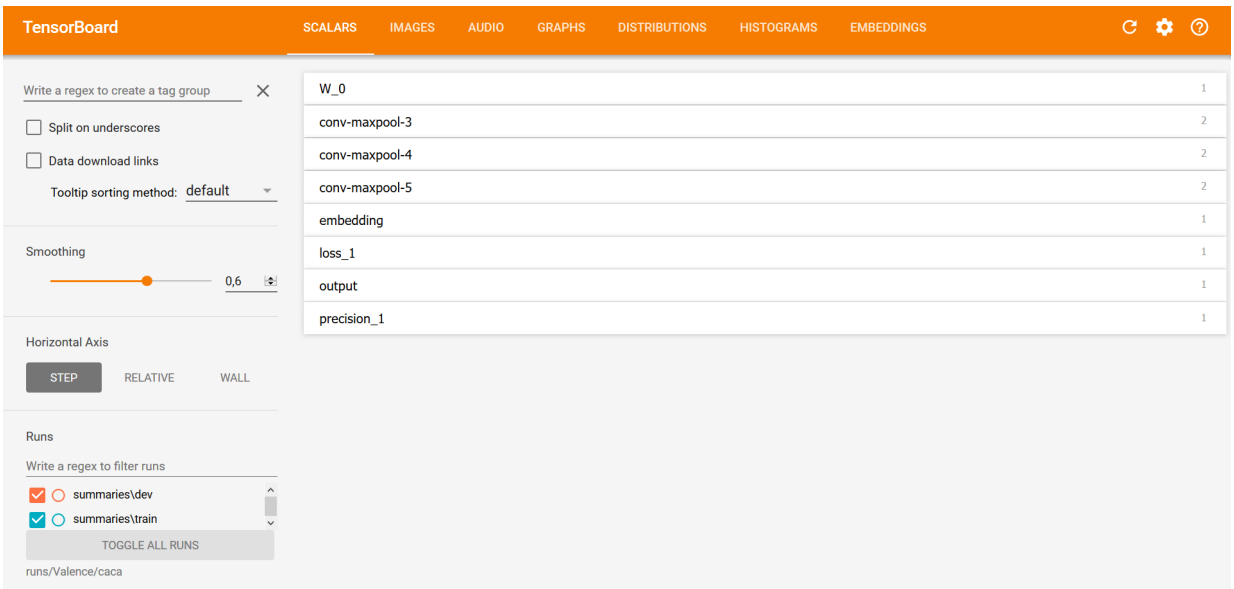


Figure 16: TensorBoard

TensorBoard nous permet de suivre les différentes valeurs de notre modèle, aussi bien durant l'apprentissage que les évaluations.

13 Prise en main sur MNIST

L'un des problèmes les plus classiques du machine Learning est classifier MNIST. Il s'agit de reconnaitre l'écriture humaine pour les chiffres de 0 à 9. Nous disposons pour cela de 55000 images en noir et blanc de 28x28 pixels représentant chacune un chiffre écrit à la main.

Notre image devient un vecteur de 784 valeurs de 0 et 1 (0 si blanc et 1 si noir). Le label est un vecteur de taille 10 avec un 1 dans la position correspondante : [0,0,1,0,0,0,0,0,0,0] correspond à un 2.

Le premier modèle est très primaire. Nous avons une couche "fully-connected" suivie d'une couche de Softmax. La sortie pour la classe i (indiquant que le chiffre est i) est :

$$\text{evidence}_i = \sum_j W_{i,\ j} x_j + b_i$$

La couche de Softmax calcule : $\text{softmax}(x) = \text{normalize}(\exp(\text{evidence}_i))$

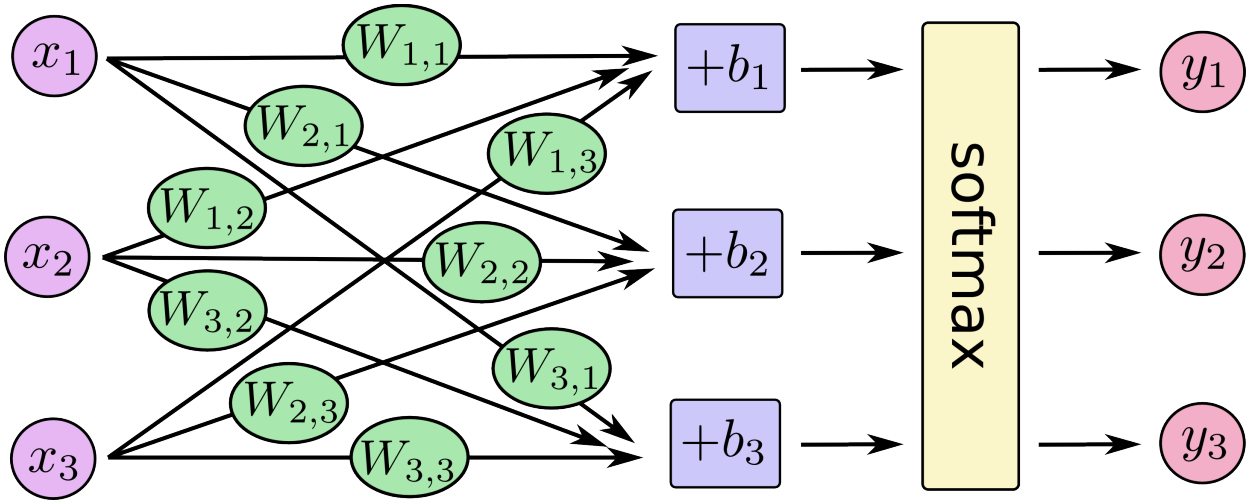


Figure 17: Représentation simplifiée : 3 classes au lieu de 10 et 3 entrées au lieu de 784

Nous utilisons pour calculer l'erreur la fonction la cross-entropie. Cela nous sert à déterminer la distance entre la réponse fournie par le réseau et la réponse attendue. Nous utilisons l'algorithme Adam pour la rétropopagation. Avec ce modèle très simpliste, nous obtenons 92,2 % de précision.

Lors de notre deuxième modèle, nous allons commencer par placer une couche de convolution. Elle va récupérer des tuiles de 5 pixels par 5 pixels. Nous allons calculer 32 convolutions en parallèle pour cette taille de tuiles. La taille de la matrice contenant tous nos poids pour la convolution est [5,5,32]. Nous finissons cette convolution en utilisant Relu comme fonction d'activation. Nous plaçons ensuite une couche de max-pooling avec des entrées 2x2. Nous avons donc 32 matrices de taille [14,14]. Nous appliquons une deuxième couche de convolution avec une même taille de filtre 5x5. Nous recherchons 64 filtres différents donc la matrice est de taille [5,5,32,64]. Nous appliquons encore une couche de pooling 2x2. Nous avons donc 64 matrices de taille 7x7. Nous appliquons maintenant une couche fully connected avec 1024 neurones. Lors de l'apprentissage, nous utilisons une probabilité de drop out de 50% pour toutes ces couches. Nous finissons avec une couche de SoftMax.

Avec ce modèle plus perfectionné, nous obtenons 99,2% de précision. Ces deux modèles sont empruntés au tutoriel officiel de TensorFlow

Part IV
Exploitation

Nous allons maintenant commencer le projet proprement dit. Nous allons présenter l'article qui nous a servi de référence Il contient un modèle de réseau de neurones de compréhenstion du langage. Nous allons ensuite essayer ce modèle sur deux problèmes. Nous allons essayer de determiner si une phrase est "positive" ou "négative", ainsi que si elle est "objective" ou "subjective". Enfin, nous allons étudier plus en détail la performance de notre système en modifiant les paramètres du modèle.

14 Présentation de l'article de référence

Nous allons utiliser un modèle semblable à celui de l'article "Convolutional Neural Networks for Sentence Classification" par Yoon Kim. Dans cette article, l'auteur développe un modèle de réseaux de neurones convolutifs et l'applique à différents problèmes de classification de phrase (Positive/Négative, Subjective/Objective ..). Nous utiliserons les résultats obtenus dans cette article comme référence.

15 Prétraitement

Nos entrées sont des phrases en anglais, il y a donc une importante phase de prétraitement.
Nous pouvons, ou non, modifier les phrases en retirant les stop-words ('a', 'of','the' etc) qui n'apportent pas de sens.
Nous pouvons aussi remplacer les mots par leur racine ('maisons' devient maison, 'ils chasseront' devient 'ils chasser'), cette manipulation est appelée "Stemming". Pour ces deux manipulations, nous utiliserons la librairie Nltk sous Python. Nous retirons aussi les majuscules, les caractères spéciaux...
L'interêt de ces deux manipulations est de réduire le "bruit" pour ne garder que l'essentiel afin de ne pas surcharger notre réseau.
Le dernier prétraitement à effectuer est le passage des mots en vecteur. Nous transformons nos phrases en matrices. Nous allons comparer l'utilisation d'un dictionnaire "neuf", appris au cours de l'apprentissage avec l'utilisation de Word2Vec. La taille de représentation d'un mot en machine est un paramètre important. L'article utilise un vecteur de 100 éléments par mot lorsqu'il construit son propre dictionnaire. Les vecteurs de Word2Vec possèdent eux une taille de 300.
Utiliser un stemming en même temps que le dictionnaire Word2Vec peut s'averer contre-productif. En effet, Word2Vec est complet et possède par exemple trois vecteurs différents pour les mots "Do", "Did" et "Done". Cela sera dommage de volontairement transformer ces trois mots différents en "Do". Lorsque les phrases ne sont pas de la même taille, nous ajoutons pad $à la fin des phrases plus courtes afin qu'elles fassent toutes les mêmes tailles$.

16 Le modèle

Nous allons commencer par appliquer une couche de convolution. Dans l'article, il y a trois tailles de convolution 3, 4 et 5. Cela signifie que notre modèle va lire "ensemble" les mots par 3, par 4 et par 5 en cherchant à détecter des motifs. Nous pourrons tester d'autres tailles.

Pour chaque taille donnée de convolution, l'auteur applique 100 filtres, ce chiffre est aussi un paramètre à faire varier pour comparer les résultats obtenus.
Nous appliquons à la suite de cela une couche de max pooling sur la sortie des convolutions afin de ne récupérer que les grandes valeurs, correspondantes à des motifs détectés. Nous appliquons ensuite une couche fully connected qui passe du vecteur des motifs à un vecteur de taille deux correspondant aux classes.

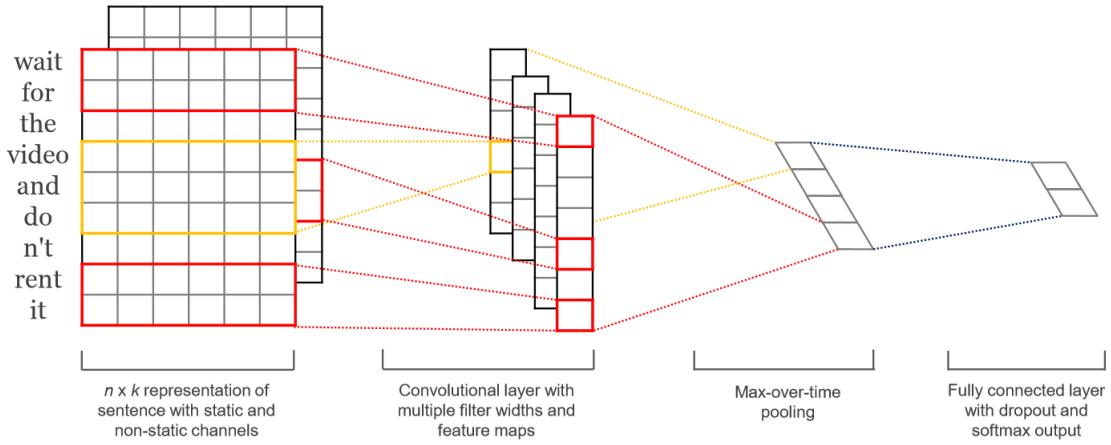


Figure 18: Schéma de notre modèle

17 Fonctionnement

Nous allons construire notre réseau dans le fichier "modeleCNN.py" et le fichier "train.py" nous sert de main (corps du programme).

Lorsque nous voulons lancer un apprentissage, nous commencons par mélanger la base de données avant de la séparer en base d'entrainement (90% des données) et d'évaluation (10%).

Nous allons fournir nos exemples de la base d'aprentissage par paquets (batch). Le nombre d'exemples dans un batch est un hyper paramètre. Une fois toutes nos données fournies, nous mélangeons les exemples de notre base d'apprentissage afin que les batchs lors de l'itération suivante soient différents. Nous recommençons autant de fois que nous le désirons. Le nombre d'itérations sur la base d'apprentissage est important. S'il est trop grand, nous pouvons entrer en sur-apprentissage ou simplement perdre du temps car les résultats ne changent plus. S'il est trop faible et que nous n'obtenons pas des résultats convenables sur notre base d'apprentissage, cela signifie que notre réseau n'a pas fini "d'apprendre". C'est un des hypers paramètres.

18 Hyper paramètres et choix

Voici la liste des hyper paramètres et choix que nous devrons faire et qu'il va falloir discuter et tester afin de les justifier.

Concernant le prétraitement :

1. Conservation ou non des Stop Word
2. Stemming des mots ou non
3. Utilisation de Word2Vec ou Non
4. Taille des vecteurs représentant les mots

Concernant notre modèle :

1. Taille des filtres choisi (2,3,4...)
2. Nombre de filtres par taille

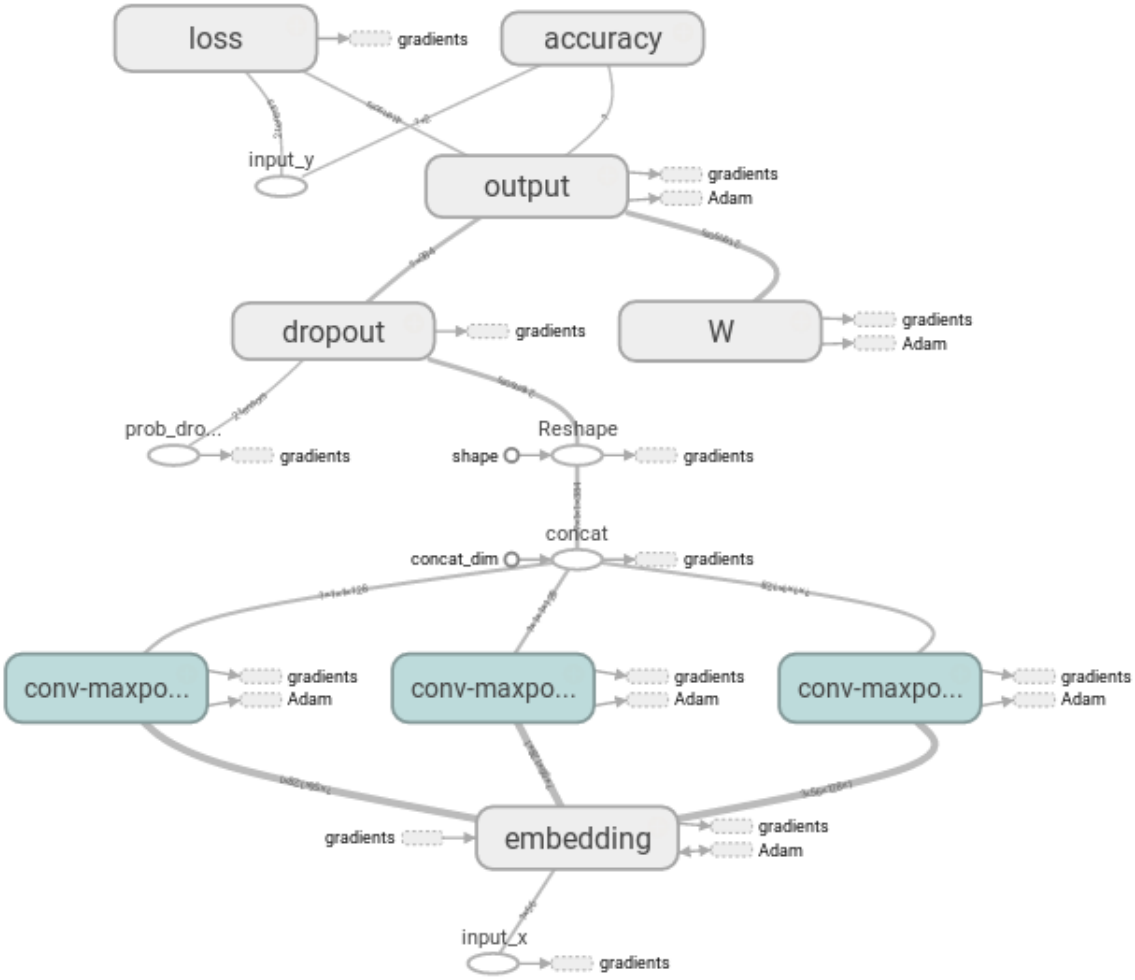


Figure 19: Modèle visualisé sur TensorBoard

3. La probabilité de Drop Out
4. Le lambda pour la L2-regularisation
5. Choix de la méthode de rétropropagation : Momentum, Nesterov accelerated gradient, Adagrad, Adam
6. Choix de la fonction pour le calcul de l'erreur : Cross-Entropy, Erreur Quadratique

Concernant l'apprentissage :

1. La taille des batchs
2. Le nombre de passage sur la base d'apprentissage

En plus d'avoir une influence sur les résultats, ces paramètres ont une influence sur le temps d'exécution.

Sur TensorFlow, nous utilisons les "Flags" comme paramètres globaux.

```
# Parametres
# Donnée
tf.flags.DEFINE_float("percent_validation", .1, "Pourcentage de donnée servant la validation")
tf.flags.DEFINE_string("database",2,"Choix de la base de donnée : 1 - Valence 2 - Subjectivité")
tf.flags.DEFINE_string("dossier_sortie","./runs/Subjective/345_100","Dossier avec le voc, les summaries ...")
tf.flags.DEFINE_string("positive_data_file", "./data/rt-polaritydata/trainPOS.pos", "Adresse pour les données positive (1)")
tf.flags.DEFINE_string("negative_data_file", "./data/rt-polaritydata/trainNEG.neg", "Adresse pour les données negative (1)")
tf.flags.DEFINE_string("objective_data_file", "./data/subjective/plot.tok.gt9.5000", "Adresse pour les données Objectives ")
tf.flags.DEFINE_string("subjective_data_file", "./data/subjective/quote.tok.gt9.5000", "Adresse pour les données Subjective")

# Parametres du prétraitement
tf.flags.DEFINE_boolean("word2vec",False, "Utilisationde Word2Vec ou embedding aléatoire")
tf.flags.DEFINE_boolean("stop_word",False, "Retrait des Stop Word")
tf.flags.DEFINE_boolean("token",False, "Tokenisation des mot")

# Parametres du modèle
tf.flags.DEFINE_integer("embedding_dim",50, "Dimension de la representation des mots")
tf.flags.DEFINE_string("filters", "2", "Taille des filtres")
tf.flags.DEFINE_integer("nb_filter",2, "Nombre de filtre par taille")

# Parametres pour l'entrainement
tf.flags.DEFINE_float("prob_dropout",0.5, "Probabilité de supprimmer un neurones")
tf.flags.DEFINE_float("l2_reg_lambda",3, "L2 regularisation lambda (default: 0.0)")
tf.flags.DEFINE_string("trainMethode","Adam", "Choix entre : Adam et Adadelta")
tf.flags.DEFINE_integer("batch_size",50, "Taille de batch")
tf.flags.DEFINE_integer("nb_iteration",20, "Nombre de fois où on passe sur les données d'entrainement")
tf.flags.DEFINE_integer("evaluate_every",50, "Evaluation toutes les X étapes")
```

Figure 20: Liste de nos Flags

19 Visualisation des résultats

Nous disposons de plusieurs options pour visualiser les résultats obtenus par notre réseau de neurones. Nous avons premièrement TensorBoard qui permet de suivre la précision globale et l'erreur.

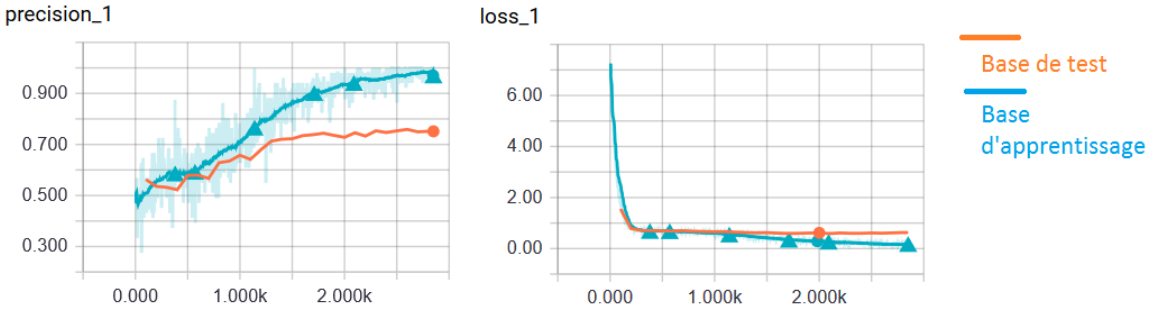


Figure 21: Exemple de résultats obtenus, chaque étape correspond à un batch

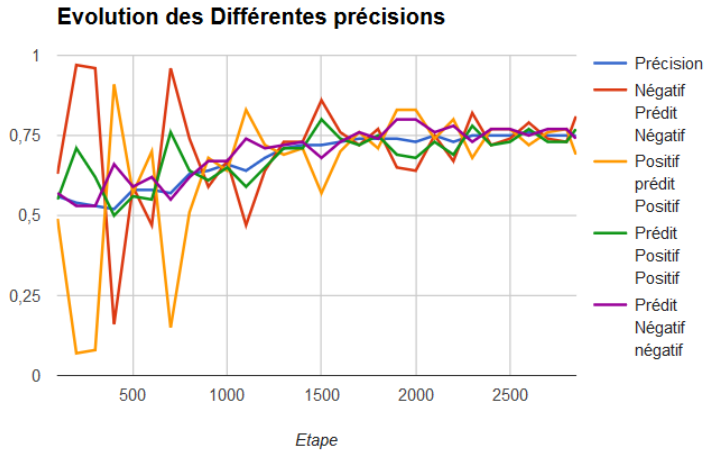


Figure 22: Evolutions des 5 précisions sur la base d'évaluation et durant l'apprentissage

De plus, nous enregistrons trois fichiers durant une exécution. Les deux premiers traquent la précision, les % de vrais positifs, de faux positifs... durant l'apprentissage et lors des évaluations.

Le derniers fichiers nous permet de comparer l'évolution globale de la précision sur la base d'évaluation et d'apprentissage durant l'apprentissage.

A chaque fois, nous enregistrons notre réseau de neurones à la fin de son entrainement. Il nous est donc possible de le recharger sans avoir à l'entrainer de nouveau. Cela peut permettre de lui soumettre une évaluation "personnelle" avec quelques phrases non présentes dans la base de données afin de voir comment notre réseau va les classifier. Le but sera d'essayer d'identifier des limites au réseau en constatant concrètement ce qu'il échoue à identifier correctement. Ces tests ne rentrent pas en compte dans l'évaluation globale des performances et sont juste faits à titre informatif.

Le terminal nous fournit aussi des informations en temps réel. Il nous indique les précisions de chaque batch fourni. Il affiche aussi ces précisions pour les évaluations en cours d'apprentissage et à la fin de celui-ci.

20 Notation de Film - Valence

Dans ce corpus, nous allons utiliser comme données 10 662 avis concernant des films relevés sur le site RottenTomatoes. La moitié de ces avis sont labélisé " Positif" et l'autre moitié "Négatif". Le but du réseau est de déterminer la valence de l'avis, c'est à dire si un avis est

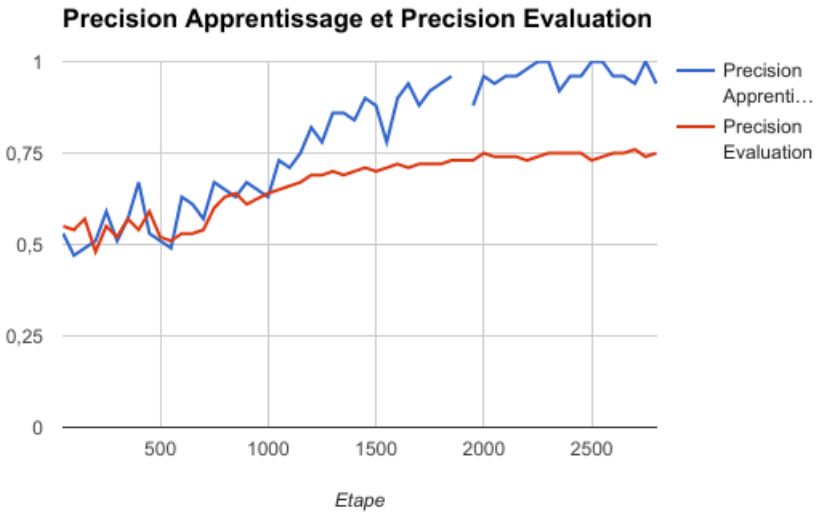


Figure 23: Exemple de l'évolution de la précision

```
Evaluation:
2017-03-25T16:20:27.483549:  acc 0.744841 NpN 0.707804 PpP 0.784466 pNN 0.778443 pPP 0.715044

('2017-03-25T16:20:35.360127',): step 2800, loss 0.203358, acc 0.941176 NpN 0.913043 PpP 0.964286 pNN 0.954545 pPP 0.93
034

Evaluation:
2017-03-25T16:20:35.896507:  acc 0.754221 NpN 0.749546 PpP 0.759223 pNN 0.769088 pPP 0.73913

('2017-03-25T16:20:44.393555',): step 2850, loss nan, acc nan NpN -1 PpP -1 pNN -1 pPP -1

Evaluation:
2017-03-25T16:20:44.914424:  acc 0.755159 NpN 0.764065 PpP 0.745631 pNN 0.762681 pPP 0.747082

Poids enregistrés dans C:\Users\Clement\Desktop\Pfe\Cnn\runs\Valence\345Filtre_15epoch_Batch-50_L2reg-3_Adam_100Emebddi
g_Stopword_token\checkpoints\model

Entrainement fini
Evaluation Finale :
2017-03-25T16:20:46.469177:  acc 0.755159 NpN 0.764065 PpP 0.745631 pNN 0.762681 pPP 0.747082
```

Figure 24: Exemple de sortie du Terminal

positif ou négatif. Chaque avis ne contient qu'une phrase, de 20 mots en moyenne. L'avis le plus long contient 57 mots. Le corpus dans son ensemble contient 18 765 mots différents dont 16 448 sont reconnus dans Word2Vec.
Le pourcentage "positifs prédits positifs" correspond au nombre d'avis correctement identifiés positifs sur le nombre d'avis positifs. Le pourcentage "Prédit Positifs Positifs" correspond au nombre d'avis qui ont été identifiés positifs et qui le sont vraiment sur le nombre d'avis qui ont été prédits positifs..

21 Analyse de la subjectivité ou objectivité

Dans cette section, nous allons utiliser comme données : 10 000 phrases. 5000 sont labélisées "Objectives" et sont issues des synopsis de film sur le site "RottenTomatoes". Les 5000 autres phrases sont "Subjectives" et issues de la section "Commentaire" des film". Il y a un léger croisement dans les données car certains synopsis sont écrits de manière subjective. Ces perturbations sont jugeées négligeables. Le but du réseau est de déterminer si une phrase est objective ou subjective. Chaque avis ne contient qu'une phrase, de 19 mots en moyenne. Le corpus dans son ensemble contient 16 185 mots différents dont 14 838 sont reconnus dans

Word2Vec.

22 Comparatif des résultats avec ceux de l'article

Cnn-Rand correspond à notre utilisation d'un dictionnaire neuf à chaque début d'initialisation. Cnn-Static correspond à l'utilisation de Word2Vec. MR correspond à Movie Rating donc le corpus sur la Valence. Subj correspond au corpus sur la Subjectivité. Les paramètres choisis sont :
Taille de filtre = 3, 4, 5
Nombre de filtres pour chaque taille = 100
Lambda pour la L2-Régularisation = 3
Taille de batch = 50

Model	MR	Subj
CNN-rand	76.1	89.6
CNN-static	81.0	93.0

Figure 25: Résultats obtenus dans l'article

	MR	Subj
Sans Word2Vec	75	89
Sans Word2Vec et avec Prétraiement	76	90
Avec Word2Vec	79	92
Avec Word2Vec et avec Prétraiement	79	91

Figure 26: Résultats que nous avons obtenus

Nous obtenons des résultats équivalents bien qu'inférieur à ceux de l'article. Les auteurs n'ont pas retirer les Stop-Words et utiliser de Stemming. Ainsi, nous dépassons leur précison sur le corpus Subjectivité, avec le prétraitement et sans Word2Vec.

23 Mise en évidence du Sur-Apprentissage

Dans l'exemple suivant, nous avons choisi d'effectuer 30 itérations (epoch) sur notre base d'apprentissage. Voici la courbe de la précision. Chaque étape (step) correspond à l'apprentissage d'un Bash qui était ici de 50 exemples. Nous constatons que la courbe rouge décroit à partir d'un point d'inflexion, c'est à dire que nous obtenons des résultats de plus en plus mauvais sur notre base d'évaluation. Nous avions pour l'exemple désactiv" le Drop-out et mis un Lambda de L2-Régularisation à 0.

Nous constatons bien que la précision sur la base d'apprentissage diminue à partir d'un point d'inflexion.

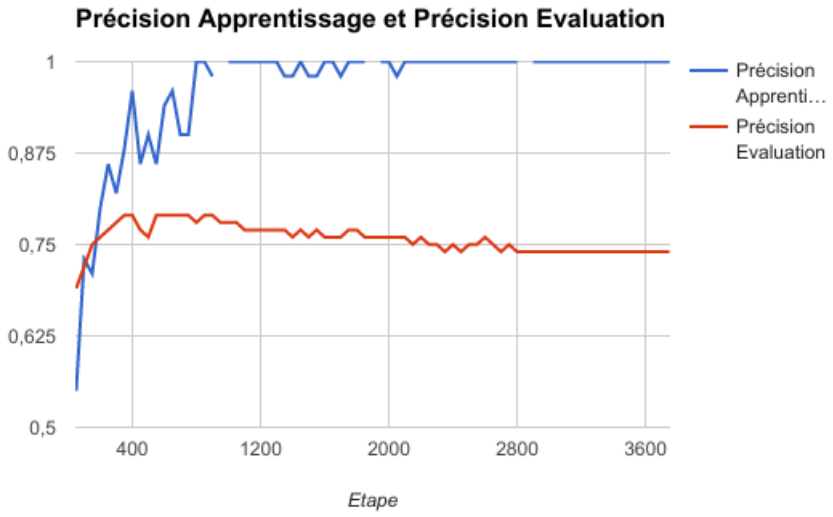


Figure 27: Exemple de sur-apprentissage

24 Test sur les différents paramètres

Notre méthode est simple. Nous devons fixer tous les paramètres sauf un et faire varier celui ci dans l'intervalle des valeurs possibles. Nous devrons ensuite assembler nos résultats et les croiser afin d'obtenir l'ensemble des paramètres fournissant les meilleurs résultats.
En plus de rechercher la combinaison de paramètres donnant le meilleur résultat, nous allons aussi montrer l'influence des paramètres sur la vitesse de convergence, le temps d'exécution...

24.1 Variation de la taille d'Embedding

Pour nos tests sur la taille d'Embedding, nous avons fixé les autres paramètres à :
Stop Word = Non (nous ne les avons pas retirés)
Token = Non (pas de stemming)
Taille des filtres = 3, 4, 5
Nombre de filtres = 100
Probabilité de Drop-Out = 50%
Lambda de L2-Regularisation = 3
Méthode de rétro propopagation = Adam
Fonction pour le calcul de l'erreur = Cross-Entropy
Taille des batchs = 50
Nombre d'époch = 15

Nous n'utilisons pas Word 2 Vec mais un dictionnaire rempli aléatoirement à chaque initialisation.

24.1.1 Taille d'Embedding = 10,50,100 & 300

Les courbes suivantes nous renseignent sur la rapidité de la convergence vers 100% de précision sur la base de d'apprentissage.

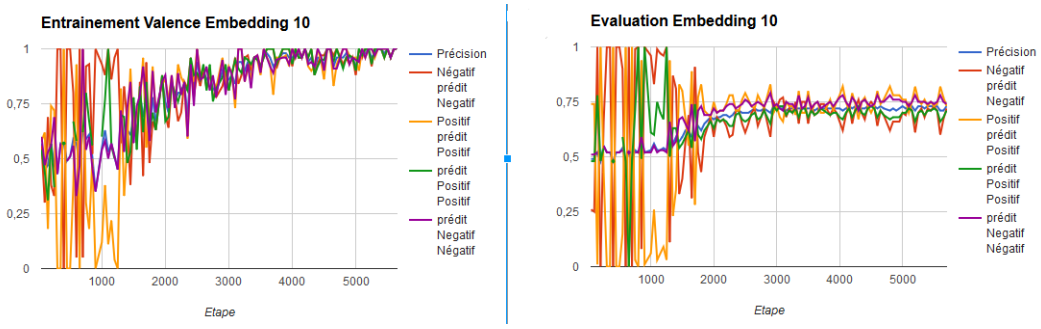


Figure 28: Courbe des 5 précisions sur le corpus "Valence" avec une taille d'Embedding de 10, 5,5 Minutes d'execution

Lorsque nous n'avions utilisé que 15 itérations, notre précision sur la base d'apprentissage n'atteignait pas 100%. Nous avons donc prolongé à 30 itérations. Le temps d'exécution restait faible.

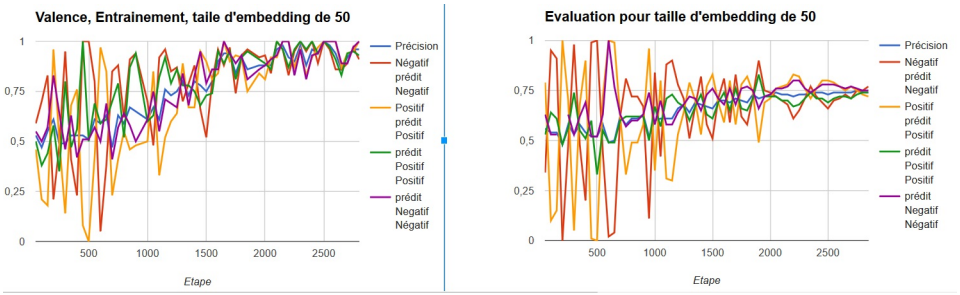


Figure 29: Courbe des 5 précisions sur le corpus "Valence" avec une taille d'Embedding de 50, 11 Minutes d'execution

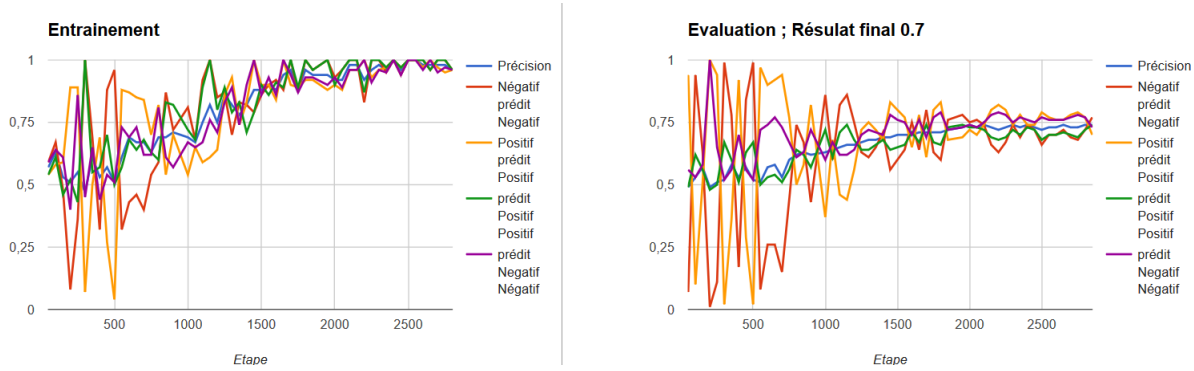


Figure 30: Courbe des 5 précisions sur le corpus "Valence" avec une taille d'Embedding de 100, 10 Minutes d'execution

Plus la taille d'Embedding est importante, plus la précision converge vite. Nous observons les mêmes résultats sur l'autre corpus

24.1.2 Récapitulatif

Tous les résultats finaux présentés sont obtenus à la suite d'une Cross Validation. Ce sont des moyennes des résultats obtenus sur dix apprentissages.

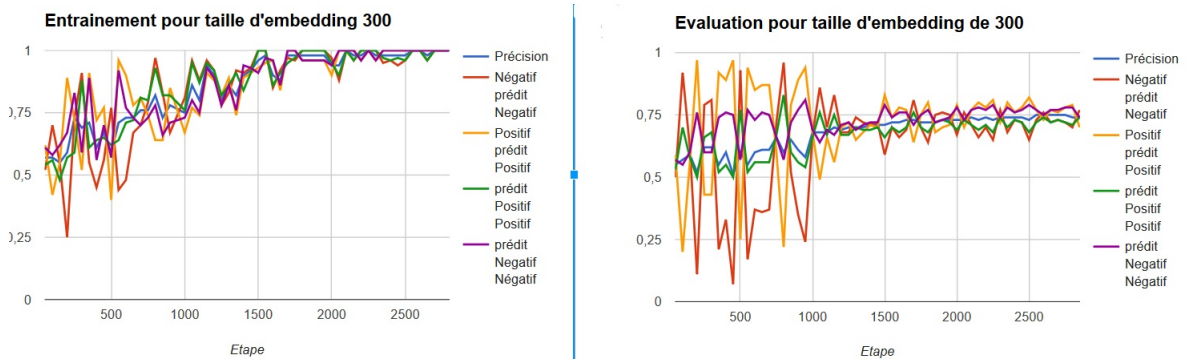


Figure 31: Courbe des 5 précisions sur le corpus "Valence" avec une taille d'Embedding de 300, 24 Minutes d'execution

	Taille d'embedding	Précision	Négatif prédit Negatif	Positif prédit Positif	prédit Positif Positif	prédit Negatif Négatif	Temps d'execution (min)
	10	0,73	0,72	0,73	0,71	0,74	3
	50	0,74	0,74	0,72	0,74	0,75	6
Valence	100	0,74	0,77	0,7	0,74	0,73	11
	300	0,74	0,77	0,7	0,74	0,74	24
	Taille d'embedding	Précision	Subjectif prédit subjectif	Objectif prédit Objectif	prédit Objectif Objectif	prédit subjectif Subjectif	Temps d'execution (min)
	10	0,88	0,91	0,85	0,9	0,86	5
	50	0,88	0,89	0,88	0,89	0,89	6,5
Subjectivité	100	0,89	0,88	0,91	0,88	0,91	12
	300	0,89	0,9	0,89	0,89	0,9	28

Figure 32: Résultats finaux obtenus en faisant varier la taille d'Embedding

Nous constatons que le temps d'exécution est très corrélé à la taille d'Embedding, ce qui est normal puisque le nombre de calculs à effectuer est proportionnel à la taille d'Embedding. En termes de résultat, nous ne constatons pas de grandes différences, tout au plus 0.1% de différence entre une taille de 10 et 300... C'est assez surprenant et montre qu'une complexité plus importante n'est pas une garantie de meilleur résultat.

24.2 Taille des Batchs

Pour nos tests sur la taille des Batchs, nous avons fixé les autres paramètres à :
Taille d'Embedding = 100
Stop Word = Non (nous ne les avons pas retirés)
Token = Non (pas de stemming)
Taille des filtres = 3, 4, 5
Nombre de filtres = 100
Probabilité de Drop-Out = 50%
Lambda de L2-Regularisation = 3
Méthode de rétro propagation = Adam
Fonction pour le calcul de l'erreur = Cross-Entropy

Nous allons tester avec des tailles de batch valant 10,50,100 et 1000.

Plus la taille de batch est importante, plus les itérations sur la base d'entrainement sont rapides. Après 15 itérations, nous n'avons pas atteint le 100% de réussite sur la base d'entrainement. C'est pourquoi nous avons augmenté le nombre d'itérations. Nous obtenons nos meilleurs résultats pour des tailles de batch 50 et 100.

	Taille	Précision	Négatif prédit Negatif	Positif prédit Positif	prédit Positif Positif	prédit Negatif Négatif	Temps d'execution (min)	Itération
	10	0,69	0,65	0,76	0,67	0,74	15	100
	50	0,74	0,77	0,7	0,74	0,73	11	;'
Valence	500	0,73	0,72	0,73	0,71	0,74	5	45
	1000	0,71	0,7	0,72	0,69	0,73	6	
	Taille	Précision	Subjectif prédit subjectif	Objectif prédit Objectif	prédit Objectif Objectif	prédit subjectif Subjectif	Temps d'execution (min)	
	1	0,86	0,94	0,78	0,93	0,82		
	50	0,89	0,91	0,87	0,9	0,88	11	
Subjectivité	100	0,88999999	0,89	0,89	0,89	0,9	15	40
	1000	0,89	0,9	0,87	0,9	0,88	250	100

Figure 33: Récapitulatif des résultats obtenus en faisant varier la taille de Batch

24.3 Word 2 Vec

Pour nos tests sur l'intérêt de Word2Vec, nous avons fixé les autres paramètres à :
Taille des filtres = 3, 4, 5
Nombre de filtres = 100
Probabilité de Drop-Out = 50%
Lambda de L2-Regularisation = 3
Méthode de rétro propagation = Adam
Fonction pour le calcul de l'erreur = Cross-Entropy
Nombre d'epoch = 15

24.3.1 World 2 Vec comparé à un dictionnaire "neuf"

Lors de nos tests avec notre dictionnaire initialisé à chaque fois, nous prendrons une taille d'Embedding de 100, nous avons montré plus haut que cela n'a pas d'influence sur les résultats. Pour l'affichage des résultats obtenus avec Word 2 Vec, nous avons limité le graphe aux 9 premières itérations car ensuite les courbes n'évoluent plus.

La vitesse de convergence est plus rapide avec Word2Vec.

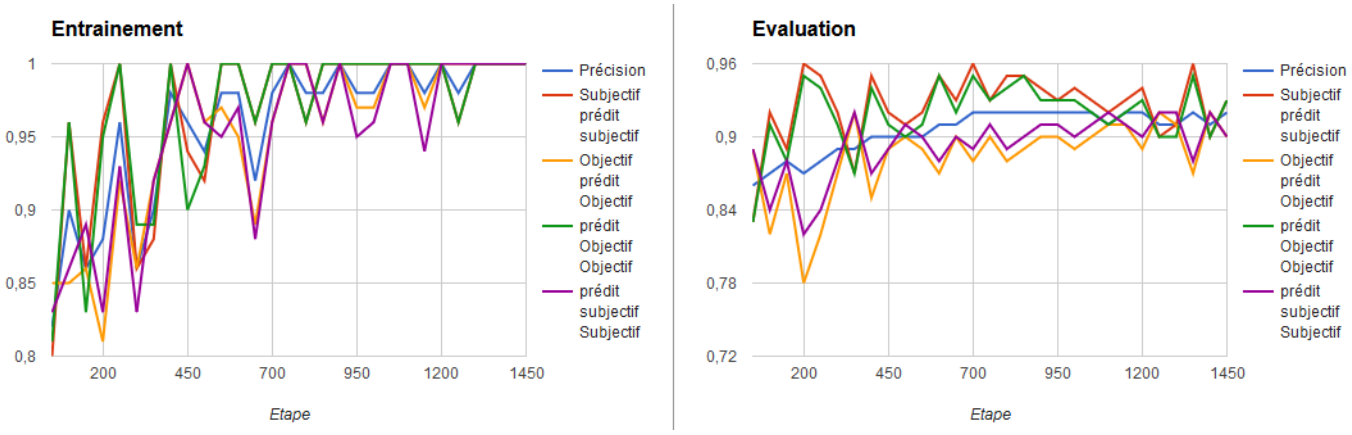


Figure 34: Courbe des 5 précisions sur le corpus "Subjectivité" avec Word2Vec, 35 Minutes

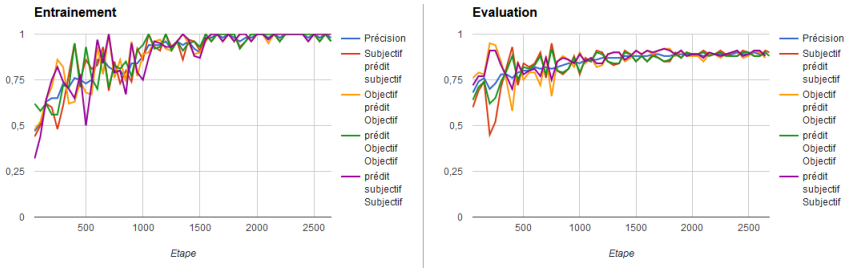


Figure 35: Courbe des 5 précisions sur le corpus "Subjectivité" sans Word2Vec, 11 Minutes

24.3.2 Récapitulatif

	Pretraitement	Précision	Négatif prédit Negatif	Positif prédit Positif	prédit Positif Positif	prédit Negatif Négatif	Temps d'execution (min)
	Word2Vec	0,79	0,79	0,79	0,78	0,8	28
	Word2Vec + StopWord + Token	0,79	0,82	0,76	0,8	0,78	29
Valence	100-Embedding	0,74	0,77	0,7	0,74	0,73	11
	100-Embedding+ StopWord + Token	0,75999999	0,76	0,75	0,75	0,76	8
	Pretraitement	Précision	Subjectif prédit subjectif	Objectif prédit Objectif	prédit Objectif Objectif	prédit subjectif Subjectif	Temps d'execution (min)
	Word2Vec	0,92000002	0,94	0,89	0,93	0,9)	35
	Word2Vec + StopWord + Token	0,91000003	0,92	0,89	0,92	0,9	36
Subjectivité	100-Embedding	0,89	0,88	0,91	0,88	0,91	12
	100-Embedding+ StopWord + Token	0,89999998	0,91	0,89	0,91	0,89	12

Figure 36: Résultats finaux obtenus

Nous obtenons des meilleurs résultats en utilisant Word2Vec. Nous ne constatons pas d'avantage au retrait des Stop-Word et au stemming. Nous pouvons expliquer cela par le faite que les petits mots sont répertoriés dans Word2Vec. Il n'est pas utile de les retirer. De même, les mots sont présents sous toutes leurs formes dans Word2Vec, il n'est pas interressant de les stemmer.
Il est interressant de noter que la "force" de Word2Vec ne vient donc pas du faite qu'à chaque mot correspond un vecteur de 300 valeurs. Nous avons tester nous même avec une taille d'Embedding de 300 et des poids aléatoires: nous n'arrivons pas à un tel score. C'est dans les valeurs que réside la force de Word2Vec, donc dans l'entrainnement du dictionnaire qui a été effectué.

24.4 La taille des filtres pour la convolution et le nombre de filtres appliqués en parrallèle

	Taille de filtre	Nombre de motif recherché	Précision	Subjectif prédit subjectif	Objectif prédit Objectif	prédit Objectif Objectif	prédit subjectif Subjectif	Temps d'execution (min)
	2	100	0,88999999	0,91	0,87	0,9	0,88	4
	2	2	0,81999999	0,82	0,83	0,8	0,84	3
	3	100	0,89999998	0,9	0,89	0,89	0,9	4
	3	2	0,81999999	0,82	0,83	0,82	0,83	3
	3;4;5	500	0,91000003	0,94	0,87	0,93	0,89	40
Subjectivité	3;4;5	100	0,91000003	0,92	0,89	0,91	0,91	8
	3;4;5	2	0,86000001	0,88	0,84	0,87	0,85	5
	2,3,4,5,6	100	0,88999999	0,85	0,93	0,86	0,93	31
	2,3,4,5,6,7,8	100	0,88999999	0,88	0,91	0,87	0,9	22

Figure 37: Résultats finaux obtenus en fonction des tailles de filtre choisis

Nous constatons que les tailles de filtre ont une influence sur les résultats. Cependant, cette influence est relativement faible. Nous obtenons seulement 2% de différence en terme de précision entre des tailles pourtant complétement différentes. Par contre, le nombre de filtres pour chaque taille est lui déterminant. Nous observons une différence de 7% entre les résultats lorsque nous calculons 100 filtres et 2 filtres. Cependant, calculer 500 filtres différents n'a pas amélioré notre précision.

24.5 Quantité de données dédié à l'apprentissage

Nous allons volontairement réduire la quantité de données que nous fournissons pour l'apprentissage. Nous utilisons habituellement 90% de nos données pour l'apprentissage. Nous allons réduire de pourcentage à 25,50 et 75. Le reste de nos données servant pour l'évaluation.

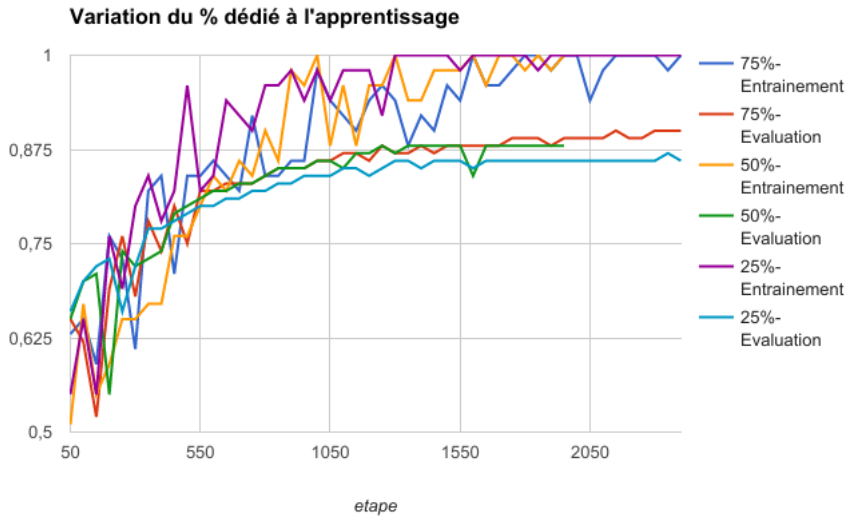


Figure 38: Comparaison des résultats en fonction de la quantité de donnée fournie pendant l'apprentissage

Nous constatons que moins nous fournissons de données à l'apprentissage, moins nos résultats sont bons. En effet, la courbe bleue claire représentant la précision sur la base d'évaluation avec seulement 25% de la base de données utilisée pour l'apprentissage et la courbe rouge représente nos résultat pour 75% de la base notre base de données utilisée pour l'apprentissage. Nous voyons que la courbe bleue est largement en dessous de la rouge. Cela nous confirme dans

notre choix de prendre un maximum de données pour l'apprentissage. Le rapport 90/10 semble parfait.

24.6 Bilan

Nous n'avons pas pu tester tous les paramètres dans leur intervalle de valeur respective. En effet, les temps d'exécution sont très longs. Néanmoins, nous avons présenté la méthode pour determiner les meilleurs paramètres. Les quatres autres précisions ("Positifs prédit positifs","Négatifs prédit Négatifs" ...) n'ont pas fait apparaitre de véritable différence exploitable dans nos résultats.

25 Test avec nos propres phrases

Nous avons soumis 10 avis de films afin de voir comment notre réseau les traiterait. Nous avons essayé en ayant entrainé notre reseau avec un dictionnaire neuf et avec Word2Vec. Le reste des paramètres choisis était :
Taille des filtres = 3, 4, 5
Nombre de filtres = 100
Probabilité de Drop-Out = 50%
Lambda de L2-Regularisation = 3
Méthode de rétro propagation = Adam
Fonction pour le calcul de l'erreur = Cross-Entropy
Nombre d'epoch = 15

Phrase	Affectiation sans TensorFlow	Affectiation avec TensorFlow
it was the worst movie ever	Négatif	Négatif
i can't belive i pay to see that movie , i rather die then see it again	Négatif	Négatif
Nobody can save this film , the actors were bad and don't mention the movie maker	Négatif	Négatif
it was not funny it was not scary it was just ridiculous during two hours	Négatif	Négatif
i hate this movie so much	Négatif	Négatif
it's an amazing movie .	Positif	Positif
i love this movie so much .	Négatif	Négatif
First i was afraid but it turn out to be interesting .	Positif	Positif
it was pleasant moment and i forgot my sad life during 2 hours .	Positif	Négatif
I think it was way better than the book which is overrated to me .	Négatif	Négatif

Figure 39: Exemple de résultat fourni par notre réseau

Il est surprenant de constaster que "I love this movie so much" a été constament considéré comme négatif. La phrase étant très exagéré, le réseau y a peut être vu de l'ironie. Nous ne savons pas si nous avons de l'ironie dans notre

Etant donnée que les batchs soumis ne sont pas exactement les mêmes entre deux entrainement, nous n'obtenons pas exactement le même réseau après deux entrainnements. Les résultats pourraient être différents si nous essayons sur un nouveau réseau. Neanmoins, si le nombre d'itération est assez important, les réseaux tendent vers les mêmes valeurs pour les poids, donc un traitement identique d'un même exemple. Ici, nous avons tester sur trois réseaux différents et nous avons obtenu les mêmes résultats.

Part V
Conclusion

Ce projet fût très interressant à mener. N'ayant eu aucun cours sur le Machine Learning j'ai du me former à la thérorie à l'aide d'articles et de cours sur Internet.
Il est important d'étudier complétement les différents aspects, même les plus théoriques comme les méthodes de rétro propagation, pour comprendre vraiment les choix pris durant l'implémentation.
Concernant l'implémentation sur TensorFlow, c'est la première fois que j'utilisais un environnement aussi complet. C'est une expérience assez lointaine de toute la programmation que j'avais pu réaliser jusque là. Il n'y avait pas de problème d'algorithmie complexe à gérer. Il fallait par contre prendre en main toutes les commandes propres à TensorFlow. Là encore, il a fallu s'entrainer par des petits problèmes avant commencer le projet proprement dit.
Le modèle que nous avons présenté de réseau convolutif, bien que structurellement simple, nous a permis d'obtenir des résultats très satisfaisants sur nos deux problèmes de classification.
L'étude détaillée des différents résultats a permis de déterminer les paramètres ayant le plus d'influence sur les résultats. Ainsi, le nombre de motifs recherchés est plus important que la taille des filtres.
Les réseaux de neurones et la compréhension du langage sont deux domaines passionnants que je compte continuer à étudier à travers mon stage à venir et les emplois qui suivront !

References

[AR] Vincent Guigue et Patrick Gallinari Abdelhalim Rafrafi. "Réseau de neurones profond et SVM pour la classification de sentiments". In: ().

[DTL] Bing Qin Duyu Tang and Ting Liu. "Deep Learning for sentiment analysis: successful approaches and future challenges". In: ().

[Kim14] Yoon Kim. "Convolutional Neural Nerworks for Sentence Classification". In: (2014).

[NS14] Alex Krizhevsky Ilya Sutskever et Ruslan Salakhutdinov Nitish Srivastava Geoffrey Hinton. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *ournal of Machine Learning Research* (2014). DOI: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

[Rec] "Recognizing Emotion in Short Texts". In: ().

[Rud] Sebastion Ruder. "An Overview of gradient descent optimization algorithms". In: (). DOI: sebastianruder.com/optimizing-gradient-descent/index.html#batchgradientdescent.

[Ten] "Implementing a CNN for Text Classification in TensorFlow". In: (). DOI: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/.

[Web] "Overfitting and Regularization". In: (). DOI: http://neuralnetworksanddeeplearning.com/chap3.html#overfitting_and_regularization.