Rapport de projet

Sorbonne Université Sciences et Ingénierie

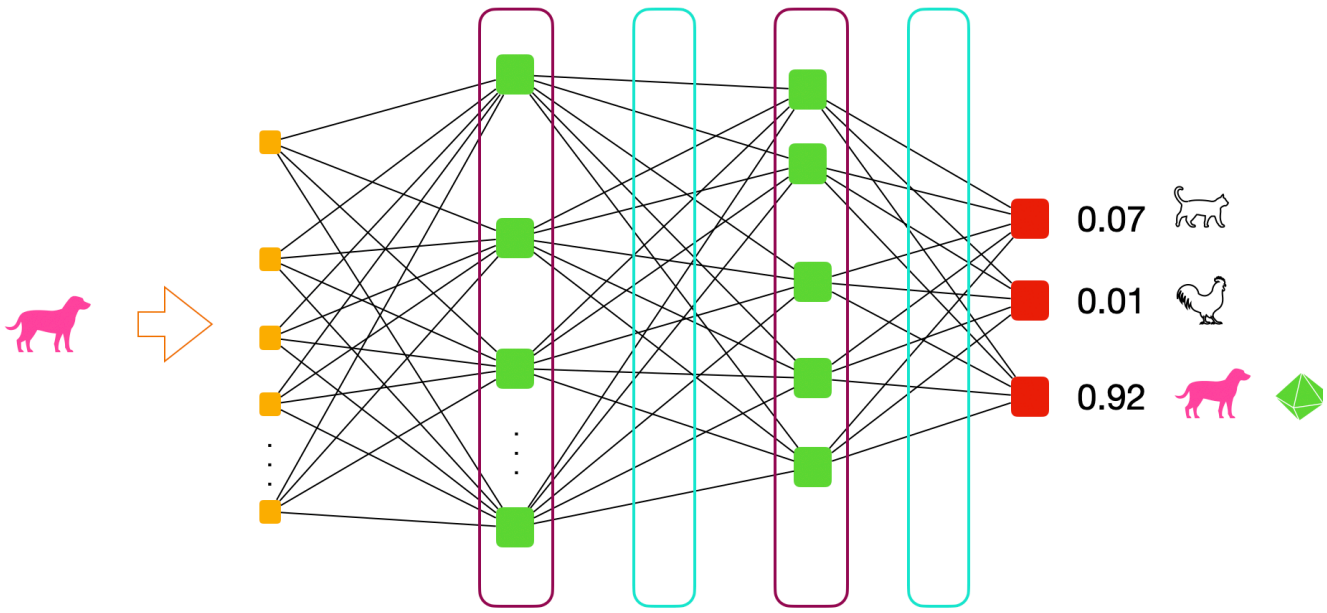
Master Données Apprentissage et Connaissances semestre 4

Machine Learning

Souleymane MBAYE - 28717871 - Gr2

Responsable : Nicolas Baskiotis

- 27 avril 2023



Introduction

Au cour de mon Master d'informatique à Sorbonne Université au semestre 2, il m'a été demandé de réaliser un projet en Machine Learning.

Ce dernier porte sur l'**implémentation d'un réseau de neurones.** Ici, l'implémentation est inspirée des anciennes versions de **PyTorch** (en Lua avant l'autograd) et des implémentations analogues qui permettent d'avoir des réseaux génériques très modulaires. Chaque couche du réseau est vu comme un module et un réseau est constitué ainsi d'un ensemble de modules. En particulier, les fonctions d'activations sont aussi considérées comme des modules (comme l'indique la figure 1).

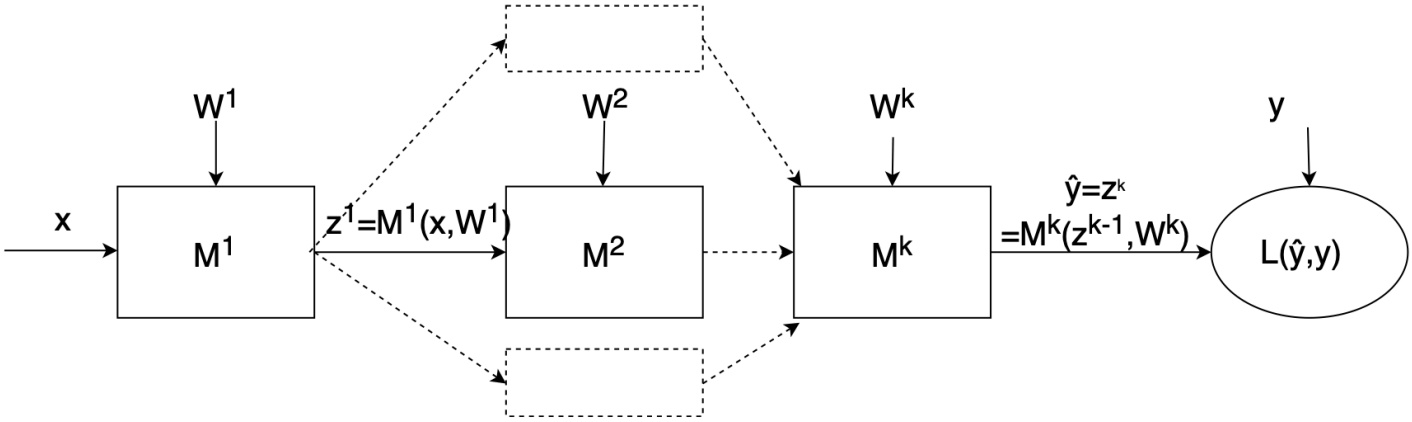


FIGURE 1 – Architecture module d'un réseau

NB:

- Le langage de programmation utilisé est Python avec sa version 3.9.16.
- J'utilise également la bibliothèque Numpy (version 1.22.4) pour des calculs et matplotlib (version 3.4.3) pour des affichages graphiques.
- J'ai organisé le code en trois fichiers python: Modules.py qui contient une classe abstraite Module et dont héritent toutes les autres classes de modules spécifiques comme Linear, Convolution … et regroupe des fonctions auxiliaires que j'utilise par la suite dans le code; Loss.py qui contient une classe abstraite Loss et dont toutes les classes calculant une loss héritent.
- Pour chaque partie, des expérimentations sont faites dans un fichier notebook.
- En fin, par rapport à l'architecture du code, il m'a spécifiquement été demandé de faire moins de boucles possibles. C'est pour cela que j'ai opté un chainage entre les modules dans un réseau. Dans chaque module, j'y ajoute un pointeur vers le module précédent dans le réseau et un autre vers le suivant.

Partie 1: Module Linéaire

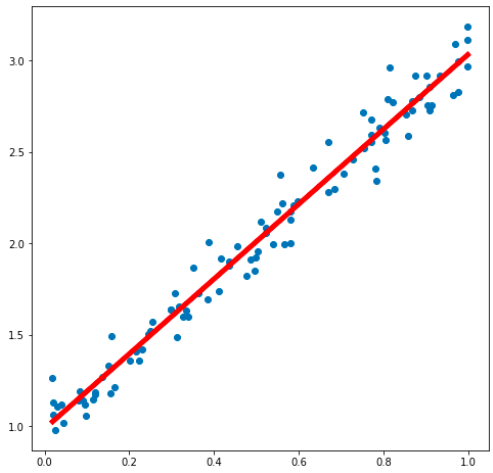
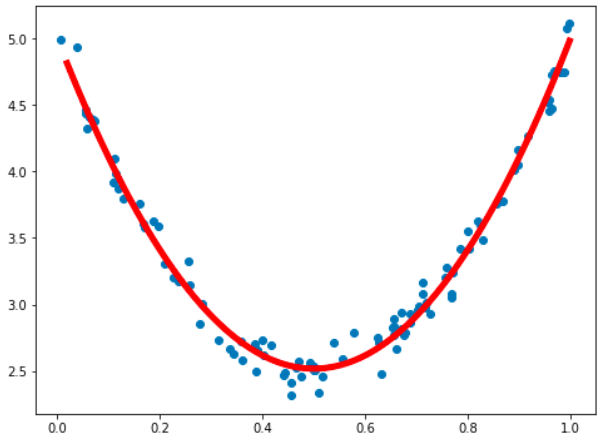
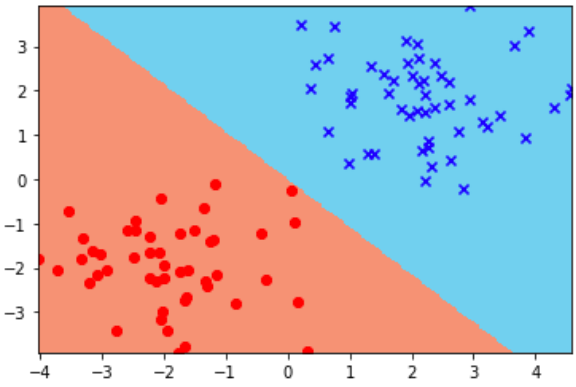
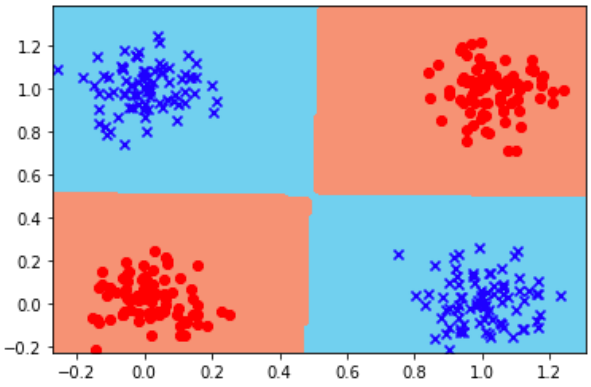
La première partie du projet consiste à coder les deux classes nécessaires pour faire une régression linéaire.

- La classe de coût MSELoss qui hérite de la classe Loss dont la méthode **forward(y, yhat)** calcule $\| y - yhat \|^2$. La supervision y et la prédiction yhat sont des matrices de taille batch x d. La fonction rend un vecteur de dimension batch (le nombre d'exemple).
- Le module linear(input,output) qui représente une couche linéaire avec **input** entrées et **output** sorties. La méthode **forward** prend une matrice de taille batch x input et produit une sortie batch x output.

Dans le fichier notebook **projet-premier-Mbaye.ipynb,** j'ai testé le module linéaire avec la MSELoss sur la classification des chiffres 1 vs 2.
J'ai créer quelques fonctions dans le notebook :

- creer_batches(X,Y,batch_size=5) qui crée des batchs de taille batch_size;
- train(batches,model,loss_fn,lr=1e-3,v=False)
- test(batches,model,loss_fn,v=False)

J'ai entrainé le réseau sur 50 epochs sur les données train. Cela donne un résulta de **99.9**% de bonne prédiction en train et **99.6**% en test.

Partie 2: Non Linéaire

Dans cette partie j'ai implémenté les modules:

- **Tanh** qui permet d'appliquer une tangente hyperbolique aux entrées
- **Sigmoide** qui permet d'appliquer une sigmoïde aux entées.

Dans le notebook projet-deuxieme-Mbaye.ipynb, j'ai testé le module sur un réseau à deux couches linéaires avec une activation tangente entre les deux couches et une activation sigmoïde à la sortie.
Les données utilisées sont les chiffres 1 contre les chiffres 2. Les chiffres 1 sont la **classe 0** et les chiffres 2 représentent la **classe des 1.**
Avec les mêmes fonctions de la partie précédente, j'ai entrainé le réseau sur **50 epochs** sur les données et les j'ai obtenu **99.6**% de bonne classification en train et **99.6**% de bonne classification en test.

Partie 3: Encapsulage

Pour éviter les opérations de chainage répétitives entre les modules lors de la descente de gradient, que cela soit sur la passe forward ou backward, j'ai codé la classe **Sequentiel** qui permet d'ajouter des modules en série et qui automatise les procédures **forward** et **backward.** J'ai préféré chainer les modules et ajouter deux méthodes dans la classe **Module: passe_foraward et passe_back** qui applique la méthode locale (forward pour la première et backward pour la deuxième) et appelle la fonction sur le module suivant (passe forward) ou précédent (passe backward).
J'ai également la classe **Optim(net,loss,eps)** pour condenser une itération de gradient: elle prend dans son constructeur un réseau net, une fonction de loss et un pas de mis à jour du gradient eps. Elle contient une seule méthode **step(batch_x, batch_y)** qui calcule la sortie du réseau sur **batch_x,** calcule le coût par rapport aux labels **batch_y**, exécuté la passe backward et met à jour les paramètres du réseau.
J'ai également implémenté la fonction **SGD** qui prend en entrée un réseau, un jeu de données, une taille de batch et un nombre d'itérations et s'occupe du découpage en batch du jeu de données (avec la fonction creer_batches) et de l'apprentissage du réseau pendant le nombre d'itérations spécifié.
Dans le notebook projet-troixieme-Mbaye.ipynb, j'ai testé le réseau précédent.

Partie 4: Multi-Classes

Dans cette partie, nous gérons le multi-classes. Ce dernier utilise en sortie de réseau une dimension par classe pour dénoter la probabilité de chaque classe. Nous n'avons pas transformer le vecteur de supervision en encodage one-hot, de par ce que ce n'est pas nécessaire mais aussi cela nous évite des multiplication par zéro inutile. Nous utilisons un **Softmax** que nous introduisons à la dernière couche pour transformer les entrées en distribution de probabilités grâce à la normalisation effectuée.
Nous n'utilisons pas la MSE comme coût, nous aurions pu, pvrcequ'il n'est pas très adapter vu qu'il a tendance à moyenner les erreurs et ne pousse par les sorties vers 0 ou 1, nous utilisons la **cross entropie** qui est plus adaptée aux distributions de probabilités.
Pour se faire nous avons coder:

- Nous avons coder le module **Softmax** qui permet d'appliquer un soft-max aux entrées.
- La classe **CELoss** qui passe un logarithme et un coût crossentropique.

Expérimentations:

Dans le fichier notebook projet-quatrieme-Mbaye.ipynb, nous expérimentons le multi-classes sur un problème de classification des chiffres de 0 à 9. Le réseau est composé de cinq modules: lineaire(256,16), activation tangente, lineaire(16,10), activation sigmoïde et un softmax en dernier avec une fonction d'erreur cross entropique. Nous entrainons le réseau sur **10 000 epochs** avec un **batch_size** de taille **5.** Nous obtenons à la fin un taux de

97.8% de bonnes classifications sur les données d'entrainement et un taux de **93.7**% sur les données de test.

Partie 5: Compression

Cette partie est consacrée à un réseau de neurones pour l'encodage et le décodage des données. Un auto-encodeur est un réseau de neurones dont l'objectif est apprendre un encodage des données dans le but généralement de réduire les dimensions. Il s'agit d'apprentissage non supervisé: il n'y a pas de classes associées aux entrées lors de l'apprentissage. Un auto-encodeur est former de deux parties:

- Un encodeur qui prend en entrée une donnée et la transforme dans un espace de plus petite dimension pour obtenir une représentation latente de l'entrée.
- Un décodeur qui prend en entrée la représentation latente d'une donnée, avec généralement un réseau symétrique à l'encodeur, et décode l'exemple vers sa représentation initiale.

L'entrainement se fait sur un coût de reconstruction: $\mathbf{L(X,\hat{X})}$, avec $\hat{X}$ = décodeur(encodeur(X)) la donnée reconstruite.
Nous utilisons comme coût de reconstruction la cross entropie binaire qui se révèle plus performante que celui aux moindres carrés car elle essaye de pousser les sorties vers 0 ou 1. La cross entropie binaire s'utilise des données comprises entre zéro et un, ainsi nous normalisons en conséquence notre jeux de données pour la faire fonctionner. Pour éviter des valeurs infini avec le passage du logarithme nous applique un np.clip sur les entrées avec une valeur minimale de 10^{-10} et maximale de $\mathbf{1}\text{-}10^{-10}$.
Vu que le réseau d'encodage et de décodage sont symétrique, nous utilisons des matrices de poids transposées du réseau d'encodage dans celui de décodage, ce que l'on appelle un partage de poids dans le réseau.

Expérimentations:

Dans le fichier notebook projet-cinquieme-Mbaye.ipynb nous faisons plusieurs expérimentations notamment: la visualisation des images reconstruite après une forte compression, étude du Clustering induit dans l'espace latent avec du k-means en utilisant les classes des exemples et étudions la performance des représentations apprises, visualisation les représentations obtenues dans un espace en deux dimensions avec l'algorithme de **t-sne**, étudions les performances en débruitage, nous ajoutons du bruit aux données et observons l'erreur entre les données débruitées par le réseau et les données originales et enfin étudions les performances en classification en utilisant comme représentation la représentation latente d'une part et d'autres part les données originales et faisons des comparaisons entre les deux.

Partie 6: Convolution

Dans cette partie nous traitons d'une couche convolutionnelle en deux dimensions. Les couches convolutionnelles sont le standard en classification d'images. Une couche convolutionnelle permet d'appliquer un filtre sur différentes parties de l'image et de sortir une valeur par sortie.
Nous avons plutôt choisit de faire de la convolution en deux dimensions. Cependant l'apprentissage prend plus de temps. Car, certainement due au fait que nous avons écrit le code avec des boucles et pas avec des multiplications matricielles.

Expérimentations:

Dans le fichier notebook projet-sixieme-Mbaye.ipynb nous avons fait beaucoup d'expérimentations sur plusieurs bases d'images: usps, mnist et flashion-mnist; avec les modules d'activations MaxPool, ReLU et AveragePool.

Convolution

Ce projet a été très intéressant pour moi dans la mesure où il m'a permis de connaitre un peu plus l'architecture d'un réseau de neurones en profondeur, le mécanisme mis en place, son entrainement, l'optimisation. Et aussi, la convolution qui est un standard dans la classification d'images.
La principale contrainte rentré est celle du temps. Les expérimentations prennent beaucoup de temps à tourner et surtout la partie sur les convolutions. Je suis personnellement plus intéresse par l'algorithmie et dès que je teste sur un exemple et que ça marche je passe.