École Polytechnique de Montreal

Rapport TP1 INF8225

Réseaux de Bayes et Régression linéaire

Valentin BOUIS
1733927

Professeur : Christopher PAL

À remettre le : 4 Février 2019

Partie 1

a) P(W = 1)

D'après la loi des sommes :

$= \sum_P P(W = 1, P)$

$= P(W = 1, P = 1) + P(W = 1, P = 0)$

$= P(W = 1|P = 1)P(P = 1) + P(W = 1|P = 0)P(P = 0)$

$= 1 \times 0.2 + 0.2 \times 0.8$

$= 0.36$

b) P(W = 1 | H = 1)

$= \frac{P(W=1,H=1)}{P(H=1)}$

Calculons P(H) :

$= \sum_P \sum_A P(H, P, A)$

$= \sum_P \sum_A P(H|P, A)P(P)P(A)$

On obtient le tableau de probabilités jointes suivant :

	P=1	**P=1**	**P=0**	**P=0**
	A=1	**A=0**	**A=1**	**A=0**
H=1	0.02	0.18	0.072	0
H=0	0	0	0.008	0.72

Ainsi P(H = 1) correspond à la somme de la première ligne du tableau :

P(H = 1) = 0.02 + 0.18 + 0.072 = 0.272

Calculons P(W = 1 , H = 1) avec la règle du produit d'un réseau de Bayes :

$= \sum_P \sum_A P(W = 1|P)P(H = 1|P, A)P(P)P(A)$

$= 0.2144$

On obtient donc :

$P(W = 1|H = 1) = \frac{0.2144}{0.272} = 0.788$

c) P(W = 1 | H = 1, A = 0)

Logiquement puisque Holmes ne peut etre mouillé que par la pluie et l'arroseur, si l'arroseur est éteint alors il pleut forcément. Si il pleut alors Watson sera forcéement mouillé. Regardons ca par le calcul :

$= \frac{P(W=1,H=1,A=0)}{P(H=1,A=0)}$

$= \sum_P \frac{P(W=1,H=1,A=0,P)}{P(H=1,A=0,P)}$

$= \sum_P \frac{P(W=1|P)P(H=1|A=0,P)P(P)P(A=0)}{P(H=1|A=0,P)P(P)P(A=0)}$

On obtient des probabilités conditionnelles qu'on connait, il nous suffit de sommer pour les différentes valeurs de P (1 et 0)

$= \frac{1\times1\times0.2\times0.9}{1\times0.2\times0.9}$

$= 1$

d) P(W = 1 | A = 0)

Connaitre l'état de A ne va pas influencer sur la probabilité de W. En effet W et A sont conditionnellement indépendants. Ainsi cela revient à calculer

$= P(W = 1)$

$= 0.36$

e) P(W = 1 | P = 1)

C'est une probabilité conditionnelle qui nous est donnée

$= 1$

Partie2

Nous entrainons ici un réseau de neuronnes à l'aide de la technique du "Descent Gradient". L'objectif est de trouver les meilleurs poids W et biais b afin de reconnaitre des images de chiffres manuscrits. Pour cela on cherche à minimiser la fonction de perte qui indique à quel point le réseau a été non-performant.

Le principe sera alors de calculer le gradient sur la fonction de perte et d'en prendre l'inverse afin de descendre vers un minimum local. On l'ajoute alors à nos poids W.

Dans cette version de l'algorithme, nous mettons à jour nos paramètres à chaque fin de minibatch. Nous verrons la performance de notre programme selon plusieurs tailles de minibatch.

Remarques sur le code :

Dans la fonction softmax(x), il est possible de simplifier la fraction en retirant le max(x) de l'exponentiel. Le souci est que sans lui l'exponentiel peut éventuellement renvoyer des valeurs trop grandes à gérer pour python et celui-ci nous lance une erreur.

La fonction getaccuracy(X,y,W) marche de la facon suivante : Pour chaque valeur de X donné on compute la valeur prédite de y à l'aide des poids W. On regarde ensuite si l'indice de la probabilité maximum prédite correspond bien au chiffre à deviner dans y. On renvoie le ratio de bonnes prédictions.

La fonction getloss(y,ypred) représente une fonction de log vraissemblance négative. Elle renvoie le log de la probabilité correspondant au chiffre à deviner.

La fonction d'activation donnée dans l'énnoncé est un mélange de softmax et de la fonction de perte par cross-entropy. Il faut faire attention de l'appeler en lui donnant seulement Wx + b (sans les y)

Ci-dessous nos courbes représentant l'évolution de la fonction de perte pour le set d'apprentissage et le set de validation, ainsi qu'une visualisation des poids pour le chiffre 4 pour différentes valeurs de taille de minibatch et de learning rate :

Remarquons que pour une taille de minibatch de 1 l'entrainement ne peut pas etre efficace puisque le gradient est réinitialisé à chaque fois il n'a pas le temps de trouver le minimum. Ainsi il arrive que les prédictions soient trop proches de 0 et que le log() renvoie des valeurs trop grandes pour python.

learning rate : 0.1, taille minibatch : 20

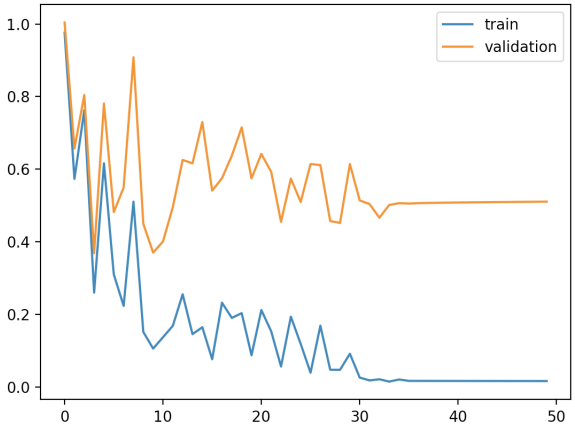

FIGURE 1

learning rate : 0.1, taille minibatch : 89

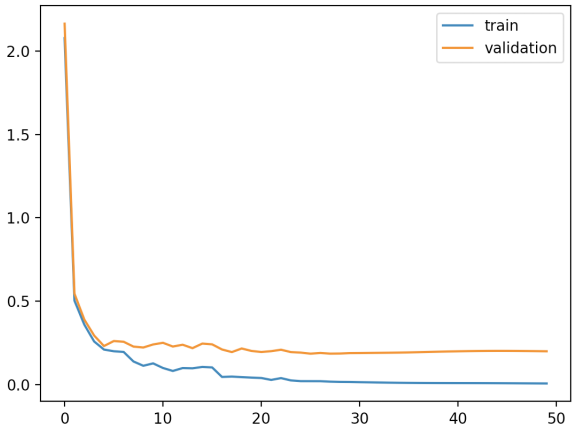

FIGURE 2

learning rate : 0.1, taille minibatch : 200

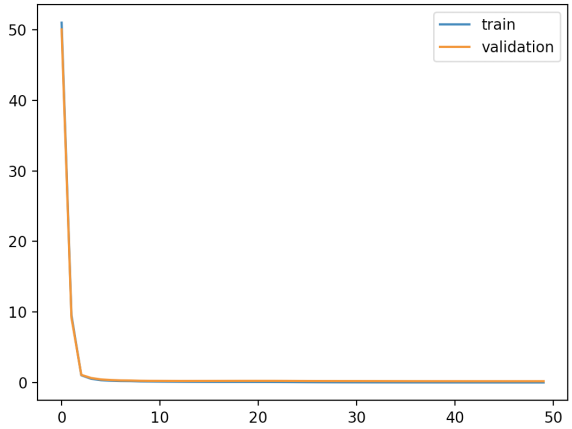

FIGURE 3

learning rate : 0.1, taille minibatch : 1000

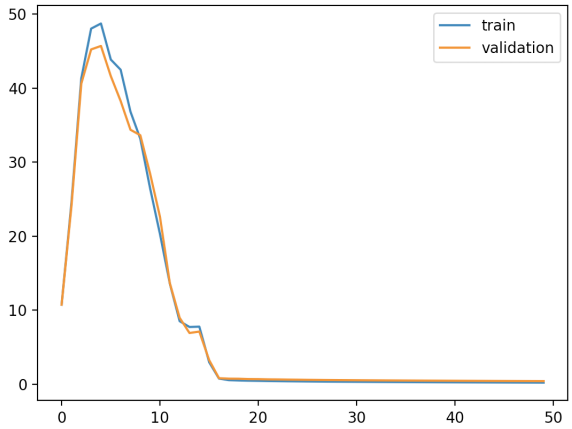

FIGURE 4

learning rate : 0.01, taille minibatch : 20

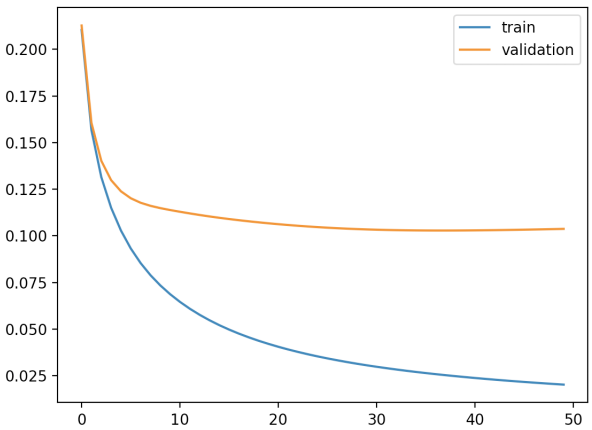

FIGURE 5

learning rate : 0.01, taille minibatch : 89

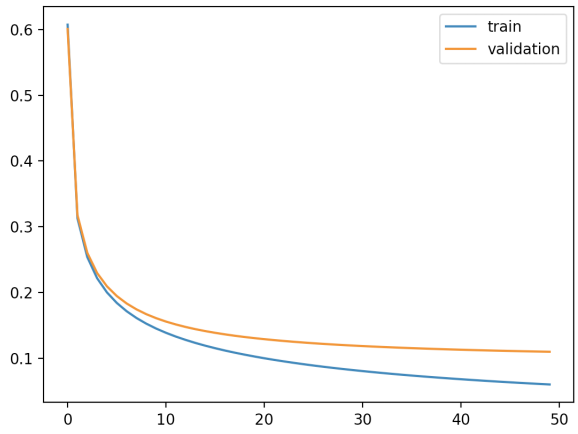

FIGURE 6

learning rate : 0.01, taille minibatch : 200

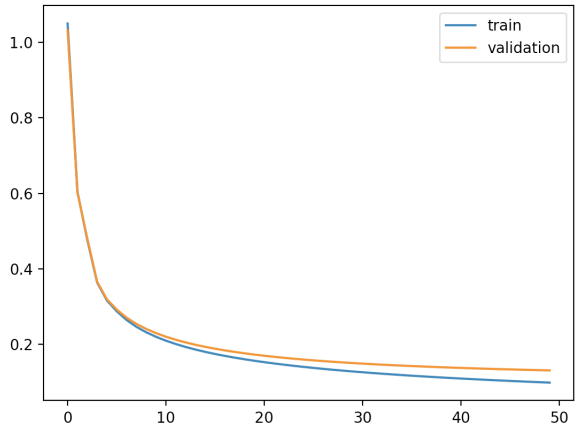

FIGURE 7

learning rate : 0.01, taille minibatch : 1000

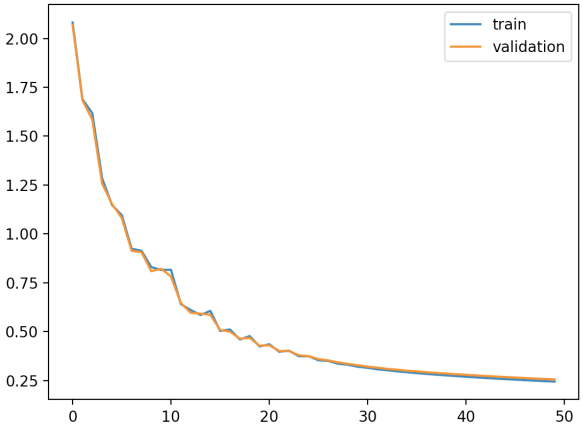

FIGURE 8

learning rate : 0.001, taille minibatch : 20

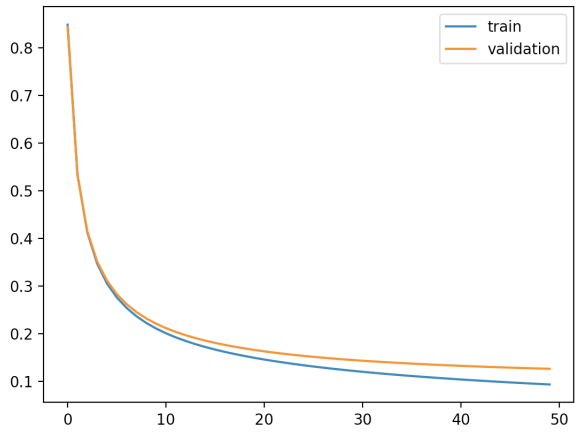

FIGURE 9

learning rate : 0.001, taille minibatch : 89

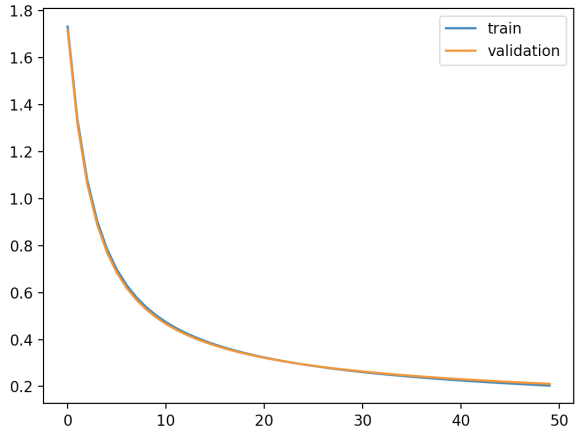

FIGURE 10

learning rate : 0.001, taille minibatch : 200

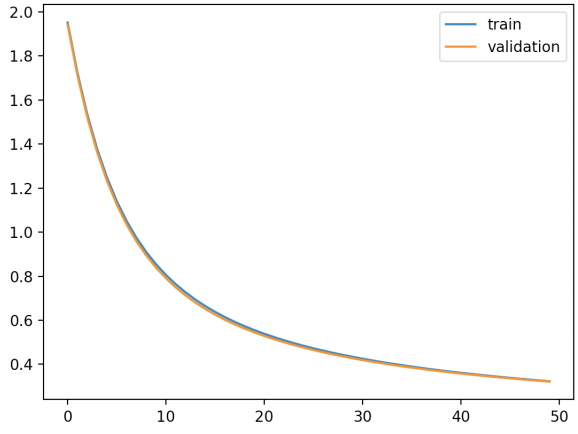

FIGURE 11

learning rate : 0.001, taille minibatch : 1000

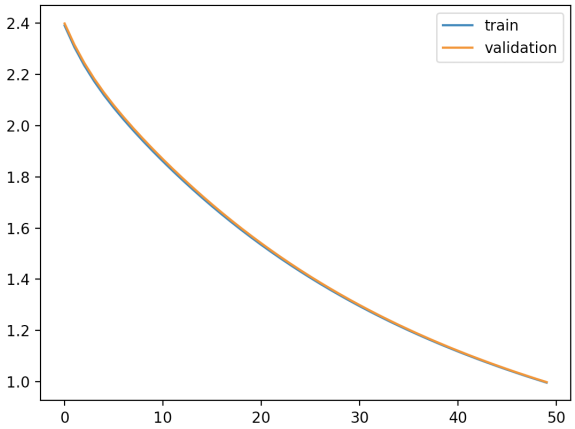

FIGURE 12

Interpretation des résultats :

Ces graphiques nous montrent bien que le choix du learning rate et de la taille des minibatchs a une grande influence sur la performance de notre programme. En effet le learning rate doit être inférieur ou égal à 0.01 pour obtenir des résultats promettants. Néanmoins si celui-ci est trop petit le réseau de neurones apprend moins vite et nécessite plus d'itérations pour atteindre un seuil de satisfaction. Concernant la taille de minibatch, elle aussi ne doit pas être trop grande afin de mettre à jour les paramètres W et b régulièrement, mais pas trop petit de manière à laisser le temps au gradient de s'accumuler pour trouver un minimum local.

b)