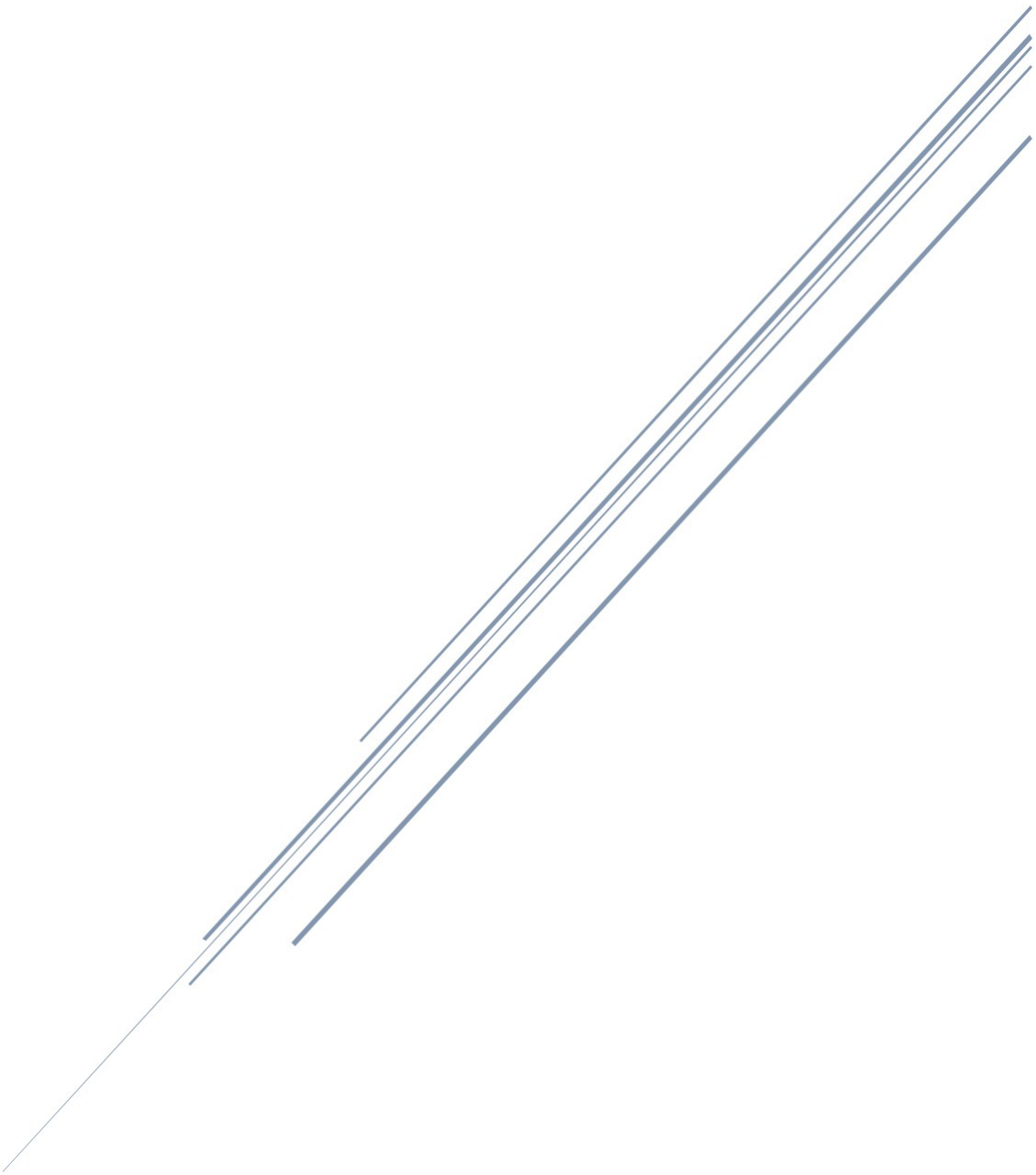
INTRODUCTION A L'APPRENTISSAGE PROFOND

Bio-Inspired Intelligence



Pierre-Henri Chupin

Partie 1 – Perceptron

Taille des tenseurs

- **Weights torch.Size([784, 10])** : En entrée, on a 784 et en sortie 10, donc w fait une taille de 784 listes de 10 éléments.
- **b torch.Size([1, 10])** : En entrée, on a 1 et en sortie 10, donc b fait une taille de 1 liste de 10 éléments.

Impact des différents hyperparamètres

- Si η = 0 alors il ne va jamais apprendre et évolué.
- Si η = 1 alors il va avoir un apprentissage trop rapide et ne pourra pas se stabiliser sur la meilleure valeur. Donc il aura un taux de bonne réponse faible.
- Si η est très petit, par exemple 0.00001, alors il va avoir de bon résultat dès le départ et pourra donc s'améliorer.
- Pour les poids :
    - Si on est entre -0.1 et 0.1
        - tensor([0.4694])
        - tensor([0.5869])
        - tensor([0.6521])
        - tensor([0.6916])
        - tensor([0.7139])
        - tensor([0.7339])
        - tensor([0.7486])
        - tensor([0.7589])
        - tensor([0.7669])
        - tensor([0.7747])
    - Si les valeurs de poids peuvent être trop dispersé, il y aura de mauvais résultat.
- Le nombre d'epochs permet d'avoir plusieurs lecture de la base de données et donc d'améliorer petit à petit le réseau de neurone.

Partie 2 – Shallow network

- Si η = 0 alors il ne va jamais apprendre et évolué.
- Si η = 1 alors il va avoir un apprentissage trop rapide et ne pourra pas se stabiliser sur la meilleure valeur. Donc il aura un taux de bonne réponse faible.
- Si η est très petit, par exemple 0.00001 alors même si le premier résultat n'est pas très bon (0.45), il va petit à petit pouvoir s'améliorer pour aller vers de meilleur résultat. Une solution qui permet d'être le plus efficace serait de mettre un poids

élevé, par exemple 0.8, puis de descendre après une itération à 0.4 puis 0.1, puis 0.001 et ainsi de suite. Comme ça il apprendrait vite au départ et on pourra ainsi peaufiner l'apprentissage par la suite.

- Si la taille de la couche caché en entrée est égale à la taille de notre première couche divisé par deux alors il arrive à apprendre correctement jusqu'à un taux de 0.8 environ.
- Si la taille de la couche caché en entrée est égale à la taille de notre réseau de neurones, alors il apprend mais très lentement. Deux epochs d'affilé, il reste sur les mêmes valeurs avant de réussir à apprendre.
- Si la taille de la couche caché en entrée est égale à 100 alors il apprend moins bien et n'arrive pas à atteindre un taux de 0.7.

Partie 3 – Deep Network

Dans cette partie, on retrouve le même fonctionnement et le même impact pour les hyper paramètre que pour ceux de la partie 2.

La taille de la première entrée de la couche cachée est égale à la taille de notre première couche divisé par 2. Puis on divise par 2 chaque entrée de la couche cachée par rapport à celle de la couche d'avant. Alors il arrive à apprendre correctement. Par exemple si on a en entrée 784 alors on aura en entrée de la première couche 392 puis en entrée de la seconde couche 196, puis 98 et ainsi de suite.

On se rend compte que ce qui a beaucoup d'importance c'est le calcul fait pour chaque couche de neurone. Par exemple si on utilise trois couches invisibles et des Sigmoids pour ces couches invisibles alors on voit que le réseau de neurones n'arrive pas à apprendre et reste à une valeur d'environ 0.11. Alors que si par exemple on utilise la combinaison Sigmoid, SELU, ReLU alors là, il s'en sort mieux sur l'apprentissage. Il commence à 0.2 puis augmente petit à petit jusqu'à 0.73 avec 20 epochs. Mais il ne fait que augmenté ce qui montre bien l'impact du choix des calculs à faire.