A.Q -> Eloise : Indications sur le Chapitre 3, partie Apprentissage.

I. Sur Tensor Flow => OK.

II. Sur la Partie Réseaux de Neurones (Section 3.3.3).

1). Donner les fonctions d'activation historiques (Sigmoide à valeurs en [0, 1], Relu différentiables, …), et leur interprétation en termes d'émulation de fonctions logiques.

2). Décrire le fonctionnement général du réseau : propagation du vecteur des impulsions d'entrée, qui représente l'objet I dont on veut apprendre le comportement, vers le vecteur O des impulsions de sorties, qui représente la caractéristique de comportement que l'on veut estimer : identificateur d'une classe d'appartenance pour un objectif de classification, propriété en {0, 1} pour un objectif de reconnaissance de forme, valeur réelle, identifiant une qualité, une performance….

3). Décrire le fonctionnement général du processus d'apprentissage :

- Pour une entrée I : Input, on calcule l'objet de sotie $A^{\lambda}(I) = O^{\lambda}$, calculée par le réseau A^{λ} (λ désigne le vecteur de coefficients synaptique et de biais à apprendre) ; Puis on évalue l'erreur $Err(O^{\lambda})$ induite par rapport à un résultat souhaité ; Puis on ajuste λ afin de diminuer cette erreur en appliquant une formule $\lambda <- \lambda - \delta.Grad(Err(O^{\lambda}))$, où δ est une petite valeur positive, qu'on nomme le pas.
- On applique ce processus pour un ensemble d'instances I (ou de paquets d'instances) dites d'entrainement, générées aléatoirement. Le processus global ainsi induit est dit de *Gradient Stochastique*. C'est parce que l'on a besoin de pouvoir calculer cette quantité gradient $Grad(Err(O^{\lambda}))$ que l'on utilise des coefficients synaptiques et des impulsions continues, ainsi que des fonctions d'activation différentiables. Il faut bien indiquer ce point.

4). Discuter des difficultés liées à la mise en œuvre des réseaux neuronaux :

- Une première difficulté tient à la conception du réseau : il n'existe à ce propos pas de méthode clairement reconnue. L'intuition est toutefois d'essayer d'épouser la logique du fonctionnement de la correspondance I : Input -> O : Output que l'on cherche à apprendre.
- Une deuxième difficulté majeure tient au fait qu'un réseau neuronal tend à mettre en jeu un nombre importan de coefficients à apprendre, ce qui signifie qu'il faut aussi un nombre important d'instances I pour les phases d'entrainement et de validation (à priori sensiblement plus que de coefficients synaptiques). Obtenir ces instances en même temps que leur résultat souhaité O peut constituer une vraie difficulté. Si le nombre d'instances est insuffisant, alors on risque de bien réussir la phase entrainement, mais d'avoir un échec sur la phase validation.

III. Sur la Partie Problématique Générale de l'Apprentissage.

1). Il faut fusionner les sections 3.3.1 et 3.3.2.

2). Il faut bien expliquer que la problématique d'apprentissage est dans la plupart des cas **une problématique d'approximation** : on est en face d'une correspondance R qui à I : Input associe O : Output complexe, que l'on veut approcher via un procédé I : Input -> A^{λ}(I) : Output, où λ est le jeu de paramètres à apprendre, et où A^{λ} est un algorithme de fonctionnement rapide : Le but est que l'erreur commise en remplaçant O par A^{λ}(I) soit faible, cette erreur pouvant être mesurée comme pourcentage de fois où O et A^{λ}(I) sont différents, dans le cas ou O désigne un identificateur de classe ou bien une propriété, ou bien une valeur moyenne Abs(A^{λ}(I)- O) dans le cas où O désigne une qualité ou un vecteur de nombres. C'est notamment le cas pour ce qui est de notre étude, puisque l'on veut remplacer un algorithme complexe de programmation dynamique par un algorithme plus simple que l'on puisse utiliser comme estimateur de solution au sein d'un processus d'optimisation.

3). Il faut lister les points durs de la mise en œuvre d'un projet de Machine Learning de la façon suivante :

- Le premier point est l'identification de la correspondance I -> O que l'on veut soumettre à apprentissage. Afin d'illustrer la difficulté, on peut se référer au jeu des échecs : on veut utiliser l'apprentissage afin de construire un logiciel qui joue bien, et donc, qui, face à une position P donnée (un état de l'échiquier) choisisse le bon mouvement M à effectuer. Que faut-il alors chercher à apprendre : la correspondance P -> Mouvement M, ou bien une correspondance auxiliaire P -> Qualité(P) qui nous fournit une estimation de la qualité du jeu du point de vue du joueur machine ? L'expérience montre que seule la deuxième approche permettra d'obtenir de bons résultats.
- Le deuxième point dur porte sur le choix du type de procédé A^{λ} dont on va se servir : ce procédé peut être une formule portant sur des indicateurs caractéristique de I ; il peut aussi être un système à base de règles (systèmes expert) ; il peut enfin être un automate relativement complexe tel qu'un réseau neuronal ou bien un automate SVM.
- Le troisième point dur porte sur l'obtention de données, c'est-à-dire de couples (I, O), où I est une instance et O un résultat théorique associé à I. Il faut que l'on dispose de nettement plus de tels couples (I, O) que de paramètres l à apprendre, et il faut veiller à ce que ces instances I soient suffisamment diversifiées (ne pas toujours apprendre le même exercice). Ceci peut s'avérer difficile et pose la question de l'apprentissage supervisé : dans le cas supervisé, on dispose de suffisamment de coule (I, O) d'entrainement, et on ajuste le vecteur de paramètres λ grâce à ces instances ; dans le cas contraire, il faut faire en sorte que le procédé A^{λ} apprenne lui-même le vecteur λ, en corrigeant de lui-même ses erreurs. On parle alors d'apprentissage non supervisé : auto-apprentissage, apprentissage par renforcement,… . Mettre en œuvre un procédé d'apprentissage non supervisé est dans tous les cas quelque chose de difficile.