Info-F-106 : Projet d'Informatique
Classification à l'aide de réseaux de neurones

Jérôme De Boeck Jacopo De Stefani Gwenaël Joret Charlotte Nachtegael

version du 8 mars 2018

Présentation générale

Le projet en trois phrases

L'objectif du projet est de réaliser un réseau de neurones en Python 3 pour effectuer de la classification de données. Le réseau est *entraîné* sur un jeu de données, par exemple des images de chiens et chats, qu'il apprend à classer correctement. Une fois entraîné celui-ci peut alors faire des *prédictions* pour de nouvelles données ; par exemple, décider si une nouvelle image est un chien ou un chat.

Les étapes de développement

La réalisation du projet est découpée en quatre parties, chacune s'etallant sur environ un mois. Voici un résumé de ce qui sera développé dans chacune de celles-ci :

— Partie 1 : implémentation d'un *perceptron*, un réseau de neurones rudimentaire effectuant de la classification binaire. Classification multiclasse en utilisant plusieurs perceptrons en parallèle.
— Partie 2 : amélioration du réseau de neurones, utilisation d'une couche interne de neurones.
— Partie 3 : réalisation d'une interface graphique à l'aide de la librairie `PyQt`.
— Partie 4 : différentes améliorations et/ou applications du réseau de neurones selon choix de l'étudiant(e), *créativité encouragée*.

Les réseaux de neurones

En informatique, les réseaux de neurones sont des systèmes de calculs inspirés des réseaux de neurones biologiques qu'on trouve dans le cerveau animal. Ces systèmes apprennent à effectuer certaines tâches progressivement sur base d'exemples. Un exemple de tâche pour laquelle un réseau de neurones peut être utile est la *classification de données*. Nous allons nous concentrer sur ce type de tâches dans ce projet. Exemples :

— classer des images de chiens et chats en 'chien' ou 'chat' ;
— classer des champignons en 'comestible' ou 'non comestible' sur base de différentes caractéristiques (couleur, taille, forme, type des spores, etc.) ;
— classer des images de chiffres entre 0 et 9 dessinés à la main selon le chiffre représenté $(0, 1, \ldots, 9)$.

Les deux premiers sont des exemples de classification *binaire*, il y a exactement deux classes. Le dernier est un exemple de classification *multiclasse* car il y a plusieurs classes (10 dans l'exemple).

Une caractéristique des réseaux de neurones qui les distinguent d'autres algorithmes étudiés en informatique est que leur fonctionnement interne est indépendant des spécificités de la tâche en question. Le même réseau peut être utilisé pour reconnaître des images de chiens ou chats, ou décider si un champignon est comestible ou non. Le réseau de neurones possède un mécanisme interne d'apprentissage (décrit plus loin) qui lui permet de s'adapter aux données fournies lors de la phase d'apprentissage pour essayer d'effectuer des prédictions les plus justes possibles par la suite. Concrètement, le code est essentiellement indépendant de l'application qui en sera fait.

Une des forces des réseaux de neurones est leur capacité à capturer des caractéristiques des données qui ne sont pas toujours faciles à formaliser précisément. Illustrons ceci avec l'exemple suivant : Le jeu de données MNIST (*Modified National Institute of Standards and Technology* dataset) est une collection

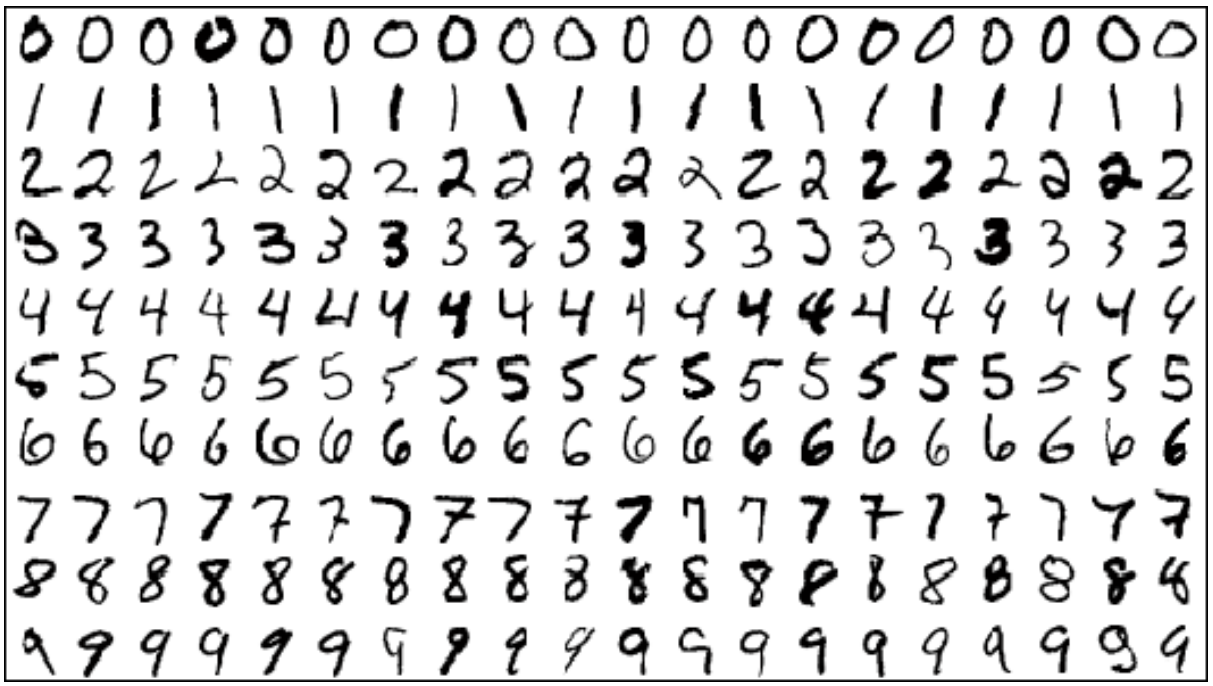


Figure 1 – Quelques images du dataset MNIST.

de 60.000 images noir et blanc, chacune de taille 28×28 représentant un chiffre entre 0 et 9 dessiné à la main. Quelques images de cet ensemble sont représentées sur la Figure 1 ; notons au passage que c'est une écriture de type américaine (voir en particulier la forme du 1). En tant qu'être humain, nous reconnaissons sans peine les différents chiffres. Cependant, il n'est pas évident de décrire formellement ce qu'est exactement un dessin d'un 0, d'un 1, etc. Imaginez un instant que vous deviez écrire un programme qui tente de reconnaître le chiffre représenté sur une nouvelle image donnée mais que vous ne connaissiez pas les réseaux de neurones ni d'autres méthodes d'apprentissage similaires (ce qui est probablement votre cas si vous lisez cet énoncé pour la première fois). Si vous n'avez que quelques heures pour écrire votre programme, probablement que votre meilleure stratégie serait de simplement ignorer l'image donnée et de choisir un nombre entre 0 et 9 au hasard. Votre taux de réussite moyen sera alors de 10%. Si vous avez plusieurs jours à disposition pour y réflechir, peut-être arriveriez-vous à améliorer ce 10% en essayant différents trucs et astuces basés sur des caractéristiques des chiffres. (Par exemple, en regardant la densité de pixels noirs dans la moitié supérieure de l'image vs la moitié inférieure, on pourrait tenter de faire la différence entre des chiffres ayant une symétrie verticale $(0, 1, 3, 8)$ et les autres, etc.) Cela prendrait néanmoins beaucoup d'efforts d'atteindre un taux de réussite de juste 50% (essayez si vous en doutez !). Par-contre, avec un réseau de neurones basique vous pouvez atteindre de biens meilleurs taux de réussite pour cette tâche au prix d'un effort minimal, comme vous le verrez vous-même dans la première partie de ce projet.

Des résultats spectaculaires ont été obtenus à l'aide de réseaux de neurones ces dernières années, comme récemment la victoire d'*AlphaGo*[1] contre le champion de Go Lee Sedol (notons tout de même que le réseau de neurones utilisé dans AlphaGo est beaucoup plus évolué que ce que nous verrons dans ce projet). Il est néanmoins bon de mentionner qu'ils ne constituent pas non plus la panacée. Ceux-ci ont en effet besoin de larges jeux de données pour apprendre (ce qui n'est pas toujours disponible), ils deviennent rapidement gourmands en ressources CPU, et sont relativement inadaptés à des tâches dont la structure mathématique est plus évidente. Par-exemple, vous pouvez apprendre à un réseau de neurones à trier des nombres mais celui-ci le fera extrêmement lentement et aura une probabilité non nulle de se tromper. (Mieux vaut utiliser un des algorithmes de tri efficaces vus au cours d'algorithmique !).

1. `https://en.wikipedia.org/wiki/AlphaGo`

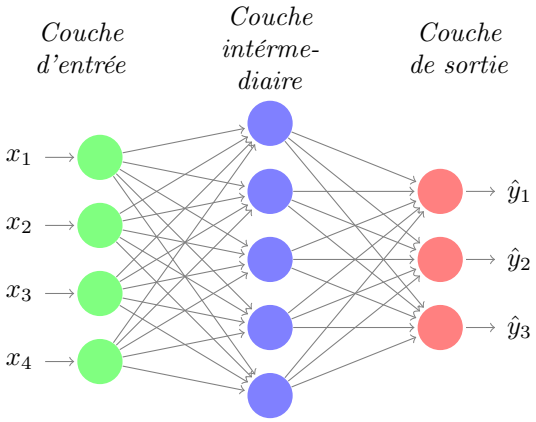


FIGURE 2 – Un réseau de neurones à trois couches.

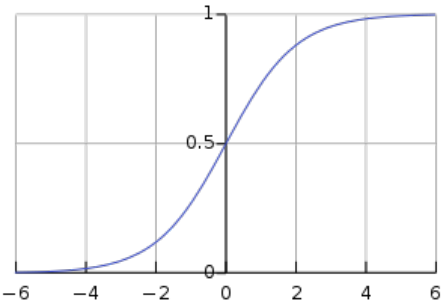


FIGURE 3 – Graphique de la sigmoïde $f(a) = 1/(1 + e^{-a})$.

Fonctionnement d'un réseau de neurones

Un réseau de neurones est composé de *neurones* et de *connexions* entre paires de neurones ; ces connexions imitent les synapses d'un cerveau animal. Les neurones sont traditionnellement arrangés hierarchiquement en *couches*, chaque neurone étant relié aux neurones de la couche précédente et de la couche suivante. Voir Figure 2 pour un exemple avec trois couches. Le nombre de couches est variable mais il y a toujours au moins une *couche d'entrée* et une *couche de sortie*.

Chaque connexion (arc) du réseau possède un *poids* qui est un nombre réel. Ces poids constituent la partie variable du réseau : ils sont itérativement mis à jour lors de la phase d'apprentissage afin d'améliorer les prédictions du réseau.

Chaque neurone est une unité de calcul : le neurone reçoit un nombre réel de chaque neurone auquel il est lié via une connexion entrante (les arcs entrants sur le dessin), il applique ensuite sa *fonction d'activation*, qui est une fonction $f : \mathbb{R} \to \mathbb{R}$ prenant en entrée le produit scalaire $\sum_{1 \leq i \leq k} z_i w_i$ du vecteur $(z_1, \ldots, z_k)$ des nombres calculés par les neurones de la couche précédente et du vecteur $(w_1, \ldots, w_k)$ des poids sur les arcs entrants du neurone. Un exemple classique de fonction d'activation est la sigmoïde

$$f(a) := \frac{1}{1 + e^{-a}}$$

qui donne un résultat entre 0 et 1, voir Figure 3 ; celle-ci sera utilisée dans la partie 2 du projet. Le résultat calculé par la fonction d'activation est alors prêt à être utilisé par les neurones de la couche suivante (les arcs sortants sur le dessin). Remarque : les neurones de la couche d'entrée sont particuliers, ils n'ont qu'un arc entrant et ne font que recopier un nombre réel donné en entrée.

L'entrée du réseau est un vecteur $(x_1, x_2, \ldots, x_n)$ de nombres réels représentant l'objet à classer. Par exemple, pour le dataset MNIST mentionné précédemment, chaque image 28×28 est représentée sous forme d'un vecteur $(x_1, x_2, \ldots, x_{784})$ ayant $28 \times 28 = 784$ entrées, où x_i est un entier entre 0 et 255 représentant le niveau de gris du $i$ème pixel de l'image (en commençant en haut à gauche).

Etant donné une entrée $(x_1, x_2, \ldots, x_n)$ du jeu de données (par exemple, une image dans le cas du MNIST), le réseau travaille couche par couche pour effectuer sa prédiction : Les neurones d'entrée ne font que recopier l'entrée. Ensuite, chaque neurone de la deuxième couche applique sa propre fonction d'activation sur base des neurones d'entrée et des poids sur les arcs. Une fois les résultats de la deuxième couche connus, les neurones de la troisième couche peuvent alors appliquer leur propre fonction d'activation sur base de ceux-ci (et des poids sur les arcs), etc. A la fin, nous obtenons un vecteur $(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_t)$ formé des nombres calculés par les neurones de la couche de sortie. Sur base de ce vecteur, une prédiction est alors calculée selon une certaine règle. Par exemple, s'il y a autant de neurones de sortie que de réponses possibles, une technique standard est de simplement choisir la classe ayant la plus grande valeur. Les neurones de sortie "votent" alors pour leur propre classe avec un certain degré de confiance.

Dans le cas d'une classification binaire il est habituel de n'avoir qu'un neurone de sortie. Si on utilisait une sigmoïde comme fonction d'activation pour ce neurone, on pourrait interpréter le résultat (un réel entre 0 et 1) comme la probabilité que l'input appartienne à la première des deux classes. Nous choisirions alors la première classe si cette probabilité est au moins $0,5$, la seconde sinon. Remarquons que cela revient en fait à regarder le signe de l'entrée a de notre sigmoïde f, puisque $f(a) \geq 0,5 \Leftrightarrow a \geq 0$. C'est ce que nous ferons dans la partie 1 du projet.

Le processus décrit ci-dessus qui, étant donné une entrée $x = (x_1, x_2, \ldots, x_n)$, effectue une prédiction sur la classe de x en utilisant le réseau est appelé *forward pass*. Dans le cas où x fait partie de notre dataset utilisé pour l'apprentissage, nous connaissons la bonne réponse pour x. Si le réseau s'est trompé dans sa prédiction, nous pouvons alors mettre à jour ses poids pour essayer de les "pousser" dans la bonne direction. C'est l'étape dite de *backpropagation*. Cette étape sera décrite plus loin pour les réseaux concrets étudiés aux parties 1 et 2.

Dans les grandes lignes, la phase d'apprentissage, aussi appelée *phase d'entraînement* (*training*), consiste à itérer un certain nombre de fois sur le dataset les processus décrits ci-dessus : forward pass et backpropagation. L'apprentissage est stoppé lorsque le réseau ne s'améliore (presque) plus. A nouveau, ceci sera expliqué en détail dans les parties 1 et 2 pour les réseaux étudiés.

Un dernier aspect qu'il convient d'aborder dans cette introduction est l'évaluation des performances de notre réseau de neurones : L'objectif final étant de donner de bonnes prédictions pour de *nouveaux* inputs, on ne peut pas juste l'évaluer sur les inputs utilisés lors de l'apprentissage (cela l'avantagerait). Dès lors, avant de commencer l'entraînement, nous mettons de côté une partie des inputs (par exemple la moitié) choisis au hasard, pour constituer un *ensemble de test* (*test set*). Le réseau de neurones est alors entraîné itérativement sur le premier ensemble, le *training set*, et est ensuite évalué sur le second.

Puisque les performances dépendent de la partition aléatoire en ces deux ensembles, la *validation croisée* (*cross validation*) du réseau consiste à recommencer ce processus un certain nombre de fois (à chaque fois de zéro), et à prendre la moyenne des performances. Cette façon standard d'évaluer un réseau de neurones sera utilisée dans les différentes parties de ce projet.

Organisation

Pour toute question portant sur ce projet, n'hésitez pas à rencontrer le titulaire du cours ou la personne de contact de la partie concernée.

Titulaire. Gwenaël Joret – gjoret@ulb.ac.be – O8.111

Assistants.

Partie 1 : Jérôme De Boeck – jdeboeck@ulb.ac.be – N3.207

Partie 2 : Jacopo De Stefani – jacopo.de.stefani@ulb.ac.be – O8.212

Partie 3 : Charlotte Nachtegael – charlotte.nachtegael@ulb.ac.be – N8.213

Partie 4 : Charlotte Nachtegael – charlotte.nachtegael@ulb.ac.be – N8.213

Consignes générales

— L'ensemble du projet est à réaliser en `Python 3`.

- Le projet est organisé autour de quatre grandes parties
- En plus de ces quatre parties à remettre, chaque étudiant doit préparer une *présentation orale* d'environ 8 minutes,, celles-ci se dérouleront durant la seconde moitiée du mois d'avril.
- Chacune des quatre parties du projet compte pour 20 points. La présentation compte également pour 20 points en tout, ce qui fait un total de 100 points.
- Chacune des quatres grandes parties devra être remise sous deux formes :
  - sur GitHub Classroom ;
  - sur papier, au secrétariat étudiants du Département d'Informatique.
- Après chacunes des trois premières parties, un correctif sera proposé. Vous serez libre de continuer la partie suivante sur base de ce correctif mais nous vous conseillons de plutôt continuer avec votre travail en tenant compte des remarques qui auront été faites.
- Les « Consignes de réalisation des projets » (cf. `http://www.ulb.ac.be/di/consignes_projets_INFO1.pdf`) sont d'application pour ce projet individuel. *(Exception : Ne tenez pas compte des consignes de soumission des fichiers, des consignes précises pour la soumission via GitHub Classroom seront données).* Vous lirez ces consignes en considérant chaque partie de ce projet d'année comme un projet à part entière. Relisez-les régulièrement !
- Si vous avez des questions relatives au projet (incompréhension sur un point de l'énoncé, organisation, *etc.*), n'hésitez pas à contacter le titulaire du cours ou la personne de contact de la partie concernée, et non votre assistant de TP.
- **Il n'y aura pas de seconde session pour ce projet !**

Veuillez noter également que le projet vaudra **zéro** sans exception si :

- le projet ne peut être exécuté correctement via les commandes décrites dans l'énoncé ;
- les noms de fonctions sont différents de ceux décrits dans cet énoncé, ou ont des paramètres différents ;
- à l'aide d'outils automatiques spécialisés, nous avons détecté un plagiat manifeste (entre les projets de plusieurs étudiants, ou avec des éléments trouvés sur Internet). Insistons sur ce dernier point car l'expérience montre que chaque année une poignée d'étudiants pensent qu'un petit copier-coller d'une fonction, suivi d'une réorganisation du code et de quelques renommages de variables passera inaperçu... Ceci sera sanctionné d'une note nulle pour toutes les personnes impliquées, sans discussion possible. Afin d'évitez ce genre de situations, veillez en particulier à ne pas partager de bouts de codes sur forums, Facebook, etc.

Soumission de fichiers

La soumission des fichiers se fait via un repository individuel par partie du projet sur la plateforme GitHub Classroom. Le lien pour s'inscrire au projet sur GitHub Classroom ainsi que des explications concernant l'utilisation de l'outil Git sont disponibles sur **les slides de présentation de Git / GitHub Classroom** sur l'UV.

Une remarque concernant les retards : Contrairement à la remise de projets pour d'autres de vos cours d'informatique, il n'est ici pas possible de remettre une de vos parties en retard. Le système de versioning offert par Git vous permet de constamment mettre à jour la version de votre code sur le serveur de GitHub Classroom. Vous êtes d'ailleurs **fortement encouragé** à le faire régulièrement lorsque vous travaillez sur une partie. Cela vous permet à vous de garder une copie de chaque version intermédiaire de votre travail, et cela nous permet à nous, en tant qu'enseignants, d'avoir une idée de la régularité de votre travail. Pour l'evaluation d'une partie, vous ne devez pas indiquer quelle est la version "finale" de votre code, nous prendrons simplement la dernière version uploadée sur le repository avant la date et heure limite (toute version uploadée après sera ignorée).

Objectifs pédagogiques

Ce projet *transdisciplinaire* permettra de solliciter vos compétences selon différents aspects.

- Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques seront mises à contribution, avec une vue à plus long terme que ce que l'on retrouve dans les divers petits projets de ces cours. L'ampleur du projet requerra une analyse plus stricte et poussée que

celle nécessaire à l'écriture d'un projet d'une page, ainsi qu'une utilisation rigoureuse des différents concepts vus aux cours.
— Des connaissances non vues aux cours seront nécessaires, et les étudiants seront invités à les étudier par eux-mêmes, aiguillés par les *tuyaux* fournis par l'équipe encadrant le cours. Il s'agit entre autres d'une connaissance de base des interfaces graphiques en `Python 3`.
— Des compétences de communication seront également nécessaires : à la fin de la partie 4, les étudiants remettront un rapport expliquant leur analyse, les difficultés rencontrées et les solutions proposées. Une utilisation correcte des outils de traitement de texte (utilisation des styles, homogénéité de la présentation, mise en page, *etc.*) sera attendue de la part des étudiants. Une orthographe correcte sera bien entendu exigée.
— En plus d'un rapport, les étudiants prépareront une présentation orale, avec *transparents* ou *slides* (produits par exemple avec LaTeX, `LibreOffice Impress`, `Microsoft PowerPoint`), ainsi qu'une démonstration du logiciel développé. À nouveau, on attendra des étudiants une capacité à présenter et à vulgariser leur projet (c'est-à-dire rendre compréhensible leur présentation pour des non informaticiens, ou en tout cas pour des étudiants ayant eu le cours de programmation mais n'ayant pas connaissance de ce projet-ci).

En résumé, on demande aux étudiants de montrer qu'ils sont capables d'appliquer des concepts vus aux cours, de découvrir par eux-mêmes des nouvelles matières, et enfin de communiquer de façon scientifique le résultat de leur travail.

Conseil concernant la rédaction du rapport

— Il n'est pas obligatoire d'utiliser LaTeX pour votre rapport et vos slides mais c'est encouragé car c'est un outil que vous devrez maîtriser par la suite (au minimum pour écrire votre mini-mémoire de bachelier et votre mémoire de master) et qui demande un certain temps d'apprentissage.

Bon travail !

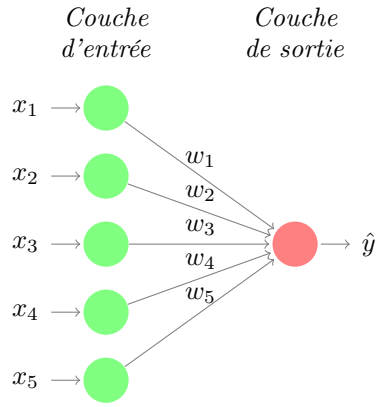
1 Partie 1 : Classification à l'aide de perceptrons

Nous utiliserons le jeu de données MNIST décrit dans l'introduction. Pour rappel, il est composé de 60.000 images noir et blanc, chacune de taille 28×28 représentant un chiffre entre 0 et 9 dessiné à la main.

L'objectif de cette partie est d'écrire un code qui, étant donné une entrée du jeu de données (une image), effectue une prédiction sur le chiffre représenté par l'image. C'est donc de la classification multiclasse. La stratégie adoptée ici est de réduire le problème à dix problèmes de classification binaire via un *perceptron*, un réseau de neurones très simple, comme suit. Tout d'abord, pour chaque chiffre $c \in \{0, 1, \ldots, 9\}$, nous verrons comment entraîner un perceptron pour qu'il puisse reconnaître le chiffre c sur une image donnée. Il devra donc distinguer les images représentant le chiffre c des autres. Ensuite, nous entraînerons dix perceptrons en parallèle, un pour chaque chiffre. Enfin, une fois la phase d'apprentissage terminée, nous les utiliserons tous ensembles pour prédire le chiffre représenté par une image donnée : Informellement, nous demanderons l'avis à chacun des perceptrons indépendamment, et nous choisirons le chiffre de celui qui est le plus "convaincu" que son chiffre est le bon. Tout ceci est expliqué en détail dans ce qui suit.

1.1 Perceptron

Commençons notre exploration des réseaux de neurones avec un perceptron, un réseau de neurones ayant juste une couche d'entrée, et une couche de sortie constituée d'un unique neurone. Il n'y a pas de couche intermédiaire. Schématiquement, un perceptron se représente comme suit :



Il y a donc un neurone d'entrée par composante x_i du vecteur $x = (x_1, x_2, \ldots, x_n)$ en entrée ($n = 5$ dans l'exemple ci-dessus), et un neurone de sortie. Le poids sur l'arc reliant le $i$ème neurone d'entrée au neurone de sortie est dénoté w_i. Notre but est d'effectuer de la classification binaire.

Remarque : Dans notre application au dataset MNIST, les deux classes correspondront à "l'image représente le chiffre c" / "l'image ne représente pas le chiffre c", pour un chiffre c donné mais la signification des classes n'est pas importante pour ce qui suit, nous les appellerons simplement première et deuxième classes.

La fonction d'activation f du neurone de sortie choisit la première classe ou la seconde en fonction du signe du produit scalaire $\sum_{i=1}^{n} w_i x_i$ donné en entrée à la fonction f :

$$f(a) := \begin{cases} 1 & \text{si } a \geq 0 \\ -1 & \text{sinon} \end{cases}$$

Nous identifions donc la première classe avec le nombre 1 et la seconde avec le nombre -1.[2] Nous noterons $\hat{y}$ la prédiction effectuée par le réseau sur l'entrée. Nous avons donc

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i\right).$$

Ceci résume le fonctionnement de la prédiction du perceptron pour l'entrée x, aussi appelée *forward pass*.

2. Remarque : il est ici plus naturel d'utiliser cette convention que $1, 2$ ou $0, 1$ à cause du fonctionnement de la mise à jour des poids, voir plus loin.

Mise à jour des poids

Supposons maintenant que nous soyons dans la phase d'entraînement du perceptron, donc x fait partie de notre *training set*, et notons $y \in \{-1, 1\}$ la bonne réponse pour l'input x. Si $\hat{y} \neq y$, nous effectuons alors une étape de *mise à jour des poids* du réseau pour qu'ils "s'approchent" un peu plus de poids qui donneraient une réponse correcte pour l'entrée x. Dans le cas de notre perceptron, cette mise à jour est un cas particulier de backpropagation [3] et consiste simplement à faire

$$w_i \leftarrow w_i + yx_i$$

pour chaque $i \in \{1, \ldots, n\}$. Illustrons le rôle de cette mise à jour sur deux exemples, afin d'obtenir de l'intuition sur ce qui se passe.

Exemple 1. Supposons $x = (1, -0.5, 3)$, $w = (1, 2, -0.5)$, et $y = 1$. Ici le produit scalaire $\sum_{i=1}^{n} w_i x_i$ vaut -1.5, qui est négatif, et donc $\hat{y} = -1 \neq 1 = y$. Puisque ici $y = 1$, la mise à jour des poids consiste à ajouter le vecteur x au vecteur w, c-à-d $w \leftarrow w + x$. Le nouveau vecteur de poids w vaut donc $w = (2, 1.5, 2.5)$. Remarquons que si nous refaisions une forward pass sur x avec ces nouveaux poids, nous obtiendrions la bonne réponse, puisque le produit scalaire vaut maintenant 8.75, qui n'est pas négatif.

Exemple 2. Supposons $x = (-1, -1, 1)$, $w = (0.5, 1, 7)$, et $y = -1$. Ici le produit scalaire $\sum_{i=1}^{n} w_i x_i$ vaut 5.5, donc $\hat{y} = 1 \neq -1 = y$. Puisque $y = -1$, la mise à jour des poids consiste à retirer le vecteur x au vecteur w, c-à-d $w \leftarrow w - x$. Le nouveau vecteur de poids w vaut donc $w = (1.5, 2, 6)$. Avec ces nouveaux poids, le produit scalaire $\sum_{i=1}^{n} w_i x_i$ vaut maintenant 2.5. Le réseau donnera donc toujours une mauvaise prédiction pour x mais remarquons que le produit scalaire a bougé dans la bonne direction, il est passé de 5.5 à 2.5, et est donc plus proche de devenir négatif.

L'intuition à retenir est donc qu'en cas de mauvaise prédiction sur x, la mise à jour des poids modifie le vecteur de poids w de façon à ce que le produit scalaire $w \cdot x$ bouge dans la bonne direction : celui-ci augmente si $y = 1$ et diminue si $y = -1$.

Entraînement

L'idée générale de la phase d'entraînement est de continuellement passer en revue tous les inputs x de notre training set, à chaque fois de calculer la prédiction $\hat{y}$ du perceptron pour x (forward pass), et d'effectuer une mise à jour des poids si celle-ci diffère de la bonne réponse y. Cette boucle infinie est arrêtée lorsque le réseau ne s'améliore plus pendant un certain temps.

Plus précisément, voici les règles qui seront utilisées lors de l'entraînement :

- Le vecteur de poids w initial est le vecteur nul $(0, 0, \ldots, 0)$. [4]
- A chaque passage de la boucle principale, on effectue une *étape d'entraînement*.
- Une étape d'entraînement consiste à passer en revue chacun des inputs x du training set une fois, et à effectuer directement une mise à jour des poids si la prédiction pour x n'est pas correcte.
- L'ordre dans lequel les inputs x du training set sont considérés lors d'une étape d'entraînement est *aléatoire* ; en d'autres mots, le training set est mélangé au début de chaque étape d'entraînement.
- A la fin d'une étape d'entraînement, on évalue la situation actuelle comme suit : On repasse en revue chacun des inputs x du training set et on calcule le score courant, défini comme le nombre de prédictions correctes. Ici on n'effectue pas de mise à jour des poids lors d'une prédiction incorrecte.
- On garde en mémoire le meilleur score réalisé et le vecteur poids correspondant.
- La boucle principale est arrêtée si le meilleur score n'a pas été amélioré lors des `max_wait` dernières étapes d'entraînements, ou si un nombre maximum `max_it` d'itérations est atteint, où `max_wait` et `max_it` sont deux paramètres (ex : 4 et 100).

3. Remarque de terminologie : dans la littérature, le terme *backpropagation* fait en général réference au mécanisme de mise à jour des poids pour des réseaux ayant au moins une couche interne de neurones. Dans le cas d'un perceptron la règle de mise à jour est très simple et est traditionnellement appelée "règle du Delta". Techniquement, cette règle est bien un cas particulier de backpropagation sauf qu'aucune "propagation" n'a lieu car le perceptron n'a pas de couche interne. Le mécanisme de backpropagation pour des réseaux avec couches internes sera décrit dans la partie 2 du projet.

4. La valeur initiale de w n'est pas importante pour un perceptron mais pourra avoir un impact pour d'autres réseaux de neurones plus sophistiqués.

— Une fois l'entraînement terminé, on garde la fonction poids correspondant au meilleur score réalisé (qui n'est pas nécessairement la dernière).

Un conseil : Avant de commencer à programmer, écrivez la phase d'entraînement sous forme d'un pseudo-code précis reprenant chacune des règles ci-dessus.

1.2 Classification à l'aide de plusieurs perceptrons

Revenons maintenant à notre application, le dataset MNIST. La stratégie adoptée est, pour chaque chiffre $c \in \{0, 1, \ldots, 9\}$, de créer un perceptron et de l'entraîner à reconnaître le chiffre c (comme vu ci-dessus). Notons w^c le vecteur de poids résultant. Une fois les dix entraînements terminés, nous avons à notre disposition les dix vecteurs de poids $w^0, w^1, \ldots, w^9$. Etant donnée une entrée x, nous allons effectuer une prédiction sur le chiffre représenté par x comme suit : Pour chaque chiffre $c \in \{0, 1, \ldots, 9\}$ on calcule d'abord le produit scalaire $w^c \cdot x$. Ensuite, on choisit simplement le chiffre c tel que le produit scalaire $w^c \cdot x$ est maximum. Nous appellerons ceci la *prédiction globale*, ou simplement la *prédiction* de notre code pour l'input x (à ne pas confondre avec la forward pass d'un perceptron individuel).

L'intuition ici est que ces produits scalaires représentent des degrés de confiance : Si $w^c \cdot x \geq 0$, le perceptron correspondant pense que l'image représente le chiffre c, et plus $w^c \cdot x$ est élevé plus il en est convaincu. C'est pour cela que s'il y a plusieurs chiffres c tels que $w^c \cdot x \geq 0$ (c-à-d plusieurs perceptrons qui pensent reconnaître leur chiffre), on choisira celui qui en est le plus confiant. De même, il se pourrait qu'il n'y ait aucun chiffre c tel que $w^c \cdot x \geq 0$; dans ce cas notre règle choisit le chiffre c tel que $w^c \cdot x$ est le plus proche de 0, c-à-d le chiffre c du perceptron qui est le moins convaincu de ne pas reconnaître c.

Validation croisée

Comme mentionné dans l'introduction, la performance de notre code (c-à-d la justesse des prédictions) sera évaluée selon une validation croisée. Celle-ci se fait comme suit :

— La boucle principale est effectuée `it_CV` fois, où `it_CV` est un paramètre.
— Au début de chaque itération, on choisit aléatoirement une fraction `alpha` (ex : 50%) de notre dataset pour constituer un training set et le reste un test set, où `alpha` est un paramètre.
— Ensuite, on crée nos dix perceptrons et on effectue une phase d'entraînement de chaque perceptron sur le training set.
— Enfin, on évalue les prédictions effectuées en utilisant les dix perceptrons ensembles sur le test set, on compte donc le nombre de prédictions correctes sur le test set (le score).
— Une fois la boucle principale terminée, on calcule le score moyen réalisé, et le pourcentage moyen de prédictions correctes sur le test set. Ce pourcentage est la quantité utilisée pour évaluer les performances du code.

A nouveau, avant de commencer à programmer nous vous conseillons d'écrire la validation croisée d'abord sous forme de pseudo-code.

Lecture de données

Le jeu de données MNIST est disponible sur l'UV, voir fichier `train.csv`. Chaque ligne du fichier représente une entrée, c-à-d une image. Un exemple de ligne est donné sur la Figure 4. Le premier nombre sur la ligne est le chiffre représenté par l'image (ici 6). Les 784 nombres suivants correspondent aux pixels de l'image (de taille 28×28), en commençant en haut à gauche. Pour chaque pixel un niveau de gris entre 0 (blanc) et 255 (noir) est indiqué.

Vu sa taille (76 MB), le fichier `train.csv` est assez lourd à utiliser. En effet, le code que vous devrez réaliser prend une dizaine de minutes à s'exécuter sur un ordinateur récent, ce qui n'est pas pratique pour tester son code. Pour cette raison, deux autres fichiers `train_small.csv` et `train_tiny.csv` sont également disponibles sur l'UV. Le premier est un sous-ensemble de 2.000 images des 60.000 du dataset tirées au hasard, le second un sous-ensemble de 100 images tirées au hasard.

Les performances (c-à-d taux de prédictions correctes) de nos réseaux de neurones seront assez proches sur `train_small.csv` et sur `train.csv`. Elles seront légèrement meilleures sur `train.csv` car le réseau aura alors un plus grand ensemble d'images pour s'entraîner mais le temps d'exécution sera plus court sur `train_small.csv`. Pour cette raison, nous vous demandons d'utiliser `train_small.csv` dans le code que vous soumettrez pour la partie 1 ; libre à vous de tester votre code sur `train.csv` de votre

```
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,2,99,192,93,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,57,254,254,216,24,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,127,254,254,179,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,42,224,254,254,107,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,174,254,254,197,22,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,47,225,254,245,53,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,25,225,254,254,171,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,38,254,254,216,22,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,66,254,254,198,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,162,254,254,151,63,63,45,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,24,232,254,254,254,254,254,236,153,12,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,129,254,254,254,254,254,254,254,254,166,11,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,156,254,254,254,237,173,79,110,254,254,137,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,156,254,254,237,130,0,0,13,171,254,193,11,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,156,254,254,98,0,0,0,0,82,254,254,30,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,46,254,254,80,0,0,0,0,82,254,254,140,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,19,216,255,132,0,0,0,32,214,254,215,30,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,108,254,251,209,113,113,215,254,254,161,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,21,214,254,254,254,254,254,254,254,104,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,4,91,218,255,255,255,153,26,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

FIGURE 4 – Une ligne du jeu de données MNIST. N.B. des passages à la ligne ont été ajoutés pour pouvoir afficher l'entièreté de la ligne et faire ressortir l'image, tous les nombres sont sur la même ligne dans le fichier.

côté bien sûr. Lorsque vous programmez, nous vous conseillons d'utiliser au début le tout petit ensemble `train_tiny.csv` par soucis de rapidité d'exécution, et de passer au jeu de données intermédiaire seulement lorsque votre code est complètement fonctionnel. (Observez au passage la dégradation des performances du réseau lorsque le jeu de données pour l'apprentissage est trop petit !)

Le repository de la partie 1 contient un fichier `readdata.py` avec la fonction suivante :

`readData(filename) : return L`
: Fonction qui lit le jeu de données contenu dans le fichier `filename` et renvoie une liste `L` des entrées du jeu de données. Chaque entrée est représentée sous forme d'une liste d'entiers (ex : `[6,0,0,0,0,0,0,0,0,0,...,0]` pour l'entrée de la Figure 4.)

Vous pouvez utiliser cette fonction pour lire le jeu données (que vous aurez au préalable mis dans le même dossier que votre programme).

1.3 Fonctions

Nous vous demandons d'implémenter les fonctions suivantes. Veillez à respecter scrupuleusement la signature des fonctions, c-à-d le nom (attention miniscules vs majuscules !) et l'ordre des paramètres. Ceci est indispensable pour les tests automatiques qui seront effectués lors de la correction. Un code ne respectant pas une des signatures sera sanctionné d'une note nulle (même si c'est dû juste à une faute de frappe), soyez donc extrêmement vigilant sur ce point, ce serait dommage de perdre des points aussi bêtement !

`forwardPass(x, w) : return s`
: Fonction qui effectue une forward pass d'un perceptron sur l'input (image) x avec le vecteur de poids w comme décrit ci-dessus. La valeur retournée est la prédiction du perceptron (1 pour la première classe, -1 pour la seconde).

`train(data, digit, max_wait = 3, max_it = 40) : return w`
Fonction qui effectue l'entraînement d'un perceptron sur le training set `data` pour reconnaître le chiffre `digit`. La première classe (1) correspond aux images représentant le chiffre `digit`, la seconde (-1) aux autres. La fonction renvoie le meilleur vecteur poids `w` trouvé, selon la procédure d'entraînement décrite ci-dessus.

`predict(x, L) : return digit`
Fonction qui effectue la prédiction (globale) du chiffre représenté par l'input (image) x sur base de la liste `L` des vecteurs poids des dix perceptrons, comme décrit ci-dessus. La fonction renvoie le chiffre `digit` prédit.

`crossValidation(data, alpha = 0.5, it_CV = 4, max_wait = 3, max_it = 40) : return None`
Fonction qui effectue la validation croisée sur le dataset `data` comme décrite ci-dessus, où `max_wait` et `max_it` sont les paramètres qui seront utilisés lors des appels à la fonction `train`. La fonction affichera à l'écran les paramètres utilisés pour la validation croisée et la moyenne des performances. Exemple :

```
alpha = 0.5, it_CV = 4, max_wait = 3, max_it = 40
Taux de réussite moyen : 72.18%.
```

Vous êtes bien sûr libre d'implémenter des fonctions supplémentaires ; suivez les bonnes pratiques vues au cours de Programmation. Pour effectuer les choix aléatoires, vous aurez besoin du module `random`, que vous pouvez donc utiliser (veillez juste à ne pas utiliser la fonction `seed`, celle-ci sera en effet utilisée dans les tests automatiques pour fixer les choix aléatoires). Pour cette partie 1, vous ne pouvez utiliser d'autres librairies. Notons que `numpy`, la librairie standard de calcul scientifique en Python, sera utilisée plus tard dans le projet mais celle-ci n'est pas nécessaire pour cette partie-ci vu la simplicité des calculs.

1.4 Consignes de remise

L'ensemble de votre code doit être contenu dans un fichier `partie1.py` (avec une minuscule !), qui sera soumis via votre repository sur GitHub Classroom. Veillez à ne pas ajouter les training sets à votre repository ! (Imaginez l'espace utilisé si 200 étudiants ajoutaient chacun un fichier de 76 MB à leur repository ...). Votre programme fera un appel à la fonction `crossValidation` avec les valeurs par défaut pour `alpha`, `it_CV` , `max_wait` et `max_it` spécifiées dans les signatures de fonctions.

Remise : Deadline sur GitHub Classroom : dimanche 12 novembre 2017 à 22 heures. Conseil : **uploadez régulièrement votre code** sur le serveur ! Version papier à remettre au plus tard le lundi 13 novembre 2017 à 14h au secrétariat.

Personne de contact : Jérôme De Boeck – jdeboeck@ulb.ac.be – N3.207

2 Partie 2 : Classification non-linéaire avec un réseau multi couche

L'objectif de cette deuxième partie reste le même, identifier le chiffre représenté par une image appartenant jeu de données MNIST (42000 images noir et blanc de taille 28×28 représentant un chiffre entre 0 et 9 dessiné à la main). La stratégie générale reste la même que pour la partie 1 : nous construirons en parallèle 10 modèles différents, un pour chaque chiffre, chaque modèle étant en charge distinguer les images représentant son chiffre c des autres. Les modèles seront ensuite utilisés ensembles pour fournir une prédiction sur le chiffre représenté sur une image donnée.

La différence principale avec la partie 1 est l'ajout d'une couche intermédiaire de neurones à chaque perceptron et l'utilisation d'une fonction d'activation non linéaire (une sigmoïde). Ces deux modifications permettent d'améliorer le pouvoir prédictif du modèle mais nécessitent une procédure spécifique de backpropagation afin d'effectuer la mise à jour des poids.

A partir de partie-ci et tout au long du projet, vous serez amenés à apprendre certains outils par vous même. Pour cette partie-ci, nous vous demandons d'explorer et d'utiliser la librairie de calcul scientifique `numpy`. Celle-ci facilitera la gestion des matrices et la génération aléatoire de nombres selon des distributions non uniformes. Notez en particulier que certaines opérations décrites dans les pseudo-codes ci-dessous peuvent se réaliser de manière très compacte en utilisant le bon produit matriciel.

2.1 Perceptron multicouche

Comme le nom le suggère, un perceptron multicouche est un modèle constitué de plusieurs couches de neurones. Dans cette partie-ci, nous considérons un réseau avec trois couches : couche d'entrée, couche intermédiaire et couche de sortie. (Notons au passage que les réseaux ayant de nombreuses couches intermédaires (*deep networks* / *deep learning*) ne seront pas considérés dans le cadre de ce projet afin de limiter la complexité de votre travail). L'architecture que vous devrez implémenter aura donc la forme suivante :

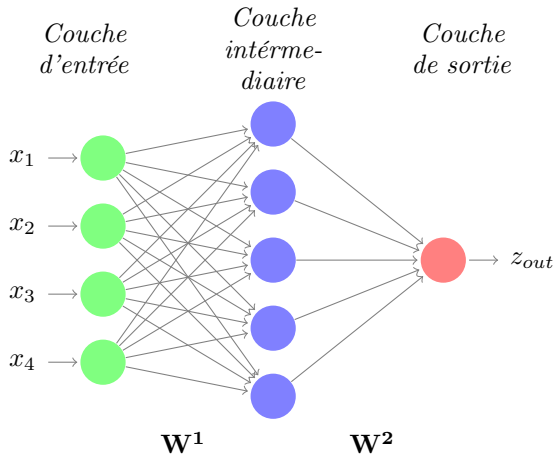


FIGURE 5 – Exemple d'architecture d'un perceptron multicouche avec 4 neurones dans la couche d'entrée et 5 neurones dans la couche intermédiaire.

Comme pour la partie 1, dans la couche d'entrée nous avons un neurone d'entrée par composante x_i du vecteur $x = (x_1, x_2, \ldots, x_n)$ en entrée ($n = 4$ dans l'exemple ci-dessus). La couche d'entrée est connectée avec une couche intermédiaire. Le nombre H de neurones de cette couche intermédiaire est un des hyperparamètres du modèle. Les poids des connexions entre ces deux couches sont encodés sous forme d'une matrice $\mathbf{W^1}$ ayant H lignes et n colonnes. Les poids des connexions entre la couche intermédiaire et l'unique neurone de sortie sont encodés dans la matrice $\mathbf{W^2}$ ayant H lignes et une colonne.

La fonction d'activation des neurones des couches intermédiaires et de sortie est une fonction non linéaire, nommé sigmoïde, dont la formulation analytique est la suivante (voir Figure 3 dans l'introduc-

tion) :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

La sigmoïde est souvent utilisée comme fonction d'activation à cause de certaines propriétés mathématiques avantageuses, comme par exemple :

$$\frac{\partial}{\partial x}\sigma(x) = (1 - \sigma(x))\sigma(x) \tag{2}$$

et

$$0 < \sigma(x) < 1 \qquad \forall x \in \mathbb{R} \tag{3}$$

La propriété 2 permet de rendre plus efficace le calcul de la dérivée première de la fonction d'activation, opération récurrente dans la procédure de backpropagation. La propriété 3 permet d'interpreter la sortie d'un neurone comme une probabilité. En particulier, la sortie z_{out} du neurone de la couche de sortie sera interpretée comme la probabilité (selon le réseau) que l'input x appartienne à la première des deux classes. Celle-ci est calculée comme suit :

$$z_{out} = \sigma\left(\sum_{i=1}^{H} \mathbf{W_i^2} \cdot \sigma\left(\sum_{j=1}^{n} \mathbf{W_{ij}^1} x_j\right)\right) \tag{4}$$

Notre but est d'effectuer de la classification binaire où les deux classes correspondront à "l'image représente le chiffre c" / "l'image ne représente pas le chiffre c", pour un chiffre c donné mais la signification des classes n'est pas importante pour ce qui suit, nous les appellerons simplement première et deuxième classes, représentées respectivement par les nombres 1 et -1 comme dans la partie 1. Donc, pour passer de la probabilité décrite par le score z_{out} au choix de la classe, nous procéderons comme suit :

$$\hat{y} = \begin{cases} 1 & \text{si } z_{out} \geq 0.5 \\ -1 & \text{sinon} \end{cases} \tag{5}$$

Ceci résume le fonctionnement de la prédiction (forward pass) d'un perceptron.

Lecture de données

Nous continuerons à utiliser le jeu de données MNIST employé lors de la partie 1, dans le trois formats (en ordre croissant de taille) `train_tiny.csv`, `train_small.csv` et `train.csv`. Pensez à adapter la fonction `readData` introduite lors la partie 1 selon l'usage que vous ferez de `numpy`.

2.2 Entraînement

Comme pour la partie 1, l'objectif de l'algorithme d'entraînement du réseau est de trouver des poids qui minimisent l'erreur de classification lors de l'apprentissage. L'algorithme utilisé pour l'entraînement est décrit à l'Algorithme 1. La mise à jour des matrices poids se fait de la manière suivante :

$$\mathbf{W^1} \leftarrow \mathbf{W^1} - \eta \cdot \Delta\mathbf{W^1} \tag{6}$$

et

$$\mathbf{W^2} \leftarrow \mathbf{W^2} - \eta \cdot \Delta\mathbf{W^2} \tag{7}$$

Les matrices $\Delta\mathbf{W^1}$ et $\Delta\mathbf{W^2}$ sont calculées lors de la backpropagation. Le paramètre η est appelé *learning rate*, celui-ci représente l'importance que nous souhaitons accorder à une correction des poids lors d'une backpropagation. Les algorithmes de forward pass et de backpropagation sont détaillés aux Algorithmes 2 et 3, respectivement. Rappelons que vous êtes libres d'améliorer la rapidité d'exécution de votre code à l'aide des fonctionnalités offertes par la libraire `numpy`.

Nous allons définir des quantités intermédiaires, s_i et z_i, respectivement pour l'entrée et la sortie d'un neurone intermédiaire, et s_{out} et z_{out}, respectivement pour l'entrée et la sortie du neurone de sortie :

$$s_i = \sum_j \mathbf{W_{ij}^1} x_j \tag{8}$$

$$z_i = \sigma(s_i) \tag{9}$$

$$s_{out} = \sum_i \mathbf{W_i^2} z_i \tag{10}$$

$$z_{out} = \sigma(s_{out}) \tag{11}$$

La définition de ces quantités nous permet d'écrire de manière élégante les formules utilisées pour les calculs des gradients :

$$grad_{out} = \frac{\partial z_{out}}{\partial s_{out}} = \sigma(s_{out})(1 - \sigma(s_{out})) \tag{12}$$

$$grad_i = \frac{\partial z_i}{\partial s_i} = \sigma(s_i)(1 - \sigma(s_i)) \tag{13}$$

Enfin, le mécanisme de "dropout" est décrit plus loin.

Algorithme 1 Pseudocode algorithme entraînement du réseau

```
Initialisation des poids
while !(Critère arrêt) do
  for chaque ligne du dataset do
    Dropout
    Forward pass
    Backpropagation
    Mise à jour des poids
  end for
end while
```

Algorithme 2 Pseudocode algorithme forward pass

```
for chaque neurone i couche intermédiaire do
  Calcul activation z_i
  Calcul gradient grad_i
end for
Calcul activation z_out
Calcul gradient grad_out
Détermination prédiction ŷ
```

Algorithme 3 Pseudocode algorithme backpropagation

```
Calcul erreur sortie e_out = ŷ − y
Calcul delta sortie δ_out = e_out · grad_out
for chaque neurone i couche intermédiaire do
  Calcul delta intermédiaire δ_i = δ_out · W_i^2 · grad_i
  for chaque neurone j couche input do
    Calcul mise à jour ΔW_ij^1 = δ_i · x_j
  end for
  Calcul mise à jour ΔW_i^2 = δ_out · z_i
end for
```

Initialisation des poids Etant donné la fonction d'activation utilisée, une initialisation des poids avec des valeurs nulles comme à la partie 1 empêche en pratique, pour des raisons d'approximation numérique, la convergence vers des valeurs optimales des poids (cf. [**?**]). La solution standard pour éviter ces problèmes numériques est d'initialiser les poids du réseau avec des valeurs tirées au hasard selon une distribution normale avec moyenne nulle et petit écart type. C'est ce que vous devrez faire pour cette partie-ci, en choisissant 0.0001 comme valeur pour l'écart type. [5]

Condition d'arrêt Nous utiliserons les deux critères de convergence suivant :

$$\|\Delta \mathbf{W^1}\| < \varepsilon \tag{14}$$

et

$$\|\Delta \mathbf{W^2}\| < \varepsilon \tag{15}$$

où

$$\|\mathbf{A}\| = \sum_i \sum_j |A_{ij}| \tag{16}$$

La condition d'arrêt de l'entraînement est basée sur deux sous conditions qui vérifient respectivement si les poids de la couche d'entrée (Eq. 14) et de la couche intermédiaire (Eq. 15) ont convergé à moins d'une différence ε près (un paramètre). Pour l'arrêt de la boucle, les deux conditions doivent être vérifiées en même temps. Remarques : (1) Notez que cette condition d'arrêt doit être testée après chaque mise à jour des poids. (2) Au cas où aucune backpropagation n'a lieu lors d'une étape d'entraînement (c-à-d que le perceptron a appris parfaitement son *training set*), il conviendra également d'arrêter l'entraînement.

Afin de limiter le temps total d'exécution de votre algorithme, nous vous demandons d'ajouter, à coté de ce critère de convergence, un nombre maximal d'itérations de la boucle principale, après lequel votre programme arrêtera l'apprentissage. Une fois l'entraînement arrêté, nous choisirons la *meilleure paire* (W_1, W_2) trouvée, qui n'est donc pas nécessairement la dernière, exactement comme à la partie 1.

Dropout Un problème commun à différents algorithmes de machine learning est le problème du surapprentissage (*overfitting*). Ce problème apparaît lorsque la méthode d'apprentissage automatique parvient à se spécialiser sur la structure de l'ensemble d'apprentissage sans être en même temps capable de bien généraliser ce qu'elle a appris vers différents jeux de données.

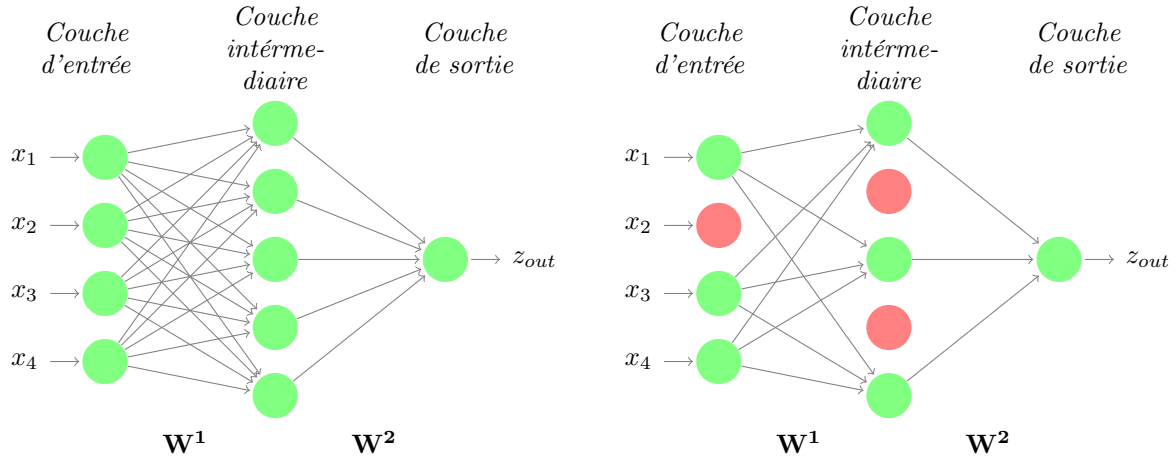
Nous vous proposons d'implémenter une technique nommée *dropout*, spécifiquement développée pour contrer les problèmes d'overfitting des réseaux de neurones. L'idée du dropout [6] est la suivante : au début de chaque itération de la phase d'entraînement, nous passons en revue chacun des neurones de la couche d'entrée et de la couche intermédiaire, et pour chaque neurone de le retirer temporairement du réseau (avec toutes ses connexions entrantes et sortantes) avec une certaine probabilité $p \in [0, 1]$, de manière indépendante. [7] Notons que cela ne s'applique pas au neurone de sortie, que nous devons garder pour produire le score. Une illustration est donnée sur la figure 8. Les neurones ainsi choisis sont désactivés durant toute l'itération, ils sont donc ignorés durant les forward passes *et* les backpropagations. Intuitivement, le réseau est ainsi forcé d'apprendre "de plusieurs manières différentes" à classer correctement les données. Cela permet d'obtenir un réseau qui est plus robuste par rapport à la présence de bruit dans les données et ayant une meilleure capacité de généralisation.

Remarque : Afin de considerer la procedure de dropout comme valide, au moins un neurone dans chaque couche doit rester activé. Au cas où tous les neurones d'une couche sont désactivés, la procédure de dropout doit être recommencée. Nous vous encourageons à tester plusieurs valeurs de p de manière à pouvoir déterminer la valeur idéale pour le jeu de données MNIST.

5. Vous êtes encouragés à tester vous même l'importance de bien choisir ce paramètre : qu'observez-vous avec un écart type trop grand ? Et à l'opposé avec un écart type nul ?

6. To drop out [EN] → retirer [FR]

7. Dans une version plus avancée du dropout, nous pourrions choisir une valeur de p pour la couche d'entrée et une autre pour la couche intermédiaire, ici nous utiliserons la même probabilité p pour chacune des deux couches.



(a) Réseau avant dropout

(b) Réseau après dropout

FIGURE 6 – Exemple d'application du dropout. Les neurones rouges ont été désactivés par effet du dropout.

2.3 Classification à l'aide de plusieurs perceptrons multicouches

De la même manière que pour la partie 1, nous entraînerons un perceptron multicouche pour chaque chiffre $c \in \{0, 1, \ldots, 9\}$, chaque modèle ayant pour but de distinguer son chiffre c des autres. Vu la nature du perceptron multicouche, nous aurons deux matrices de poids $\mathbf{W^{1,c}}$ et $\mathbf{W^{2,c}}$ pour chaque modèle, pour un total de 20 matrices une fois les dix entraînements terminés.

Pour la classification d'une image x, nous allons procéder de la manière suivante :

1. Pour chaque chiffre $c \in \{0, 1, \ldots, 9\}$ nous effectuons d'abord une forward pass [8] avec les matrices $\mathbf{W^{1,c}}$ et $\mathbf{W^{2,c}}$, ce qui nous permet d'obtenir le score $z_{out,c}$ du perceptron c considéré, qui représente son degré de confiance concernant la présence du chiffre c sur l'image.
2. Ensuite la *prédiction globale* $\hat{y}$ est calculée comme suit :

$$\hat{y} = \arg\max_c z_{out,c} \tag{17}$$

Donc exactement comme dans la partie 1, nous choisissons le chiffre c du percepetron affichant le plus haut degré de confiance.

2.4 Validation croisée

La validation croisée devra être effectuée exactement comme lors de la partie 1.

2.5 Affichage matrice de confusion

Afin de mieux interpréter les résultats de votre classificateur, nous vous demandons de construire sa *matrice de confusion* C. C'est une matrice 10×10 telle chaque entrée C_{ij} représente le nombre d'entrées du jeu de données pour lesquels la classe prédite est j alors que la classe réelle est i. Donc, dans le cas d'un classificateur parfait, toutes les entrées hors diagonale sont nulles. En pratique, nous souhaitons minimiser les entrées hors diagonale.

Dans notre cas, chaque ligne i de la matrice représente le résultat de la classification des images représentant le $i$ème chiffre. Puisque le nombre d'images pour chaque chiffre dans le *test set* ne sont pas exactement les mêmes d'un chiffre à l'autre, nous devons normaliser les lignes de la matrice de confusion comme suit pour pouvoir les comparer entres elles : Redéfinissons C_{ij} comme la *proportion* du nombre d'entrées dont la classe réelle est i et pour lesquels la classe prédite est j.

8. Notons que nous n'appliquons plus le mécanisme de dropout ici puisque la phase d'entraînement est terminée

Code Couleur - Hors diagonale	Code Couleur - Diagonale
227	227
221	191
215	115
209	119
203	83

TABLE 1 – Tableau des codes couleurs ANSI pour l'affichage de la matrice de confusion

Pour visualiser ces valeurs entre 0 et 1, nous vous proposons d'utiliser deux échelles, une pour les éléments sur la diagonale, et une pour les éléments hors diagonale (5 couleurs, chacune) que le terminal permet d'afficher via les codes ANSI (cf. Section 2.5).

Nous utiliserons les gradients suivants pour les couleurs :

— **Hors-diagonale** : Jaune → Rouge

— **Diagonale** : Jaune → Vert

Le codes couleurs ANSI correspondants sont repris dans la table 1.

Vous êtes libres de choisir vous-même les découpes de l'intervalle $[0, 1]$ en 5 sections pour les couleurs hors-diagonales et pour celles sur la diagonale. Notez que la découpe uniforme $[0.0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1]$ ne sera pas très informative vu les très bonnes performences de notre classificateur sur notre jeu de données ; réflechissez donc à une découpe plus informative, et précisez-là sur une légende.

L'affichage de la matrice de confusion devra se faire à la fin de chaque itération de la validation croisée.

Rafraichissement de l'écran Pour rendre la visualisation, vous devrez d'abord rafraichir l'écran lors de l'affichage de la matrice de confusion. Pour ce faire, vous pouvez utiliser le module `os` pour faire appel à une commande du système d'exploitation (avec la fonction `system`) ou pour détecter le système d'exploitation courant (avec la fonction `name`).

Affichage en couleur Une fonction `print_color(string,color,sep)` devra être implémentée pour afficher le texte `string` dans la couleur `color` et le terminateur `sep`. Ce dernier paramètre correspondra à la valeur du paramètre `end` de la fonction `print` (qui vaut `'\n'` par défaut). Par exemple :

```
print_color("Hello World!","38;5;232"," ")
print_color("Goodbye World!","38;5;251","")
↓
Hello World! Goodbye World!

print_color("Hello World!","38;5;232","\n")
print_color("Goodbye World!","38;5;251","")
↓
Hello World!
Goodbye World!
```

L'affichage d'une chaîne de caractères dans une couleur ayant un code couleur c demande une syntaxe particulière. Pour afficher `Hello World!` dans la couleur 232 (noir) vous devez exécuter :

`print("\x1B[ 38;5;232mHello World!\x1B[0m")` ⟶ `Hello World!`

Vos chaînes de caractères doivent donc être précédées de `"\x1B[`c`m"` et suivies de `"\x1B[0m"` où c est le code de la couleur. [9]

Il est également possible de changer la couleur du fond du texte avec une syntaxe similaire (notez le changement 38 → 48 dans le code ANSI :

`print("\x1B[ 48;5;244mHello World!\x1B[0m")` ⟶ `Hello world!`

9. Un tableau décrivant la signification des différents codes est disponible au lien : `https://en.wikipedia.org/wiki/ANSI_escape_code#Colors`

Remarque : certaines versions de Windows n'interpretent pas correctement les balises couleurs telles que décrites ci-dessus. Deux solutions possibles :

— installez Linux (ou Bash Ubuntu sous Windows 10 [10]) !
— testez l'affichage des couleurs de votre code sur les ordinateurs des salles machines du bâtiment NO.

2.6 Fonctions

Nous vous demandons d'implémenter les fonctions suivantes. Veillez à respecter scrupuleusement la signature des fonctions, c-à-d le nom (attention miniscules vs majuscules !) et l'ordre des paramètres. Ceci est indispensable pour les tests automatiques qui seront effectués lors de la correction. Un code ne respectant pas une des signatures sera sanctionné d'une note nulle, comme pour la partie 1.

`forwardPass(x,W1,W2) : return (z,z_out,grad,grad_out)`
Fonction qui effectue une forward pass d'un perceptron sur l'input (image) `x` avec les matrices de poids `W1` (couche entrée → couche intermédiaire) et `W2` (couche intermédiaire → couche sortie) comme décrit ci-dessus. Les valeurs retournées sont :
— `z` - Les activations des neurones de la couche intermédiaire (vecteur de la même taille que la couche intermédiaire, chacun est un nombre réel entre 0 et 1).
— `z_out` - L'activation du neurone de sortie (scalaire, nombre réel entre 0 et 1).
— `grad` - Les dérivées des activations des neurones de la couche intermédiaire (vecteur de la même taille que la couche intermédiaire, chacun est un nombre réel entre 0 et 1).
— `grad_out` - La dérivée de l'activation du neurone de sortie (scalaire, nombre réel entre 0 et 1).

`backpropagation(x,y,y_hat,z,W1,W2,grad,grad_out) : return (DeltaW1,DeltaW2)` Fonction qui effectue la backpropagation d'un perceptron multicouche avec :
— l'input (image) `x`
— les matrices de poids `W1` et `W2`
— les dérivées `grad` et `grad_out` précalculées pendant la forward pass
— les activations `z` des neurones de la couche intermédiaire
— la valeur réelle `y` et prédite `y_hat` pour la classe de l'image
comme décrit ci-dessus.
La fonction renvoie deux matrices `DeltaW1` et `DeltaW2` ayant respectivement les mêmes tailles que `W1` et `W2`, contenant les valeurs pour la mise à jour des poids.

`dropout(W1,p) : return (D_input,D_middle)`
Fonction qui implémente la procédure de dropout avec une probabilité de sélection `p` qui, étant donné la matrices de poids `W1`, renvoie :
— les vecteurs `D_input` et `D_middle` des booléens, indiquant l'état des neurones dans la couche d'entrée et dans la couche intermédiaire, respectivement, où 1 indique un neurone activé et 0 un neurone désactivé.

`train(data,digit,p=0.5,epsilon=0.0001,H=10,learning_rate=0.001,max_it=20) : return (W1,W2)`

Fonction qui effectue l'entraînement d'un perceptron sur le training set `data` pour reconnaître le chiffre `digit`. La première classe correspond aux images représentant le chiffre `digit`, la seconde aux autres. La fonction renvoie les meilleures matrices des poids `W1,W2` trouvées, selon la procédure d'entraînement décrite ci-dessus. Le nombre `H` de neurones de la couche intermédiaire est donné comme paramètre, ainsi que le `learning_rate` utilisé pour la mise à jour des poids (cf Eq. 6 et 7). Le paramètre `epsilon` est utilisé pour vérifier la convergence de la méthode d'apprentissage et `max_it` est la borne sur le nombre d'itérations lors de l'apprentissage. Enfin, p représente la probabilité de sélection d'un neurone pour la procédure de dropout.

10. Instructions disponibles au lien : `https://www.nextinpact.com/news/99572-bash-ubuntu-sous-windows-10-comment-installer.htm`

`predict(x, L) : return digit`
Fonction qui effectue la prédiction (globale) du chiffre représenté par l'input (image) `x` sur base de la liste `L` des matrices poids des dix perceptrons multicouches, comme décrit ci-dessus. La fonction renvoie le chiffre `digit` prédit.

`initWeights(nb_rows,nb_columns) : return W`
Fonction qui initialise une matrice de poids `W` de taille `nb_rows` x `nb_columns` avec des valeurs tirées de manière aléatoire comme décrit ci-dessus.

`displayConfusionMatrix(C) : return None`
Fonction qui effectue l'affichage de la matrice de confusion `C` comme décrit dans la section 2.5.

`convergenceCondition(DeltaW1,DeltaW2,epsilon=0.0001) : return bool`
Fonction qui implémente la condition d'arrêt décrite aux equations 14 et 15 sur base des matrices `DeltaW1` et `DeltaW2` contenant les valeurs pour la mise à jour des poids, et du paramètre `epsilon`.

`crossValidation(data,alpha=0.5,it_CV=4,p=0.5,epsilon=0.0001,H=10,learning_rate=0.001,max_it=20) : return None`
Fonction qui effectue la validation croisée sur le dataset `data` comme décrite ci-dessus, où `p`, `epsilon`, `H`, `learning_rate`, et `max_it` sont les paramètres qui sont utilisés lors des appels à la fonction `train`.

`print_color(string,color,sep) : return None`
Fonction auxiliaire pour l'affichage de la sequence de caractères `string` coloré selon le code couleur ANSI `color`, également passé comme chaine de caractères et terminé avec la sequence de caractères `sep`.

Vous êtes bien sûr libre d'implémenter des fonctions supplémentaires ; suivez les bonnes pratiques vues au cours de Programmation.

2.7 Fichiers à soumettre

A l'exception de la boucle principale, toutes les fonctionnalités décrites ci-dessus, et éventuellement vos fonctions supplémentaires, doivent être mises dans un fichier nommé `mlpFunctions.py`. Chaque fonction sera accompagnée d'un bref commentaire décrivant ce qu'elle fait.

La boucle principale, qui lit les données et lance la procédure de validation croisée, devra elle être présente dans un fichier séparé nommé `mlp.py`.

Rappel : sur Github Classroom vous devez soumettre les deux fichiers comme expliqué dans les consignes sans inclure les jeux de données !

Veuillez à respecter scrupuleusement les signatures des fonctions décrites ci-dessus, tout projet ne les respectant pas sera sanctionné d'une note nulle. (C'est en effet essentiel pour que vos correcteurs puissent facilement tester votre code !)

Votre programme fera un appel à la fonction `crossValidation` avec les valeurs par défaut pour les différents paramètres, sur le jeu de données `train_small.csv`.

Voici le lien pour obtenir accès à votre repository pour la partie 2 sur GitHub Classroom : `https://classroom.github.com/a/CWxO6vHK`

2.8 Rendu

Personne de contact : Jacopo De Stefani – jacopo.de.stefani@ulb.ac.be – O8.212

À remettre : Les scripts `mlp.py` et `mlp_functions.py` en `Python3` :

— Deadline sur GitHub Classroom : Mercredi 20 décembre 2017 à 22 heures. Conseil : **uploadez régulièrement votre code** sur le serveur !

— Remarque : pas de remise papier pour cette partie 2

2.9 Dernières remarques

Bien que pour la remise nous vous demandons d'utiliser les valeurs par défaut des paramètres et d'utiliser le jeu de données `train_small.csv` (par soucis de rapidité d'exécution), vous êtes encouragés à tester également votre code de votre côté sur `train.csv` et à jouer avec les différents paramètres pour voir l'impact sur le taux de réussite de votre classificateur. Testez par-exemple l'impact d'un plus grand nombre de neurones sur la couche intermédiaire ; d'autres critères d'arrêt (et quid si on renvoyait la dernière paire (W_1, W_2) trouvée au lieu de la meilleure ?) ; d'une initialisation des poids différentes, du paramètre p du dropout, etc.

3 Partie 3 : Interface graphique

La troisième partie du projet consiste en la mise en œuvre d'une *interface graphique* (*Graphical User Interface* en anglais, d'où GUI) pour votre logiciel qui permettra à l'utilisateur d'interagir avec votre programme de manière plus conviviale. Dans ce but, nous vous demandons d'utiliser la bibliothèque graphique `PyQt4` [11] (notons que d'autres bibliothèques graphiques pour `Python` existent également, telles que `TkInter`, `PyGTK` ou encore `wxPython`). Par ailleurs, nous vous demandons qu'au terme de cette troisième partie, votre code gère les erreurs produites lors de l'exécution à l'aide de gestion d'exceptions comme vu au cours de programmation.

3.1 Structure et fonctionnement d'une GUI

Une GUI est typiquement composée d'un ensemble d'éléments nommés *widgets* agencés ensembles. Une fenêtre, un bouton, une liste déroulante, une boîte de dialogue ou encore une barre de menus sont des exemples de widgets typiques. Il peut aussi s'agir d'un *conteneur* qui a pour rôle de contenir et d'agencer d'autres widgets.

Une GUI va typiquement être structurée, de manière interne, sous la forme d'un *arbre* de widgets. Dans `PyQt4`, tout programme doit avoir un widget « racine », qui est une instance de la classe `QWidget`, créant une fenêtre principale, dans laquelle on peut ensuite ajouter d'autres widgets.

Réaliser la structure visuelle d'une GUI (aussi appelée *maquette*) ne constitue que la moitié du travail, il faut que votre code des parties précédentes puisse être également lié à cette interface. Votre programme ne va donc plus être réduit à se lancer, effectuer des opérations et se terminer. À l'exécution, une GUI va réagir aux *événements* (par exemple, un clic d'un bouton). `PyQt4` offre notamment la possibilité de lier le clic d'un bouton à l'appel d'une fonction donnée. Donc, le principe d'une GUI est d'exécuter une boucle qui va traiter les événements jusqu'á la terminaison de l'application. Il vous faudra utiliser intelligemment ces possibilités pour mettre à jour l'affichage au besoin.

11. `https://riverbankcomputing.com`

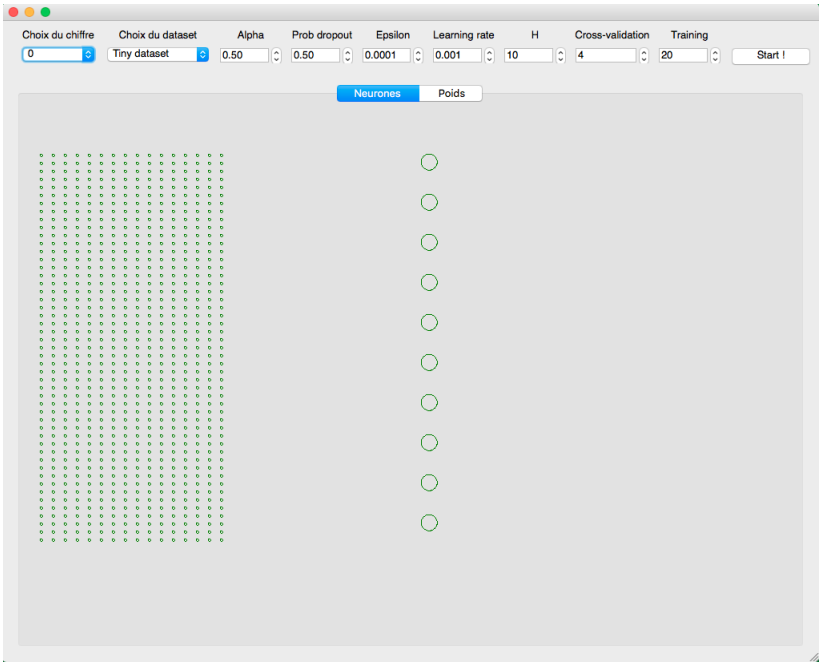


FIGURE 7 – Exemple du canevas du perceptron multicouche.

3.2 Exemple de maquette et contraintes

La Figure 7 vous donne un exemple de maquette. Il doit être possible de fermer votre programme en cliquant sur le bouton de fermeture de la fenêtre. Notez que la créativité est encouragée ; vous n'êtes pas tenu de reproduire au pixel près cet exemple. Veillez à ce que votre interface graphique soit ergonomique, agréable visuellement et simple d'utilisation.

Remarque sur la performance. Si vous liez une fonction qui nécessite un certain temps de calcul à un bouton de votre interface, celle-ci restera « gelée » après l'événement (clic) jusqu'à ce que la fonction termine son exécution. Dans le cadre de ce projet, nous nous contenterons de ce comportement (et ne le pénaliserons certainement pas lors de la notation). Pour éviter ce phénomène, il faut exploiter le concept de *multithreading* (qui sort du cadre de ce projet) enseigné aux cours INFO-F201 (Systèmes d'exploitation) et INFO-F202 (Langages de programmation 2).

3.3 Comportement de la GUI

Votre GUI est composée de deux parties : la partie supérieure où vous pouvez sélectionner les paramètres de votre perceptron multicouche et une partie inférieure où deux façons de visualiser le perceptron sont implémentés dans un *TabWidget*.

La partie supérieure doit contenir neuf paramètres pour la simulation du perceptron. L'un des paramètres vous permet de sélectionner uniquement le perceptron multicouche d'un chiffre pour la visualisation et un autre pour le choix du jeu de données à utiliser. Les autres paramètres sont les mêmes paramètres qu'utilisés durant la partie 2 du projet. Pensez bien aux limites imposées par les paramètres et les valeurs possibles que ceux-ci peuvent prendre lors de votre implémentation de la GUI. Un *Bouton* vous permettra de lancer la simulation avec les valeurs des paramètres entrées dans votre GUI.

La partie inférieure est un *TabWidget* composés de deux différents *tabs*. Le premier montre les neurones de la première couche et de la couche intermédiaire. À chaque itération lors de la fonction `train` de votre perceptron multicouche, certains de vos neurones sont en état de dropout ou pas. La GUI montrera les neurones activés à chaque étape en vert et les neurones désactivés en rouge, comme montré en exemple dans la Figure 8. Le second vous montre les poids de votre vecteur `W1`, donc les poids entre les neurones de votre première couche et votre couche intermédiaire. Dans ce cas-ci, votre perceptron multicouche **doit** être composé de **10** neurones pour la couche intermédiaire. Vous allez représenter 10 matrices de dimension *28x28*, représentant donc chacune les poids reçus par chaque neurone de la couche intermédiaire depuis les neurones de la première couche. Vos poids seront mis à jour dans les matrices à chaque fois que le score de votre perceptron multicouche s'est amélioré. Un exemple de représentation vous est montré à la Figure 9.

3.4 Utilisation des classes

L'utilisation des classes pour structurer votre code est plus que conseillée et fera l'objet d'une évaluation. Nous vous conseillons d'implémenter chacune des visualisations de votre perceptron multicouche dans une classe différente afin de pouvoir agir sur les éléments durant la simulation du perceptron. N'hésitez pas à adapter les fonctions du code la partie précédente aux besoins de chacune des visualisations.

Vous pouvez structurer vos classes dans différents fichiers. Il vous suffit alors d'importer les fonctions créées dans un autre fichier (par exemple le fichier `FileName.py`) grâce à la ligne `from FileName import *` dans votre script python. Veillez simplement à ce que votre GUI finale se lance grâce à la ligne de commande : `python3 partie3.py`. Veillez également à commenter les fonctions de votre code à l'aide de docstrings, ainsi que d'indiquer vos nom et prénom dans un docstring au début de votre code.

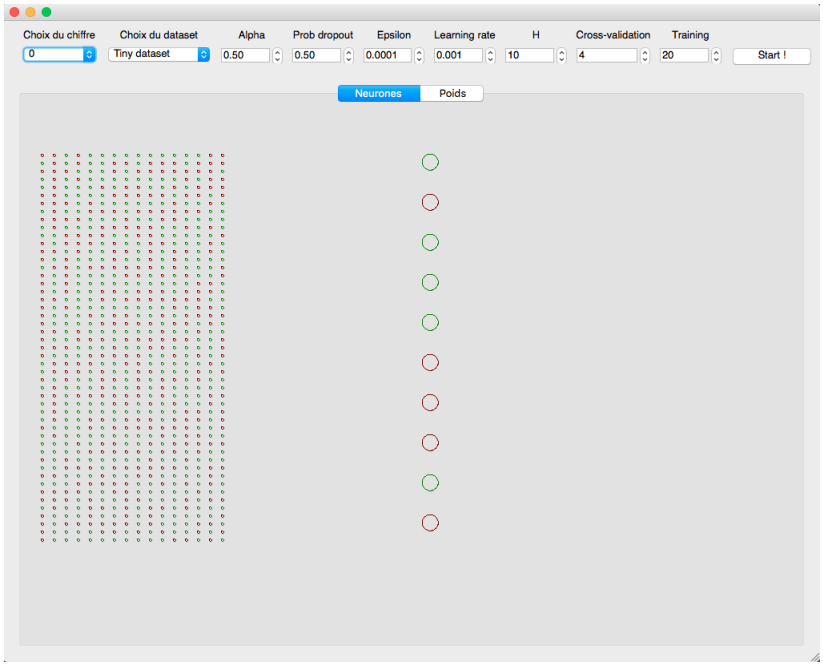


FIGURE 8 – Exemple d'une simulation avec les neurones désactivés en rouge et les neurones activés en vert.

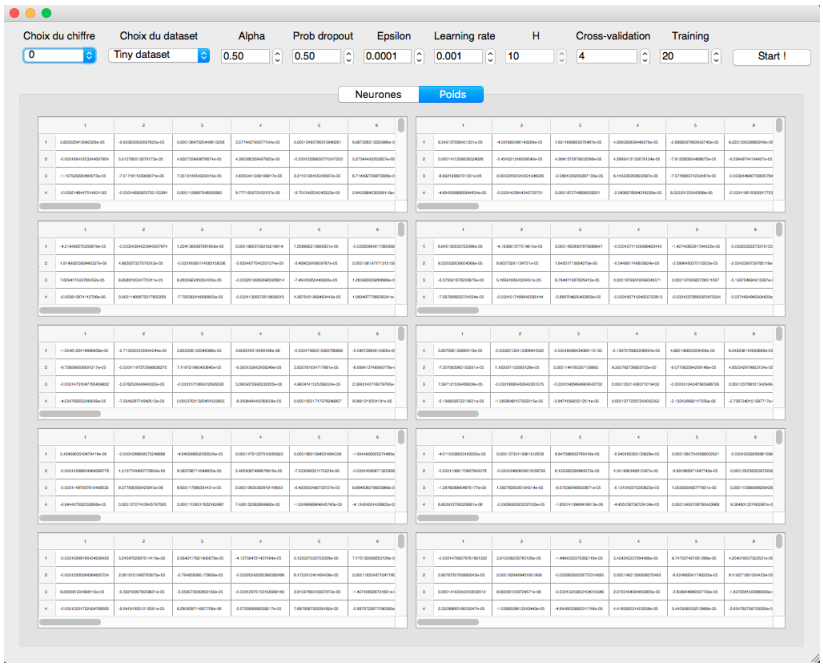


FIGURE 9 – Exemple du tab *Poids*.

3.5 Dessin sur un canevas `PyQt4`

La Figure 10 donne un exemple de code `PyQt4` créant un widget qui contient une zone de texte.

De la même manière, on peut ajouter des objets de types `Button`, `Label`, `Menu`, `Text` (et beaucoup d'autres) sur un `Window`. Nous vous invitons à explorer les différents éléments graphiques qui existent dans `PyQt4` afin de choisir les éléments qui conviennent le mieux à votre interface. N'hésitez pas à utiliser

```
import sys
from PyQt4 import QtGui

def window():
    app = QtGui.QApplication(sys.argv)
    w = QtGui.QWidget()
    b = QtGui.QLabel(w)
    b.setText("Hello World!")
    w.setGeometry(100,100,200,50)
    b.move(50,20)
    w.setWindowTitle("PyQt")
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    window()
```

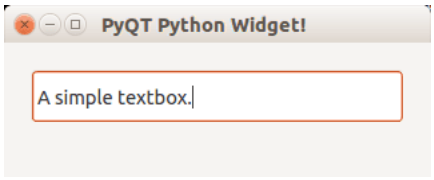


FIGURE 10 – Exemple de maquette *PyQt4*.

des couleurs dans votre interface graphique.

3.6 Références utiles

La documentation sur le module `PyQt4` est disponible à l'adresse suivante et pourra vous aider à développer votre interface graphique : `https://wiki.python.org/moin/PyQt`. De plus, vous avez la possibilité d'utiliser `Qt Designer` si vous le souhaitez, un outil d'aide à la conception d'interfaces graphiques. Sa documentation est disponible à l'adresse suivante : `http://pyqt.sourceforge.net/Docs/PyQt4/designer.html`. Les modules `PyQt4` et `Qt Designer` sont installés aux salles informatiques. Un tutoriel simple est disponible à l'adresse suivante : `https://www.tutorialspoint.com/pyqt`

3.7 Rendu

Personne de contact : Charlotte Nachtegael – charlotte.nachtegael@ulb.ac.be – N8.213

À remettre : Les scripts et les autres fichiers nécessaires au fonctionnement du projet (par exemple des images ou autres fichiers texte ou scripts python que vous utiliseriez, sans les jeux de données), dont le fichier `partie3.py` :

- Deadline sur GitHub Classroom : Dimanche 4 mars 2018 à 22 heures. Conseil : **uploadez régulièrement votre code** sur le serveur ! Lien pour obtenir l'accès au repository Github : `https://classroom.github.com/a/4qNiEf4S`
- Remarque : pas de remise papier pour cette partie

4 Partie 4 : Rapport et ajout de fonctionnalités supplémentaires

La quatrième partie du projet d'année consiste à améliorer votre projet selon différentes directions proposées ci-dessous. Nous vous demandons de **choisir au minimum deux de ces suggestions** pour votre partie 4. Vous êtes bien évidemment encouragés à implémenter plusieurs de ces propositions si vous avez le temps et l'envie. Vous verrez qu'une bonne part est laissée libre à votre créativité dans cet énoncé. Nous vous encourageons à en profiter et à personnaliser votre projet au maximum. Vous pouvez également proposer vous-même de développer d'autres aspects non mentionnés dans cet énoncé ; vous êtes alors encouragés à en discuter préalablement avec l'assistante en charge de cette partie, Charlotte Nachtegael, pour vous assurer de la pertinence de vos propositions.

Un rapport motivant les choix d'implémentation est également à rédiger, qui devra être remis une semaine après la partie 4. Remarque concernant la cotation : le projet dans son entiereté est côté sur 100. Chacune des quatre parties vaut 20 points, le rapport vaut 5 points, et enfin la présentation orale vaut 15 points.

4.1 Jeux de données Fashion MNIST et Cifar-10

Outre le jeu de données MNIST, nous vous proposons deux autres jeux de données sur l'UV, Fashion MNIST et Cifar-10. Ce sont d'excellents jeux de données pour évaluer différentes variantes de vos perceptrons.

Le jeu de données Fashion MNIST est un jeu de données créé par l'entreprise Zalando qui a exactement le même format que MNIST : 50000 images 28×28 noir et blanc (256 niveaux de gris) de vêtements vendus par Zalando, avec 10 classes de vêtements (0 T-shirt/top, 1 Pantalon, 2 Pullover, 3 Robe, 4 Manteau, 5 Sandale, 6 Shirt, 7 Sneaker, 8 Sac, 9 Botte). Par rapport au MNIST classique où atteindre un taux de réussite $> 90\%$ est facile (comme vous l'avez vu), reconnaître les images de Fashion MNIST est déjà un peu plus difficile. Attendez-vous à des résultats un peu au-dessus de 80% (cela dépend évidemment des améliorations faites à vos perceptrons, voir ci-dessous).

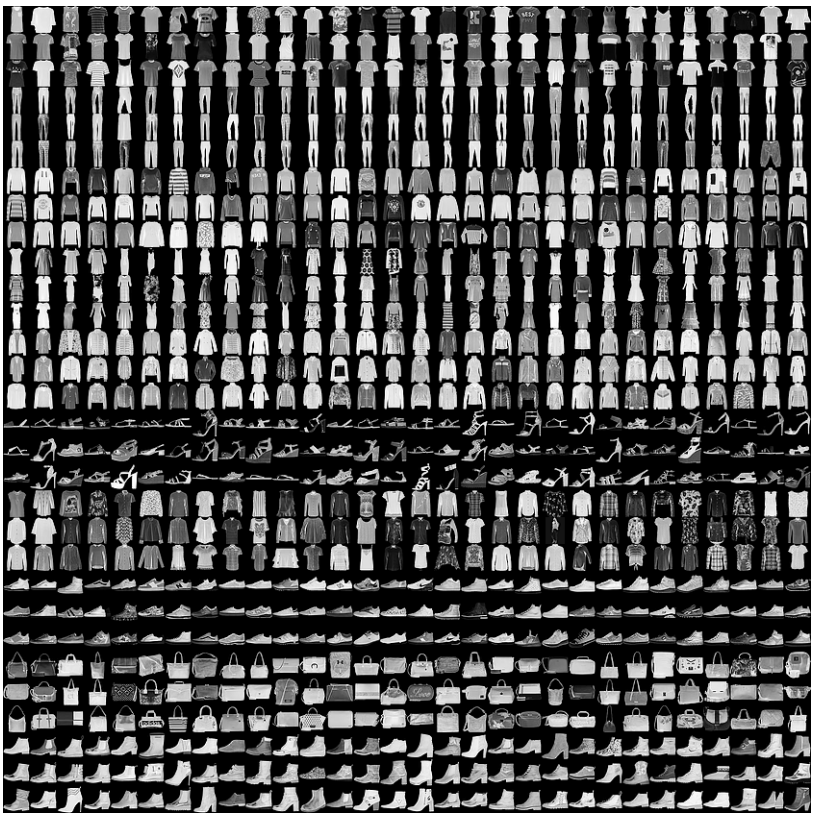


FIGURE 11 – Quelques images du jeu de données Fashion MNIST.

Sur l'UV vous trouverez le jeu de données Fashion MNIST (`fashion-mnist_train.csv`) ainsi qu'une version 'light' de 2000 images (`fashion-mnist_train_small.csv`).

Le jeu de données Cifar-10 est un jeu de données classique composé de 50000 images 32×32 en couleurs, avec 10 classes d'images (avion, voiture, oiseau, chat, cerf, chien, grenouille, cheval, bateau, camion). Ce jeu de données est beaucoup plus difficile, attendez-vous à des résultats dans les 20-30%.

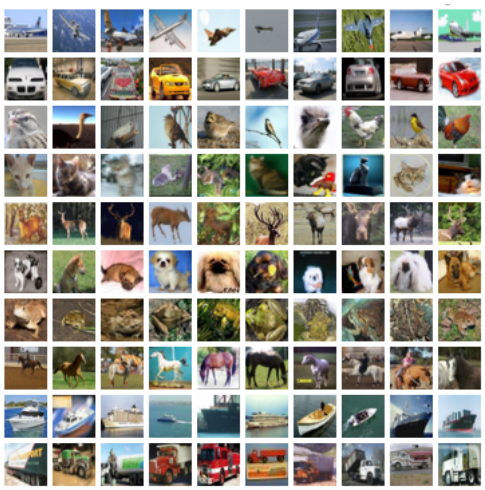


FIGURE 12 – Quelques images du jeu de données Cifar-10.

Sur l'UV vous trouverez le jeu de données Cifar-10 encodé et compressé à l'aide de `pickle`. Ce jeu de données est beaucoup plus volumineux (> 1 GB si on utilisait le format `csv`), d'où notre choix de le compresser. Nous vous fournissons une fonction vous permettant de lire directement le jeu de données, voir `readCifar10.py`. Notez le paramètre optionnel `small`, qui permet de ne lire que les 2000 premières images.

Vous pouvez également choisir vous-même d'autres jeux de données, voir le site web Kaggle (`https://www.kaggle.com/datasets`) pour une grande collection de jeux de données de tous les styles, allant de statistiques compilées par des gouvernements au jeu Pokemon [12].

4.2 Perceptron multiclasse et multicouche

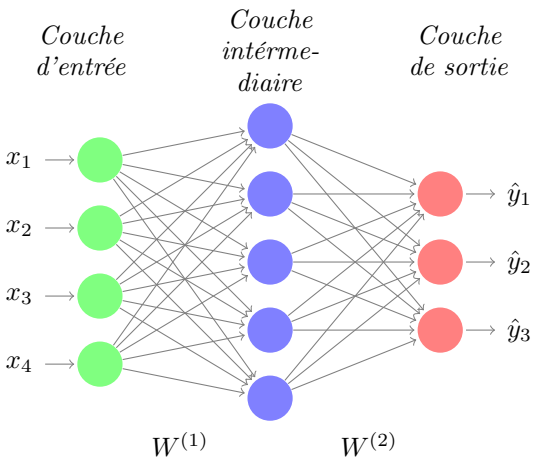


FIGURE 13 – Exemple d'un perceptron à trois classes avec une couche interne

12. 1 point bonus à l'étudiant(e) qui arrivera à faire quelque chose d'intéressant avec ce dernier ;)

Durant les parties 1 et 2, nous avons approché le problème de classer les images du MNIST à l'aide de 10 perceptrons binaires, chacun en charge de reconnaître son propre chiffre. La classification globale était alors faite en choisissant le chiffre du perceptron le plus convaincu d'avoir reconnu son chiffre. Cette approche a le mérite d'être très simple tout en donnant de bons résultats. Cependant, il est en général un peu plus performant de ne créer qu'un seul perceptron mais ayant autant de neurones sur la couche de sortie que de classes. C'est que nous vous proposons de faire ici ; nous vous donnons les algorithmes de forward pass et de back-propagation dans le cas multiclasse pour un nombre arbitraires de couches internes sous forme de pseudo-code. A vous de les implémenter.

Nous noterons d la *profondeur* du réseau, le nombre de couches hormis la couche d'entrée. (Les perceptrons de la partie 2 sont donc de profondeur 2). Numérotons les couches de 0 à d, la couche 0 étant la couche d'entrée, et notons $s_0, s_1, \ldots, s_d$ leurs tailles respectives. (Donc $s_0 = 784$ et $s_d = 10$ dans le cas du MNIST). Notons également $W^{(i)}$ la matrice des poids des connexions entre les couches $i-1$ et i, pour chaque $i \in \{1, \ldots, d\}$. La matrice $W^{(i)}$ est une matrice $s_{i+1} \times s_i$. Nous utiliserons f pour la fonction d'activation.

Pour une entrée (image) représentée sous forme d'un vecteur x, l'algorithme de forward pass est une généralisation directe de celui de la partie 2 :

$h^{(0)} \leftarrow x$
Pour $i = k, \ldots, 1$ faire
  $a^{(i)} \leftarrow W^{(i)} h^{(i-1)}$
  $h^{(i)} \leftarrow f(a^{(i)})$

Le vecteur $a^{(i)}$ est donc le vecteur des activations des neurones de la couche i, c-à-d les valeurs calculées avant l'application de la fonction d'activation f, et $h^{(i)}$ est ce même vecteur une fois la fonction d'activation appliquée à chaque entrée. (NB : La notation $f(a^{(i)})$ signifie appliquer la fonction f à chaque entrée du vecteur $a^{(i)}$). Les valeurs calculées par les neurones de la couche de sortie sont donc reprises dans le vecteur $h^{(k)}$, et la prédiction sur la classe de x est faite via un $\arg\max$.

Considérons maintenant la back-propagation. Soit y un vecteur de taille s_d représentant la bonne réponse : l'entrée correspondant à la classe de x est mise à 1, et les autres entrées sont mises à 0. La *perte* $J = \frac{1}{2} \sum_{j=1}^{s_d} (y_j - h_j^{(k)})^2$ est la mesure standard de l'erreur de classification du réseau de neurone. (Remarquez que J est simplement le carré de la distance Euclidienne entre les vecteurs y et $h^{(k)}$; nous souhaiterions que le réseau produise un vecteur $h^{(k)}$ le plus proche possible de y). Le but de l'algorithme de back-propagation est de calculer les gradients $\nabla_{W^{(i)}} J$ pour chaque matrice de poids $W^{(i)}$. Une fois ces gradients calculés, la mise à jour des poids se fait simplement en "suivant" le gradient pour diminuer la perte J :

$$W^{(i)} \leftarrow W^{(i)} - \eta \cdot \nabla_{W^{(i)}} J$$

Ici η est le *learning rate*, un des paramètres de votre modèle à choisir.

L'algorithme de back-propagation est une (très belle !) illustration de la règle de dérivation $\frac{\partial z}{\partial w} = \frac{\partial z}{\partial v} \frac{\partial v}{\partial w}$: L'objectif étant de calculer les dérivées partielles de J par rapport aux variables apparaissant dans les matrices de poids, l'algorithme va d'abord calculer des dérivées partielles de J par rapport à des variables intermédiaires et utiliser cette règle afin d'arriver à son but. Il fonctionne comme suit :

$g \leftarrow \nabla_{h^{(k)}} J = h^{(k)} - y$
Pour $i = k, \ldots, 1$ faire
  $g \leftarrow \nabla_{a^{(i)}} J = g \odot f'(a^{(i)})$
  $\nabla_{W^{(i)}} J \leftarrow g \, h^{(i-1)\intercal}$
  $g \leftarrow \nabla_{h^{(i-1)}} J = W^{(i)\intercal} g$

Ici f' représente la dérivée de la fonction d'activation f, et comme avant, $f'(a^{(i)})$ signifie qu'on applique f' à chaque entrée du vecteur $a^{(i)}$. La notation $\odot$ signifie le produit "composante par composante" (produit d'Hadamard), donc $(u_1, u_2, u_3) \odot (v_1, v_2, v_3) = (u_1 v_1, u_2 v_2, u_3 v_3)$ par exemple. Aussi, $A^\intercal$ signifie la transposée de la matrice A. Enfin, attirons l'attention sur le produit $g \, h^{(i-1)\intercal}$ entre le vecteur g et la *transposée* du vecteur $h^{(i-1)}$, qui donne donc une matrice ayant le même nombre de lignes que g et le

même nombre de colonnes que $h^{(i-1)}$. Par exemple, si $u = (u_1, u_2, u_3, u_4)$ et $v = (v_1, v_2, v_3)$, alors

$$uv^\intercal = \begin{cases} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \\ u_3v_1 & u_3v_2 & u_3v_3 \\ u_4v_1 & u_4v_2 & u_4v_3 \end{cases}$$

Pour finir, la validation croisée se fait exactement comme précedemment.

A explorer. Voici des suggestions de choses que vous pouvez tester et comparer dans votre rapport (graphiques encouragés !) sur un ou plusieurs jeux de données.

- *Finetuning* des paramètres d'initialisation des poids aléatoires (l'écart type de la normale) et du learning rate η, qui ont une importance cruciale sur l'apprentissage. Si votre code est correctement écrit mais que le réseau ne semble pas apprendre, c'est fort probablement que vous devez jouer avec ces paramètres. [a]
- Nombre et taille des couches internes de neurones. [b]
- Nombre d'itérations sur le jeu de données d'entraînement. Vous êtes libres d'implémenter votre perceptron comme vous le souhaitez, vous pouvez utiliser un nombre d'itérations fixe ou un critère de convergence.
- Mise à jour des poids après chaque image. Lors des parties 1 et 2, on effectuait une mise à jour des poids seulement lorsqu'une image était mal classée. Cela fonctionne déjà bien sur MNIST et le code est ainsi assez rapide. Cependant il est en général conseillé d'effectuer une back-propagation et mise à jour des poids systématiquement après chaque image, l'idée étant de renforcer aussi les bonnes réponses.

a. Notez que des valeurs qui fonctionnaient bien pour les perceptrons de la partie 2 ne sont pas garanties d'être de bons choix ici, il faut tester.

b. Augmenter ces paramètres devient rapidement très gourmand en ressources CPUs. Le but ici n'est pas d'aller le plus haut possible mais bien d'étudier l'impact sur le taux de réussite lorsque vous jouez avec ces paramètres. Ce n'est pas grave si l'ordinateur du NO (ou le vôtre) est limité à de petites valeurs, explorez différentes valeurs dans un intervalle réaliste pour la machine, pas besoin de faire tourner du code pendant une journée ! Ce qui nous intéresse ce n'est pas le meilleur taux de réussite atteint mais bien votre capacité à analyser et comparer des résultats.

4.3 Fonctions d'activation

Durant la partie 2, vous avez utilisé la sigmoïde comme fonction d'activation :

$$f(x) = \frac{1}{1 + e^{-x}} \tag{18}$$

On peut travailler avec d'autres fonctions d'activation et comparer leurs performances. Bien qu'historiquement la sigmoïde était une fonction d'activation prisée, ces dernières années la fonction d'activation la plus souvent utilisée est la fonction *ReLU* (rectified linear unit) :

$$f(x) = max(0, x) \tag{19}$$

Cette fonction remplace donc simplement toute valeur négative par 0. Elle possède de nombreuses propriétés qui en font une fonction d'activation de choix.

Voici d'autres exemples de fonctions d'activation qui sont parfois utilisées :

- Fonction tangente hyperbolique : $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Leaky ReLU : $f(x) = max(0.1x, x)$
- SWISH : $f(x) = \frac{x}{1+e^{-x}}$

D'autres fonctions d'activation sont également disponibles sur internet. Nous vous encourageons à tester les différentes fonctions et à voir l'impact de leur utilisation sur vos résultats.

4.4 Dropout

Le mécanisme de dropout reste le même pour des perceptrons multiclasses et multicouches. Vous pouvez ici étudier par-exemple l'impact de la probabilité p du dropout.

Pour rappel, dropout n'est utilisé que lors de la phase d'entraînement et pas lors de la phase de test. Un aspect ignoré lors de la partie 2 et qu'il convient de mentionner ici est qu'il faut idéalement normaliser

les poids *lors de la phase de test* (et uniquement celle-ci) comme suit : chaque matrice poids $W^{(i)}$ est multipliée par un facteur correcteur $(1-p)$. En effet, si on pense aux connexions entrantes d'un neurone lors de la phase d'entraînement, chacune des connexions ne sera active qu'avec la probabilité $1-p$, *en moyenne* on s'attend donc à ce que l'activation de ce neurone soit $(1-p)$ fois l'activation s'il n'y avait pas de dropout, d'où le facteur correcteur lors de la phase de test.

Dropout fonctionne aussi mieux avec certaines fonctions d'activation (comme ReLU) que d'autres, c'est un aspect qui pourrait également être exploré.

4.5 Améliorer l'interface graphique

Durant la partie 3, vous avez implémenté la visualisation de la phase d'entraînement de deux façons : en observant la phase de dropout des neurones et à travers la mise à jour des poids entre la couche d'entrée et la couche intermédiaire.

En particulier, nous n'avons pas exploré la visualisation de la prédiction. Une façon de faire pourrait être de visualiser les différents perceptrons multicouches et les scores obtenus pour une certaine image (qui pourrait aussi apparaître) pour chacun d'entre eux, la prédiction obtenue en conséquence et si cette prédiction est la bonne ou la mauvaise. Vous pourriez également améliorer la visualisation que vous avez implémentée lors de la partie 2 en convertissant les tables en images pour voir quels pixels sont considérés importants pour déterminer à quel chiffre nous avons affaire (voir l'exemple de la Figure 14).

Toute autre idée pour enrichir l'interface graphique est la bienvenue, créativité encouragée !

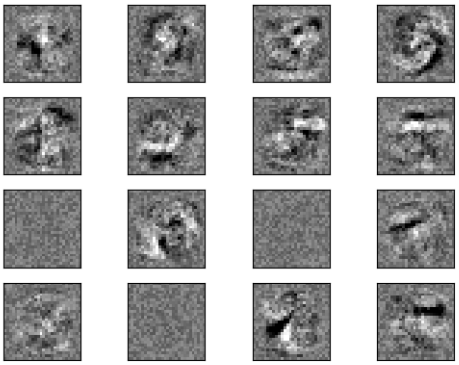


FIGURE 14 – Exemple de visualisation graphique des poids.

4.6 Rapport

En plus du code, nous vous demandons de rédiger un rapport faisant entre 3 et 5 pages (page de garde, annexes et table des matières non comprises). Ce rapport devra contenir les éléments suivants :

- une page de garde qui reprend vos coordonnées, la date, le titre, *etc.* ;
- une table des matières ;
- une introduction et une conclusion ;
- des sections décrivant ce qui a été réalisé, les comparaisons de performance, etc. ;
- des exemples d'exécution de votre code ;
- éventuellement des références bibliographiques si applicable.

Le rapport doit détailler les éventuelles difficultés rencontrées, l'analyse et les solutions proposées, ainsi qu'une bonne explication des algorithmes (e.g. pseudo-code avec des exemples et/ou des diagrammes, pas de code source). Il doit également contenir, si applicable, une comparaison de performances entre vos implémentations ou jeux de données utilisés. Il doit être clair, et compréhensible pour un étudiant imaginaire qui aurait suivi le cours INFO-F-101 (Programmation) mais n'aurait pas fait le projet ; ni

trop d'informations ni trop peu. Toutes les parties doivent être détaillées et présentées dans un ordre logique. Expliquez toutes les notions et terminologie que vous introduisez.

Le rapport peut être écrit soit en LaTeX, soit à l'aide des logiciels de traitement de texte `LibreOffice Writer`, `OpenOffice Writer` ou `Microsoft Word`. Il est obligatoire d'utiliser les outils de gestion automatique des styles, de la table de matière et de numérotation des sections. Nous vous conseillons d'utiliser le système LaTeX, très puissant pour une mise en page de qualité. Il est évident qu'une bonne orthographe sera exigée.

4.7 Rendu

Personne de contact : Charlotte Nachtegael charlotte.nachtegael@ulb.ac.be – N8.213

À remettre : Les scripts et les autres fichiers nécessaires au fonctionnement du projet ainsi que le rapport dans le répertoire :

— Deadline sur GitHub Classroom : Dimanche 8 avril 2018 à 22 heures. Conseil : **uploadez régulièrement votre code** sur le serveur ! Lien pour obtenir l'accès au repository Github : `https://classroom.github.com/a/2Sonp5Ae`

— Remise du rapport uniquement en version papier au secrétariat : Lundi 16 avril avant 13h.