TP 3 : Réseau de neurones - classification

L'objectif de ce TP est de mettre en œuvre un réseau de neurones de type perceptron multicouches pour la classification d'un grand nombre d'images. Vous utiliserez tout d'abord uniquement les librairies usuelles comme `numpy`, `pandas` et `matplotlib` afin de développer toutes les fonctions permettant d'entraîner et d'utiliser un réseau de neurones. Vous utiliserez ensuite la librairie `keras` intégrée à l'outil d'apprentissage automatique `TensorFlow` développé par Google. Cette librairie vous permettra de construire facilement un réseau de neurones et d'étudier l'impact de son architecture sur les performances finales.

1 Développement d'un réseau de neurones de A à Z

Vous allez tout d'abord développer votre propre réseau de neurones dans l'objectif de classifier les images de la base de données MNIST (*Mixed National Institute of Standards and Technology*). Ces images de 28×28 pixels correspondent à des chiffres de 0 à 9 écrits à la main (cf. panneau gauche de la Fig. 1). Votre objectif consiste donc à entraîner un réseau de neurones pour qu'il associe un chiffre de 0 à 9 à l'image d'entrée. L'architecture du réseau de neurones à développer est très simple :

- une couche d'entrée contenant autant de neurones que de pixels présents dans les images à traiter, *i.e.* 784 neurones
- une couche cachée contenant 10 neurones avec une fonction d'activation ReLU
- une couche de sortie contenant autant de neurones que de classes possibles, *i.e.* 10, avec une fonction d'activation softmax pour associer une probabilité à chaque classe

L'architecture de ce réseau de neurones est représentée sur le panneau droit de la Fig. 1. Elle contient donc deux matrices de poids W^0 et W^1 ainsi que deux vecteurs de biais b^0 et b^1.

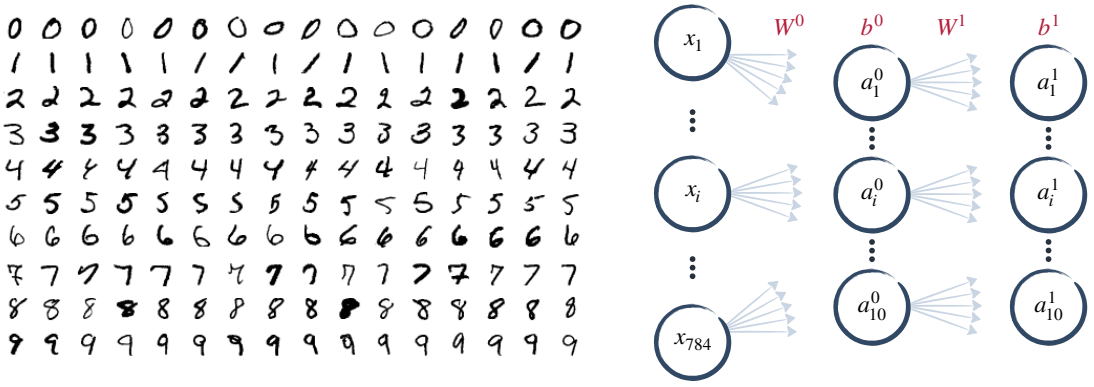

Figure 1: **Gauche :** Exemples d'images présentes dans la base de données MNIST. **Droite :** Représentation schématique du réseau de neurones à mettre en œuvre.

Vous trouverez un fichier de données dans le dossier TP sur Moodle nommé train.csv.zip. Une fois décompressé, ce fichier contient les images de la base de donnée MNIST ainsi que leur classe respective permettant l'entraînement et la validation de votre réseau de neurones.

1.1 Échantillons d'entraînement et de validation

La première étape de ce projet consiste à définir les échantillons d'entraînement et de validation.

- Utilisiez la librairie `pandas` pour ouvrir le fichier train.csv et convertissez le data-frame correspondant en tableau `numpy`. Affichez les dimensions de ce tableau. Combien y a-t-il d'images au total ? Pourquoi y a-t-il 785 valeurs dans la deuxième dimension ?

- Utilisez la fonction `random.shuffle` de `numpy` afin de mélanger l'ordre des images de la base de données.

- Définissez votre échantillon de validation X_dev en sélectionnant les 1000 premières images du tableau. Définissez également le tableau Y_dev contenant les classes associées à ces 1000 images.

- Définissez les tableaux X_train et Y_train contenant les images et les classes d'entraînement en utilisant toutes les images restantes après les 1000 premières.

- Normalisez les images d'entraînement et de validation pour que les valeurs des pixels varient entre 0 et 1.

1.2 Initialisation - Propagations avant et arrière

Maintenant que les échantillons d'entraînement et de validation sont définis, vous allez développer les fonctions nécessaires pour entraîner le réseau de neurones. **Aucune des fonctions que vous développerez dans cette section ne nécessite de boucle for.**

- Implémentez une fonction qui initialise les valeurs contenues dans les matrices W^0, W^1, b^0 et b^1 suivant une loi uniforme comprise entre -0.5 et 0.5 et qui retourne ces quatre matrices.

- Définissez une fonction ReLU qui prend un tableau Z en argument et qui renvoie les valeurs de $\text{ReLU}(Z)$ sous forme de tableau.

- De même, définissez une fonction softmax qui renvoie un tableau contenant les valeurs de $\text{softmax}(Z)$.

- Implémentez une fonction qui réalise la propagation avant pour une image X_i donnée (sous la forme d'un vecteur de 784 valeurs). Utilisez les formules vues en cours pour que cette fonction renvoie les vecteurs Z^0, A^0, Z^1 et A^1.

- Définissez une fonction qui renvoie la dérivée de la fonction ReLU pour un vecteur Z donné.

- Pour pouvoir comparer la sortie du réseau de neurones avec la classe attendue pour une image donnée, il est nécessaire que la classe prenne la forme d'un vecteur de 10 valeurs contenant un 1 à la position correspondant au chiffre affiché sur l'image et des 0 aux 9 autres positions (*exemple : 7 devient [0,0,0,0,0,0,0,1,0,0]*). Implémentez une fonction qui renvoie la classe Y_i considérée (chiffre) sous forme d'un vecteur de 10 valeurs.

- Définissez une fonction qui réalise la propagation arrière pour une image X_i et une classe Y_i données. Cette fonction doit renvoyer les matrices dJdW0, dJdb0, dJdW1 et dJdb1 correspondant aux dérivées de la fonction de perte (entropie croisée) par rapport à W^0, W^1, b^0 et b^1. Utilisez les formules vues en cours pour calculer ces matrices.

- Finalement, implémentez une fonction qui actualise les valeurs contenues dans W^0, W^1, b^0 et b^1 à partir de listes dJdW0_L, dJdb0_L, dJdW1_L et dJdb1_L contenant les matrices dJdW0, dJdb0, dJdW1 et dJdb1 calculées pour chaque couple (X_i, Y_i) et d'un taux d'apprentissage λ. Cette fonction renvoie les nouvelles matrices W^0, W^1, b^0 et b^1.

1.3 Entraînement du réseau de neurones - exactitude

Vous disposez à présent de toutes les fonctions permettant de mettre en œuvre l'algorithme d'entraînement de votre réseau de neurones. Vous allez maintenant développer les outils donnant le taux de succès du réseau de neurones à différentes phases de l'entraînement et mettre en place l'algorithme final pour obtenir les matrices W^0, W^1, b^0 et b^1.

- Définissez une fonction qui renvoie la classe de plus haute probabilité pour une sortie du réseau de neurones donnée.
- Implémentez une fonction qui renvoie le taux de succès du réseau de neurones en comparant un tableau de prédictions obtenues avec le tableau des classes attendues.
- Définissez une fonction qui effectue l'entraînement du réseau de neurones à partir des tableaux X_train et Y_train en utilisant les fonctions que vous avez développées jusqu'à maintenant. Suivez l'algorithme vu en cours pour construire cette fonction. Cette fonction devra contenir deux boucles. La boucle principale sur le nombre d'itérations effectuées (nombre d'actualisation des matrices de poids et des vecteurs de biais). Une boucle insérée dans la boucle principale sur les images contenues dans l'échantillon d'entraînement afin de remplir les listes dJdW0_L, dJdb0_L, dJdW1_L et dJdb1_L. Cette fonction d'entraînement devra renvoyer les matrices W^0, W^1, b^0 et b^1 obtenues à la dernière itération.
- Utilisez les fonctions donnant la classe prédite par le réseau de neurones pour une sortie donnée ainsi que le taux de succès pour afficher ce dernier toutes les 10 itérations.
- Lorsque vous arrivez à cette étape appelez votre enseignant pour vérifier l'efficacité de vos codes (la phase d'entraînement peut être très longue...). Lancez l'entraînement de votre réseau de neurones en utilisant 100 itérations ainsi qu'un taux d'apprentissage de 1.0 et sauvegardez les poids et biais obtenus dans un fichier `numpy` au format .npz.
- Définissez une fonction qui permette d'obtenir la classe prédite pour les images de l'échantillon de validation à partir des matrices W^0, W^1, b^0 et b^1 obtenues à la fin de l'entraînement. Quel taux de succès obtenez-vous sur l'échantillon de validation ?
- (Bonus) Utilisez les matrices W^0, W^1, b^0 et b^1 obtenues à la fin du premier entraînement pour initialiser un second entraînement de 100 itérations avec un taux d'apprentissage de 0.1. Sauvegardez les nouveaux poids et biais obtenus. Le taux de succès obtenu sur l'échantillon de validation est-il meilleur ?

2 Utilisation d'une librairie dédiée : Tensorflow-Keras

Maintenant que la mécanique des réseaux de neurones n'a plus de secret pour vous, vous pouvez utiliser les nombreuses librairies disponibles pour mettre en place et entraîner efficacement une architecture quelconque. Parmi tous ces outils, la librairie d'apprentissage `TensorFlow` développée par Google est l'une des plus utilisée. Depuis 2017, `TensorFlow` intègre la librairie `Keras` dédiée aux réseaux de neurones profonds. Vous allez à présent utiliser cet outil pour étudier l'impact de l'architecture du réseau de neurones et du taux d'apprentissage sur le taux de succès de classification des images MNIST.

- Importez la sous-librairie `keras` de `tensorflow`. Utilisez le module `mnist` de `keras.datasets` afin de définir les tableaux X_train, Y_train, X_test et Y_test. Normalisez les tableaux X_train et X_test pour que chaque image varie entre 0 et 1. Utilisez la fonction `reshape` associée aux tableaux X_train et X_test pour aplatir les images, *i.e.* transformer les

deux dimensions (28,28) en une unique dimension (784). Finalement, utilisez la fonction `to_categorical` de `keras.utils` pour transformer les chiffres contenus dans Y_train et Y_test (classes associées aux images) en vecteurs de 10 valeurs contenant un 1 à la position correspondant au chiffre affiché sur chaque image.

- Utilisez le module `Sequential` de `keras.models` afin d'initialiser un objet *model* qui correspondra à votre réseau de neurones. Utilisez le module `Input` de `keras.layers` ainsi que la méthode `add` associée à votre objet *model* afin d'ajouter la couche d'entrée de 784 neurones. De même, utilisez le module `Dense` de `keras.layers` afin d'ajouter la couche cachée de 10 neurones en spécifiant la fonction d'activation ReLU. Ajoutez finalement la couche de sortie de 10 neurones de fonction d'activation softmax.

- Utilisez la fonction `Adam` de `keras.optimizers` afin d'initialiser un objet *opt* correspondant à la méthode employée pour effectuer la minimisation de la fonction de perte. Vous considérerez un taux d'apprentissage de 10^{-2}.

- Utilisez la méthode `compile` associée à votre objet *model* afin d'associer l'objet *opt* à votre réseau de neurones. Vous utiliserez l'entropie croisée ('categorical_crossentropy') comme fonction de perte et vous demanderez également l'accès à l'évolution du taux de succès en fonction des itérations de la phase d'entraînement en ajoutant le mot clé metrics = ['accuracy'].

- Utilisez la méthode `fit` associée à votre objet *model* afin d'entraîner votre réseau de neurones à partir des tableaux X_train et Y_train. Vous fixerez l'option batch_size au nombre d'images dans X_train et le nombre d'itérations (epochs) à 300. Utilisez également l'option validation_data pour donner les tableaux X_test et Y_test afin que le taux de succès soit calculé pour l'échantillon de validation à chaque étape de l'entraînement. Stockez les sorties de cette phase d'entraînement dans une variable *out*.

- Quelles sont les différentes clés associées au dictionnaire out.history ? Tracez l'évolution de la fonction de perte et du taux de succès pour les échantillons d'entraînement et de validation en fonction de l'itération. Quel est le taux de succès maximal atteint avec cette architecture de réseau de neurones ? Ce dernier semble-t-il avoir convergé ? Augmentez le nombre d'époques de la phase d'entraînement pour qu'il converge à $\sim 10^{-3}$ près. Quelle valeur maximale obtenez-vous ?

- Effectuez le même entraînement avec un taux d'apprentissage de 0.2. Quelle valeur maximale du taux de succès obtenez-vous ?

- Définissez un nouveau réseau de neurones en augmentant le nombre de neurones dans la couche cachée significativement (en le multipliant par 50 par exemple). Comment évolue le taux de succès final ?

- Ajoutez une nouvelle couche cachée de fonction d'activation ReLU après la première pour augmenter la profondeur de votre réseau. Utilisez un nombre important de neurones (par exemple 700). Effectuez un entraînement de 200 époques. Quel est le taux de succès final pour les échantillons d'entraînement et de validation ?

- À partir du réseau défini à l'étape précédente, modifiez la variable batch_size de la phase d'entraînement en la divisant par 10. Répétez la phase d'entraînement pour 200 époques. Comment évolue la fonction de perte de l'échantillon de validation ? À partir de quelle époque peut-on considérer que l'on atteint un régime de surentraînement ?