IFT-3395/6390 Fundamentals of Machine Learning
Professors: Guillaume Rabusseau / Ioannis Mitliagkas

Homework 3 - Practical component

Devoir 3 - Partie pratique

- This homework must be done and submitted to Gradescope in teams of 3. You are welcome to discuss with students outside of your group but the solution submitted by a group must be its own. Note that we will use Gradescope's plagiarism detection feature. All suspected cases of plagiarism will be recorded and shared with university officials for further handling.
  Ce devoir doit être déposé sur Gradescope et doit être fait en équipe de 3. Vous pouvez discuter avec des étudiants d'autres groupes mais les réponses soumises par le groupe doivent être originales. À noter que nous utiliserons l'outil de détection de plagiat de Gradescope. Tous les cas suspectés de plagiat seront enregistrés et transmis à l'Université pour vérification.

- The practical part should be coded in python (the only external libraries you can use are numpy and matplotlib) and all code will be submitted as a python file to Gradescope. To enable automated code grading you should work off of the template file given in this homework folder. Do not modify the name of the file or any of the function signatures of the template file or the code grading will not work for you. You may, of course, add new functions and any regular python imports.
  La partie pratique doit être codée en python (avec les librairies numpy et matplotlib), et envoyée sur Gradescope sous la forme d'un fichier python. Pour permettre l'évaluation automatique, vous devez travailler directement sur le modèle donné dans le répertoire de ce devoir. Ne modifiez pas le nom du fichier ou aucune des fonctions signatures, sinon l'évaluation automatique ne fonctionnera pas. Vous pouvez bien sûr ajouter de nouvelles fonctions et importations python

- Any graphing, charts, derivations, or other practical report parts should be submitted to Gradescope included at the end of your report for the theoretical part of the homework.

You are of course encouraged to draw inspiration from what was done in lab sessions.
Les figures, courbes et réponses aux questions (autre que du code) doivent être déposées sur Gradescope en les incluant dans le pdf que vous soumettez pour la partie théorique. Vous êtes bien sûr encouragés à vous inspirer de ce qui a été fait en TP.

1 Neural network implementation [Undergraduates 50 pts][Graduates 50 pts]

This part consists of the implementation of a feedforward neural network (or multi-layer perceptron, MLP), with an arbitrary number of layers, for multiclass classification. You will need to implement the necesary functions inside the `NN` class provided in the solution template. In the next section, you will use the CIFAR-10 dataset.

The output layers of the neural networks considered here are made of m neurons that are fully connected to the previous hidden layers. They are equipped with a **softmax** non-linearity.

The output of the j^{th} neuron of the output layer gives a score for the class j which is interpreted as the probability of x being of class j.

You will train your neural network with minibatch stochastic gradient descent, using the cross entropy loss.

The `NN` class is initialized with the following parameters:

Cette partie consiste en l'implémentation d'un réseau de neurones (perceptron multicouches, MLP), avec un nombre de couches cachées arbitraire, pour la classification multiclasse. Vous devrez implémenter les fonctions nécessaires dans la classe `NN` qu'on vous donne dans le template de la solution. Dans la section suivante, vous utiliserez le jeu de données CIFAR-10.

Les dernières couches (output layers) des réseaux considérés ici sont fait de m neurones connectés à la couche précédente. Ils sont équipés de la non-linéarité **softmax**.

L'output du $j^{\text{ème}}$ neurone de la dernière couche donne un score pour la classe j, qui est interprété comme étant la probabilité que x appartienne j.

Vous entraînerez votre réseau de neurones avec la descente de gradient stochastique (minibatch), en utilisant la fonction de coût entropie croisée (cross entropy loss).

La classe `NN` est initialisée avec les paramètres suivants:

- `hidden_dims`: a tuple containing the number of neurons in each hidden layer. The size of the tuple is the number of **hidden** layers of your network (meaning that if there are $L-1$ hidden layers, there should be $L+1$ layers in total if we count the input and output layers). - un tuple contenant le nombre de neurones dans chaque couche cachée. La taille du tuple est le nombre de couches cachées de votre réseau (ç-à-d s'il y a $L-1$ couches cachées, il devrait y avoir $L+1$ couches au total en comptant les couches d'entrée et de sortie).

- `datapath`: a string containing the path to the dataset. The code to load the dataset and to split it into training, validation, and test sets is given to you in the `__init__` function. - une chaîne de caractères contenant le chemin vers le jeu de données. Le code pour charger les données et les diviser en ensembles d'entraînement, validation, et test vous est donnée dans la fonction `__init__`:

- `n_classes`: the number of classes in the classification problem (also the number of neurons of the output layer). - le nombre de classes pour ce problème de classification (et donc aussi le nombre de neurones dans la couche de sortie)

- `epsilon`: a number $\varepsilon \in (0, 1)$ that is used to clip very small probabilities and very large probabilities when evaluating the cross-entropy loss. You don't need to change its value from the default 10^{-6}. - un nombre $\varepsilon \in (0, 1)$ utilisé pour arrondir les petites et grandes probabilités lors de l'évaluation de la fonction de coût d'entropie croisée. Vous ne devez pas changer sa valeur par défaut (10^{-6}).

- `lr`: the learning rate used to train the neural network with minibatch stochastic gradient descent (SGD). - le taux d'apprentissage utilisé pour entraîner le réseau de neurones avec la descente de gradient stochastique (SGD) minibatch.

- `batch_size`: the batch size for minibatch SGD. - la taille des batch pour SGD.

- `seed`: the random seed that ensures that two runs with the same seed yield the same results. - la graine aléatoire qui permet de s'assurer que vos résultats sont reproductibles.
- `activation`: a string describing the activation function. It can be `"relu"`, `"sigmoid"`, or `"tanh"`. We remind you of the three activation functions: - une chaîne de caractères décrivant la fonction d'activation à utiliser. Elle peut être `"relu"`, `"sigmoid"`, ou `"tanh"`. On vous rappelle les 3 fonctions d'activation:
  - $RELU(x) = \max(0, x)$
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
  - $\tanh(x) = \frac{e^{x}+e^{-x}}{e^{x}-e^{-x}}$

The `__init__` function is provided for you. In addition to loading the dataset and storing the parameters in class variables, this function initializes a dictionary of training logs, that contain information about the losses and accuracies on the training and validation sets during training. You will need to populate it during training.

La fonction `__init__` vous est donnée. En plus du chargement du jeu de données et l'initialisation des variables de classe, cette fonction initialise un dictionnaire de logs d'entraînement, qui contient des informations sur les coûts et les précisions sur les ensembles d'entraînement et de validation pendant l'entraînement. Vous devrez le remplir pendant l'entraînement.

1. Undergraduates 5 pts Graduates 5 pts

   Complete the `NN.initialize_weights` function. This function sets the random seed and creates a dictionary `self.weigths` (that also contains the biases). The input of this function is a list `dims` of size 2, containing the input dimension and the number of classes.

   As you know, it is necessary to randomly initialize the parameters of your neural network (trying to avoid symmetry and saturating neurons, and ideally so that the pre-activation lies in the bending region of the activation function so that the overall networks acts as a non linear function). You have to sample the weights of a layer from a uniform distribution in $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$, where n_c is the number of inputs for **this layer** (changing from one layer to the other). Biases should be initialized at 0.

You have to use the following naming convention: The keys of the dictionary are "W1", "b1", ..., "WL", "bL", where $L-1$ is the number of hidden layers. For example, if your network has one hidden layer, the keys should be the strings "W1", "b1", "W2" and "b2".

For simplicity, you can use Python's f-Strings to easily access the weights (e.g. `self.weights[f"b{layer_n}"]` if `layer_n` is a variable used to loop through the number of layers).

Complétez la fonction `NN.initialize_weights`. Cette fonction initalise la graine aléatoire et crée un dictionnaire des poids du réseau `self.weigths` (il contient aussi les biais). L'input de cette fonction est une liste `dims` de taille 2, contenant la dimension de l'input et le nombre de classes.

Comme vous le savez, il est nécessaire d'initialiser aléatoirement les poids de votre réseau pour casser les symétries. Vous devez tirer les poids aléatoirement de la distribution uniforme sur $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$, où n_c est le nombre d'inputs pour **cette couche** (changeant d'une couche à l'autre). Les biais doivent être initialisés à 0.

Vous devez utiliser la convention suivante: Les clés du dictionnaire sont "W1", "b1", ..., "WL", "bL", où $L-1$ est le nombre de couches cachées. Par exemple, si votre réseau a une seule couche cachée, les clés seraient les chaînes suivantes: "W1", "b1", "W2" et "b2".

Par simplicité, vous pouvez utiliser les f-Strings de Python pour accéder facilement aux poids (e.g. `self.weights[f"b{layer_n}"]` si `layer_n` est une variable utilisée pour parcourir les couches).

2. Undergraduates 6 pts Graduates 6 pts

In this question, you will need to implement the three possible activation functions. Each activation function takes as input two parameters: the variable to apply the function to, `x`, and a boolean `grad` that determines whether we are interested in the value of the function at `x` or the value of its derivative. **Make sure your functions work with number inputs, but also numpy arrays**.

Dans cette question, vous devez implémenter les trois fonctions d'activation possibles. Chaque fonction d'activation prend comme input deux paramètres: la variable sur laquelle on applique la fonction, `x`, et un booléen `grad` qui détermine si on est intéressés par la valeur de la fonction en `x` ou la valeur de sa dérivée. **Assurez-vous que vos**

fonctions marchent tout aussi bien avec les inputs scalaires qu'avec les inputs vectoriels (numpy array).

(a) Complete the `NN.relu` function, that returns for an input x: $RELU(x)$ when `grad` is set to `False`, and $RELU'(x)$ otherwise. - Complétez la fonction `NN.relu` qui retourne pour un input x: $RELU(x)$ quand `grad` est `False`, et $RELU'(x)$ sinon.

(b) Complete the `NN.sigmoid` function, that returns for an input x: $\sigma(x)$ when `grad` is set to `False`, and $\sigma'(x)$ otherwise. - Complétez la fonction `NN.sigmoid` qui retourne pour un input x: $\sigma(x)$ quand `grad` est `False`, et $\sigma'(x)$ sinon.

(c) Complete the `NN.tanh` function, that returns for an input x: $\tanh(x)$ when `grad` is set to `False`, and $\tanh'(x)$ otherwise. - Complétez la fonction `NN.tanh` qui retourne pour un input x: $\tanh(x)$ quand `grad` est `False`, et $\tanh'(x)$ sinon.

(d) Using the three functions you just implemented, complete the `NN.activation` function, that returns the right activation function, evaluated at its input x (with the variable `grad` taken into account). Remember that the activation function used is stored in the `self.activation_str` variable. - En utilisant les trois fonctions que vous venez d'implémenter, complétez la fonction `NN.activation` qui retourne la bonne fonction d'activation, évaluée en son input x (en prenant en compte la variable `grad`). Rappelez-vous que la fonction d'activation utilisée est dans la variable `self.activation_str`.

3. Undergraduates 4 pts Graduates 4 pts Write the `NN.softmax` function. The function takes as input a numpy array `x` and should return an array (of the same size as `x`) corresponding to $softmax(\mathbf{x})$. **Your function should also work for an input representing a mini-batch of examples, i.e. a** $batch\_size \times n\_dimensions$ **matrix.**

   You will need to compute a **numerically stable softmax**, i.e. you should use the fact that $softmax(\mathbf{x}) = softmax(\mathbf{x} - c)$ for any constant c, where $\mathbf{x} - c = (\mathbf{x}_1 - c, \ldots, \mathbf{x}_n - c)$. A usual choice for c to avoid overflow errors is $\max(\mathbf{x})$.

   Implémentez la fonction `NN.softmax`. La fonction prend comme input un vecteur (numpy array) `x` et doit retourner un vecteur de la même taille correspondant à $softmax(\mathbf{x})$. **Votre fonction doit aussi fonc-**

tion pour un input représentant un mini-batch d'exemples, i.e. une matrice $batch\_size \times n\_dimensions$**.**

Vous devrez calculer un **softmax numériquement stable**, i.e. vous devrez utiliser le fait que $softmax(\mathbf{x}) = softmax(\mathbf{x} - c)$ pour toute constante c, où $\mathbf{x} - c = (\mathbf{x}_1 - c, \ldots, \mathbf{x}_n - c)$. Un choix commun pour c pour éviter les erreurs de dépassement (overflow) est $\max(\mathbf{x})$.

4. Undergraduates 10 pts Graduates 10 pts Complete the `NN.forward` function. This function propagates its input x through the layers of the neural network. The results of the forward propagation should be stored in the `cache` dictionary. The keys of the dictionary should be called "Z0", "A1", "Z1", ..., "AL", "ZL", where $L - 1$ is the number of hidden layers. For simplicity, you can use Python's f-Strings to easily access the weights (e.g. `cache[f"Z{layer_n}"]`) if `layer_n` is a variable used to loop through the number of layers.

   `cache["Ai"]` should store the pre-activation at layer i (i.e. before applying the activation function). `cache["Zi"]` should store the activation at layer i. Following this logic, `cache["ZL"]` should be a vector of probabilities.

   Remember that `NN.forward` should return the whole `cache` dictionary, and not only the output of the final layer, given that `cache` will be used for backward propagation as well.

   Your function should also work for an input representing a mini-batch of examples, i.e. a $batch\_size \times n\_input\_dimensions$ **matrix.** In this case for instance, `cache["ZL"]` should be a matrix of size $batch\_size \times n\_classes$.

   Complétez la fonction `NN.forward`. Cette fonction propage l'input x à travers les couches du réseau de neurones. Les résultats de cette propagation doivent être enregistrés dans le dictionnaire `cache`. Les clés de ce dictionnaire doivent être appelées "Z0", "A1", "Z1", ..., "AL", "ZL", où $L - 1$ est le nombre de couches cachées. Par simplicité, vous pouvez utiliser les f-Strings de Python pour accéder facilement aux poids (e.g. `cache[f"Z{layer_n}"]` si `layer_n` est une variable utilisée pour parcourir les couches).

   `cache["Ai"]` doit enregistrer les préactivations à la couche i (i.e. avant d'appliquer la non-linéarité). `cache["Zi"]` doit enregistrer les activations à la couche i. Selon cette logique, `cache["ZL"]` doit être un vecteur de probabilités.

Votre fonction doit aussi fonctionner pour un inputs représentant un mini-batch d'exemples, i.e. une matrice de taille *batch_size*×*n_input_dimensions*. Dans ce cas par exemples, `cache["ZL"]` doit être une matrice de taille *batch_size* × *n_classes*.

5. Undergraduates 9 pts Graduates 9 pts Complete the `NN.backward` function. This function takes as input the `cache` evaluated using the `NN.forward` function on a mini-batch, and the `labels` of the examples in the minibatch as a matrix of size *batch_size*×*n_classes*. Note that `labels` are the one-hot encodings of the labels (c.f. question 7). The function should populate the `grads` dictionary and return it. `grads` contains the gradients of the loss (evaluated on the current mini-batch) with respect to the network parameters and the activations and pre-activations. `cache` should have $4L-1$ entries, where $L-1$ is the number of hidden layers. The keys of the dictionary should be called "dAL", "dWL", "dbL", "dZ(L-1)", "dA(L-1)", "dW(L-1)", ..., "dZ1", "dA1", "dW1", "db1".

   Remember that `NN.backward` should return the whole `grad` dictionary, and not only the gradients with respect to the network parameters.

   For example, if your network has 1 hidden layer of 20 neurons, `grads["dA2"]` should be a matrix of size *batch_size* × *n_classes*, `grads["dW2"]` a matrix of size 20×*n_classes*, `grads["db2"]` a vector of size *n_classes*, `grads["dZ1"]` and `grads["dA1"]` two matrices of size *batch_size*×20, `grads["dW1"]` a matrix of the same size as `self.weights["W1"]`, and `grads["db1"]` a vector of the same size as `self.weights["b1"]` (i.e. 20).

   Your function should work for a `cache` and `labels` inputs representing a mini-batch of examples.

   Hint: You can first evaluate `grads[f"dAself.n_hidden+1"]`, then loop through the layers in reverse, from the output layer to the first hidden layer.

   Complétez la fonction `NN.backward`. Cette fonction prend comme input le dictionnaire `cache` évalué avec la fonction `NN.forward` sur un mini-batch, et les `labels` des exemples du minibatch sous forme d'une matrice de taille *batch_size* × *n_classes*. Notez que `labels` sont les one-hot encodings des étiquettes (c.f. question 7). La fonction doit remplir le dictionnnaire `grads` et le retourner. `grads` contient les gradients de la fonction coût (évaluée sur le mini-batch actuel)

par rapport aux paramètres du réseau et des activations et des pré-activations. `cache` doit avoir $4L-1$ entrées, où $L-1$ est le nombre de couches cachées. Les clés du dictionnaire doivent être appelées "dAL", "dWL", "dbL", "dZ(L-1)", "dA(L-1)", "dW(L-1)", ..., "dZ1", "dA1", "dW1", "db1".

On rappelle que `NN.backward` doit retourner le dictionnaire `grad` en entier, et non pas seulement les gradients par rapport aux paramètres du réseau.

Par exemple, si votre réseau a 1 couche cachée de 20 neurones, `grads["dA2"]` doit être une matrice de taille $batch\_size \times n\_classes$, `grads["dW2"]` une matrice de taille $20 \times n\_classes$, `grads["db2"]` un vecteur de taille $n\_classes$, `grads["dZ1"]` et `grads["dA1"]` deux matrices de taille $batch\_size \times 20$, `grads["dW1"]` une matrice de la même taille que `self.weights["W1"]`, et `grads["db1"]` un vecteur de la même taille que `self.weights["b1"]` (i.e. 20).

Votre fonction doit aussi fonctionner pour des inputs `cache` et `labels` représentant un mini-batch d'exemples.

Indication: Vous pouvez d'abord évaluer `grads[f"dAself.n_hidden+1"]`, puis parcourir les couches en partant de la couche de sortie à la couche d'entrée.

6. Undergraduates 4 pts Graduates 4 pts Complete the `NN.update` function that updates the network parameters W and b using the gradients stored in the input `grads`. You will need to loop through all the elements of the `self.weights` dictionary in order to update them. The `grads` input of this function contains the gradients of the loss function evaluated on the current mini-batch with the `NN.backward` function.

   Note that this is the function you are going to use in your training function, when you are going to loop through mini-batches. Remember that the learning rate is stored in the `self.lr` variable.

   Complétez la fonction `NN.update`. Cette fonction met à jour les paramètres W et b en utilisant les gradients enregistrés dans l'input `grads`. Vous devrez parcourir tous les éléments du dictionnaire `self.weights` pour les mettre à jour. L'input `grads` de cette fonction contient les gradients de la fonction coût évalués sur le mini-batch actuel avec la fonction `NN.backward`.

   Notez que cette fonction est celle que vous utiliserez dans votre fonction d'entraînement, quand vous parcourrez les mini-batch. Nous vous

rappelons que le taux d'apprentissage est enregistré dans la variable `self.lr`.

7. Undergraduates 4 pts Graduates 4 pts

   (a) Write the `NN.one_hot` function. This function should transform an array `y` of size *batch_size* containing values from 0 to *n_classes* − 1 into a matrix of size *batch_size* × *n_classes* containing the one-hot encodings of the true labels `y`.

   For example, if the number of classes is 5, then `one_hot(np.array([1, 0, 3, 4]))` should return the following matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

   Implémentez la fonction `NN.one_hot`. Cette fonction doit transformer un vecteur `y` de taille *batch_size* contenant des valeurs entières entre 0 to *n_classes*−1 en une matrice de taille *batch_size*× *n_classes* contenant les one-hot encodings des vraies étiquettes `y`.

   Par exemple, si le nombre de classes est 5, alors `one_hot(np.array([1, 0, 3, 4]))` devrait retourner la matrice suivante:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

   (b) Complete the `NN.loss` function that takes as input a matrix `predictions` of size *batch_size*×*n_classes* containing the probabilities that the `NN.forward` function evaluate, and an array`labels` of size *batch_size* × *n_classes* containing the one-hot encodings of the true labels. The function should return the cross-entropy loss.

   To avoid numerical errors, we pre-process the evaluated probabilities by clipping very small values (i.e. close to 0) to a fixed value ε and very large values (i.e. close to 1) to $1 - \varepsilon$. **You should not modify this part of the code**.

Complétez la fonction `NN.loss` qui prend comme input une matrice `predictions` de taille $batch\_size \times n\_classes$ contenant les probabilités évaluées par la fonction `NN.forward`, et un vecteur `labels` de taille $batch\_size \times n\_classes$ contenant les one-hot encodings des vraies étiquettes. La fonction doit retourner l'entropie croisée (fonction coût).

Pour éviter les erreurs numériques, nous arrondissons les petites valeurs des probabilités à une valeur fixe ε et les grandes valeurs à $1 - \varepsilon$. **Ne modifiez pas cette partie du code**.

8. Undergraduates 5 pts Graduates 5 pts For this question, you should use the previous functions you implemented.

   Complete the `NN.train_loop` function. The only parameter of this function is `n_epochs`, that represents the number of full passes on the training dataset.

   For each epoch, you will need to loop through the mini-batches, and for each mini-batch, you will need to update the network parameters using the gradients evaluated using the current mini-batch.

   In addition to performing the updates, the `NN.train_loop` function keeps a training log in the `self.train_logs` dictionary, containing the training/validation accuracies and losses after each epoch. This part of the code is already provided for you and you should not change it. These logs will be useful for you to plot learning curves.

   Note that no data shuffling is done here. You should not add any.

   Pour cette question, vous devrez utiliser les fonctions précédentes que vous avez implémentées.

   Complétez la fonction `NN.train_loop`. Le seul paramètre de cette fonction est `n_epochs`, qui représente le nombre de fois qu'on passe sur le jeu d'entraînement en entier.

   Pour chaque epoch, vous devrez parcourir les mini-batch, et pour chaque mini-batch, vous devrez mettre à jour les paramètres du réseau en utilisant les gradients évalués en utilisant le mini-batch actuel.

   En plus de faire les mises-à-jour, la fonction `NN.train_loop` enregistre les logs d'entraînement dans le dictionnaire `self.train_logs`, qui contient les précisions d'entraînement de validation, en plus des coûts (losses) sur les jeux d'entraînement de validation, à la fin de chaque epoch. Cette partie du code vous est déjà donnée, et vous

ne devrez pas la changer. Ces logs vous seront utiles pour tracer les courbes d'apprentissage.

Notez qu'aucun mélange de données (shuffling) n'est fait. Et vous ne devrez pas en faire!

9. Undergraduates 3 pts Graduates 3 pts Complete the `NN.evaluate` function. Your function should return the average loss and accuracy on the test set. Remember that, similar to `NN.train_loop`, you can use the `NN.compute_loss_and_accuracy` function (that is already provided for you).

   Complétez la fonction `NN.evaluate`. Cette fonction doit retourner le coût moyen et la précision moyenne sur l'ensemble de test. Comme pour `NN.train_loop`, vous pouvez utiliser la fonction `NN.compute_loss_and_accuracy` (qui vous est déjà donnée).

2 Trying your implementation on the CIFAR-10 dataset - Essayez votre implémentation sur le jeu de données CIFAR-10 [Undergraduates 20 bonus pts Graduates 20 pts]

In this part of the homework, you are going to test your implementation on the CIFAR-10 dataset.

You **should** download the dataset from here.

The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. We are going to use 49000 images for training, 1000 images for validation, and 10000 images for testing. Each image, originally represented with a $3 \times 32 \times 32$ matrix, has been flattened, meaning that each example is represented with an array of size 3072. The file you downloaded is a pickle file, that can be processed with the code provided to you in `NN.__init__`. If you initialize your `NN` class by providing the path to the pickle file you downloaded in the parameter `datapath`, you will end up with the variables `self.train, self.valid, self.test` containing the dataset. Each of these three variables is a tuple. For example, `self.valid[0]` is a matrix of size 1000×3072, and `self.valid[1]` is an array of size 1000 containing the labels of the 1000 validation set images, from 0 to 9. Please double check that the six arrays have the right size!

The answers to the questions of this section, along with the required figures, should be written in the report you will submit to Gradescope for the theoretical part of the homework.

Dans cette partie du devoir, vous allez tester votre implémentation sur le jeu de données CIFAR-10.

Vous **devez** télécharger le jeu de données depuis ce lien.

Le dataset CIFAR-10 contient 60000 images colorées de taille 32×32 dans 10 classes, avec 6000 images par classe. Nous allons utiliser 49000 images pour l'entraînement, 1000 images pour la validation, 10000 images pour l'ensemble de test. Chaque image, à la base représentée par une matrice de taille $3 \times 32 \times 32$ a été aplatie, ç-à-d que chaque exemple est représenté par un vecteur de taille 3072. Le fichier que vous avew téléchargé est un fichier pickle, qui peut être lu et transformé avec le code que nous avons mis dans `NN.__init__`. Si vous initialisez votre classe `NN` en donnant le chemin vers le fichier pickle que vous avez téléchargé dans le paramètre`datapath`, vous finirez avec des variables `self.train, self.valid, self.test` contenant le jeu de données. Chacune de ces trois variables est un tuple. Par example, `self.valid[0]` est une matrice de taille 1000×3072, et `self.valid[1]` est un vecteur de taille 1000 contenat les étiquettes des 1000 images de l'ensemble de validation, de 0 à 9. Vérifiez que les 6 vecteurs/matrices ont la bonne taille!

Les réponses aux questions de cette partie, en plus des figures demandées doivent être écrites dans le rapport que vous soumettrez sur Gradescope pour la partie théorique.

1. Undergraduates 5 bonus pts Graduates 5 pts Train a neural network with 2 hidden layers, of size 512 and 256 respectively on the CIFAR-10 dataset, for 50 epochs. Use a learning rate of 0.003, and a batch size of 100. Use the RELU activation function. For reproducibility purposes, **please set the random seed to 0**.

   Include in your report the two following figures:

   - A figure containing the evolution of both the training and validation accuracies during training
   - A figure containing the evolution of both the training and validation losses during training.

For verification purposes, you should get a training accuracy larger than 0.8 at the 50th epoch.

Entraînez un réseau de neurones à deux couches cachées, de tailles 512 and 256 respectivement sur CIFAR-10, pendant 50 epochs. Utilisez un taux d'apprentissage de 0.003, un batch size de 100. Utilisez la fonction d'activation RELU. Pour que vos résultats soient reproductibles, **assignez 0 à la graine aléatoire (seed)**.

Incluez dans votre rapport les figures suivantes:

- Une figure représentant l'évolution des précisions sur les ensembles d'entraînement et de validation pendant les 50 epochs.
- Une figure représentant l'évolution des coûts sur les ensembles d'entraînement et de validation pendant les 50 epochs.

Assurez-vous d'obtenir une précision sur l'ensemble d'entraînement d'au moins 0.8 à l'epoch 50.

2. Undergraduates 5 bonus pts Graduates 5 pts

   (a) Explain in no more than two sentences why the performances on the validation set are not as good as on the training set.
   Expliquez en une ou deux phrases pourquoi les performances sur l'ensemble de validation ne sont pas aussi bonnes que sur l'ensemble d'entraînement.

   (b) How could the performance gap be lowered? Make two propositions in the form of short bullet points.
   Comment réduire ce gap de performance? Faites deux propositions sous la forme de bullet points.

   (c) How many learnable parameters (scalars) does the previous neural network have ?
   Combien de paramètres scalaires votre réseau de neurone doit-il apprendre?

3. Undergraduates 5 bonus pts Graduates 5 pts In this question, we are going to compare how the depth and width of the neural network affects the performances of this classification problem. For this purpose, we consider a deep neural network with equal number of neurons amongst the hidden layers starting from the second one. In order to have a fair comparison, we need to ensure that this new neural network has approximately the same number of learnable parameters as

the previous one. We consider a neural network with *n_hidden* hidden layers. The first hidden layer has 512 neurons, and all the remaining ones have 120 neurons each.

- Find *n_hidden* such that the new network has a number of parameters that is as close as possible to the number of parameters of the initial neural network. Include this number, along with your reasoning, in the report.
- Train the new network with the same parameters as in the first question (same learning rate, batch size, activation function, and same seed). Include in your report two figures containing the training/validation accuracies and losses, similar to the first question.

Dans cette question, nous allons comparer comment la profondeur et la largeur du réseau de neurones affectent les performances sur cette tache de classification. Pour cela, on considère un réseau profond avec un nombre égal dans les couches cachées, à partir de la seconde couche. Pour avoir une comparaison juste, nous devons nous assurer que ce nouveau réseau de neurones a le même nombre de paramètres à apprendre; à peu près. On considère donc un réseau avec *n_hidden* couches cachées. La première couche contient 512 neurones, et les couches cachées restantes contiennent 120 neurones chacune.

- Trouvez *n_hidden* pour que le nouveau réseau ait un nombre de paramètres aussi proche que possible que le nombre de paramètres du réseau initial. Incluez ce nombre et le raisonnement qui vous a permis d'y arriver dans votre rapport.
- Entraînez le nouveau réseau de neurones avec les mêmes paramètres qu'à la première question (même taux d'apprentissage, batch size, fonction d'activation, et même graine aléatoire). Incluez dans votre rapport deux figures contenant les précisions et les coûts sur les ensembles d'entraînement et de validation, de façon similaire à la première question.

4. Undergraduates 5 bonus pts Graduates 5 pts

- Why isn't it sufficient to visually compare the figures of the previous question to the figures of the first question to decide which neural network performs best ?

- Train both neural networks for 50 epochs, using 3 different seeds of your choice (you need to report them), with the same parameters as in the first question (you thus have to train 6 networks). Include in your report one figure containing the average training and validation accuracies of both neural networks during training, along with error bars corresponding to the standard deviations you obtain in the 3 different runs multiplied by a factor of your choice **that you need to specify in your report**

- Pourquoi n'est-il pas suffisant de comparer visuellement les figures de la question précédente aux figures de la première question pour décider quel réseau de neurones a les meilleures performances.
- Entraînez les deux réseaux de neurones avec les mêmes paramètres, pendant 50 epochs, mais en utilisant 3 graines aléatoires différentes de votre choix (donc vous devez entraîner 6 réseaux). Vous devrez inclure dans votre rapport les 3 graines que vous avez choisies. Incluez dans votre rapport une figure contenant les précisions moyennes sur les ensembles d'entraînement et de validation, avec des barres d'erreur correspondant aux écarts types que vous avez obtenus avec les 3 différentes graines, multipliés par un facteur de votre choix **que vous devrez spécifier dans votre rapport**.