TP1 : implémentation d'un réseau multi-couche

Dans ce TP, nous allons exploiter votre modèle de perceptron 'optimisé' pour créer et tester un réseau multi-couche, et augmenter le taux de réussite du réseau.

I préparation

- Commencez par dupliquer votre projet *2_perceptronV2* et renommez la copie *3_MultiLayer*

- Renommez votre classe *'Perceptron'* en *'Layer'* (*File → rename*), et effectuez les corrections si nécessaires. Vérifiez que le projet est toujours fonctionnel.

II Modification de la classe Layer

- Comme on l'a vu, un réseau multi-couche ne peut pas démarrer avec des poids à 0. Dans le constructeur de *'Layer'*, initialisez aléatoirement les poids avec des valeurs entre -0,005 et 0,005

  Note : fonction *random* en java (retourne un double) : `Math.random()`
  Cette fonction retourne un nombre aléatoire entre 0 et 1.

Il faut maintenant modifier la fonction d'apprentissage *learn*. En effet, on va séparer le calcul des deltas et la mise à jour des poids. D'autre part, il faut pouvoir calculer les deltas de la couche de sortie et les deltas des couches cachées.

- Remplacez la fonction *learn* par les fonctions suivantes :
  - `setLastDeltas(int[] output)` qui calcule les deltas dans le cas d'une couche de sortie. C'est également la couche de sortie qui calculera la somme des valeurs absolues des deltas.
  - `setDeltas(Layer next)` qui calcule le delta dans le cas d'une couche cachée. Le paramètre *next* sera un pointeur vers la couche suivante permettant de récupérer ses poids et deltas.
  - `learn(float[] input)` qui met effectivement à jour les poids à partir des deltas calculés

III préparation du réseau

Nous allons maintenant créer un réseau avec deux couches

- Remplacez le pointeur du *Layer* (initialement du *Perceptron*) et remplacez-le par un vecteur de *Layer*

```
public Layer[] layers;
```

- Initialisez dans *Main( )* ce vecteur à une taille de 2 et initialisez les Layers comme suit :

```
layers=new Layer[2];
layers[0]=new Layer(size_x*size_y, nb_layer1);
layers[1]=new Layer(nb_layer1, nb_layer2);
```

(vous initialiserez deux variables globales *nb_layer1* et *nb_layer2* à 10)

- pour l'apprentissage, ordonnancez les calculs de la façon suivante :

  calcul de la couche 0 (à partir du vecteur d'entrées)
  calcul de la couche 1 (à partir des résultats de la couche 0)

  calcul dernier delta sur la couche 1
  calcul du delta de la couche 0

  mise à jour poids de la couche 0 à partir du vecteur d'entrée
  mise à jour poids de la couche1 à partir du vecteur de résultats de la couche 0

- pour la partie test, on utilise seulement le calcul des valeurs de sorties :

  calcul de la couche 0 (à partir du vecteur d'entrées)
  calcul de la couche 1 (à partir des résultats de la couche 0)

- corrigez le calcul du delta moyen et le calcul de l'erreur (on utilise la couche de sortie).

```
sumdelta+=layers[1].sum_delta;
```

- corrigez le calcul du nombre d'erreurs (toujours avec la couche de sortie)

- corrigez l'afficheur (DisplayPanel) pour afficher les poids de la première couche. Ajoutez le code suivant pour afficher les poids de la seconde couche :

```
for (int n=0;n<main.nb_layer2;n++){
       for (int i=0;i<main.nb_layer1;i++){
          val=(int)(main.layers[1].weights[n][i]*50)+128;
          if (val<0) val=0;
          if (val>255) val=255;
          g.setColor(new Color(val,val,val));
          g.fillRect(680+6*i, 10+10*n, 5, 5);
     }
}
```

Le réseau est maintenant fonctionnel !

- Testez le réseau : on n'observe rien et le nombre d'erreur est énorme !
Augmentez le nombre d'epoch à 1000 (retirez la temporisation) et augmentez le learning rate à 0,05.
Observez maintenant les structures qui apparaissent sur les poids des neurones de la première couche.

Comparez le résultat avec celui du perceptron. Qu'observez-vous ?

IV Nouvelle base d'apprentissage

Le fait d'ajouter des neurones implique une augmentation du nombre de poids à modifier et donc du nombre de tests à effectuer pour entraîner les neurones.

Nous allons effectuer l'apprentissage sur l'ensemble de la base MNIST complète. Celle-ci comporte une base de 60.000 images de chiffres manuscrits pour l'apprentissage, et une base de 10.000 images pour le test du réseau. Ces images font 28x28 pixels.

- téléchargez les 4 fichiers sur le site https://github.com/fgnt/mnist et décompressez-les dans un sous-dossier de votre répertoire de travail (celui désigné par 'PATH') que vous appellerez 'Numbers'.

Contrairement à la version précédente de notre réseau, nous allons charger l'ensemble des images dans une matrice (qui aura une taille de 60.000 x 784), et les résultats attendus dans un vecteur, avant de procéder à l'apprentissage

- dupliquez le projet 3_MultiLayer et renommez la copie 3_MultiLayer_MNIST

- Remplacez le vecteur *img* et le *bufferedImage* par les deux matrices suivantes, et mettez à jour *size_x* et *size_y* pour les mettre à 28. Vous pouvez supprimer les variables *offset_x*, *offset_y* et *nb_samples*. Ajoutez également une variable globale *test* qui servira d'index pour parcourir les images.

```
public static int size_x=28;
public static int size_y=28;

public float[][] matrixImages;
public int[] matrixLabels;

public int test=0;
```

Les fichiers de la base MNIST sont dans un format très particulier : les pixels des images sont encodés par des octets, mis les uns à la suite des autres.

Le fichier *readFunction* sur le github vous donne une fonction permettant d'initialiser les matrices et de lire et enregistrer à la fois les images dans la matrice et les résultats attendus dans le vecteur dédié.

- intégrez cette fonction dans la classe *Main*, et utilisez-là (avant l'initialisation du *display*) pour charger les fichiers `"Numbers/train-images.idx3-ubyte"` et `"Numbers/train-labels.idx1-ubyte"`

- Modifiez la boucle de test : nous n'avons en effet besoin que d'une seule boucle pour parcourir les 60 000 images de la matrice de test (au lieu des deux pour parcourir l'image d'échantillons utilisée précédemment). Cette boucle utilisera la variable globale *test* comme itérateur.
/!\ ne modifiez pas la boucle des epochs.

- Supprimez la lecture de l'image : l'image envoyée au réseau peut maintenant être récupérée depuis la matrice matrixImages :

```
matrixImages[test]
```

- Modifiez également le calcul du vecteur de résultats en vous servant du vecteur matrixLabel :

```
matrixLabels[test]
```

- Modifiez ensuite la partie test. Vous devrez charger `"Numbers/t10k-images.idx3-ubyte"` et `"Numbers/t10k-labels.idx1-ubyte"` avec la fonction de lecture juste après la partie apprentissage.

- Modifiez la double boucle de test par une boucle unique (rappel : il y a 10 000 images de tests). Comme pour l'apprentissage, retirer la partie lecture d'image et mettez à jour le calcul des erreurs.

- Mettez à jours l'affichage de l'image de test dans le DisplayPanel :

```
int val=0;
for (int i=0;i<Main.size_x;i++){
 for (int j=0;j<Main.size_y;j++){
          val=(int)(main.matrixImages[main.test][i+Main.size_x*j]*255);
          g.setColor(new Color(val,val,val));
          g.fillRect(10+5*i, 10+5*j, 5, 5);
 }
}

for (int n=0;n<main.nb_layer1;n++){
 for (int i=0;i<Main.size_x;i++){
          for (int j=0;j<Main.size_y;j++){
```

- modifiez l'affichage final du taux d'erreur avec la ligne suivante :

```
System.out.println("total : "+errors+" erreurs -> "+((float)errors/100)+"%");
```

- Testez votre réseau avec la nouvelle base d'apprentissage et de test (baissez le nombre d'epoch à 50) Observez les formes qui apparaissent au niveau des poids des neurones de la première couche.

- Dupliquez le projet 2_Perceptron et renommez la copie 2_Perceptron_MNIST.
Appliquez les mêmes changements (à coup de copier-coller ou en utilisant prudemment la fonction '*compare with*' d'Eclipse) pour utiliser la base MNIST avec le perceptron. Comparez ensuite les résultats avec le multi-couche.

- Avec le réseau multi-couches, testez différents nombres de neurones pour la première couche : 15, 20, 25, 30. Notez les taux d'erreurs pour chaque test.

V Pour aller plus loin

Marre des chiffres ? Nous allons tester notre réseau sur un autre jeu de données : le dataset *MNIST Fashion*. Il s'agit d'une base d'image de dix types de vêtements, que le réseau va devoir apprendre à reconnaître.

Chaque type possède un label, de 0 à 9, ce qui va nous permettre de tester directement notre réseau.

- Télécharger la base d'image sur :
https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion
téléchargez les quatre fichiers et décompressez-les dans un sous-dossier de votre répertoire de travail que vous appellerez 'Fashion'.

- Dupliquez le projet 3_MultiLayer_MNIST et appelez la copie 3_MultiLayer_MNIST_Fashion.

- Modifiez les paramètres des deux appels à la fonction setData pour les remplacer par les fichiers :
Pour l'apprentissage :
`"Fashion/train-images-idx3-ubyte"` et `"Fashion/train-labels-idx1-ubyte"`
Pour le test :
`"Fashion/t10k-images-idx3-ubyte"` et `"fashion/t10k-labels-idx1-ubyte"`

- Testez votre réseau avec une première couche à 20 neurones et notez le taux d'erreur.

La reconnaissance d'image s'avère un problème plus complexe que la reconnaissance de chiffres. On va complexifier le réseau.
- Ajoutez une couche cachée supplémentaire à votre réseau. Effectuez les modifications nécessaires.

Essayez avec un réseau avec une première couche à 20 neurones, une seconde à 15 et la couche de sortie à 10 neurones.

Testez différentes configurations : pour chaque test, notez :
- la configuration du réseau (nombre de couches, nombre de neurone par couche)
- le learning rate
- le nombre d'epoch
- le taux d'erreur
- les valeurs min et max pour l'initialisation des poids.

Vous limiterez le nombre total de neurone du réseau à 100. Évitez de mettre le feu à votre poste !