GPA759 - Laboratoire 2

Chargé de laboratoire : Marc-André Carbonneau

INTRODUCTION

Un réseau à rétropropagation du signal d'erreur a été étudié comme une généralisation du perceptron. Il est souvent d'ailleurs appelé le perceptron multicouche. Il est composé d'une première couche de neurones d'entrée, d'une ou plusieurs couches de neurones cachés, reliés à la couche précédente par des connexions modifiables par apprentissage, et d'une couche de sortie, qui est la seule en contact avec la couche de supervision. Le but de ce laboratoire est de mettre au point un réseau de neurones qui décide automatiquement si un pixel dans une image appartient ou non au contour d'un objet. Autrement dit, il s'agit de construire un réseau de neurones multicouche qui permettra de transformer une image de niveaux de gris en une image binaire. Cette opération devrait pouvoir mettre en évidence la structure de l'image et en extraire les primitives qui nous permettent d'en différencier le contenu au moyen des contours et arêtes présents dans l'image. Nous procéderons à un apprentissage supervisé par l'exemple dans lequel le réseau sera entraîné pour déceler dans des imagettes la présence d'une arête au centre de l'imagette.

PRÉSENTATION DU PROBLÈME

Concevoir un robot capable de se déplacer de façon autonome et de localiser et reconnaître un objet dans la scène, analyser les images satellite pour prévoir les secousses sismiques, identifier les ventricules gauche et droite du cœur, ou détecter les fissures sur la chaussée d'une route ne sont que quelques exemples impliquant l'extraction de l'information utile d'une image. La Figure 1 montre une image d'éclairement lumineux codée en niveaux de gris (a) et l'information condensée qu'on peut extraire de cette image (b). L'information condensée ne contient que les primitives de l'image, constituées dans ce cas des contours des divers objets présents dans la scène. L'analyse des primitives d'une image permet très souvent d'identifier et de localiser les divers objets présents sur l'image. Un ordinateur qui chercherait, par exemple, à localiser l'homme dans l'image de la fig. 1 (a) devrait analyser 256×256 = 65 536 pixels codés sur 8 bits ou 256 niveaux de gris, ce qui représente une quantité prohibitive d'information à traiter. Pour accélérer le traitement, on doit obtenir une représentation compacte de l'image qui contienne l'information la plus pertinente. Ainsi, on reconnaît aisément les principaux éléments de la fig 1 (b), où ne figurent que les arêtes de l'image originale. On réussit donc à décrire le contenu de l'image avec les 17 167 pixels noirs, soit 26,2% de l'information initiale. De plus, chaque pixel de l'image des arêtes est codé sur un seul bit (noir ou blanc), ce qui représente une nette diminution par rapport aux 256 niveaux de gris codés sur 8 bits pour chacun des pixels de l'image d'éclairement.

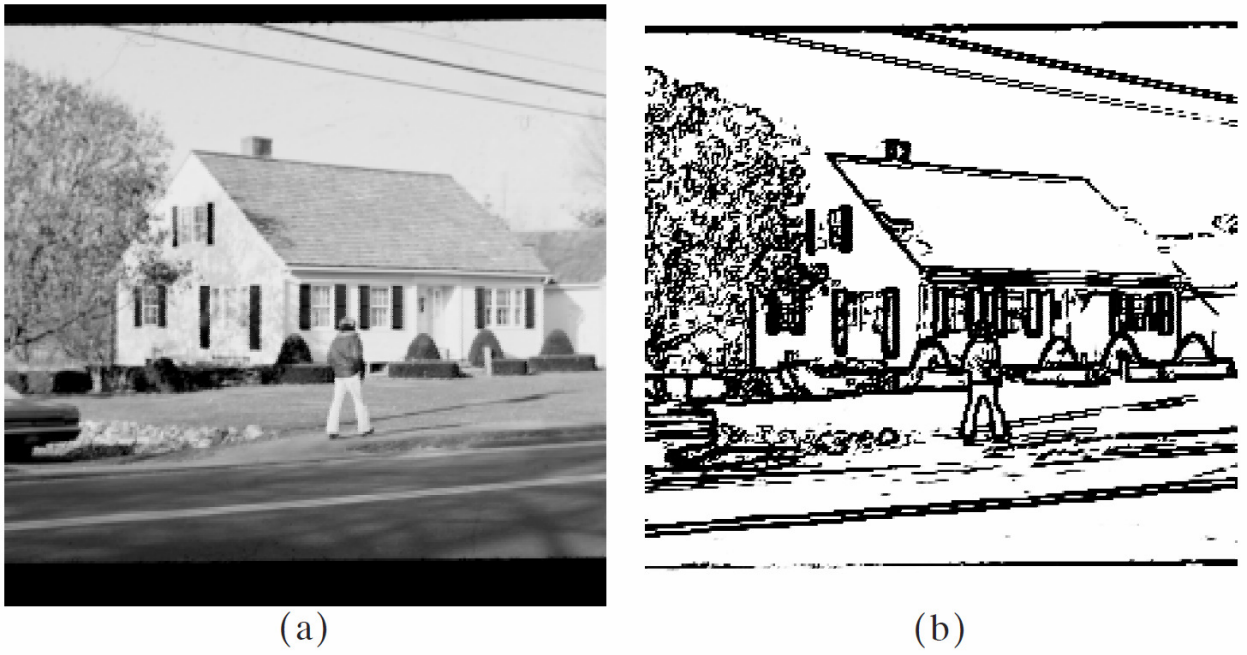

Figure 1 - L'extraction de l'information utile d'une image. (a) Image de niveaux de gris, (b) image binaire des arêtes détectées dans (a).

Principe d'opération

En pratique, il serait peu utile de fournir l'image complète en une seule passe au réseau : selon ce scénario, pour convertir par exemple une image de 256 × 256 pixels, il faudrait que la couche d'entrée du réseau comporte autant de neurones. Il serait alors difficile de convertir une image de taille différente, sans compter le temps d'apprentissage qui serait extrêmement long. Pour pallier ces inconvénients, nous choisissons de déplacer progressivement une fenêtre de petite taille sur l'image d'éclairement; les valeurs des pixels dans cette fenêtre seront fournies à l'entrée du réseau, qui devra indiquer si le pixel central appartient ou non à une arête.

Cette stratégie est illustrée à la Figure 2 La « rétine » du réseau est déplacée successivement sur chaque pixel de l'image d'entrée, de gauche à droite et de haut en bas. Pour chaque position de cette fenêtre, le réseau classifiera le pixel au centre du masque; la classe d'appartenance indiquée par le réseau sera alors attribuée au pixel correspondant de l'image de sortie. Par convention, la sortie du réseau sera 1 pour un pixel appartenant à une arête, et 0 dans le cas contraire.

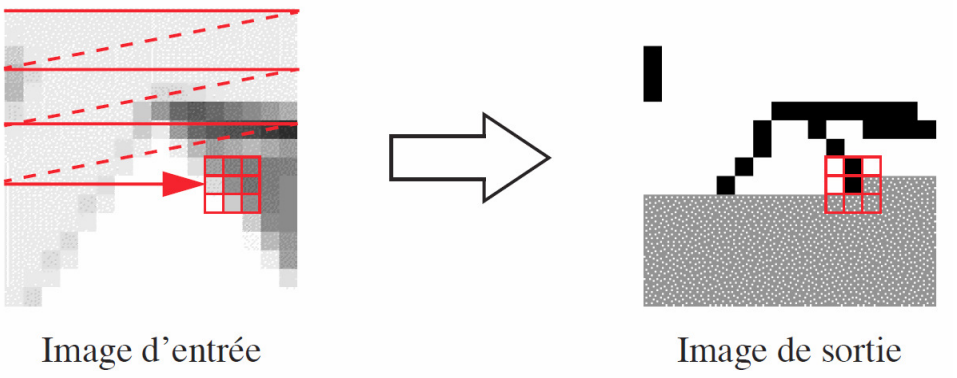

Figure 2 - Balayage de l'image par un réseau de neurones. La partie grise de l'image de sortie correspond aux endroits de l'image qui n'ont pas encore été analysés par le réseau.

Architecture du réseau de neurones

On ne dispose pas encore de règles formelles pour définir l'architecture exacte d'un réseau perceptron multicouche. C'est à l'utilisateur de trouver l'architecture la plus adéquate. L'application projetée impose cependant certaines contraintes :

• La couche de sortie ne requiert qu'un seul neurone, puisqu'on veut seulement classifier les pixels en deux catégories, contour ou non-contour.

• La couche d'entrée doit contenir neurones, avec n impair et n ≥ 3. Ceci garantit qu'il existe un pixel au centre du champ récepteur, comme dans le masque illustré à la Figure 2. C'est pour ce pixel central que le réseau détermine s'il s'agit ou non d'un pixel de contour à partir des pixels voisins délimités par le champ récepteur formé par le masque.

Vous devrez choisir vous-même le nombre de neurones sur la couche cachée.

VUE D'ENSEMBLE DU SYSTÈME

Lors de la première partie du laboratoire, nous construirons, dans Matlab, un réseau MLP. C'est ce réseau qui sera entrainé, par l'exemple, à extraire les contours d'une image. Tout d'abord, les images au format .tif seront transformées en matrices et en vecteurs d'entrainement. Ensuite, une base d'apprentissage sera constituée à partir des vecteurs contenus dans les fichiers .vx et .vd. Seulement certains vecteurs seront retenus pour participer à l'entrainement. En utilisant les vecteurs sélectionnés, nous entrainerons notre réseau MLP. Ce réseau sera ensuite utilisé pour extraire les contours d'une image qu'il n'aura jamais vue. Le processus complet est illustré à la Figure 3.

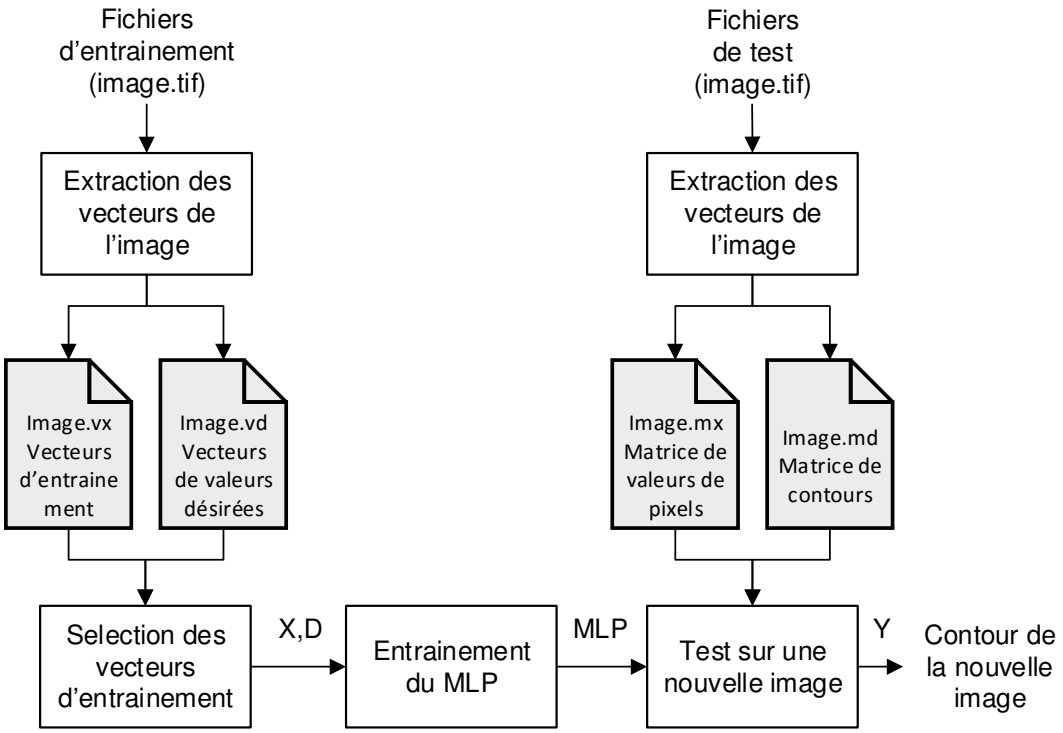

Figure 3 - Vue d'ensemble du système

Extraction des vecteurs et matrices

Pour chaque image, 4 fichiers seront extraits. Deux de ces fichiers (*.vx, *.vd) seront utilisés dans un contexte d'entrainement alors que les autres (*.mx, *.md) seront utilisés pour tester le réseau.

Les fichiers *.vx contiennent une liste de vecteurs correspondant à des imagettes extraites dans l'image. Dans ces fichiers, chaque ligne correspond à une nouvelle imagette alors que chaque colonne correspond à un pixel dans l'imagette. Les fichiers *.vd contiennent les valeurs désirées pour chacune des imagettes. Les valeurs désirées sont obtenus par l'opérateur de Canny, fort utilisé dans les systèmes de vision par ordinateur.

Les fichiers *.mx contiennent une matrice au format identique à celui de l'image. Les valeurs contenues dans la matrice sont les valeurs des pixels de l'image normalisées entre 0 et 1. Les fichiers *.md contiennent aussi une matrice au format identique à celui de l'image. Cette fois elle contient les valeurs désirées correspondant à chacun des pixels.

Une fonction Matlab est fournie pour l'extraction des fichiers *.vx, .*vd, *.mx et *.md :

`extraireVecteursdesImages(path,patchSize,step)`

Cette fonction extrait les fichiers (*.vx, .*vd, *.mx et *.md) pour chacune des images au format *.tif contenues dans le dossier spécifié par la chaine de caractères « path ». La grandeur des imagettes est spécifiée par « patchSize ». Si patchSize = 3, des imagettes de 3 × 3 seront extraites. Le paramètre « step » définit à quels intervalles les imagettes seront extraites. Les fichiers générés par cette fonction seront placés dans le dossier spécifié par « path ».

Choix des données d'apprentissage

Il est inutile de faire apprendre complètement toutes les images au réseau : le temps d'apprentissage serait très long, sans compter l'énorme redondance d'information. Par exemple, si la partie unie de la pelouse dans Figure 1 a) contient 6 000 fenêtres de pixels, inutile de répéter 6 000 fois au réseau que ces fenêtres ne comportent aucune arête…

Les imagettes sont donc choisies par la fonction MatLab nommée selectionDesExemples.m. La sélection fait en sorte qu'elles soient suffisamment différentes les unes des autres. Le critère de sélection est celui de la distance euclidienne entre les vecteurs associés aux imagettes.

```
[X,D] = selectionDesExemples(path,minDist)
```

Cette fonction choisit des vecteurs parmi les ceux contenus dans les fichiers *.vx appartenant au dossier spécifié par la chaine de caractères « path ». Le paramètre « minDist » spécifie la distance minimale que les exemples doivent avoir entre eux. Une plus petite distance résulte donc en une plus grande base de données. La fonction retourne une matrice X qui contient tous les exemples retenus. Chaque ligne de la matrice correspond à un nouvel exemple. Le vecteur ligne D contient les valeurs désirées associées à ces exemples.

Le réseau MLP

Le réseau MLP est un objet Matlab qui contient les poids associés à chacune des couches du réseau. Le réseau ne contiendra qu'une couche cachée. Il contient aussi les définitions pour la fonction d'activation des neurones et sa dérivée. Cet objet sera programmé dans la première partie du laboratoire. Il devra comprendre 3 fonctions :

```
obj = initialisation(obj, n_entree, n_cache, n_sortie)
```

Cette fonction ne fait qu'initialiser les matrices contenant les poids de la couche cachée et les poids de la couche de sortie. Les paramètres d'entrée correspondent au nombre d'entrées du réseau et au nombre de neurones sur la couche cachée et la couche de sortie.

```
[Y] = propagation(obj,X)
```

Cette fonction propage le vecteur X à travers le réseau afin d'obtenir Y à sa sortie. Optionnellement, cette fonction devrait pouvoir classifier une matrice X et donner les prédictions Y pour chacune des lignes de X.

```
[dW_s,dW_c] = retroPropagation(obj,X,D,eta)
```

Cette fonction exécute l'algorithme de rétropropagation sur le réseau pour le vecteur d'entrée X et sa valeur désirée D. « eta » est la constante d'apprentissage utilisée. La fonction retourne les corrections pour les matrices de poids de la couche cachée et de la couche de sortie. Ces corrections devront subséquemment être appliquées au réseau en temps opportun.

Test sur une nouvelle image

Une fois le réseau entrainé, ses performances seront évaluées sur une image qu'il n'a jamais vue. Pour ce faire, en plus des fichiers *.vx et *.vd, les fichiers *.mx et *.md seront utilisés. Le fichier *.md

contient l'image contours extraite par l'opérateur de Canny. Il s'agit de l'image désirée. À partir de notre réseau MLP et de l'image *.mx nous obtiendrons une image (mY) qui sera comparée à *.md. La fonction

```
mY = extraireContourMLP(mX,MLP,patchSize)
```

Utilisera l'objet MLP, contenant notre réseau entrainé et classifiera chacun de pixel de mX. Il faut spécifier à la fonction la dimension des imagette qui a été utilisée précédemment.

Manipulations

Dans la première partie du laboratoire, vous devrez compléter l'objet myMLP.m. Il faudra implémenter les trois fonctions de l'objet :

```
obj = initialisation(obj, n_entree, n_cache, n_sortie)
```

```
[Y] = propagation(obj,X)
```

```
[dW_s,dW_c] = retroPropagation(obj,X,D,eta)
```

Un fichier de départ vous sera fourni ainsi qu'un fichier vous permettant de tester votre implémentation.

Dans la seconde partie du laboratoire, vous expérimenterez avec votre réseau et des images. Différentes architectures de réseau seront testées pour résoudre le problème d'extraction de primitives d'une image. Un script d'exemple sera fourni pour débuter le travail. Il vous est suggéré d'utiliser les paramètres utilisés dans ce script afin de valider votre travail jusqu'ici.

Expérience 1

Vérifiez l'effet de la constante d'apprentissage. Entrainez le réseau en utilisant diverses valeurs et tracez l'erreur quadratique moyenne obtenue sur la base d'apprentissage à chaque époque. Sur un même graphique, rapportez les courbes d'un choix de constante trop élevé, un choix de constante basse (ex. 0.001) et d'une constante qui vous semble appropriée pour le problème. Commentez les différences. Comment choisiriez-vous une constante appropriée à votre problème?

Taille du réseau : 9 entrées, 5 cachées, 1 sortie
patchSize = 3;
step = 8;
minDist = 0.8;
nEpoch = 5000;

Expérience 2

Réalisez un entrainement par lot et incrémental. Dans un apprentissage par lot, on présente toutes les données au réseau avant d'ajuster les poids. Dans l'apprentissage incrémental, les poids sont ajustés pour chaque exemple. Tracez les courbes d'erreur et comparez-les. A votre avis, quelle méthode est la plus rapide en terme de temps de calcul? Quels sont les avantages respectifs des deux méthodes?

Taille du réseau : 9 entrées, 5 cachées, 1 sortie
patchSize = 3;
step = 8;
minDist = 0.8;

eta = 0.05;
nEpoch = 5000;

Expérience 3

Augmentez le nombre d'époques d'entrainement à 20 000 ou plus. À chaque époque, testez votre apprentissage sur les images contenues dans le dossier de validation. Sur un graphique, tracez l'erreur quadratique moyenne obtenue sur la base d'apprentissage avec celle obtenue sur la base de validation. Commentez ces courbes.

Taille du réseau : 9 entrées, 5 cachées, 1 sortie
patchSize = 3;
step = 8;
minDist = 0.8;
eta = 0.05;
nEpoch = 20000;

Expérience 4

En vous basant sur l'erreur obtenue sur la base de validation, déterminez le bon moment pour arrêter votre apprentissage. À ce moment, sauvegardez l'état de votre réseau et continuez l'apprentissage jusqu'à 10 000 époques. Vous possédez donc un réseau correctement entrainé et un réseau surentrainé. Servez-vous de ces deux réseaux pour extraire les contours des images de la base de test. Rapportez ces images dans votre rapport et commentez. Calculez l'erreur de classification pour ces images en utilisant les fichiers *.vx et *.vd. Pour ce faire, vous devez transformer les sorties (Y) du réseau en décision. Il suffit simplement de considérer les sortie Y > 0.5 comme des 1 et les autres comme des 0. Comptez ensuite la proportion de prédictions ratées. Est-ce que proportion de prédictions ratées est la même pour les deux classes? la Rapportez ces valeurs et commentez.

Taille du réseau : 9 entrées, 5 cachées, 1 sortie
patchSize = 3;
step = 8;
minDist = 0.8;
eta = 0.05;
nEpoch = 10000;

Expérience 5

Expérimentez afin d'obtenir le meilleur réseau pour accomplir la tâche. ATTENTION, il ne faut pas faire les choix en fonction de la base de test mais bien de la base de validation. Faites varier 3 paramètres dans la liste ci-bas et rapporter les performances sur la base de validation. Discutez des effets de ces paramètres sur les performances de classification et sur le temps (en époques et en secondes) requis par le réseau pour effectuer son entrainement et sa classification.

Choisissez votre meilleure configuration et utilisez-la sur la base de test. Expliquez le protocole et la métrique de performance utilisée pour le choix de votre modèle.

Les paramètres suivant peuvent être explorés :

- Ratio de quantité de vecteurs d'apprentissage appartenant à chaque classe.
- Distance minimale entre les vecteurs conservés dans la base d'apprentissage
- Nombre de neurones sur la couche cachée
- Dimension des imagettes
- Utilisations du momentum
- Utilisation de constante d'apprentissage pour chaque poids
- Autre fonction d'activation
- Une suggestion de votre cru…

Faites valider vos choix par le chargé de laboratoire avant de commencer.

Barème :

À la deuxième semaine, la fonctionnalité de votre MLP devra être démontrée.

25% des points attribués à ce laboratoire seront accordés à ce moment.

70% seront attribués en fonction du rapport.

5% pour les commentaires dans le code

Structure du rapport

- Description de la problématique
- Description de l'algorithme de descente de gradient
- Résultats des expériences 1 à 5 (n'oubliez pas de répondre aux questions)
- Conclusion

Le code commenté devra aussi être remis avec le rapport.