

Rapport de première soutenance

Ekip

Angel BOCHENKO, Sami MAZIRT, Pefouho Edmond NGUEFEU , Quentin SIX

24 octobre 2020

Table des matières

1	Présentation des membres du groupe	4
1.1	Edmond	4
1.2	Angel	4
1.3	Sami	5
1.4	Quentin	5
2	Tableau de répartition des tâches	6
3	Le projet - Chargement et pré-traitement de l'image	6
3.1	Suppression des couleurs	6
3.1.1	Niveaux de gris	6
3.1.2	Noir et blanc	7
3.2	Pré-traitement	8
3.2.1	Suppression de bruit	8
3.2.2	Renforcement des contrastes	9
3.3	La segmentation de l'image - Sami	10
3.4	Interface graphique	12
3.5	Extraction	13
3.6	Le fichier principal - Main.c	14
4	Le projet - Le réseau neuronal	14
4.1	Problèmes rencontrés	14
4.2	Implémentation des différentes parties	15
4.2.1	La structure neurone	15

4.2.2	La structure couche	16
4.2.3	La structure réseau neuronal	17
4.2.4	Conclusion sur les structures	19
4.3	Les fonctions du réseau	19
4.3.1	Initialiser les paramètres du réseau	19
4.3.2	L'entraînement du réseau	20
4.3.3	forward(network n_n)	21
4.3.4	calculate_final_cost(int i,network n_n)	22
4.3.5	back_prop(int p,network n_n)	23
4.3.6	update_weights_biases(network n_n)	25
4.3.7	La fonction pour tester le réseau	25
5	Conclusion	26
6	Sources	27

1 Présentation des membres du groupe

1.1 Edmond

Ce projet peut m'apporter des compétences en C qui est un langage tout nouveau pour moi pour l'instant et sur le compilateur gcc. Cela me permettra aussi de mieux travailler en équipe. Enfin, je pense en apprendre beaucoup plus sur le traitement de l'image et comment les machines réalisent certaines opérations sur les images, cela pourra m'aider en fonction de la spécialisation que j'aurais plus tard.

1.2 Angel

Le développement de cet OCR est un véritable défi puisqu'avant d'entamer celui-ci je n'avais ni de connaissances en langage C, ni en apprentissage profond. Ce projet me permet donc d'amasser davantage de connaissances dans des domaines incontournables de l'informatique, bien que je ne souhaite pas évoluer dans le domaine de l'intelligence artificielle, ce savoir est primordial pour un ingénieur informatique. Les compétences qui me seront le plus utiles et que je vais continuer de développer en finissant cet OCR sont l'utilisation de l'outil gcc, l'écriture d'un code propre, commenter celui-ci, la rigueur de la syntaxe du langage, l'utilisation de Vim et certaines commandes de gestion de fichiers.

1.3 Sami

Ce projet me permet de m'améliorer en C. En effet je n'ai jamais fait un projet aussi concret et aussi important et technique. Je pense que c'est une bonne expérience pour voir la complexité des technologies que nous utilisons au quotidien.

Pour me documenter, j'ai fait beaucoup de recherches sur internet et j'ai également utilisé des livres que j'avais chez moi (notamment sur le C).

1.4 Quentin

Je trouve que le sujet du réseau neuronal est quelque chose de très intéressant. Ce projet me permettra d'avoir un avant-goût de ce que sont les réseaux de neurones plus complexes, de voir si la programmation d'un réseau neuronal me plaît ainsi que d'apprendre un nouveau langage.

2 Tableau de répartition des tâches

Taches	Angel	Sami	Edmond	Quentin
Structure du réseau	Suppléant			Responsable
Fonctions du réseau	Responsable			Suppléant
Interface graphique		Responsable	Responsable	
Segmentation de l'image		Responsable	Suppléant	
Traitement de l'image		Suppléant	Responsable	
Reconnaissance caractères	Responsable			Responsable

3 Le projet - Chargement et pré-traitement de l'image

3.1 Suppression des couleurs

3.1.1 Niveaux de gris

La première fonction concernant la suppression de couleurs est celle des niveaux de gris. Son principe est très simple, pour chaque pixel de l'image, nous stockons la valeur du pixel, nous obtenons à partir de cette valeur, les valeurs de Rouge, Vert et Bleu sous forme d'un tableau à 3 entrées RGB et nous appliquons la formule suivante :

$$intgris = 0.3 * r + 0.59 * g + 0.11 * b$$

Nous obtenons ensuite notre valeur, et nous remplaçons chaque valeur du tableau RGB par cette dernière.



3.1.2 Noir et blanc

Cette fonction est plus simple que la précédente et elle fonctionne de manière binaire. On sait que le code RGB (0,0,0) correspond à la couleur noire et que le code RGB (255,255,255) correspond à la couleur blanche. Pour chaque pixel, nous allons donc faire la moyenne de ses pixels rouge, vert et bleu. Si cette moyenne est inférieure ou égale à 127, on passe le pixel en noir, dans le cas inverse on passe le pixel en blanc.

titre d'exemple, un pixel bleu, de code RGB (0,0,255) donnera : $(0 + 0 + 255)/3 = 85$. 85 étant inférieur à 127 il passera donc en noir. Un autre exemple, un pixel jaune, de code RGB (255,255,0) donnera : $(255 + 255 + 0) / 3 = 170$. 170 étant supérieur à 127, on passera donc ce pixel en blanc.

On s'déteste et on s'aime, de façon calamiteuse

Car chez nous le ciel est terne comme une station de la ligne 2

Ils comprennent pas nos vies parce qu'ils savent pas c'qu'on a vécu

On pourrait faire de faux sourires mais même ça on y arrive plus

Dis aux racistes qu'on les baise

Dis à l'huissier qu'on bougera pas

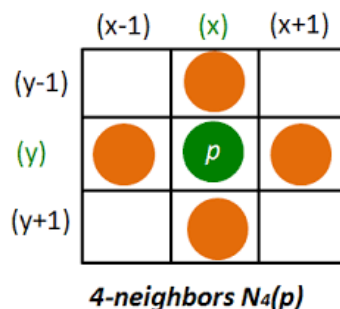
Qu'il retourne se branler sur des catalogues Conforama

On s'déteste et on s'aime, de façon calamiteuse
 Car chez nous le ciel est terne comme une station de la ligne 2
 Ils comprennent pas nos vies parce qu'ils savent pas c'qu'on a vécu
 On pourrait faire de faux sourires mais même ça on y arrive plus
 Dis aux racistes qu'on les baise
 Dis à l'huissier qu'on bougera pas
 Qu'il retourne se branler sur des catalogues Conforama

3.2 Pré-traitement

3.2.1 Suppression de bruit

Pour l'algorithme de suppression de bruits parasites et de tâches, la fonction doit encore être implémentée mais le principe est compris. Ici, on travaillera avec 2 fichiers, l'un pour l'image d'entrée, et l'autre où l'on va rentrer pixel par pixel l'image de sortie, corrigée. Pour chaque pixel, on fonctionnera encore à partir de la moyenne des 3 valeurs de Rouge, Vert et Bleu. Si la moyenne des pixels RGB est de 255 ou de 0 et donc si le pixel est noir ou blanc, et qu'il ne fait pas partie de bordures de l'image, on prendra ses 8 voisins directs et on recueillera la valeurs de leur pixels RGB. On en fait la moyenne, en utilisant faisant la somme de `get_pixel` sur chacun d'entre eux et en la divisant par 8. Ensuite, on prendra `set_pixel` pour donner au pixel du milieu la valeur de la moyenne.



3.2.2 Renforcement des contrastes

Pour le renforcement des contrastes, nous utiliserons la méthode d'OTSU à partir de l'image d'entrée. La méthode d'OTSU consiste à diviser tous les pixels d'une image en 2 classes C1 et C2, C1 comportant les pixels compris entre 0 et k avec un $k < 255$ donné. L'objectif est de trouver un k afin qu'il sépare au mieux le fond et les objets traités. Pour trouver k, qui sera la variance interclasse, nous allons procéder en 6 étapes.

Tout d'abord on va faire un histogramme des niveaux de gris de l'image. Pour chaque pixel, on ajoute 1 dans la case du tableau de 256 valeurs correspondante. Nous allons ensuite construire un nouvel histogramme à partir du premier, lui sera normalisé afin que toutes les valeurs soient comprises entre 0 et 1. Pour ce faire, nous allons pour chaque valeur x de l'histogramme, utiliser la fonction de normalisation suivante :

$$f(x) = (x - \min) / (\max - \min)$$

(avec min étant la plus petite valeur de l'histogramme et max la plus grande)

Notre nouvel histogramme nous donne donc une distribution de probabilités. Par exemple, la probabilité qu'un pixel de l'image ait un niveau de gris de 150 sera la valeur correspondant à 150 dans l'histogramme normalisé.

On calcule ensuite la probabilité qu'un pixel de l'image appar-

tienne à la classe C1. Pour cela on fait la somme de toutes les colonnes de l'histogramme comprises entre 0 et k. Pour calculer après cela la probabilité qu'un pixel appartienne à C2, il nous suffira de calculer $1 - P(C1)$, avec $P(C1)$ trouvé précédemment. Enfin, nous calculons la variance interclasse avec la formule suivante :

$$var = (Moy * P(C1) - Moy(C1))/P(C1) * P(C2)$$

On répète les opérations pour tous les k allant de 0 à 254 et le k optimal sera le k pour lequel on aura la plus haute valeur de variance.

C'est donc pour cette valeur que les contrastes seront les plus importants. On prendra ensuite tous les pixels de l'image, on rendra tous les pixels inférieurs à k noirs et tous ceux supérieurs à k, blancs.

3.3 La segmentation de l'image - Sami

Concernant la segmentation, j'ai commencé par un traitement des lignes, pour que ce soit plus simple de séparer les caractères par la suite.

Pour séparer les lignes, je parcours chaque ligne de pixel de l'image, je repère ainsi la ligne où il y a un pixel noir (car l'image a été au préalable traitée pour qu'elle soit en noir et blanc) synonyme de présence de caractère et celle du dessus où il n'y a que des pixels non noirs. J'ai ainsi le moyen de tracer

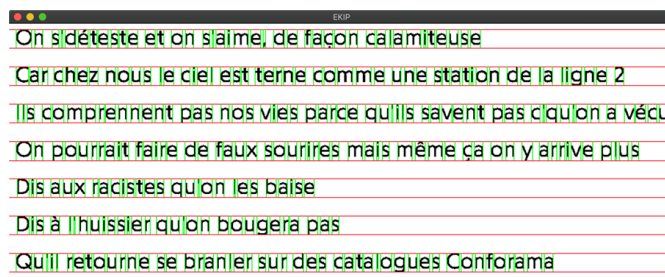
un trait(rouge) au-dessus de la ligne (de texte/caractères).

De la même manière, je peux repérer quelle ligne (de pixels) marque la fin d'une ligne de caractère. J'ai ainsi deux variables, `pligne` et `dligne` qui contiennent les coordonnées de la première ligne de pixels qui marque le début d'une ligne de texte et la dernière ligne de pixels qui marque la fin de la ligne de texte. Cette méthode n'est pas une des plus optimales, en effet si nous avons deux lignes de texte qui se touchent, la fonction de segmentation ne pourra pas séparer ces deux lignes et elle les laissera en une grosse ligne et les résultats seront donc erronés. J'ai donc essayé une méthode qui emploie un histogramme. Pour se faire, je compte le nombre de pixel noir sur une ligne et je l'insère dans l'histogramme. Je regarde ensuite dans l'histogramme l'endroit où il y a le moins de pixels noirs et j'effectue un calcul afin de savoir quelle ligne a le plus de chance d'être la ligne de séparation. Avec cette méthode de segmentation, le problème précédent ne l'est plus. Malheureusement j'ai décidé d'abandonner cette méthode du a de nombreux bugs.

Concernant la segmentation verticale, c'est-à-dire la séparation des caractères 1 à 1, je procède de la même manière mais cette fois ci, j'ai les variables `pligne` et `dligne` qui me permettent de savoir où les caractères sont situés et donc qui m'évite de devoir parcourir toute l'image mais seulement les endroits où il y a des lignes.

Pareillement a la fonction de segmentation horizontal, le pro-

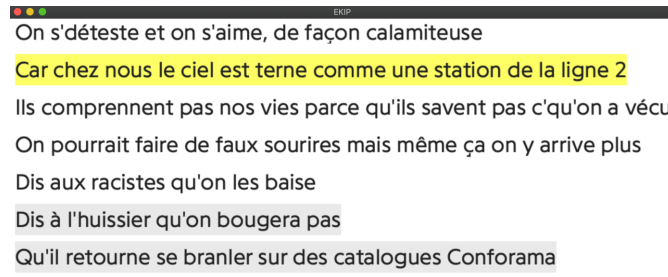
blème de deux caractères collés est présent. La méthode qui emploie un histogramme peut être utilisée pour y remédier.



Sur cette image, vous pouvez voir ce que l'algorithme de segmentation fait sur un texte. Les lignes sont séparées par des traits rouge et les caractères par des traits vert. Nous pouvons voir qu'entre les caractères il y a des cases vides (hors espaces). Ces cases ne seront pas traitées lors de l'extraction. En effet, dans ma fonction de segmentation vertical, en plus d'avoir les coordonnées de haut et de bas de ligne, j'ai également les coordonnées pour chaque « bordures de caractères », pour l'extraction donc, seules les cases avec des caractères seront traitées.

3.4 Interface graphique

Pour l'interface graphique nous avons choisi d'utiliser SDL2. Nous avons fait une fonction qui attend qu'on appuie sur une touche pour passer les étapes. J'ouvre donc l'image et ensuite on progresse dans les différentes étapes en appuyant sur une touche. Vous pouvez voir en dessous ces différentes étapes.

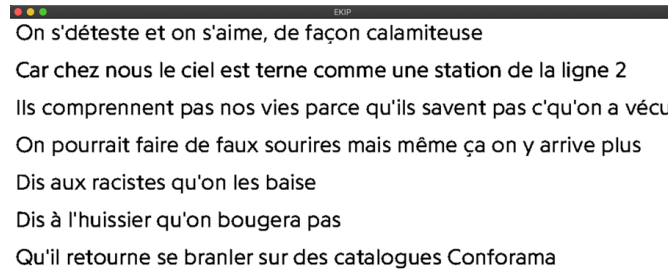


```

On s'déteste et on s'aime, de façon calamiteuse
Car chez nous le ciel est terne comme une station de la ligne 2
Ils comprennent pas nos vies parce qu'ils savent pas c'qu'on a vécu
On pourrait faire de faux sourires mais même ça on y arrive plus
Dis aux racistes qu'on les baise
Dis à l'huissier qu'on bougera pas
Qu'il retourne se branler sur des catalogues Conforama

```

L'image s'ouvre.

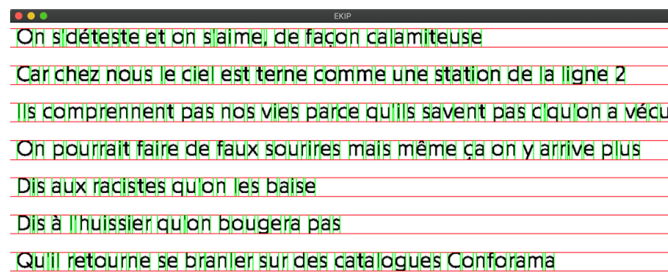


```

On s'déteste et on s'aime, de façon calamiteuse
Car chez nous le ciel est terne comme une station de la ligne 2
Ils comprennent pas nos vies parce qu'ils savent pas c'qu'on a vécu
On pourrait faire de faux sourires mais même ça on y arrive plus
Dis aux racistes qu'on les baise
Dis à l'huissier qu'on bougera pas
Qu'il retourne se branler sur des catalogues Conforama

```

En appuyant sur une touche, nous avons le traitement qui est effectué (noir et blanc, bruit, ...).



```

On s'déteste et on s'aime, de façon calamiteuse
Car chez nous le ciel est terne comme une station de la ligne 2
Ils comprennent pas nos vies parce qu'ils savent pas c'qu'on a vécu
On pourrait faire de faux sourires mais même ça on y arrive plus
Dis aux racistes qu'on les baise
Dis à l'huissier qu'on bougera pas
Qu'il retourne se branler sur des catalogues Conforama

```

En appuyant sur une touche on a la segmentation qui apparait.

3.5 Extraction

Concernant l'extraction, je crée une matrice de taille correspondant à la case correspondant au caractère à extraire. Je récupère ensuite la valeur de chaque pixel de la case que j'insère à la bonne place dans la matrice. Je redimensionne ensuite la matrice pour ne pas avoir de problèmes de tailles avec les autres cases qui pourraient être de taille différente.

La prochaine étape, sera de faire tourner le réseau de neurones sur cette matrice afin qu'il puisse reconnaître le caractère et l'écrire dans le fichier final.

3.6 Le fichier principal - Main.c

Dans cette fonction « principale », j'exécute les fonctions, je déclare les variables nécessaires a la bonne exécution de notre OCR. Je charge l'image donnée en paramètre dans un premier temps, j'attends ensuite qu'une touche soit activée pour traiter l'image (bruit, noir et blanc, ...). A la deuxième touche appuyée, je segmente l'image et au troisième touché j'extrait les caractères.

4 Le projet - Le réseau neuronal

4.1 Problèmes rencontrés

Un des nombreux problèmes rencontrés lors du codage des fonctions liées aux réseau neuronal a été comment manipuler les différents composants du réseau. L'implémentation d'un réseau neuronal étant assez compliqué pour des personnes n'ayant jamais fait de langage C et n'ayant jamais (ou presque) codé de réseau neuronal avant, il nous a fallut nous documenter. Après avoir regardé sur plusieurs sites internet différents, visionné de nombreuses vidéos sur le fonctionnement des fonctions liés à l'apprentissage du réseau et bien avoir assimilé les points communs

entre tous les réseaux neuronaux nous avons donc commencé à coder.

4.2 Implémentation des différentes parties

Après avoir essayé par plusieurs moyens de coder (et n'ayant jamais abouti à quelque chose de concret) le projet nous avons donc décidé de commencer par l'implémentation des bases de tous les réseaux, l'architecture. Cela nous permet, une fois les avoir bien implémenté de "simplement appliquer" les formules mathématiques liées à l'apprentissage du réseau. Ce dernier est donc composé de trois éléments différents comportant ses propres caractéristiques ; le neurone, la couche (ou "layer") et le réseau neuronal en lui même.

4.2.1 La structure neurone

La première structure implémentée fut celle du neurone. Plusieurs neurones (ou un seul) formeront ensuite une couche. Chaque neurone sera donc unique et aura plusieurs paramètres :

- Une valeur d'activation (comprise entre 0 et 1)
- La dérivée partielle de la fonction d'activation (utilisée dans le système de "backpropagation")
- Un tableau de poids (sa taille dépend du nombre de neurones dans la couche suivante, s'il y a x neurones dans la couche suivante alors le tableau aura x emplacements)
- Un tableau des dérivés partielles des poids (utilisé dans le sys-

tème de "backpropagation")

- Une valeur notée "z" (qui se trouve en appliquant une formule mathématique)
- La dérivée partielle de "z" (utilisée dans le système de "backpropagation")
- Un biais (compris entre 0 et 1)
- La dérivée partielle du biais (utilisée dans le système de "backpropagation")

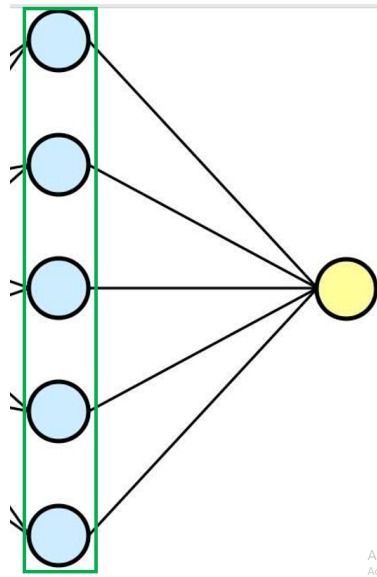
A la création d'un neurone tous les paramètres sont initialisés à 0 dans l'attente de la première fonction liée au réseau, celle de l'initialisation des poids et des biais.

4.2.2 La structure couche

La seconde structure implémentée fut celle de la couche. Plusieurs couches (au moins deux) formeront ensuite le réseau neuronal. Chaque couche aura plusieurs paramètres :

- Un nombre de neurones
- Un tableau de neurones

Sur le schéma ci-dessous, la couche (encadrée en vert) comporte 5 neurones et aura donc dans son tableau de neurones 5 emplacements (le neurone du haut étant à l'indice 0).



4.2.3 La structure réseau neuronal

La dernière structure est tout simplement le réseau neuronal en lui même. Il a ses propres caractéristiques qui sont les suivantes :

- Une valeur pour le coût final (qui nous permet de savoir si l'apprentissage se passe bien)
- Un tableau de coût (qui permet de calculer le coût final et est utilisé dans le processus de "backpropagation")
- Un nombre de couches (minimum 2)
- Un tableau de couche (qui se fait en fonction du nombre de couche, chaque emplacement du tableau correspondra à une couche du réseau)
- Un tableau de nombre de neurones (qui se fait en fonction du nombre de couches, chaque emplacement du tableau correspondra au nombre de neurones dans une certaine couche)
- Une valeur pour le taux d'apprentissage (qui influence sur l'efficacité des fonctions d'apprentissage du réseau)

- Un nombre d'exemple d'entraînement (dans le cas de la fonction XOR il y aura quatre exemples d'entraînements)
- Une matrice de valeur d'entrées (utilisée pour les fonctions d'apprentissage), la première dimension de la matrice sera le numéro de l'exemple d'entraînement, entre 0 et n-1 (avec n étant le nombre d'exemples d'entraînement) et la deuxième dimension du tableau sera pour dire les valeurs désirées pour la couche d'entrée pour l'exemple en question. Pour faire le lien avec la fonction XOR, 4 exemples seront nécessaires pour l'entraînement du réseau : 0 0, 1 1, 1 0 et 0 1
- Une matrice de valeur désirées (utilisée pour les fonctions d'apprentissage), la première dimension de la matrice sera le numéro de l'exemple d'entraînement, entre 0 et n-1 (avec n étant le nombre d'exemples d'entraînement) et la deuxième dimension du tableau sera pour dire les valeurs désirées pour la couche de sortie pour l'exemple en question. Pour faire le lien avec la fonction XOR et en reprenant les valeurs du point précédant, les sorties désirées seront donc dans l'ordre 0, 0, 1 et 1.

Au contraire du neurone ou de la couche le réseau a des paramètres constants, par exemple pour cette première soutenance avec la mise en place de la fonction XOR, le réseau possède trois couches, une pour les entrées (qui possède deux neurones), une couche "cachée" (qui en possède dix) et enfin une couche de sortie (qui possède un seul neurone) qui est le résultat de la fonction

XOR.

4.2.4 Conclusion sur les structures

Chaque structure a son fichier code source associé : `neuron.c` et `neuron.h` pour le neurone, `layer.c` et `layer.h` pour la couche et enfin `neural_network.c` et `neural_network.h` pour le réseau neuronal.

4.3 Les fonctions du réseau

La première "fonction" du réseau est celle qui permet d'allouer les différents espaces mémoires ainsi que de créer la structure du réseau (tous les paramètres des neurones seront d'abord initialisés à 0). C'est tout simplement une fonction qui construit une structure "neural_network".

4.3.1 Initialiser les paramètres du réseau

Après avoir créée la structure du réseau la première chose à faire avant de pouvoir lancer les fonctions pour entraîner le réseau est d'initialiser les différents poids et biais des différents neurones. La fonction `init_weights_bias` est donc appelée et va simplement parcourir tous les neurones du réseau et initialiser pour le biais une valeur comprise entre 0 et 1 quand cela sera nécessaire. En effet pour la première couche l'initialisation des biais est inutile. Pour les poids des neurones le même principe s'applique sauf que là ce sont les tableaux de poids des neurones qui sont

remplis sauf pour la dernière couche (pas possible).

4.3.2 L'entraînement du réseau

L'entraînement du réseau se fait dans la fonction `train_neural_net` (`network n_n`). Cette dernière est un enchaînement de diverses fonctions permettant au réseau d'apprendre grâce à des valeurs passées en paramètres. La fonction `forward(network n_n)` est dédiée à la propagation avant. Les informations mises en paramètres de la première couche du réseau neuronal (aussi appelée `input layer`) se déplacent en traversant l'unique couche cachée de notre réseau pour finir leurs trajets dans la troisième et dernière couche du réseau. Celle-ci contient les données finales (en anglais : `outputs`) que l'on va tenter d'affiner en entraînant notre réseau à l'aide des fonctions à venir. L'intérêt premier de celles-ci est donc d'apprendre au réseau à renvoyer les bonnes sorties en fonctions des entrées. Pour ce faire, il faut agir sur ce qu'on appelle les poids et les biais de chaque neurone. Le but est alors de déterminer les plus sensibles et les moins sensibles d'entre eux (`Hebbian theory`) sur la fonction de coût, et de les modifier plus ou moins selon leur sensibilité (`update_weights_biases(network n_n)`) afin d'obtenir une fonction de coût plus précise (`calculate_final_cost(int i, network n_n)`). La détermination de tout ceci s'effectue à l'aide de dérivées partielles (`back_prop(int p, network n_n)`). Pour la suite de l'explication de l'entraînement de notre réseau de neurones nous

poserons ceci :

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

z étant une application linéaire du poids (w) du neurone courant multiplié par la fonction d'activation (a) du neurone précédent, le tout additionné au biais (b) du neurone. Cette équation vaut pour un réseau contenant un neurone dans chaque couche.

4.3.3 forward(network n_n)

C'est à partir de cette fonction que l'apprentissage commence. Elle va sommer les applications linéaires z précédemment vues de tous les neurones de la couche cachée et de la couche de sortie. Ensuite, nous attribuons à chacun des neurones de la couche cachée la fonction d'activation ReLU (Rectified Linear Unit, ou unité linéaire rectifiée). Elle consiste à remplacer les résultats négatifs obtenus par 0.

$$f(x) = \max(0, x)$$

On applique donc la formule

$$a^{(L)} = \sigma(z^{(L)})$$

par conséquent si z est inférieur à 0, a est nul, sinon a prend la valeur de z . Pour la couche de sortie nous avons attribué la fonction sigmoïde.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Notez que ces équations ne sont pas prises au hasard, on les établit afin de calculer certains paramètres nécessaires à l'avancement de l'entraînement. Notre but ultime étant de calculer la sensibilité de la fonction de coût pour des petits changements de w ainsi que pour des petits changements de b . Ces calculs sont les suivants :

$$\frac{\partial C_0}{\partial w_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

4.3.4 `calculate_final_cost(int i, network n_n)`

Cette fonction permet de connaître le coût du réseau neuronal. Plus ce dernier est proche de 0, plus celui-ci sera précis et donc efficace. Pour notre OCR, il est calculé grâce à une fonction quadratique.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

n étant le nombre d'exemples de l'entraînement, L le nombre de couches du réseau et y le résultat attendu. La fonction est donc chargée de faire une moyenne du coût pour tout le réseau. A chaque entraînement son résultat est affiché afin de bien constater que le réseau neuronal est en train d'apprendre si le résultat converge vers 0.

4.3.5 back_prop(int p, network n_n)

L'étape de la propagation arrière se fait dans cette fonction. Elle permet de déterminer les dérivées nécessaires à l'amélioration des biais et des poids du réseau. Nous allons déterminer les dérivées de w et de b à l'aide de ces équations :

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

Cette étape est appelée propagation arrière puisque comme visible sur la première image ci-dessus, elle met en relation la fonction d'activation du neurone courant avec celle du neurone de la couche précédente. On va donc parcourir le réseau de la couche de sorties jusqu'à la couche d'entrées, c'est-à-dire dans le sens contraire de la fonction forward. A cette étape, on a donc tous les paramètres nécessaires pour trouver les dérivées de w et de b.

A partir de ces équations on déduit que la dérivée de w est égale à la dérivée de z multipliée par $a^{(L-1)}$. La dérivée de b est égale à la dérivée de z.

La dérivée de a est égale à w multipliée également par celle de z. Ces trois affirmations nous sont extrêmement utiles afin de coder ces formules bien plus facilement qu'avec des dérivées partielles.

Cependant, les termes de droite des équations ci-dessus incluent une dérivée de fonction d'activation. On va donc procéder en deux fois puisque notre couche de sorties et notre couche cachée ont des fonctions d'activation différentes. Débutons par la couche de sorties.

On commence par initialiser dz (dérivée de z) qui va nous être très utile puisqu'elle se retrouve en facteur de toutes les autres dérivées. Elle est égale à $2 \cdot (a - y) \cdot a \cdot (1 - a)$, avec y la valeur de sortie attendue. Ensuite, pour chaque neurone de cette couche sont calculées dw et da , respectivement égales à $dz \cdot a$ et $w \cdot dz$. db est ensuite initialisé à dz .

Le déroulement des calculs de la couche cachée est très similaire. La seule différence réside dans le fait que l'on teste pour chaque neurone si z est supérieur ou égal à 0. Si c'est le cas dz est égal à da , sinon dz est égal à 0. En faisant ceci on déconnecte en quelque sorte les neurones non utilisés par les données reçues des neurones d'entrée puisque si dz est égal à 0, dw , da et db le sont aussi.

$$\begin{aligned}
 &\text{backprop:} \\
 &\left. \begin{array}{l} \text{output} \\ \text{signal} \end{array} \right\} \begin{aligned} &d_{z_2} = 2(a' - y)(a')(1 - a') \\ &dw_0 = d_{z_1} * a_0 \\ &da_0 = w_0 * d_{z_1} \\ &db = d_{z_2} \end{aligned} \\
 &\left. \begin{array}{l} \text{hidden} \\ \text{ReLU} \end{array} \right\} \begin{aligned} &\text{si } z_1 \geq 0: \\ &\quad d_{z_1} = da \\ &\text{sinon:} \\ &\quad d_{z_1} = 0 \\ &dw = d_{z_2} * a \\ &dw = w * d_{z_2} \\ &db = d_{z_1} \end{aligned}
 \end{aligned}$$

4.3.6 update_weights_biases(network n_n)

C'est par cette fonction que chaque entraînement se termine. En effet, elle permet d'améliorer la précision des poids et des biais. Effectivement, la fonction précédente nous a permis de récupérer les valeurs des dérivées de w et de b . Grâce à ces dernières, on obtient un réseau neuronal plus intelligent, en changeant w et b par $w - (\alpha * dw)$ et $b - (\alpha * db)$. Alpha étant le pas de chaque itération de l'apprentissage du réseau, fixé à 0.15 ici.

4.3.7 La fonction pour tester le réseau

Enfin, une fois avoir entraîné le réseau avec les exemples d'entraînements, la fonction `test_nn` est utilisée pour permettre à l'utilisateur de mettre ses propres valeurs d'entrée (parmi les 4 possibles de la fonction XOR). Les valeurs d'entrée seront donc deux valeurs numériques (0 ou 1) séparés par un espaces ou par un retour à la ligne.

```
Enter input to test:
0 0
Output: 0
Output: 0.003923

Enter input to test:
1 0
Output: 1
Output: 0.996434
```

5 Conclusion

Après avoir fait les bases pour cette première soutenance aussi bien au niveau du chargement et du pré-traitement de l'image que du réseau neuronal nous allons pour la prochaine et dernière soutenance logiquement lier ces deux grandes parties pour aboutir au but premier du projet qui est la reconnaissance optique de caractères.

6 Sources

<https://www.youtube.com/watch?v=aircAruvnKk&t=2s>

<https://www.youtube.com/watch?v=IHZwWFHwa-w&t=1028s>

<https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=552s>

<https://www.youtube.com/watch?v=tIeHLnjs5U8&t=499s>

<http://neuralnetworksanddeeplearning.com/chap2.html>

<http://neuralnetworksanddeeplearning.com/chap1.html>

https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels

<https://towardsdatascience.com/understanding-backpropagation-al>

<https://towardsdatascience.com/segmentation-in-ocr-10de176cf373>

<https://www.geeksforgeeks.org/>

https://www.wikiwand.com/en/Otsu%27s_method