ASDIA Projet 1

Préliminaires

Partie 1 : Apprentissage supervisé pour la classification

Objectifs de l'apprentissage supervisé

On suppose qu'une catégorie binaire, $y \in \{0, 1\}$, est associée un vecteur d'observations multidimensionelles, notées $\mathbf{x} \in \mathbb{R}^D$. Par exemple, $\mathbf{x}$ peut représenter un image, un texte, un fichier audio, etc. La classification peut indiquer la présence d'un animal domestique (chat vs chien), une appréciation subjective (sentiment positif ou négatif), un style musical (rock ou pop), etc. Dans tous ces exemples, on aura spécifié la representation vectorielle des observations d'une manière ou d'une autre, par une méthode d'encodage spécifique à l'application considérée.

L'objectif de l'apprentissage supervisé est de construire un modèle de prédiction probabiliste, $q(\mathbf{x}, \theta) \in [0, 1]$, évaluant, pour tout $\mathbf{x}$, la probabilité pour que l'observation $\mathbf{x}$ provienne de la catégorie 1. Pour cela, la méthode adoptée consiste à minimiser une fonction de perte moyenne

$$L(\theta) = \mathbb{E}[L(q(\mathbf{x}, \theta), y)] \,.$$

Les coefficients du modèle, θ, sont appelés *paramètres* ou *poids*. Le problème se résume à minimiser la fonction de perte moyenne exprimée en fonction de θ, c'est à dire calculer les coefficients suivants

$$\hat{\theta} = \arg\min_{\theta} L(\theta) \,.$$

D'autres coefficients que les paramètres du modèle peuvent aussi entrer en compte dans le processus de minimisation. Ces coefficients supplémentaires sont appelés les *hyperparamètres*. Ils peuvent représenter le nombre d'itérations de l'algorithme de minimisation, la dimension ou la complexité du modèle (nombre de neurones ou de couches de neurones par exemple), des paramètres de régularisation, la condition initiale de l'algorithme de minimisation (seed du générateur), la méthode numérique choisie pour réaliser la minimisation de $L(\theta)$, etc.

La valeur théorique de l'espérance étant difficile à calculer, la perte moyenne est généralement évaluée à partir d'un échantillon de données d'apprentissage $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$. Cet ensemble de données est généralement de grande taille et il permet (sous des hypothèses raisonnable) d'approcher la valeur moyenne de la fonction de perte de la manière suivante

$$L(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} L(q(\mathbf{x}_i, \theta), y_i) \,.$$

La fonction de perte empirique définie ci-dessus est alors utilisée pour calculer $\hat{\theta}$. Un ensemble de test est généralement utilisé pour évaluer les performances prédictives du modèle et choisir un ensemble d'hyperparamètres conduisant à des performances satisfaisantes.

Perte log-loss (entropie croisée)

La fonction de perte *log-loss* est très fréquemment considérée par les algorithmes d'apprentissage probabiliste. Elle est définie de la manière suivante

$$L(q(\mathbf{x}), y) = -y \log q(\mathbf{x}) - (1-y) \log(1 - q(\mathbf{x}))\,,$$

où $q(\mathbf{x})$ est identifiée à la probabilité d'observer la classe 1 sachant $\mathbf{x}$, $q(\mathbf{x}) = q_1(\mathbf{x})$.

La perte moyenne associée à la fonction de perte **log-loss** est égale à

$$\mathbb{E}[L(q(\mathbf{x}), y)] = \int \mathbb{E}[L(q(\mathbf{x}), y)|\mathbf{x}]p(\mathbf{x})d\mathbf{x}\,.$$

En termes de théorie de l'information, minimiser la perte log-loss moyenne revient à minimiser l'information perdue lorsque l'on approche la probabilité cible $p(y = 1|\mathbf{x})$ par un prédicteur probabiliste donné, $q(\mathbf{x})$.

L'objet de cette partie est de démontrer le résultat suivant.

Théorème. *Soit $q(\mathbf{x})$ une fonction arbitraire à valeur dans $[0, 1]$ et*

$$q_{\text{opt}}(\mathbf{x}) = p(y = 1|\mathbf{x})\,, \quad \forall \mathbf{x}.$$

Alors, nous avons

$$\mathbb{E}[L(q(\mathbf{x}), y)] \geq \mathbb{E}[L(q_{\text{opt}}(\mathbf{x}), y)].$$

- **Conditionnement** : Montrer que pour minimiser l'espérance $\mathbb{E}[L(q(\mathbf{x}), y)]$, il suffit de minimiser l'espérance conditionnelle $\mathbb{E}[L(q(\mathbf{x}), y)|\mathbf{x}]$ pour tout $\mathbf{x}$.
- **Entropie croisée** : On définit l'*entropie croisée* de deux lois de Bernoulli p et p' (attribuant des valeurs à 0 ou 1 uniquement) de la manière suivante

$$h(p, p') = -p(y = 0) \log p'(y = 0) - p(y = 1) \log p'(y = 1).$$

Démontrer que

$$h(p, p') = \mathbb{KL}(p\|p') + h(p) \geq h(p)\,,$$

où $\mathbb{KL}(p\|p')$ est la divergence Kullback-Leibler et $h(p)$ l'entropie de p.

- **Entropie croisée et log-loss** : On appelle $q_.(\mathbf{x})$ la loi de Bernoulli de paramètre $q(\mathbf{x})$. Démontrer que

$$\mathbb{E}[L(q(\mathbf{x}), y)|\mathbf{x}] = h(p_{\mathbf{Y}}(.|\mathbf{x}), q_.(\mathbf{x})).$$

- Démontrer le théorème.

En conclusion, minimiser l'espérance conditionnelle $\mathbb{E}[L(q(\mathbf{x}), y)|\mathbf{x}]$ revient à minimiser la divergence $D_{KL}(p_{\mathbf{y}|\mathbf{x}}\|q(\mathbf{x}))$. Ainsi, nous pouvons affirmer que minimiser la perte moyenne log-loss revient à minimiser l'information perdue lorsque l'on approche la loi cible $p(y|\mathbf{x})$ par un modèle probabiliste $q(\mathbf{x})$.

- On suppose que la perte moyenne est évaluée à partir d'un échantillon de données d'apprentissage $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, indépendantes et de même loi. On suppose que la loi conditionnelle est modélisée à l'aide d'ensemble de paramètres, θ, par la fonction suivante

$$p(y = 1|\mathbf{x}, \theta) = q(\mathbf{x}, \theta)\,.$$

Démontrer que le minimum de la fonction de perte correspond au maximum de la vraisemblance pour ce modèle.

Réseaux de neurones

Neurone probabiliste

Soit $\mathbf{x} \in \mathbb{R}^D$. On appelle *neurone probabiliste* un prédicteur probabiliste de la forme suivante

$$q(\mathbf{x}, \theta) = \sigma(\mathbf{w}^T \mathbf{x} - b)\,, \quad \mathbf{x} \in \mathbb{R}^D,$$

où la fonction sigmoide (σ) est définie par

$$\sigma(x) = \frac{1}{1 + e^{-x}}\,, \quad x \in \mathbb{R},$$

et le paramètre θ possède $(D + 1)$ coefficients

$$\theta = (b, \mathbf{w}).$$

Le vecteur de coefficients $\mathbf{w}$ est de dimension D et b est un scalaire représentant un seuil.

La motivation initiale d'un modèle neuronal est d'imiter un neurone biologique. En effet, les variables $\mathbf{x}$ représentent des stimuli externes, les coefficients $\mathbf{w}$ représentent des poids synaptiques, et la fonction "sigmoide" correspond à la fonction d'activation d'un neurone biologique. Le neurone emet un potentiel d'action avec une probabilité d'autant plus grande que la somme pondérée des stimuli est grande.

La notation peut être simplifiée en ajoutant une variable d'entrée x_0 fixée de manière constante à la valeur -1 ($x_0 = -1$). Dans ce cas, nous pouvons augmenter la dimension du vecteur $\mathbf{w}$ à $D + 1$, et prendre $w_0 = b$. La fonction de decision neuronale est alors décrite par la formule suivante

$$q(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x})\,, \quad \mathbf{x} \in \mathbb{R}^D.$$

Dans cette notation (abusive), nous n'avons pas augmenté la dimension de $\mathbf{x}$ à $D + 1$ pour bien mettre en évidence que l'espace des observation est resté identique.

Notons que la fonction sigmoide n'est pas la seule fonction d'activation proposée dans les réseaux neuronaux. La fonction *rectified linear unit* (*relu*) est souvent utilisée dans les couches cachées pour son efficacité numérique. La fonction *relu* est définie par

$$\mathrm{relu}(x) = \max(0, x), \quad x \in \mathbb{R}.$$

Elle est parfois remplacée par la fonction *softplus* définie de la manière suivante

$$\mathrm{softplus}(x) = \log(1 + e^x), \quad x \in \mathbb{R}.$$

Lorsque l'on encode les variables de classes y en variables catégorielles, $y = 1$ est codé par (0,1) et $y = 0$ est codé par $(1, 0)$ on remplace la fonction sigmoide par la fonction softmax. Cette dernière peut être généralisée à plusieurs catégories. Cela a un intêret s'il y a plus de deux catégories.

Réseau de neurones probabilistes

L'intelligence artificielle propose d'approcher le prédicteur optimal par des combinaisons de fonctions neuronales agencées en couches successives, comme dans le cerveau humain. L'idée n'est pas d'utiliser une base fonctionnelle particulière mais d'apprendre une base adaptable au problème considéré, afin d'utiliser les meilleures représentations des variables observées dans chaque problème particulier.

Par exemple, un réseau de neurones possédant une couche cachée s'écrit de la manière suivante

$$q(\mathbf{x}, \theta) = \sigma(\mathbf{W_0}^T \sigma(\mathbf{W_1}^T \mathbf{x}))\,.$$

où le paramètre global, θ, inclut les seuils d'activation et les coefficients synaptiques. Il correspond à la formule suivante

$$\theta = (\mathbf{W_0}, \mathbf{W_1}).$$

Les notations $\mathbf{W_0}^T$ et $\mathbf{W_1}^T$ désignent désormais des matrices de dimensions respectives $(L \times 1 + 1)$ et $(D \times L + L)$ et la fonction *sigmoide* est appliquée à chaque coordonnée des vecteurs. Le paramètre L correspond au nombre de neurones dans la couche cachée. Le paramètre global θ est donc de dimension $(D + 2)L + 1$. Ainsi, pour deux dimensions ($D = 2$) et 20 neurones, nous aurons 81 paramètres distincts dans le modèle.

La formule peut se généraliser à l'utilisation de plusieurs couches de neurones. On parle alors de réseaux multi-couches. On qualifie les réseaux multi-couches de *réseaux profonds* à partir d'une douzaine de couches (hum deux selon certains auteurs). Par exemple, un réseau de neurones possédant 3 couches cachées s'écrit

$$q(\mathbf{x}, \theta) = \sigma(\mathbf{W_0}^T \sigma(\mathbf{W_1}^T \sigma(\mathbf{W_2}^T \sigma(\mathbf{W_3}^T \mathbf{x})))\,, \quad \mathbf{x} \in \mathbb{R}^D.$$

[Un réseau de neurones artificiels. Le réseau prend une variable de dimension 3 en entrée et la classe en deux catégories (softmax) en sortie. Il y a une couche cachée possédant 4 neurones.]

Gradient stochastique

L'apprentissage, c'est à dire la minimisation effective de la fonction de perte $L(\theta)$, est généralement réalisé par une méthode d'optimisation numérique. Il existe de nombreuses méthodes d'optimisation numérique permettant de trouver un minimum local de la fonction $L(\theta)$.

La méthode de prédilection des réseaux de neurones est la méthode du gradient. Il s'agit d'une méthode de minimisation locale. La méthode du gradient repose sur le calcul des dérivées partielles de la fonction $L(\theta)$

$$\nabla L = \left(\frac{\partial L}{\partial \theta_1}, \ldots, \frac{\partial L}{\partial \theta_m} \right)^T .$$

D'après la formule de Taylor (ou d'après la définition de l'application différentielle), la fonction $L(\theta)$ peut être développée localement de la manière suivante

$$L(\theta + d\theta) = L(\theta) + \nabla L(\theta)^T d\theta + o(d\theta)\,.$$

Ainsi, les déplacements $d\theta$ conduisant aux plus grandes variations de la fonction $L(\theta)$ au voisinage de θ s'effectuent dans une direction parallèle au vecteur $\nabla L(\theta)$.

La méthode de gradient est une méthode rudimentaire cherchant à suivre la ligne de plus grande pente afin de minimiser la fonction de perte, $L(\theta)$. Il s'agit d'une méthode itérative qui met à jour le paramètre θ de la manière suivante

$$\theta^{t+1} = \theta^t - \eta \nabla L(\theta^t)\,.$$

La constante η est petite et elle est appelée *taux d'apprentissage* (learning rate). Lors d'une mise à jour de l'algorithme, la valeur de la fonction L a ainsi tendance à décroitre

$$L(\theta^{t+1}) - L(\theta^t) \approx -\eta \|\nabla L(\theta^t)\|_2^2 < 0\,,$$

et θ^t converge vers un minimum local, un point selle ou un plateau de $L(\theta)$.

Cette méthode peut être raffinée en utilisant les dérivées secondes de la fonction $L(\theta)$. Ceci est pertinent lorsque le paramètre θ est de dimension raisonnable, de l'ordre de quelques dizaines de coefficients. En grande dimension, la méthode du du gradient reste la méthode favorite dans les implémentations logicielles des réseaux neuronaux.

Les méthodes de *gradient stochastique* sont utilisées en très grande dimension. Elles implémentent l'algorithme suivant

1. Choisir une valeur initiale θ et une valeur du taux d'apprentissage η.
2. Répéter les étapes suivantes jusqu'à ce qu'un minimum acceptable soit atteint

- permuter aléatoirement les éléments de l'échantillon d'apprentissage
- pour tout i de 1 à n, mettre à jour les paramètres de la manière suivante

$$\theta^{t+1} = \theta^t - \eta_t \nabla L(q(\mathbf{x}_i, \theta^t), y_i)$$

  Un tel cycle aléatoire s'appelle une *époque*.

La phase d'initialisation est une phase cruciale dans les applications. Pour la reconnaissance des images, de nombreux réseaux de neurones pré-entrainés existent et atteignent d'excellentes performances de classification une base de données publiques appelée "Imagenet". Les logiciels actuels permettent de récupérer les matrices de coefficients de réseaux pré-entrainés, simplifiant considérablement la phase d'initialisation d'un réseau de neurones destiné à reconnaître des images particulières, proche de celles annotées dans la base. On peut, en quelques heures, entraîner un réseau de neurones à reconnaître des animaux domestiques à partir de quelques milliers d'images. Construire et entraîner un tel réseau sans cette condition initiale peut nécessiter des ressources considérables.

Plutôt que de parcourir les données de l'échantillon les unes après les autres (point 2 de l'algorithme), un compromis entre l'algorithme de gradient et l'algorithme de gradient stochastique consiste à considérer des paquets ("batch") de données lors de la mise à jour des paramètres. Cette technique est plus rapide mais un peu moins précise que le gradient stochastique. La taille du batch devient un hyperparamètre supplémentaire sur lequel agir lors de la phase d'apprentissage.

L'algorithme de gradient stochastique s'accompagne de conditions concernant le taux d'apprentissage. Ce paramètre de l'algorithme peut décroître avec le temps. Une condition classique consiste à choisir η_t tel que

$$\sum_{t=1}^{\infty} \eta_t = \infty \quad \text{et} \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

De nombreuses variantes d'algorithmes d'optimisation sont disponibles dans les programmes de réseaux de neurones. Il est souvent utile de pouvoir en tester plusieurs.

Partie 2 : Reconnaissance de chiffres manuscrits

Dans cet exercice, nous déterminerons les capacités des réseaux de neurones de la bibliothèque *Tensorflow* à correctement reconnaître des chiffres écrits à la main. La reconnaissance de l'écriture manuscrite est un problème difficile, et constitue un test pour les algorithmes d'apprentissage. Cet exemple est historique pour les réseaux de neurones et largement traité sur internet (il vous sera donc facile de savoir si vous faites bien).

Les données de caractères manuscrits sont issues de la base MNIST (Modified National Institute of Standards and Technology). La base de données MNIST est une base de données de chiffres écrits à la main. Elle regroupe 60000 images de chiffres manuscrits et 10000 images de test. Ce sont des matrices carrées ayant 28 pixels de côté, dont les valeurs sont des niveaux de gris allant de 0 à 255.

Lecture des données

Les données sont accessibles depuis la librairie R `keras` qui les télécharge directement depuis l'url de dépot (il faut donc être connecté à internet pour réaliser le TP). Chaque donnée est représentée par un tableau 3d (`array`) contenant les informations (images, width, height) en niveaux de gris. Pour coder le vecteur **x**, nous aurons besoin de vectoriser ces données. Pour ne pas se limiter aux tutoriaux disponibles sur le web, nous nous intéressons à la reconnaissance des chiffres 2 et 7.

- Lire les données MNIST. Il y a 4 ensembles de variables.

```
library(keras)
library(magrittr)
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

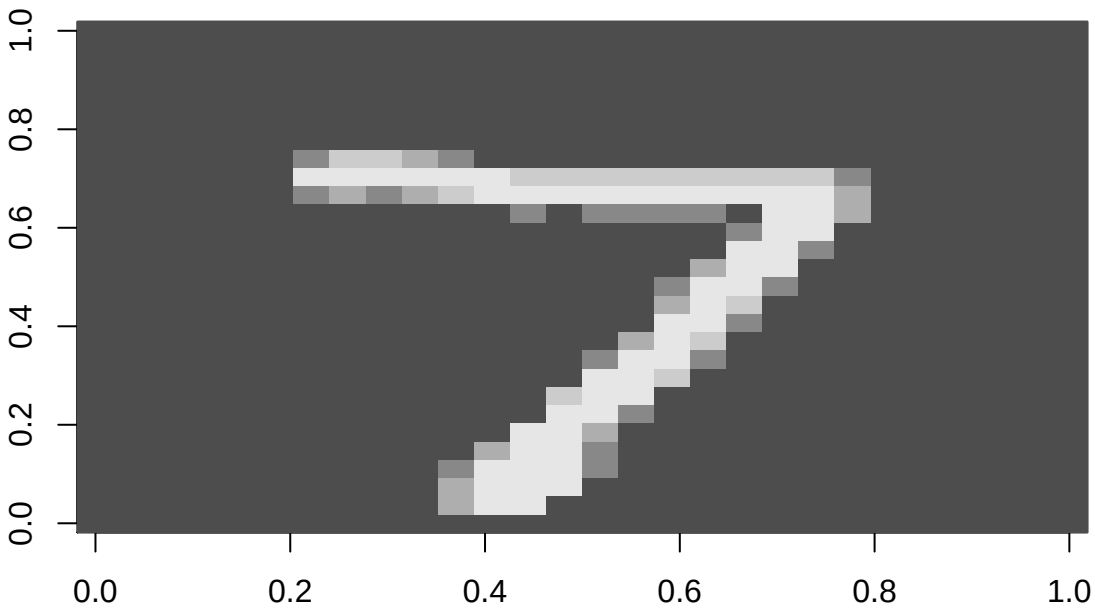
- Filtrer les données correspondant aux chiffres 2 et 7. Commenter et compléter le code suivant. Exécuter le code en changeant l'option de chunck.

```
# comment 1
boo_train <- y_train == change_me | y_train == change_moi
x_train <- mnist$train$x[boo_train,,]
y_train <- mnist$train$y[boo_train]

# comment 2
boo_test <- y_test == change_me | y_test == change_moi
x_test <- mnist$test$x[boo_test,,]
y_test <- mnist$test$y[boo_test]
```

- Utiliser la fonction `image()` pour visualiser le premier chiffre test de la base de données réduite. C'est un 7.

```
image(t(x_test[1, 28:1,]), col = grey.colors(5))
```



- Les images de dimension 28x28 doivent être converties en vecteurs de longueur 784 ($= 28 \times 28$). Cela

peut se faire de plusieurs manières. En particulier, la fonction `array_reshape()` de keras est très utile pour cela.

```
# reshape
  x_train <- array_reshape(x_train, c(nrow(x_train), 784))
  x_test <- array_reshape(x_test, c(nrow(x_test), 784))
```

- Normaliser les données pour obtenir des valeurs réelles (flottants) entre 0 et 1 en divisant les valeurs présentes par 255. Utiliser la fonction `image()` pour visualiser le premier chiffre test de la base de données réduite dans cette nouvelle représentation.

```
# rescale
  x_train <- x_train/255
  x_test <- x_test/255

# le symbol %>% est similaire au 'pipe' d'unix (library(magrittr))
  dim(x_test)
  x_test[1,] %>% matrix(nrow = 28) %>% .[,28:1] %>% image(col = grey.colors(5))
```

- Les données de classe sont des entiers 2,7. Convertir ces données en variables booléennes ou binaires. La valeur 1 ou `TRUE` correspondra à un 7.

```
  y_train <- change_moi
  y_test <-  change_me
```

Ajuster un réseau de neurone avec keras

- Construire un réseau de neurones à deux couches cachées à l'aide de la fonction `keras_model_sequential()`, en faisant varier les paramètres des couches comme ci-dessous. Définir les termes apparaissant dans la construction du modèle.
- Commenter et compléter le code suivant.

```
# commenter les lignes suivantes
  model <- keras_model_sequential()
  model %>%
    layer_dense(units = 256, activation = 'relu', input_shape = 784) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 128, activation = 'relu') %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 1, activation = 'sigmoid')
```

- Donner un tableau de correspondance entre les notions mathématiques introduites dans la section précédente et les termes apparaissant dans la construction du modèle.

```
terme <- c("units = 256", "input_shape")
definition <- c("Dimension de la première couche cachée (W_2)", "A compléter")
data.frame(terme, definition)
```

```
##         terme                                      definition
## 1 units = 256 Dimension de la première couche cachée (W_2)
## 2 input_shape                                    A compléter
```

- Compiler le modèle en précisant la fonction de perte (binary) et demander de visualiser le taux de bonne classification (accuracy). Le choix de l'optimiseur est spécifié.

```
# commenter et compléter les lignes suivantes
  model %>% compile(
```

```
    loss = 'also_sprach_zarathustra',
    optimizer = optimizer_rmsprop(lr = 0.001, rho = 0, decay = 0),
    metrics = c('ex_nihilo_nihil')
)
```

- Donner un tableau de correspondance entre les notions mathématiques introduites dans la section précédente et les termes apparaissant dans la compilation du modèle.

```
terme <- c("loss", "optimizer", "lr", "rho", "decay", "accuracy")
definition <- c("A compléter","","","","","")
data.frame(terme, definition)
```

```
##       terme  definition
## 1      loss A compléter
## 2 optimizer
## 3        lr
## 4       rho
## 5     decay
## 6  accuracy
```

- Allons-y Alonso, pour ajuster le réseau (apprentissage)

```
  history <- carpe_diem %>% fit(
                        x_train,
                        y_train,
                        epochs = 20,
                        batch_size = 128,
                        validation_data = list(x_test, y_test)
)
```

- Donner un tableau de correspondance entre les notions mathématiques introduites dans la section précédente et les termes apparaissant dans l'apprentissage du modèle.
- Quelle est signification des courbes `loss` et `val_loss` que l'on voit tracées.

```
plot(history)
```

- Evaluer le modèle sur les données de test (erreur de classification et perte log loss)

```
model %>% ex_nihilo_nihil(x_test, y_test)
```

- Donner une matrice de confusion pour les classements effectués par le modèle sur le jeu test

```
# help(table)
pred_class <- model %>% predict_classes(x_test)
quod_errat_demonstratum(predicted = pred_class, observed = mnist$test$y[boo_test])
```

- Montrer deux chiffres manuscrits que la machine n'a pas réussi à classer correctement. Donner la liste des mal classés et les probabilités de classement pour chacune des erreurs.

Défi MNIST

Pour les données de l'exercice précédent, répondre aux questions suivantes.

- Reporter dans un tableau 6x2 (`data.frame`), puis dans deux diagrammes en barres (`barplot`), les valeurs des erreurs de classification et de *logloss* obtenues sur l'ensemble test pour 6 réseaux de neurones distincts dont on aura fait varier les paramètres de la manière suivante
- nombre de couches cachées : 1 ou 5,

- nombre de neurones par couche cachée : 10 ou 100,
- valeur de `dropout` par couche cachée : 0.2 ou 0.8.

Inclure dans tableau une colonne indiquant le nom de chaque modele. Par exemple, un modèle à 5 couches cachées, 10 neurones par couche et une valeur "dropout" de 0.2 pourra s'appeler "modele_5_10_2"

```
paste(c("modele", 5, 10, 2), collapse = "_")
```

```
## [1] "modele_5_10_2"
```

- Quel modèle de prédiction vous parait être le meilleur ?
- Donner la liste des chiffres mal classés et les probabilités de classement pour chacune des erreurs.

Partie 3 : Analyse de critiques de films (IMDB)

L'objectif de cette séance est de répondre à un défi de classification portant sur l'analyse d'opinion à partir de documents textuels ("natural language processing").

Les documents analysés sont des critiques de films écrites par des utilisateurs du site web "Internet Movie Data Base" (IMDB). À chaque critique est associée une note donnée par l'utilisateur du site. Seules les notes extrêmes ont été conservées et converties en valeurs binaires représentant des opinions positives ou négatives envers le film.

La base de données comporte 50000 critiques de films. Chaque document est prétraité et représenté sous le format d'un *sac de mots* ("bag of words") pour en faciliter l'analyse. Un sac de mots peut être vu comme une représentation des termes d'un document partir des fréquences d'occurrence des mots dans la base IMDB.

Dans cette séance de TP, nous téléchargerons un échantillon de la base de données et constituerons un sous-échantillon comportant 10000 documents annotés. Chaque document est associé à une évaluation traduisant une opinion positive (valeur 0) ou négative (valeur 1) des utilisateurs.

Le but de ce TP est de prédire le mieux possible l'opinion ou le *sentiment* des utilisateurs à partir des fréquences d'occurrence de certains termes apparaissant dans les textes. Un échantillon de test comportant 5000 critiques indépendantes sera utilisé par l'enseignant pour évaluer la méthode sélectionnée dans le défi final.

```
library(magrittr)
library(keras)
```

Index des termes et sacs de mots.

À chaque terme d'un texte (aussi appelé *document*) est associé une fréquence d'apparition globale dans la base de données IMDB. Les termes sont référencés dans un index à l'aide d'un nombre indiquant leur rang d'apparition dans la base de données. Dans l'index, les termes sont triés du plus fréquent au moins fréquent.

L'index peut être consulté à partir de la bibliothèque `keras`, sous forme de liste dont les attributs sont les termes utilisés dans la base de données. Pour cela, on utlise la fonction `dataset_imdb_word_index()`. L'index est illustré ci-dessous.

Le numéro trouvé dans l'index correspond au rang d'un terme donné. Pour trouver le mot le plus fréquemment utilisé dans l'IMDB, on peut chercher le terme dont la valeur est égale au rang 1. L'index contient aussi quatres valeurs spéciales (start, pad, unknown, unused).

```
  index <- keras::dataset_imdb_word_index()
  names(index[index == 1])
```

```
## [1] "the"
```

Sans surprise, le terme le plus fréquemment utilisé est l'article "the". Sa valeur dans l'index est donc égale à 1.

L'article "the" se retrouve à la position 85976 dans l'index. L'index n'est donc pas ordonné par la fréquence des termes, mais par une permutation. Nous pouvons utiliser la fonction `order()` pour réordonner l'index et trouver les 10 termes les plus utilisés dans l'ensemble des critiques de films.

```
o <- as.numeric(index) %>% order()
index[o[1:10]] %>% names()
```

```
##  [1] "the" "and" "a"   "of"  "to"  "is"  "br"  "in"  "it"  "i"
```

Nous voyons qu'il s'agit d'articles, de prépositions ou de termes non-informatifs, comme par exemple, des éléments extraits des balises html (br). Il sera peut-être préférable d'éliminer les termes les plus utilisés. L'entrée 49 correspondant au terme "good".

Pour réduire le temps de calcul, nous conservons uniquement les 5000 termes les plus fréquents. Dans la suite, chaque document sera représenté par une suite de fréquences de termes représentés par les codes de l'index. Le jeu de données résultant de cette étape est nommé `imbd`.

```
imbd <-  keras::dataset_imdb(path = "imdb.npz",
                             num_words = 5048,
                             skip_top = 49,
                             oov_char = -2,
                             start_char = -1,
                             index_from = 0)
```

Les données enregistrées dans l'objet `imbd` se présentent sous la forme de listes de documents (train et test). On utilisera les "double-crochets" ou le symbole dollar pour accéder aux attributs de ces listes.

```
summary(imbd)
```

```
##       Length Class  Mode
## train 2      -none- list
## test  2      -none- list
```

Par exemple le document numéro 12 dans l'ensemble d'apprentissage est lu de la manière suivante.

```
document <- imbd$train$x[[12]]
document
```

```
##  [1]   -2   51   -2 1607   -2   -2   -2   66   52  361 1395   -2   51   -2
## [15]  216   -2   -2 1703   -2   -2   -2   -2  326   -2  173  326   71   -2
## [29]   -2  870   -2  153   68   75   -2   -2  319   -2   -2   -2   -2  246
## [43]   -2   62   -2   -2  376   -2   97  154   -2   -2  907   -2  546   -2
## [57]   -2 1493   -2   -2   -2   -2   -2   -2  209   -2   -2   -2 1319  988
## [71]   -2 2999   -2  422   -2   70 2215  546   -2   -2  152   -2   97  760
## [85]  376   -2  100  348   -2   -2  199   -2 2238   -2   -2  317   -2   -2
## [99]  454
```

Nous voyons que le terme d'indice -2 est fréquent. Après filtrage, il correspond en fait à la chaîne de caractères **UNK** signifiant unknown. Cette chaîne apparait car nous nous sommes restreint à un dictionnaire de 5000 mots.

Les fréquences des termes apparaissant dans le document 12 peuvent être données par la fonction `table()`. Cette fonction utilise l'ordre alphabétique pour représenter les comptages de chaque terme.

```
table(sapply(document[document > - 2], FUN = function(x) names(index[index == x])))
```

```
##
##            0            5       actors          add        after
##            1            1            1            1            1
```

```
##          bad       chosen       comedy        could         dead
##            1            1            1            2            1
##       disney       except expectations     expected         give
##            1            2            1            1            1
##          had         half    laughable         less          lot
##            1            1            1            2            1
##          low  masterpiece      murders     realized       rented
##            1            1            1            1            1
##          saw        story       stupid    stupidity         than
##            1            1            2            1            1
##        thing        title         very         well         were
##            1            1            1            1            1
##         when         wife         work        worst
##            2            1            1            1
```

Le document 12 contient des termes plutôt négatifs. Il est associé à une opinion négative (valeur 0). Cela se vérifie en affichant la variable y.

```
imbd$train$y[12]
```

```
## [1] 0
```

Pour la suite, c'est à vous de jouer. Pour ne pas entrer dans les subtilités du traitement du langage, nous nous appuierons sur un codage simplifié des textes basé sur les fréquences des mots de l'index (one-hot encoding). Pour un encodage par 'embeddings', une méthode particulièrement dédiée à l'analyse de texte, on peut trouver des solutions sur le web.

Défi "analyse de sentiments"

Lecture des données

- Ecrire une ligne de commande R permettant convertir le document 12 en un vecteur de longueur 5000, indiquant le nombre d'apparition de chacun des indices allant de 1 à 5000 dans ce document.

```
# comment 1
help(sapply)
result <- sapply(49:5048, FUN = function(x) cogito_ergo_sum)
result[1:100]
```

- Constituer un jeu de données comportant 10000 documents choisis pour moitié dans l'ensemble "train" et pour moitié dans l'ensemble "test" de l'IMBD. Techniquement nous le constituerons en 20 étapes, pour limiter l'impact sur la mémoire. Commenter et exécuter le code suivant

```
x_imbd <- NULL

  for (i in 1:10){

    x_imbd_500 <- NULL

      for (j in (500*(i-1)+1):(500*i)){

        # comment 1

        doc_temp <- imbd$train$x[[j]]
        x_imbd_500 <- rbind(x_imbd_500,
                            sapply(49:5048,
```

```
                                FUN = function(ind) sum(doc_temp == ind)))

        if (j%%500 == 0) print(j) # ca rassure
    }
    x_imbd <- rbind(x_imbd, x_imbd_500)
  }

for (i in 1:10){

    x_imbd_500 <- NULL

      for (j in (500*(i-1)+1):(500*i)){

        # comment 2

        doc_temp <- imbd$test$x[[j]]
        x_imbd_500 <- rbind(x_imbd_500,
                            sapply(49:5048,
                                   FUN = function(ind) sum(doc_temp == ind)))

        if (j%%500 == 0) print(j) # ca rassure
    }
    x_imbd <- rbind(x_imbd, x_imbd_500)
  }
```

- Que contient l'objet `x_imbd` ?
- Définir les classes $y = 0$ ou $y = 1$ pour chaque élément de `x_imbd`

```
y_imbd <- c(imbd$train$y[1:5000], imbd$test$y[1:5000])
```

Et voilà. On est en pleine forme et on dispose d'une base d'apprentissage comportant les fréquences d'apparition des mots de l'index pour 10000 documents (`x_imbd`) et les opinions des utilisateurs `y_imbd`. Le défi peut vraiment commencer.

Etude d'association

Le but d'une étude d'association est d'identifier les termes les plus associés aux opinions positives ou négatives des utilisateurs. Pour cela, nous evaluons la correlation au carré entre l'occurrence de chaque terme et l'opinion de l'utilisateur (présence d'un 1). Il se peut que certaines valeurs de corrélation ne soient pas calculables à cause d'un écart-type nul.

- Calculer le coefficient de corrélation au carré entre les fréquences d'apparition des termes de l'index et opinion des utilsateurs (5000 valeurs).

```
# comment
x <- x_imbd

# comment
y <- y_imbd

# comment
r <- find_me
r2 <- r^2
```

- Montrer les termes dont la valeur d'association r^2 est supérieure à 3 pour cent (0.02), puis supérieure à

0.02, et à 0.01. *Note* : Il faut effectuer un décalage de 48 indices dans l'index pour trouver le codage correct.

```
  # quelque chose à changer
  index[o[ which(r2 > 0.02) + 48 ]] %>% names()
```

- Dans quelles proportions les termes de valeur d'association r^2 supérieure à 0.02 apparaissent-ils dans les documents ? Représenter graphiquement ces proportions à l'aide d'un diagramme en barre.

```
# Calculer la frequence des termes realisant la condition
  freq <- x[, conditio_logicae ] %>% apply(2, mean)

# mots dans l'index et barplot
  names(freq) <-  index[o[which(r2 > 0.02) + 48]] %>% names()
  barplot(sort(freq, decreasing = TRUE), col = "lightblue", las = 2)
```

- Dans quelles proportions les termes de valeur d'association r^2 supérieure à 0.02 apparaissent-ils dans les documents ? Représenter graphiquement ces proportions à l'aide d'un diagramme en barre.

```
  yes_we_can
```

- Dans quelles proportions les termes de valeur d'association $r = cor(x, y)$ supérieure à 0.1 apparaissent-ils dans les documents ? Représenter graphiquement ces proportions à l'aide d'un diagramme en barre.

```
  yes_we_can
  barplot(sort(freq, decreasing = TRUE), col = "pink", las = 2)
```

- Dans quelles proportions les termes de valeur d'association r inférieure à -0.1 apparaissent-ils dans les documents ? Représenter graphiquement ces proportions à l'aide d'un diagramme en barre.

```
  i_have_a_dream
  barplot(sort(freq, decreasing = TRUE), col = "palegreen", las = 2)
```

Modèles d'apprentissage

- 'A l'aide des outils de *keras*, ajuster des modèles d'apprentissage aux données contenues dans le défi : "x_imbd" et "y_imbd". Considérer la graine du générateur aléatoire `seed` comme un hyper-paramètre (`set.seed(seed)`).
- Dans un tableau, décrire les performances de 6 méthodes choisies pour des échantillons d'apprentissage et de test que vous aurez créés vous-mêmes à partir des données "x_imbd" et "y_imbd". Les performances seront mesurées par les erreurs de classification et d'entropie croisée (log loss).
- Donner le code R correspondant au meilleur modèle que vous avez ajusté (chunk ci-dessous). On considèrera la graine du générateur aléatoire comme un hyperparamètre supplémentaire du modèle. Donner la valeur finale d'entropie croisée estimée sur votre ensemble test. On fera en sorte que le résulat soit complètement reproductible à partir du code proposé.

```
# code de mon meilleur modèle
# Ne pas oublier d'inclure la seed du générateur aléatoire
# set.seed(seed = 5427381)
  model <-  deus_ex_machina
```

Règles de rendu

- Ce projet donne lieu à un compte rendu séparé, dans lequel on aura pris soin de reporter les sections de ce document comportant vos réponses, commentaires, et vos codes complétés. Merci de ne pas reporter les sections "théoriques".

- Les codes R doivent être inclus dans le texte du compte-rendu (menu **Insert**) et l'ensemble doit être commenté avec précision. **Les commentaires compteront pour une part importante dans la note**.
- Le compte-rendu doit être déposé **dans TEIDE**.
- Le compte-rendu doit être déposé **au format HTML uniquement** et intitulé "Rendu_Projet_1.html". Utiliser la fonction **knitr** du menu de rstudio pour obtenir le document au format souhaité. **Les fichiers sources (Rmd ou nb.html) ne seront pas acceptés**.