Documentation

Rougetet Arnaud, Beauprez Ellie

Dépôt Github : https://github.com/Arnaud58/test-tensorflow

Ce document est un complément du rapport que nous avons rédigé.
À noter que la documentation détaillée des fonctions se trouve dans le dossier *doc* sur le dépôt Github.

Le projet utilise du JS pur et ne demande aucune installation particulière pour fonctionner, à part un navigateur internet. Il est recommandé d'avoir "Chrome" et "Firefox" d'installés. Certaines fonctionnalités peuvent ne marcher qu'avec Chrome (comme par exemple la manipulation des fichiers json).

Les librairies JS utilisé sont :

- TensorFlow Version 0.11.1
- TensorFlow Vis
- P5
- Material Design

Voir code :

```
<!-- TensorFlow et  TensorFlow-Vis -->
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.11.1">
</script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis"></script>

<!-- P5 -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.6.0/p5.js">
</script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.6.0/addons/p5.dom.js">
</script>

<!-- Material Design -->
<link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.green-
amber.min.css">
<script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?
family=Material+Icons">
```

Des tutos de tensorFlow et des exemples peuvent être trouvés ici :

- Vidéo Youtube sur le tensorFlow
- Code correspondant à la vidéo

Le fichier index.html utilise 5 fichier JS qui permette de créer son propre réseau neuronal et de lui faire apprendre des positions de rectangle.

La documentation des 5 fichier JS peut être trouvé ici :

- Sketch.js Implémente la fonction Setup de p5 qui ai lancé au chargement de la page. Ce fichier gère l'initialisation de la page et l'affichage des résultat sur le canvas.
- Params.js Gère les paramètre rentré dans l'onglet "Param" de la page. Les paramètres permette de géré ce que l'on prend en compte pour apprentissage : la taille, les liens, la couleur du rectangle ou une combinaison linéaire des 3.
- Neural.js qui permet de géré la création d'un réseau de neurone. La création se fait en fonction des paramètre mis dans l'onglet "Neural". On peut réglé le nombre de couche, le nombre de neurone par couche et la fonction d'activation dans chaque couche.
- Training.js Gère l'entraînement du réseau en lui donnant des exemples de rectangle et leur placement. On peut lui en donner un particulier en lui précisant sa taille, sa couleur et son nombre de lien ou en géré automatiquement en continue.
- Predict.js Gère la prédiction via le modèle de certain rectangle. On peut donné un rectangle particulier en lui précisant sa taille, sa couleur et son nombre de lien et le modèle tente de prédire sa place sur le canvas. On peut aussi prédire tous les rectangles d'entraînement. Donne alors un % de réussite.

Utilisation de TensorFlow.js

Configuration du réseau neuronal

Dans cette partie nous expliquons comment fonctionne la création du réseau neuronal et ce à quoi correspondent les paramètres choisis pour sa configuration.

- **Initialisation**
  L'initialisation du modèle se fait de cette manière : `let model = tf.sequential();`
  Le mode "*sequential*" est le plus facile à manipuler. Cela fonctionne comme une suite linéaire de couches dont les sorties de l'une sont reliées aux entrées de la suivante. Une autre possibilité est d'utiliser `tf.model(layers)` qui offre plus de contrôle et permets de faire des branchements plus personnalisés entre les couches.

- **Création des couches cachées**

```
let layer = tf.layers.dense({
    inputShape: [2], //nombre de paramètres d'entrée, seulement requis pour
la 1ère couche cachée
    units: nbNeurons, //nombre de neurones de cette couche
    activation: activationFunction //fonction d'activation utilisée
});
model.add(firstHiddenLayer); //ajoute la couche au modèle
```

Rq : *dense()* signifie que c'est une couche entièrement connectée.

- **Compilation du modèle**
  Cette étape sert à préparer le modèle pour l'entraînement et l'évaluation.

```
model.compile({
    optimizer: 'sgd',
    loss: 'meanSquaredError',
    lr: learningRate
});
```

optimizer : Méthode de minimisation d'erreur utilisée. Ici, nous utilisons *sgd* pour la *descente de gradient stochastique*.
loss : fonction utilisée pour le calcul de l'erreur. Ici nous utilisons *meanSquaredError*.
lr : Learning rate. Représente un coefficient lors de la minimisation pour déterminer quel sera le point suivant. Avec un learning rate petit, l'algorithme sera plus lent (car on fait des plus petits pas) mais sera plus précis pour trouver le minimum de la fonction d'erreur. Avec un learning rate grand, le calcul sera plus rapide mais potentiellement moins précis car avec des pas trop grands on risque de rater le minimum.

Description de l'interface réalisée

Branche "master"

Onglet *Neural* :
Une interface permet de paramétrer le réseau neuronal (nb de couches, et pour chacune de ces couches le nb de neurones et la fonction d'activation, learning rate)

Onglet *Param* :
Permet de cocher les paramètres que l'on veut prendre en compte pour la classification (taille, nb de liens, couleurs). Fonctionnel, que l'on choisisse la taille, les couleurs, les liens ou plusieurs paramètres en même temps.
Pour pouvoir tester notre réseau, nous avons choisi arbitrairement une classification qui découpe l'aire de travail en plusieurs zones, selon les paramètres choisis.
Par exemple, lorsque l'on choisi la taille et la couleur comme paramètres, les grands rectangles roses doivent être placés dans la 1ère zone.
Pour simplifier le problème, lorsque qu'on coche la taille, la couleur et le nombre de liens en même temps, le nombre de liens devient prioritaire par rapport à la taille.
Remarque : Les rectangles avec un grand nombre de liens (>10) seront affichés avec un contour vert.

Onglet *Learning* :
Interaction avec des boutons permettant de décider le nombre de rectangles qui vont servir pour l'apprentissage et décider des données d'entrée pour la prédiction.

- *Add one learn square*. Ajoute un seul rectangle à l'apprentissage du modèle
- *Learn from a file*. Permet de charger des données à partir d'un fichier et de faire l'apprentissage (pas encore fonctionnel)
- *Ajout auto*. Ajoute un rectangle à l'apprentissage du modèle par seconde.
- *Save*. Permet d'enregistrer les données générées dans un fichier json (ne marche pas sur Firefox, mais fonctionne sur Chrome)

Onglet *Predict* :

- *Predict*. Essaye de prédire la position d'un rectangle et l'affiche à droite. (hauteur et largeur doivent être < 400 !)
- *Predict the test*. Envoie les données d'apprentissage sur la prédiction et affiche le pourcentage de réussite.
- *Predict from a file*. Permet d'afficher les prédictions à partir de données stockées dans un fichier json

Onglet *Save and Load Model* : (**REMARQUE :** Ne fonctionne qu'avec Google Chrome et Chromium)

- *Save Model*. Télécharge la configuration du modèle et les poids dans deux fichiers `my-model-1.json` et `my-model-1.weights.bin`
- *Upload Json Model* et *Upload weights*. Charge le fichier json et le .bin contenant le modèle et les poids (de la même forme que ceux téléchargeables via *Save Model*)
- *Load Model*. Charge le modèle après que l'on ait préalablement chargé les fichiers requis via les deux boutons précédents.

Onglet *Graph* :
Affiche le graphe d'erreurs et la matrice de confusion

Branche "interactive"

Dans cette branche, un onglet **Square** a été ajouté où l'utilisateur peut créer des rectangles lui-même et ensuite les déplacer à sa guise sur le canvas.
Remarque : Cette fois, le nombre de liens est affiché sur les rectangles (alors que dans la branche master, on avait préféré ne pas l'afficher et simplement colorer le contour du rectangle en vert lorsqu'il est "fortement lié" pour des soucis de visibilité lorsqu'on a beaucoup de rectangles qui s'empilent).

Détails techniques

Classification

Comme expliqué dans le rapport, nous avons dû pour des raisons pratiques fixer nous même la façon dont sont classifiés les rectangles selon les paramètres pris en compte.
Ci-dessous sont décrits les types de classifications choisis pour chaque combinaison de paramètres.
Par défaut, un rectangle est considéré comme "grand" si son aire dépasse 30000 pixels, et il est considéré comme "fortement lié" si son nombre de liens dépasse 10. Ces valeurs peuvent être modifiées via les variables *areaLimit* et *links_max* dans sketch.js

```
TAILLE :
*-------*
|   0   |
*-------*
|   1   |
*-------*
0 : Grand rectangle
1 : Petit rectangle

COULEUR :
*-------*-------*-------*
```

```
|   0   |   1   |   2   |
*-------*-------*-------*
0 : rose
1 : jaune et orange
2 : bleu et vert

COULEUR + TAILLE :
*-------*-------*-------*
|   0   |   2   |   4   |
| (0,0) | (1,0) | (2,0) |
*-------*-------*-------*
|   1   |   3   |   5   |
| (0,1) | (1,1) | (2,1) |
*-------*-------*-------*
(x,0) : Grand rectangle
(x,1) : Petit rectangle
(0,x) : rose
(1,x) : jaune et orange
(2,x) : bleu et vert
Exemple : un petit rectangle rose sera dans la zone 1, un grand rectangle
jaune sera dans la zone 2.


TAILLE + NB LIENS
*-------*-------*
|   0   |   2   |
| (0,0) | (1,0) |
*-------*-------*
|   1   |   3   |
| (0,1) | (1,1) |
*-------*-------*
(x,0) : Bcp de liens
(x,1) : Peu de liens
(0,x) : Grands rectangles
(1,x) : Petits rectangles


COULEUR + NB LIENS
*-------*-------*-------*
|   0   |   2   |   4   |
| (0,0) | (1,0) | (2,0) |
*-------*-------*-------*
|   1   |   3   |   5   |
| (0,1) | (1,1) | (2,1) |
*-------*-------*-------*
(x,0) : Bcp de liens
(x,1) : Peu de liens
(0,x) : rose
(1,x) : jaune et orange
(2,x) : bleu et vert

TAILLE + COULEUR + NB DE LIENS
Dans ce cas, le nb de liens devient prioritaire par rapport et la taille ne
```

```
compte plus
Même classification que pour COULEUR + LIENS
```

Documentation des différents fichiers js dans l'ordre :

Neural.js

addLayer()

> Fonction déclenchée lorsque l'on demande l'ajout d'une couche supplémentaire. Ajoute une couche paramétrable sur l'interface graphique.

removeLayer()

> Fonction déclenchée lorsque l'on demande la suppression d'une couche. Supprime la dernière couche paramétrable dans le tableau de l'interface graphique de l'onglet Neural.

getNetworksParam()

> Récupère les paramètres de configuration choisis par l'utilisateur

saveModel()

> Sauvegarde le modèle et le télécharge dans deux fichiers nommés "my-model-1.json" et "my-model-1.weights.bin"

loadModelFromFiles()

> Charge le modèle à partir des fichiers donnés. Ces fichiers doivent correspondre aux mêmes types que ceux renvoyés par model.save

addLayer()

Fonction déclenchée lorsque l'on demande l'ajout d'une couche supplémentaire. Ajoute une couche paramétrable sur l'interface graphique.

Kind: global function

removeLayer()

Fonction déclenchée lorsque l'on demande la suppression d'une couche. Supprime la dernière couche paramétrable dans le tableau de l'interface graphique de l'onglet Neural.

Kind: global function

getNetworksParam()

Récupère les paramètres de configuration choisis par l'utilisateur

Kind: global function

saveModel()

Sauvegarde le modèle et le télécharge dans deux fichiers nommés "my-model-1.json" et "my-model-1.weights.bin"

Kind: global function

loadModelFromFiles()

Charge le modèle à partir des fichiers donnés. Ces fichiers doivent correspondre aux mêmes types que ceux renvoyés par model.save

Kind: global function

Params.js

checkActiveParams()

Récupère les paramètres cochés dans l'onglet Params et calcule le nombre de zones de classification nécessaires en conséquence.

setNbZones()

Retourne le nombre nécessaire de zones pour la classification, en se basant sur les paramètres activés par l'utilisateur.

setNbZonesXY()

Met à jour les variables xZones et yZones en leur donnant les nombres de zones selon les axes horizontal et vertical.

sliceInZones(height, width, nombre, nombre)

Découpe l'aire de travail en plusieurs zones pour permettre la classification

computeNbinputShape()

Calcule la taille de l'entrée du réseau neuronal en fonction des paramètres actifs

generateTensorFor1Square()

Génère un tenseur représentant le rectangles dont les caractéristiques sont données en arguments en fonction des paramètres actifs pour l'apprentissage

generateTensorForAllSquare()

Génère un tenseur représentant les données pour tous les rectangles en fonction des paramètres actifs pour l'apprentissage

expectedZone(height, width, color, nbLinks)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques. On calcule ici en quelque sorte les labels correspondants aux données générées.

expectedZoneScale(height, width)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est la taille

expectedZoneLinks(nbLinks)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est le nombre de liens

expectedZoneColor(color)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est la couleur

expectedZoneScaleColor(height, width, color)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la taille et la couleur

expectedZoneScaleLinks(height, width, nbLinks)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la taille et le nombre de liens

expectedZoneColorLinks(color, nbLinks)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la couleur et le nombre de liens

vectorFromExpectedZone(zoneExpected)

Retourne un vecteur correspondant au résultat attendu en sortie du réseau neuronal en fonction des paramètres actifs et de la zone attendue Exemple : Si les paramètres actifs sont la taille et la couleur et qu'un rectangle doit être placé dans la zone 2, alors vectorFromExpectedZone(2) renvoie [0,0,1,0,0,0]

checkActiveParams()

Récupère les paramètres cochés dans l'onglet Params et calcule le nombre de zones de classification nécessaires en conséquence.

Kind: global function

setNbZones()

Retourne le nombre nécessaire de zones pour la classification, en se basant sur les paramètres activés par l'utilisateur.

Kind: global function

setNbZonesXY()

Met à jour les variables xZones et yZones en leur donnant les nombres de zones selon les axes horizontal et vertical.

Kind: global function

sliceInZones(height, width, nombre, nombre)

Découpe l'aire de travail en plusieurs zones pour permettre la classification

Kind: global function

Param	Type	Description
height	`int`	hauteur de la zone à découper
width	`int`	largeur de la zone à découper
nombre	`xZones`	de zones voulue sur l'axe horizontal
nombre	`yZones`	de zones voulue sur l'axe vertical

computeNbinputShape()

Calcule la taille de l'entrée du réseau neuronal en fonction des paramètres actifs

Kind: global function

generateTensorFor1Square()

Génère un tenseur représentant le rectangles dont les caractéristiques sont données en arguments en fonction des paramètres actifs pour l'apprentissage

Kind: global function

generateTensorForAllSquare()

Génère un tenseur représentant les données pour tous les rectangles en fonction des paramètres actifs pour l'apprentissage

Kind: global function

expectedZone(height, width, color, nbLinks)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques. On calcule ici en quelque sorte les labels correspondants aux données générées.

Kind: global function

Param	Type	Description
height	`int`	la hauteur du rectangle

Param	Type	Description
width	`int`	la largeur du rectangle
color	`Array.<int>`	valeurs RGB de la couleur du rectangle
nbLinks	`int`	le nombre de liens associé au rectangle

expectedZoneScale(height, width)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est la taille

Kind: global function

Param	Type	Description
height	`int`	la hauteur du rectangle
width	`int`	la largeur du rectangle

expectedZoneLinks(nbLinks)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est le nombre de liens

Kind: global function

Param	Type	Description
nbLinks	`int`	le nombre de liens associé au rectangle

expectedZoneColor(color)

Retourne le numéro de la zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où le seule paramètre actif est la couleur

Kind: global function

Param	Type	Description
color	`Array.<int>`	valeurs RGB de la couleur du rectangle

expectedZoneScaleColor(height, width, color)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la taille et la couleur

Kind: global function

Param	Type	Description
height	`int`	la hauteur du rectangle

Param	Type	Description
width	`int`	la largeur du rectangle
color	`Array.<int>`	valeurs RGB de la couleur du rectangle

expectedZoneScaleLinks(height, width, nbLinks)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la taille et le nombre de liens

Kind: global function

Param	Type	Description
height	`int`	la hauteur du rectangle
width	`int`	la largeur du rectangle
nbLinks	`int`	le nombre de liens associé au rectangle

expectedZoneColorLinks(color, nbLinks)

Retourne le numéro zone correspondant à un rectangle en fonction de ses caractéristiques dans le cas où les paramètres actifs sont la couleur et le nombre de liens

Kind: global function

Param	Type	Description
color	`Array.<int>`	valeurs RGB de la couleur du rectangle
nbLinks	`int`	le nombre de liens associé au rectangle

vectorFromExpectedZone(zoneExpected)

Retourne un vecteur correspondant au résultat attendu en sortie du réseau neuronal en fonction des paramètres actifs et de la zone attendue Exemple : Si les paramètres actifs sont la taille et la couleur et qu'un rectangle doit être placé dans la zone 2, alors vectorFromExpectedZone(2) renvoie [0,0,1,0,0,0]

Kind: global function

Param	Type	Description
zoneExpected	`int`	le numéro de la zone attendue

predict.js

predictFromUser()

Fonction qui ajoute aux tableau à prédire la séléction de l'utilisateur

checkResZone(resArray) ⇒

Retourne le numéro de la zone obtenue en résultat en fonction du tableau retourné par le réseau neuronal

predictTheTests()

Prédit les donnée d'apprentisage (all_squares_learn.squareLearn) et les affiches sur la partie droite du canvas Donne un % de réussite des prédictions

predictAndDisplay(lgr, htr, color, link) ⇒ `Array.<Array>`

Predie le rectangle donné en paramètre Renvoie le résultat de la prédiction (voir model.predict()) et si la prédiction est corrécte ou mauvaise

loadAndPredict()

Charge un fichier JSON et fait la prédiction sur les données qu'il contient

predictFromUser()

Fonction qui ajoute aux tableau à prédire la séléction de l'utilisateur

Kind: global function

checkResZone(resArray) ⇒

Retourne le numéro de la zone obtenue en résultat en fonction du tableau retourné par le réseau neuronal

Kind: global function
Returns: le numéro de la zone correpondante

Param	Type	Description
resArray	`Array.<float>`	le tableau contenant les résultats retournés par TensorFlow

predictTheTests()

Prédit les donnée d'apprentisage (all_squares_learn.squareLearn) et les affiches sur la partie droite du canvas Donne un % de réussite des prédictions

Kind: global function

predictAndDisplay(lgr, htr, color, link) ⇒ `Array.<Array>`

Predie le rectangle donné en paramètre Renvoie le résultat de la prédiction (voir model.predict()) et si la prédiction est corrécte ou mauvaise

Kind: global function
Returns: `Array.<Array>` - Le tableau contient un tableau qui représente le tensor de la prédiction et un boolen, le boolen vaux Vrai si la prédiction est mauvaise et Faux sinon

Param	Type	Description

Param	Type	Description
lgr	`int`	La largeur du rectangle
htr	`int`	La hauteur du rectangle
color	`Array.<int>`	tableau contenant les valeurs RGB de la couleur
link	`int`	nombre de liens associé au rectangle

loadAndPredict()

Charge un fichier JSON et fait la prédiction sur les données qu'il contient

Kind: global function

sketch.js

download(content, fileName, contentType)

Sert à télécharger une variable sur son bureau Ne fonctionne pas sous firefox

textToUser(msg)

Envoie un message à l'utilisateur dans une div en bas de son éran

addToDisplayLearn(l, h)

Ajoute un rectangle sur la partie gauche du canvas

download(content, fileName, contentType)

Sert à télécharger une variable sur son bureau Ne fonctionne pas sous firefox

Kind: global function

Param	Type	Default	Description
content	`any`		La variable à télécharger
fileName	`String`		Le nom du fichier que l'on va télécharger
contentType	`String`	`application/json`	Le format sous lequel on veux le télécharger (JSON par défault)

textToUser(msg)

Envoie un message à l'utilisateur dans une div en bas de son éran

Kind: global function

Param	Type	Description
msg	`String`	Le message à afficher

addToDisplayLearn(l, h)

Ajoute un rectangle sur la partie gauche du canvas

Kind: global function

Param	Type
l	`int`
h	`int`

training.js

addSquare()

> Ajoute un rectangle aux données d'apprentissage, et réalise l'apprentissage avec le modèle et les données actuelles

trainSquare(l, h, color, link)

> Entraîne le modèle à partir du rectangle dont les caractéristiques sont données en paramètres. Le retour de la fonction d'entraînement est stocké dans une promesse dans la variable oldHistory

trainAllSquares()

> Entraîne le modèle avec tous les rectangles dont les données sont contenues dans all_squares_learn Le retour de la fonction d'entraînement est stocké dans une promesse dans la variable oldHistory

loadAndTrain()

> Charge les données depuis un fichier JSON et entraîne le modèle à partir des données ainsi récupérées

addSquare()

Ajoute un rectangle aux données d'apprentissage, et réalise l'apprentissage avec le modèle et les données actuelles

Kind: global function

trainSquare(l, h, color, link)

Entraîne le modèle à partir du rectangle dont les caractéristiques sont données en paramètres. Le retour de la fonction d'entraînement est stocké dans une promesse dans la variable oldHistory

Kind: global function

Param	Type	Description
l	`int`	la largeur normalisée (avec une valeur comprise entre 0 et 1)
h	`int`	la hauteur normalisée

Param	Type	Description
color	`Array.<int>`	tableau contenant les valeurs RBG de la couleur
link	`int`	nombre de liens associé au rectangle

trainAllSquares()

Entraîne le modèle avec tous les rectangles dont les données sont contenues dans all_squares_learn Le retour de la fonction d'entraînement est stocké dans une promesse dans la variable oldHistory

Kind: global function

loadAndTrain()

Charge les données depuis un fichier JSON et entraîne le modèle à partir des données ainsi récupérées

Kind: global function