RAPPORT DE TP

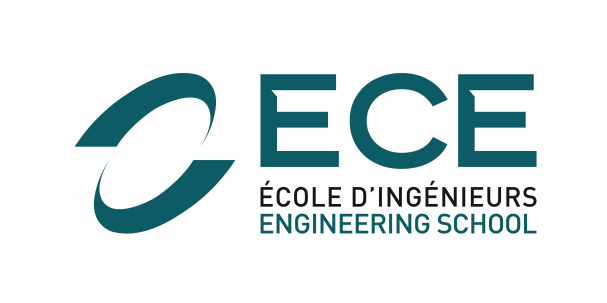
TP 6 : Apprentissage Profond

Auteurs :

Thomas DUCLOS
Hugo VIDAL
Sarah ABERGEL

Enseignant :

M. DERRAZ



Nous attestons que ce travail est original, qu'il est le fruit d'un travail commun au trinôme et qu'il a été rédigé de manière autonome.

Paris, le 05/04/2023

Table des matières

Introduction

Le but de ce TP est de mettre en place un réseau de neurones reccurent (RNN) et convolutionnel (CNN) simple afin de se familiariser avec ces modèles d'apprentissage profond et la façon de les entrainer : la backpropagation du gradient.

Liste des abréviations

— **RNN** : Réseau de Neurones Récurrents (Recurrent Neural Network en anglais)
— **CNN** : Réseau de Neurones Convolutifs (Convolutional Neura Network en anglais)
— **NN** : Réseau de Neurones (Neural Network en anglais)

Exercice II - RNN simple pour Arduino

A - But de l'exercice

Le but de l'exercice est de simuler un réseau de neurones récurrents (**RNN**) pour **prédire** une séquence de sortie en fonction d'une séquence d'entrée. Le **RNN** est composé de trois couches de neurones : une couche d'entrée, une couche cachée et une couche de sortie. Les poids et les biais de chaque couche sont initialisés aléatoirement à l'aide de la fonction d'initialisation des poids. Les sorties sont calculés par l'algorithme de **"propagation avant"** de l'entrée à travers le réseau pour calculer la sortie en utilisant la fonction d'activation **sigmoid**. Le seuil de prédiction de sortie est donné par la valeur lu du d'une fonction **valpot** et ajuste le seuil de prédiction en conséquence. La prédiction est initialisé par la séquence d'entrée avec des valeurs aléatoires et démarre la séquence de prédiction. Dans la boucle **loop()**, la séquence d'entrée est avancée à chaque itération et la séquence de sortie est prédite. La valeur prédite est affichée sur la **LED interne** de l'arduino Due en fonction du seuil de prédiction.

Le code de cet exercice est fourni en annexe.

B - Réponse à l'exercice

1. **Quel est le rôle de la fonction initializeWeights ?**

   Le rôle de la fonction initializeWeights est d'intialiser les valeurs des poids et des biais contenues dans les différentes matrices **weightsIH**, **weightsHH**, **weightsHO**, **biasH** et **biasO**. Les valeurs utilisées pour initialiser les différentes valeurs sont des valeurs aléatoires comprises entre 0 et 1.

2. **Comment est calculé l'état caché (hidden state) dans la fonction forwardPass ?**

   La fonction **forwardPass** calcule l'état caché en effectuant une série d'opérations de calculs sur les entrées fournies dans le tableau d'entrées ainsi que sur les différents biais et poids du réseaux stockés globalement.
   Plus précisément, la fonction calcule l'état caché en effectuant les opérations suivantes :

   (a) Initialiser les valeurs de l'état caché à 0.

   (b) Pour chaque neurone d'entrée, multiplier la valeur de l'entrée correspondante par le poids correspondant entre l'entrée et le neurone caché.

   (c) Pour chaque neurone caché, multiplier la valeur de son état actuel par le poids correspondant entre ce neurone et le neurone caché considéré.

   (d) Ajouter la somme de ces produits pondérés à la valeur du biais du neurone caché considéré.

   (e) Appliquer une fonction d'activation à la somme pondérée obtenue pour obtenir la valeur finale de l'état caché pour le neurone considéré.

Ces étapes sont effectuées pour chaque neurone caché dans le réseau, ce qui permet de calculer l'état caché complet du réseau de neurones.

3. **Quel est l'avantage de l'utilisation de la fonction d'activation sigmoid pour calculer les sorties du réseau de neurones ?**

   Les avantages de l'utilisation de la fonction **sigmoïd** en tant que fonction d'activation est qu'elle possède une plage de sortie entre 0 et 1, ce qui est particulièrement adapté pour des problèmes de classification binaire, notre cas d'étude nécéssite une sortie binaire 0 ou 1 pour l'état de la LED.

4. **A quoi sert la fonction readPoten et comment est-elle utilisée pour ajuster le seuil de prédiction ?**

   La fonction **readPoten** permet de lire la valeur du potentiomètre. Par la suite elle map cette valeur entre 0 et 1 et set le treshold à cette valeur.
   Le treshold permet de considérer si un output est une valeur positive ou négative. Dans notre cas positive représente allumer la LED et négative représente éteindre la LED.

5. **A quoi sert la fonction startPrediction et comment est-elle utilisée pour calculer la prédiction ?**

   La fonction startPrediction initialise les valeurs du tableau d'entrée entre 0 et 1. Par la suite, elle met la valeur de la variable **predict** à la valeur **true**. Cette valeur de la variable **predict** permet d'indiquer à la boucle **loop** du programme de commencer la prédiction du réseau sur le tableau d'entrée.
   En résumé cette fonction possède à la fois un rôle d'initialisation mais également d'initiatrice du réseau de neurone.

6. **Expliquer comment le RNN calcule la sortie avec la fonction forwardPass et comment on peut faire pour étendre le nombre des cellules RNN ?**

   Le **RNN** utilise la fonction **forwardPass** pour calculer la **sortie** du réseau en focntion de **l'état caché** et des **entrées**. Cette fonction prend en entrée un **tableau d'entrée**, un **tableau de sortie** et un **tableau d'état caché**. La fonction calcule l'**état caché** en utilisant les **entrées** et l'état **caché** de la couche précédente, en utilisant les **poids** et les **biais**, puis passe cette somme pondérée à une **fonction d'activation** pour obtenir l'état caché de la cellule. Ceci est répété pour chaque cellule de la couche cachée.
   Ensuite la fonction **forwardPass** calcule la séquence de sortie en bouclant sur les **cellules de sortie** du **RNN**. Pour chaque cellule de sortie, la fonction calcule la **somme pondérée** des états cachés de toutes les cellules RNN en utilisant les **poids weightsHO** et le biais **biasO**. Cette somme pondérée est ensuite passée à la fonction d'activation pour **obtenir la sortie** de la cellule.

   Pour étendre le nombre de **cellules RNN** dans le réseau, il faut ajouter des cellules RNN supplémentaires à la couche cachée. Cela implique d'augmenter la taille des tableaux **hidden**, **weightsHH**. Il faut également modifier la fonction **forwardPass** pour prendre en compte ces nouvelles cellules.

Exercice III - CNN sur Arduino Due

1. **A quoi servent les constantes `INPUT_SIZE, KERNEL_SIZE, PADDING_SIZE, STRIDE_SIZE et POOL_SIZE` ?**

   Les constantes **`INPUT_SIZE, KERNEL_SIZE, PADDING_SIZE, STRIDE_SIZE et POOL_SIZE`** sont utilisées pour définir les paramètres du réseau de neurones **CNN**. Plus particulièrement les constantes représentent :
   - **`INPUT_SIZE`** : représente la taille de la matrice d'entrée, ici 9×9 pixels.
   - **`KERNEL_SIZE`** : représente la taille du noyeau de convolution utilisé pour filtrer la matrice, ici 3×3 pixels.
   - **`PADDING_SIZE`** : représente la taille de remplissage ajoutée autour de la matrice, ici 1 pixel.
   - **`STRIDE_SIZE`** : représente la taille du pas de la fenêtre de convolution lors du glissement de l'image d'entrée, ici 1 pixel.
   - **`POOL_SIZE`** : représente la taille la fenêtre utilisée pour effectuer le max-pooling sur la sortie de la convolution, ici 2×2 pixels.

   Ces paramètres sont utilisées dans les différentes fonctions afin de traiter la matrice d'entrée et produire une sortie.

2. **Que fait la fonction convolution2D et quels sont les arguments qu'elle prend en entrée ?**

   La fonction convolution2D effectue une convolution 2D entre l'image d'entrée (définie par la matrice input), le noyau (défini par la matrice kernel) et un biais (défini par le scalaire bias), et stocke le résultat de la convolution dans la matrice de sortie output. Les arguments d'entrée de cette fonction sont :
   - **input** : une matrice carrée de dimensions **`INPUT_SIZE`**×**`INPUT_SIZE`**, représentant l'image d'entrée.
   - **kernel** : une matrice carrée de dimensions **`IKERNEL_SIZE`**×**`KERNEL_SIZE`**, représentant le noyau de convolution.
   - **output** : une matrice carrée de dimensions **`IUTPUT_SIZE`**×**`OUTPUT_SIZE`**, représentant la sortie de la convolution.
   - **bias** : un scalaire représentant le biais de la couche de convolution.

3. **Que fait la fonction maxPooling et quels sont les arguments qu'elle prend en entrée ? Préciser les dimensions des matrices d'entrée et de sortie.**

   La fonction **maxPooling** est une opération de sous-échantillonage, c'est à dire une opération qui consiste à réduire la taille spatiale de la matrice d'entrée. Cela permet notamment de réduire la complexité du modèle et de diminuer la quantité de donnée traitées tout en conservant les données importantes.

   La fonction prend en entrée deux paramètres qui sont :
   - **poolinput** : Une matrice d'entrée de taille 9×9
   - **pool** : Une matrice de sortie de taille 4×4

4. **Que fait la fonction flatten2vector et quels sont les arguments qu'elle prend en entrée ? Préciser les dimensions de la matrice d'entrée et la taille du vecteur de sortie.**

   La fonction flatten2vector prend en entrée une matrice multidimensionnelle et renvoie un vecteur à une dimension en "aplatissant" toutes les dimensions de la matrice d'entrée.
   Plus précisément, la fonction prend en entrée une matrice de dimension 4×4 et renvoie un vecteur de dimension 16×1.

5. **Ajouter une fonction Printflatten2vector en code Arduino pour afficher la taille du vecteur.**

   On crée la fonction **Printflatten2vector** suivante, afin d'afficher la taille du vecteur de sortie de la fonction **flatten2vector** :

```
/**
* @brief Prints the flattened vector
*/
void printflatten2vector()
{
  Serial.print("Flattened Vector Size: ");
  Serial.println(NumberOf(eflattened));
  Serial.println();
}
```

   Après exécution de ce code, on obtient la sortie suivante :

```
Flattened Vector Size: 16
```

   On obtient donc bien la sortie attendue de 16.

6. **Peut on définir le vecteur expectedOutput. Si oui comment vous pouvez le générer ?**

7. **Est-il toujours possible d'appliquer NN.BackProp ?**

8. **Executer le code arduino ci-dessous pour générer une sortie de CNN.**

9. **Ajoutez une deuxième couche à votre CNN (Convolution 2D et Max-pooling) et exécutez à nouveau le code Arduino. N'oubliez pas d'ajouter des matrices de taille appropriée pour la deuxième couche et assurez-vous que le nouveau CNN génère un vecteur "flatten" de taille plus petit.**

Annexes

A - Code d'un RNN

```
/**
* @brief This code illustrate a small RNN
* @author M. DERRAZ
*/
#include <Arduino.h>

// Constants
const int inputSize = 5;        // Number of input neurons
const int hiddenSize = 10;      // Number of hidden neurons
const int outputSize = 1;       // Number of output neurons
const int sequenceLength = 20; // Length of the sequence
    to be predicted
const int ledPin = 13;          // Pin for the LED display
// Variables
float inputs[inputSize];                    // Input sequence
float outputs[outputSize];                  // Output
    sequence
float hidden[hiddenSize];                   // Hidden state
float weightsIH[hiddenSize][inputSize];   // Input-hidden
    weights
float weightsHH[hiddenSize][hiddenSize]; // Hidden-hidden
    weights
float weightsHO[outputSize][hiddenSize]; // Hidden-output
    weights
float biasH[hiddenSize];                    // Hidden biases
float biasO[outputSize];                    // Output biases
float threshold = 0.5;                      // Prediction
    threshold
bool predict = false;                       // Flag for
    starting prediction sequence

/// Funtions declaration
void initializeWeights();
void forwardPass(float *input, float *output, float *
    hidden);
float activation(float x);
void readPoten();
void startPrediction();
```

Code A.1 – Constants for a simple RNN

```
void setup()
{
    // Setup pins
    Serial.begin(115200);
    pinMode(ledPin, OUTPUT);
    // Initialize weights and biases
    initializeWeights();
}

void loop()
{
    // Read potentiometer value and adjust threshold
    readPoten();
    // Check if start button is pressed

    startPrediction();

    if (predict)
    {
    forwardPass(inputs, outputs, hidden);

    // Output predicted value on LED display
    if (outputs[0] > threshold)
    {
        digitalWrite(ledPin, HIGH);
    }
    else
    {
        digitalWrite(ledPin, LOW);
    }

    Serial.print("Output: ");
    Serial.println(outputs[0]);

    // Shift input sequence
    for (int i = sequenceLength - 1; i > 0; i--)
    {
        inputs[i] = inputs[i - 1];
    }
    inputs[0] = outputs[0];
    }
}
```

Code A.2 – Loop and Setup functions for a simple RNN

```
/**
* @brief Initialize weights and biases
*/
void initializeWeights()
{
    // Initialize input-hidden weights
    for (int i = 0; i < hiddenSize; i++)
    {
    for (int j = 0; j < inputSize; j++)
    {
        weightsIH[i][j] = random(-100, 100) / 100.0;
    }
    }
    // Initialize hidden-hidden weights
    for (int i = 0; i < hiddenSize; i++)
    {
    for (int j = 0; j < hiddenSize; j++)
    {
        weightsHH[i][j] = random(-100, 100) / 100.0;
    }
    }
    // Initialize hidden-output weights
    for (int i = 0; i < outputSize; i++)
    {
    for (int j = 0; j < hiddenSize; j++)
    {
        weightsHO[i][j] = random(-100, 100) / 100.0;
    }
    }
    // Initialize biases
    for (int i = 0; i < hiddenSize; i++)
    {
    biasH[i] = random(-100, 100) / 100.0;
    }
    for (int i = 0; i < outputSize; i++)
    {
    biasO[i] = random(-100, 100) / 100.0;
    }
}

/**
* @brief Forward pass of the RNN to compute the output
    sequence
*/
void forwardPass(float *input, float *output, float *
    hidden)
{
```

```
46      // Compute hidden state
47      for (int i = 0; i < hiddenSize; i++)
48      {
49      hidden[i] = 0;
50      for (int j = 0; j < inputSize; j++)
51      {
52          hidden[i] += weightsIH[i][j] * input[j];
53      }
54      for (int j = 0; j < hiddenSize; j++)
55      {
56          hidden[i] += weightsHH[i][j] * hidden[j];
57      }
58      hidden[i] += biasH[i];
59      hidden[i] = activation(hidden[i]);
60      }
61      // Compute output
62      for (int i = 0; i < outputSize; i++)
63      {
64      output[i] = 0;
65      for (int j = 0; j < hiddenSize; j++)
66      {
67          output[i] += weightsHO[i][j] * hidden[j];
68      }
69      output[i] += biasO[i];
70      output[i] = activation(output[i]);
71      }
72  }
73
74  /**
75  * @brief Sigmoid activation function
76  */
77  float activation(float x)
78  {
79      // Sigmoid activation function
80      return 1.0 / (1.0 + exp(-x));
81  }
82
83  /**
84  * @brief Read potentiometer value and adjust prediction
        threshold
85  */
86  void readPoten()
87  {
88      // Read poten value and map to prediction threshold
89      int potValue = 0.5; // exemple
90  }
91
92
```

```
93 /**
94 * @brief Start prediction sequence by initializing input
       sequence with random values
95 * and by setting the predict flag to true
96 */
97 void startPrediction()
98 {
99     // Initialize input sequence with random values
100     for (int i = 0; i < sequenceLength; i++)
101     {
102     inputs[i] = random(0, 10) / 10.0;
103     }
104     // Set flag to start prediction sequence
105     predict = true;
106 }
```

Code A.3 – RNN Functions to activate, predict and train the NN