المركز الجهوي لمهن التربية و التكوين سوس ماسة
ⴰⵎⵔⴰⵙ ⴰⵏⴰⵎⵓⵔ ⵏ ⵜⵓⵙⵙⵉⵔⵜ ⵏ ⵙⵓⵙ ⵎⴰⵙⵙⴰ - ⵓⵙⵓ ⵎⴰⵙⵙⴰ
Centre Régional des Métiers d'Education et de Formation - Souss Massa

REALISATION D'UN CHATBOT EDUCATIF POUR L'ENSEIGNEMENT D'ALGORITHMIMIQUE

Projet de fin de module

Réalisé par:

- HOUSSEY Abderrahim
- LAHDAR Taib
- HARBECH Maryam

Encadré par:

- EL HAJJI Mohamed
- Pr. Tarek AIT BAHA

Table des matières

Liste des figures

Introduction

Dans le cadre de la formation des enseignants-cadres des Académies régionales de l'éducation et de la formation -Filière informatique- au sein du centre régional des métiers d'éducation et de formation sous massa Agadir, les professeurs profitent de l'opportunité d'explorer le module 'Production didactique et TIC'. Comme projet de fin de ce dernier nous avons à réaliser un Chatbot éducatif.

Au niveau le plus fondamental, un chatbot est un programme informatique qui simule et traite une conversation humaine (écrite ou parlée), permettant aux humains d'interagir avec des terminaux digitaux comme s'ils communiquaient avec une personne réelle. Les chatbots peuvent être aussi simples que des programmes rudimentaires répondant à une requête simple avec une réponse sur une seule ligne, ou aussi sophistiqués que des assistants digitaux qui apprennent et évoluent pour fournir des niveaux de personnalisation croissants à mesure qu'ils collectent et traitent des informations.

Le présent rapport explique les différentes étapes suivies pour réaliser notre chatbot. Nous commençons notre rapport par une définition du projet dont nous allons présenter le cahier de charge et le mind map du projet.

Ainsi, nous proposons dans ce projet une approche basée sur l'utilisation de l'apprentissage profond (Deep Learning) en utilisant la bibliothèque Keras pour ensuite l'appliquer sur notre base de données afin de réaliser un chatbot éducatif qui peut interagir avec les utilisateurs.

1. Définition de projet

1.1. Cahier de charge du projet

Dans une salle de classe, chaque élève a des besoins et des intérêts d'apprentissage différents. Par conséquent, chacun a besoin de l'aide d'un tuteur spécialisé. Malheureusement, ce type de service n'est pas disponible même dans les écoles les plus chères du monde.

La question qui se pose est : Quelle est l'alternative la plus pratique et la plus économique pour résoudre ce problème ?

Ce projet vise l'acquisition des compétences clef́ nécessaires en Algorithmique et en Programmation. Il a pour objectif de créer un chatbot pour les formateurs et les étudiants à utiliser dans le secteur de l'enseignement. Le projet sera divisé en quatre niveaux :

- **Niveau 1 :** Création d'un chatbot basique qui répond aux questions.
- **Niveau 2 :** Création d'un chatbot qui propose des exercices et corrige les TPs.
- **Niveau 3 :** Création d'un chatbot avec un avatar personnalisable aux enseignants.
- **Niveau 4 :** Création d'un chatbot hautement personnalisable.

1.2. Mind Map de projet

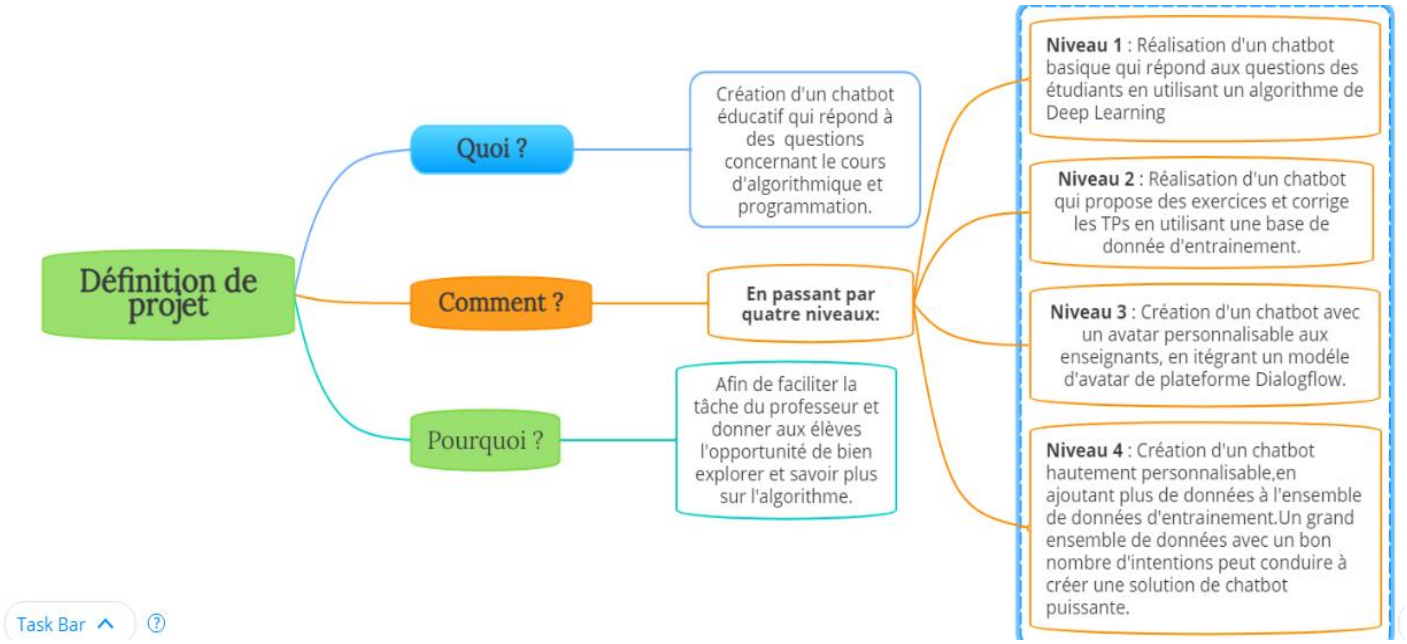
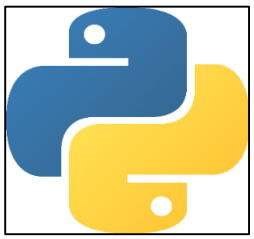


Figure 1 : Schéma descriptive du projet sous forme de mind map

2. Réalisation et Implémentation

2.1. Environnement de développement

2.1.1. Langage de programmation Python



Python est un langage de programmation interprété, portable, dynamique, extensible et multi-plateformes, il favorise la programmation impérative

structurée fonctionnelle et orientée objet, c'est un langage open source **créé par le programmeur Guido van Rossum en 1991**. Il tire son nom de l'émission Monty Python's Flying Circus [31]. Un programme interpréteur permet d'exécuter le code Python sur n'importe quelle machine. Ceci permet de voir rapidement les résultats d'un changement dans le code, ce qui rend ce langage plus rapide qu'un langage compilé comme le C. Ainsi, écrire des programmes prend moins de temps que dans un autre langage.

2.1.2. Anaconda

Anaconda est une distribution libre et open source des langages de programmation Python et appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique). Les versions de paquetages sont gérées par le system de gestion de paquets *conda*. La Distribution Anaconda est utilisée par plus de 6 millions d'utilisateurs, et il comprend plus de 250 paquets populaires en science des données adaptés pour Windows, Linux et MacOS.

2.1.3. PyCharm

PyCharm est un environnement de développement intégré utilisé pour programmer en Python. Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django et flask.

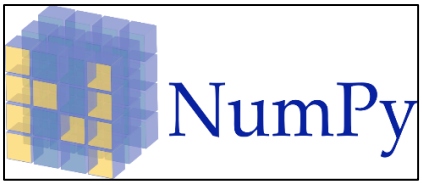
2.2. Bibliothèques et packages :

2.2.1. TensorFlow

TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache. Initialement, le but de TensorFlow était d'optimiser les calculs numériques complexes, mais

aujourd'hui il est très connu pour résoudre des problèmes de Deep Learning. Cette bibliothèque permet notamment d'entraîner et d'exécuter des réseaux de neurones pour la classification de chiffres écrits à la main, la reconnaissance d'image, les réseaux de neurones récurrents, les modèles sequence-to-sequence pour la traduction automatique, ou encore le traitement naturel du langage.

2.2.2. NumPy



NumPy est une extension du langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux. Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes.

2.2.3. Keras

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et capable de fonctionner sur TensorFlow ou Theano. Il a été développé en mettant l'accent sur l'expérimentation rapide. Être capable d'aller de l'idée à un résultat avec le moins de délai possible est la clé pour faire de bonnes recherches. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet, un ingénieur Google.

2.2.4. FlasK

Flask est un micro framework open-source de développement web en Python. Il est classé comme microframework car il est très léger. Flask a pour objectif de garder un noyau simple mais extensible. Il n'intègre pas de système d'authentification, pas de couche d'abstraction de base de données, ni d'outil de validation de formulaires. Cependant, de nombreuses extensions permettent d'ajouter facilement des fonctionnalités.

2.3. Implémentation

2.3.1. Les étapes à suivre :

a. Définition des intentes :

Une « intention » est l'intention de l'utilisateur d'interagir avec un chatbot ou l'intention derrière chaque message que le chatbot reçoit d'un utilisateur particulier. Afin de répondre aux questions, de rechercher dans la base de connaissances du domaine et d'effectuer diverses autres tâches pour poursuivre les conversations avec l'utilisateur, notre chatbot a besoin de comprendre ce que les utilisateurs disent ou ce qu'ils ont l'intention de faire.

b. Préparation de données :

Création d'un fichier nommé « intents. JSON » contient les intents de notre modèle :

```
{"intents": [
   {"tag": "Chatbot",
    "patterns": ["Comment tu t'appelle?", "nom", "Dis-moi comment tu t'appelles ?", "Dites-moi comment vous
    "responses": [" Bonjour ,Je m'appelle Algorithme Chatbot "]
   },
   {"tag": "algorithme",
    "patterns": ["Algorithme", "C'est quoi un algorithme", "algorithmique","ce quel algorithme ?","Qu'est-c
    "responses": ["Un algorithme est une suite d'actions ou d'instructions qui doivent être exécutées dans
   },
   {"tag": "Donnée",
    "patterns": ["Notion de donnée ", "ce quoi une Notion de donnée ?", "Les données d'un Algorithmes ?", "
    "responses": ["Les données sont des informations nécessaires au déroulement d'un algorithme.", "Les don
   },
   {"tag": "Constante",
    "patterns": ["constante", "ce quoi une constant","Constante ?","Les constantes "],
    "responses": ["Une constante est une donnée fixe qui ne varie pas durant l'exécution d'un algorithme. U
   },
   {"tag": "variable",
   "patterns": ["ce quoi une variable ?", "Variable", "définition d'une variable"],
   "responses": ["Une variable est un objet dont le contenu peut être modifié par une action", "Une variabl
   },
   {"tag": "representation",
   "patterns": ["représenter un algorithme", "Comment représenter un algorithme ?","la structure d'un algor
   "responses": ["On peut représenter un algorithme à l'aide d'un pseudo-code ou d'un organigramme. Un algo
   },
```

Figure 2 : Le fichier « intents. JSON » qui contient les patterns et les réponses

Par la suite on va extraire les données de fichier « intents. JSON »

```
with open('intents.json') as file:
    data = json.load(file)

training_sentences = []
training_labels = []
labels = []
responses = []


for intent in data['intents']:
    for pattern in intent['patterns']:
        training_sentences.append(pattern)
        training_labels.append(intent['tag'])
    responses.append(intent['responses'])

    if intent['tag'] not in labels:
        labels.append(intent['tag'])

num_classes = len(labels)
```

Figure 3 : L'instruction d'extraction des données requises.

La variable "*Training_Sencences*" détient toutes les données d'entrainement (qui sont les échantillons de messages dans chaque catégorie d'intention) et la variable "*Training_Labels*" contient toutes les étiquettes cible correspond à chaque donnée d'entrainement. Ensuite, nous utilisons "*Labelencoder ()"* Fonction fournie par Scikit-Apprendre à convertir les étiquettes cible en un modèle compréhensible.

```
lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)
```

Figure 4 : La fonction LabelEncoder ().

Ensuite, nous vectorisons notre corpus de données texte en utilisant la classe "*Tokenizer*" et nous permet de limiter notre taille de mots jusqu'à un nombre défini.

```
vocab_size = 1000
embedding_dim = 16
max_len = 20
oov_token = "<OOV>"

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_len)
```

Figure 5 : La classe « Tokenizer » de vectorisation.

La méthode "***padded_sequences***" est utilisée pour effectuer toutes les séquences de texte d'entrainement dans la même taille.

c. Prédiction et évaluation :

Définissons tout d'abord notre architecture de réseau de neurone pour le modèle transposé, et pour ça nous utilisons la classe « sequential » de Keras.

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

model.summary()
```

Figure 6 : La classe « sequentiel » d'entrainement.

Notre architecture modèle ressemble à la suivante :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 20, 16)            16000
_________________________________________________________________
global_average_pooling1d (Gl (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 16)                272
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 136
=================================================================
Total params: 16,680
Trainable params: 16,680
Non-trainable params: 0
_________________________________________________________________
```

Figure 7 : L'architecture modèle d'entrainement.

Maintenant, nous sommes prêts à entrainer notre modèle. Nous pouvons simplement appeler la méthode "*fit*" avec des données d'entrainement et des étiquettes.

```
1   epochs = 500
2   history = model.fit(padded_sequences, np.array(training_labels), epochs=epochs)
```

Figure 8 : La fonction fit ().

d. Test du modèle

Nous allons mettre en œuvre une fonction de discussion pour dialoguer avec un véritable utilisateur. Lorsqu'un nouveau message utilisateur est reçu, le chatbot calculera la similitude entre la nouvelle séquence de texte et les données d'entrainement.

```
def chat():
    # load trained model
    model = keras.models.load_model('chat_model')

    # load tokenizer object
    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

    # load label encoder object
    with open('label_encoder.pickle', 'rb') as enc:
        lbl_encoder = pickle.load(enc)

    # parameters
    max_len = 20

    while True:
        print(Fore.LIGHTBLUE_EX + "User: " + Style.RESET_ALL, end="")
        inp = input()
        if inp.lower() == "quit":
            break

    while True:
        print(Fore.LIGHTBLUE_EX + "User: " + Style.RESET_ALL, end="")
        inp = input()
        if inp.lower() == "quit":
            break

        result = model.predict(keras.preprocessing.sequence.pad_sequences(tokenizer.texts_to_sec
                                            truncating='post', maxlen=max_len))
        tag = lbl_encoder.inverse_transform([np.argmax(result)])

        for i in data['intents']:
            if i['tag'] == tag:
                print(Fore.GREEN + "ChatBot:" + Style.RESET_ALL , np.random.choice(i['responses'

        # print(Fore.GREEN + "ChatBot:" + Style.RESET_ALL,random.choice(responses))

print(Fore.YELLOW + "Start messaging with the bot (type quit to stop)!" + Style.RESET_ALL)
chat()
```

Figure 9 : La fonction chat ().

On a défini la fonction ***chat ()*** (figure 9), pour dialoguer avec un vrai utilisateur. Lorsqu'un nouveau message d'utilisateur est reçu, le chatbot calcule la similitude entre la nouvelle séquence de texte et les données d'entraînement. Compte tenu des probabilités obtenus pour chaque catégorie, il classe le message de l'utilisateur dans une intention avec la probabilité d'appartenant à l'intention le plus élevé.

Le test sur un console donne le résultat suivant :

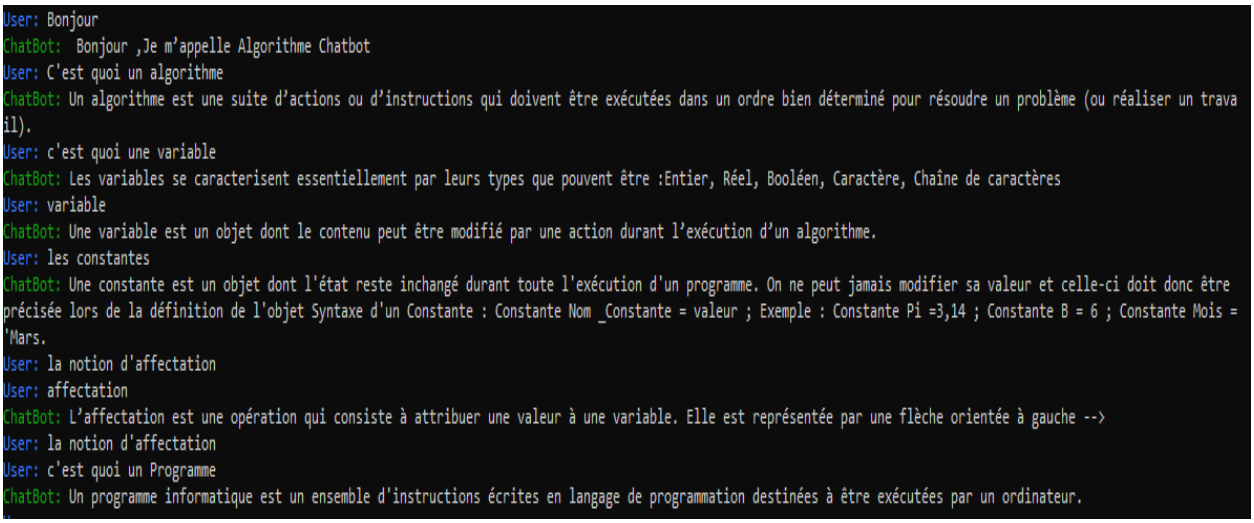


Figure 10 : Le résultat de la fonction chat ().

3. Appelez le modèle avec Flask

Flask, est un framework Open-Source d'une simplicité et d'une efficacité. Flask permet l'ajout de composants additionnels mais aussi intègre un véritable moteur de template pour y associer des couches HTML/CSS.

Dans le bloc de code suivant, nous importons d'abord l'objet Flask du paquet flask. Nous l'utilisons ensuite pour créer notre instance d'application Flask avec le nom app. La variable spéciale « __name__ » qui contient le nom du module Python actuel. Il est utilisé pour indiquer à l'instance où elle se trouve.

```python
from flask import Flask, render_template, request
from chat import chat

app = Flask(__name__)


@app.route("/")
def home():
    return render_template("index.html")

@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    answear = chat(userText)

    return answear #str(english_bot.get_response(userText))


if __name__ == "__main__":
    app.run()
```

Figure 11 : l'instanciation d'application Flask

Après l'utilisation des modèles HTML et CSS et JavaScript notre interface est donné le résultat suivant :

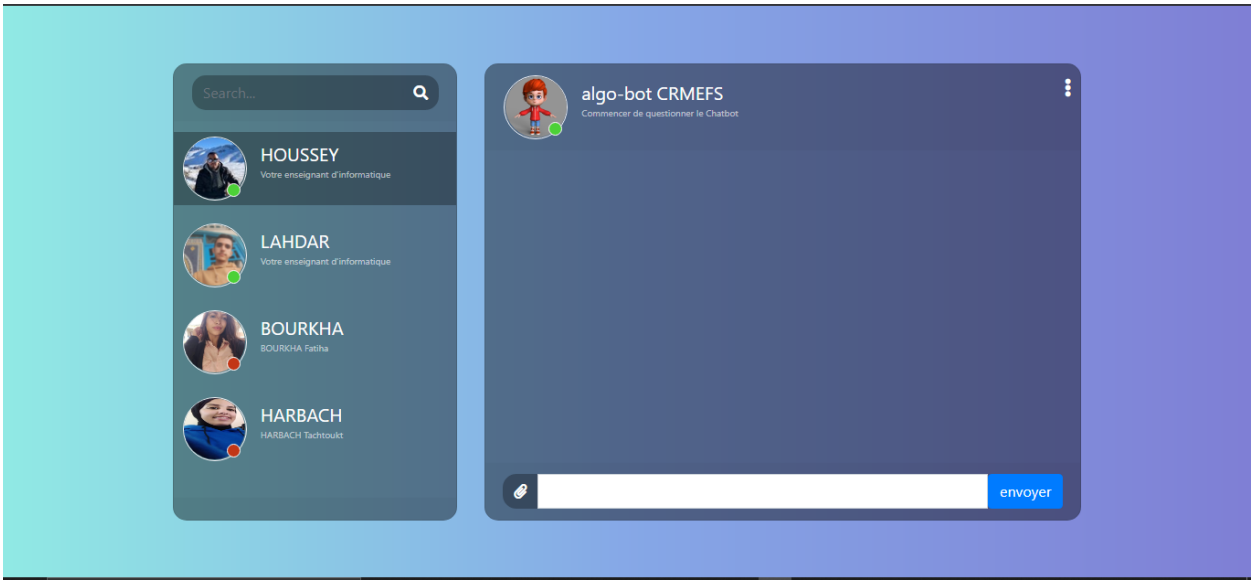


Figure 12 : l'interface final de notre modèle dans un serveur web Flask

4. Création d'avatar avec la plateforme « botlibre »

4.1. La plateforme botlibre

Bot Libre est une plate-forme de chatbot open source qui vous permet de télécharger et d'installer votre propre plate-forme de chatbot sur site, sur votre propre serveur ou service cloud.

Il vous permet de créer des bots de chat avec plusieurs options comme des avatars, des bots…Il vous permet aussi de créer des travails espaces.

Ce qui nous intéresse dans la plateforme est la génération des bots avec avatar 3D pour l'ajouter à notre serveur flask.

4.2. Avatar 3D

La plateforme botlibre donne l'option de créer votre propre chatbot avec l'avatar 3D.

Figure 13 : Création d'un nouveau bot dans plateforme « botlibre »

Après donner le bot des informations nécessaires, la plateforme donne l'option de l'intégration dans votre site web ou bien votre local serveur.

Figure 14 : Option d'intégration de bot dans plateforme « botlibre »

Par la suite, l'avatar sera prêt de fonctionner sur notre serveur web.

Figure 15 : Le bot avec l'avatar 3D dans le serveur web.

5. Conclusion

Malgré la contrainte de temps, on a pu atteindre le niveau 3 de ce challenge et on a appris énormément de chose en élaborant ce travaille parmi elles :

L’esprit de travail en groupe, le rôle de cet outil dans notre profession, l’importance des chatbots dans l’enseignement.