```
import string, random
# Importation de la librairie Natural Language Tool Kit pour les traitements NLP
sur le text
import nltk
import numpy as np
from nltk.stem import WordNetLemmatizer
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
# importation de la fonction load_model pour le chargement du model
from tensorflow.python.keras.saving.save import load_model

# chargement des différents dictionnaires de mots
nltk.download('omw-1.4')
nltk.download("punkt")
nltk.download("wordnet")

# initialisation de lemmatizer pour obtenir la racine des mots
lemmatizer = WordNetLemmatizer()

# Listes des features généré
words = []

# Liste des intentions
classes = []

doc_X = []
doc_y = []

# Utilisation d'un dictionnaire pour représenter un fichier JSON d'intentions
data = {
    "intents": [
        {
            "tag": "grettings",
            "patterns": ["salut à toi!", "hello", "comment vas tu?",
"salutations!", "enchanté", "hey"
                    "hey hey", "he", "heyyy"
                    "bonjour!",
                    "salut, comment ca va",
                    "bonjour, comment ca va",
                    "salut, comment vas-tu",
                    "comment vas-tu",
                    "enchantée.",
                    "salut, content de te connaitre.",
                    "un plaisir de te connaitre.",
                    "passe une bonne journée",
                    "quoi de neuf"],
            "responses" : ["Salut", "Bonjour", "Hello !", "Hi"]
        },
        {
            "tag": "services_status",
            "patterns": ["afficher l'état des services", "statut des services",
"Je veux connaitre l'état des services",
                    "comment fonctionne le serveur", "fonctionnement du
serveur",
```

```
                    "Etat de marche du serveur", "regime de fonctionnement",
"fonctionnement des services"],
            "responses": ["Voici le statut des services", "voici le rapport de
fonctionnement des services", "fonctionnement des services"]
        },
        {
            "tag": "signature_database",
            "patterns": ["afficher la base de signatures", "montrer la base
virale", "afficher la liste des règles",
                "règle de l'IDS", "signatures des attaques", "liste des
attaques", "liste des règles", "Montre moi les règles"],
            "responses": ["Voici la base de signature", "Voici la base de
signature la plus à jour"]
        },
        {
            "tag": "simba_rules",
            "patterns": ["quelles sont le fichier des règles d'alerte", "quelles
sont tes règles", "montre moi le fichier règles",
            "fichier de règle de simba", "affiche le fichier des règles",
"fichier de règle"],
            "responses": ["voici le contenu du fichier de règle personnalisé"]
        },
        {
            "tag" : "intrusion_report",
            "patterns" : ["je veux le rapport d'intrusion dans le réseau",
"rapport d'intrusion dans le réseau",
                    "liste des intrusions dans le réseau", "rappoort d'alertes",
"liste alertes", "log des alertes",
                    "afficher les attaques", "affiches les alertes", "montre moi
les alertes", "liste des attaques "],
            "responses" : ["Voici la liste des alertes de ce jours"]
        },
        {
            "tag" : "send_intrusion_report",
            "patterns" : ["envoi moi le rapport d'intrusion", "envoi du rapport
d'alert", "envoyer le rapport par mail",
                    "envoyer les alertes dans le réseau par mail"],
            "responses" : ["Envoi du rapport d'intrusion"]
        },
        {
            "tag" : "block_user_rule",
            "patterns" : ["bloque un utilisateur", "bloque une machine", "stop
une machine",
                        "stop une adresse machine", "arrêter un utilisateur"],
            "responses" : ["blocage d'une utilisateur"]
        },
        {
            "tag" : "add_rule",
            "patterns" : ["ajouter une règle", "définir une règle","ajout d'une
règle",
                        "ajoute une règle", "je veux ajouter une règle", "je
veux modifier les règles"],
            "responses" : ["Ajout d'une règle"]
        },
        {
            "tag" : "firewall",
            "patterns" : ["pare-feu", "pare feu", "parefeu" "afficher la
configuration du pare-feu",
```

```
                        "afficher les règles du pare-feu", "afficher le pare-
feu", "afficher la table ACL"
                        "configuration pare-feu"],
            "responses" : ["Voici la configuration actuel du pare-feu"]
        },
        {
            "tag" : "red_code",
            "patterns" : ["code rouge", "code code rouge", "arrêter tout les
services", "éteindre le réseau",
                        "arrêter les serveurs", "stoper les serveur", "éteindre
les serveurs",
                        "éteindre les services"],
            "responses" : ["Code rouge activé"]
        },
        {
            "tag": "ssh_connections",
            "patterns": ["afficher la liste des connexions SSH", "Afficher les
dernière connexions SSH",
                    "connexion SSH"],
            "responses": ["Voici la liste des dernière connexions SSH"]
        },
        {
            "tag": "stop_simba_client",
            "patterns": ["Au revoir", "A plus", "Bye", "Stop", "cya", "Au
revoir"],
            "responses": ["C'était sympa de vous parler", "à plus tard", "A
plus!"]
        }
]}

# Nous iterrons sur tous les intentions et nous tokenisons chaque patterns que
nous ajoutons à la liste words
for intent in data["intents"]:
    for pattern in intent["patterns"]:
        tokens = nltk.word_tokenize(pattern)
        words.extend(tokens)
        doc_X.append(pattern)
        doc_y.append(intent["tag"])

    # Ajouter le tag aux classes
    if intent["tag"] not in classes:
        classes.append(intent["tag"])
# Conversion en minuscule de tous les mots du vocalbulaire
# et lemmatisation
# On evite les caractères de pontuation
words = [lemmatizer.lemmatize(word.lower())
         for word in words if word not in string.punctuation]

# trie par ordre alphabétique et supression des doubles en convertissant les
listes en set
words = sorted(set(words))
classes = sorted(set(classes))

"""
Une fois lancée, cette fonction permet de construire et d'entrainer le modèle.
Le modèle issue est sauvegardé dans une fichier simba_model au format HDF5
"""
def train_model() -> None:
```

```
    global model
    # liste pour les données d'entraînement
    training = []
    out_empty = [0] * len(classes)
    # création du modèle d'ensemble de mots
    for idx, doc in enumerate(doc_X):
        bow = []
        text = lemmatizer.lemmatize(doc.lower())
        for word in words:
            bow.append(1) if word in text else bow.append(0)
        # marque l'index de la classe à laquelle le pattern atguel est associé à
        output_row = list(out_empty)
        output_row[classes.index(doc_y[idx])] = 1
        # ajoute le one hot encoded BoW et les classes associées à la liste
training
        training.append([bow, output_row])
    # mélanger les données et les convertir en liste
    random.shuffle(training)
    training = np.array(training, dtype=object)
    # séparer les features et les labels(différentes classes)
    train_X = np.array(list(training[:, 0]))
    train_y = np.array(list(training[:, 1]))

    # définition des paramètres pour la création du modèle
    input_shape = (len(train_X[0]),)
    output_shape = len(train_y[0])
    epochs = 200

    # Modèle Deep Learning de Simba
    model = Sequential()
    # Couche d'entrée du réseau de neurones
    model.add(Dense(128, input_shape=input_shape, activation="relu"))
    model.add(Dropout(0.5))
    # Couche cachée L=1
    model.add(Dense(64, activation="relu"))
    model.add(Dropout(0.3))
    # Couche de sortie
    model.add(Dense(output_shape, activation="softmax"))
    # Ajout de la fonction d'optimisation Adam
    adam = tf.keras.optimizers.Adam(learning_rate=0.01, decay=1e-6)

    # Définition des paramètres pour la retropropagation
    model.compile(loss='categorical_crossentropy',
                optimizer=adam, metrics=["accuracy"])

    # Entrainement du modèle sur 200 itérations
    model.fit(x=train_X, y=train_y, epochs=200, verbose=1)

    # sauvegarde du modèle
    model.save('simba_model.hdf5')

    # Affichage du bilan de l'entrainement
    print("****************************************")
    print("FIN DE L'ENTRAINEMENT DU MODELE")
    print(f"Nombre de classes : {len(classes)}")
    print(f"Nombre de features : {len(words)}")
    print("****************************************")
```

```
# Chargement du modèle le plus à jour
model = load_model('simba_model.hdf5')

# fonction utilisée pour reformater l'entrée de l'utilisateur
# en utilisant un tokeniseur et le lemmatiseur
def clean_text(text):
    tokens = nltk.word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return tokens

# Fonction de transformation des tokens en base 2 (0,1) pour l'envoi des listes
de données dans le
# réseau de neurones et l'application des fonctions mathématiques
def bag_of_words(text, vocab):
    tokens = clean_text(text)
    bow = [0] * len(vocab)
    for w in tokens:
        for idx, word in enumerate(vocab):
            if word == w:
                bow[idx] = 1
    return np.array(bow)


def class_prediction(text, vocab, labels):
    bow = bag_of_words(text, vocab)
    result = model.predict(np.array([bow]))[0]
    thresh = 0.2
    y_pred = [[idx, res] for idx, res in enumerate(result) if res > thresh]
    y_pred.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in y_pred:
        return_list.append(labels[r[0]])
    return return_list


def get_intent(intents_list, json_intents):
    tag = intents_list[0]
    list_of_intents = json_intents["intents"]
    for intent in list_of_intents:
        if intent["tag"] == tag:
            break
    return intent

# lancement de l'agent conversationnel pour le test
if __name__ == '__main__' :
    while True:
        message = input("")
        intents = class_prediction(message.lower(), words, classes)
        result = random.choice(get_intent(intents, data)["responses"])
        print(result)
```