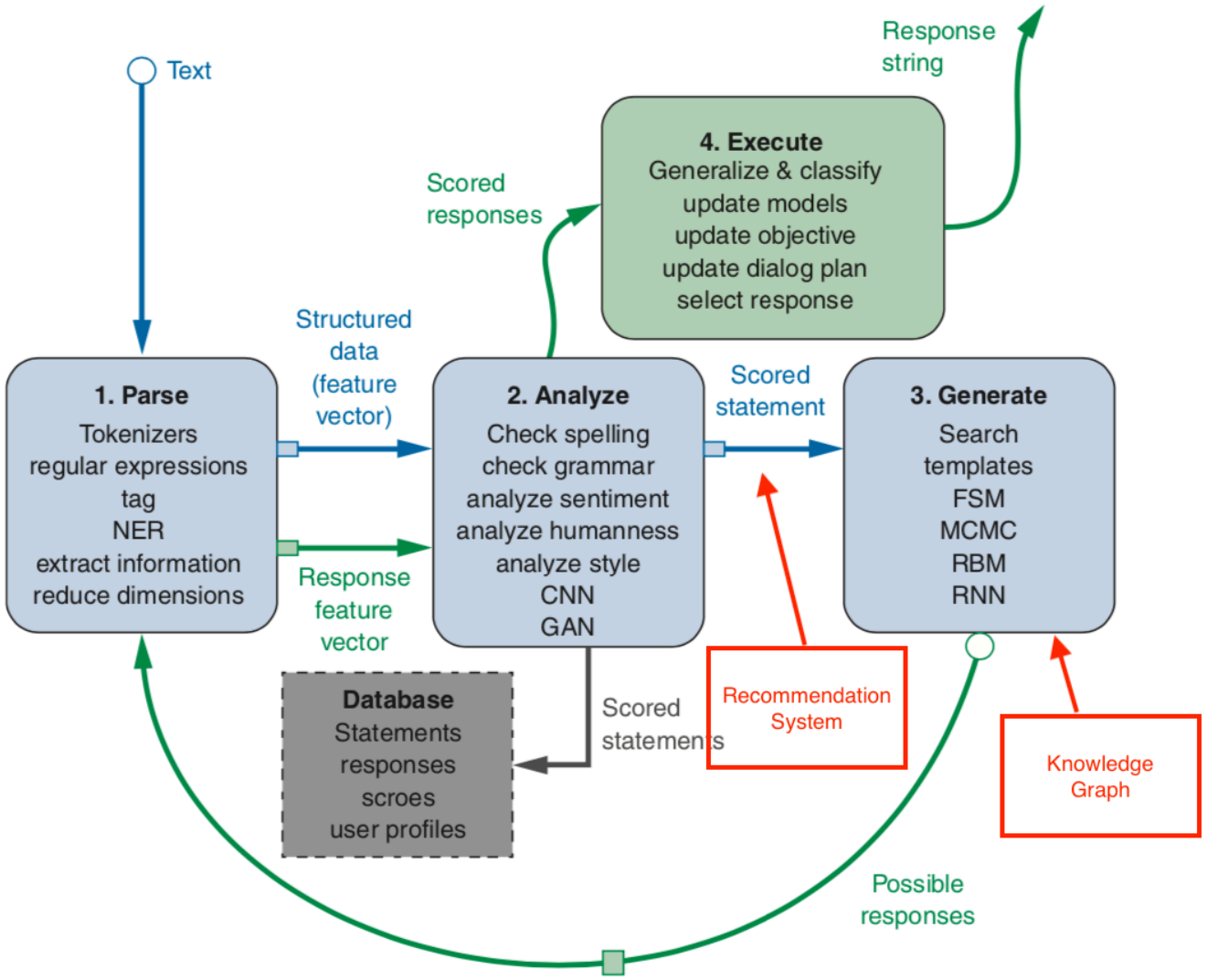


Processus de fonctionnement du chatbot dans le système

NLP – NLU - Deep Learning

1. Analyse des besoins (cont.)

- Challenges:
  - Small corpus of documents
  - Highly specialized language

* ™ ® ©

3. Text pre-processing

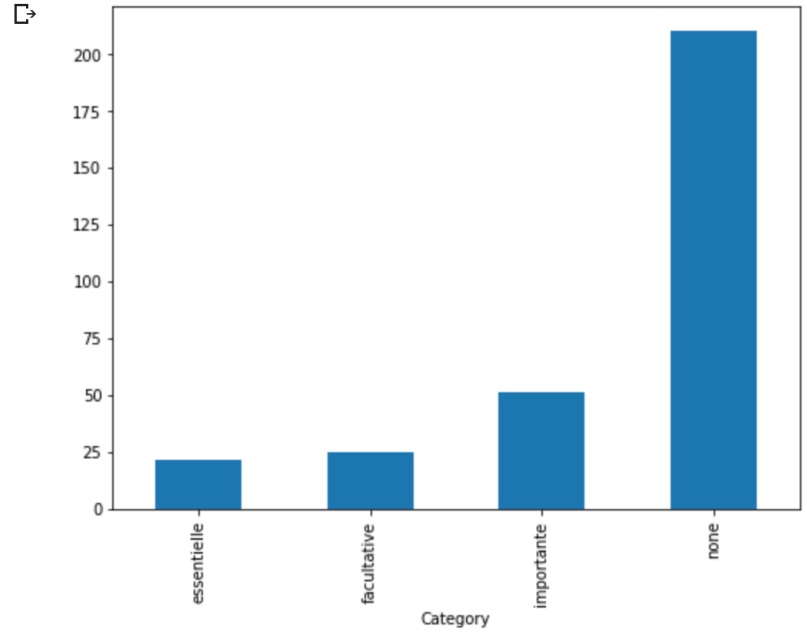
	SPACY	NLTK	CORENLP
Programming language	Python	Python	Java / Python
Neural network models	✓	✗	✓
Integrated word vectors	✓	✗	✗
Multi-language support	✓	✓	✓
Tokenization	✓	✓	✓
Part-of-speech tagging	✓	✓	✓
Sentence segmentation	✓	✓	✓
Dependency parsing	✓	✗	✓
Entity recognition	✓	✓	✓
Entity linking	✓	✗	✗
Coreference resolution	✗	✗	✓

	SPACY	NLTK	ALLEN-NLP	STANFORD-NLP	TENSOR-FLOW
I'm a beginner and just getting started with NLP.	✓	✓	✗	✓	✗
I want to build an end-to-end production application.	✓	✗	✗	✗	✓
I want to try out different neural network architectures for NLP.	✗	✗	✓	✗	✓
I want to try the latest models with state-of-the-art accuracy.	✗	✗	✓	✓	✓
I want to train models from my own data.	✓	✓	✓	✓	✓
I want my application to be efficient on CPU.	✓	✓	✗	✗	✗

3. Text pre-processing (cont.)

- The usual pre-treatment and exploration:
  - Cleaning, lemmatization, stopwords, tokenization, text tiling
  - Table of contents extraction and comparison
  - Text vectorization
  - Multi-class classification
  - Topic extraction

```
[ ]  # Distribution of clauses by importance

     import matplotlib.pyplot as plt
     fig = plt.figure(figsize=(8,6))
     df.groupby('Category').Text.count().plot.bar(ylim=0)
     plt.show()
```



4. Building the Chatbot (in Python)

- Version 1 : Hard-coded Q&A (with tflearn)
- Version 2 : TF-IDF + cosine similarity
- Version 3 : Text generation with LSTM (Long-Short Term Memory)
- Version 4 : RNN with external memory (unfinished)

Chatbot v.1

```
# ==================
# TRAINING WITH TF
# ==================

tf.reset_default_graph()

# Creating a connected neural network
# Input is the length of any training matrix row, so could be [0]

# To do: split into training/testing.

net = tflearn.input_data(shape=[None, len(training[0])])
net = tflearn.fully_connected(net, 8) # 8 neurons for the first hidden layer
net = tflearn.fully_connected(net, 8) # 8 neurons for the second hidden layer
net = tflearn.fully_connected(net, len(output[0]), activation = "softmax")
net = tflearn.regression(net, optimizer='adam', loss='categorical_crossentropy')

model = tflearn.DNN(net)

try:
    model.load("model.tflearn")
except:
    model.fit(training, output, n_epoch = 1000, batch_size=8, show_metric=True) #
    model.save("model.tflearn")

# =========
# CHATTING
# =========

# Transform user's phrase into a bag of words

def bag_of_words(phrase, words):
    bag = [0 for _ in range(len(words))]  # just one document

    tokens = nltk.word_tokenize(phrase)
    tokens = [stemmer.stem(w.lower()) for w in tokens if w not in "?"]

    # Choose only familiar words that we already know.
    # Create a bag of words.

    for token in tokens:
        for i, w in enumerate(words):
```

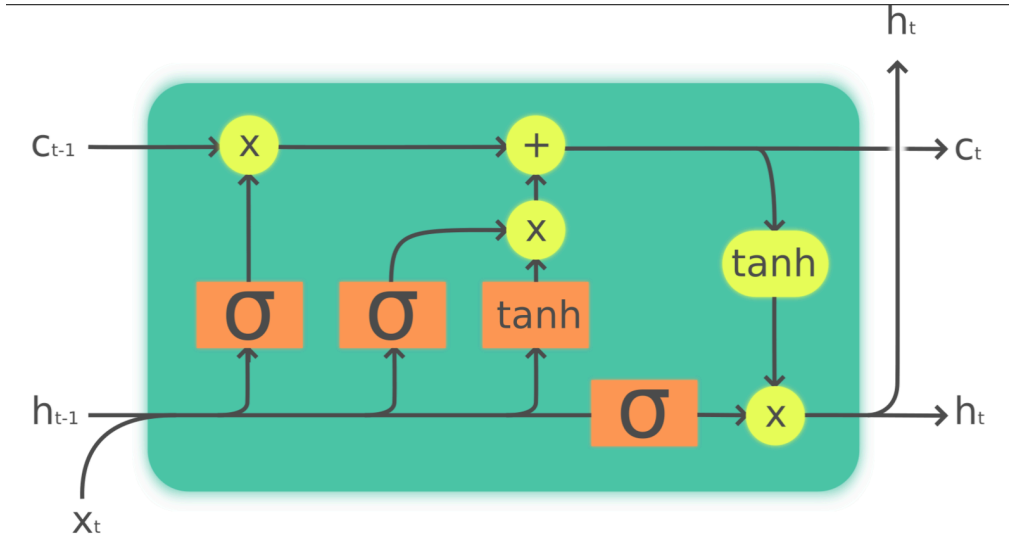
```
{"intents": [
        {"tag": "greeting",
         "patterns": ["Hi", "How are you", "Is anyone there?", "Hello", "Good day", "Whats up"],
         "responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"],
         "context_set": ""
        },
        {"tag": "goodbye",
         "patterns": ["cya", "See you later", "Goodbye", "I am Leaving", "Have a Good day"],
         "responses": ["Sad to see you go :(", "Talk to you later", "Goodbye!"],
         "context_set": ""
        },
        {"tag": "age",
         "patterns": ["how old", "how old is tim", "what is your age", "how old are you", "age?"],
         "responses": ["I am 18 years old!", "18 years young!"],
         "context_set": ""
        },
        {"tag": "name",
         "patterns": ["what is your name", "what should I call you", "whats your name?"],
         "responses": ["You can call me Tim.", "I'm Tim!", "I'm Tim aka Tech With Tim."],
         "context_set": ""
        },
        {"tag": "shop",
         "patterns": ["Id like to buy something", "whats on the menu", "what do you reccommend?", "coulc
         "responses": ["We sell chocolate chip cookies for $2!", "Cookies are on the menu!"],
         "context_set": ""
        },
        {"tag": "hours",
         "patterns": ["when are you guys open", "what are your hours", "hours of operation"],
         "responses": ["We are open 7am-4pm Monday-Friday!"],
         "context_set": ""
        }
    ]
}
```

```
Start talking with the bot (type quit to stop)!
You: hello
Good to see you again!
You: what hours are you open
We are open 7am-4pm Monday-Friday!
You: want to buy something
Cookies are on the menu!
You: good bye
Hi there, how can I help?
You: bye
I didn't get that, try again.
You: goodbye
Talk to you later
You:
```

Chatbot v.2

Retrieve correct answers based on cosine similarity of sentences

```
#Set the chatbot response to an empty string
robo_response = ''

#Append the users response to the sentence list
sent_tokens.append(user_response)

#Create a TfidfVectorizer Object
TfidfVec = TfidfVectorizer(tokenizer = LemNormalize, stop_words='fr')

#Convert the text to a matrix of TF-IDF features
tfidf = TfidfVec.fit_transform(sent_tokens)

#Get the measure of similarity (similarity scores)
vals = cosine_similarity(tfidf[-1], tfidf)

#Get the index of the most similar text/sentence to the users response
idx = vals.argsort()[0][-2]

#Reduce the dimensionality of vals
flat = vals.flatten()

#sort the list in ascending order
flat.sort()

#Get the most similar score to the users response
score = flat[-2]

if(score == 0):
  robo_response = robo_response+"I apologize, I don't understand."
else:
  robo_response = robo_response+sent_tokens[idx]
```




```
# PARAMETERS CHOICE

# Activation = RELU
# The size of the output layer is 'vocabulary_size'
# Loss = 'categorical_crossentropy'

def create_model(vocabulary_size, seq_len):
    model = Sequential()
    # Embedding turns positive integers(indexes) into dense vectors of fixed size (see docs).
    model.add(Embedding(vocabulary_size, 20, input_length=seq_len))
    model.add(LSTM(150, return_sequences=True)) # better to take multiples of seq_len; smalle batches => faster
    model.add(LSTM(150))
    model.add(Dense(150, activation='relu'))

    model.add(Dense(vocabulary_size, activation='softmax'))

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

    model.summary()

    return model
```

Chatbot v.3 (Python and Keras)

Text generation with LSTM (Long-Short Term Memory cells)

x(t) – input, h(t) output from previous revolution (recurrence), c(t) is a cell state (an approximation of memory)

Chatbot v.3 (Python and Keras) (cont.)

Text generation with LSTM (Long-Short Term Memory cells)

```
[ ]  seed_text = ' '.join(random_seed_text)
     seed_text
```

```
'cadre d’ un contrat antérieur avec un organisme public du québec fait l’ objet d’ une évaluation de rendement insatisfaisant de'
```

```
[ ]  ## GENERATED NEW TEXT !!!

     generate_text(model,tokenizer,seq_len,seed_text=seed_text,num_gen_words=20)
```

```
'la part de cet organisme public ne pas faire l’ objet d’ une requête en faillite volontaire ou involontaire ou'
```

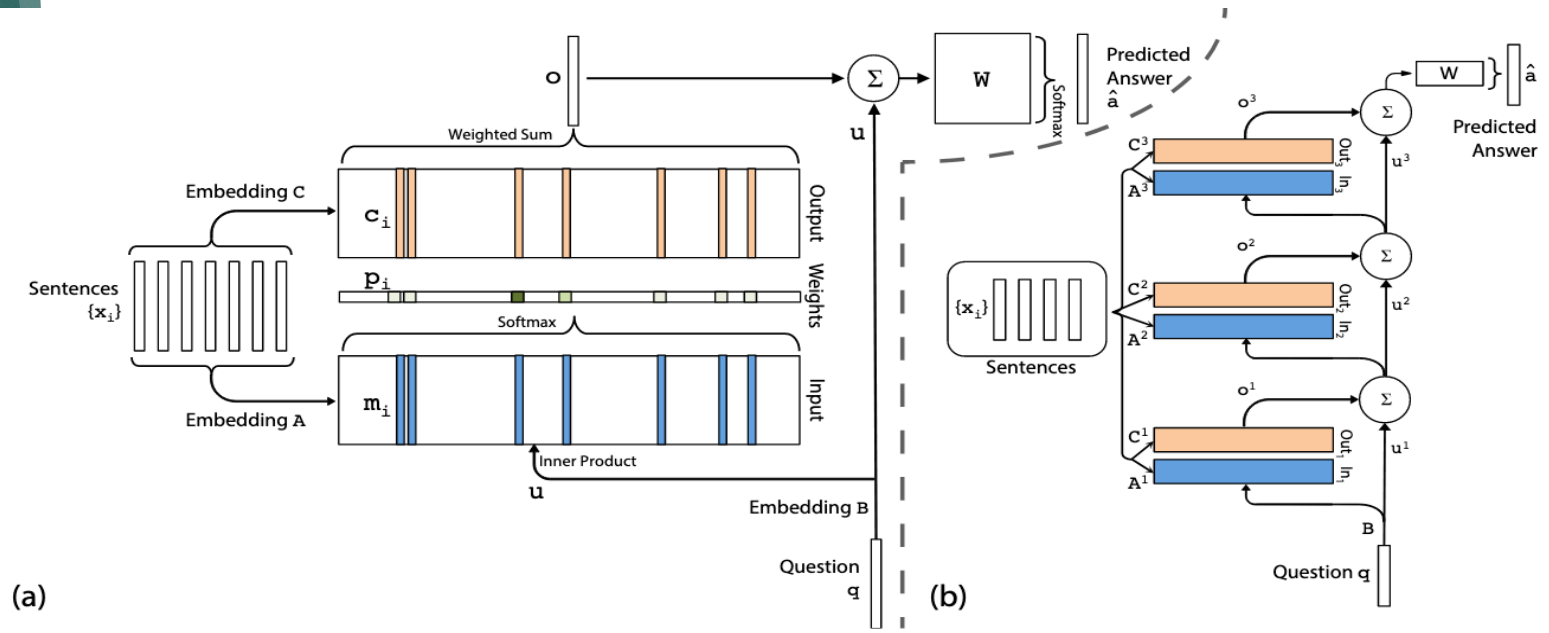
Chatbot v.4
(unfinished)

RNN with a recurrent attention model over a possibly large external memory.

On the left: a single memory hop operation.

On the right: RNN made of multiple memory hop cells

This model has an outside vocabulary V.



This model requires training on pre-prepared questions and answers