Data Science Project

Deep Learning : Creating a Cook Recipe using Neural Networks.

Team Members :

Nicolas BOURASSEAU

Giang NGUYEN

David CANO Date : 24th of October 2022

## 

[**1) Topic introduction**](#_heading=h.qx3837bxziok) **3**

[**2) Dataset Introduction.**](#_heading=h.2x5q6nfv6fnz) **3**

[2.1 Data set exploration :](#_heading=h.cvjkld2d2iy) 3

[1) Recipe NLG Dataset:](#_heading=h.irc9papq12ov) 3

[2) Recipe\_foodcom Dataset:](#_heading=h.7tulmq43i0kb) 5

[**3) Modelisation**](#_heading=h.p9uqw0pnr2rn) **8**

[1. LSTM text generation model](#_heading=h.3z0rhbwahn6t) 8

[Data preparation](#_heading=h.7sbav5mj3z9s) 8

[Text Generation Function](#_heading=h.6x26qxrodam4) 9

[2. Fine-tuning of Transformer model « DistilGPT2 » from HuggingFace](#_heading=h.e8fa3mul5e3) 9

[3. Encoder\_Decoder models](#_heading=h.a41c2y2c8dbq) 10

[Data preparation](#_heading=h.vx462bgrc6x9) 11

[Tokenization](#_heading=h.32ys91j6sovs) 11

[Model creation and Training](#_heading=h.7rebaea9kbjk) 12

[Model utilisation with text generation function](#_heading=h.rmv1x37u971f) 12

[**4) Evaluation and result interpretation**](#_heading=h.2p3hgzyfh82j) **13**

[The BLEU evaluation :](#_heading=h.30j0zll) 13

[BLEU 1 - For the uni-grams (word par word).](#_heading=h.m8d0tb4uaob4) 14

[BLEU 2 - For the bigrams (pair of words).](#_heading=h.cnlpec7q8zfe) 14

[BLEU 3 - For the trigrams (triple of words).](#_heading=h.h5wj6izanalx) 14

[BLEU 4 - For the quadgrams ( of words).](#_heading=h.4s80bu75jjqu) 14

[The ROUGE Evaluation :](#_heading=h.1fob9te) 14

[ROUGE-1](#_heading=h.ojdom8l9tev9) 15

[ROUGE-2](#_heading=h.aqxispajjxpa) 15

[ROUGE-L](#_heading=h.o63zj9xcqfu6) 15

[The METEOR Evaluation :](#_heading=h.vfx9ypguhe28) 15

[METEOR](#_heading=h.elg1z3rpvt0s) 15

[**5) General Conclusion.**](#_heading=h.sjbojkpehgoa) **15**

## 

## 1) Topic introduction

The Project assigned to our Team is: **auto completion assistant**, this would be the basis around which, as a team, we would have to define a theme and an objective that would use our knowledge acquired in the DataScientest course in a real-life based problem.

In our case, the team decided to create a recipe completion assistant. The objective of the project is to use neural networks from Deep Learning in order to use a database obtained from the internet and use this data to train a model that can create a recipe from a given sentence/word.

## 2) Dataset Introduction.

The first thing the team did was to search what kind of datasets exist on the web in order to understand which type of databases already exists and how to relate the database to our problem. There were several options as language based deep learning projects are relatively common. This richness allowed us to narrow down our choice of a dataset to two interesting options that could work for our project.

Both are datasets containing cooking recipes found from a website called Kaggle. The datasets can be found here :

1. Recipe NLG Dataset : <https://www.kaggle.com/datasets/paultimothymooney/recipenlg>
2. Recipe\_foodcom :<https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions?select=RAW_recipes.csv>

### **2.1 Data set exploration** :

In order to choose which one of these data sets we used we did an exploration on each data set in order to identify the one that adapts better to our problem, the results are as follows:

#### Recipe NLG Dataset:

* **Number of recipes in the dataset :** 2,231,142.
* **Authors** : Michał Bien, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, Agnieszka Ławrynowicz.

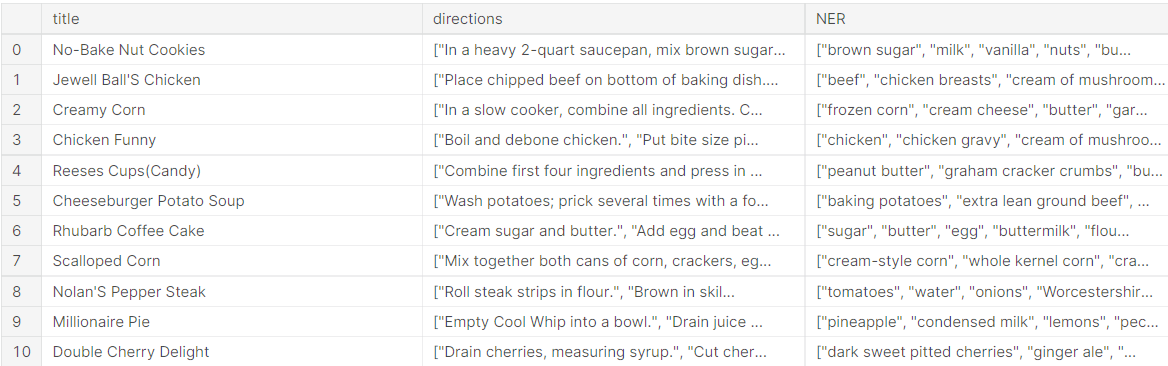
Dataset structure:



By doing a recipe.head(10) we see the first ten recipes as follows :

recipe\_df = pd.read\_csv('../input/recipenlg/RecipeNLG\_dataset.csv')

recipe\_df[['title','directions','NER']].head(10) :



This Dataset contains all needed information in order to train a model in order to try and autocomplete a recipe as it contains a breakdown of ingredients, directions, name of recipe.

Going further, we can do a word cloud for the most important columns in order to understand what are the most used words. This can help identify what input we are giving our model for training :

For the column **‘Title’**, using the following function :

def minimal\_wordcloud(df, column):

    text = str(df[column].values)

    wordcloud = WordCloud().generate(text)

    image = wordcloud.to\_image()

    return image

minimal\_wordcloud(recipe\_df,'title')

We obtain the word cloud as follows :

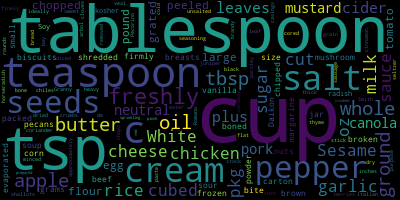


It looks like cookies is a very common word probably linked to Bake. In all, it is what we would expect from recipe titles with words like Chicken, Pan, Roasted, etc.

For the column **‘Ingredients’** :

Using the following code : minimal\_wordcloud(recipe\_df,'ingredients')

We obtain the word cloud as follows :



We see that most of the words in ingredients contain a measurement such as cup, tablespoon, etc. This could represent a problem since these words are linked to American ways of writing recipes and therefore the measurements are not as standardised as we would like them to be.

**Advantages :**

What we found in this dataset is that it has 7 main columns, so less data to work with, while containing the ingredients list, instructions and a big number of recipes, which means better training possibilities for our model.

**Disadvantages :**

The Dataset has an important issue for us, as the measurements for the ingredients are not as clear as in the metric system, this could create bugs as it could result in a recipe with further incoherences.

#### Recipe\_foodcom Dataset:

* **Number of recipes in the dataset :** 180,000 recipes.
* **Authors** : Bodhisattwa Prasad Majumder\*, Shuyang Li\*, Jianmo Ni, Julian McAuley.

Dataset structure:

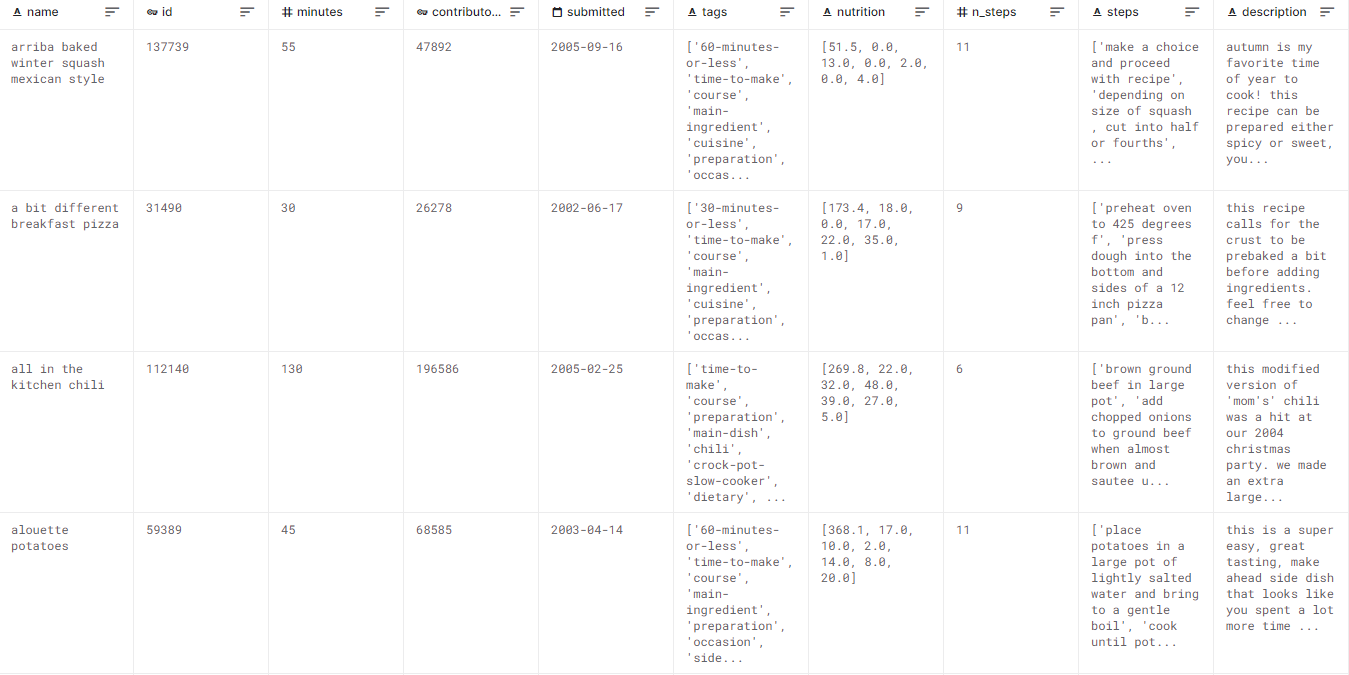


By doing

Recipe\_foodcom = pd.read\_csv("drive/My Drive/DataScientest - NLP\_Mail/Foodcom Recipes \_ archive/RAW\_recipes.csv", sep = ",")

Recipe\_foodcom.head(4)

We get the first 4 recipes as follows:

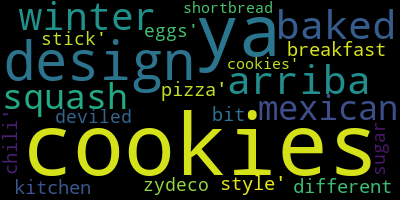


This Dataset has more columns, data that we consider important are the number of steps per recipe, the steps cut in an standardised manner and the ingredients needed for the recipe.

By using again the minimal\_wordcloud function we can see the following word clouds :

For the column **‘Name’**

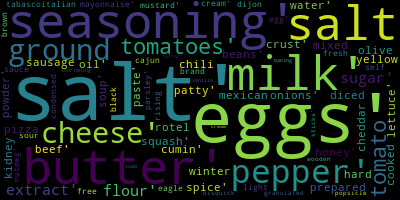
minimal\_wordcloud(Recipe\_foodcom, 'name')



From this wordcloud we see that cookies and baked are amidst the most common words, the same as in the previous dataset.

For the column **‘Ingredients’**

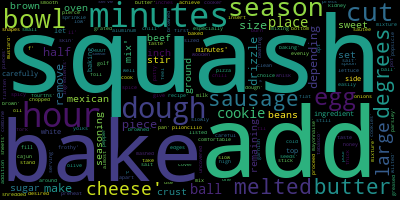
minimal\_wordcloud(Recipe\_foodcom, 'ingredients')



In this wordcloud we see proper ingredients such as Salt, Pepper, Eggs, etc. This word distribution is much clearer and ingredient centred vs the previous dataset.

For the column **‘Steps’**

minimal\_wordcloud(Recipe\_foodcom, 'steps')



The steps here are a plus since they are cut already in phrases and they begin with an action like squash, season, etc.

Dataset choice.

After doing this analysis on the data sets the team chose to work with the second data set Recipe\_foodcom because of the following reasons:

* It contained better cut steps of recipes, not only where the number of steps enumerated but also they were precisely cut in phrases that contained an action verb such as squash, add, bake, etc.
* The ingredients were already isolated and there was no real need to further work on these, this means one less cleaning step of the database.
* The database contains “only” 180 k recipes, but the team decided it was more than enough since +2m recipes can also be too many for the training of data

As an important note, we could still do more explorations on the data set, like the number of phrases in each recipe, the category of recipe, if there could have been any classification of steps or recipe. But for the intents and purposes of the project we decided that we already had enough elements to choose the second dataset.

## 3) Modelisation

### 1. LSTM text generation model

For our first text generation model, we had chosen to build it with a LSTM layer in order to get familiar with the training and layer functioning of RNN type. The model will be of type “many to one”.

#### Data preparation

The data is prepared with the following process :

* To help the model to distinguish between the end of the ingredient part and the beginning of the recipes steps, we have added as boundary token a sequences of « | » symbols
* Combine all recipes into one single string variable
* Split this text in sequences of 20 tokens, each sequence being formed by shifting previous one by one token
* These sequences will serve as an input for the model, with target being the next immediate token.

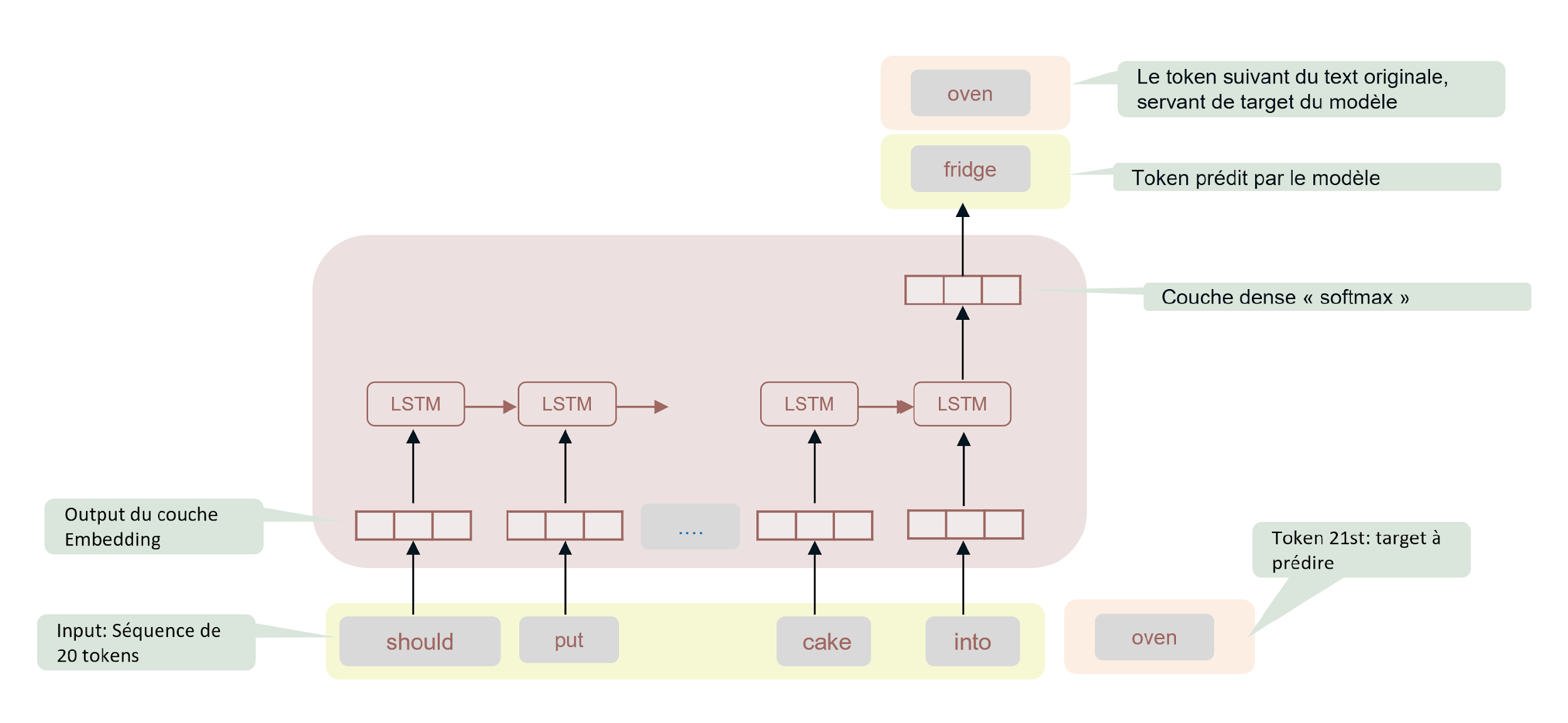
Model structure

The model is constituted of 4 layers as follows :

|  |  |
| --- | --- |
| LSTM Model | * Embedding (dim = 100) * LSTM (units = 256, return\_sequence = False) * Dropout (0.2) * Dense (units = size\_vocabulaire, activation = softmax) |

The LSTM layer returns an output only after having treated the last token of the input sequence (return\_sequences = False).

The model is then compiled with the loss function « categorical crossentropy » and trained on 5 epochs.



#### Text Generation Function

As the model predicts only the next token in each launch, we have created an iterative function to generate new phrases. In each loop, the function will add the token predicted by the model to the input sequence, then relaunch the prediction with this new sequence as input.

The output layer of Dense type with softmax activation produces a vector of the vocabulary size, containing the probability of each word. If we always choose the word with the highest probability, the resulting sequence will contain repetition of a few words. This is why we have created a sampling function to add variability on the choice of the predicted word at each relaunch of the model.

### 2. Fine-tuning of Transformer model « DistilGPT2 » from HuggingFace

On our project mentor's suggestion, we created the 2nd model using HuggingFace, a platform for hosting Transformers models which have been pre-trained on very large databases and ready for fine-tuning. This exercise makes us familiar with the process of transfer-learning because fine tuning pre-trained models, instead of training from scratch, is common practice in Deep Learning.

Besides being a Transformer host, HuggingFace provides libraries like Pipeline, Model & Tokenizer to standardise and facilitate the exploitation and fine-tuning of models.

We have chosen to fine-tune the DistilGPT2 model of the Text Generation use case. The process consists of the following steps :

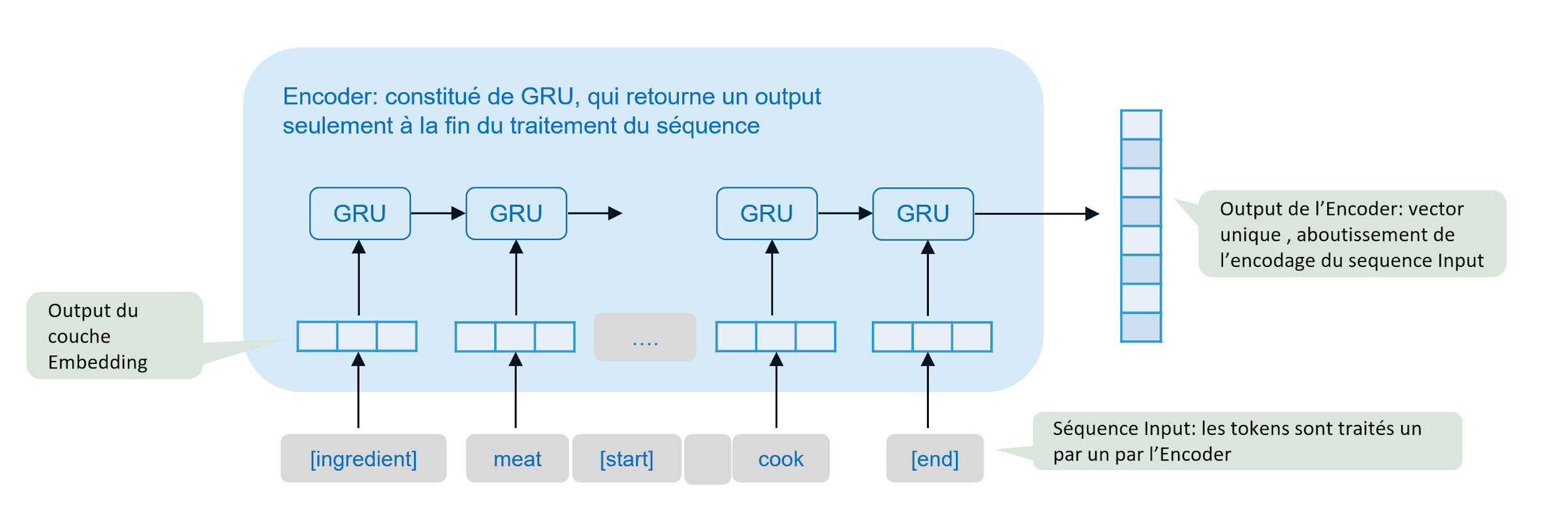
* From the initial dataframe file, only keep the “Recipe” column, which for each recipe, contains the list of ingredients and all the steps in one single string
* Load the « Recipe » column into the Dataset container of HuggingFace (it is easier to use their Dataset object as container to prepare the data and then feed them into the training loops)
* Pre-processing the data by following steps:
  + Define a Tokenize function which standardised le texte (removing special symbols) then apply the tokenizer associated with DistilGPT2 model
  + Define a Group\_Texte function, which concatenates all the recipes into one single texte, then splits this texte into sequences of fixed length (50 tokens). The goal is to obtain fixed length sequence as input for the model
  + Apply Tokenize then Group\_Text function on all dataset via .map method of HuggingFace Dataset
* Download, compile and train the DistilGPT2 model with tokenized data resulting from the previous step.

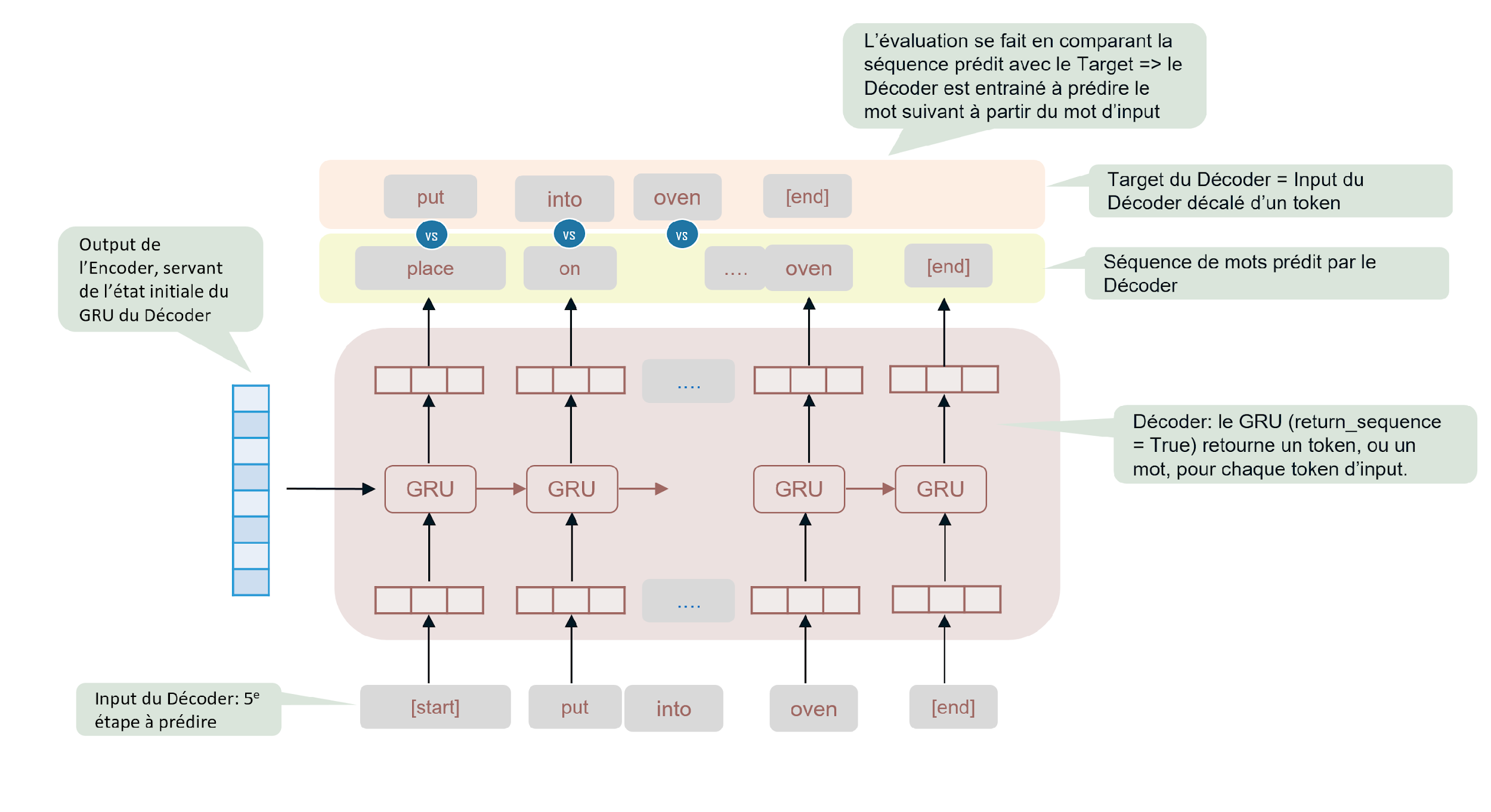
### 3. Encoder\_Decoder models

Following the text generation model, where the target is the input sequence shifted by one token, we have worked on a Encoder-Decoder model, of sequence-to-sequence type, where the target & input sequences are totally distinct. The goal is to increase the types of models in our project as well as to the difference of results between the two approaches.

To understand and develop this model type, whose structure is more complex than a text generation model, we have learned from the RNN course of DataScienceTest, as well as from Deep Learning & Tensorflow books. These books give detailed explanations on the inner workings of sequence to sequence models. One challenge for us is that their code examples are applied to translation use case. Thus we have to spend time and effort to understand them and adapt to our specific use case.

Small reminder on Encoder-Decoder models :





#### Data preparation

The following schema is chosen for preparation of inputs and targets:

* Input for Encoder : ingredient sequence and four first steps of recipe
* Input for Decoder : the fifth step of the recipe
* Target for Decoder: the fifth step shifted by one token

These choices were done to simplify the process, while keeping a significant number of steps in the inputs to enable the prediction of the model. (vs input of only 1 or 2 steps from the beginning).

Marking of sequences : to guide the model, we add the following marking tokens :

* [ingredient] & [/ingredient] to mark the beginning and end of ingredient sequence
* [start] & [end] to mark the beginning and end of the 4 steps in the input
* [start] & [end] to mark the beginning and end of the 5th step, (token [start] serving as first token fed to the decoder)

#### Tokenization

The tokenisation is done word by word and with the TextVectorisation from Keras.

To fix the size of the vocabulary, we calculated the number of unique words in our samples, with their appearance frequency. We chose a size of 70k words, covering around 90% of words of frequency above 3.

To choose the tokenisation max length, we have calculated the statistique distribution of inputs & targets lengths, then took a value of around the 75% percentile + gap (75% - 50%)

* max\_input\_len : 56 + (56 – 46) => max\_input\_len = 70
* max\_target\_len : 13 + (13 – 9) => max\_target\_len = 20

|  |  |
| --- | --- |
| Distribution of Input sequence lengths | Distribution of Target sequence lengths |
|  |  |

After vectorisation, the data is put into the following format, using the map function of DataSet object in Tensorflow, to feed it to the model :

Tuple ( dictionary { « encoder\_input » : Vectorized encoder input

« decoder\_input » : vectorized 5th step, excluding the last token }

, vectorized 5 thstep, excluding the 1st token )

The 1st element of the tuple is a dictionary, serving as inputs for the Encoder and Decoder. The second element serves as a target for the Decoder.

#### Model creation and Training

We built the model with the following structures :

|  |  |
| --- | --- |
| Encoder | * Embedding (dim = 256) * GRU (units = 1024) |
| Decoder | * Embedding (dim = 256) * GRU (units = 1024, initial state = encoder\_output) * Dropout (0.5) * Dense (units = size\_vocabulaire, activation = softmax) |

As the evaluation is done comparing 2 sequences of words, the loss function is of « Sparse categorical crossentropy » type.

#### Model utilisation with text generation function

As the model is trained to predict only the next word in each go, to predict a whole step we need to launch the prediction model in several loops, reusing the previous loop’s output as an input for the next loop.

We created then a function to iterate this process:

* Encoder input = ingredients and the 4 first steps.
* Initiate the decoder\_input : decoder\_input = [start]
* Launch predicting model with encoder\_input & decoder\_input
* Add the last token of predicted sequence to decoder\_input
* Relaunch the predicting model with new decoder\_input
* repeat the last 2 steps until obtaining the token [end] or reaching the maximum length of sequence target (=20 tokens).

## 4) Evaluation and result interpretation

In order to produce the most reliable model possible we need to measure it. The model is firstly evaluated by the human method at the training level in order to understand if a standard phrase is plausible. Once the phrase is validated, we run a more scientific evaluation based on calculated metrics. The metrics we have chosen to evaluate our model are BLEU, ROUGE, METEOR.

The evaluations have been sampled on several phrases (5) and evaluated against our recipe corpus

#### The BLEU evaluation :



The BLEU metric or **BiLingual Evaluation Understudy** gives a percentage of similarity between a candidate text (our generated text) and a reference text (our corpus). This metric is decomposed in n-grams, which consists in counting the number of uni-grams (unique words), bigrams (pair of words), trigammes and quadgrammes (i = 1,…, 4) corresponding between the two elements, generated text and the recipe corpus.   
  
The unigram (Bleu N-1), allows us to validate that each generated word exists in our reference corpus. Then the same for bigrams, trigrams and quadgrams to verify the fluency of the candidate and if the linking of words is correct.

This means that we can divide our evaluations in the following way :

##### BLEU 1 - For the uni-grams (word par word).

We are above 90% score, this means that the words used are included in the recipe book.

##### BLEU 2 - For the bigrams (pair of words).

We are at an average of 80% with a downgrading to 60% in some cases. This means that the model is able to link two words that are coherent between them.

##### BLEU 3 - For the trigrams (triple of words).

We are at an average of around 65%. This means that the model is still able to link three words that are coherent between them, however the evaluation score is weakening compared to BLEU 1 and BLEU 2. This could be explained by the fact that the word is not the exact word in the recipe book but a synonym.

##### BLEU 4 - For the quadgrams ( of words).

We are at an average of around 40%, with changes between 70% and 10%. This means that the coherence is inconsistent between the words generated and the recipe book.

#### The ROUGE Evaluation :

The ROUGE metric, or **Recall-Oriented Understudy for Gisting Evaluation** was initially conceived to evaluate the synthesis of texts. However, we can use it for the evaluation of text translation, text syntax and the correspondence of text.

The ROUGE-N decomposes the number of matches from the generated text with the candidate text. Then in principle for the precision, it is very similar to the BLEU metric. The decomposition of data is done with the precision, recall and F1.

The recall :

The recall on the other hand, is expected to be very high as our generated text is based on the reference text.

F1:

ROUGE-N F1-score = 2 \* (precision \* recall) / (precision + recall)

We can divide our results in the following way :

##### ROUGE-1

We are at a 65% average score. This means that the coherence of words with the recipe is correct for creation, closer to 100% is more a search tool, and below 50% the coherence is not good enough for creation.

##### ROUGE-2

We are at around 15% of average score. This means that we have words that are linked to recipes but the coherence between them don’t make for actual recipes.

##### ROUGE-L

We are at around 40% of average score. This says the percentage of the phrase generated that is in the recipe book. So, 40% of our phrase is found on the book, so a certain logic is kept in the phrase.

#### The METEOREvaluation **:**

The METEOR metric or **Metric for Evaluation of Translation with Explicit ORdering,** is based on the BLEU and ROUGE metrics principle. METEOR nonetheless adds the benefit of being able to match the n-grams with exact value, verbally different or synonym. This is one of the most polished methods , but it is heavier compared to the other 2 and therefore demands more resources to process it.

##### **METEOR**

We are at an average score of 45%. This means that the F1 score between BLEU and ROUGE multiplied by the penalty gives 45%.

## 5) General Conclusion.

The objectives of the project were to produce a model of text generation that was linked to our DataScientest courses and that would be functional in creating recipes from an input. These objectives were met, as we managed to create different models that do text generation.

The models create phrases that are understandable by a human, however from the 8th word we begin to lose the coherence of the phrases and unwanted repetitions. For us this means that from a scholar point of view it is successful as we learned collectively how to create these models within the competences we gained in the course.

If we wanted to push this project more to a professional side, we have the following suggestions to improve it :

- The lack of experience showed at the beginning of the project since we had no previous work on model creation, so we had many push backs in order to realign the project as we learned how to work on deep learning.

- The lack of understanding on how to configure the computer resources made it difficult to train our model, so we lost time that would have otherwise been used to improve our model. As an example we put the computer to work through the night and many times the process was stopped due to a problem while launching the model.

**What did we learn from this project?**

- Finding a data source for deep learning

- Understanding how the data source could be used for the project

- Cleaning and tokenizing of data

- How to create an NLP model

- How to evaluate the model according to standardised metrics

- How to mutualise a coding project between several team members

**How we could have improved our model?**

- We understood there were some grammatical errors in the texts produced by the model, which do not have any punctuation. The reason is that we have removed all punctuation from the text during the preprocessing phase. We could have improved the model by keeping the punctuation

- Add more layers into the model structure (more LSTM layers instead of a single one, more GRU layers in the encoder or decoder).

- We could have created a Reinforcement Learning model, where we could apply a different way of recipe writing which could make the generated recipes more coherent.

- We could have also gone deeper on how to understand the recipes, actually we are based only on text, an alternative is to base ourselves also on cooking limitations such as cooking times or reaction between ingredients. For example the model could say “cook pork for 5 minutes” syntax is correct, but there is a problem as the end result would not be proper in real life.

As a final conclusion, we would like to thank Pierre Adeikalam, our project mentor, for his advice during the project. We also appreciate the project content, which despite being very challenging, has been very formative for us.