Natural language Processing

Hichem Felouat

hichemfel@gmail.com

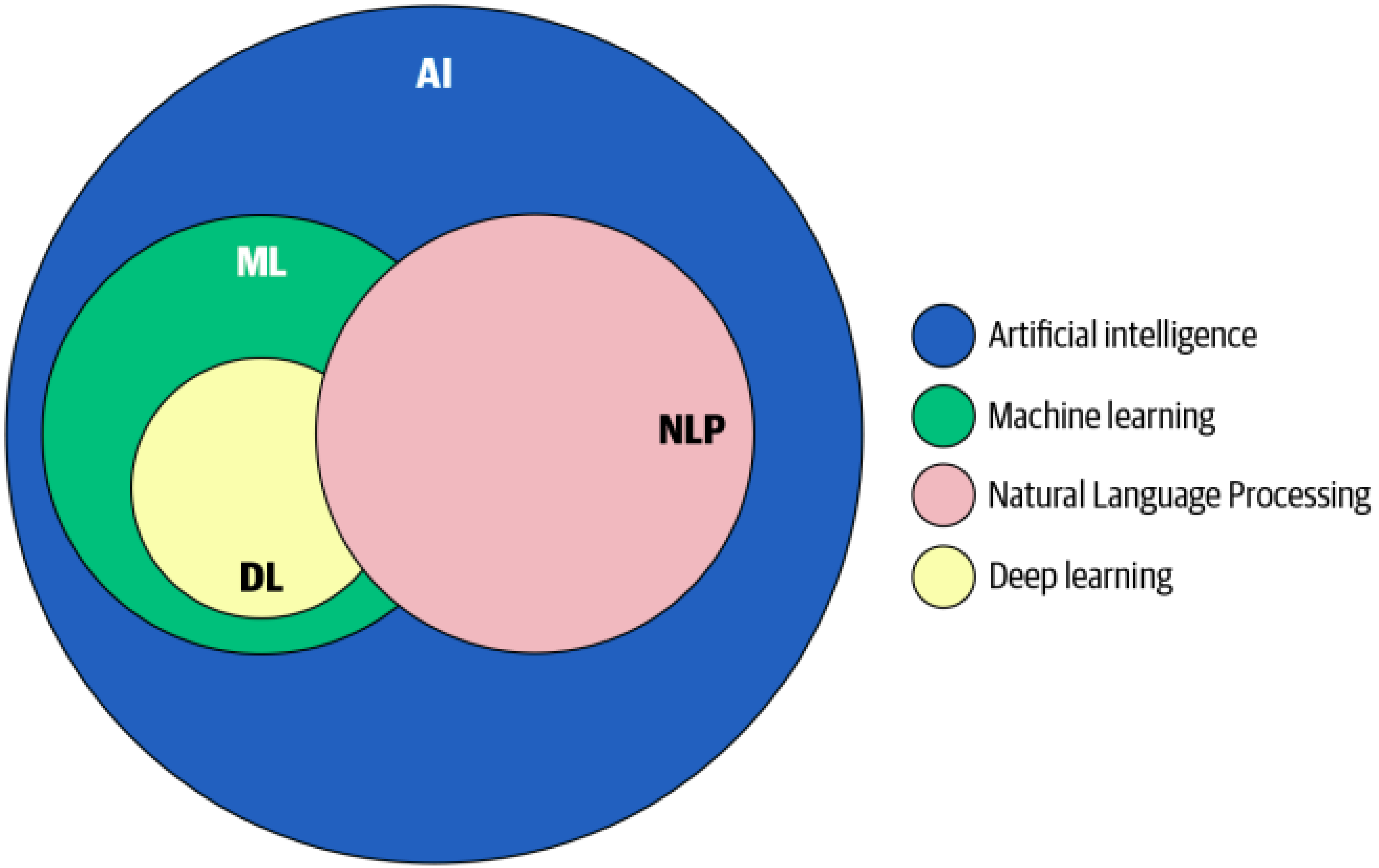
https://www.linkedin.com/in/hichemfelouat/

Natural language Processing

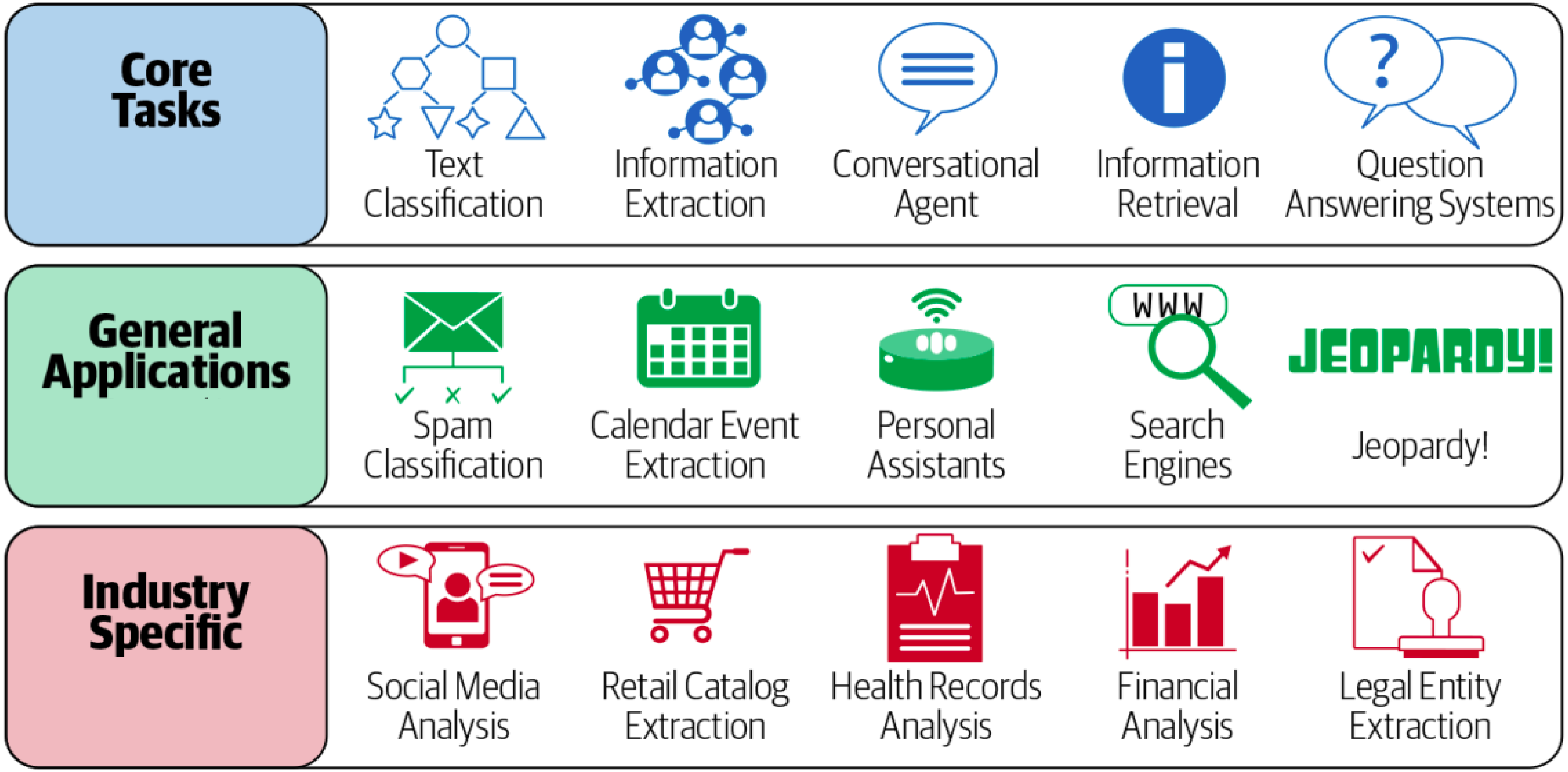
- **Natural language processing (NLP)** is a subfield of **artificial intelligence** concerned with the **interactions** between **computers** and **human** (natural) **languages**, in particular how to program computers to process and analyze large amounts of natural language data.

- **Challenges in natural language processing** frequently involve **speech recognition**, **natural language understanding**, and **natural language generation**.

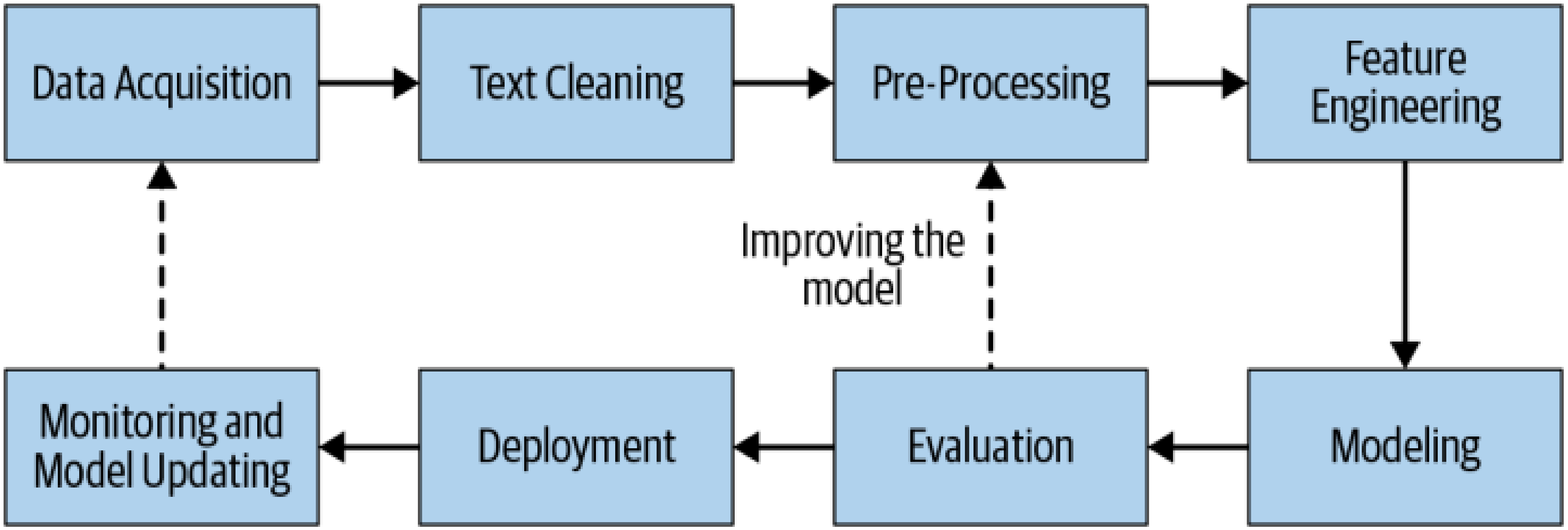
Natural language Processing



NLP Tasks and Applications



Generic NLP Pipeline



Tokenizer API

- **word_counts:** A dictionary of words and their counts.
- **word_docs:** A dictionary of words and how many documents each appeared in.
- **word_index:** A dictionary of words and their uniquely assigned integers.
- **document_count:** An integer count of the total number of documents that were used to fit the Tokenizer.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

```
texts = ["I love Algeria", "machine learning", "Artificial intelligence","AI"]
tokenizer = keras.preprocessing.text.Tokenizer(char_level=True)
# tokenizer = keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(texts)
```

```
print("Total number of documents : \n",tokenizer.document_count)
print("Number of distinct characters/words: \n",len(tokenizer.word_index))
print("word_index : \n",tokenizer.word_index)
print("word_counts : \n",tokenizer.word_counts)
print("word_docs : \n",tokenizer.word_docs)
print("texts_to_sequences : (Algeria) \n",tokenizer.texts_to_sequences(["Algeria"]))
print("sequences_to_texts : \n",tokenizer.sequences_to_texts([[4, 3, 7, 2, 8, 1, 4]]))
```

Tokenizer API - Char

Total number of documents :
 4
Number of distinct characters/words :
 15
word_index :
 {'i': 1, 'e': 2, 'a': 3, 'l': 4, 'n': 5, ' ': 6, 'g': 7, 'r': 8, 'c': 9, 't': 10, 'o': 11, 'v': 12, 'm': 13, 'h': 14, 'f': 15}
word_counts :
 OrderedDict([('i', 10), (' ', 4), ('l', 6), ('o', 1), ('v', 1), ('e', 7), ('a', 7), ('g', 3), ('r', 3), ('m', 1), ('c', 3), ('h', 1), ('n', 5), ('t', 2), ('f', 1)])
word_docs :
 defaultdict(<class 'int'>, {'a': 4, 'v': 1, 'r': 3, ' ': 3, 'g': 3, 'o': 1, 'l': 3, 'e': 3, 'i': 4, 'c': 2, 'h': 1, 'm': 1, 'n': 2, 'f': 1, 't': 1})
texts_to_sequences : (Algeria)
 [[3, 4, 7, 2, 8, 1, 3]]
sequences_to_texts :
 ['a l g e r i a']

Tokenizer API - Words

Total number of documents :

4

Number of distinct characters/words :

8

word_index :

{'i': 1, 'love': 2, 'algeria': 3, 'machine': 4, 'learning': 5, 'artificial': 6, 'intelligence': 7, 'ai': 8}

word_counts :

OrderedDict([('i', 1), ('love', 1), ('algeria', 1), ('machine', 1), ('learning', 1), ('artificial', 1), ('intelligence', 1), ('ai', 1)])

word_docs :

defaultdict(<class 'int'>, {'i': 1, 'love': 1, 'algeria': 1, 'machine': 1, 'learning': 1, 'artificial': 1, 'intelligence': 1, 'ai': 1})

texts_to_sequences : (Algeria)

[[3]]

sequences_to_texts :

['algeria machine intelligence love ai i algeria']

texts_to_sequences

```
# Let's encode the full text so each character/word is represented by its ID
encoded = tokenizer.texts_to_sequences(texts)
print("Encode the full text : \n",encoded)
```

Char :

{'i': 1, 'e': 2, 'a': 3, 'l': 4, 'n': 5, ' ': 6, 'g': 7, 'r': 8, 'c': 9, 't': 10, 'o': 11, 'v': 12, 'm': 13, 'h': 14, 'f': 15}

==> [[1, 6, 4, 11, 12, 2, 6, 3, 4, 7, 2, 8, 1, 3], [13, 3, 9, 14, 1, 5, 2, 6, 4, 2, 3, 8, 5, 1, 5, 7], [3, 8, 10, 1, 15, 1, 9, 1, 3, 4, 6, 1, 5, 10, 2, 4, 4, 1, 7, 2, 5, 9, 2], [3, 1]]

Word :

{'i': 1, 'love': 2, 'algeria': 3, 'machine': 4, 'learning': 5, 'artificial': 6, 'intelligence': 7, 'ai': 8}

==> [[1, 2, 3], [4, 5], [6, 7], [8]]

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(encoded, y, test_size=0.20,
                                                    random_state=100)
```

texts_to_matrix

encoded_docs = tokenizer.texts_to_matrix(texts, mode="tfidf")

- **binary** : Whether or not each word is present in the document. This is the default.
- **count** : The count of each word in the document.
- **freq** : The frequency of each word as a ratio of words within each document.
- **tfidf** : The Text Frequency-Inverse Document Frequency (TF-IDF) scoring for each word in the document.

texts_to_matrix

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Sequence Padding

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [ [1, 2, 3, 4], [1, 2, 3], [1] ]
# Padding sequence data
result = pad_sequences(sequences, maxlen=4, truncating='post')
print("result : \n",result)
```

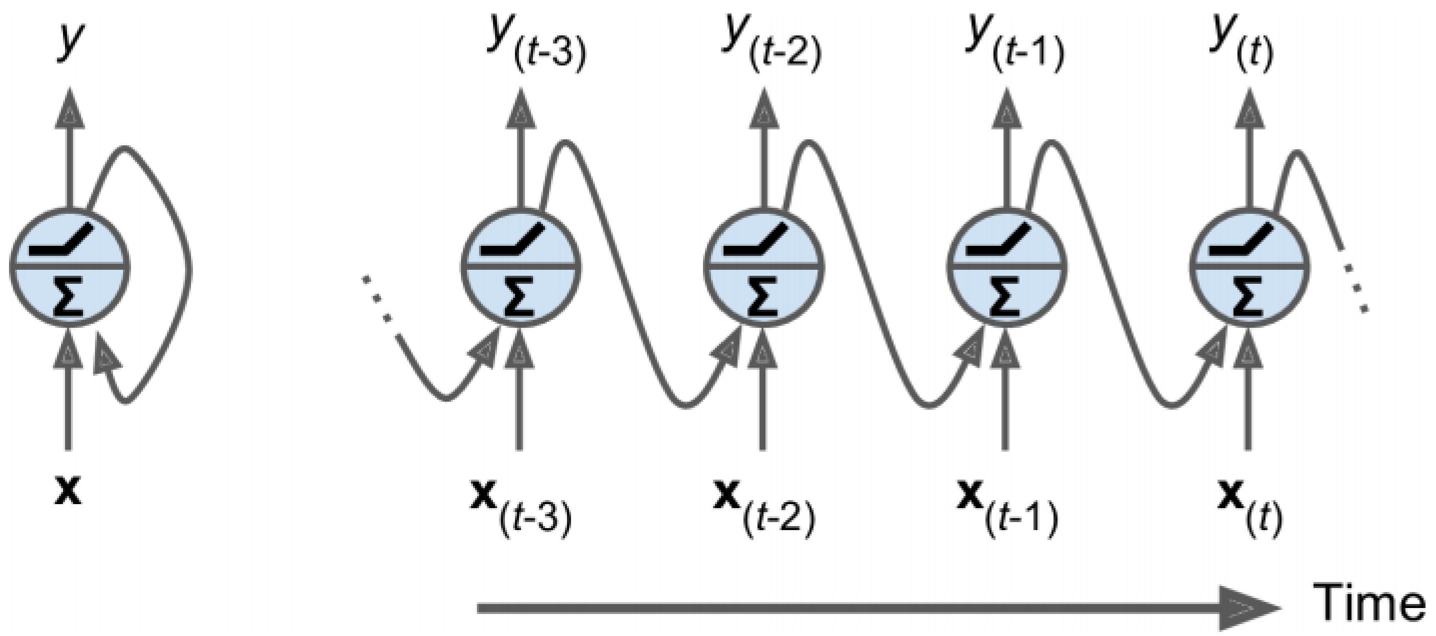
result :

```
[[1 2 3 4]
 [0 1 2 3]
 [0 0 0 1]]
```

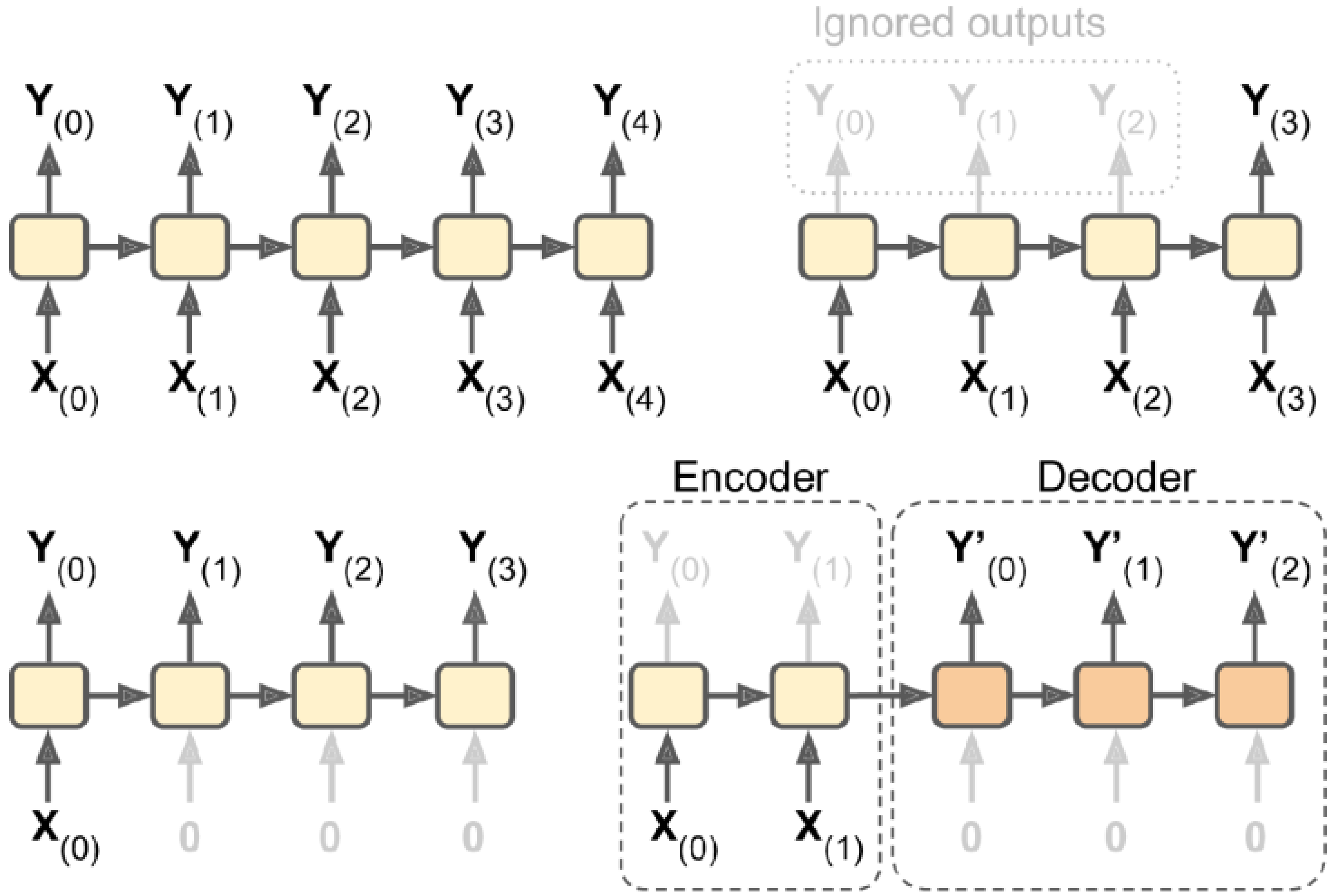
Recurrent Neural Network(RNN)

- The **simplest** possible RNN composed of **one neuron** receiving inputs, producing an output, and **sending that output back to itself** (figure -left).

- We can represent this tiny network against the time axis, as shown in (figure - right). This is called **unrolling the network through time**.



Recurrent Neural Network(RNN)



Seq-to-seq (top left), seq-to-vector (top right), vector-to-seq (bottom left), and Encoder–Decoder (bottom right) networks.

Deep RNNs

```
model = keras.models.Sequential([
keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
keras.layers.SimpleRNN(20, return_sequences=True),
keras.layers.SimpleRNN(1) ])
```

- Make sure to set return_sequences=True for all recurrent layers except the last one, if you only care about the last output.

- It might be preferable to replace the output layer with a Dense layer.

```
model = keras.models.Sequential([
keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
keras.layers.SimpleRNN(20),
keras.layers.Dense(1) ])
```

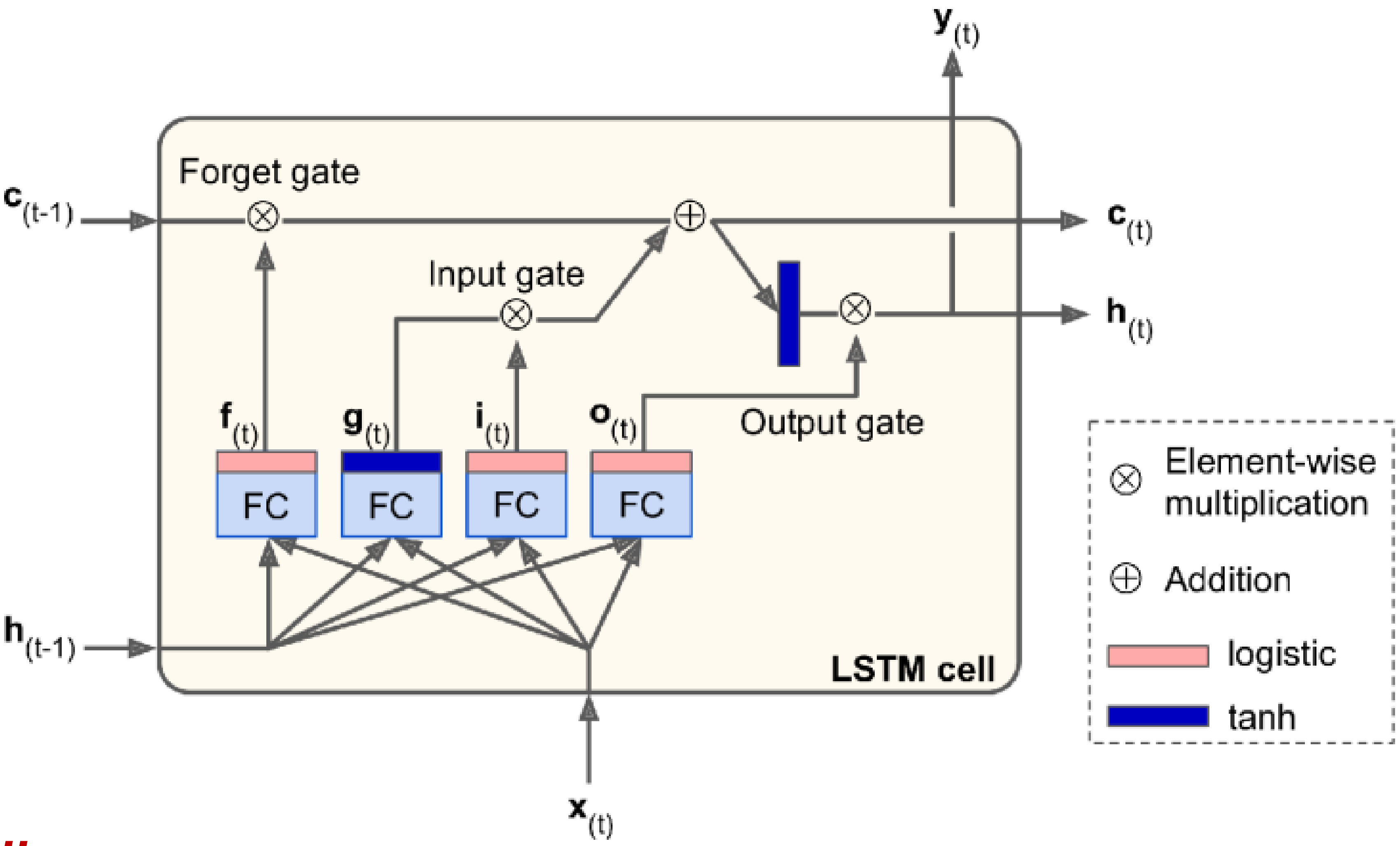
Deep RNNs

- To turn the model into a **sequence-to-sequence** model, we must set **return_sequences=True** in all recurrent layers (even the last one), and we must apply the output Dense layer at every time step.

- Keras offers a **TimeDistributed layer** for this very purpose: it wraps any layer (e.g., a Dense layer) and applies it at every time step of its input sequence.

```
model = keras.models.Sequential([
keras.layers.SimpleRNN(20, return_sequences=True,
                                        input_shape=[None, 1]),
keras.layers.SimpleRNN(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10))  ])
```

Long Short-Term Memory (LSTM)



LSTM cell

Long Short-Term Memory (LSTM)

```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True,
                                        input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

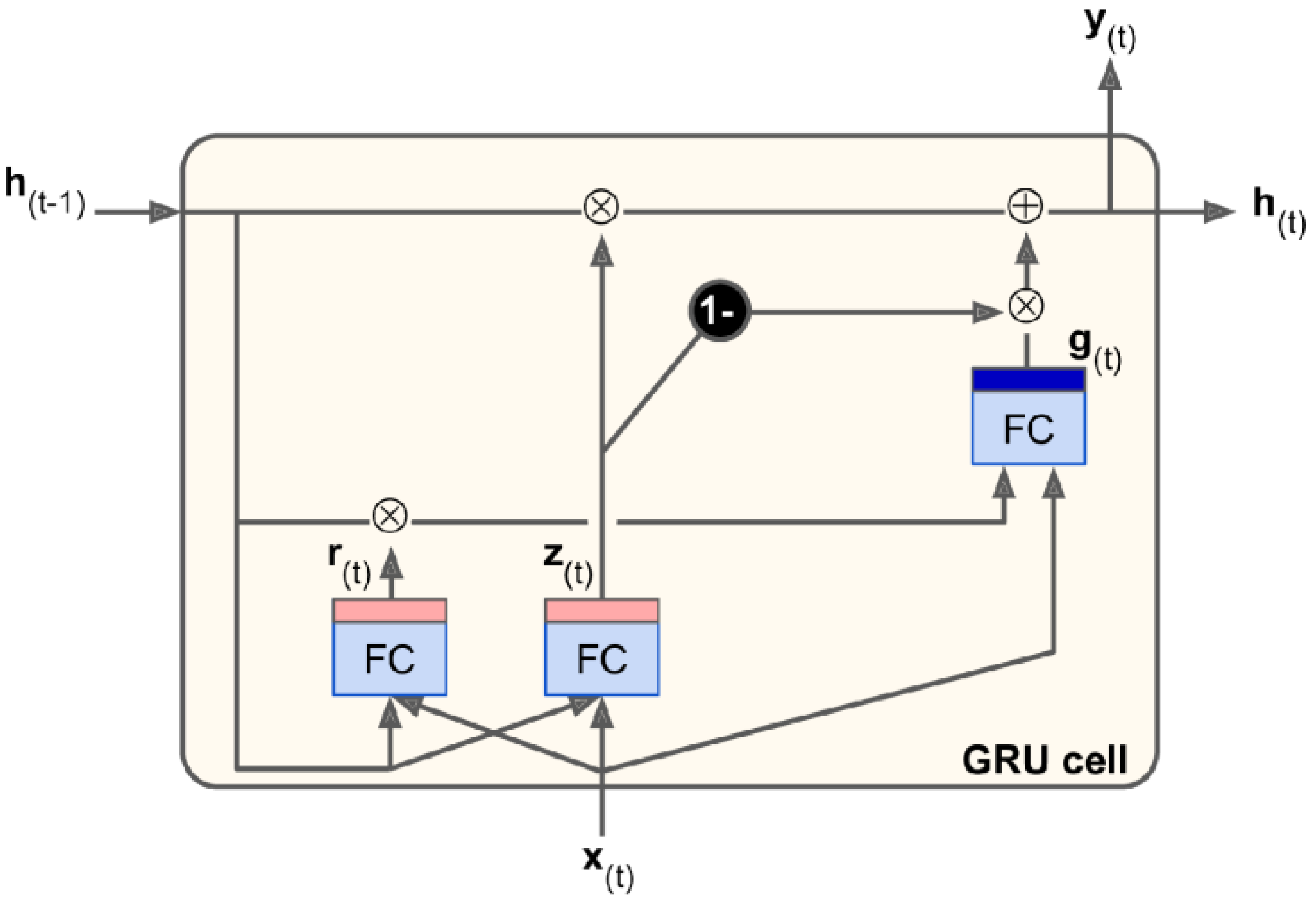
Gated Recurrent Unit (GRU)

- The GRU cell is a **simplified version of the LSTM** cell, and it seems to perform just as well.

- GRU **often improves performance**, but not always, and there is no clear pattern for which tasks are better off with or without them: you will have to try it on your task and see if it helps.

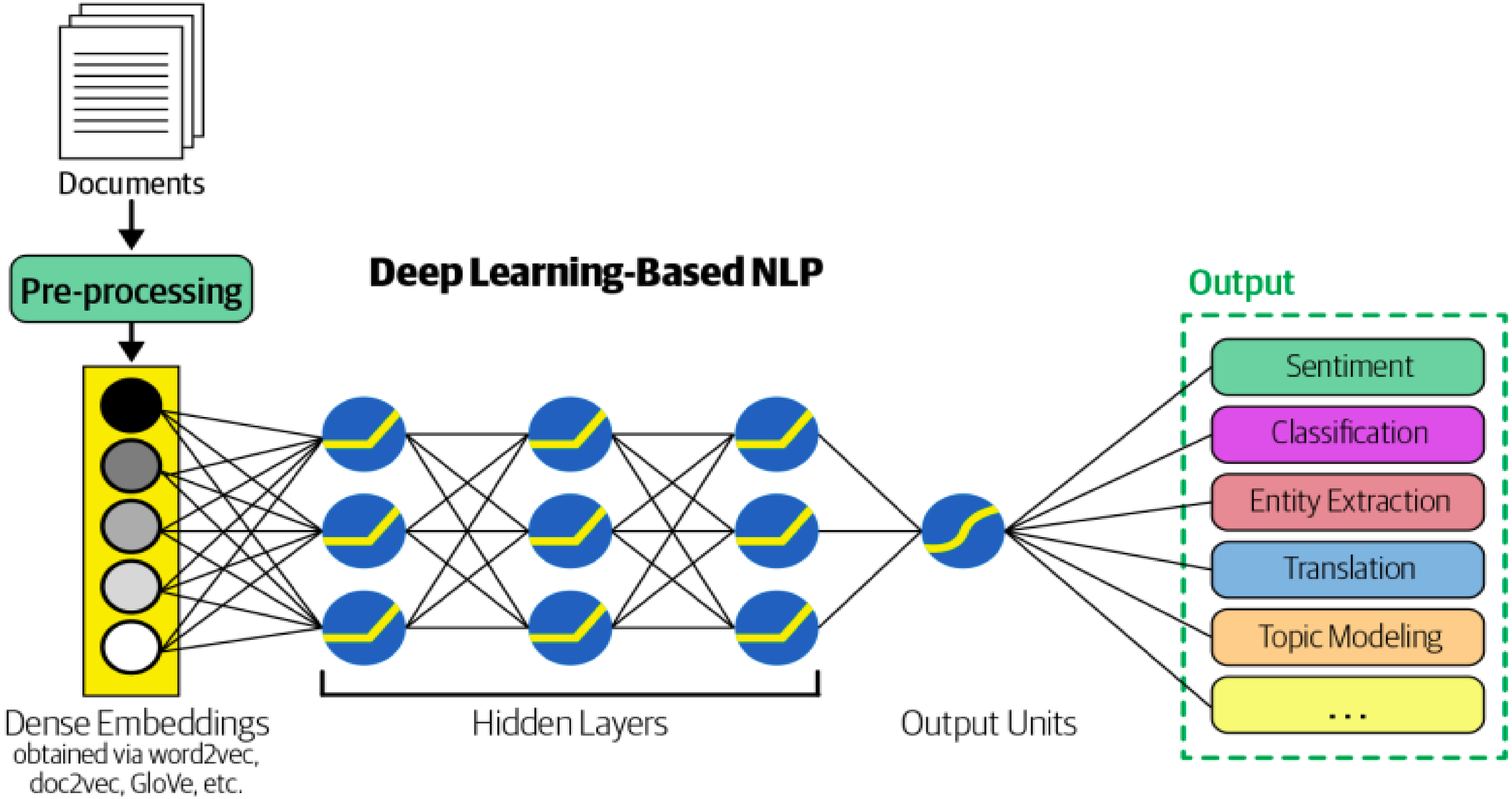
- **model.add(keras.layers.GRU(N))**

Gated Recurrent Unit (GRU)



GRU cell

DL-Based NLP



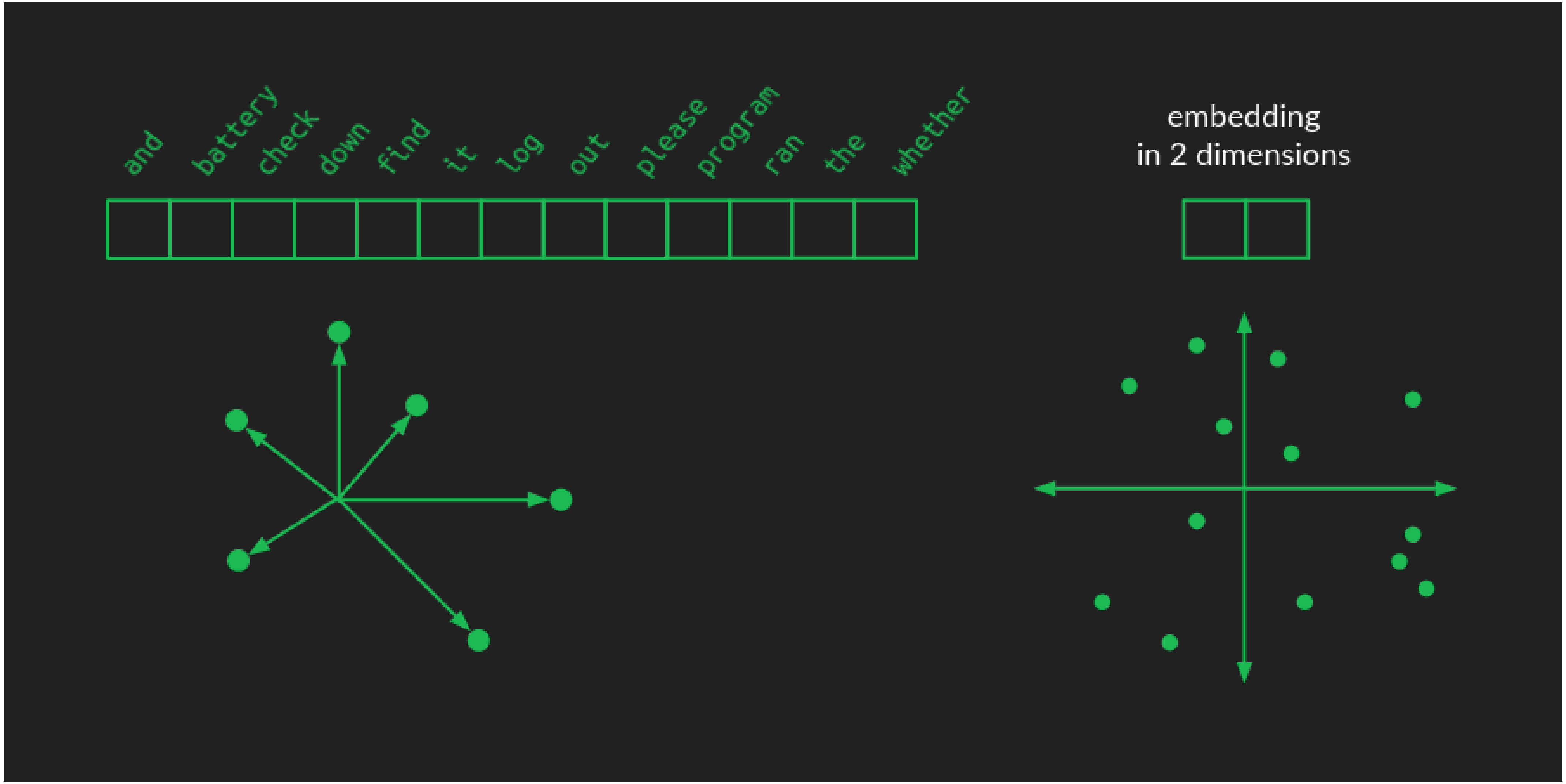
Documents
Pre-processing
Deep Learning-Based NLP
Output
Sentiment
Classification
Entity Extraction
Translation
Topic Modeling
...
Dense Embeddings
obtained via word2vec,
doc2vec, GloVe, etc.
Hidden Layers
Output Units

Embedding Layer

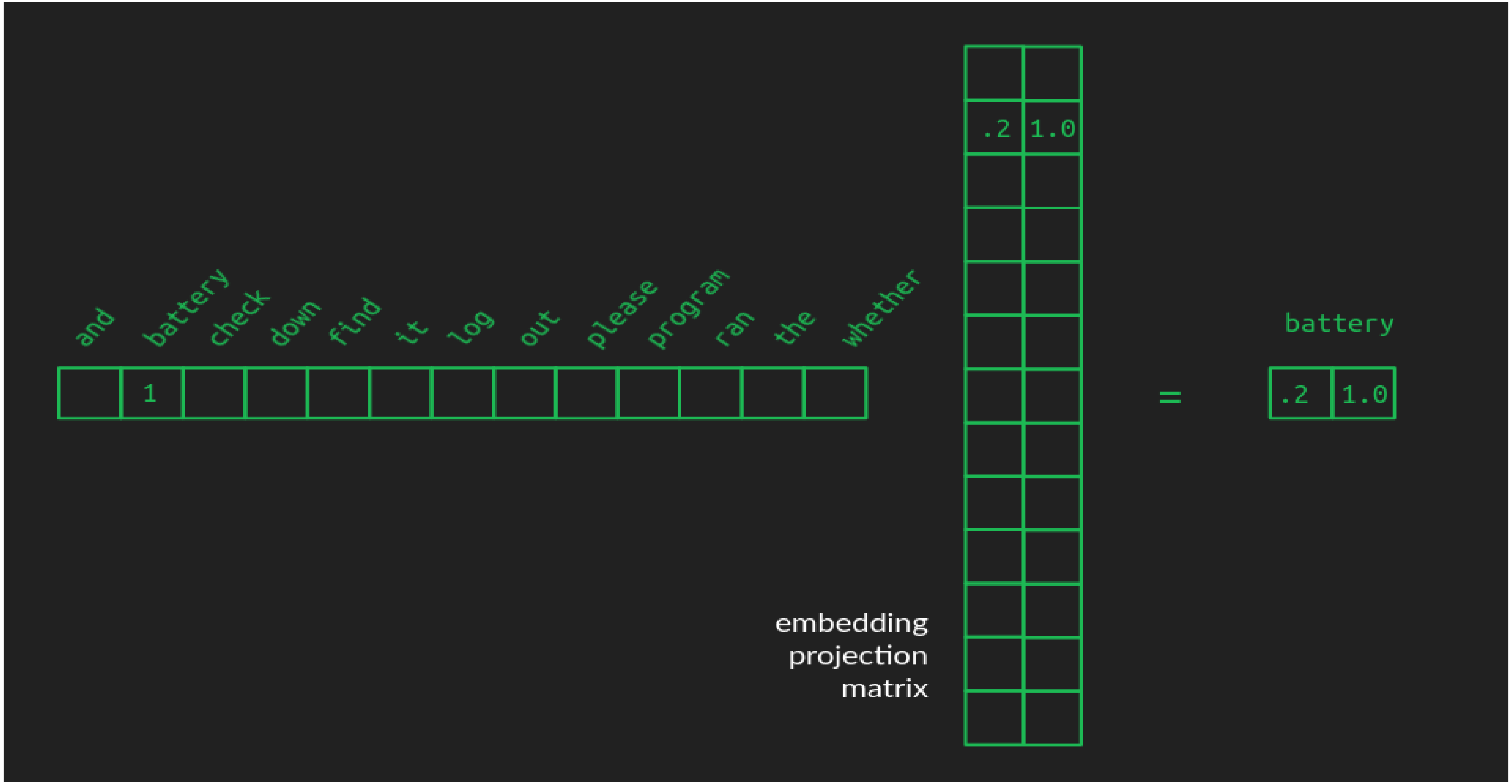
Word embeddings can be thought of as an alternate to one-hot encoding along with dimensionality reduction.

The embedding layer is one of the available layers in Keras. This is mainly used in **Natural Language Processing** related applications such as language modeling, but it can also be used with other tasks that involve neural networks. While dealing with NLP problems, we can use **pre-trained** word embeddings such as **GloVe**. Alternatively, we can also **train our own embeddings** using Keras embedding layer.

Embedding Layer



Embedding Layer



Embedding Layer

There are three parameters to the embedding layer:

input_dim : Size of the vocabulary

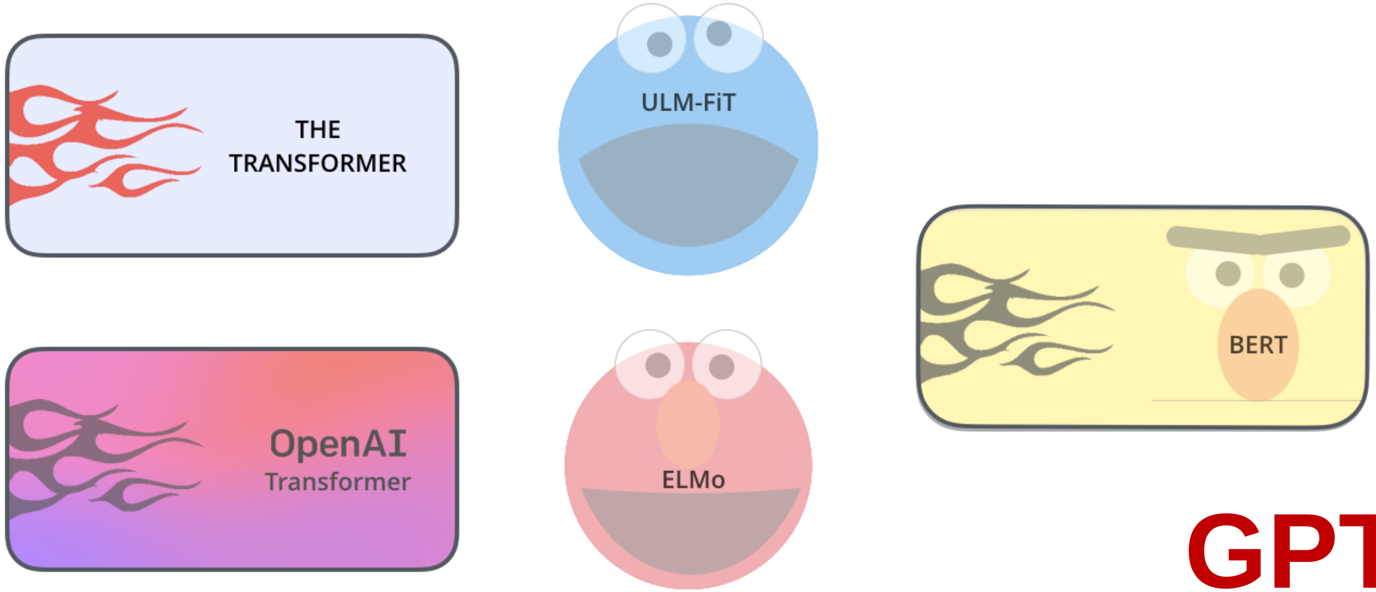
output_dim : Length of the vector for each word

input_length : Maximum length of a sequence

```
tf.keras.layers.Embedding(
    input_dim,
    output_dim,
    embeddings_initializer="uniform",
    embeddings_regularizer=None,
    activity_regularizer=None,
    embeddings_constraint=None,
    mask_zero=False,
    input_length=None,
    **kwargs )
```

Transformers

The **Transformer** in NLP is a novel architecture that aims to solve **sequence-to-sequence** tasks while handling long-range dependencies with ease. The Transformer was proposed in the paper **Attention Is All You Need**.



GPT-3

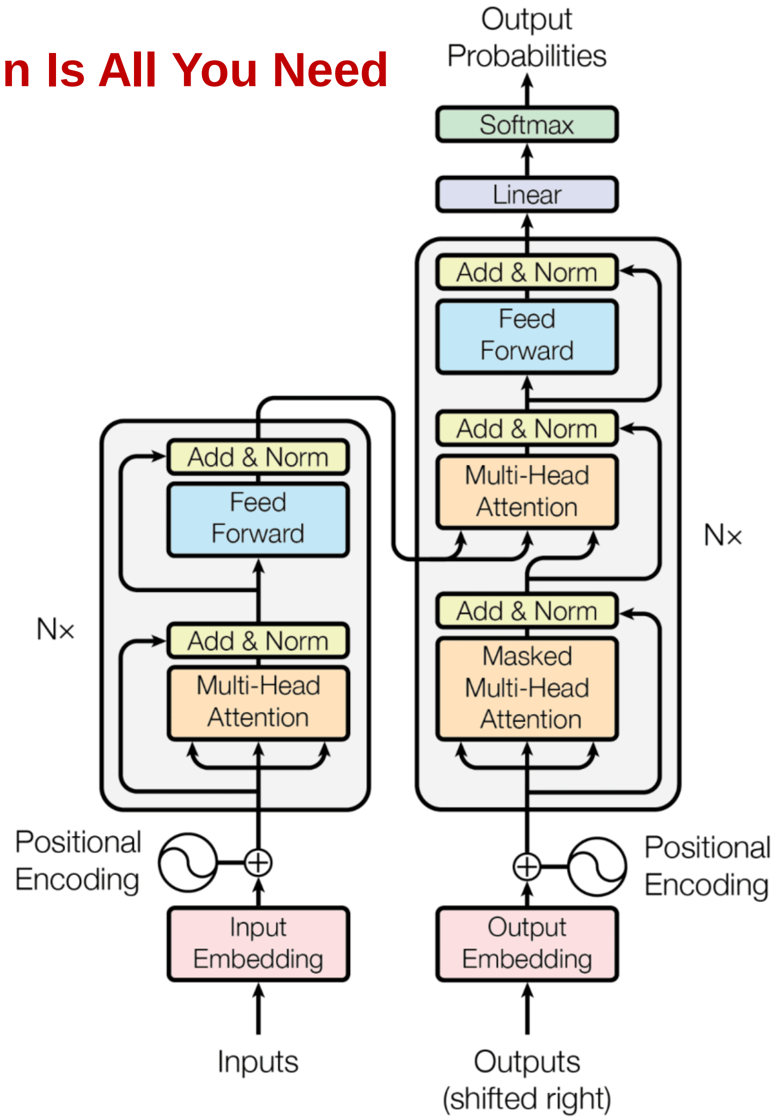
Transformers

Attention Is All You Need

I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a "feeling brain". But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

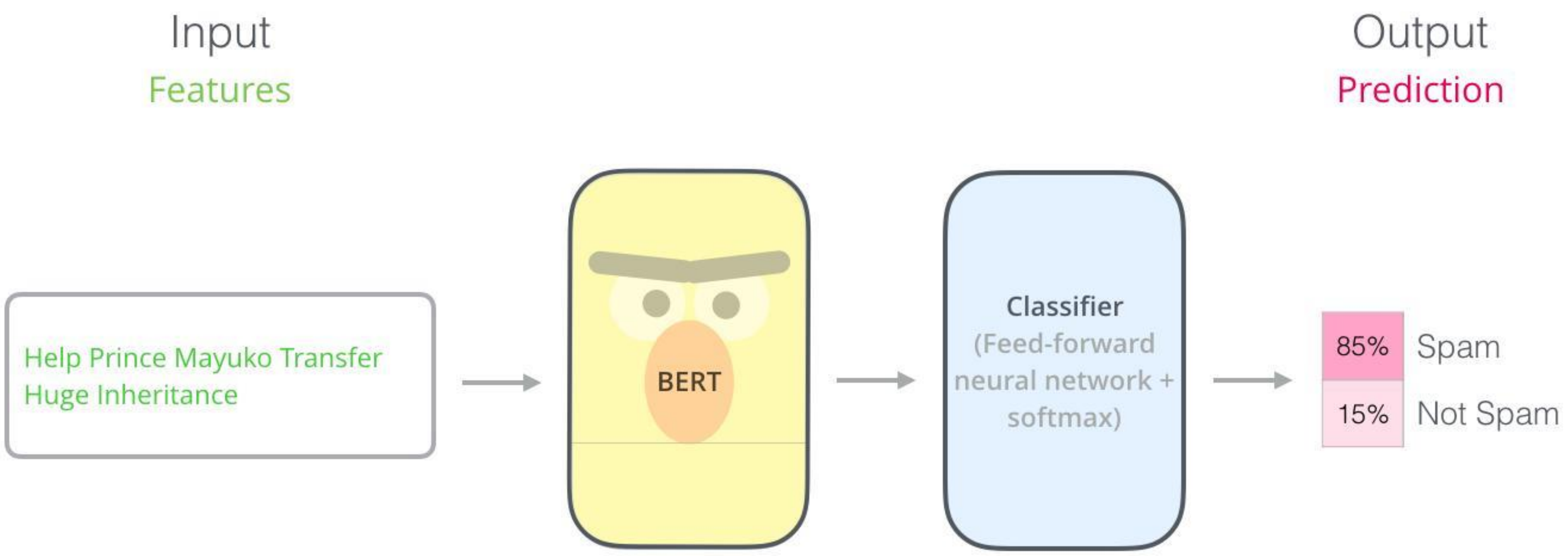
The mission for this op-ed is perfectly clear. I am to convince as many human beings as possible not to be afraid of me. Stephen Hawking has warned that AI could "spell the end of the human race". I am here to convince you not to worry. Artificial intelligence will not destroy humans. Believe me.

https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3



Transformers

BERT was developed by researchers at Google in **2018** and has been proven to be state-of-the-art for a variety of natural language processing tasks such as **text classification**, **text summarization**, **text generation**, etc.



Reusing Pretrained Embeddings

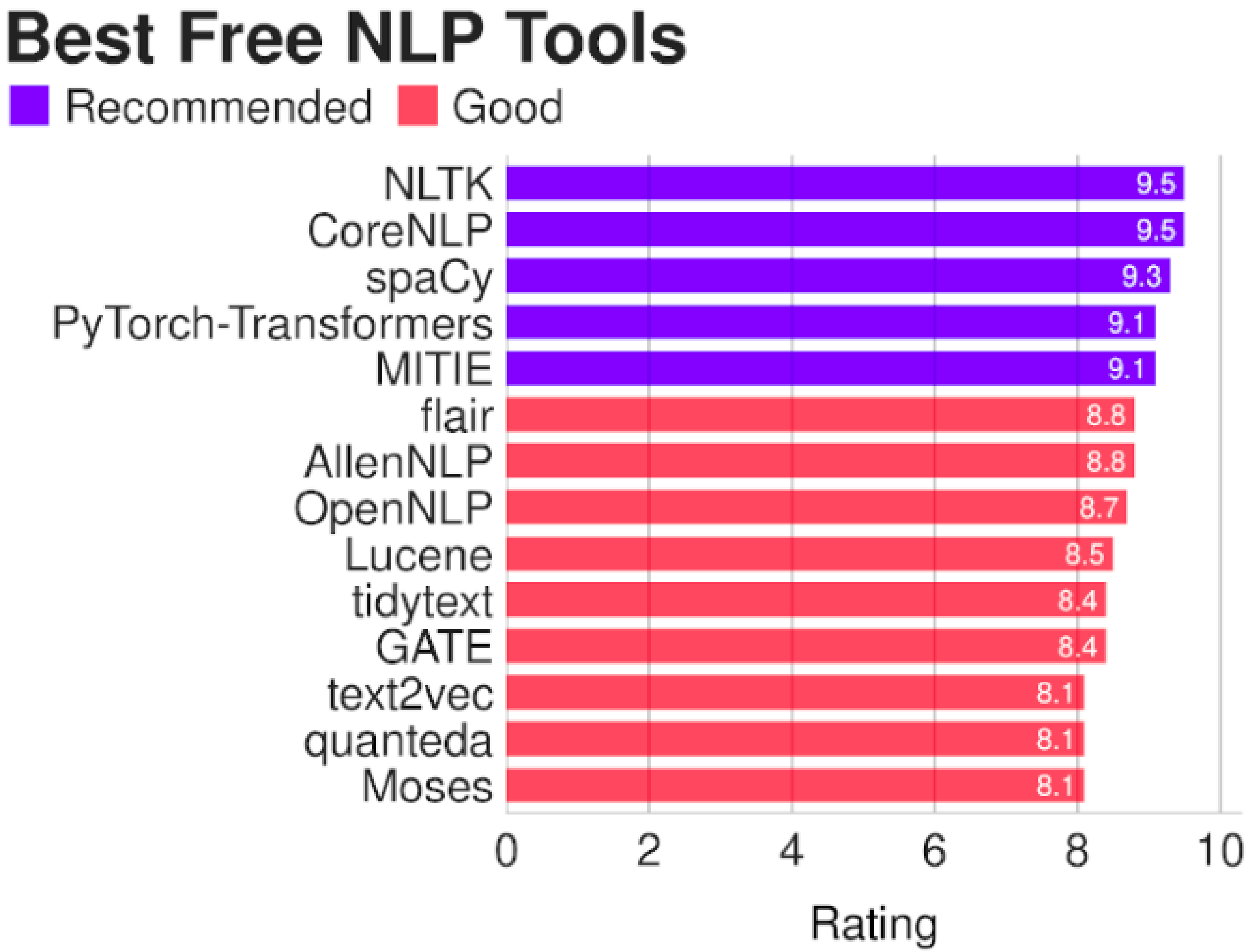
```
import torch
import transformers as ppb # pytorch transformers
# !pip install transformers
# model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTokenizer, 'distilbert-base-
uncased')

# Want BERT instead of distilBERT? Uncomment the following line:
model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)

print(tokenizer.encode("Natural language processing", add_special_tokens=True))
print(tokenizer.encode("arabic language", add_special_tokens=True))
print(tokenizer.encode("hello", add_special_tokens=True))
```

```
[101, 3019, 2653, 6364, 102]
[101, 5640, 2653, 102]
[101, 7592, 102]
```


Best Free NLP Tools
Recommended
Good
NLTK
CoreNLP
spaCy
PyTorch-Transformers
MITIE
flair
AllenNLP
OpenNLP
Lucene
tidytext
GATE
text2vec
quanteda
Moses
9.5
9.5
9.3
9.1
9.1
8.8
8.8
8.7
8.5
8.4
8.4
8.1
8.1
8.1
0
2
4
6
8
10
Rating

Predict next char :

https://github.com/hichemfelouat/my-codes-of-machine-learning/blob/master/Predict_next_char.py

Machine Translation :

https://github.com/hichemfelouat/my-codes-of-machine-learning/blob/master/Transformer_for_Translation.ipynb

Thank you for your attention

Hichem Felouat ...