Natural Language Processing

AIS

Romain Benassi

Course 3

Spring 2022

Course Schedule

- **Course 1**: NLP introduction
- **Course 2**: Word embedding
- **Course 3**: Long Short-Term Memory (LSTM) architecture
- **Course 4**: “Attention” mechanism and Transformer architectures
- **Course 5**: Chatbot implementation

Course Schedule

- **Course 1**: NLP introduction
- **Course 2**: Word embedding
- **Course 3**: **Long Short Term Memory (LSTM) architecture**
  - Recurrent Neural Network (RNN)
  - LSTM principle and architecture
  - Text preprocessing (to feed an LSTM)
  - Use case studies
    - Sentiment analysis
    - Translation
    - Text generation

This Week

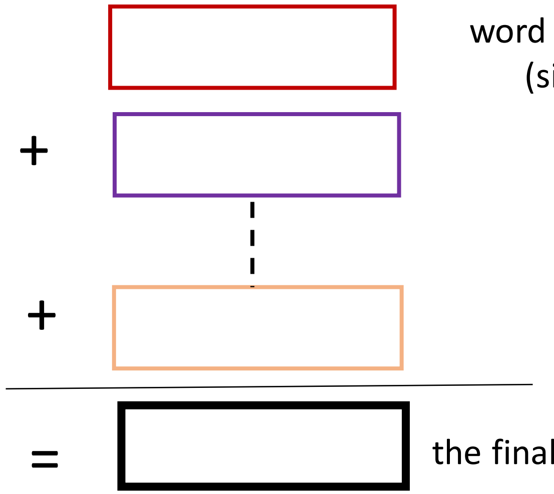
- **Course 4**: “Attention” mechanism and Transformer architectures
- **Course 5**: Chatbot implementation

Previously in Course 2

Embedding: Text embedding

One can get a **text embedding** in **averaging** the embedding vectors of all the tokens in the text

To *be* *or* *not* *to* *be*

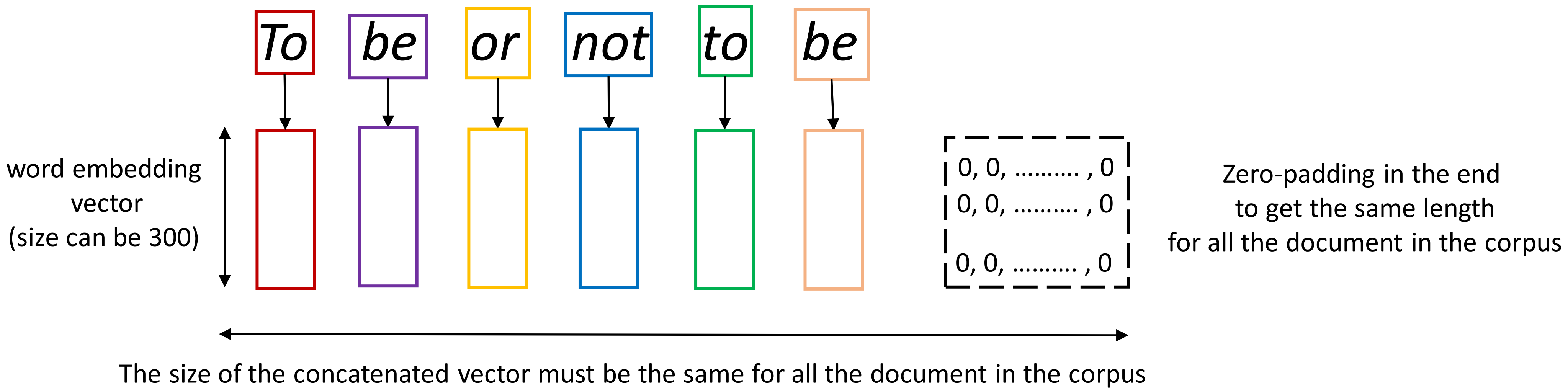


Remarks

- The dimension of the text embedding remains quite **low**
- There is a significant **loss of information** in comparison with the previous method

Embedding: Sequence embedding

One solution is to concatenate the embedding vectors of all the tokens of the text



Remarks

- to keep **the same dimensions** for each text of a corpus, there is a need to **truncate** the number of tokens or to **add padding** (depending on the number of tokens)
- The dimension of each text representation can be very **high** (e.g., several thousands)

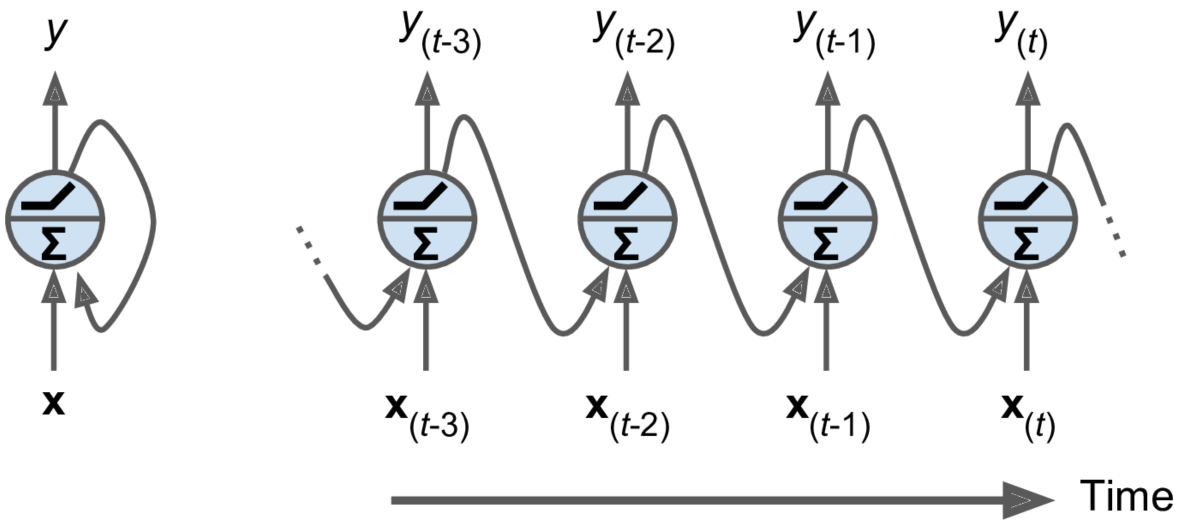
Course 3: Long Short Term Memory (LSTM)

Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN): Definition

- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**

A simple RNN

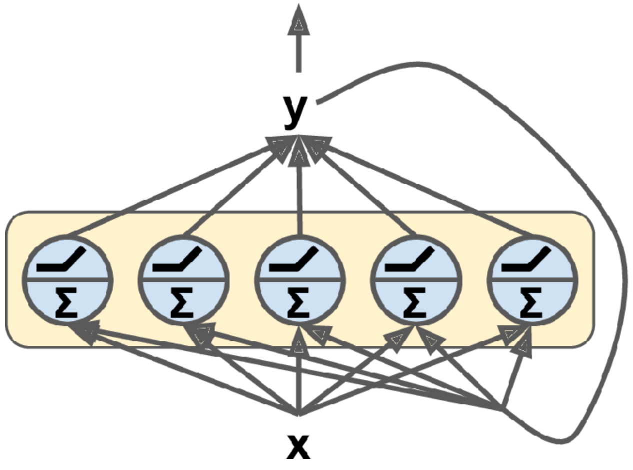


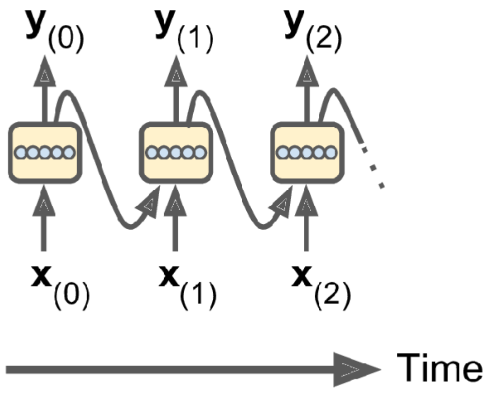
Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Definition

- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**

A multi-neuron RNN



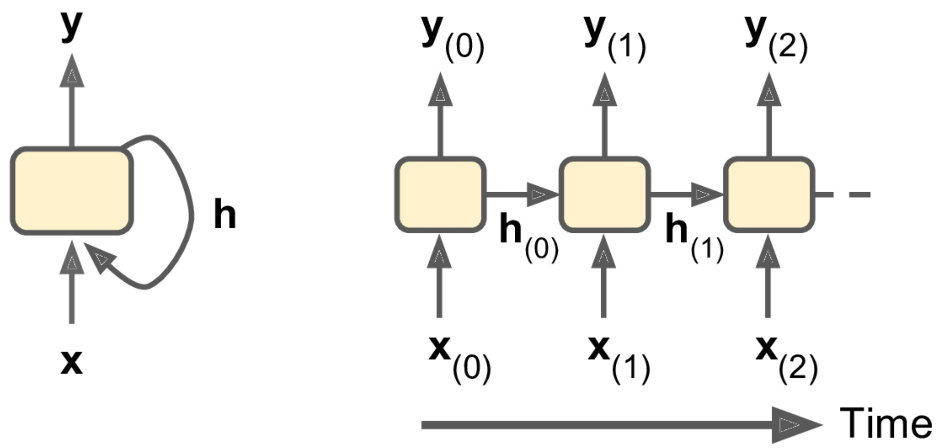


Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Definition

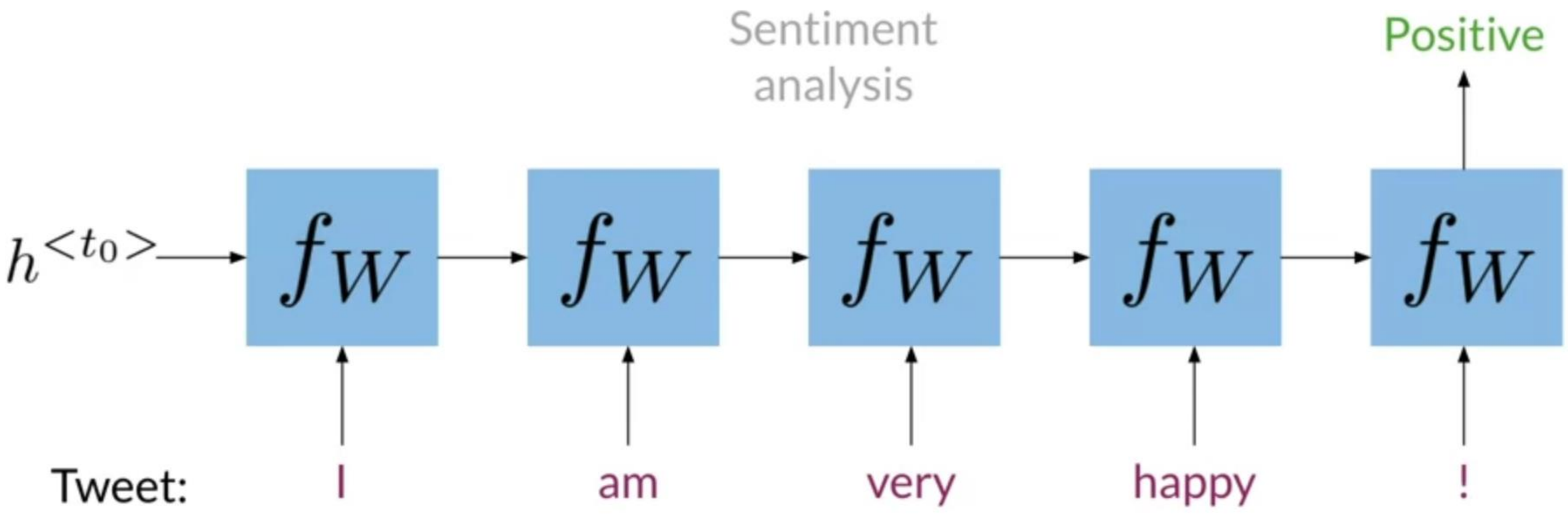
- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**
- **A cell hidden state h and the output y may be different**

A hidden state RNN



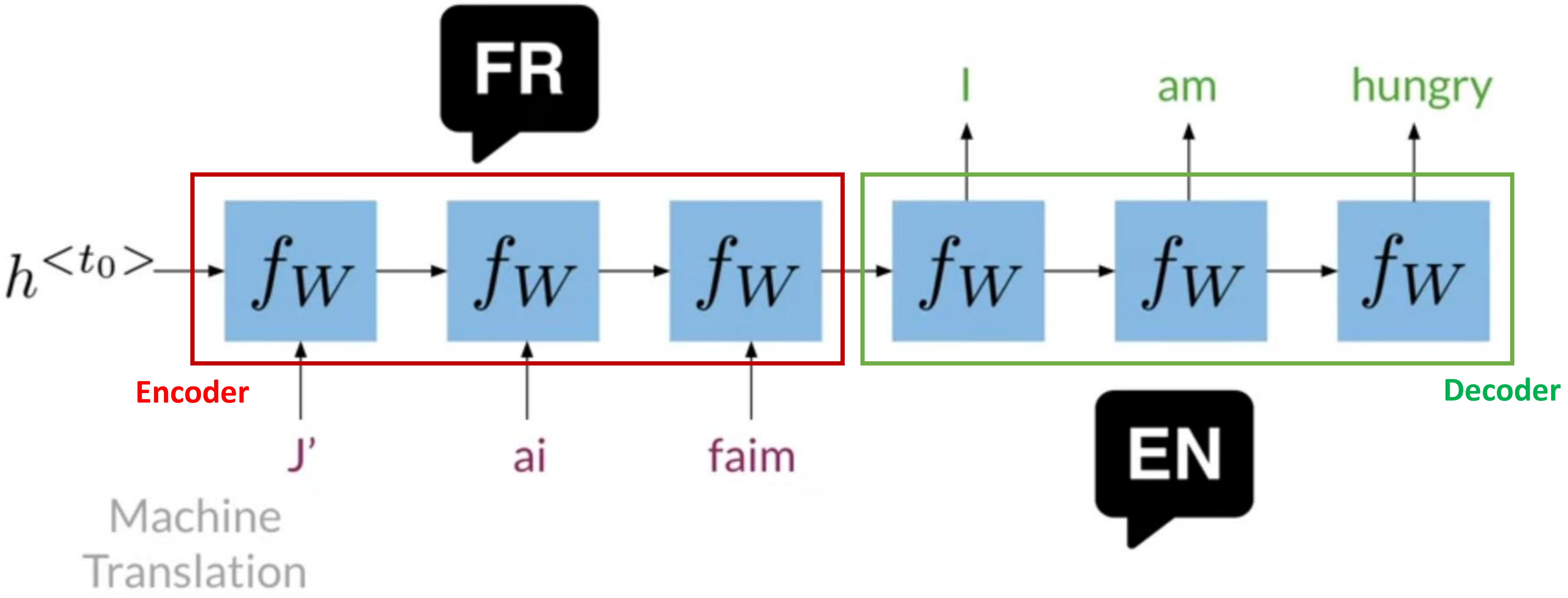
Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

RNN Applications : Many to one



https://www.coursera.org/learn/sequence-models-in-nlp/lecture/VLEGc/applications-of-rnns

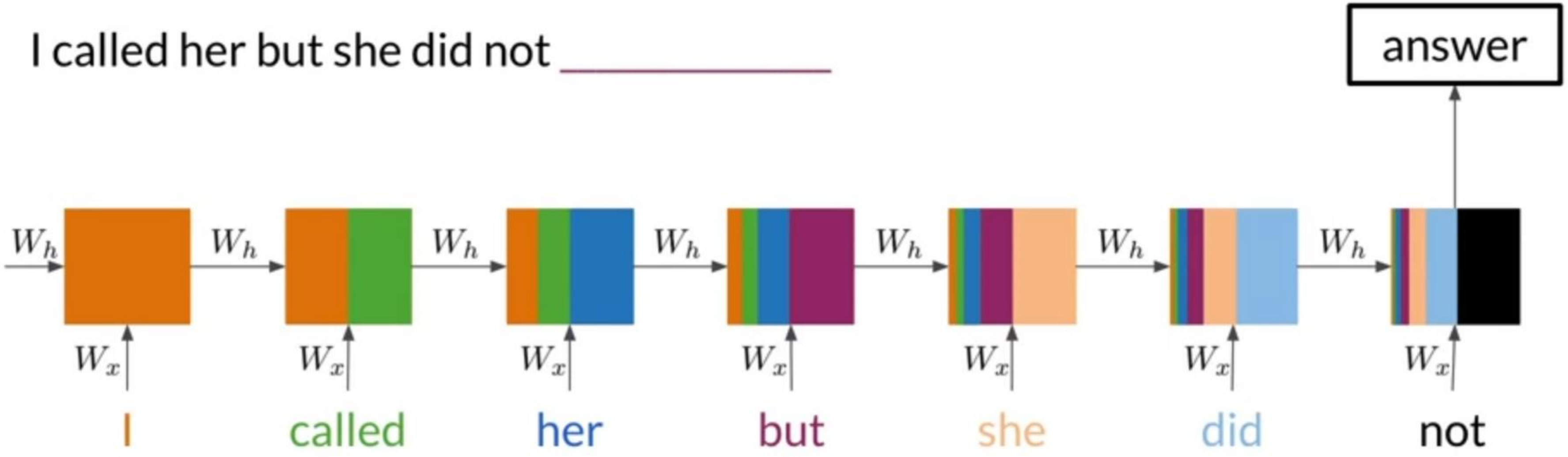
RNN Applications : Many to many



https://www.coursera.org/learn/sequence-models-in-nlp/lecture/VLEGc/applications-of-rnns

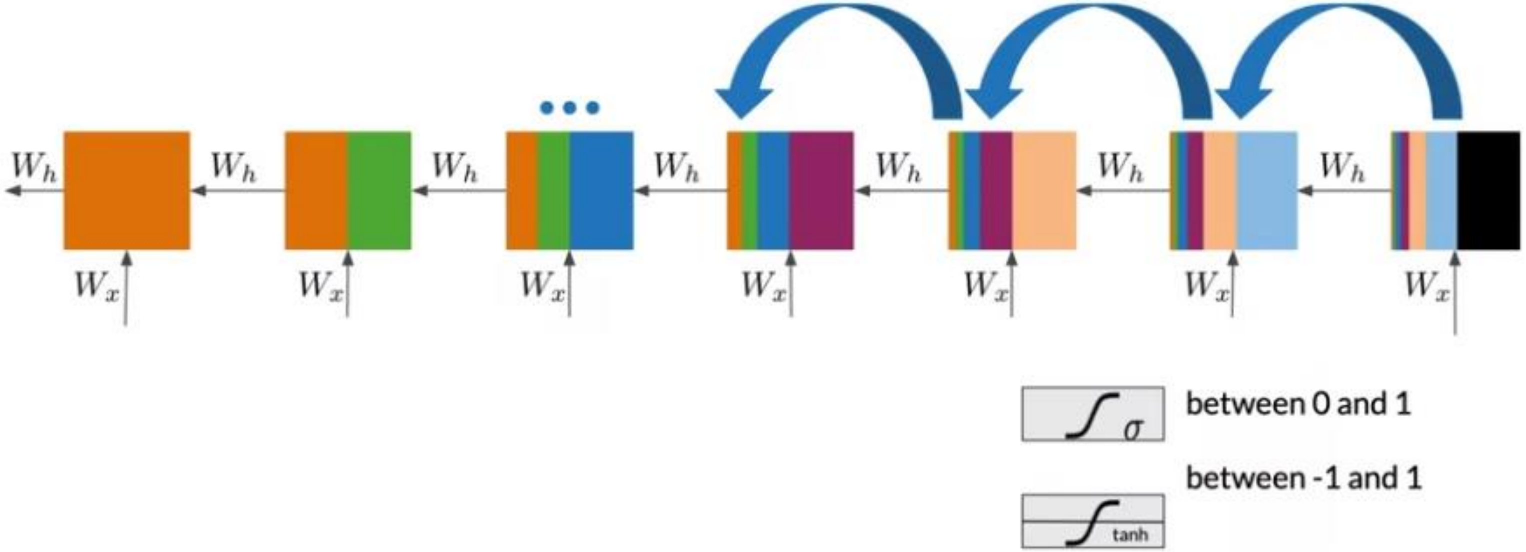
RNN : Long sequence issue

- As shown in the illustration, **RNN** can be used for **text generation**
- However, we can see that, after a **few** cells only, the **impact** of each word almost **disappear**



RNN : Vanishing gradient

- **Vanishing gradient** is another common problem with RNN especially for very long sequences => **makes training non efficient**
- Having a close look at **gradient backpropagation equations** shows that the gradient issues are due to the **derivative of the *tanh*** activation function (<1)



LSTM principle and architecture

LSTM (Long Short Term Memory)

Principle

- The classical deep neural network (e.g., **RNN**) cannot avoid both **vanishing gradient problem** and **loss of early information**

- **LSTM networks** were invented by **Hochreiter** and **Schmidhuber** in 1997

    1. To deal with the **vanishing gradient problem**

    2. To proceed **entire sequences** of data **without forgetting** the meaning of the **early information** proceeded

LSTM (Long Short Term Memory)

Example

I used to live in France, I speak French fluently

The word ***French*** can describe both a language or a nationality. Here the **context** help us to know it corresponds to the language.

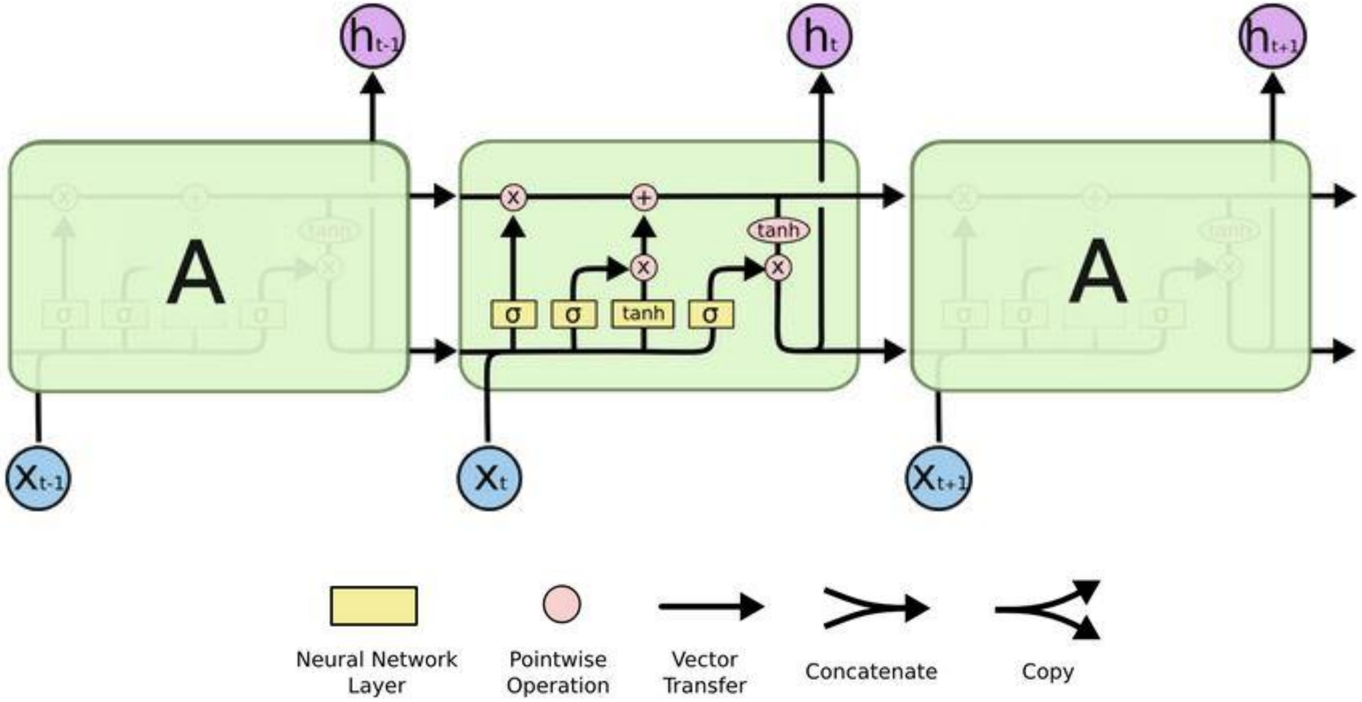
LSTM aims to make good use of the context to determine the meaning of the analyzed words

LSTM (Long Short Term Memory)

A **LSTM architecture** is composed of a set of cells, each of those containing **three gates**

- **An input gate**: decides what to add
- **An output gate**: decides what the next hidden state will be
- **A forget gate**: decides what to keep

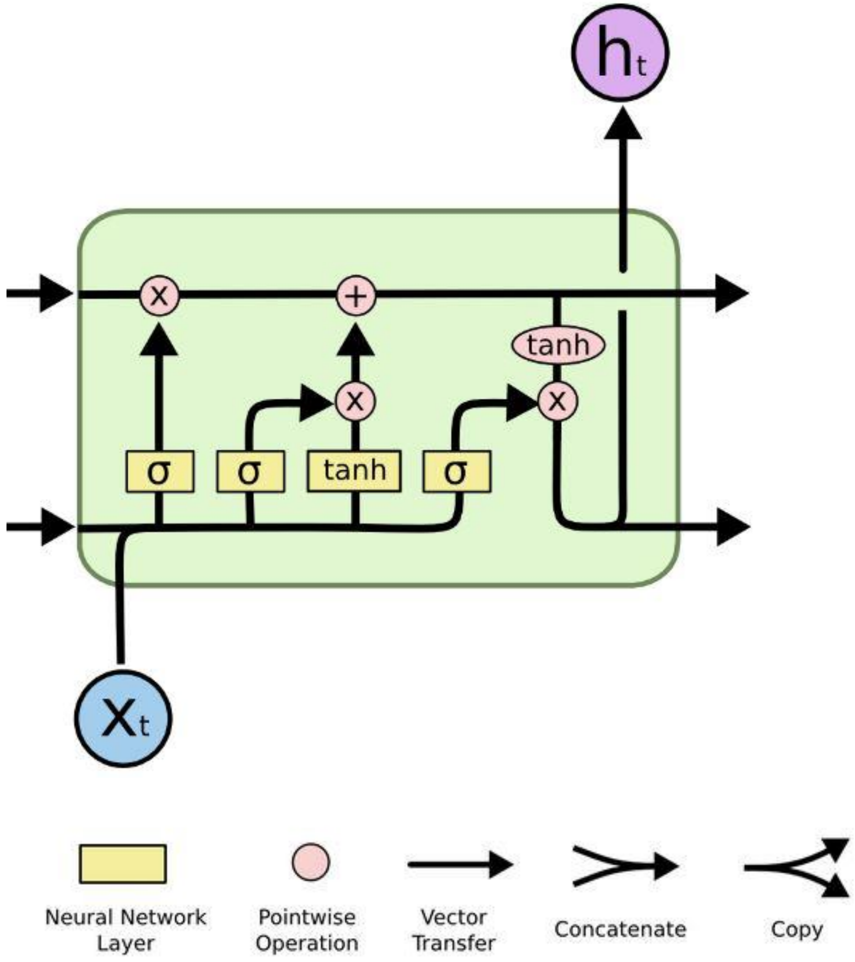
Repeated LSTM modules
(each of them containing four layers)



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

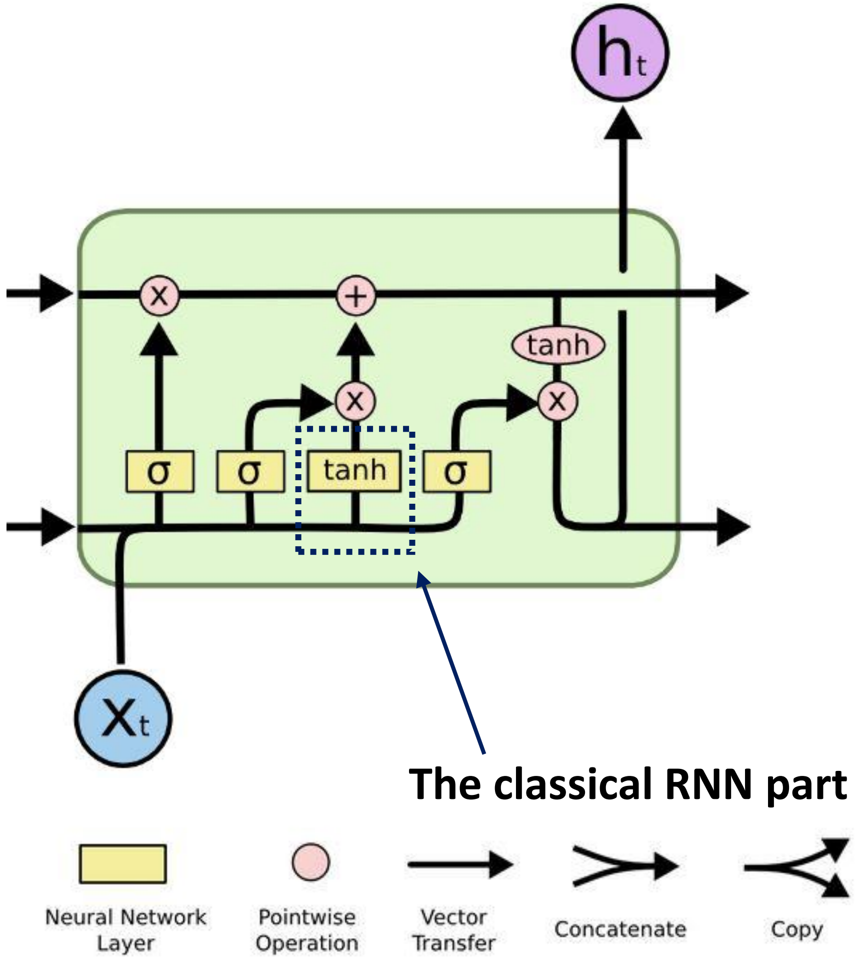
LSTM module



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

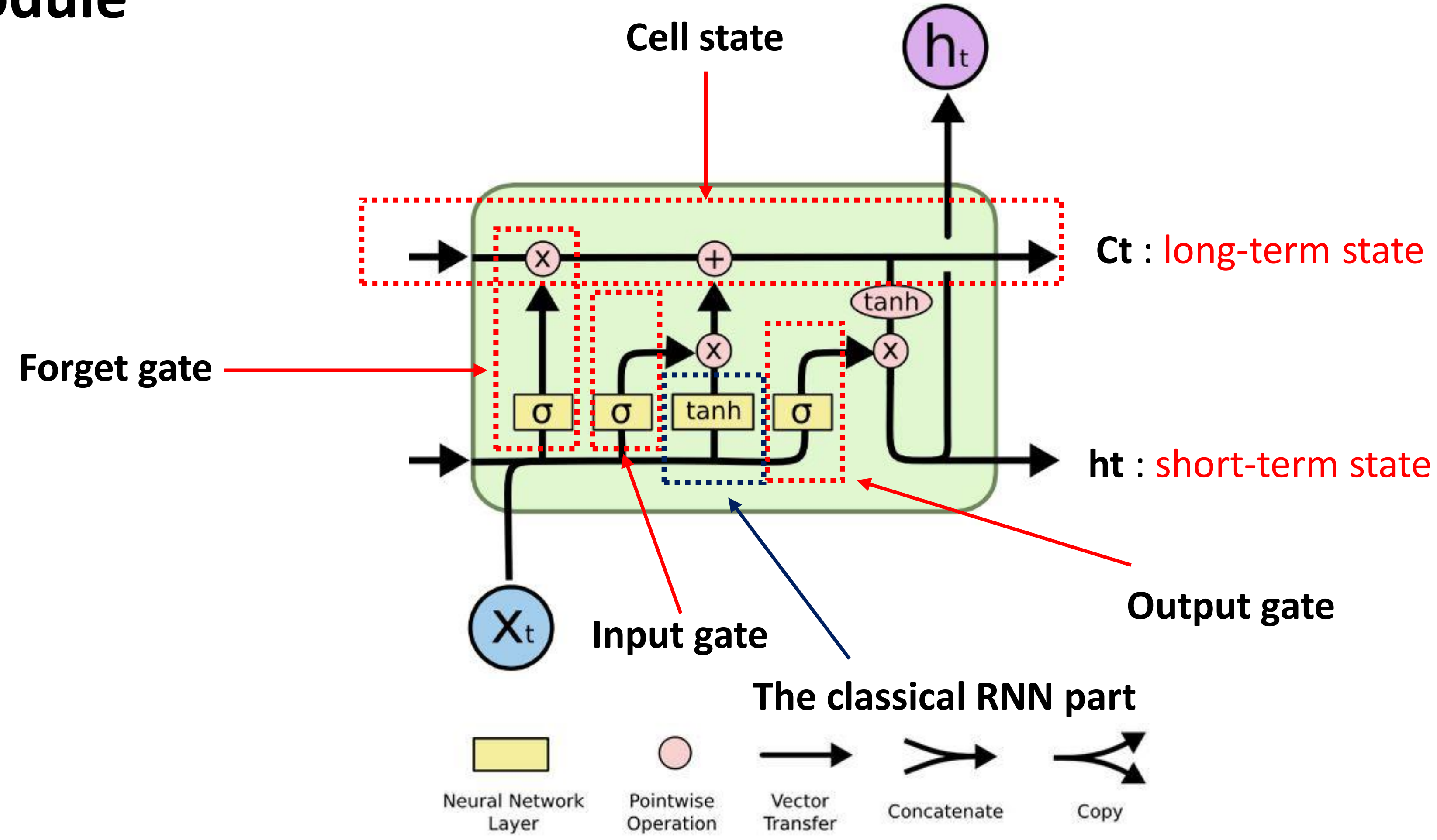
LSTM module



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

LSTM module

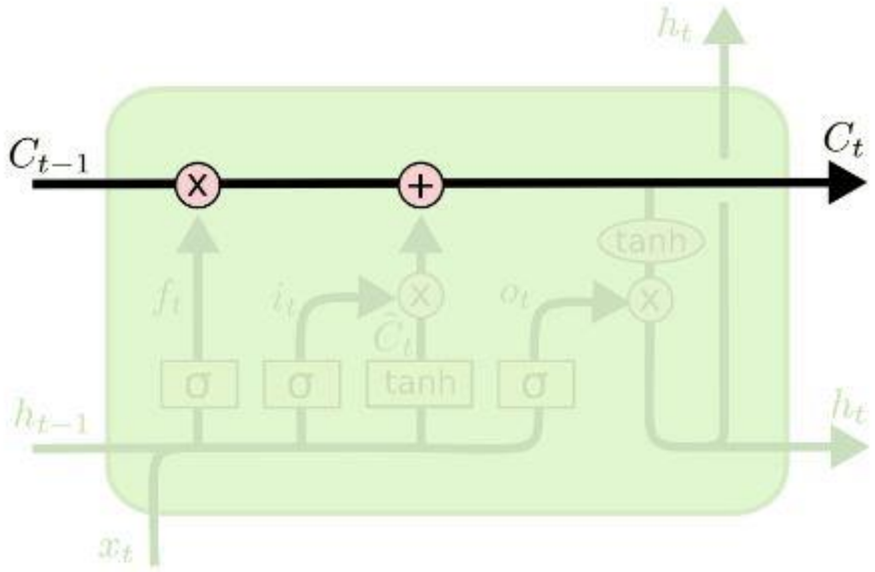


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

Cell state: C_t

The cell state runs down the entire chain with some linear interactions through each LSTM module met down the way



Gates allow LSTM to optionally remove or add information to the cell state

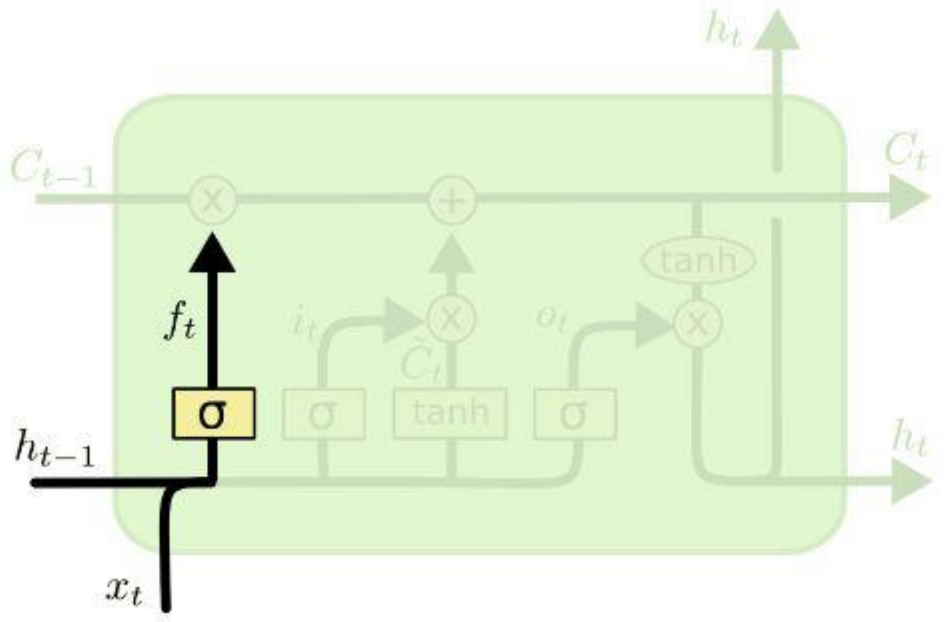
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

Forget gate: decide what information we drop from the cell state

The sigmoid output a number between 0 and 1 for each number in the cell state C_{t-1}

Example: the cell state might include the gender of a present subject.
If a new subject is seen => forget the previous one



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

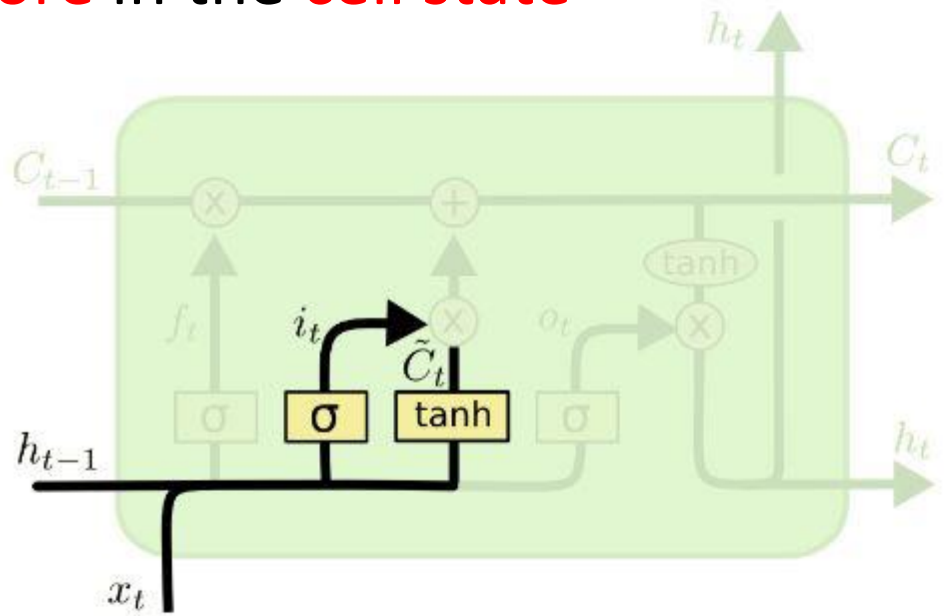
LSTM (Long Short Term Memory)

Input gate: decide what new information we store in the cell state

Two parts:

- **A sigmoid layer to decide which values we update**
- **A tanh layer to create a vector of new candidate values for C_t**

Example: we add the gender of the new subject to replace the previous one which has been forgotten



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

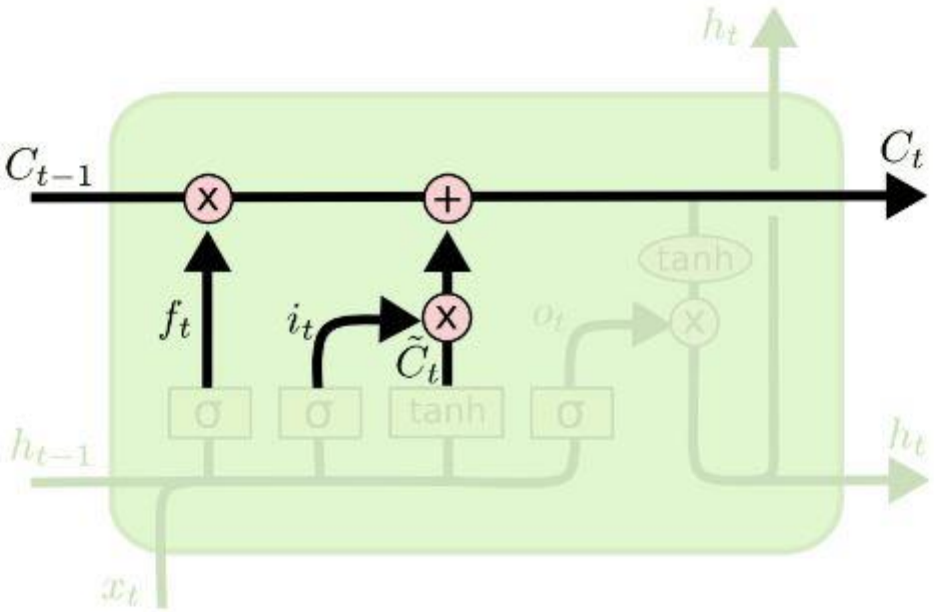
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM (Long Short Term Memory)

Cell state update: $C_{t-1} \Rightarrow C_t$

- **The previous state is multiplied by *ft*, to forget the things we decided earlier**
- **We add the new candidate values, scaled by how much we decided to update earlier**

Example: we drop the information about the old gender and add the information related to the new one



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

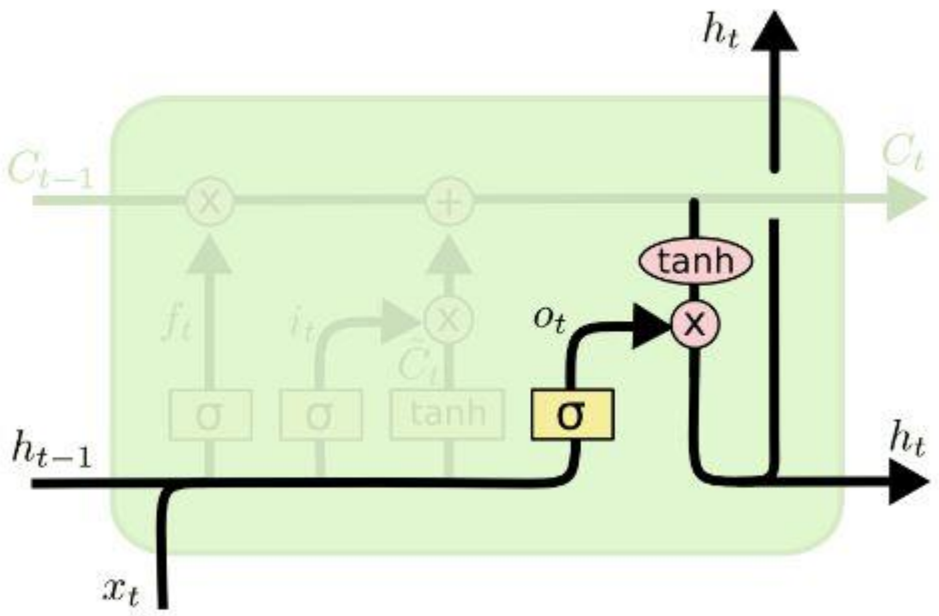
LSTM (Long Short Term Memory)

Output gate: decide what information to output

The output is a filtered version of the cell state

- **First, we run a sigmoid layer to decide what part of the cell state to output**
- **Then, we rescale the cell state (using a tanh layer) and multiply it by the result of the output gate**

Example: it might output whether the subject is singular or plural, in order to know how to conjugate the following verb



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

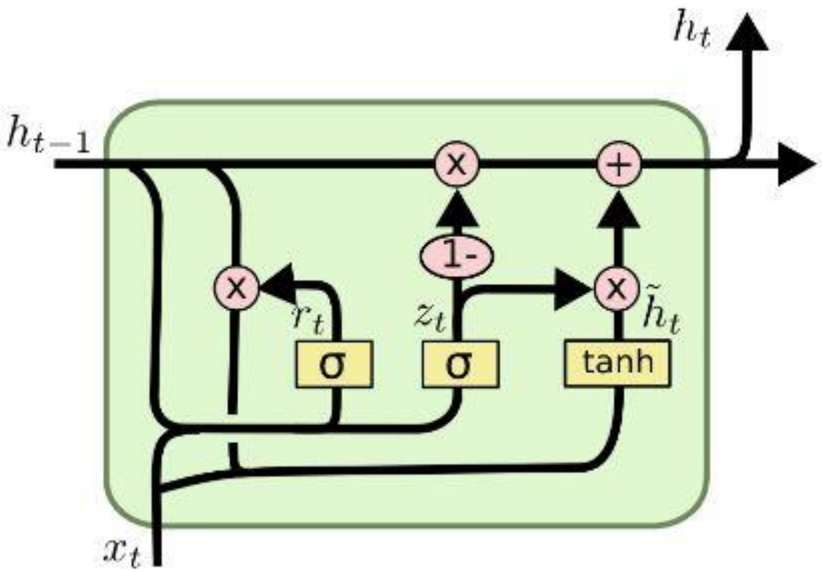
$$h_t = o_t * \tanh\left(C_t\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

Introduced in 2014, it corresponds to a simplified LSTM variant with only two gates

- The **forget** and **input** gates are combined into a single **update** gate which defines how much information to keep
- A **reset** gate is defined to determine how much information to forget
- Both LSTM and GRU can be useful depending on the specific application



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

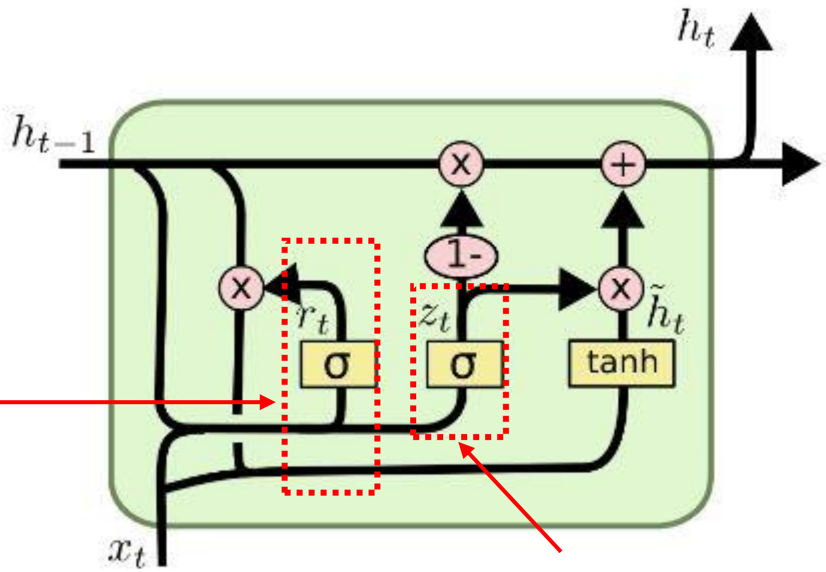
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

Introduced in 2014, it corresponds to a simplified LSTM variant with only two gates

- The **forget** and **input** gates are combined into a single **update** gate which defines how much information to keep
- A **reset** gate is defined to determine how much information to forget
- Both LSTM and GRU can be useful depending on the specific application

Reset gate



Update gate

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

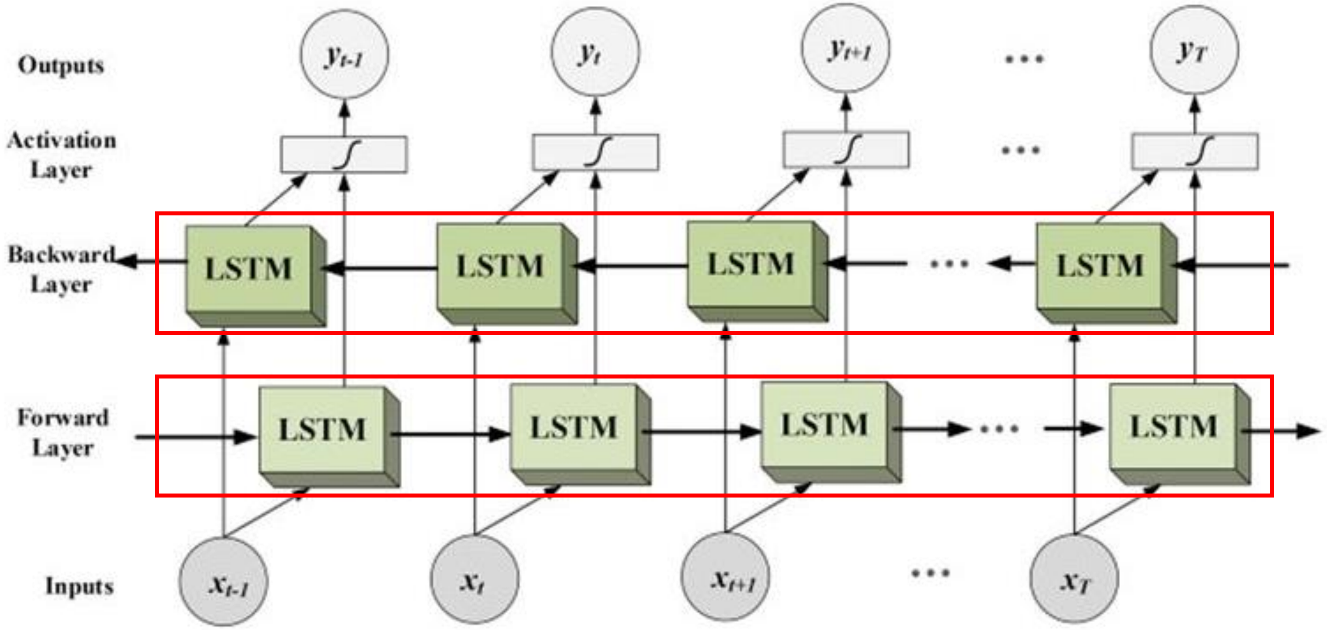
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional LSTM

In a **classic LSTM** architecture, the sentence/sequence is processed only in **one direction** (generally from the past to the present or future)

The **bidirectional** structure process the data in **both directions** => can be useful for some applications



https://medium.com/the-official-integrate-ai-blog/what-you-need-to-know-about-natural-language-processing-2c8240e6c38e

The **Queen** of the United Kingdom

The **queen** of hearts

The **queen** of bees

Here, the word queen will be encode differently
if read in reverse order

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

LSTM (Long Short Term Memory)

Applications

- **Text analysis**
- **Text translation**
- **Next-word prediction**
- Chatbots
- Music composition
- Image captioning
- Speech recognition

Text preprocessing (to feed an LSTM)

Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM** Network with **chunks of text**:

1. Each chunk of text must be split into sequences => **Tokenization**
2. Each sequence must have the same length => **Padding**
3. Each element in a sequence should be converted into numerical values => **Embedding**

Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM** Network with **chunks of text**:

1. **Each chunk of text must be split into sequences => Tokenization**
2. Each sequence must have the same length => **Padding**
3. Each element in a sequence should be converted into numerical values => **Embedding**

Tokenization

Principle

To be or not to be

Tokenization

Principle

To *be* *or* *not* *to* *be* **Tokenization**

Tokenization

Principle

To be or not to be

[3, 21, 36, 15, 3, 21]

Tokenization

Each token is associated to its **number** in the **vocabulary** considered

Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM** Network with **chunks of text**:

1. Each chunk of text must be split into sequences => **Tokenization**
2. **Each sequence must have the same length => Padding**
3. Each element in a sequence should be converted into numerical values => **Embedding**

Padding

Principle

To *be* *or* *not* *to* *be*

Tokenization

[3, 21, 36, 15, 3, 21]

Padding

[3, 21, 36, 15, 3, 21, 0, 0, 0, 0]

We add **additional zeros** in the **end** in order to consider a **fixed length** for every sequence

Padding (Variant)

Principle

To *be* *or* *not* *to* *be*

Tokenization

[3, 21, 36, 15, 3, 21]

Padding

[0, 0, 0, 0, 3, 21, 36, 15, 3, 21]

We can as well add the **additional zeros** in the **beginning** instead

The choice may depend on the application

Text preprocessing

The **LSTM architecture** was conceived to process **long sequences** without any **long-term dependency issues**

We train an **LSTM** Network with **chunks of text**:

1. Each chunk of text must be split into sequences => **Tokenization**
2. Each sequence must have the same length => **Padding**
3. **Each element in a sequence should be converted into numerical values => Embedding**

Embedding

Once we get the padded sequence

To *be* *or* *not* *to* *be*

Tokenization and padding

[3, 21, 36, 15, 3, 21, 0, 0, 0, 0]

two possibilities concerning the next step

1. Use of a pre-trained embedding (see **Course 2**)
2. **Learn a new embedding from scratch (use of an Embedding layer in Keras)**

Embedding

As we saw during last **Course**:

*Word embeddings come from neural network training on **HUGE** datasets: need to use **pre-trained libraries** for general use cases*

BUT in some cases, we are not interested by a model with a good representation of a language **in general** but only for a **specific** application.

In that case, and if the training data set is **big enough**, we can learn from scratch a new embedding for **the specific task** we are interested in.

(Another possibility is to **fine-tune** a pre-trained model – let's see that later in **Course 4**!)

Embedding: One-hot encoding step

To feed an embedding training network, we must use a relevant format for the input

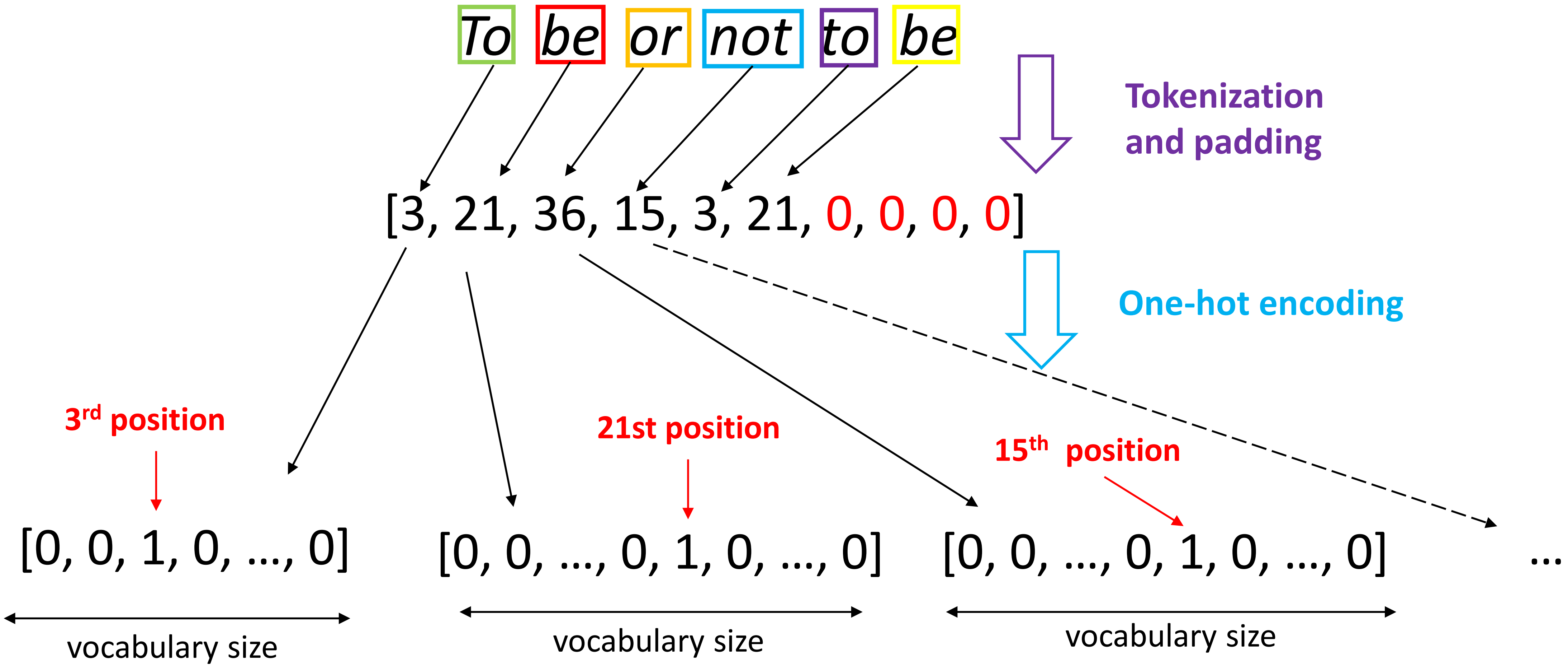
Normally, we cannot use the token sequence directly as inputs of the neural network because the numerical values are misleading

[3, 21, 36, 15, 3, 21, 0, 0, 0, 0]

Indeed, all tokens are **independent**, the network should **NOT** consider that tokens “2” and “3” are closer than, let’s say, tokens “2” and “3000”

Because of that, a **more neutral** numerical representation is **needed** => **One-hot encoding**

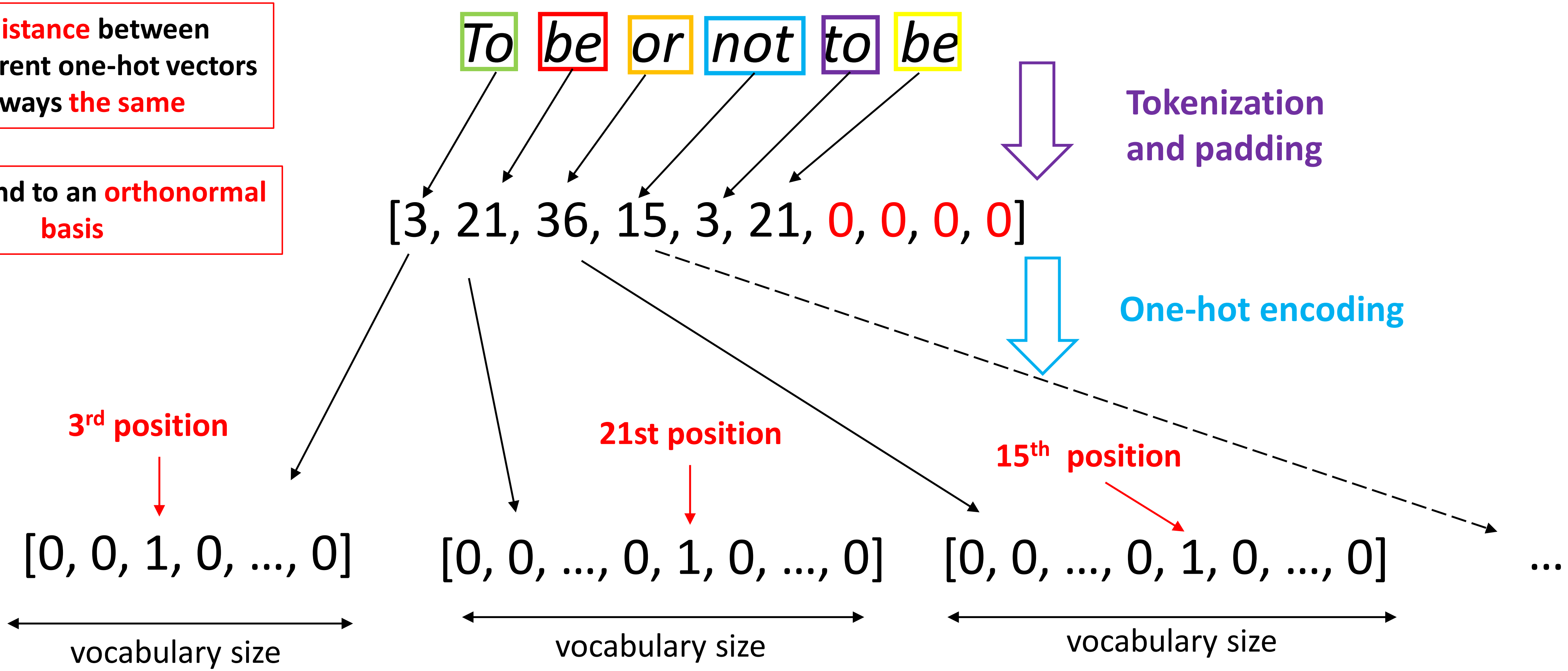
Embedding: One-hot encoding step



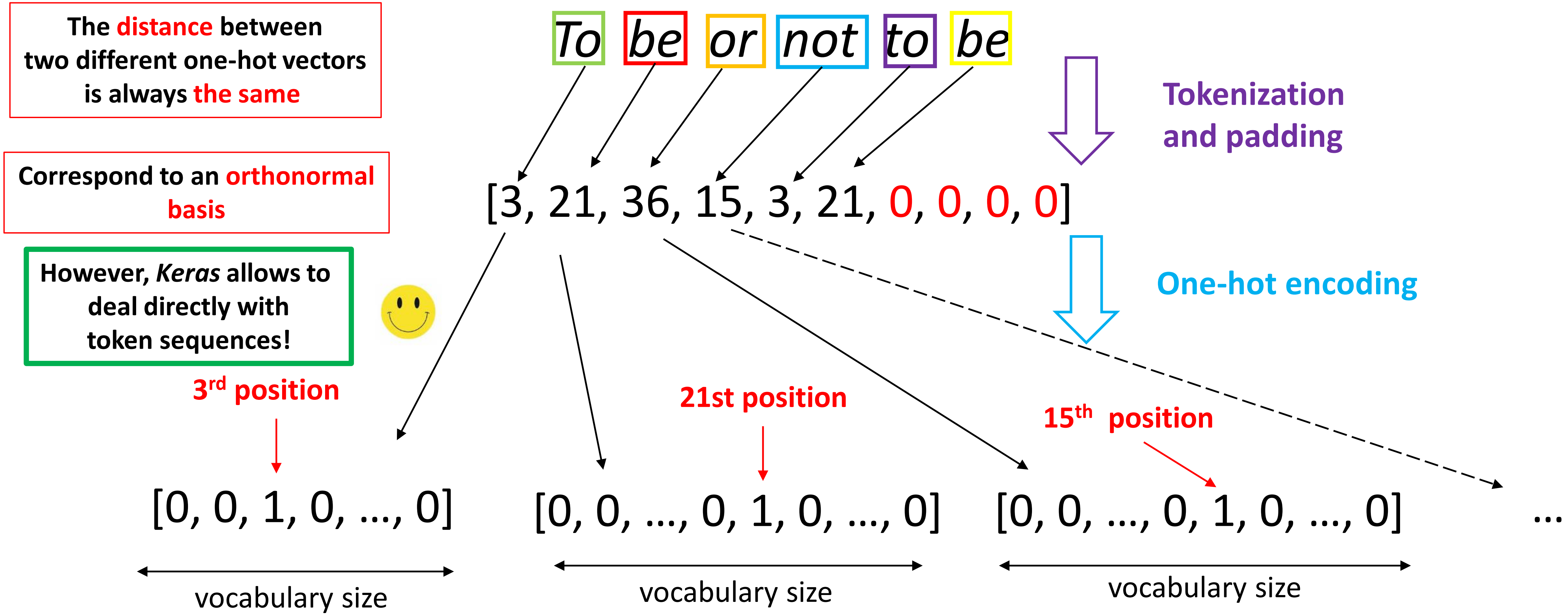
Embedding: One-hot encoding step

The distance between two different one-hot vectors is always the same

Correspond to an orthonormal basis



Embedding: One-hot encoding step



Embedding: *Embedding* Keras layer

In order to **train an embedding from scratch** (using **directly** the **token** sequences), one can use the ***Embedding* Keras layer**

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

Embedding: *Embedding* Keras layer

In order to **train an embedding from scratch** (using **directly** the **token** sequences), one can use the ***Embedding* Keras layer**

Size of the wanted embedding
=> The choice here is up to the user

Fixed length of the padded sequences

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

Number of tokens in the vocabulary

Text sequence preprocessing: Exercise

course3_text_sequence_prepocessing_ex.ipynb

Goal: Get used to the basics of text sequence preprocessing before feeding an RNN/LSTM network **IMDB dataset**

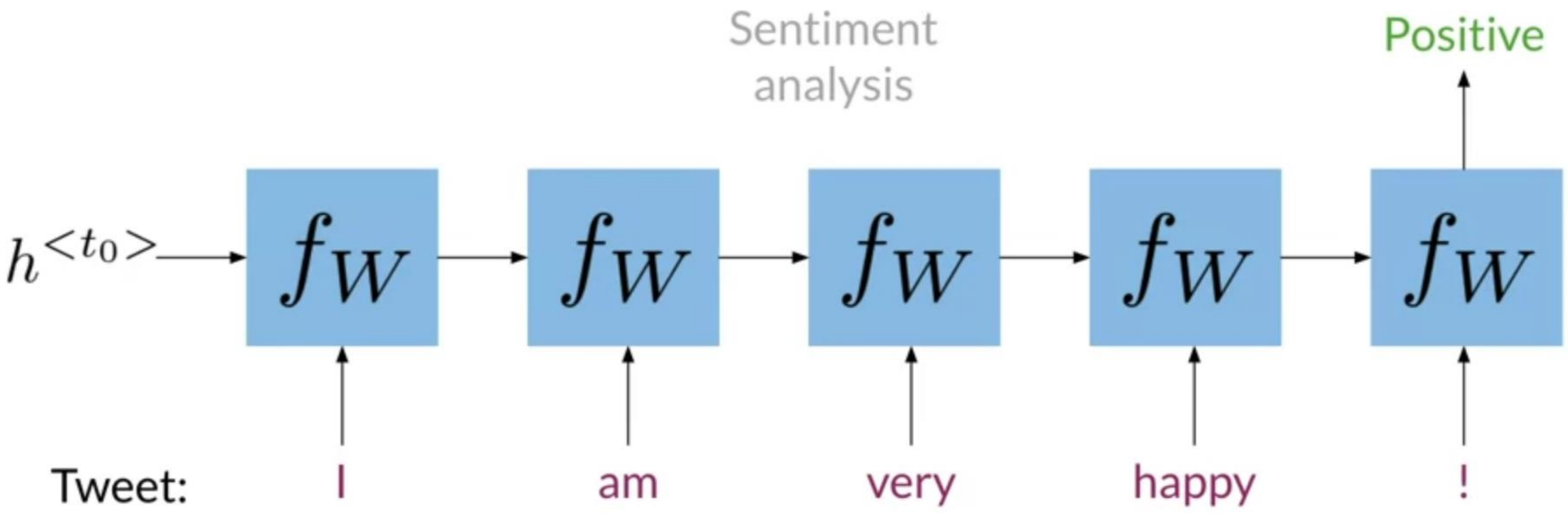
Remarks:

- We will see how to compute **tokenization**, **padding** and **one-hot encoding** from the **Keras** API
- We use (again) **IMDB reviews dataset** for this exercise

Use cases

Use cases: Sentiment analysis

Sentiment analysis: Illustration



https://www.coursera.org/learn/sequence-models-in-nlp/lecture/VLEGc/applications-of-rnns

Sentiment analysis: Exercise

course3_sentiment_analysis_LSTM_ex.ipynb

Goal: Use of **text sequences** to feed neural networks such as **LSTM** for a **sentiment analysis** use case

Remarks:

- The **main difference** with last **Course** sentiment analysis exercises is that we do not use **an average text embedding** as input any longer, but the **concatenated token embedding vector sequences**
- Another difference is that we do not use a pre-trained embedding model but **learn it from the dataset itself**

Sentiment analysis : LSTM model

Use of a Bidirectional LSTM model

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

Sentiment analysis : GRU model

Use of a Bidirectional GRU model

```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

Sentiment analysis : LSTM vs GRU

LSTM

```
# Model Definition with LSTM
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(LSTM(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 120, 16)           160000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                12544
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 172,941
Trainable params: 172,941
Non-trainable params: 0
_________________________________________________________________
```

172, 941

GRU

```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
bidirectional (Bidirectional (None, 64)                9600
_________________________________________________________________
dense (Dense)                (None, 6)                 390
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 169,997
Trainable params: 169,997
Non-trainable params: 0
```

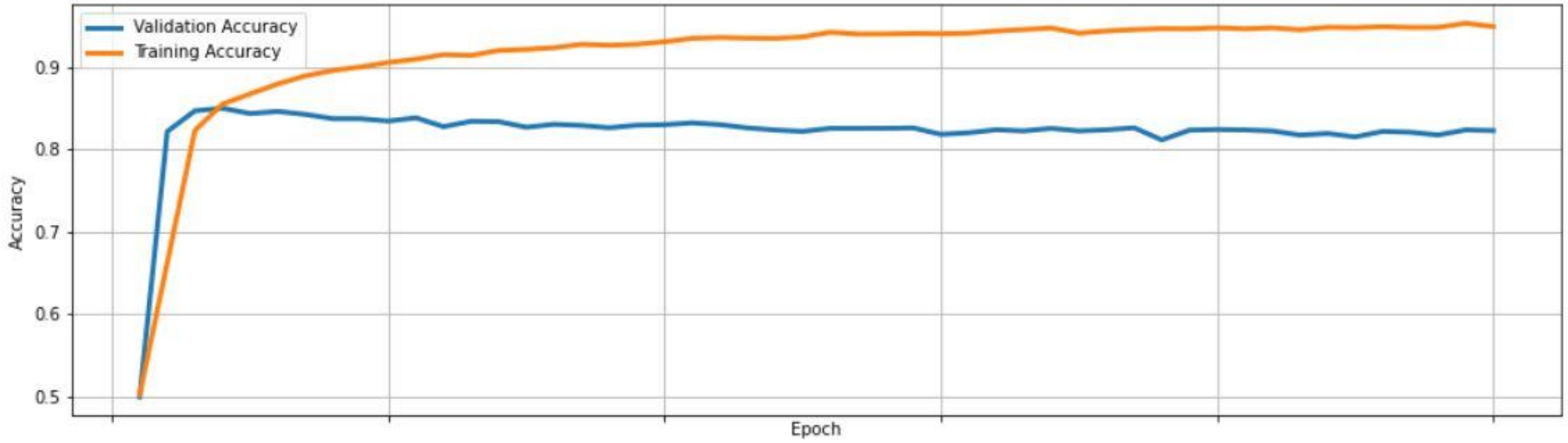
169, 997

A (little) less trainable parameters for the GRU model

Sentiment analysis : Results

- **Using a GRU model (with Dropout)**
- **50 epochs**

```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Dropout(0.5),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

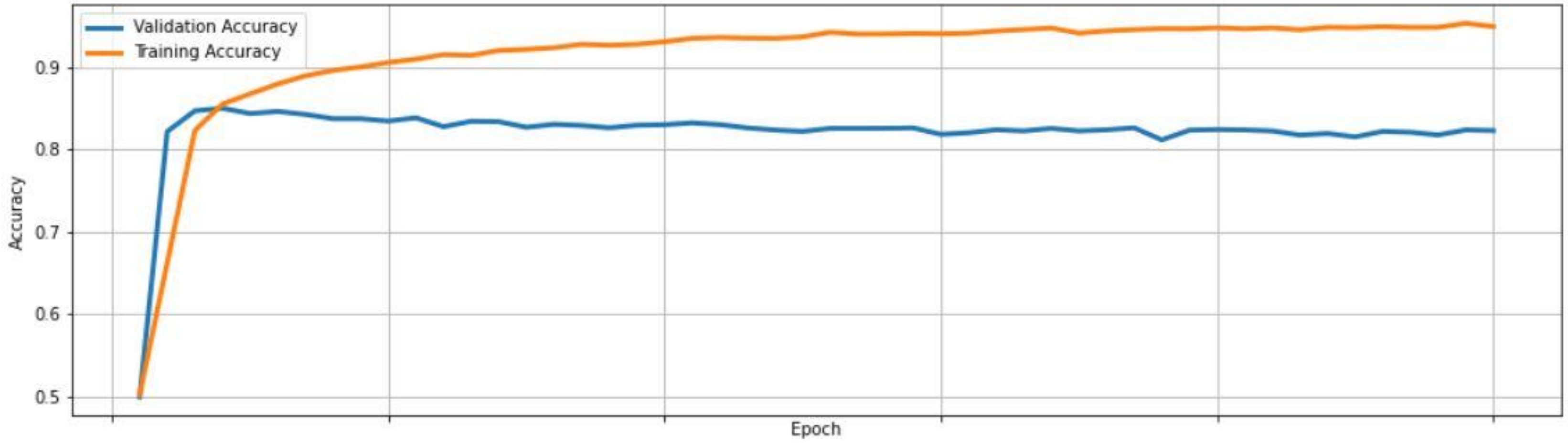


Sentiment analysis : Results

- **Using a GRU model (with Dropout)**
- **50 epochs**

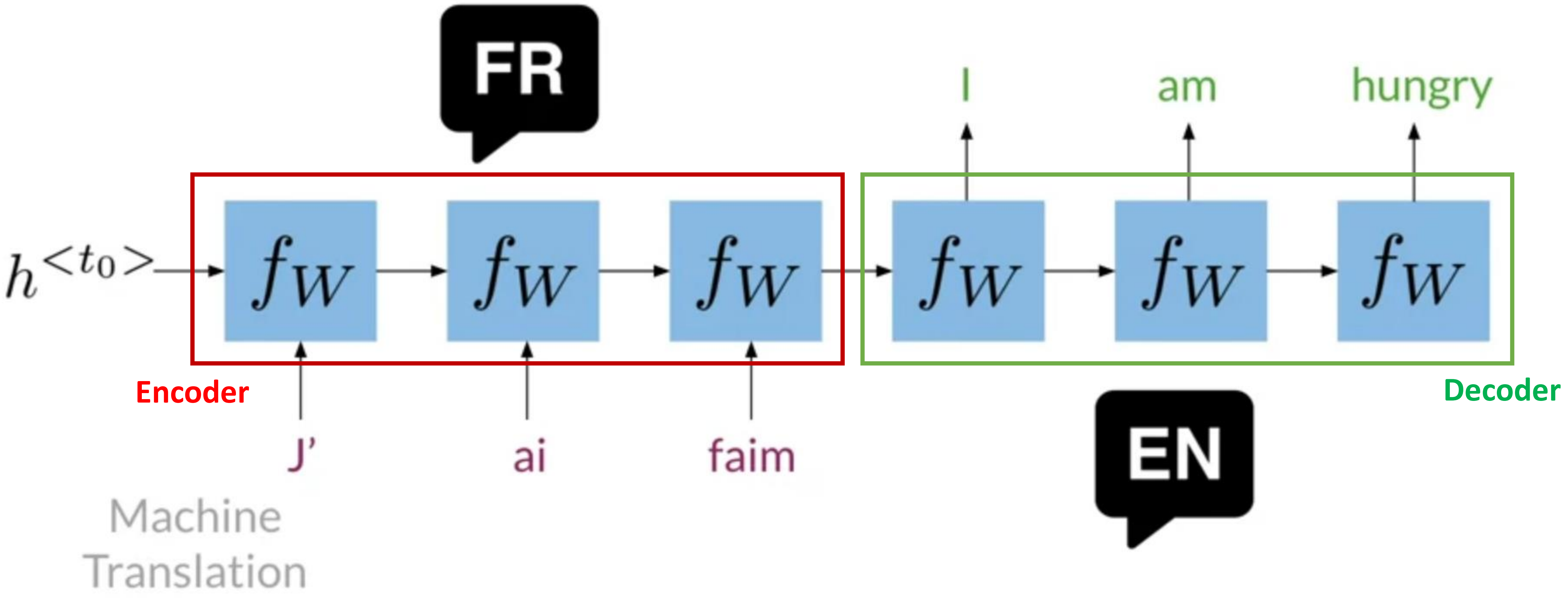
```
# Model definition with GRU
model = tf.keras.Sequential([
    Embedding(vocab_size, embedding_dim, input_length=max_length),
    Dropout(0.5),
    Bidirectional(GRU(32)),
    Dense(6, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

You can definitively achieve better results by tweaking the model a little!



Use cases: Translation

Translation: Illustration



https://www.coursera.org/learn/sequence-models-in-nlp/lecture/VLEGc/applications-of-rnns

Translation: Exercise

course3_translation_LSTM.ipynb

Goal: Have a look at how an English-French translation LSTM model can be trained

Remarks:

- The model is an **adaptation** of some **articles** found on the internet (the references are given inside the notebook)
- The students **do not have to complete anything**, the model would be **too long to train** to work on it during the class
- The results obtain in the current notebook comes from a model which has **not converged yet**, so the results are currently quite **low**

Translation : A closer look at the model

```
model = Sequential()
model.add(Embedding(eng_vocab_size, 512, input_length=eng_length, mask_zero=True))
model.add(LSTM(512))
#------------------------- Choose either the RepeatVector of the Bidirectional
model.add(RepeatVector(fren_length))
#model.add(Bidirectional(LSTM(fren_length), return_sequences = True))
#----------------------------
model.add(LSTM(512, return_sequences=True))
model.add(Dense(fren_vocab_size, activation='softmax'))
```

Translation : A closer look at the model

```
model = Sequential()
model.add(Embedding(eng_vocab_size, 512, input_length=eng_length, mask_zero=True))
model.add(LSTM(512))
#------------------------- Choose either the RepeatVector of the Bidirectional
model.add(RepeatVector(fren_length))
#model.add(Bidirectional(LSTM(fren_length), return_sequences = True))
#--------------------------
model.add(LSTM(512, return_sequences=True))
model.add(Dense(fren_vocab_size, activation='softmax'))
```

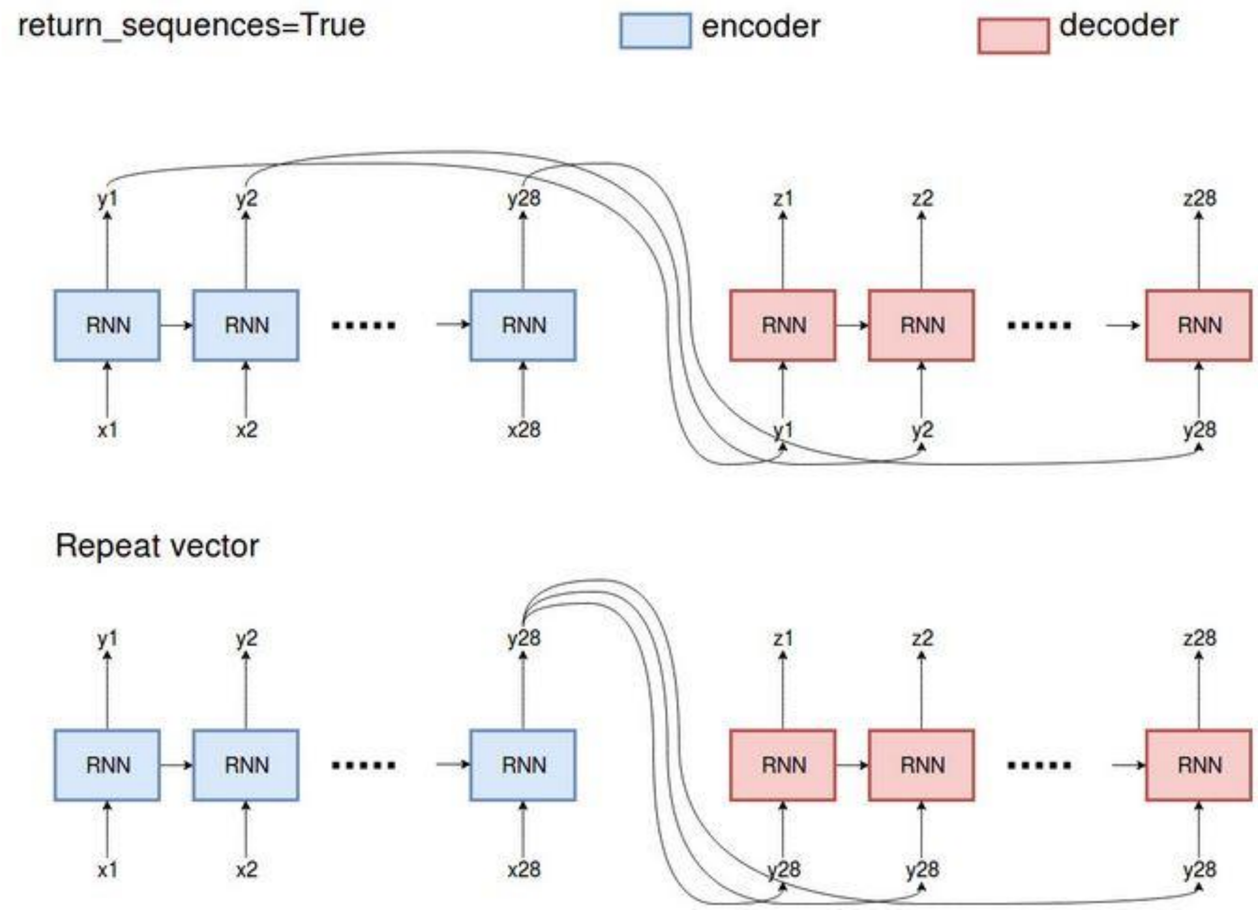
Be careful, this option is needed to feed the LSTM layer coming just after

Two possibilities on this **specific layer** between **RepeatVector** and **Bidirectional(LSTM)**

The goal is to feed the **LSTM layer** coming next after

Translation: ReturnSequence vs RepeatVector

LSTM
with **return_sequences=True**



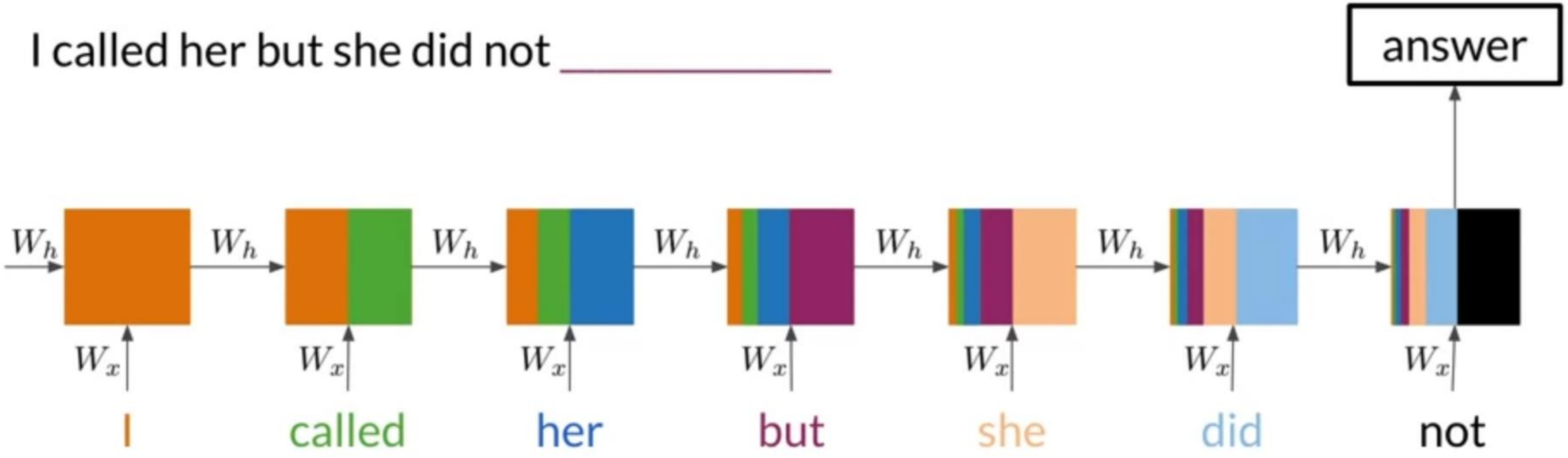
RepeatVector

https://stackoverflow.com/questions/51749404/how-to-connect-lstm-layers-in-keras-repeatvector-or-return-sequence-true

RepeatVector will only return the **last value** of the output sequence to the next layer

Use cases: Text generation

Text generation: Illustration



Text generation: Shakespeare sonnets

- We want to train a model able to generate text with the **same writing style as Shakespeare**

- For that, we use as database a compilation of his sonnets

- To train the model, we use sequences from the sonnet corpus

- As input, we give the **beginning of a sentence** and, as label, the word coming just after

Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model*:*

1. *To* ***be***
2. *To be* ***or***
3. *To be or* ***not***
4. *To be or not* ***to***
5. *To be or not to* ***be***

Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model*:*

1. *To* ***be***
2. *To be* ***or***
3. *To be or* ***not***
4. *To be or not* ***to***
5. *To be or not to* ***be***

labels

Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model*:*

1. *To* ***be***
2. *To be* ***or***
3. *To be or* ***not***
4. *To be or not* ***to***
5. *To be or not to* ***be***

labels

Input sequences for training the model

Text generation: Shakespeare sonnets

Example: the sentence ***To be or not to be*** gives **5 training sequences** to feed the model:

1. *To* ***be***
2. *To be* ***or***
3. *To be or* ***not***
4. *To be or not* ***to***
5. *To be or not to* ***be***

labels

Input sequences for training the model

A n-token sequence in the corpus gives n-1 input sequences for training

Text generation: Exercise

course3_text_generation_LSTM_ex.ipynb

<u>Goal</u>: To solve a complex NLP problem from the text processing to model tuning. In the end, we should get a model able of generating text with Shakespeare characteristic style!

<u>Remark:</u>

- We are not looking for especially good predictions, we only want the model being able to generate text looking like Shakespeare
- If the generated text is not too repetitive and looks like Shakespeare, even if it does not make much sense, you can consider it being good enough

Take-away from Course 3

- For some **complex NLP** applications (**translation**, **text generation**…), it is needed to use **text** as temporal **sequences**
- The **Recurrent Neural Network** (**RNN**) are designed to process such data
- However, **RNN** have some **limitations** such as **loss of information for long sequences** as well as **vanishing gradient issues**
- A **good alternative** are the **LSTM** and their variants (**GRU**)
- Some **specific preprocessing steps** must be performed in order to use texts as sequences able to **train** that kind of model
- If the **train dataset** is **big enough** and the **task specific enough**, Keras allows to easily **train from scratch embeddings** (see ***Embedding* layer**)

References

Book

A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2019)

Online formations

- https://www.udemy.com/course/nlp-natural-language-processing-with-python
- https://www.coursera.org/specializations/natural-language-processing
- https://www.coursera.org/learn/natural-language-processing-tensorflow

Internet sites

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://blog.engineering.publicissapient.fr/2020/09/23/long-short-term-memory-lstm-networks-for-time-series-forecasting/
- https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577