SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

Unione Parmense degli Industriali

GIA Gruppo Imprese
Gruppo Imprese Artigiane

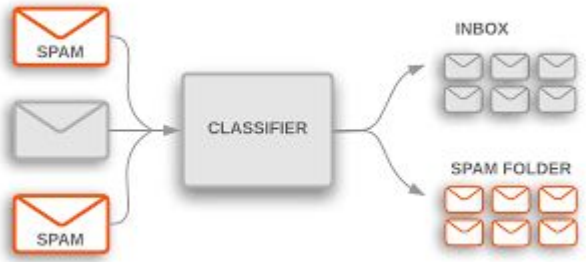
Text classification and LSTM

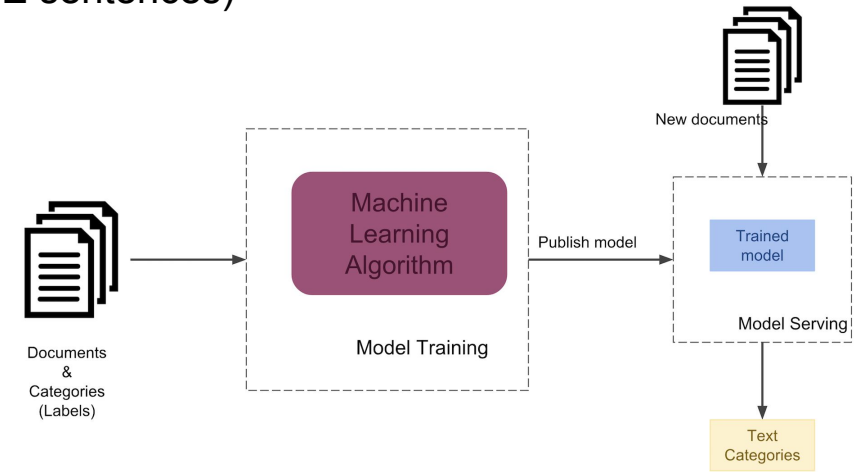
Artificial Intelligence and Machine Learning

Gianfranco Lombardo MEng, Ph.D Candidate in ICT
gianfranco.lombardo@unipr.it

Text classification

- The goal of text classification is to assign documents (such as emails, posts, text messages, product reviews, etc...) to one or multiple categories.
- Text classification is a core problem to many applications:
  - Spam detection (SPAM or NOT SPAM messages)
  - Sentiment Analysis (POSITIVE or NEGATIVE sentences)
  - Smart Replies (gmail)

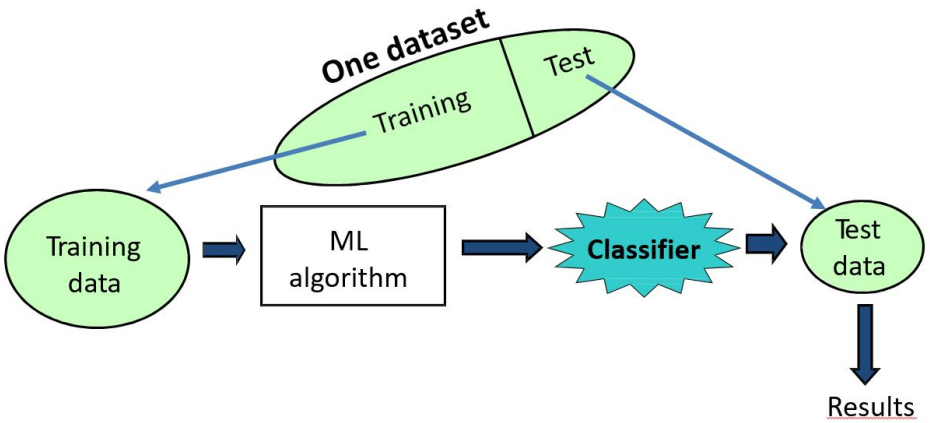




Text mining

Training Set	
Attribute 1	**Class**
• This movie is awsome	POSITIVE
• I didn't like that movie so much	NEGATIVE

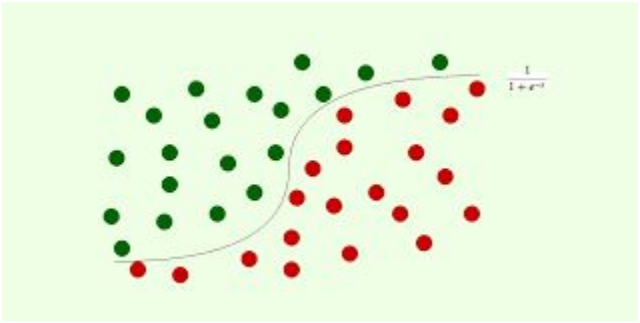
Test Set	
Attribute 1	**Class**
• I really enjoyed that movie	POSITIVE



Problem: What are the features ?

Classical Dataset for Classification

X_1	X_2	Class
4.2	1.3	Class A
4.1	1.6	Class B
..	..	..



Sentence Classification

Attribute 1	Class
• This movie is awsome	POSITIVE
• I didn't like that movie so much	NEGATIVE

?

The Bag-Of-Word model (BoW)

- We build a vocabulary with all of the words and encode each sentence with binary vectors
- Vectors' size is the same of the vocabulary. Each vectors' component is a feature
- We can group words to get more sophisticated features (N-gram)

(1) John likes to watch movies.
(2) John also likes to watch football games.

Collect the Bag of words

John	likes	to	watch	movies	also	football	games	Mary	too

Use the Bag of Words to represent the sentences with numbers

BOW with Count Value

	John	likes	to	watch	movies	also	football	games	Mary	too
(1)	1	1	1	1	1	0	0	0	0	0
(2)	1	1	1	1	0	1	1	1	0	0

BoW with Python

```
from sklearn.feature_extraction.text
import CountVectorizer

corpus = ['This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?']

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names())

print(X.toarray())
```

Output

```
vectorizer.get_feature_names()

['and', 'document', 'first',
'is', 'one', 'second', 'the',
'third', 'this']

----------------------------

X.toarray()

[[0 1 1 1 0 0 1 0 1]
[0 2 0 1 0 1 1 0 1]
[1 0 0 1 1 0 1 1 1]
[0 1 1 1 0 0 1 0 1]]
```

BoW with Python - Ngrams

```
...

corpus = ['This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?']

...

CountVectorizer(ngram_range=(1,2))

...
```

Output

```
vectorizer.get_feature_names()

['and', 'and this', 'document',
'document is', 'first', 'first
document', 'is', 'is the', 'is
this', 'one', 'second', 'second
document', 'the', 'the first',
'the second', 'the third',
'third', 'third one', 'this',
'this document', 'this is',
'this the']
```

BoW with Python - Most frequented words

- We can select a maximum number of desidered features
- Only most frequent words in the corpus are encoded

```
...

corpus = ['This is the first document.' ,
'This document is the second document.' ,
'And this is the third one.' ,
'Is this the first document?' ]

...

CountVectorizer(max_features=3)

#it takes the most frequent

...
```

Output

```
vectorizer.get_feature_names(
)
['document', 'is', 'the']

X.toarray()

[[1 1 1]
[2 1 1]
[0 1 1]
[1 1 1]]
```

TF-IDF

- The tf–idf or TFIDF, short for ***Term Frequency–Inverse Document Frequency***, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

  - Document **A** has words**,** *«hello»* appears **5** times in the document **A**:

$$\mathbf{Tf}_{\mathbf{hello,A}} = \frac{5}{100} = 0.05$$

  - If we have **1000** documents in the collection, and *«hello»* appears in **10** documents:

$$\mathbf{Idf}_{\mathbf{hello}} = \log\frac{1000}{10} = 2$$

  - So:

$$\mathbf{Tf\text{-}Idf}_{\mathbf{hello,A}} = 0{,}05 * 2 = 0.1$$

BoW with TF-IDF

```
from sklearn.feature_extraction.text
import TfidfVectorizer

corpus = ['This is the first document.',
'This document is the second document.',
'And this is the third one.',
'Is this the first document?']

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names())
print(X.toarray())
```

Output

```
vectorizer.get_feature_names()

['and', 'document', 'first', 'is',
'one', 'second', 'the', 'third',
'this']

----------------------------

X.toarray()

[[0.  0.46 0.58 0.38 0.  0.  0.38 0.
0.38]
 [0.  0.68 0.  0.28 0.  0.53 0.28 0.
0.28]
 [...]
]
```

Exercise: Sentiment analysis

- Classify the review in “corpus.csv” (Sentiment Analysis)
  - The structure is class#!#document
    - class: Category to predict, can be positive or negative
    - document: Content of reviews
    - sep =”#!#”

- Read the csv with pandas and create a vector y and a matrix with documents X

- Divide in train and test
  - X_train, X_test, y_train, y_test = train_test_split(X, y)
- Encode categorical y_train and y_test as in the code:

```
# label encode the target variable Y
encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.fit_transform(y_test)
```

- Follow the example for BoW
- Design a classification system choosing an algorithm among Logistic regression, Decision tree, Adaboost, Gradient boosting or XGBoost
- Compute the accuracy

Exercise: Sentiment analysis

- Visualize (matplotlib) how the accuracy change by changing the max_features (step = 1000)
  - CountVectorizer(max_features=1000) … CountVectorizer(max_features=10000)

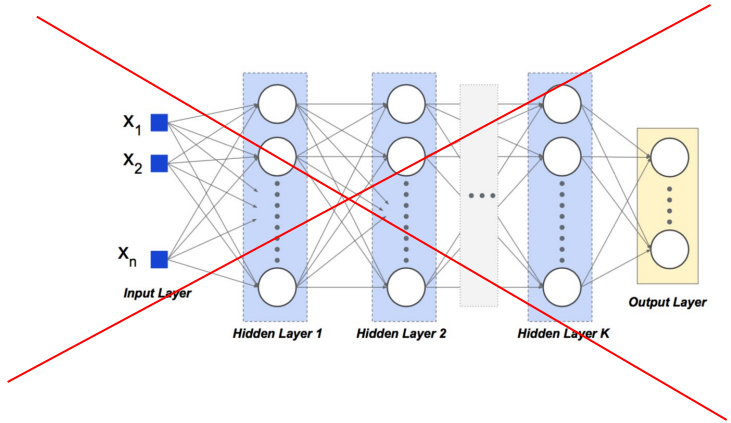
- Find out which is the best ngram_range to consider:
  - CountVectorizer(ngram_range=(1)) … CountVectorizer(ngram_range=(1,3))

- Try The tf-Idf Vectorizer
  - TfidfVectorizer()

BREAK

Towards Recurrent Neural Network

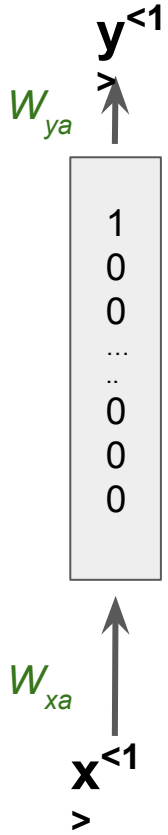
- If we want to deal with text or time series traditional Neural networks are not a good choice:
  - For text: If i have a vocabulary of 10k words, with one-hot encoding my input layers will have a 10k*n°words dimension or 10k for BoW with sentences
  - The ANN doesn't share features learned across different position of text (or signal) -> no temporal information
  - We need to use different models!
    - CNN with unit kernel can be a possibility
    - Recurrent Neural Network



Recurrent Neural Network (RNN)

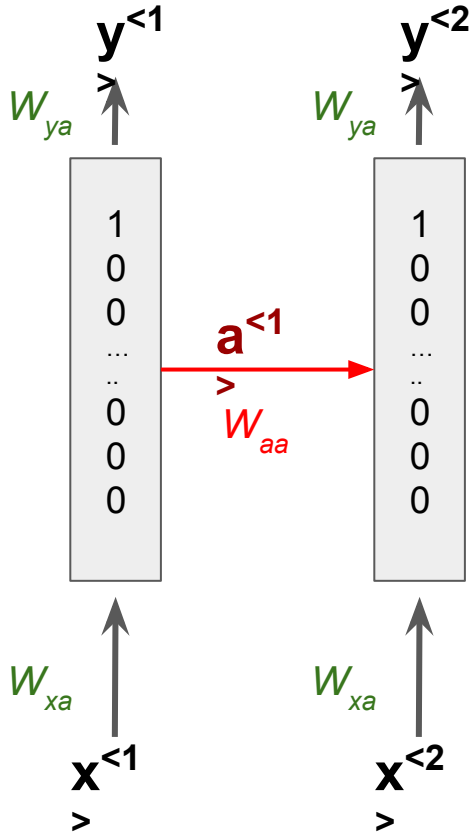
- Let’s consider text translation (From english to italian for example):
  - We encoded our text with one-hot encoding, so we have a list of words in the form of vectors
  - Each word in english is $\mathbf{x}^{<j>}$ and the correspondent in italian is $\mathbf{y}^{<j>}$ where j is the position index inside the text
  - For sake of simplicity let’s also make the hypothesis that english and italian sentences have the same length T = Tx = Ty , so words go from 1….T
  - Once we translated $\mathbf{x}^{<1>}$ into $\mathbf{y}^{<1>}$, when we have to translate $\mathbf{x}^{<2>}$ we want to take into account the previous word $\mathbf{x}^{<1>}$

Recurrent Neural Network (RNN)



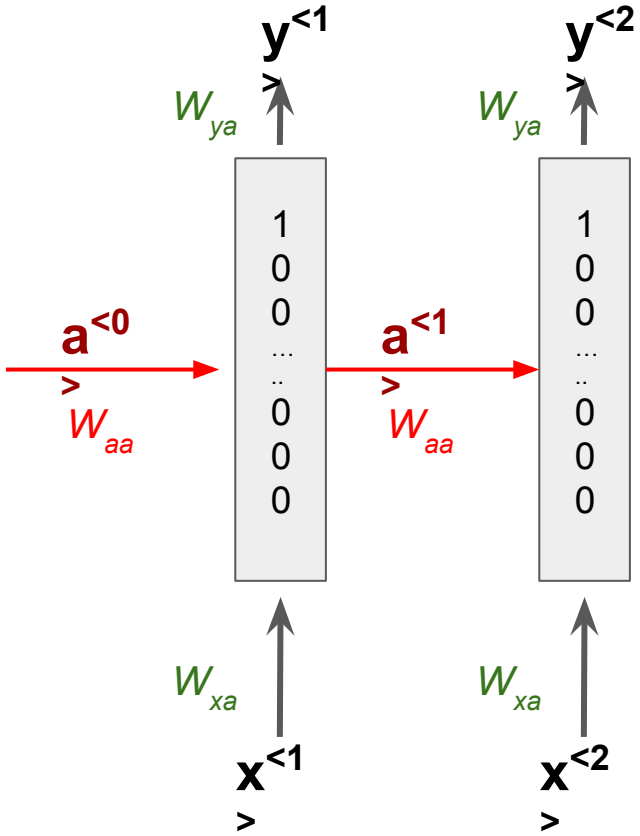
- A unit can be seen as a traditional ANN where we have an input and a predicted output
- Input and output are associated with two weights matrices W_{xa} and W_{ya} (random initialized)
- What happens when we want to classify $x^{<2>}$ by keeping into account $x^{<1>}$?
- We want to have memory of the previous words each time !

Recurrent Neural Network (RNN)



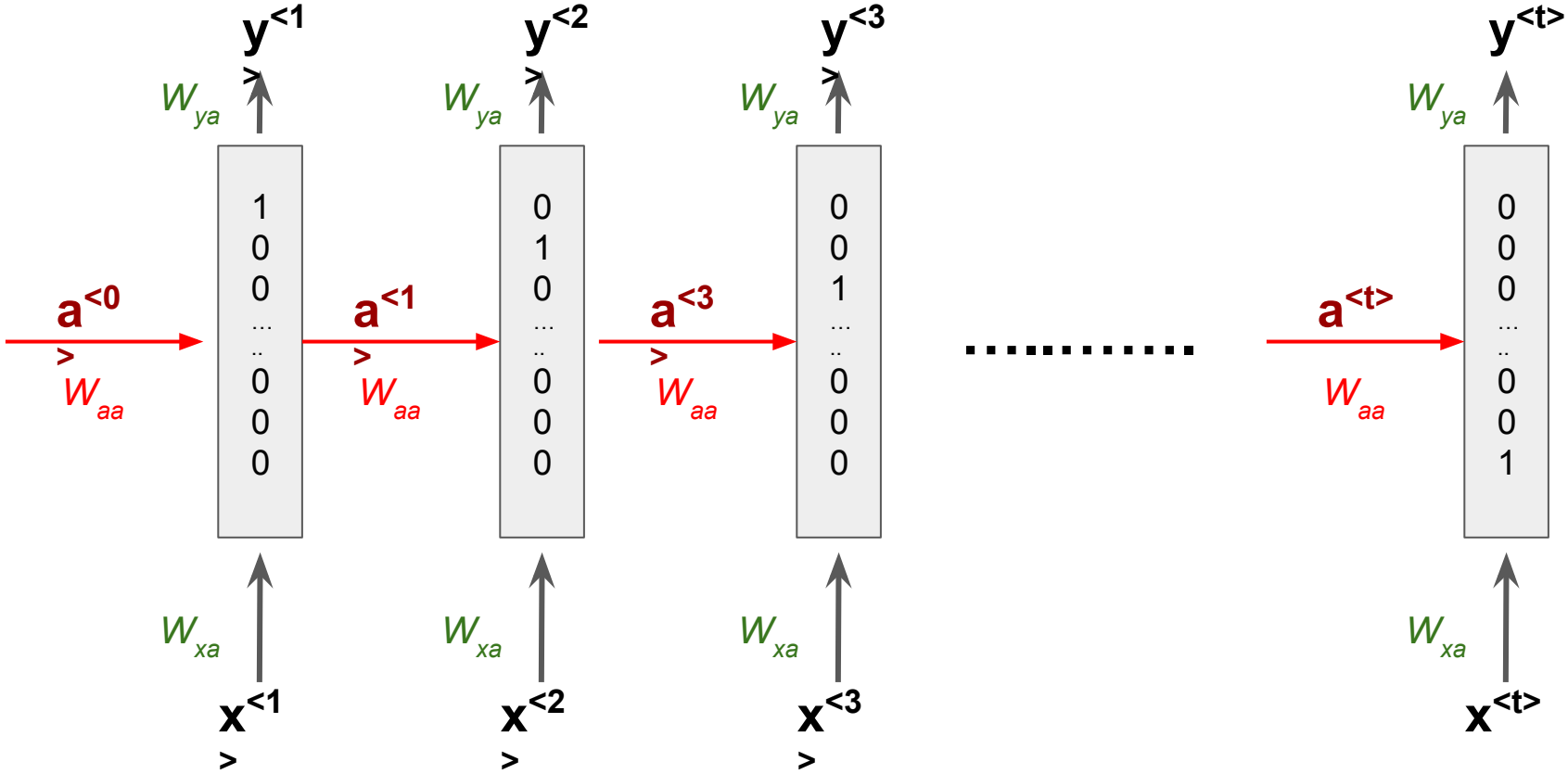
- When we consider the 2° word $x^{<2>}$, to predict the label $y^{<2>}$ we will use also some information from the previous steps in the form of the "activation value" $a^{<1>}$
- A new matrix of weights W_{aa} governs the transmission of the past information
- W_{ax} , W_{ay} and W_{aa} are shared across the sentence processing

Recurrent Neural Network (RNN)

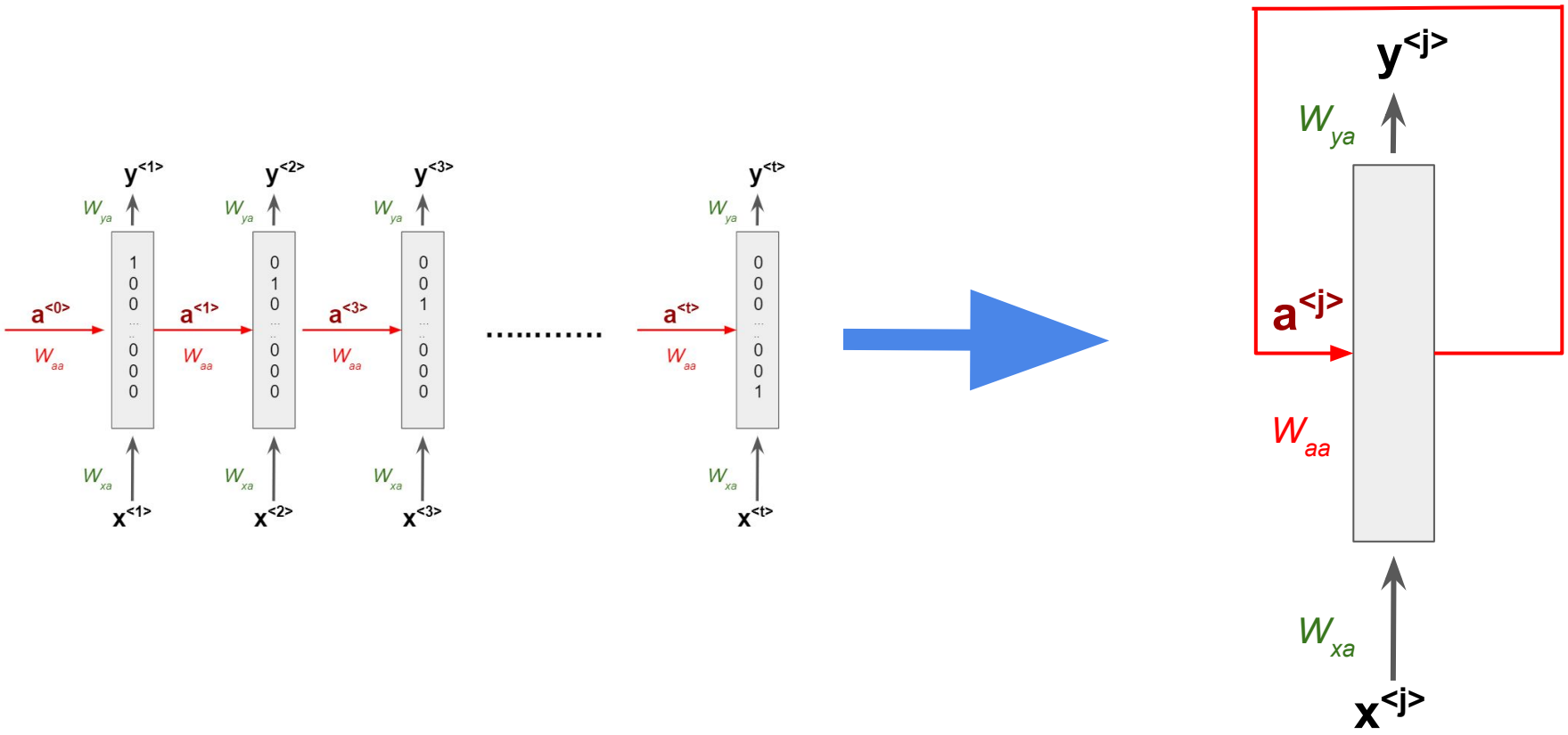


- For consistency we add an initial activation value for the processing of the first word: $\mathbf{a}^{<0>}$
- $\mathbf{a}^{<0>}$ is usually initialized as zero

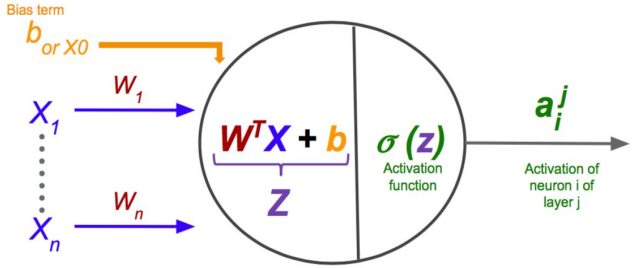
Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



Forward propagation



- Think to the RNN's unit as a neuron in a traditional ANN
- $a^{<0>} = 0$
- $a^{<1>} = \sigma(W_{aa} \cdot a^{<0>} + W_{xa} \cdot x^{<1>} + b_a)$
- $y^{<1>} = \sigma(W_{ya} \cdot a^{<1>} + b_y)$

More general notation

$$a^{<j>} = \sigma\,(W_{aa} \bullet a^{<j-1>} + W_{xa} \bullet x^{<j>} + b_a) \longrightarrow a^{<j>} = \sigma(W_A \bullet [\,a^{<j-1>}, x^{<j>}] + b_a)$$

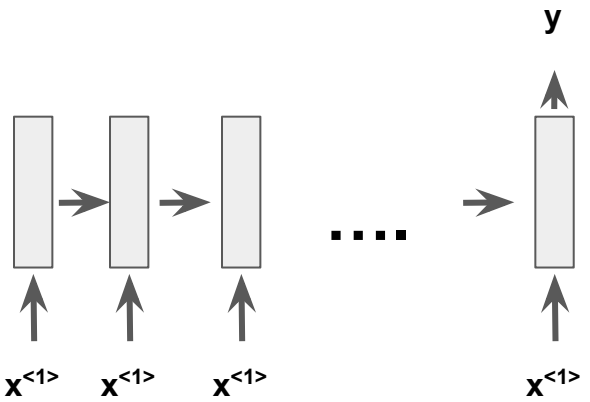
$$y^{<j>} = \sigma(W_{ya} \bullet a^{<j>} + b_y)$$

Backpropagation Through Time

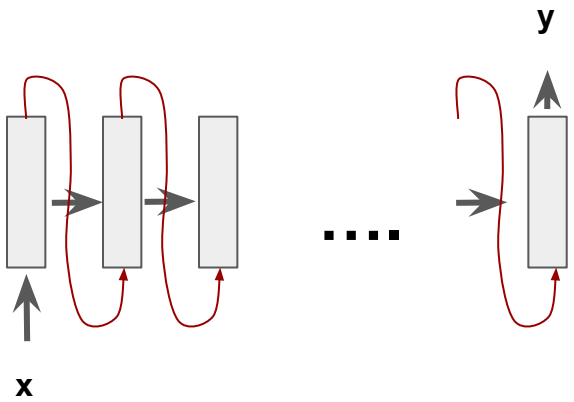
- Considering the forward propagation in the previous slide:
  - $a^{<j>} = \sigma(W_A \bullet [\, a^{<j-1>}, x^{<j>}] + b_a)$
  - $y^{<j>} = \sigma(W_{ya} \bullet a^{<j>} + b_y)$
- Things that “goes back” are from up to down and also from right to left (time axis)
- The Loss function (e.g., Cross-entropy) for each block (so for each word) is:
  - $L^{<j>}(y^{*<j>}, y^{<j>}) = - y^{<j>} \bullet \log(y^{*<j>}) - (1 - y^{<j>}) \bullet \log(1- y^{*<j>})$
- Loss for the entire sequence is defined as the sum from 1 to T_y of the loss

Different RNN architectures

Many-to-one (e.g., Sentiment analysis)

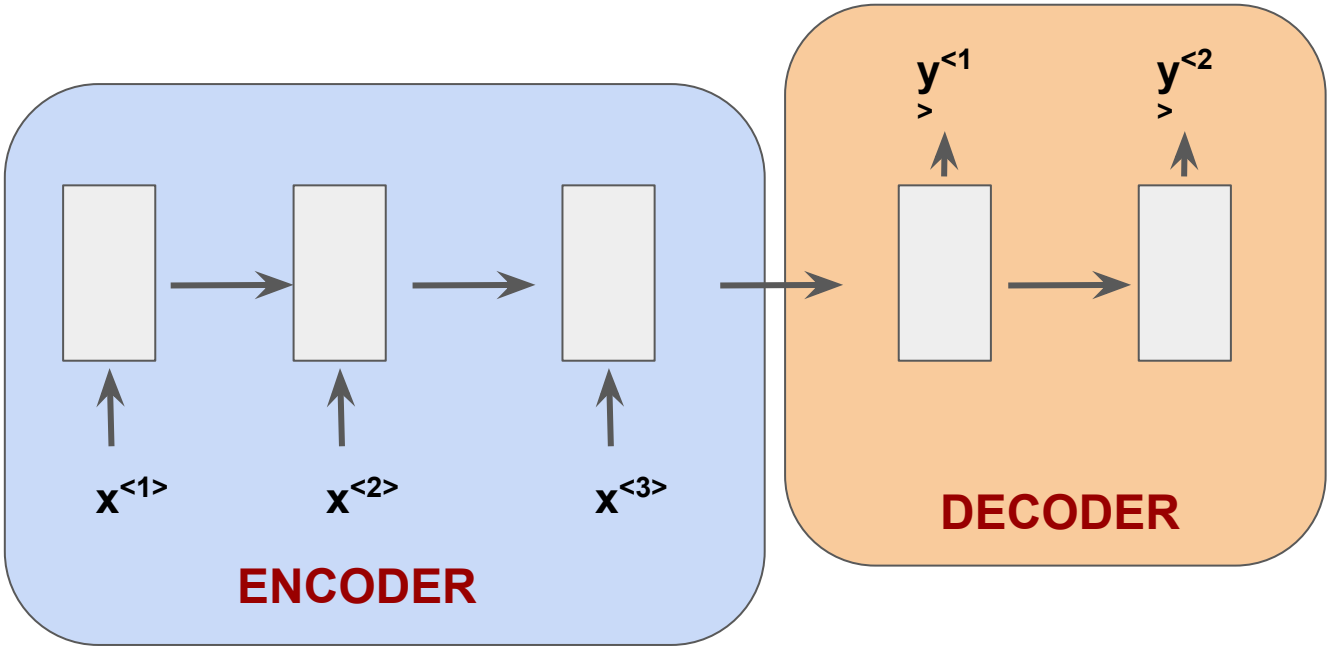


One-to-many



Different RNN architectures

- Many-to-many (e.g., Machine translations) and T_x and T_y can have different size



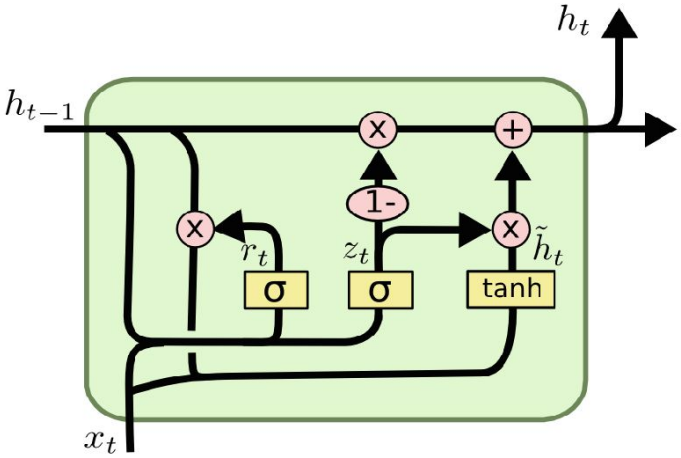
Problems with RNN

- The RNN is not good at capturing long-term dependencies
  - We should make a very deep neural network
  - Back-propagation is difficult because on the basis of the final y^*, changes will affect also the starting layers
  - The gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. (***Vanishing gradient***)

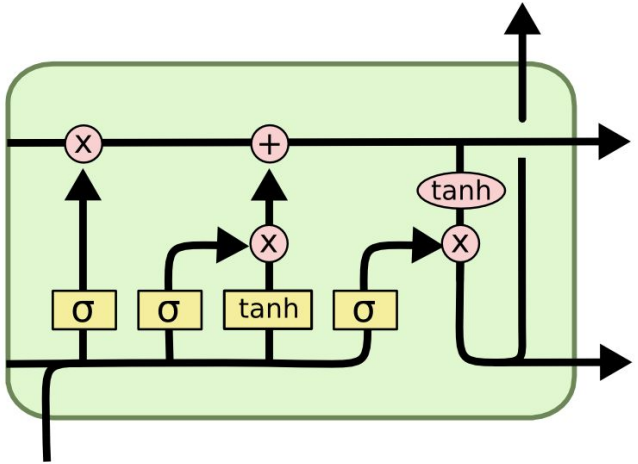
  - The opposite problem is that gradient could explode with deep RNN
    - The value of weights become NaN because it overflows (***Exploding gradient***).
    - One solution is Gradient Clipping: Rescale values when above a certain threshold

Change the hidden unit to reduce vanishing problem

Gated Recurrent Unit (GRU)

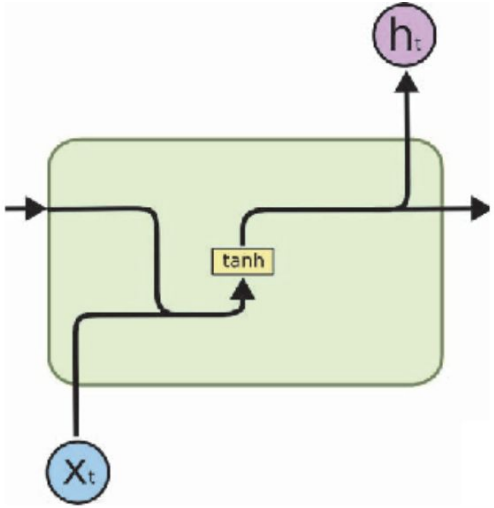


Long Short Term Memory (LSTM)

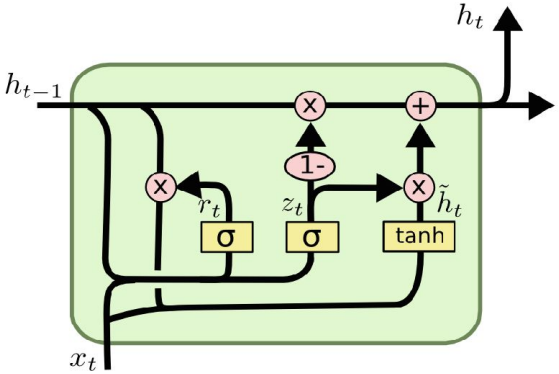


Comparison among different types of Recurrent Neural Network

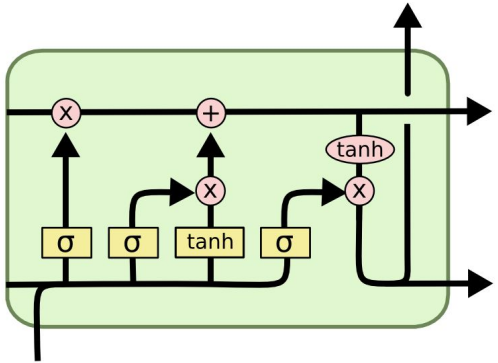
RNN



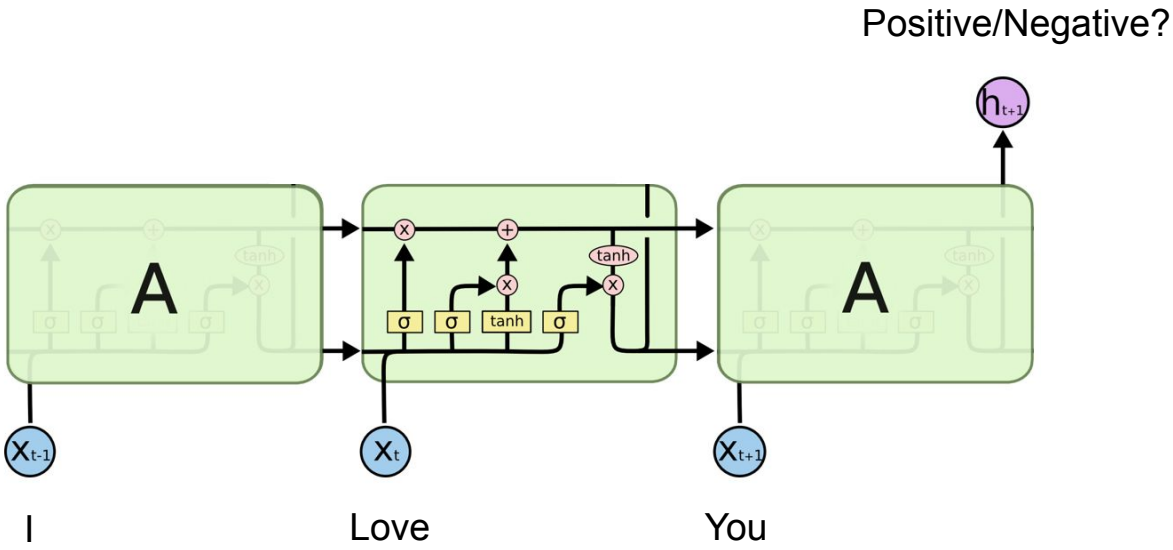
GRU



LSTM



LSTM for text classification



LSTM for text classification

One Hot Encoding

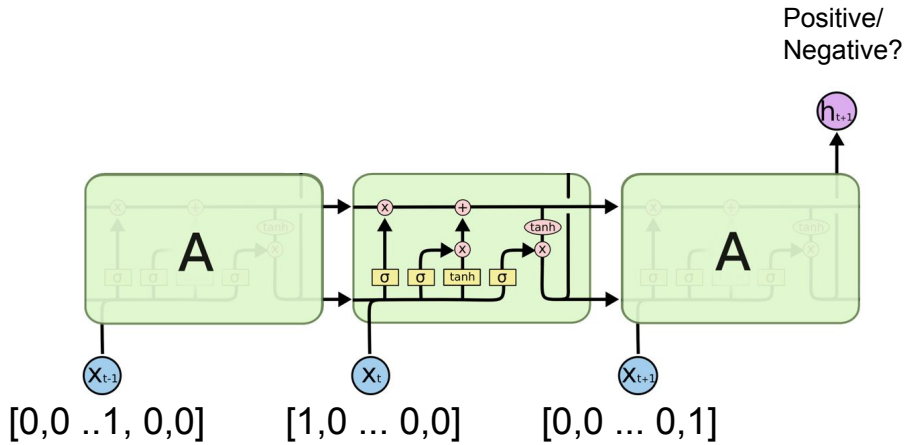
Bag of Words Model

Rome	Paris	To	watch	movies	also	football	too

Rome
Paris
word V

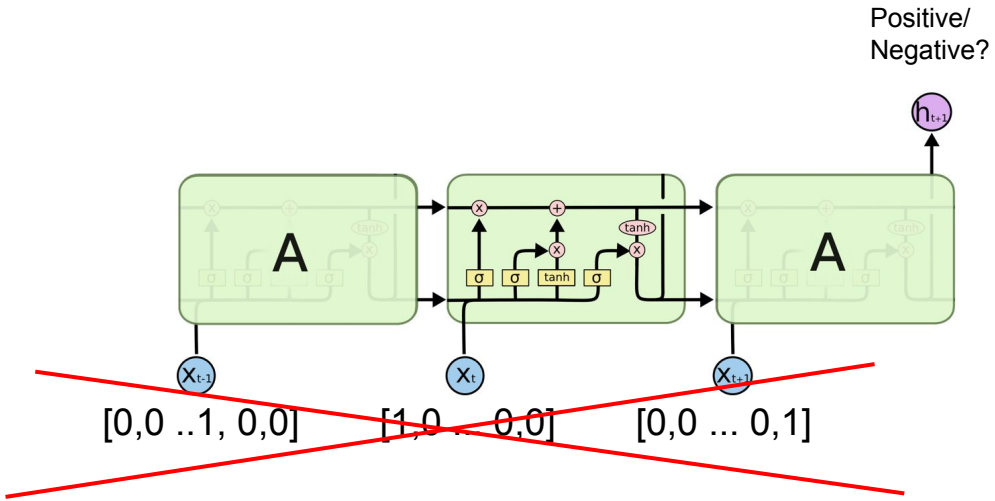
```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

LSTM Network with One Hot Encoding



One-hot-encoding is not efficient for LSTMs

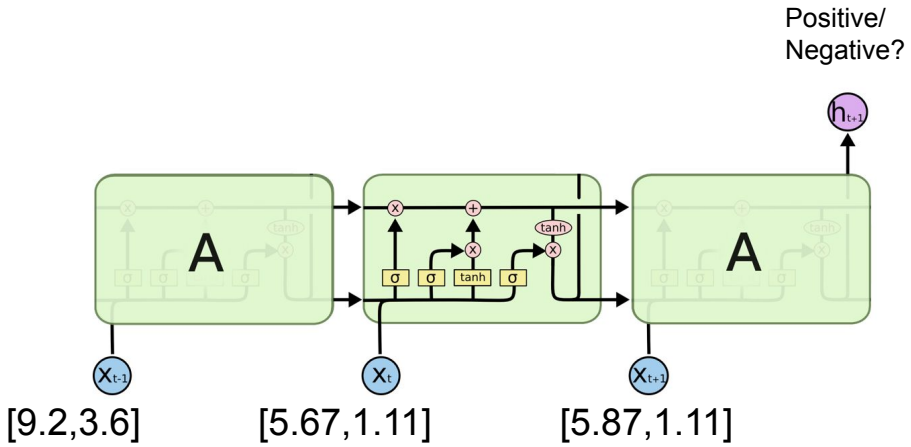
- One hot encoding is a very inefficient way to represent words for text classification with LSTM
- The dimension of the encoding vector increase with the number of words inside the Bag of Words Model
- [0,0,0,0,0,0,0,0,0,0,0 ..... 1..... 00000]



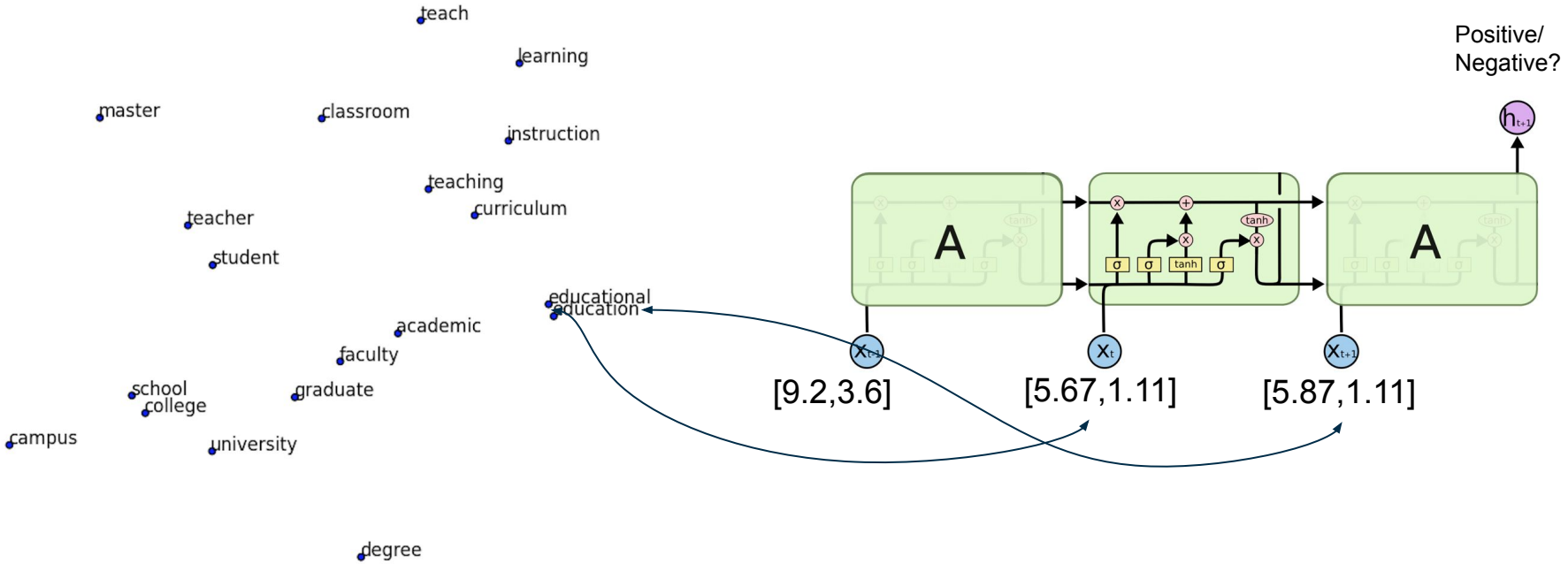
Word Embedding layer

```
#....
model.add(Embedding(...))
#....
```

- Embedding Layer turns words into real vectors (non sparse vector, computationally efficient)
- The vector is N-dimensional and the dimension is a parameter that we can set



Word Embedding layer



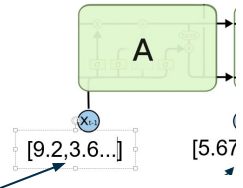
Transform the dataset

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences


max_fatures = 15000

tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(reviews)
X = tokenizer.texts_to_sequences(reviews)
X = pad_sequences(X, maxlen=150)
```

- Keras offers its own Tokenizer (as Sklearn offers CountVectorizer and tfdif vectortizer)
  - Padding sequences is necessary when sentences are shorter than maxlen

LSTM with Keras

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```




```
model = Sequential()
```

The length of the vector for the words representation

```
model.add(Embedding(max_fatures, 64, input_length = X.shape[1]))
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Exercise 2 - Sentiment Analysis with LSTM

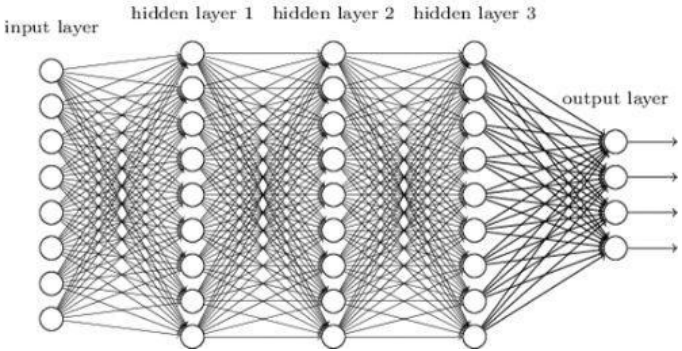
- Repeat the exercise of sentiment analysis starting from corpus.csv
- Design your own LSTM deciding all the parameters (try embedding dimension of 100 to begin)
- Plot the loss or the accuracy with “history object” to understand if you are overfitting or not

Exercise 2 - Sentiment Analysis with LSTM

- Classify your own sentences !!

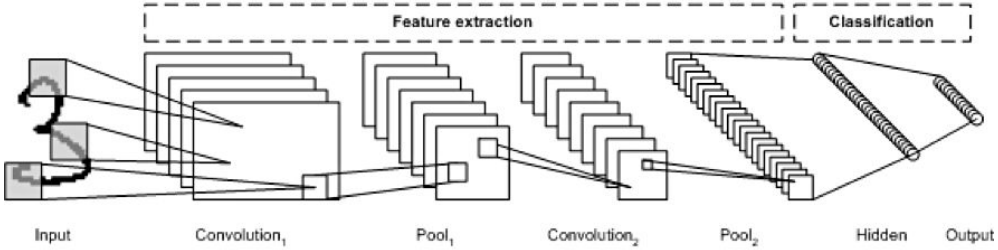
```
# replace with the data you want to classify

newtexts = ["i love you", "I Hate you", "yes, this movie is amazing" ,"New
Zealand is beautiful"]

# note that we shouldn't call "fit" on the tokenizer again
sequences = tokenizer.texts to sequences(newtexts)
data = pad sequences(sequences, maxlen=150)
# get predictions for each of your new texts
predictions = model.predict_classes(data)

print(predictions)
print(encoder.inverse_transform(predictions))
```

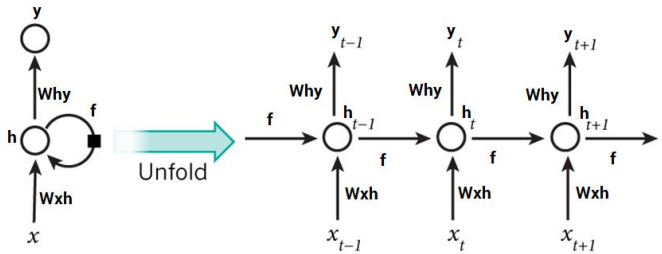
Summary on Deep Learning



Deep Neural Network

Convolutional Neural Network (CNN)





Recurrent Neural Network (RNN)

Why Deep Learning ?