UNIVERSITÀ DI PARMA

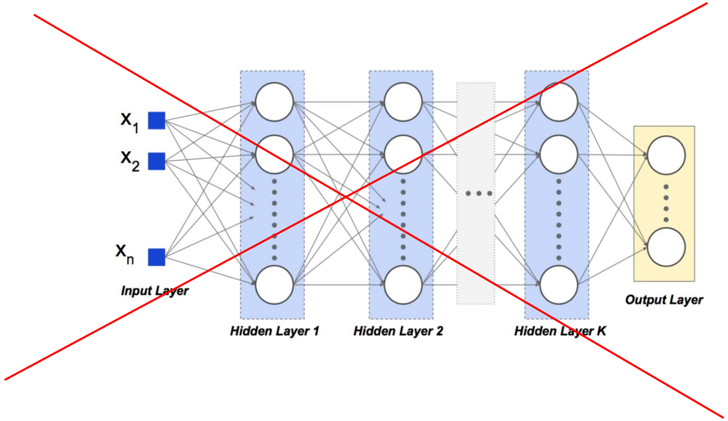
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Learning from sequences- RNN

Gianfranco Lombardo, Ph.D
gianfranco.lombardo@unipr.it

Towards Recurrent Neural Network

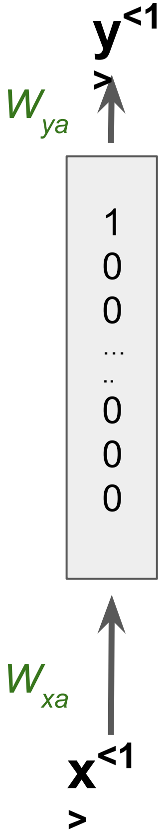
- If we want to deal with text or time series traditional Neural networks are not a good choice:
  - For text: If i have a vocabulary of 10k words, with one-hot encoding my input layers will have a 10k*n°words dimension or 10k for BoW with sentences
  - The ANN doesn’t share features learned across different 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)

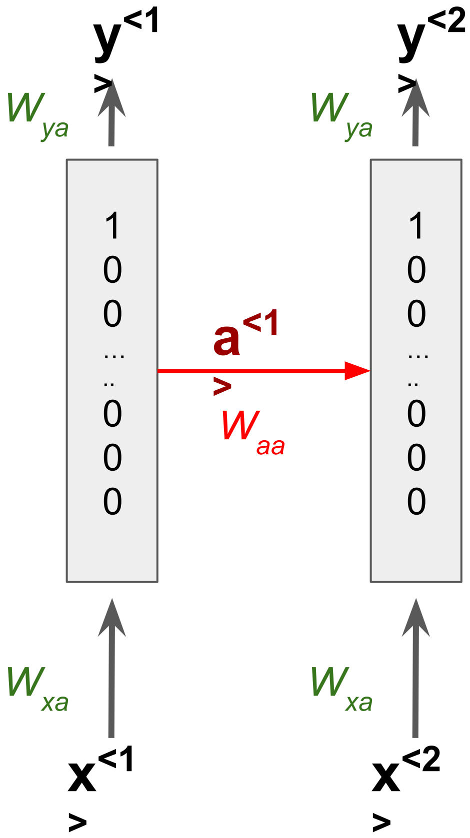
- Let's consider text translation (From english to italian for example):
  - We encoded our text with one-hot encoding, so we have a list of words in the form of vectors
  - Each word in english is $\mathbf{x}^{<j>}$ and the correspondent in italian is $\mathbf{y}^{<j>}$ where j is the position index inside the text
  - For sake of simplicity let's also make the hypothesis that english and italian sentences have the same length $\boldsymbol{T = Tx = Ty}$, 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)



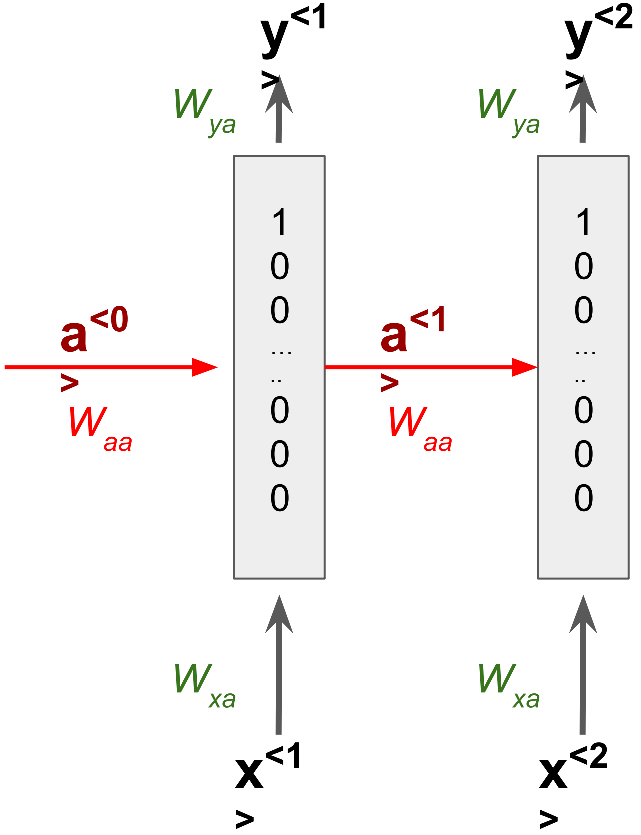
- A unit can be seen as a traditional ANN where we have an input and a predicted output
- Input and output are associated with two weights matrices W_{xa} and W_{ya} (random initialized)
- What happens when we want to classify $x^{<2>}$ by keeping into account $x^{<1>}$?
- We want to have memory of the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

Recurrent Neural Network (RNN)



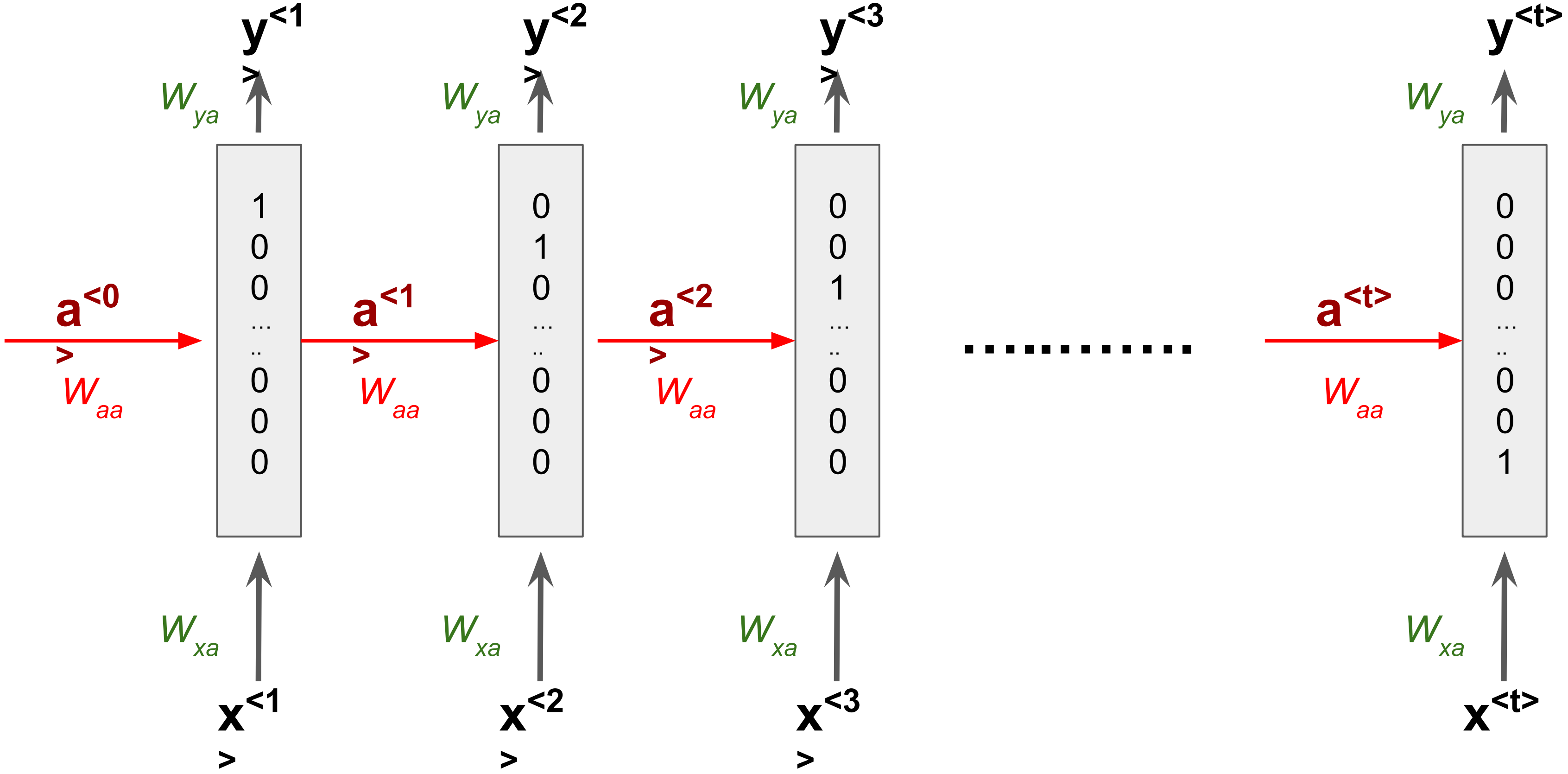
- When we consider the 2° word $x^{<2>}$, to predict the label $y^{<2>}$ we will use also some information from the previous steps in the form of the "activation value" $a^{<1>}$
- A new matrix of weights W_{aa} governs the transmission of the past information
- W_{ax} , W_{ay} and W_{aa} are shared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

Recurrent Neural Network (RNN)



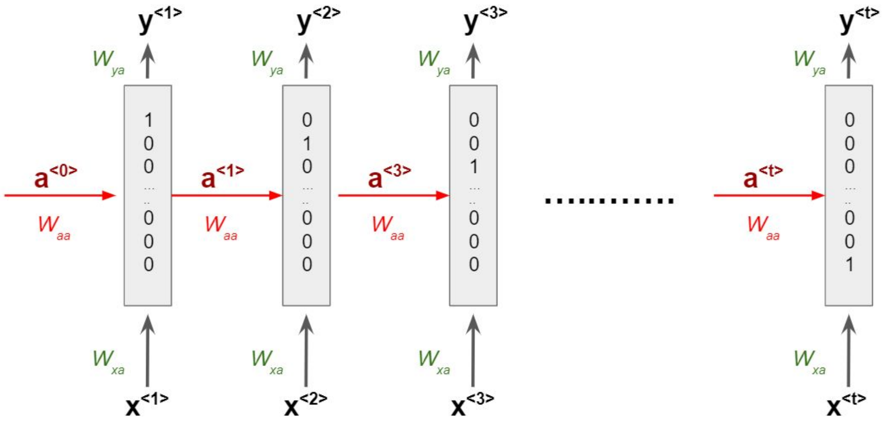
- For consistency we add an initial activation value for the processing of the first word: $\mathbf{a}^{<0>}$
- $\mathbf{a}^{<0>}$ is usually initialized as zero

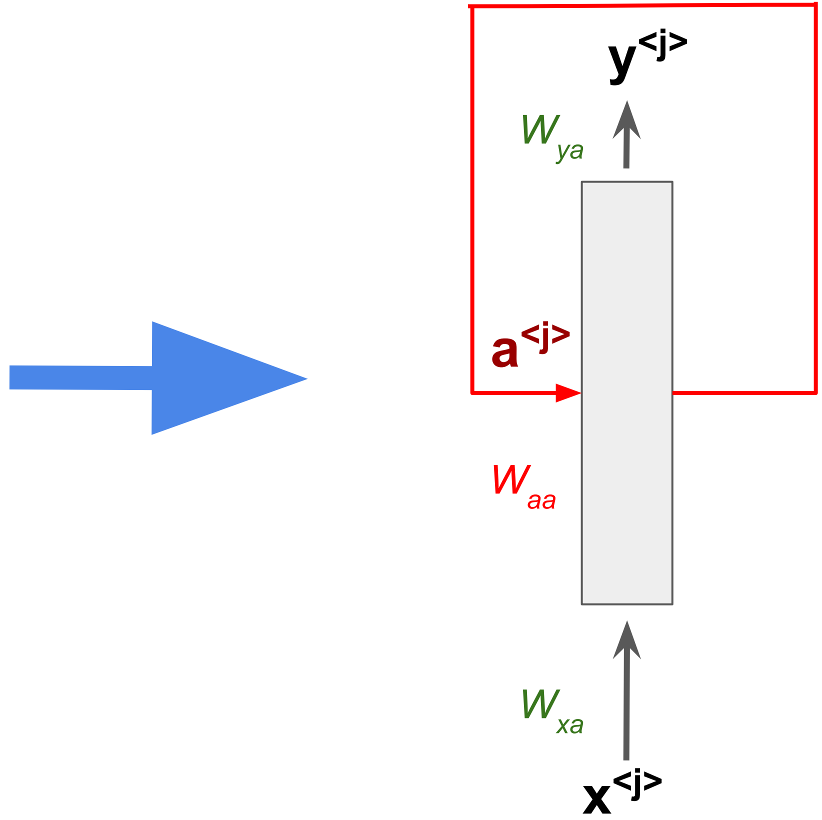
Recurrent Neural Network (RNN)



Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

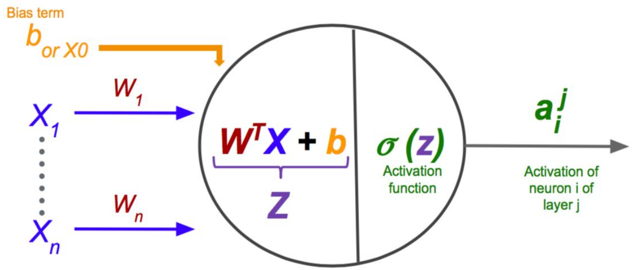
Recurrent Neural Network (RNN)





Forward propagation

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



More general notation

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

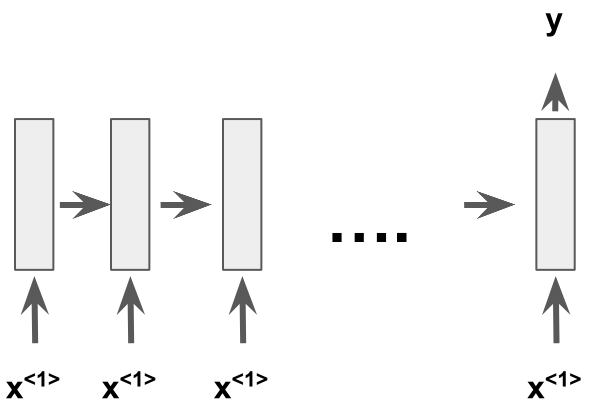
$$y^{<j>} = \boldsymbol{\sigma}(W_{ya} \bullet a^{<j>} + b_y)$$

Backpropagation Through Time

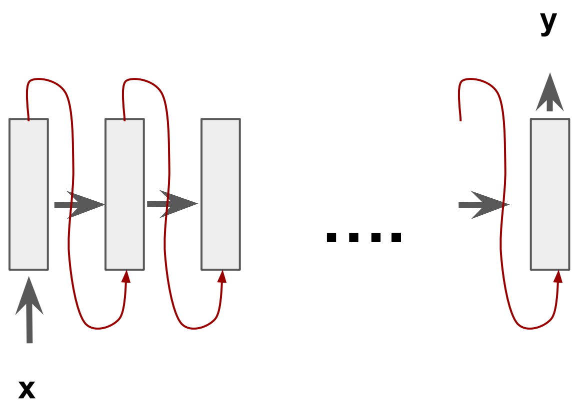
- Considering the forward propagation in the previous slide:
  - $a^{<j>} = \boldsymbol{\sigma}(W_A \bullet [\, a^{<j-1>}, x^{<j>}] + b_a)$
  - $y^{<j>} = \boldsymbol{\sigma}(W_{ya} \bullet a^{<j>} + b_y)$
- Things that “goes back” are from up to down and also from right to left (time axis)
- The Loss 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)

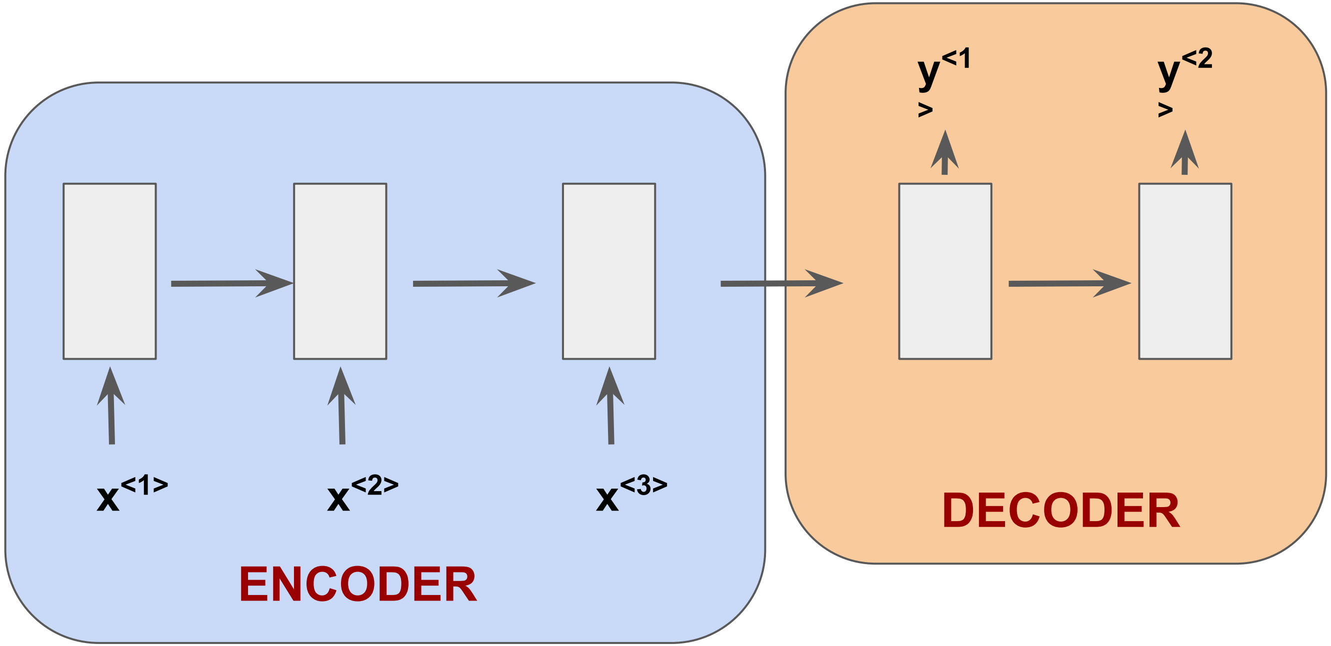


One-to-many



Different RNN architectures

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



BREAK

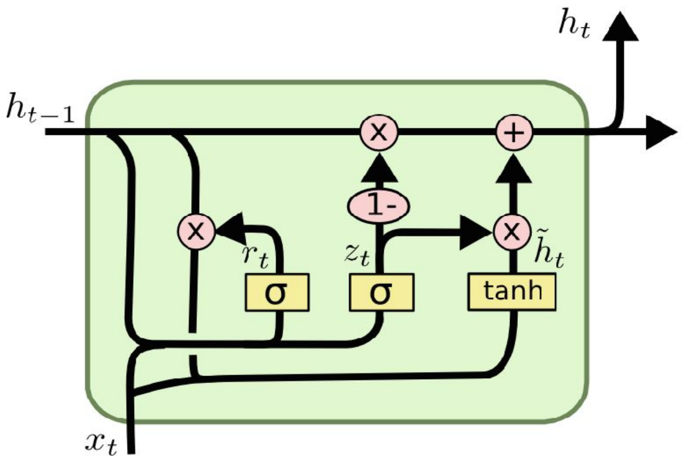
Problems with RNN

- 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***)

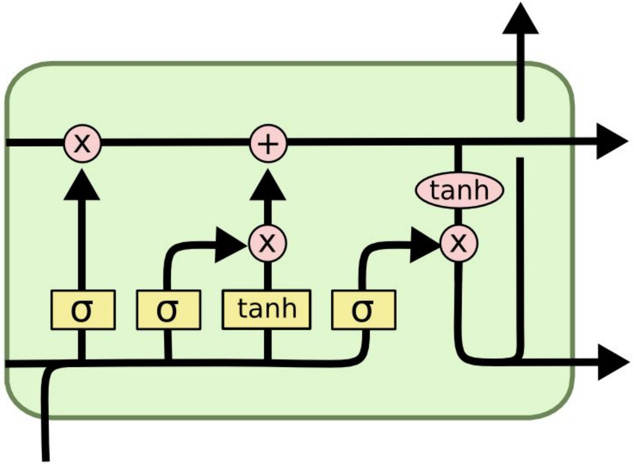
  - The opposite problem is that gradient could explode with deep RNN
    - The value of weights become NaN because it overflows (***Exploding gradient***).
    - One solution is Gradient Clipping: Rescale values when above a certain threshold

Change the hidden unit to reduce vanishing problem

Gated Recurrent Unit (GRU)

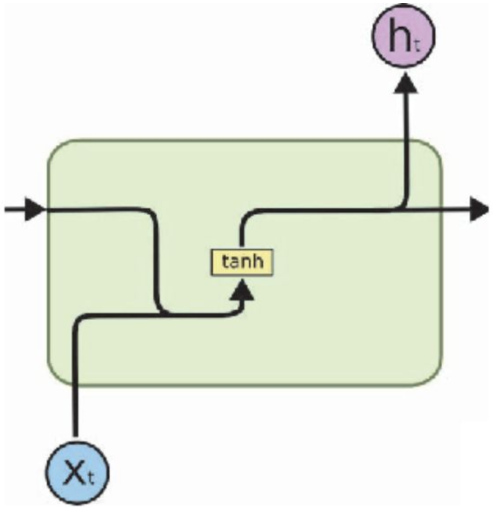


Long Short Term Memory (LSTM)

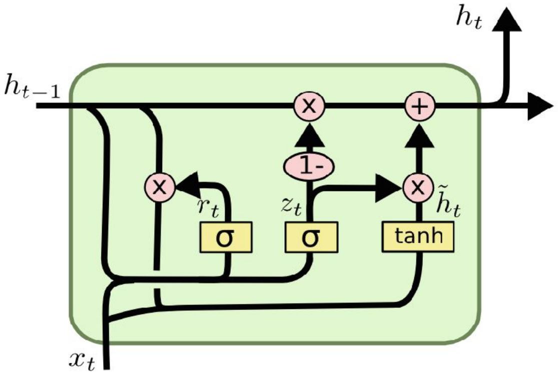


Comparison among different types of Recurrent Neural Network

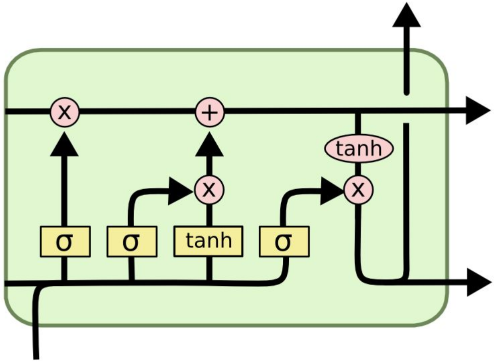
RNN



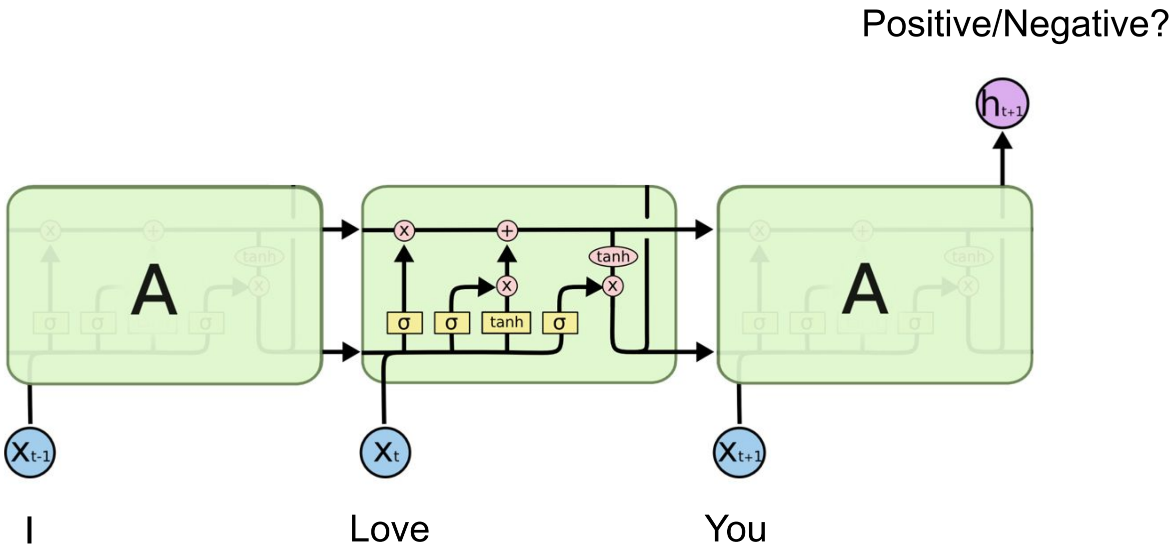
GRU



LSTM



LSTM for text classification



LSTM for text classification

One Hot Encoding

Bag of Words Model

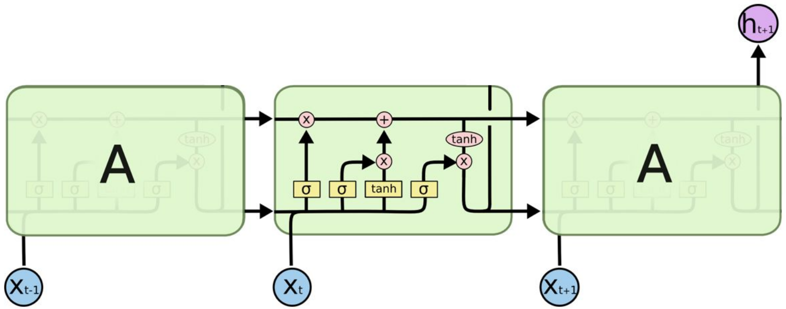
Rome	Paris	To	watch	movies	also	football	too

Rome → 1st position, Paris → 2nd position, word V → last position

```
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

Positive/
Negative?



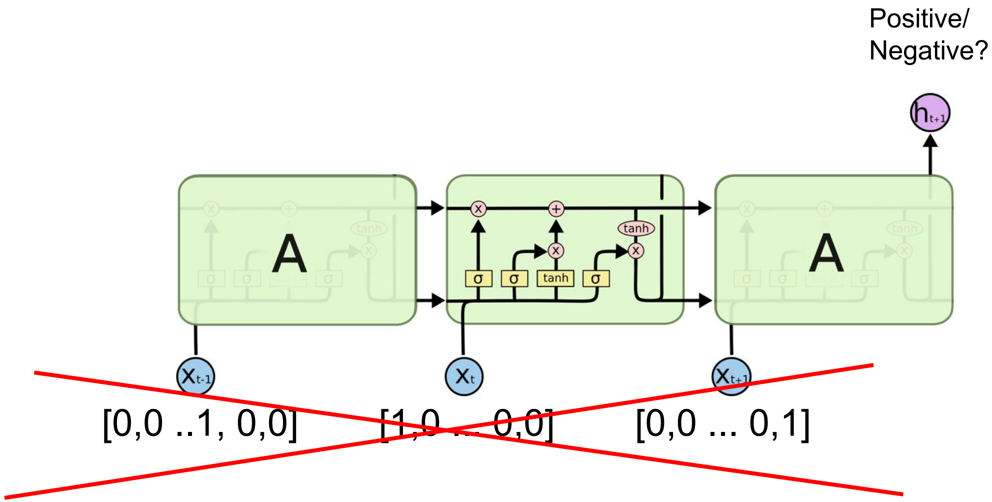
[0,0 ..1, 0,0] [1,0 ... 0,0] [0,0 ... 0,1]

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

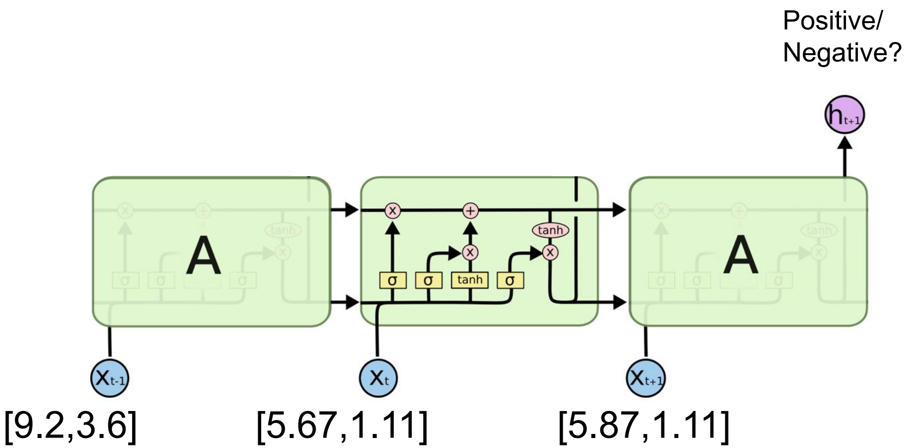
- □ [0,0,0,0,0,0,0,0,0,0,0 .... 1...... 00000]



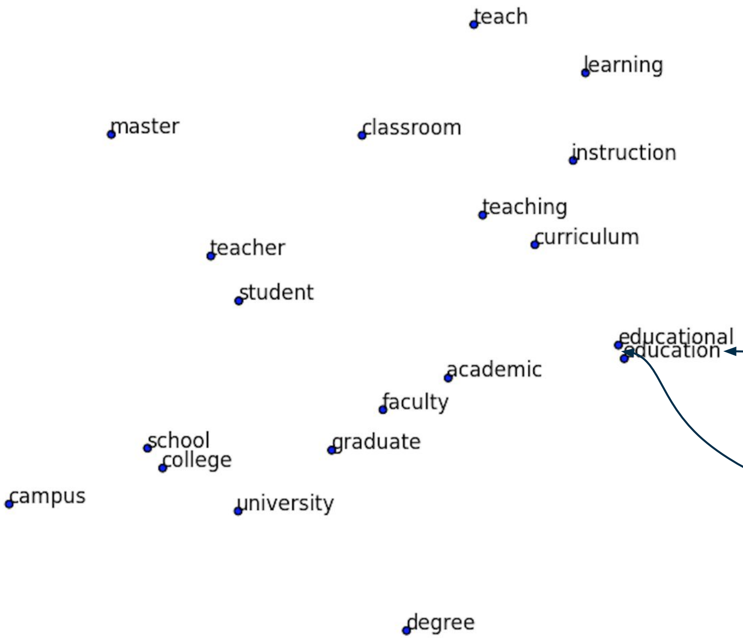
Word Embedding layer

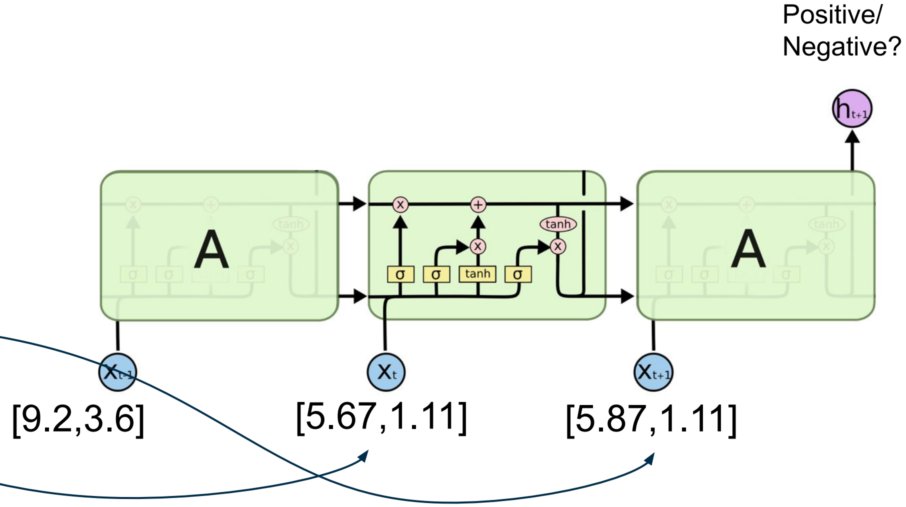
```
#....
model.add(Embedding(...))
#....
```

- 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
- It acts as a look-up table
- This table is learned during training



Word Embedding layer





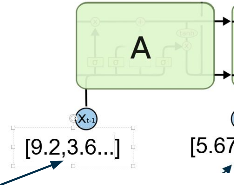
Transform the dataset

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.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 tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.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'])
```