

# Deep Learning

Big Data & Machine Learning Bootcamp - Keep Coding



# Outline

1. Why sequence models?
2. Notation and Recurrent neural network (RNN) model
3. Different types of RNNs
4. Language model and sequence generation
5. Gated Recurrent Unit (GRU)
6. Long Short Term Memory (LSTM)
7. Bidirectional RNN
8. Deep RNNs



# Why sequence models?

## Examples of sequence data

A lot of these problems can be addressed as supervised problems where we have X as inputs and Y as labels.

***BUT, inputs and outputs vary in size!!***

Speech recognition



“The quick brown fox jumped over the lazy dog.”

Music generation



Sentiment classification

“There is nothing to like in this movie.”



DNA sequence analysis

AGCCCCTGTGAGGAACTAG



AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec moi?



Do you want to sing with me?

Video activity recognition



Running

Name entity recognition

Yesterday, Harry Potter met Hermione Granger.



Yesterday, **Harry Potter** met **Hermione Granger**.  
Andrew Ng



Sources:  
- Coursera

© All rights reserved. [www.keepcoding.io](http://www.keepcoding.io)

# Why sequence models?

## Recurrent neural network (RNN) model

Why not using a standard network? Why do we need to bother with yet another architecture?

### Main problems using the standard network:

- Inputs and outputs can be of different lengths in different examples in the training/test set (i.e. phrases can have different lengths)
- The standard network doesn't share features learned across different position of text (This will make more sense later)



Sources:  
- Coursera

# Notation and Recurrent neural network (RNN) model

**Notation: How do we represent words in a sequence?**

One hot representation

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]



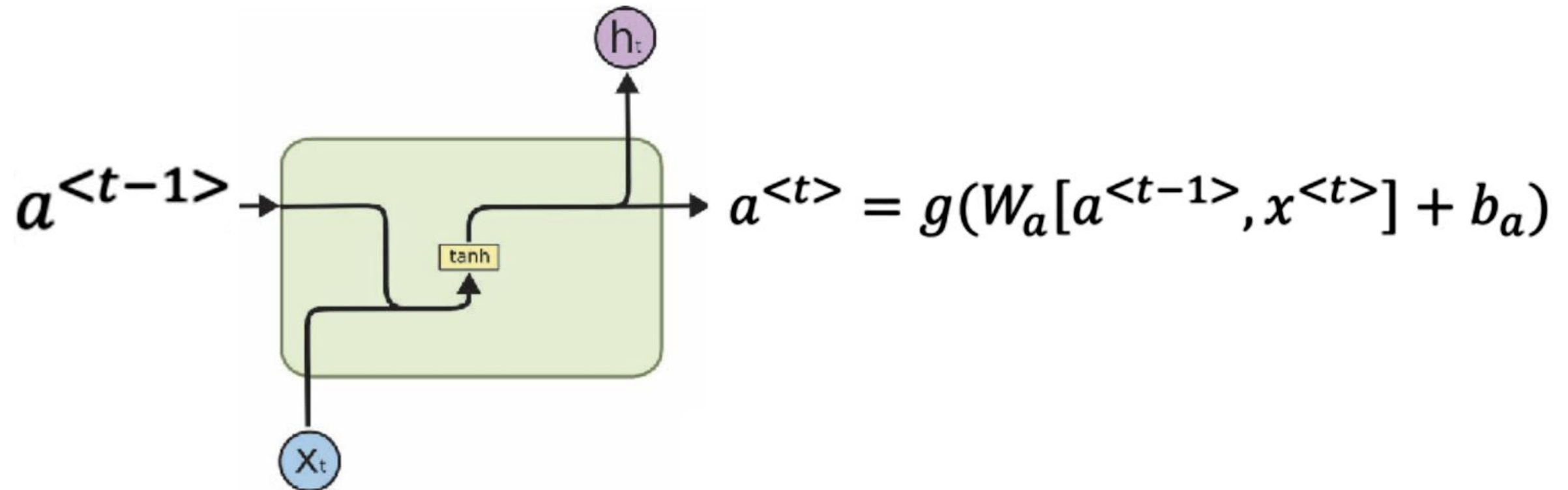
# Basic RNN Unit

## Forward pass equations

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

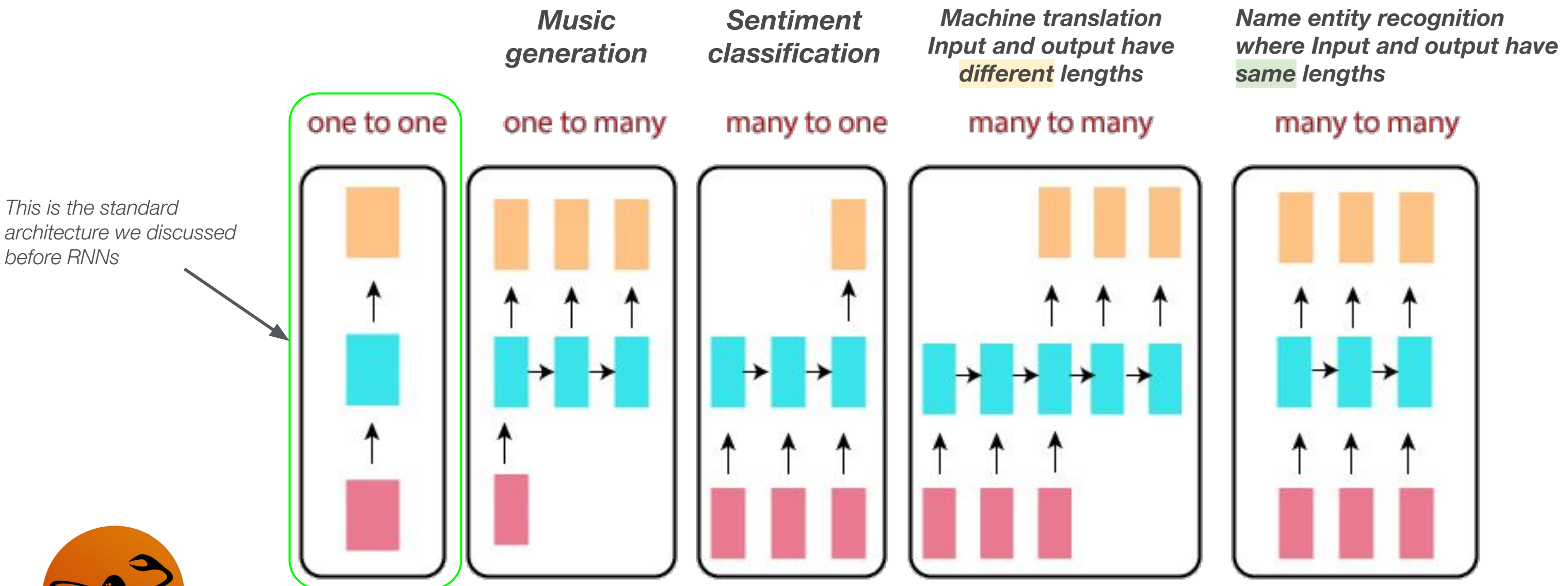
Where  $g$  is the activation function





# Different types of RNNs

Different types of RNN. They depend on the input and output sizes



Sources:  
- Coursera



# Language model and sequence generation

## Steps to create a language model

- **Training set:** Large corpus of english text (or any other language)  
Corpus = A very large set of English sentences
- **Tokenize the phrases.** We represent the phrase in **one hot encoding**.

*We can also have one hot encoding for **<EOS>** End of Sentence*

Each word will be represented by a vector of 9999 zeros and 1 in the position that word occupies in the dictionary. *Supposing we have a dictionary of 10.000 words*



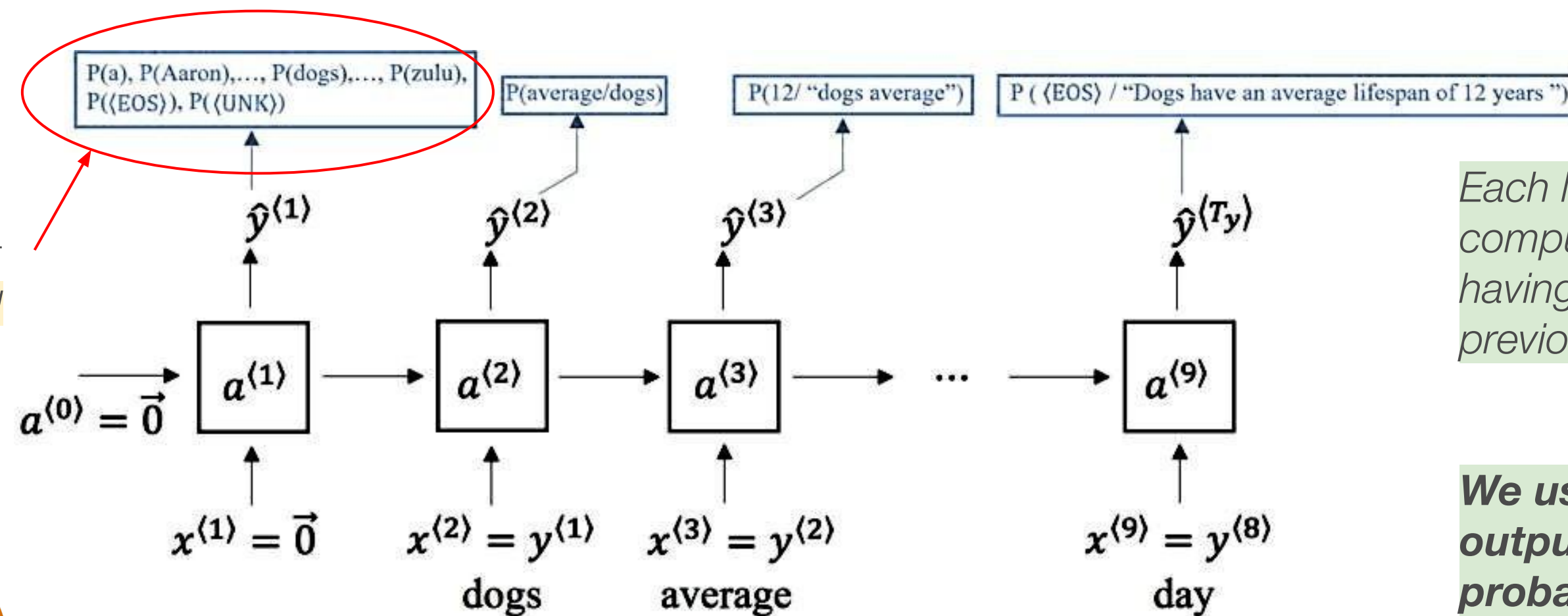
Sources:  
- Coursera



# Language model and sequence generation

## Training the RNN

Let's say we have in our Corpus the phrase: **Dogs have an average lifespan of 12 years**



The first one will compute probability of the first word

Each layer in the RNN computes the probability of having a word given the previous ones.

We use 10,000 softmax outputs to compute the probability



Sources:

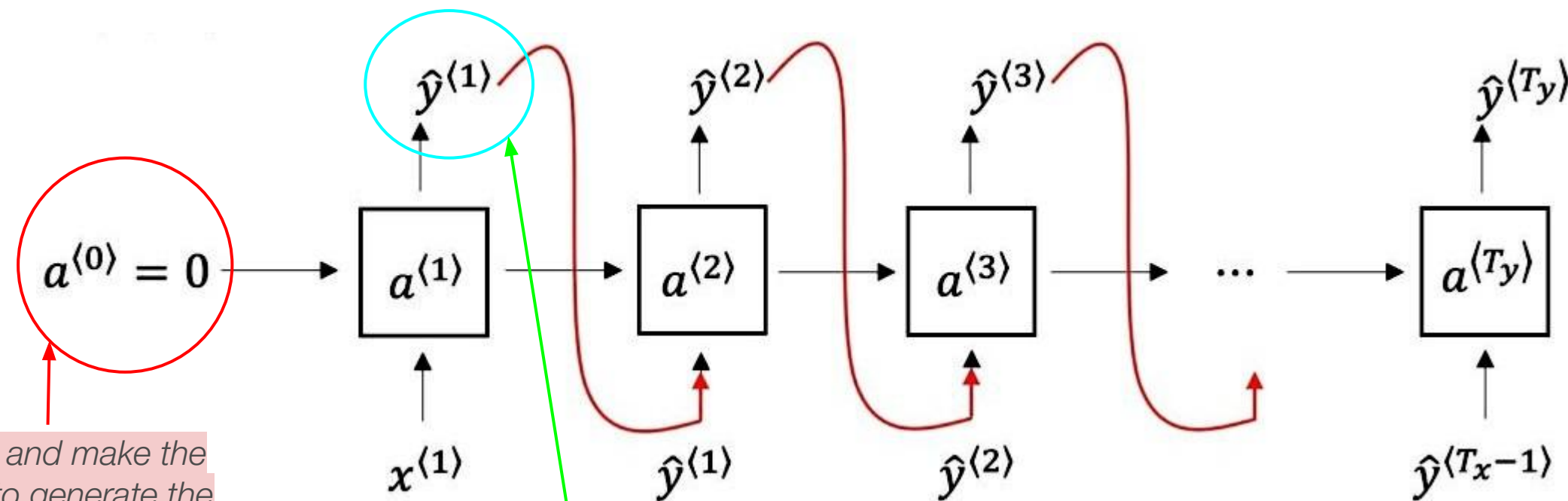
- Coursera

- <https://datahacker.rs/004-rnn-language-modelling-and-sampling-novel-sequences/>

# Language model and sequence generation

Once the language model is trained, you can do sampling.

Sampling is essentially to make the model to generate phrases.



*This process of sampling makes the system generates phrases! Some time they make sense!!*

***You can fix the max length of the phrases or just wait until the system generates the <EOS> token***

*You can also a token for unknown word (<UNK>)*

*Input zero and make the first layer to generate the 10.000 probabilities*

*From the 10.000 probabilities, randomly pick a word and input it to the next layer*



Sources:

- Coursera
- <http://datahacker.rs/004-rnn-language-modelling-and-sampling-novel-sequences/>

# Gated Recurrent Unit (GRU)

## GRU simplified

$$\tilde{c}^{<t>} = \tanh(W_c [c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u [c^{<t-1>}, x^{<t>}] + b_u)$$

For the GRU unit, the combination is a bit more complex. But the main idea is that there is a gate that manages the next activations.

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

That gate will act as a memory when using several GRU units

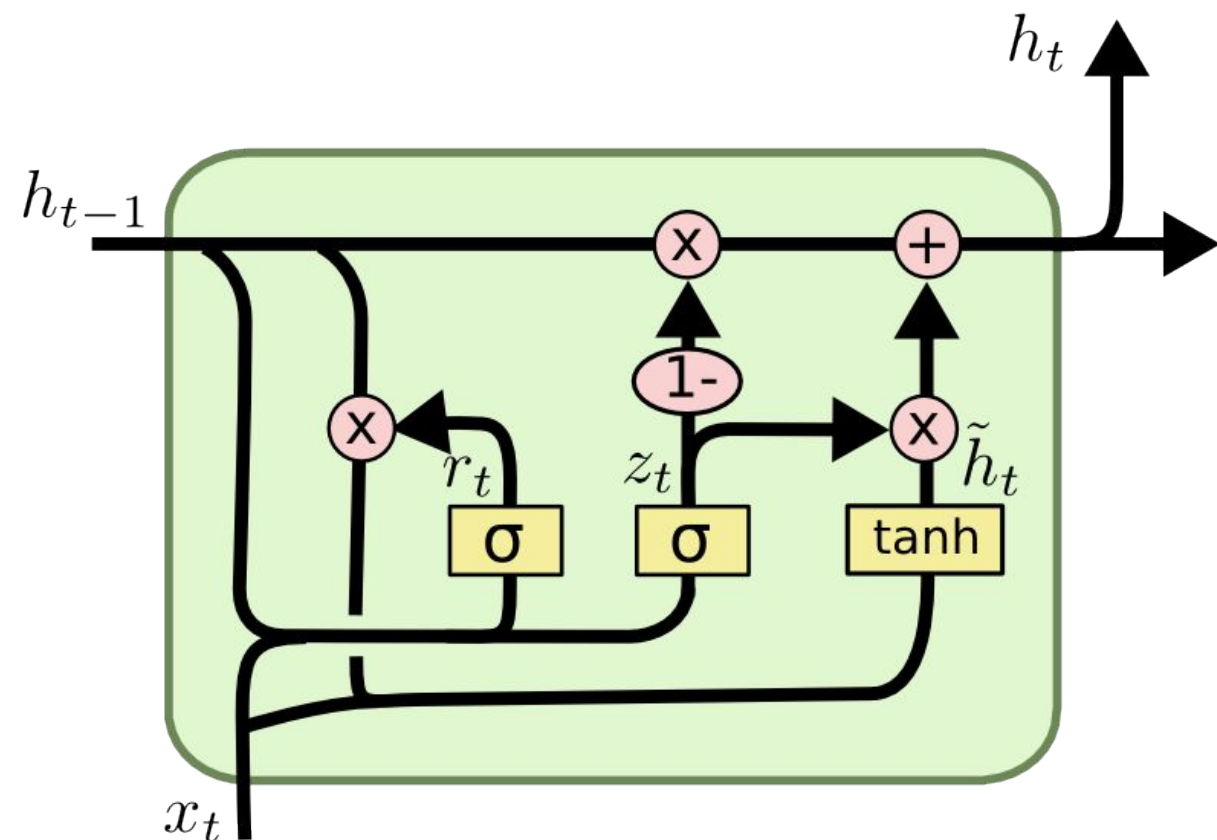
This is the gate driving the output activations  $c$



# Gated Recurrent Unit (GRU)

## Full GRU

You'll agree with me that analysing the simplified version first will help to understand better how the GRU works



**Sigmoid activation function**

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

The full GRU version has actually two gates. BUT still, their function is the same.

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

To control the output activations

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

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

You can imagine how much time researchers spent investigating the best gate combination to then come up with this idea





# Long Short Term Memory (LSTM)

LSTM is a variation of the GRU. It is currently the most used RNN layer!

It was published in 1997

It also performs really well capturing meaning in long phrases and reducing vanishing gradients

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

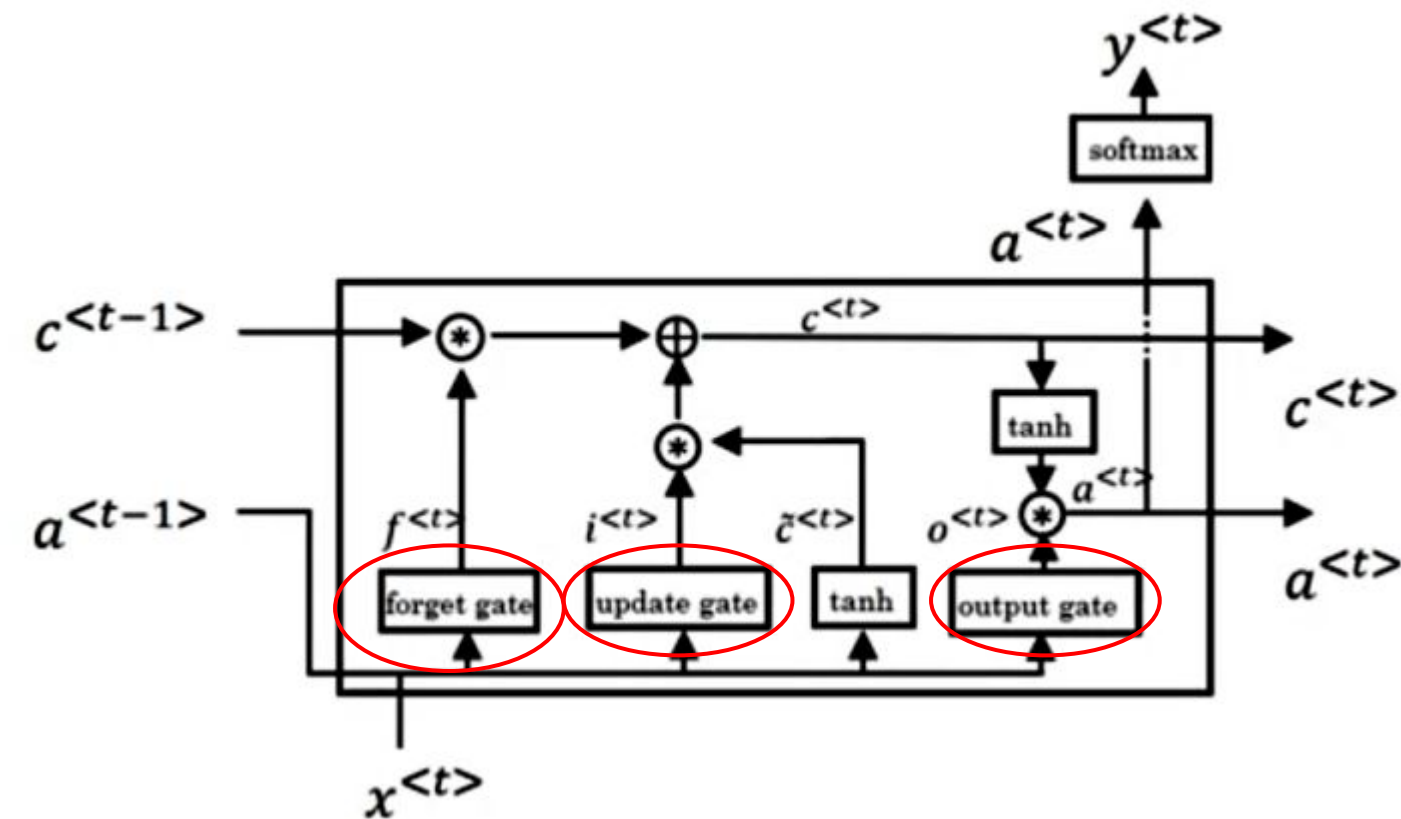
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



Instead of having two, it has **3** gates!!



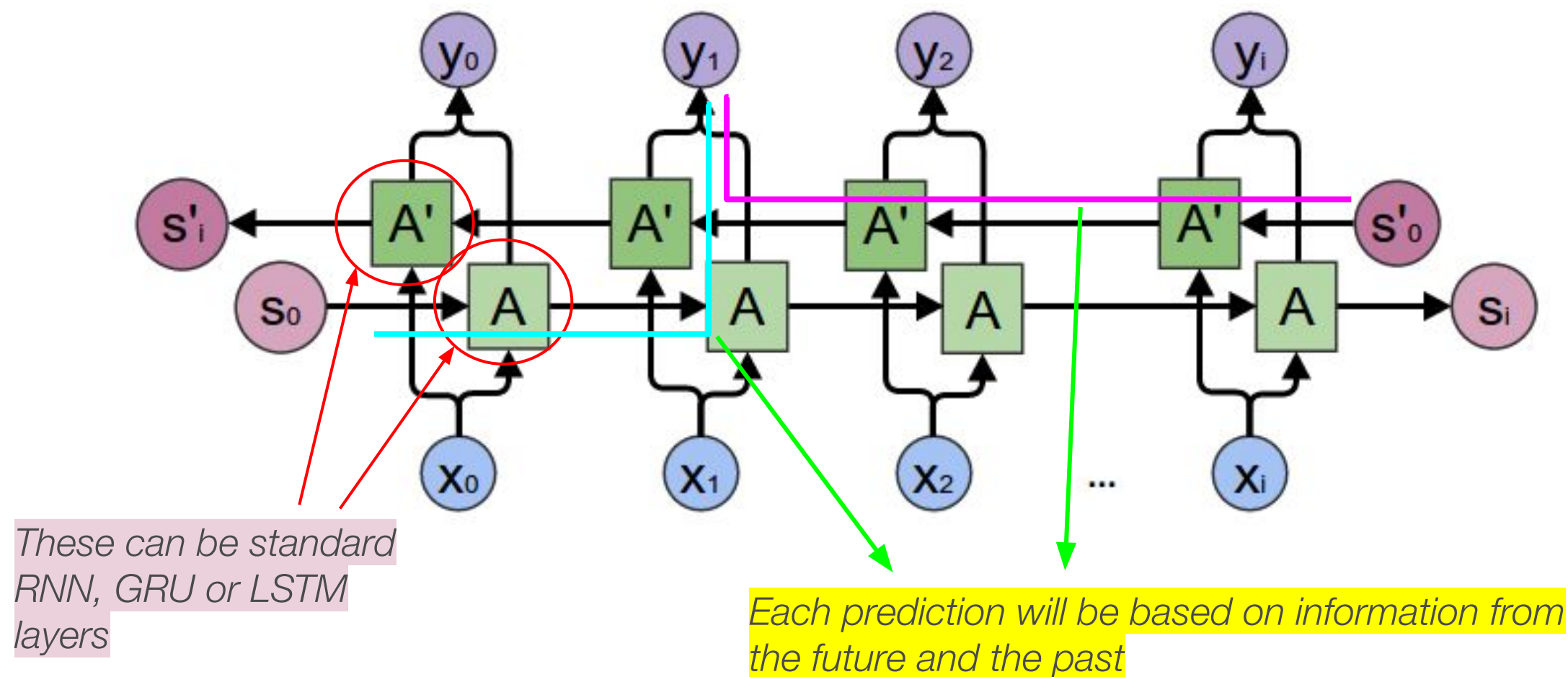
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

Sources:  
- Coursera

# Bidirectional RNN

## Getting information from the future

Bidirectional RNNs are essentially two RNNs working in both directions: From left to right and from right to left



**The main disadvantage of this architecture is that we need the entire sequence to make predictions!**

**For instance, for speech recognition you'll need the person to stop talking to then make predictions**

Still, this architecture is widely used in other tasks where you have access to the entire sequence

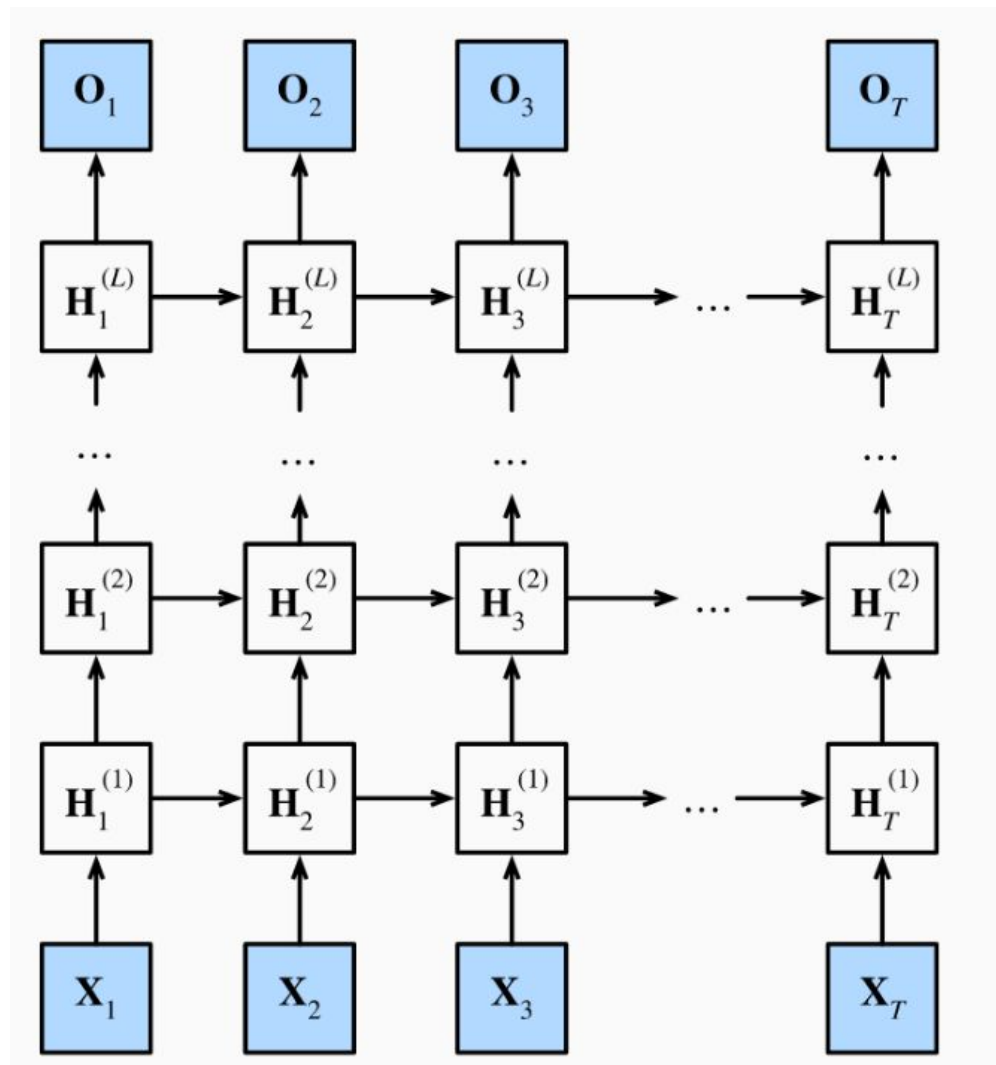




# Deep RNNs

## Deep RNN example

Similar to computer vision tasks, we can also have multiple RNN layers!



*As there is the time dimension, having three RNN layers is already quite a lot*

