Deep Learning

Big Data & Machine Learning Bootcamp - Keep Coding

Outline

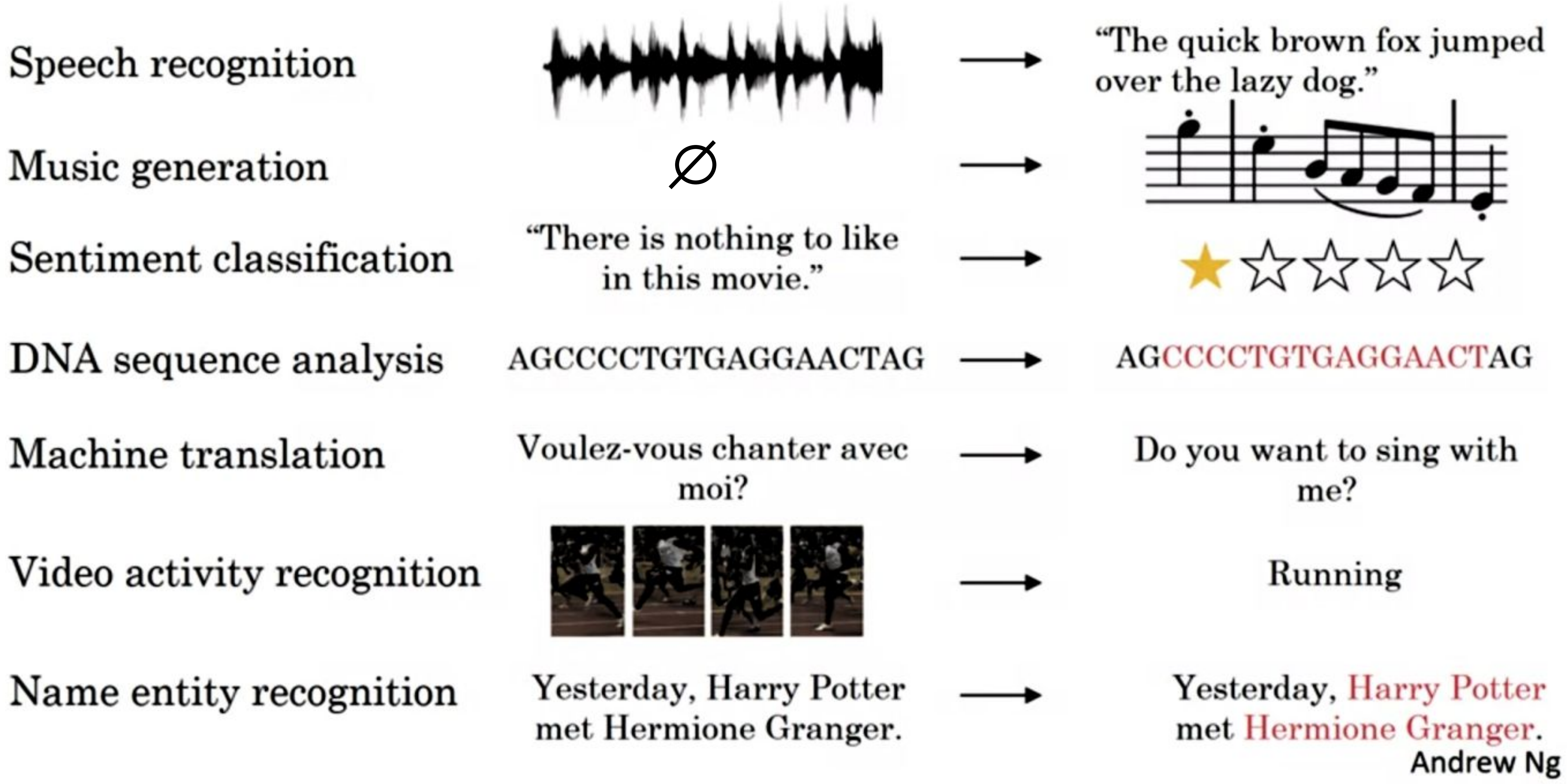
1. Why sequence models?
2. Notation and Recurrent neural network (RNN) model
3. Different types of RNNs
4. Language model and sequence generation
5. Gated Recurrent Unit (GRU)
6. Long Short Term Memory (LSTM)
7. Bidirectional RNN
8. Deep RNNs

1. 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!!



Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

Notation: How do we represent words in a sequence?

Let's assume we have a vocabulary with around 10.000 words (some commercial systems have vocabulary of 30.000 or even 100.000 words)

$$\textbf{Vocabulary} = \left.\begin{bmatrix} \text{A} \\ \text{Aaron} \\ \text{And} \\ . \\ . \\ . \\ \text{Harry} \\ . \\ \text{Potter} \\ \text{Zulu} \end{bmatrix}\right\} \text{10.000 words}$$

This vocabulary is created by looking the most common 10.000 words or all the words that appear in the training set.

And then use one hot representation.

This means, the word A will be a vector of a 1 in the first position and 9999 zeros!

Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

Notation: How do we represent words in a sequence?

One hot representation

Paris
Rome
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]
```


Sources:
- Coursera
- http://hunterheidenreich.com/blog/nlp-count-vectorization/

2. Notation and Recurrent neural network (RNN) model

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)

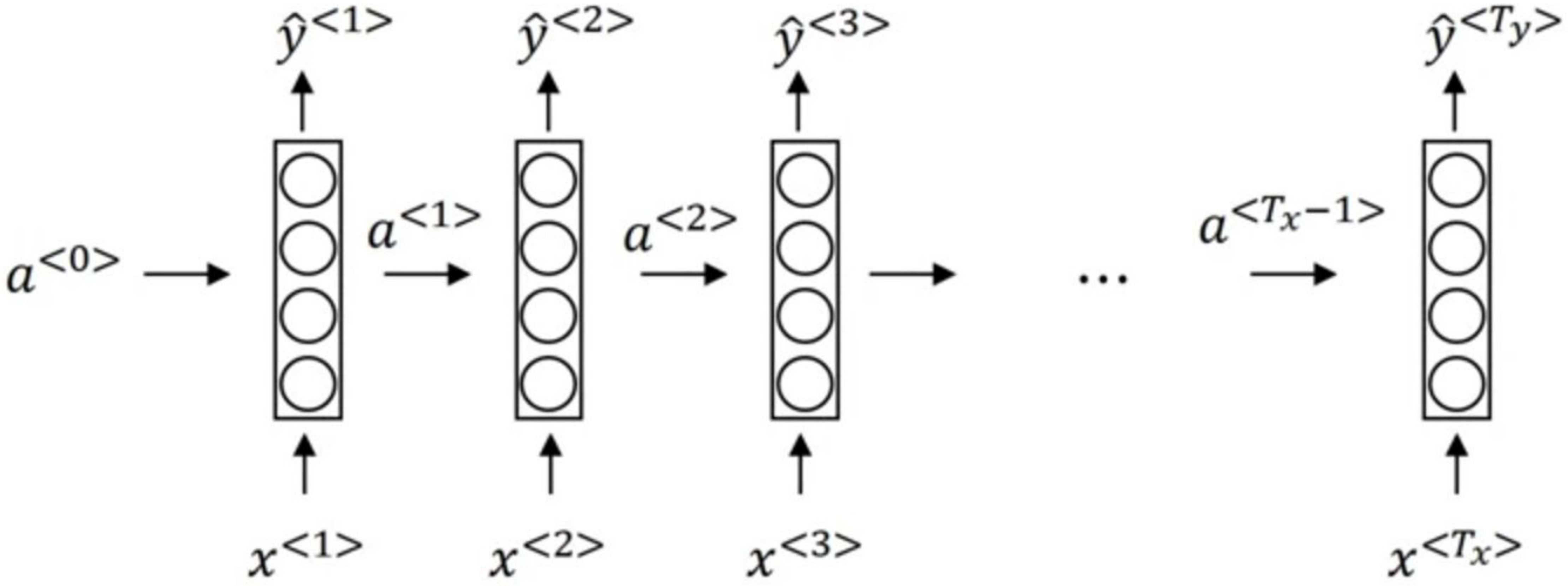
Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

Recurrent neural network (RNN) model architecture

The RNN has one neural network layer for each time step (or word).

Each NN layer not only has the time step/word as input, but also the activations from the previous word.



Sources:
- Coursera

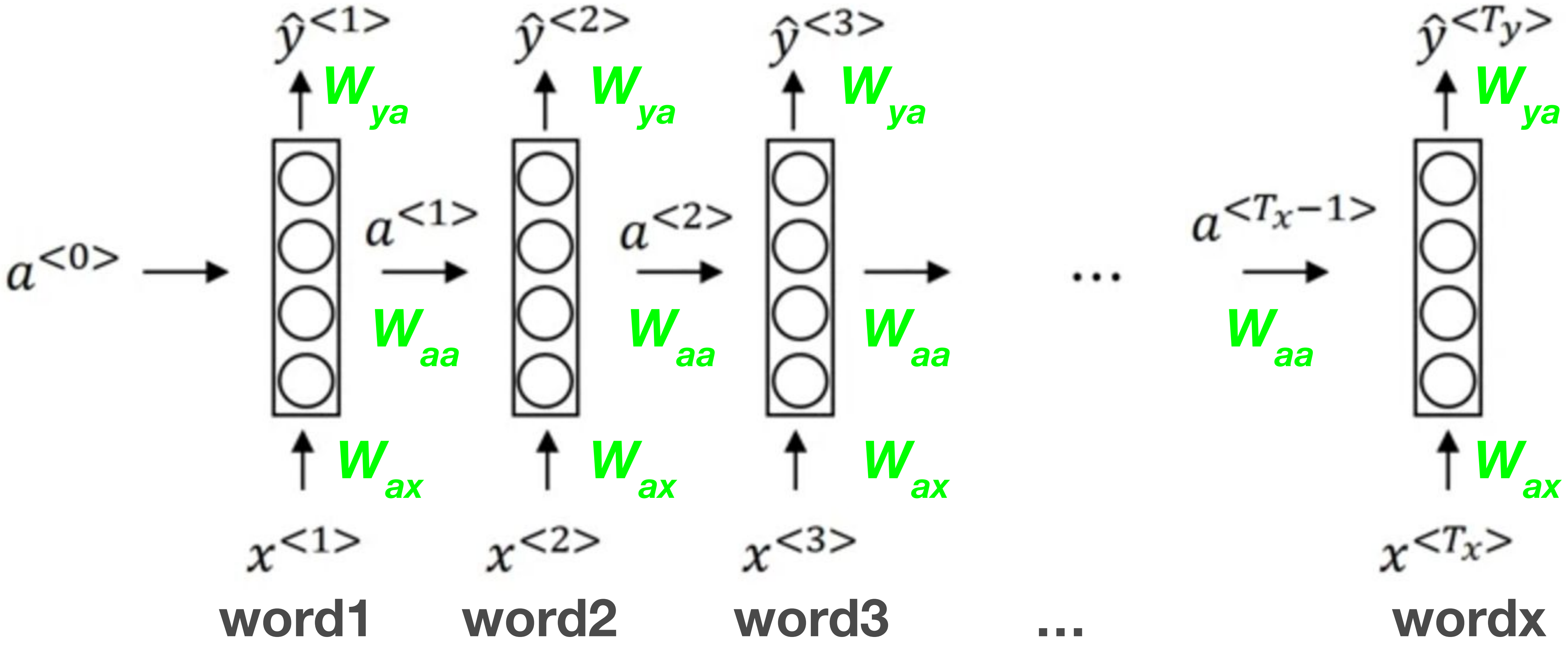
2. Notation and Recurrent neural network (RNN) model

Shared weights

Weights in all *NN layers are shared.*

This means, the number of weights in an RNN are the same as the NN layer.

W_{ax} , W_{aa} *and* W_{ya}

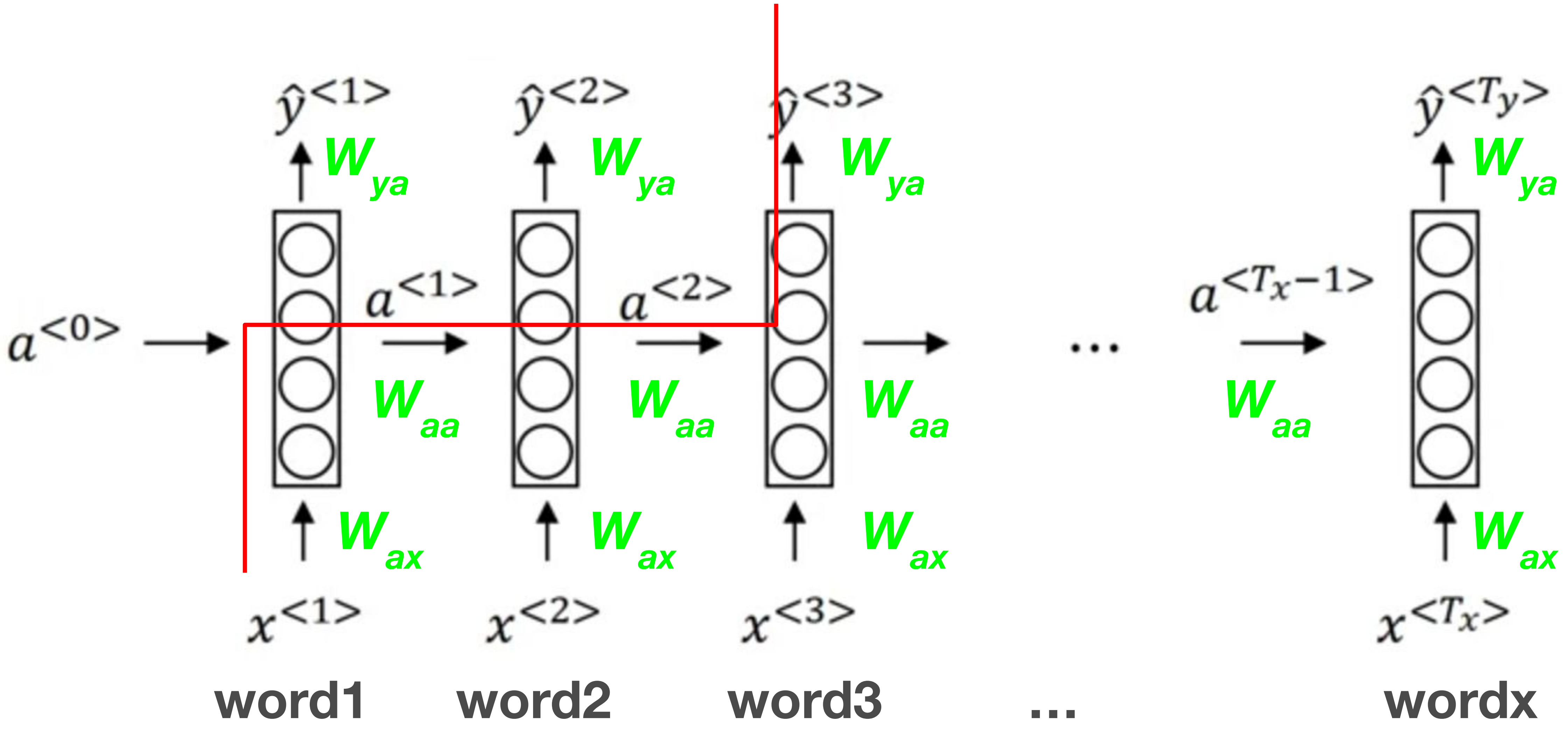


Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

How information flows

Information can flow from previous words but not from future words



Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

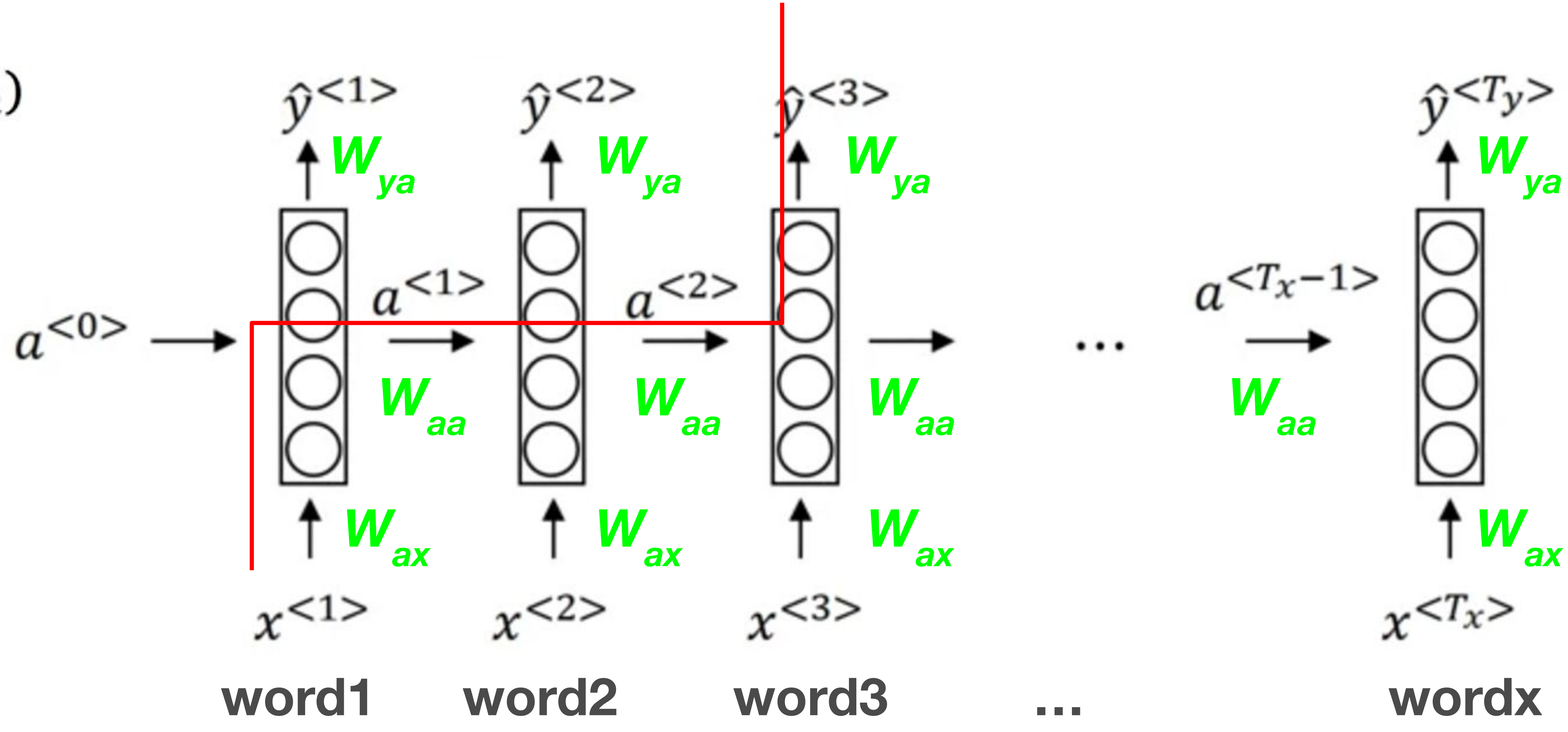
Forward pass equations

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

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

Where g is the activation function

Information can flow from previous words but not from future words



Sources:
- Coursera

2. Notation and Recurrent neural network (RNN) model

Matrix W_a that multiplies both a and x

This version will help us to derive more complex problems later

Forward pass equations: A simplified version

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

$$a^{<t>} = g(\boxed{W_a}[a^{<t-1>}, x^{<t>}] + b_a)$$

$$W_a = [W_{aa} \vdots W_{ax}]$$

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ \hline x^{<t>} \end{bmatrix}$$

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

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

Backward propagation is a bit more complex.

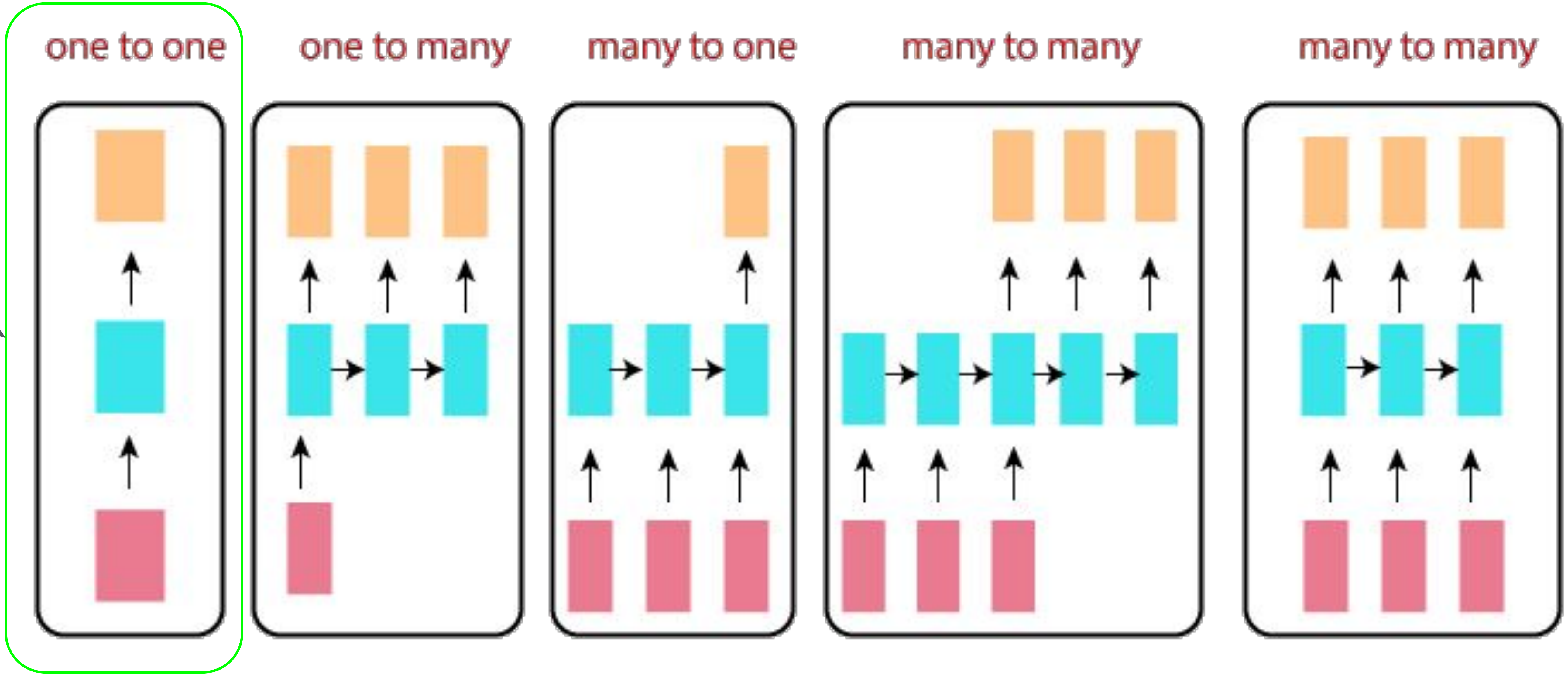
Most programming frameworks will take care of that

Sources:
- Coursera

3. Different types of RNNs

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

This is the standard architecture we discussed before RNNs



Sources:
- Coursera

3. Different types of RNNs

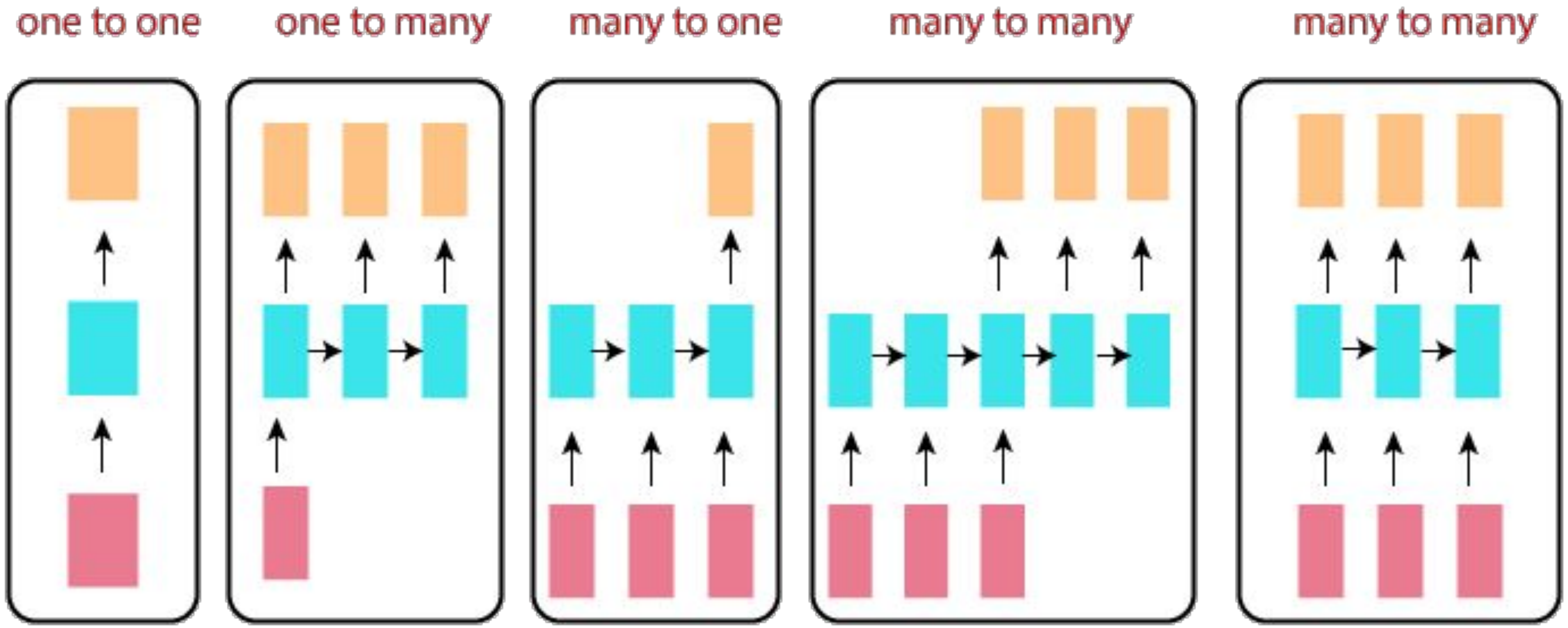
Different types of RNN. They depend on the input and output sizes

Music generation

Sentiment classification

Machine translation Input and output have different lengths

Name entity recognition where Input and output have same lengths



Sources:
- Coursera

4. Language model and sequence generation

What is a language model?

A language model is one the most basic and important tasks in language processing. It is a model that outputs words according to their probability. This means, the most probable word will be the next one in a phrase.

Let’s see how we can build it!

Sources:
- Coursera

4. 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*

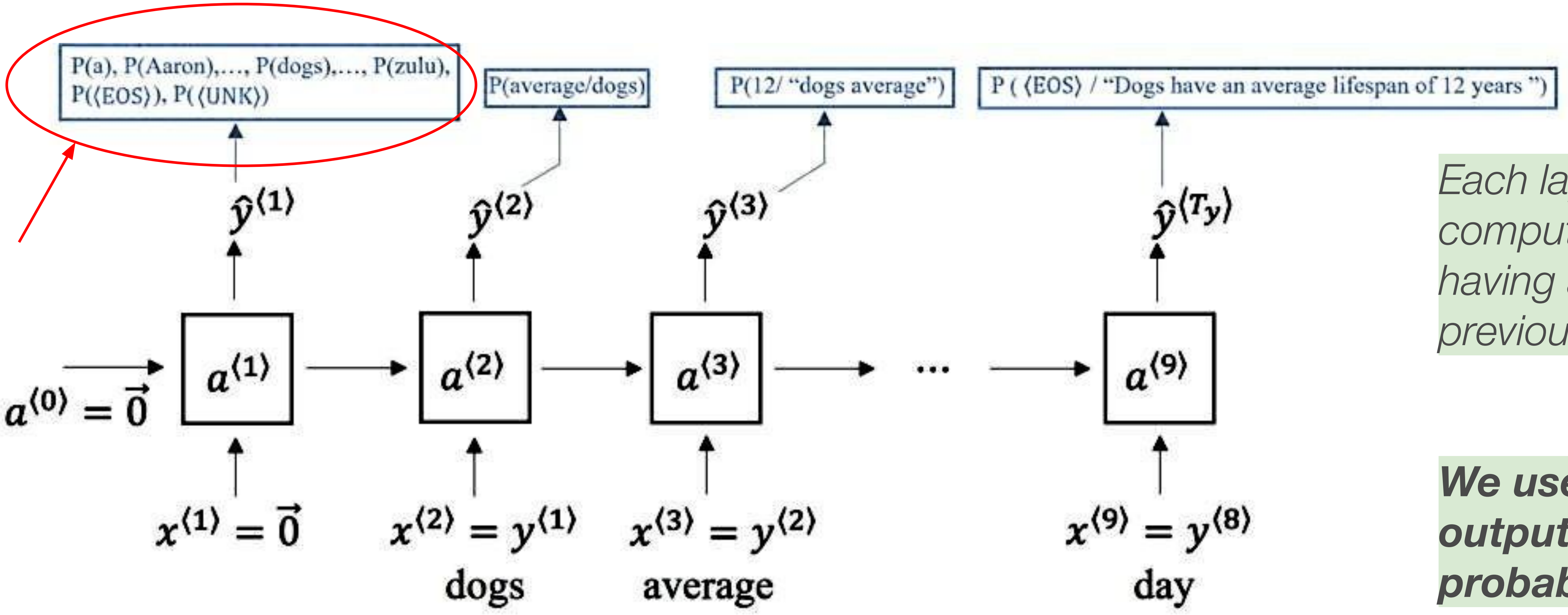
Sources:
- Coursera

4. 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.

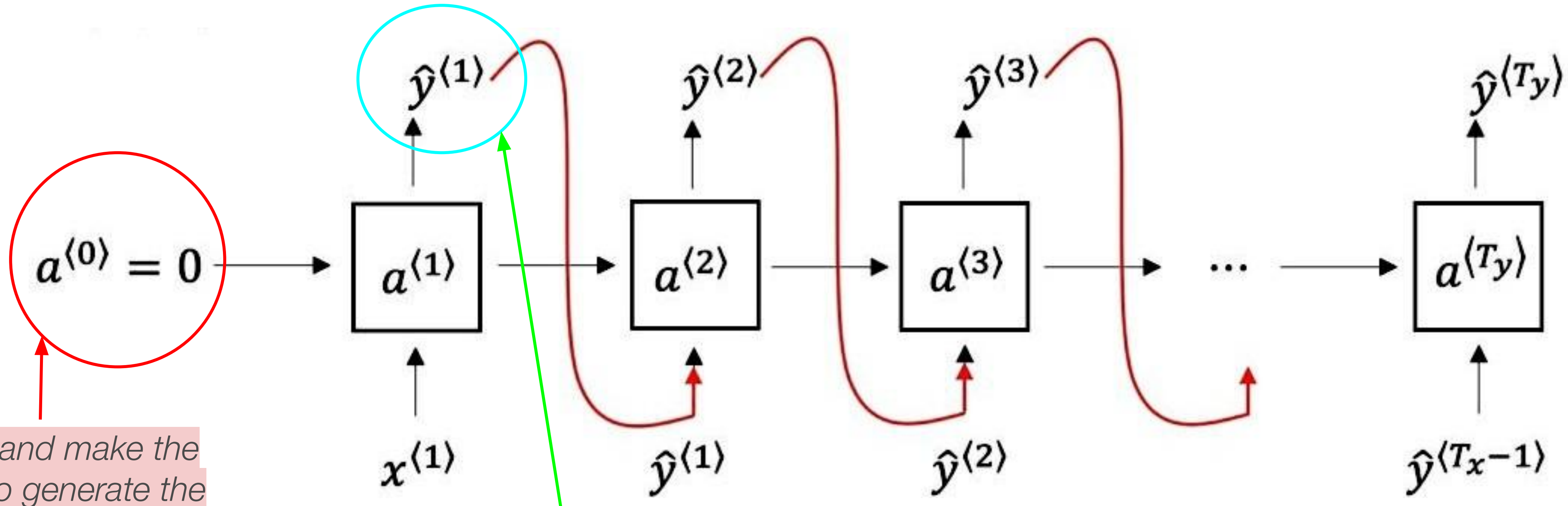
We uses 10.000 softmax outputs to compute the probability

Sources:
- Coursera
- https://datahacker.rs/004-rnn-language-modelling-and-sampling-novel-sequences/

4. 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.



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

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>)

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

4. Language model and sequence generation

Instead of a word-level model, you can also have a **character-level language model.**

This means, instead of generating words, you generate characters.

The vocabulary will be much more smaller. But the system will be much computationally heavy to train.

Character-level language models are still used for some applications, though

Sources:
- Coursera

5. Gated Recurrent Unit (GRU)

Before GRU, let's first comment on vanishing gradients and memory

Vanishing gradients (very small gradients/derivatives) is a problem when using standard RNNs. Also, when training RNNs for long phrases, they don't perform well and they don't have good "memory"

For instance, they cannot catch plural or singular in these two phrases:

1. The **cat**, which actually ate the fish in the chair, **was** full
2. The **cats,** which actually ate the fish in the chair, **were** full

Sources:
- Coursera

5. Gated Recurrent Unit (GRU)

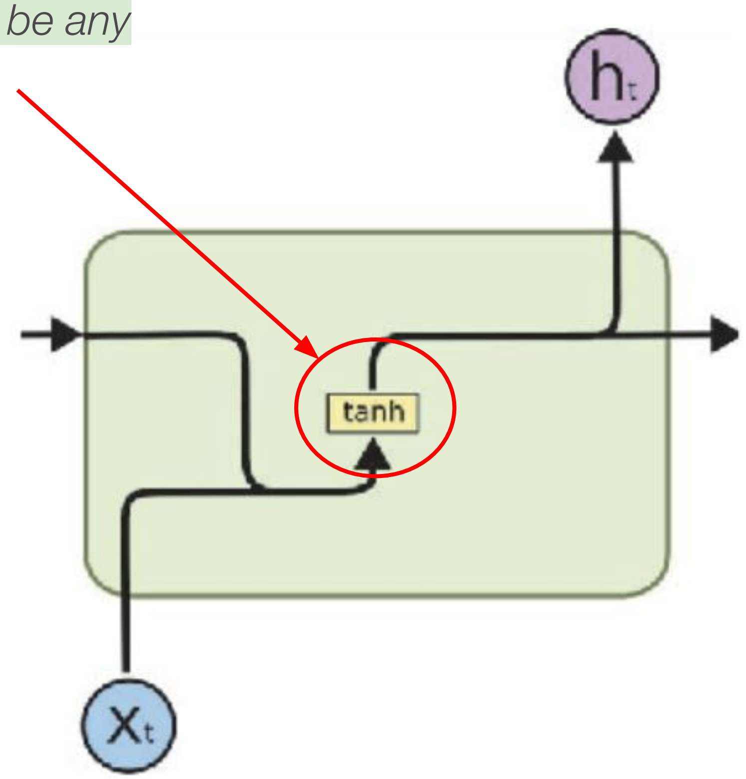
Gated Recurrent Unit is a modification of the RNN layer

It helps a lot to solve the vanishing gradient problem and memory.

This is another way of represinting the RRN unit we've talked so far

This is g in the equation. It can be any activation function

$a^{<t-1>}$



$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

Sources:
- Coursera

5. Gated Recurrent Unit (GRU)

GRU simplified

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

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

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

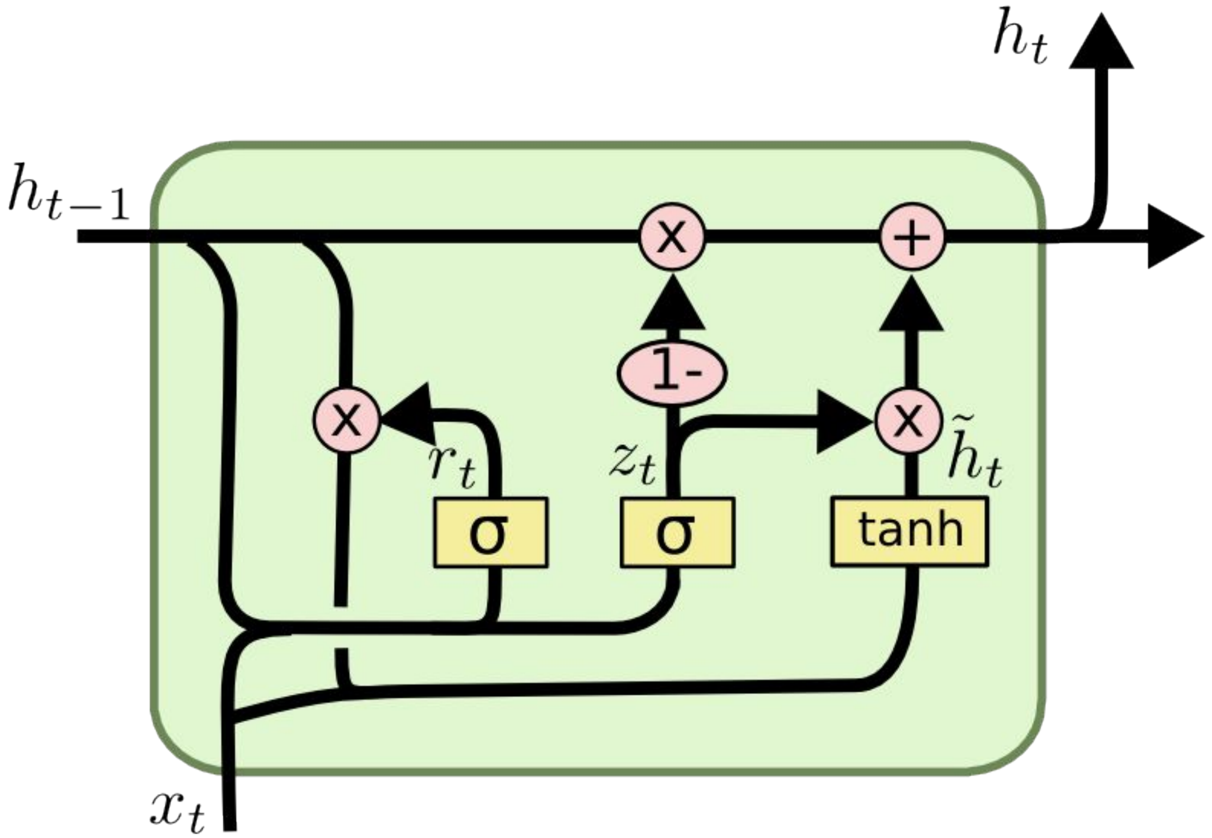
This is the gate driving the output activations ***c***

Sources:
- Coursera

5. 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

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

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

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

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

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

To control the output activations

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

Sources:
- Coursera
- https://stats.stackexchange.com/questions/241985/understanding-lstm-units-vs-cells

6. 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

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

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

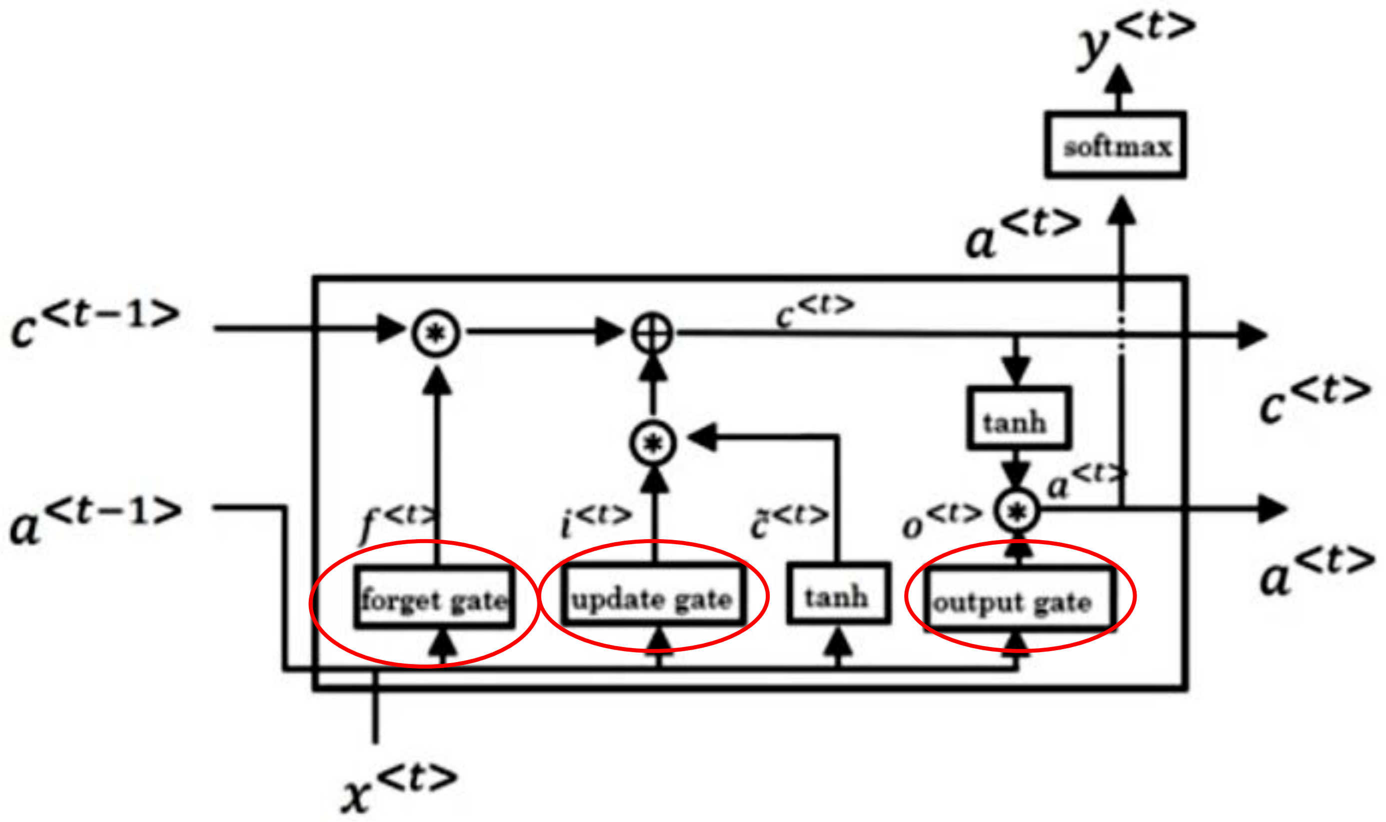
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

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

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

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

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



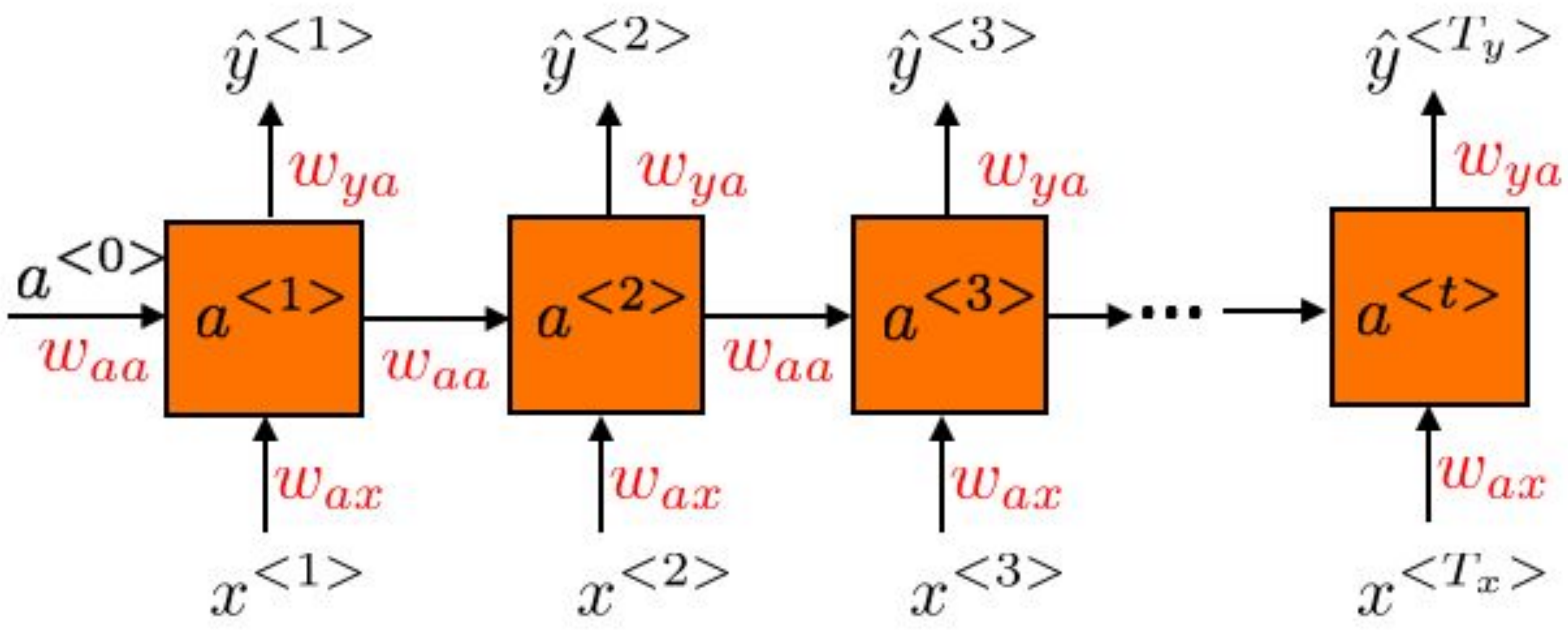
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

Sources:
- Coursera

6. Long Short Term Memory (LSTM)

Key messages to take away.

- Researchers spent a huge amount of time investigating the best gate combination
- GRU and LSTM are very good memorizing information.
- Although LSTM are more used than GRU, their performance is similar

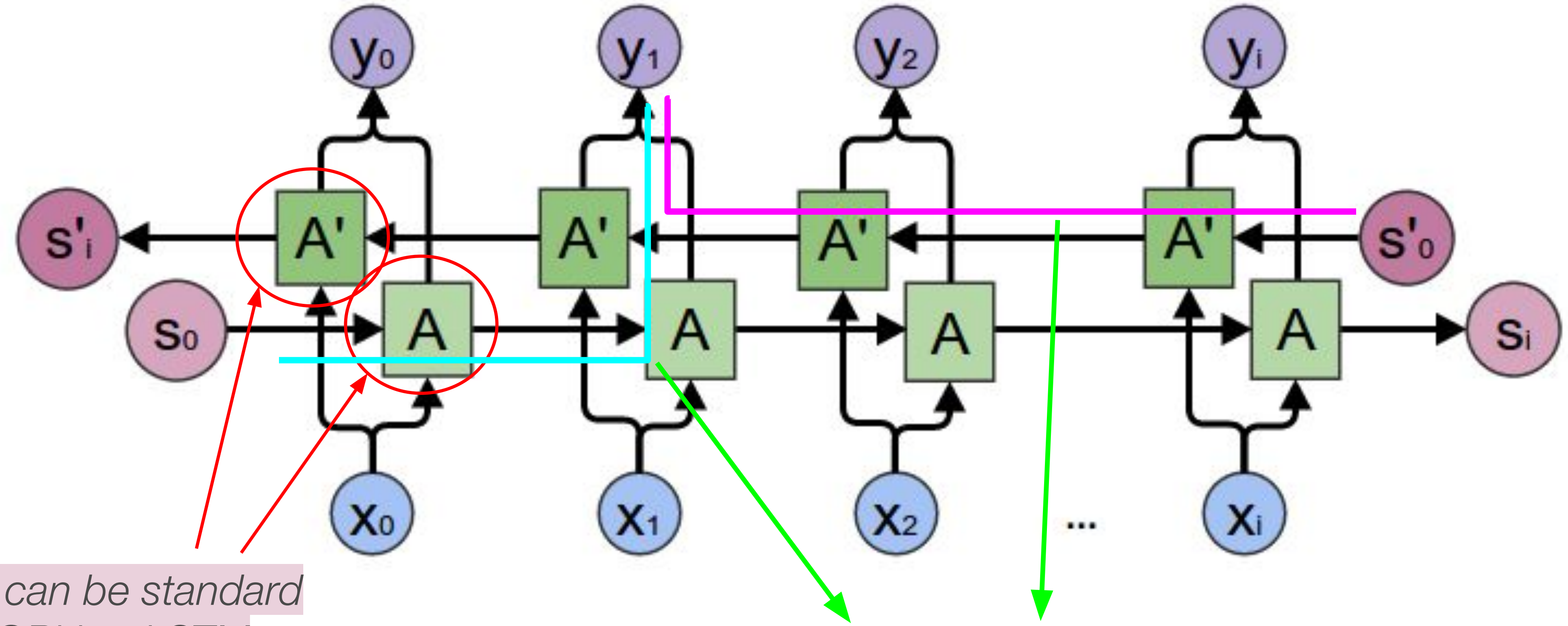


Sources:
- Coursera
- https://anhreynolds.com/blogs/rnn.html

7. 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



These can be standard RNN, GRU or LSTM layers

Each prediction will be based on information from the future and the past

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 :D

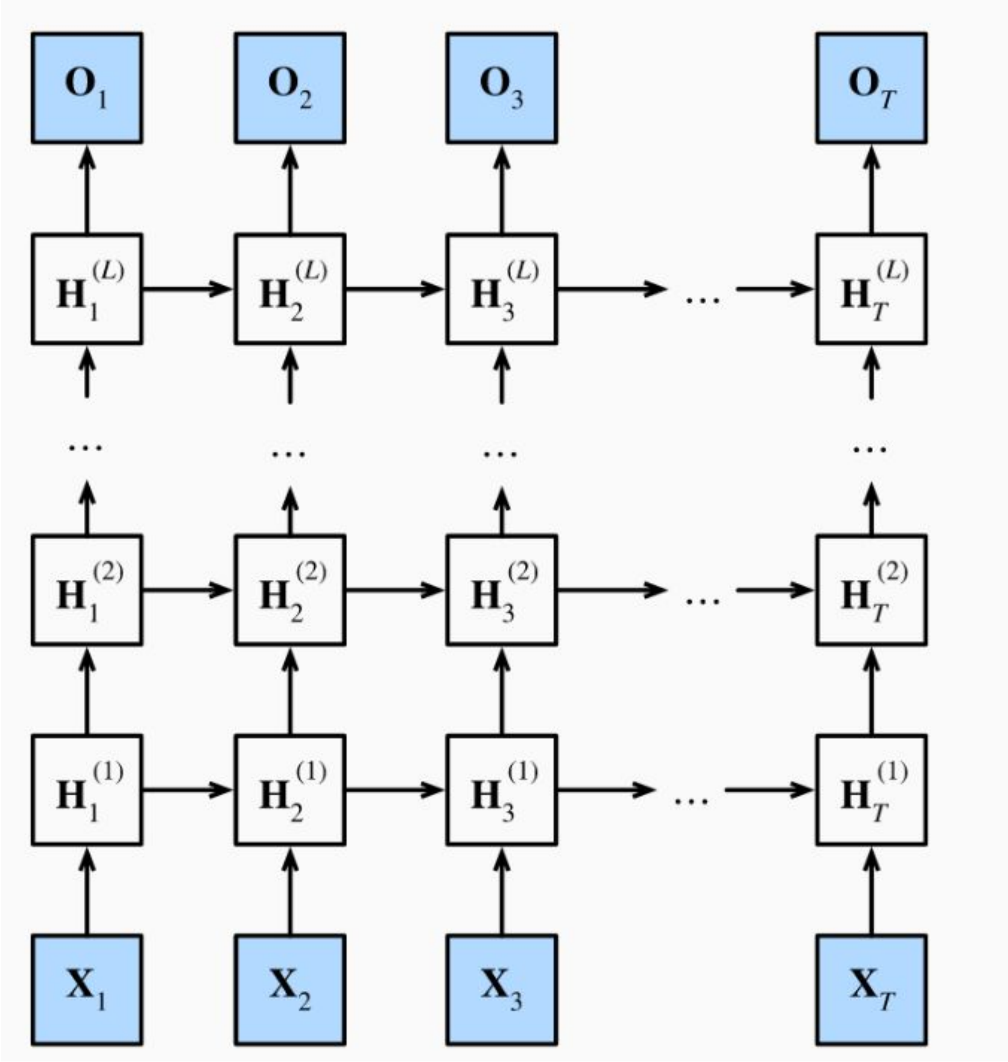
Still, this architecture is widely used in other tasks where you have access to the entire sequence

Sources:
- Coursera
- https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66

8. 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

Sources:
- Coursera
- https://d2l.ai/chapter_recurrent-modern/deep-rnn.html

Let’s move to Google Colab!

Notebooks:

- *10_Character_level_language_model.ipynb*
- *11_Improvise_Jazz_Solo_LSTM_Network.ipynb*

Sources:
- Coursera