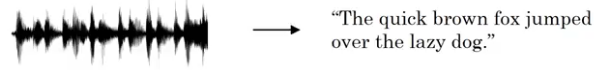
**Sequence Models**

***Speech recognition:***

Quite a common application these days (everyone with a smartphone will know about this). Here, the input is an audio clip and the model have to produce the text transcript. The audio is considered a sequence as it plays over time. Also, the transcript is a sequence of words.



***Sentiment Classification:***

Another popular application of sequence models. We pass a text sentence as input and the model has to predict the sentiment of the sentence (positive, negative, angry, elated, etc.). The output can also be in the form of ratings or stars.

https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/01/Screenshot-from-2019-01-15-16-56-38.png  
***DNA sequence analysis:***

Given a DNA sequence as input, we want our model to predict which part of the DNA belongs to which protein.

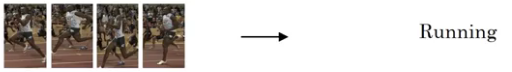
https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/01/Screenshot-from-2019-01-15-17-02-02.png

***Machine Translation:***

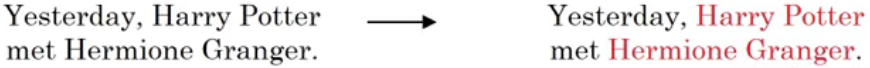
We input a sentence in one language, say French, and we want our model to convert it into another language, say English. Here, both the input and the output are sequences:

https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/01/Screenshot-from-2019-01-15-17-11-23.png  
***Video activity recognition:***

This is actually a very upcoming (and current trending) use of sequence models. The model predicts what activity is going on in a given video. Here, the input is a sequence of frames.

  
***Name entity recognition:***

Definitely my favorite sequence model use case. As shown below, we pass a sentence as input and want our model to identify the people in that sentence:



**Notation**

* In this section we will discuss the notations that we will use.
* **Motivating example**:
  + Named entity recognition example:
    - X: "Harry Potter and Hermoine Granger invented a new spell."
    - Y: 1 1 0 1 1 0 0 0 0
    - Both elements has a shape of 9. 1 means its a name, while 0 means its not a name.
* We will index the first element of x by x<1>, the second x<2> and so on.
  + x<1> = Harry
  + x<2> = Potter
* Similarly, we will index the first element of y by y<1>, the second y<2> and so on.
  + y<1> = 1
  + y<2> = 1
* Tx is the size of the input sequence and Ty is the size of the output sequence.
  + Tx = Ty = 9 in the last example although they can be different in other problems.
* x(i)<t> is the element t of the sequence of input vector i. Similarly y(i)<t> means the t-th element in the output sequence of the i training example.
* Tx(i) the input sequence length for training example i. It can be different across the examples. Similarly for Ty(i) will be the length of the output sequence in the i-th training example.

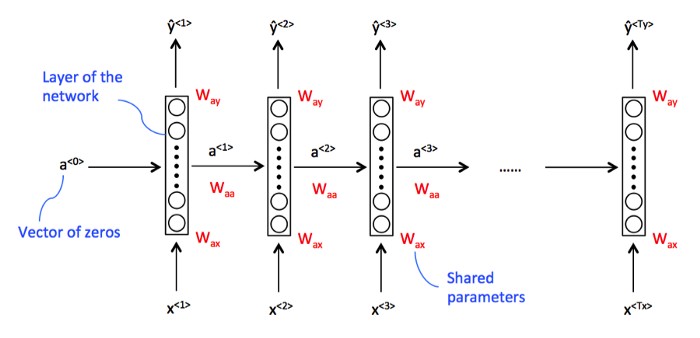
**Lesson 1: Why not a standard network?**

Traditional feedforward neural networks do not share features across different positions of the network. In other words, these models assume that all inputs (and outputs) are independent of each other. This model would not work in sequence prediction since the previous inputs are inherently important in predicting the next output. For example, if you were predicting the next word in a stream of text, you would want to know at least a couple of words before the target word.

Inputs, outputs can be different lengths in different examples.

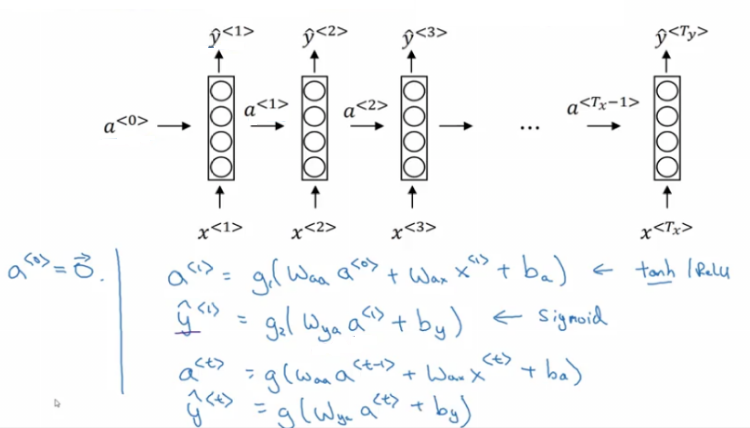
* This can be solved for normal NNs by paddings with the maximum lengths but it's not a good solution.

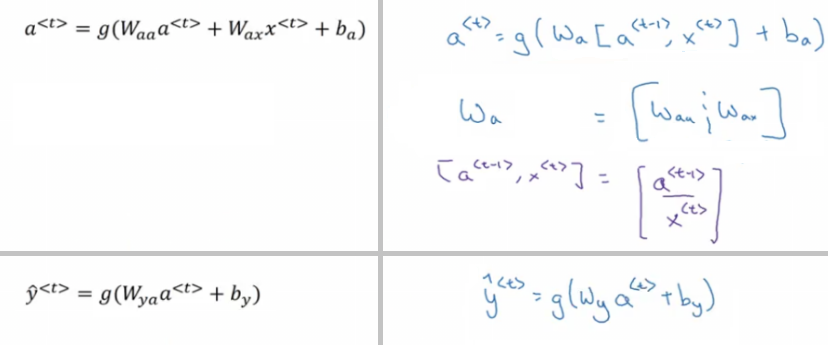
**RNN**



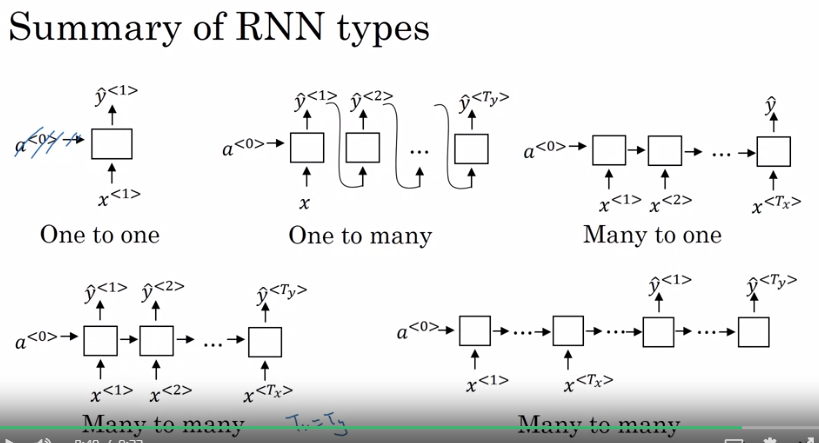
* In this problem Tx = Ty. In other problems where they aren't equal, the RNN architecture may be different.
* a<0> is usually initialized with zeros, but some others may initialize it randomly in some cases.
* There are three weight matrices here: Wax, Waa, and Wya with shapes:
  + Wax: (NoOfHiddenNeurons, nx)
  + Waa: (NoOfHiddenNeurons, NoOfHiddenNeurons)
  + Wya: (ny, NoOfHiddenNeurons)
* The weight matrix Waa is the memory the RNN is trying to maintain from the previous layers.

Now let's discuss the forward propagation equations on the discussed architecture:

[](https://github.com/mbadry1/DeepLearning.ai-Summary/blob/master/5-%20Sequence%20Models/Images/04.png)

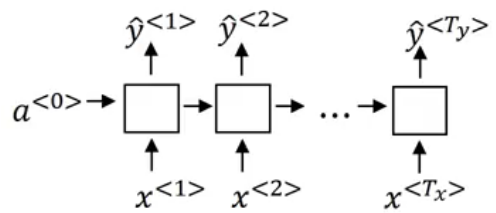
* + The activation function of a is usually tanh or ReLU and for y depends on your task choosing some activation functions like sigmoid and softmax. In name entity recognition task we will use sigmoid because we only have two classes.
* In order to help us develop complex RNN architectures, the last equations needs to be simplified a bit.
* **Simplified RNN notation**:  
  [](https://github.com/mbadry1/DeepLearning.ai-Summary/blob/master/5-%20Sequence%20Models/Images/05.png)
  + wa is waa and wax stacked horizontally.
  + [a<t-1>, x<t>] is a<t-1> and x<t> stacked vertically.
  + wa shape: (NoOfHiddenNeurons, NoOfHiddenNeurons + nx)
  + [a<t-1>, x<t>] shape: (NoOfHiddenNeurons + nx, 1)

**Types of RNN**



***Many-to-many***

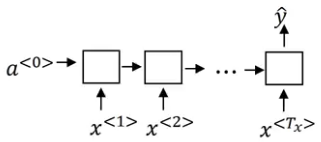
The name entity recognition examples we saw earlier fall under this category. We have a sequence of words, and for each word, we have to predict whether it is a name or not. The RNN architecture for such a problem looks like this:



For every input word, we predict a corresponding output word.

***Many-to-one***

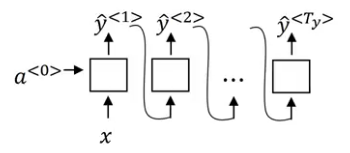
Consider the **sentiment classification** problem. We pass a sentence to the model and it returns the sentiment or rating corresponding to that sentence. This is a many-to-one problem where the input sequence can have varied length, whereas there will only be a single output. The RNN architecture for such problems will look something like this:



Here, we get a single output at the end of the sentence.

***One-to-many***

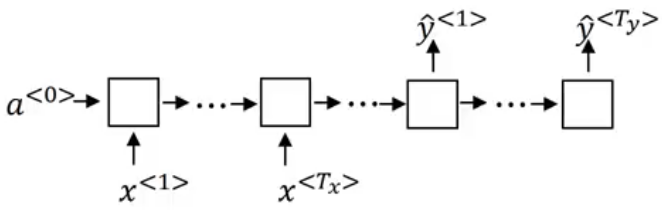
Consider the example of **music generation** where we want to predict the lyrics using the music as input. In such scenarios, the input is just a single word (or a single integer), and the output can be of varied length. The RNN architecture for this type of problems looks like the below:



***Many to many***

There is one more type of RNN which is popularly used in the industry. Consider the machine translation application where we take an input sentence in one language and translate it into another language. It is a many-to-many problem but the length of the input sequence might or might not be equal to the length of output sequence.

In such cases, we have an encoder part and a decoder part. The encoder part reads the input sentence and the decoder translates it to the output sentence:

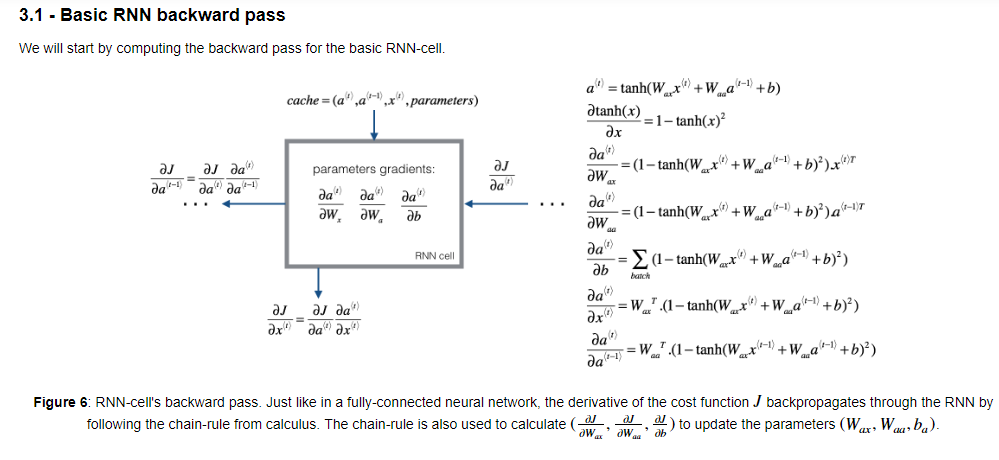


**Advantages of Recurrent Neural Network**

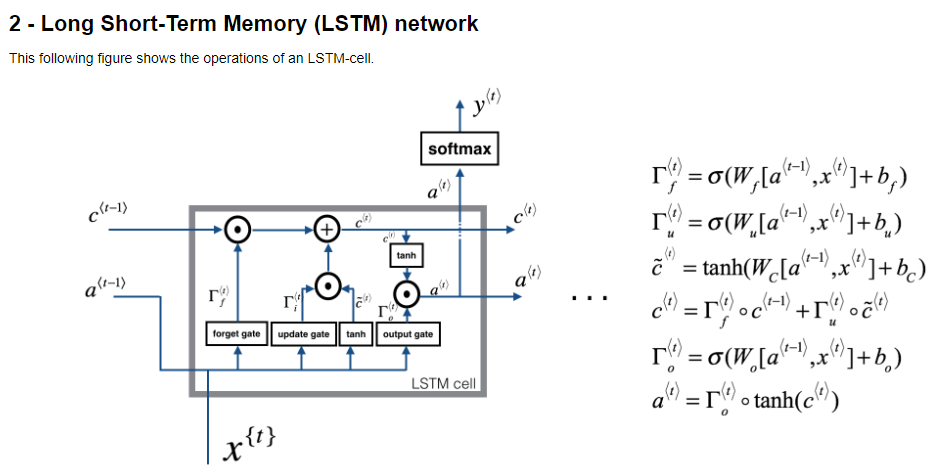
1. An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

**Disadvantages of Recurrent Neural Network**

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.



## **Long Short-Term Memory (LSTM) network**



This following figure shows the operations of an LSTM-cell.

**Figure 4**: LSTM-cell. This tracks and updates a "cell state" or memory variable c⟨t⟩c⟨t⟩ at every time-step, which can be different from a⟨t⟩a⟨t⟩.

Similar to the RNN example above, you will start by implementing the LSTM cell for a single time-step. Then you can iteratively call it from inside a for-loop to have it process an input with TxTx time-steps.

### About the gates

#### **- Forget gate**

For the sake of this illustration, let's assume we are reading words in a piece of text, and want use an LSTM to keep track of grammatical structures, such as whether the subject is singular or plural. If the subject changes from a singular word to a plural word, we need to find a way to get rid of our previously stored memory value of the singular/plural state. In an LSTM, the forget gate let's us do this:



Here, Wf are weights that govern the forget gate's behavior. We concatenate [a⟨t−1⟩,x⟨t⟩] and multiply by Wf. The equation above results in a vector Γ⟨t⟩ with values between 0 and 1. This forget gate vector will be multiplied element-wise by the previous cell state c⟨t−1⟩. So if one of the values of Γ⟨t⟩fΓf⟨t⟩ is 0 (or close to 0) then it means that the LSTM should remove that piece of information (e.g. the singular subject) in the corresponding component of c⟨t−1⟩c⟨t−1⟩. If one of the values is 1, then it will keep the information.

#### **- Update gate**

Once we forget that the subject being discussed is singular, we need to find a way to update it to reflect that the new subject is now plural. Here is the formula for the update gate:



Similar to the forget gate, here Γ⟨t⟩uΓu⟨t⟩ is again a vector of values between 0 and 1. This will be multiplied element-wise with c̃ ⟨t⟩c~⟨t⟩, in order to compute c⟨t⟩c⟨t⟩.

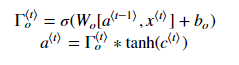
#### **- Updating the cell**

To update the new subject we need to create a new vector of numbers that we can add to our previous cell state. The equation we use is:

#### 

#### **- Output gate**

To decide which outputs we will use, we will use the following two formulas:



Where in equation 5 you decide what to output using a sigmoid function and in equation 6 you multiply that by the tanhtanh of the previous state.