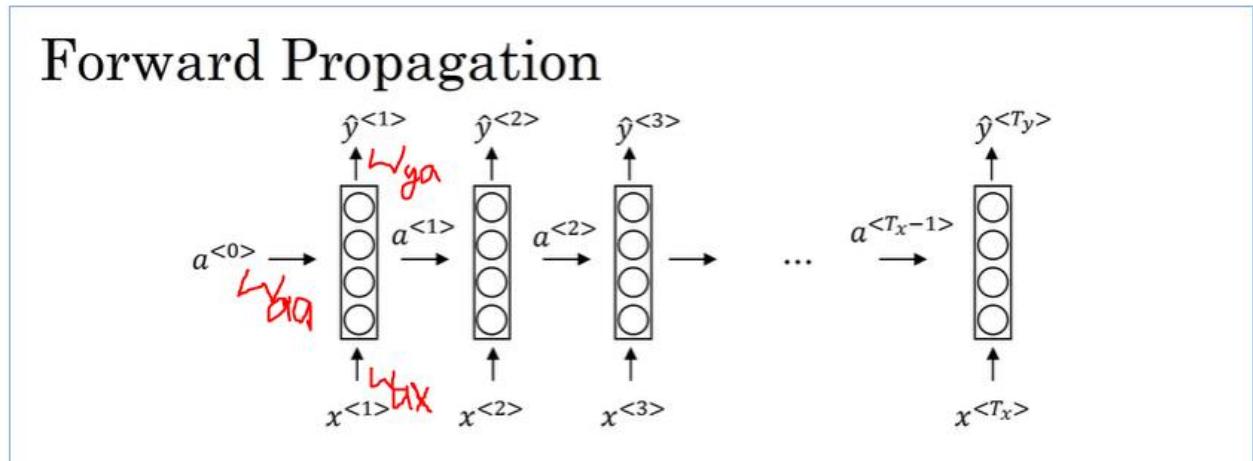


Why not a standard network ?

1. Inputs and outputs can be of different lengths in different example, like music generation, sentiment classification for a given input sequence.
2. Doesn't share features learned previously,  
i.e. for name entity recognition if a word harry is learned as person then in the feature instance if harry appears then we can use previous learning to reduce cost.

RNN model :

Recurrent Neural Network Model



- Every time RNN passes the previous activation solving the problem 2 mentioned above.
- For every time stamp we pass the same weights  $W_{aa}$ ,  $W_{ax}$  for activation and  $W_{ya}$  to calculate the output  $y^{<1>}$  (predicted value).

$a^{<0>}$  is a zero vector.

$$a^{<1>} = g_1(W_{ax}X^{<1>} + W_{aa} a^{<0>} + b_a), y^{<1>}_{pred} = g_2(W_{ya} a^{<1>} + b_y)$$

generalizing for t timestamp we get :

$$a^{<t>} = g_1(W_{ax}X^{<t>} + W_{aa} a^{<t-1>} + b_a)$$

$$y^{<t>}_{pred} = g_2(W_{ya} a^{<t>} + b_y), \text{ Where } g_1 \text{ and } g_2 \text{ are different activation functions}$$

We can also reduce the computation cost by combining the weights and (input, activation), i.e.

$$W_a = (W_{aa} \mid W_{ax})$$

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

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

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

**Drawback** with RNN is that it considers information only from previous steps but not from future,

Ex: In name entity recognition, suppose we have some training examples as below,

1. He said, "Teddy Roosevelt was a great President" .
2. He said, "Teddy bears are on sale".

Then if our model didn't consider feature instance it will incorrectly classify teddy in second case as name which is not .

Q→ Why we are using same weights in RNN?

The accepted answer focuses on the practical side of the question: it would require a lot of resources, if their parameters are not shared. However, the decision to share parameters in an RNN has been made when *any* serious computation was a problem (1980s according to [wiki](#)), so I believe it wasn't the main argument (though still valid).

There are pure theoretical reasons for parameter sharing:

- It helps in applying the model to examples of different lengths. While reading a sequence, if RNN model uses different parameters for each step during training, it won't generalize to unseen sequences of different lengths.
- Oftentimes, the sequences operate according to the same rules across the sequence. For instance, in NLP:

"On Monday it was snowing"

"It was snowing on Monday"

...these two sentences mean the same thing, though the details are in different parts of the sequence. Parameter sharing reflects the fact that we are performing the same task at each step, as a result, we don't have to relearn the rules at each point in the sentence.

LSTM is no different in this sense, hence it uses shared parameters as well.

Answer resource : <https://stats.stackexchange.com/questions/221513/why-are-the-weights-of-rnn-lstm-networks-shared-across-time>

For tasks like object detection( eg: face recognition) each hidden layer of the DNN/CNN will focus on some task, like in first hidden layer it finds vertical edges, latter layers it finds horizontal lines, latter it will find facial features...., where as in task like name entity recognition, the sequence models(like RNN's) focus on same task at every time stamp, i.e. to find whether the word is a person name or not, so because of that reason we will use same set of learning parameters(weights and bias).

**Back Propagation through time :**




For back prop we will require a loss function.

Loss function for a time step  $\langle t \rangle$  is given as

$$L^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -\hat{y}^{\langle t \rangle} \log \hat{y}^{\langle t \rangle} - (1 - \hat{y}^{\langle t \rangle}) \log(1 - \hat{y}^{\langle t \rangle})$$

Over-all loss function is given as,  $L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$ , Where  $T_y = T_x$

**Different Types of RNN :**

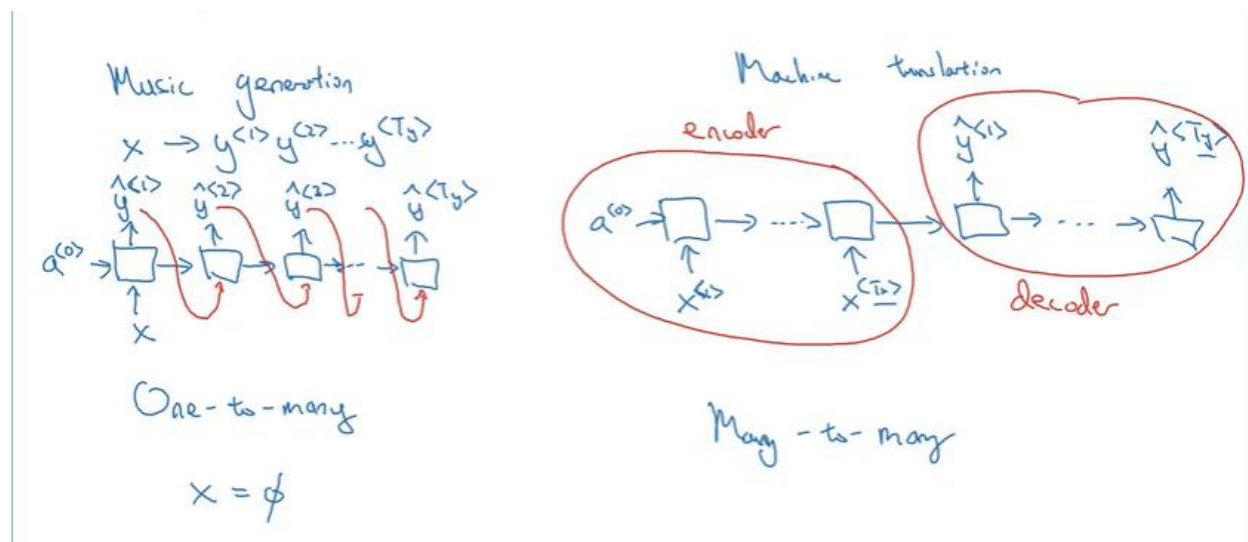
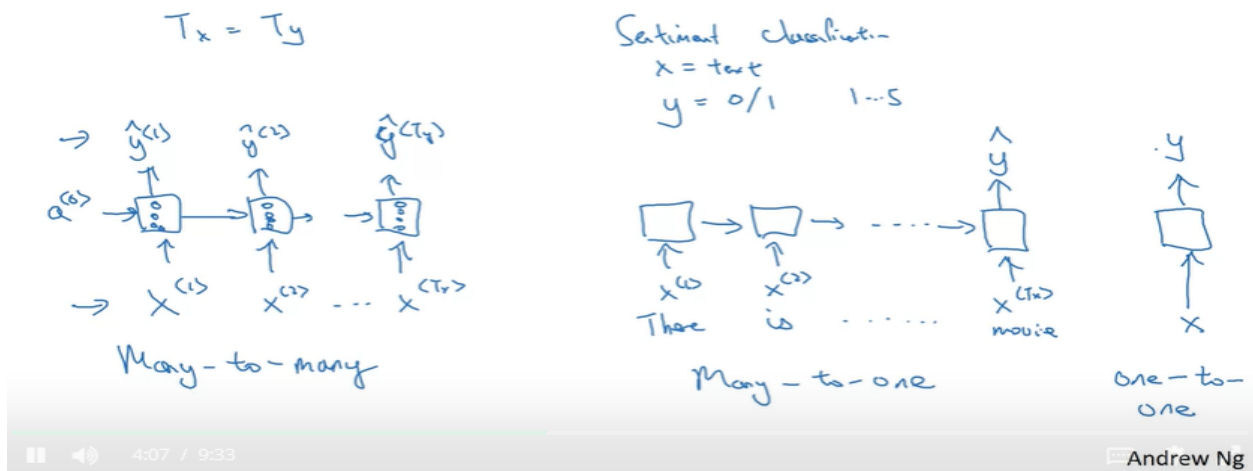
Examples of sequence data		
Speech recognition		→ "The quick brown fox jumped over the lazy dog."
Music generation	$\emptyset$	→ 
Sentiment classification	"There is nothing to like in this movie."	→ ★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→ AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→ Do you want to sing with me?
Video activity recognition		→ Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→ Yesterday, <b>Harry Potter</b> met <b>Hermione Granger</b> .

0:32 / 9:33 Andrew Ng

inspired by a blog post by Andrej Karpathy, titled, The Unreasonable Effectiveness of Recurrent Neural Networks.

Examples of RNN architectures :

1. Many-to-Many
  - a. When  $T_y = T_x \rightarrow$  Name entity recognition
  - b. When  $T_y \neq T_x \rightarrow$  Machine translation(Spanish – to – English translation)
2. Many-to-one  $\rightarrow$  Sentiment Analysis
3. One-to-One
4. One-to-Many  $\rightarrow$  Sequence generation(music generation)



Language Model and Sequence Generation :

➤ Speech Recognition System:

The Language model will tell, what is the probability of that particular sentence, and by probability of sentence, I mean, if you were to pick up a random newspaper, open a random email, or pick a random webpage, or listen to the next thing someone says, the friend of you says, what is the chance that the next sentence you read somewhere out there in the world will be a particular sentence like the apple and pear salad?