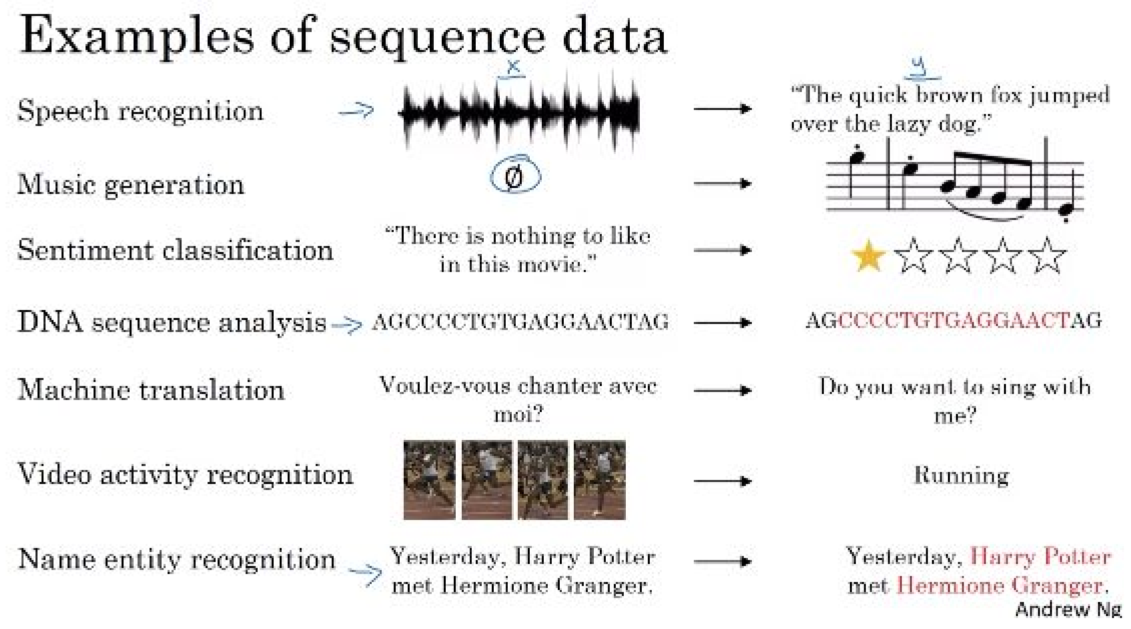
Andrew Ng's Recurrent Neural Networks

Week 1: Recurrent Neural Networks

Why Sequence Models?

- Sequence Models, like Recurrent Neural Networks, have transformed speech recognition, NLP and other areas. Let's start with some examples of sequence data: an audio clip, music generation, sentiment classification, machine translation, video activity recognition, etc. In general, we are given an input X and we are asked to map it to a transcript Y. Therefore, all these problems can be addressed with supervised learning and with labeled data X,Y as the training set. Finally, both the input X and the output Y are sequences.



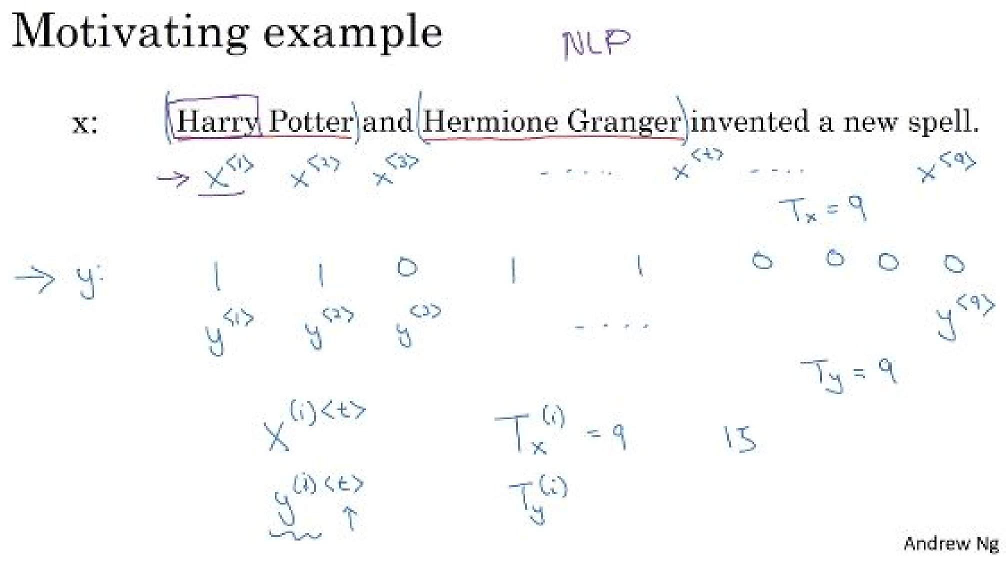
Examples of sequence data

	x		y
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, Harry Potter met Hermione Granger.

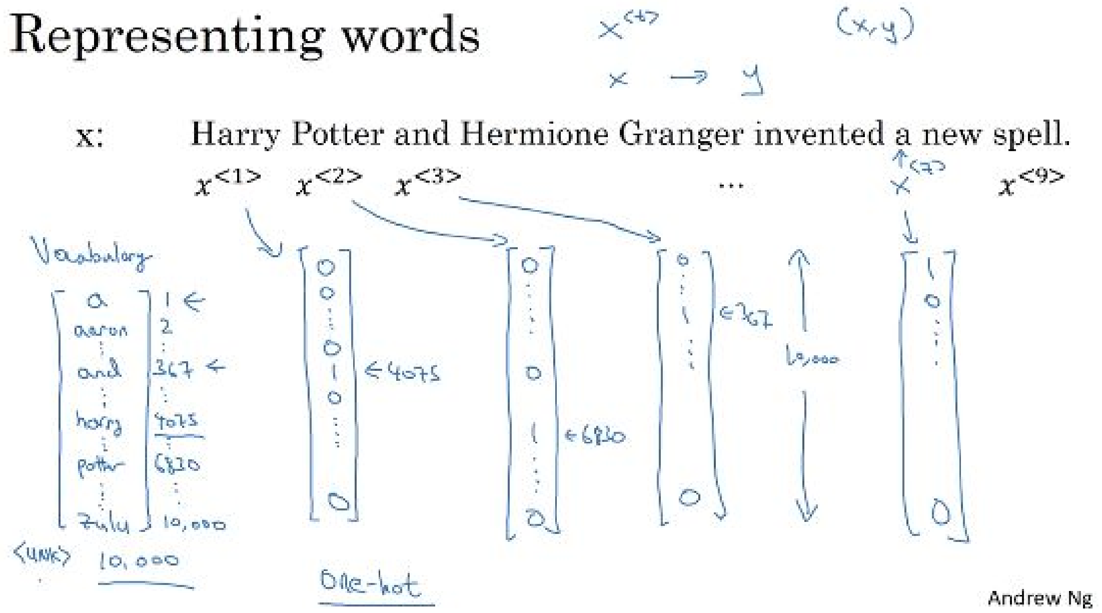
Andrew Ng

Notation

- Now we will spend some time going over the notation we are going to use. We'll illustrate it using the example below. We have an input x which has 9 words and we want to know which words are part of a name so our output y will be a vector of 0's (word is not part of a name) and 1's (word is part of a name). Let's start with the notation: To index into our input x we use $x^{<i>}$ so, for example, the first word is $x^{<1>}$, the second word is $x^{<2>}$ and so on. We have the same case for our output vector y, the first value would be $y^{<1>}$, the second one would be $y^{<2>}$ and so on. The length of our input is denoted by T_x, in our example below $T_x = 9$, similarly for output y its length is $T_y = 9$. Since we can have different training examples (and different output vectors) we will have different lengths denoted by $T_x^{(i)}$ and $T_y^{(i)}$. Finally, in to index into a word in training example i we would use the notation $x^{(i)<t>}$ and to do the same thing for the output we would use $y^{(i)<t>}$.



- Next let's look at how we will represent individual words in a sentence. Suppose we continue with the same sentence. The first thing we do to represent a word in a sentence is to create a vocabulary/dictionary which is just a list of words we are going to use in our representations. For this example we are going to use a vocabulary of 10,000 words but in practice NLP problems use bigger dictionaries, say 100,000 words. What we do next is use a one-hot vector to represent each word.

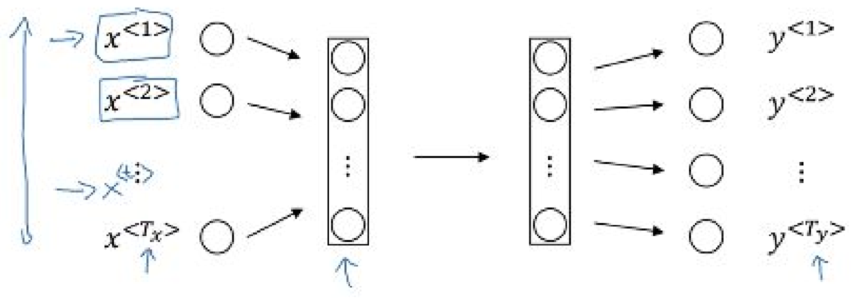


For example for the first word in the sentence below we would have a 10,000-dimensional vector with a 1 in position 4075, another 10,000-dimensional vector with a1 on position 6830 for the second word in our sentence and so on. If we run into a word that we don't know we then use the label '<UNK>'. Our goal is, given this representation of X is to learn a mapping using using a sequence model to the target output Y.

Recurrent Neural Network Model

- In this session we are going to build a neural network to learn the mapping from x to y. The first question is why we cannot use a standard network? For a couple of reasons: the inputs and outputs can have different lengths in different examples and there is no sharing of features learned across different text positions. For example, if we learned that Harry in position 1 is part of a name it would be nice to know that it is part of a name in another position? A recurrent neural network doesn't have these problems

Why not a standard network?

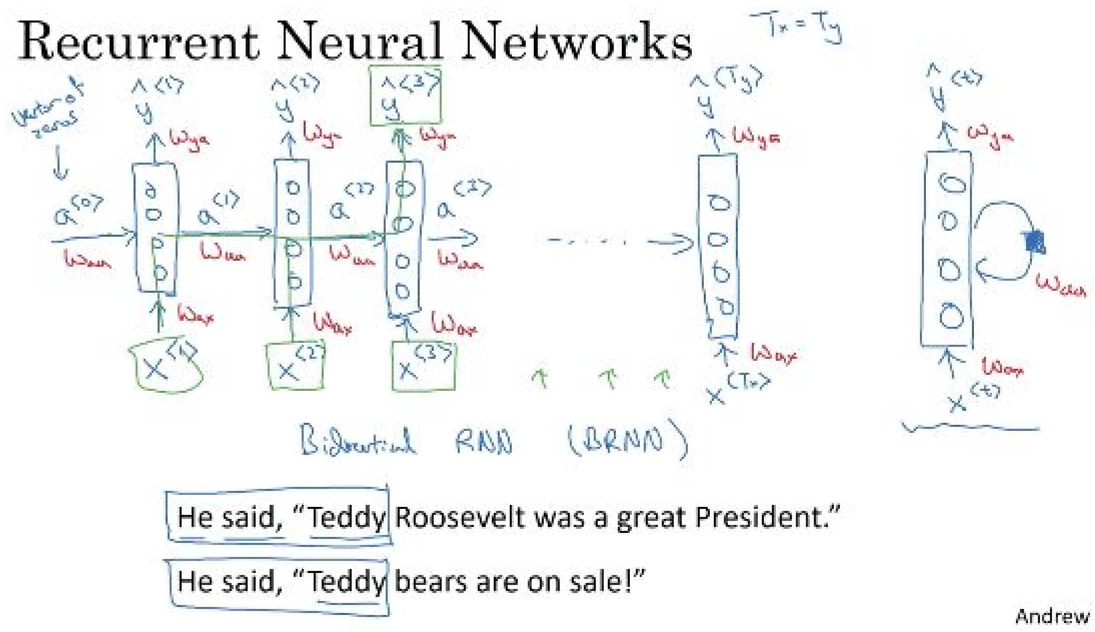


Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

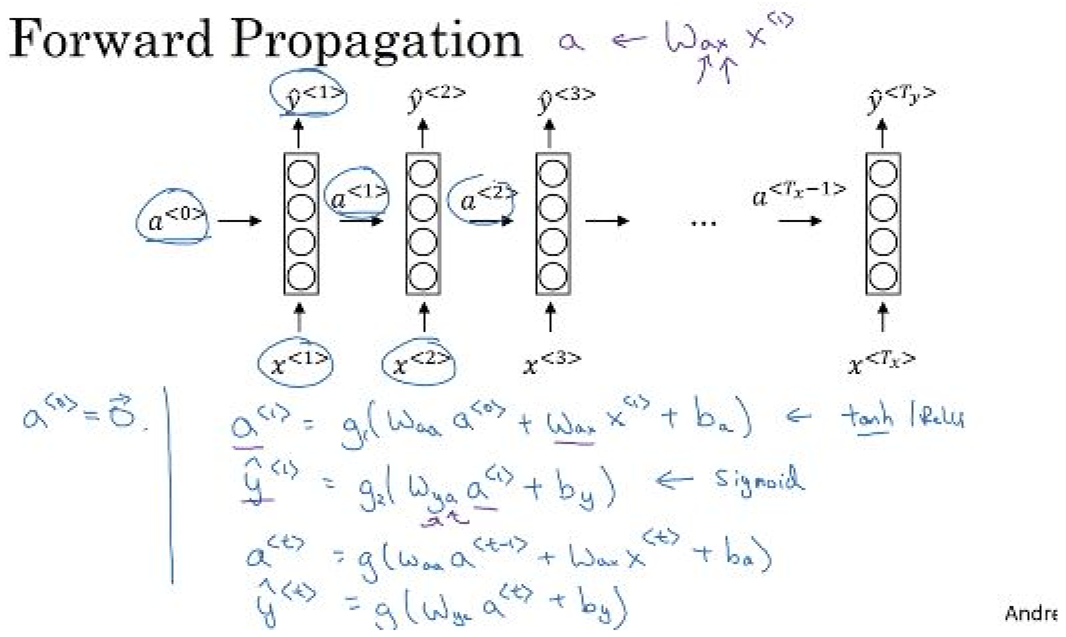
Andrew N

- What is a recurrent neural network? We are going to build one. We take the first word $x^{<1>}$ and feed it to a neural network layer which will try to predict if the word is part of a name and will output $yhat^{<1>}$. What happens next is we take the 2nd word, $x^{<2>}$, and feed it to another neural network layer but to do its prediction $yhat^{<2>}$ it will receive information from the previous step. In particular, it will receive activation $a^{<1>}$. The same process repeats itself across the sequence. Input $x^{<3>}$ will go into another neural network layer that will receive $a^{<2>}$ to help with prediction $yhat^{<3>}$ and so on. To kick it off, the initial activation $a^{<0>}$ is initialized usually with a vector of zeros.

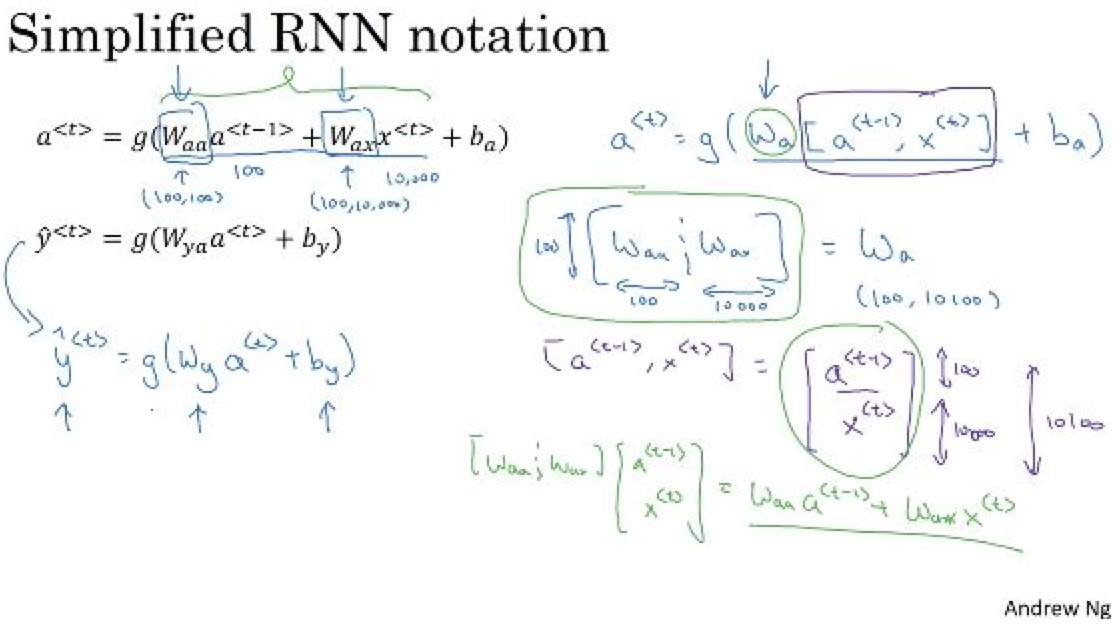


One weakness of this architecture is that it doesn't receive help from later parts of the network, only from 'earlier' parts. We'll address this with bidirectional recurrent neural networks (BRNNs). Also, the parameters that each layer uses are shared so, for example, the parameters for the connection from $x^{<1>}$ to the neural network layer are shared and we will identify them as W_{ax}. We do the same for the other parts of the neural network: W_{ya} for the output predictions and W_{aa} for the horizontal connections.

- Now let's define how forward propagation (from left to right) works. We know that we will initialize $a^{<0>}$ as a vector of zeros. $a^{<1>} = g(W_{aa} * a^{<0>} + W_{ax} * x^{<1>} + b_a)$. $yhat^{<1>} = g(W_{ya} * a^{<1>} + b_y)$. The activation for a is usually *tanh* or *ReLu* and the activation for yhat is usually a sigmoid. Generalizing, we have $a^{<t>} = g(W_{aa} * a^{<t-1>} + W_{ax} * x^{<t>} + b_a)$ and $yhat^{<t>} = g(W_{ya} * a^{<t>} + b_y)$.

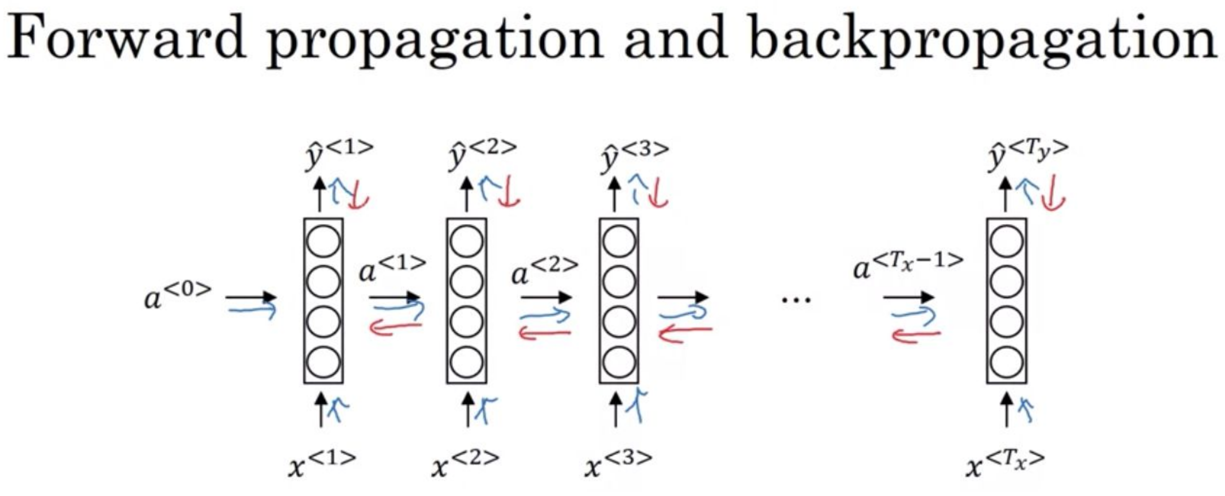

Forward Propagation
$a \leftarrow W_{ax} x^{<1>}$
$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<3>}$ $\hat{y}^{<T_y>}$
$a^{<0>}$ $a^{<1>}$ $a^{<2>}$ $a^{<T_x-1>}$
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<T_x>}$
$a^{<0>} = \vec{0}$
$a^{<1>} = g_1(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a) \leftarrow$ tanh / ReLU
$\hat{y}^{<1>} = g_2(W_{ya} a^{<1>} + b_y) \leftarrow$ sigmoid
$a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a)$
$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$
Andre

- Now we are going to simplify the notation as follows: we take our equation for $a^{<t>}$ and rewrite as $g(W_a * [a^{<t-1>}, x^{<t>}] + b_a$ so what we are doing is that we are stacking matrices $a^{<t-1>} and\, x^{<t>}$ as shown below. If $a^{<t-1>}$ has a dimension of 100 and $x^{<t>}$ has a dimension of 10,000 the resulting matrix has a dimension of 10,100. The advantage of this simplification is that we are carrying just one parameter instead of two and this will be useful when we develop more complex models. We also rewrite $yhat^{<t>}$ as $yhat^{<t>} = g(W_y * a^{<t>} + b_y)$.

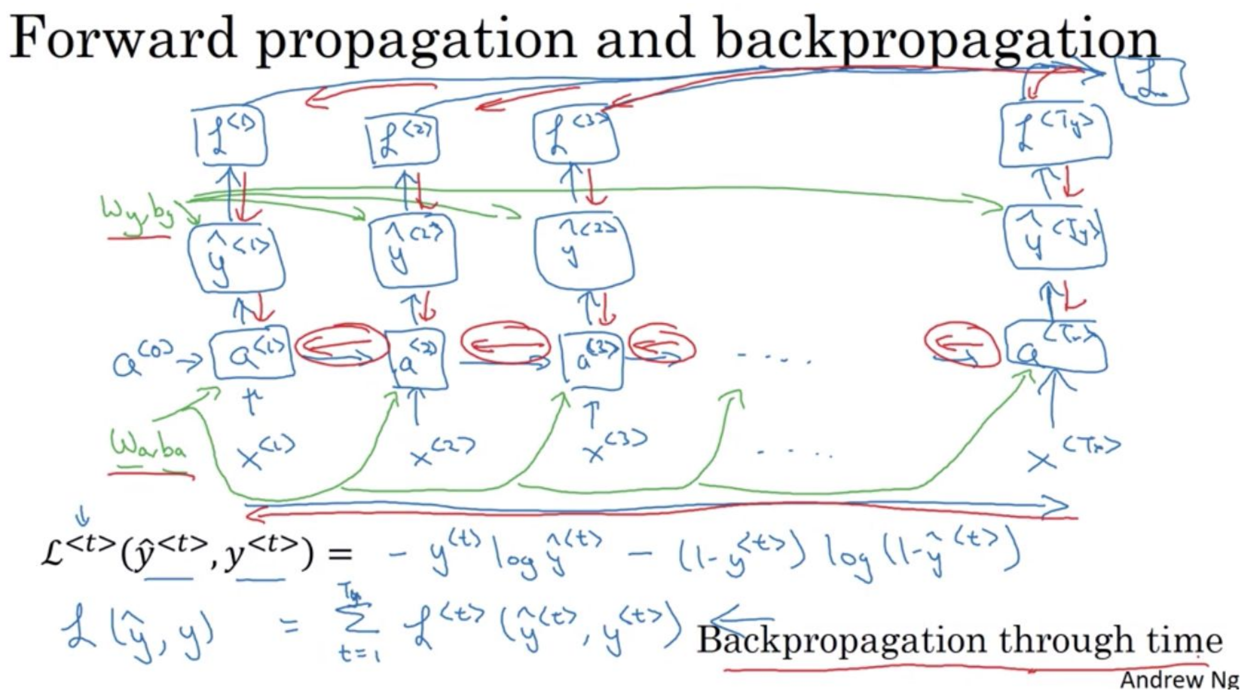

Simplified RNN notation
$a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a)$
(100,100) 100 (100,10,000) 10,000
$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$
$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$
$a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$
$[W_{aa} \vdots W_{ax}] = W_a$ 100 100 10000 (100, 10100)
$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$ 100 10000 10100
$[W_{aa} \vdots W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$
Andrew Ng

Backpropagation through time

- Now we are going to see how backprop works for a RNN. We remember that forward propagation is from left to right so backward propagation is pretty much the opposite: from right to left.

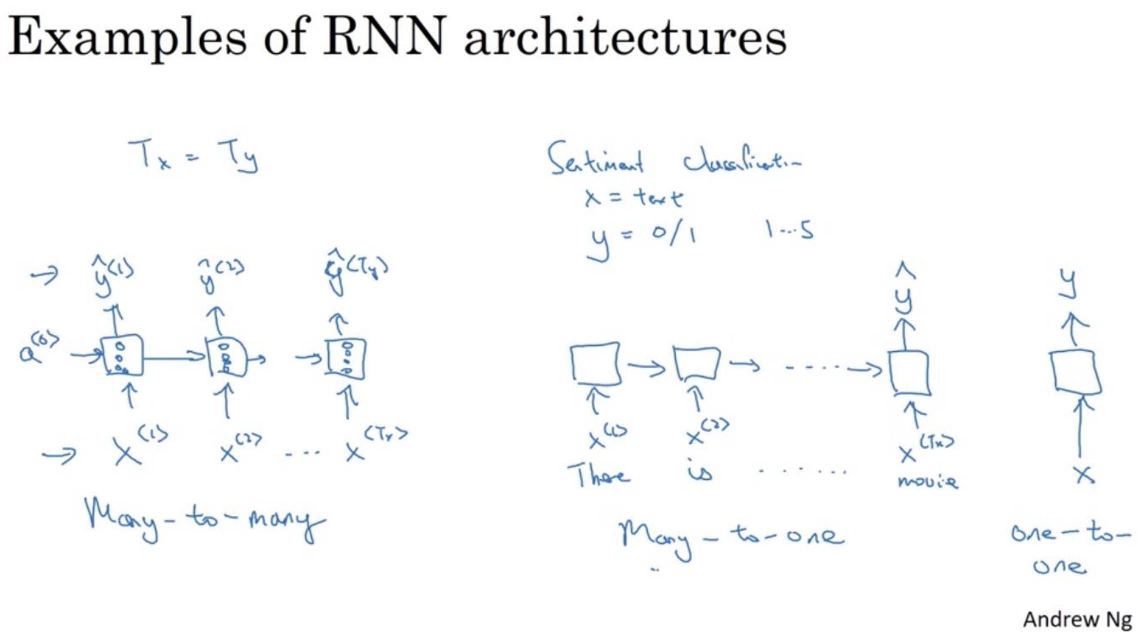


- Let's look at the forward prop calculations again. We start, from left to right, with $a^{<0>} and\, x^{<1>}$ which are used to calculate $a^{<1>}$, then $a^{<1>}$ $and\, x^{<2>}$ are used to calculate $a^{<3>}$ and so on. We also use parameters W_a $and\, b_a$ in every node. Given $a^{<1>}$ we can then compute $yhat^{<1>}$, given $a^{<2>}$ we can then compute $yhat^{<2>}$ and so on. Now we can do backprop and we need a loss function. The loss function for one node is defined as $L^{<t>}(yhat^{<t>}, y^{<t>}) = -y^{<t>} * log(yhat^{<t>}) - (1 - y^{<t>}) * log(1 - yhat^{<t>})$. The overall loss function $L(yhat, y)$ is then just the summation of the element-wise loss function and is expressed as $L(yhat, y) = \sum_{t=1}^{T_x} L^{<t>}(yhat^{<t>}, y^{<t>})$. Since we are going backwards and we call each step timestep this backprop is known as backpropagation through time.



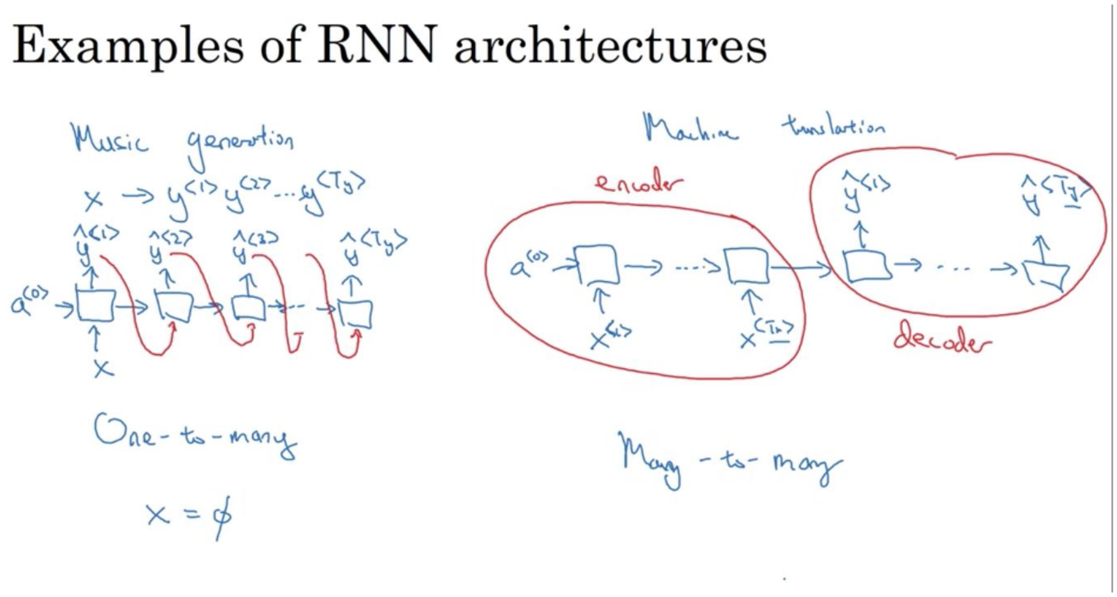
Different types of RNNs

- So far we have seen RNNs where the number of inputs and the number of outputs is the same. However, there are applications where this is not the case so let's review a few different RNN architectures. The architecture we know is called many-to-many and is illustrated on the left below.

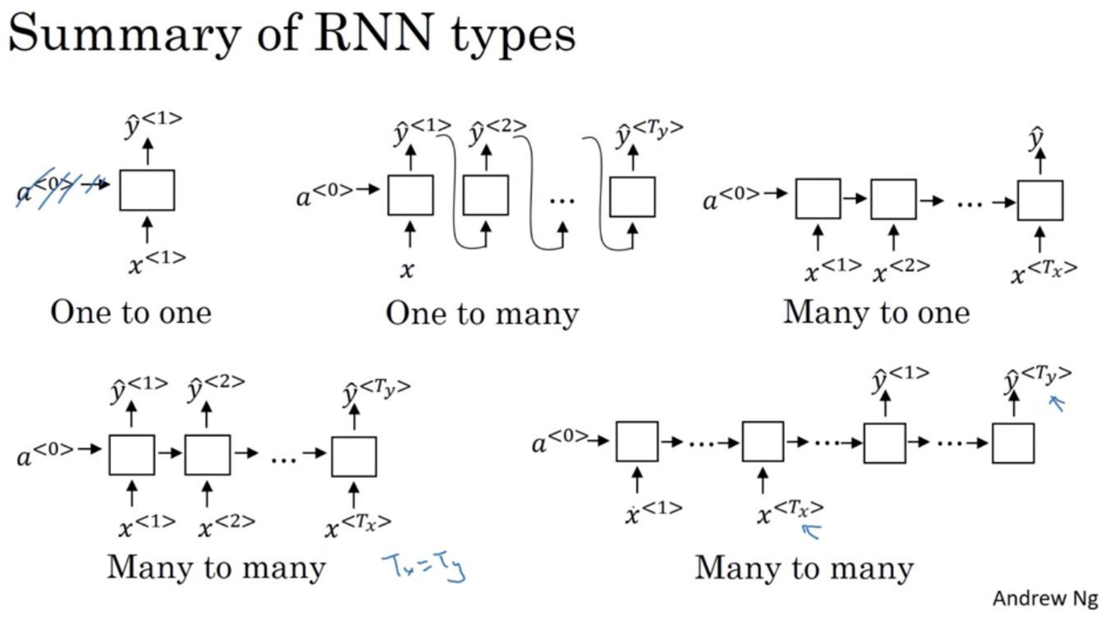


Other architecture types are: many-to-one illustrated by the sentiment classification problem shown above. In this case we have many inputs, $x^{<1>}, x^{<2>}$...., $x^{<t>}$ but only one output $yhat$ because we are trying to determine if the text entered is a positive/negative review or a 1-5 scale. Another architecture is one-to-one where we have one input and one output.

- There is also a one-to-many architecture and a many-to-many architecture, both illustrated below. An application that uses a one-to-many-architecture is music generation where there is only one x and the y's are calculated using the previous y's,. An example of many-to-many is machine translation where we first read all the text in the first language (say french) and then we output the translated version in say, English. The first part is called encoder and the second part is called decoder.



- Summarizing, the architecture we looked at are shown below.

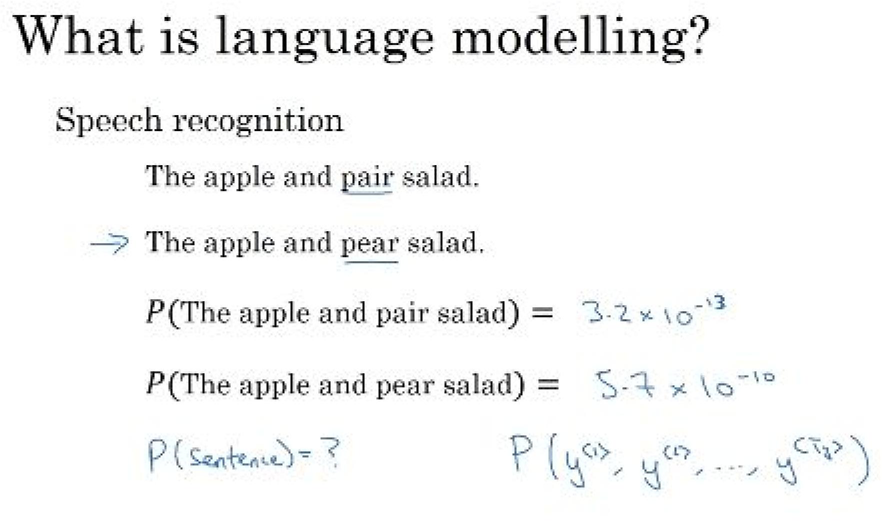


They are:

- One-to-one
- One-to-many and an example was music generation
- Many-to-one and the example was sentiment classification
- Many-to-many which has two version
    - One where the inputs and outputs are the same and
    - One where the inputs and outputs are not the same and the example we had was machine translation from one language to another.

Language Model and Sequence Generation

- Language Model is one of the basic building blocks in NLP, let's see what it means. Let's start with an example. I say a sentence and how do we know which option (from the 2 options below) is the one we should pick? We choose by selecting the option with the highest probability. Language modeling then gives us the probability of a sentence given an input sentence that we will denote as y's: $y^{<1>}, y^{<2>}, \dots, y^{<T_y>}$ so we can write language modeling as $P(y^{<1>}, y^{<2>}, \dots, y^{<T_y>})$.



What is language modelling?

Speech recognition

The apple and $\underline{\text{pair}}$ salad.

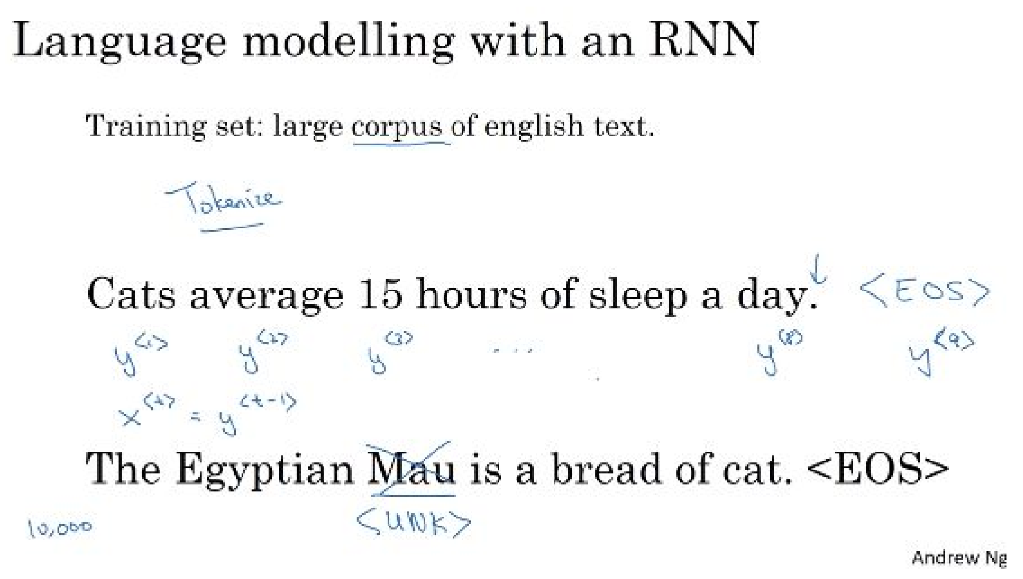
$\rightarrow$ The apple and $\underline{\text{pear}}$ salad.

$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$

$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$

$P(\text{sentence}) = ?$ $P(y^{<1>}, y^{<2>}, \dots, y^{<T_y>})$

- How do we build a language model? First we need a training set which is composed of a large body (corpus) of english text. Once we get a sentence the first thing we do is tokenize it. We tokenize it using the y notation we discussed above. We also add an $< EOS >$ token at the end of the sentence. If we have a word that is not in our dictionary (perhaps our dict is small, say 10,000 words) we use the $< UNK >$ token and we model the probability using the $< UNK >$ token. Lastly, the input for our network will be as follows: the input for $x^{<t>} = y^{<t-1>}$.



Language modelling with an RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day. <EOS>

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ . . . $y^{<8>}$ $y^{<9>}$

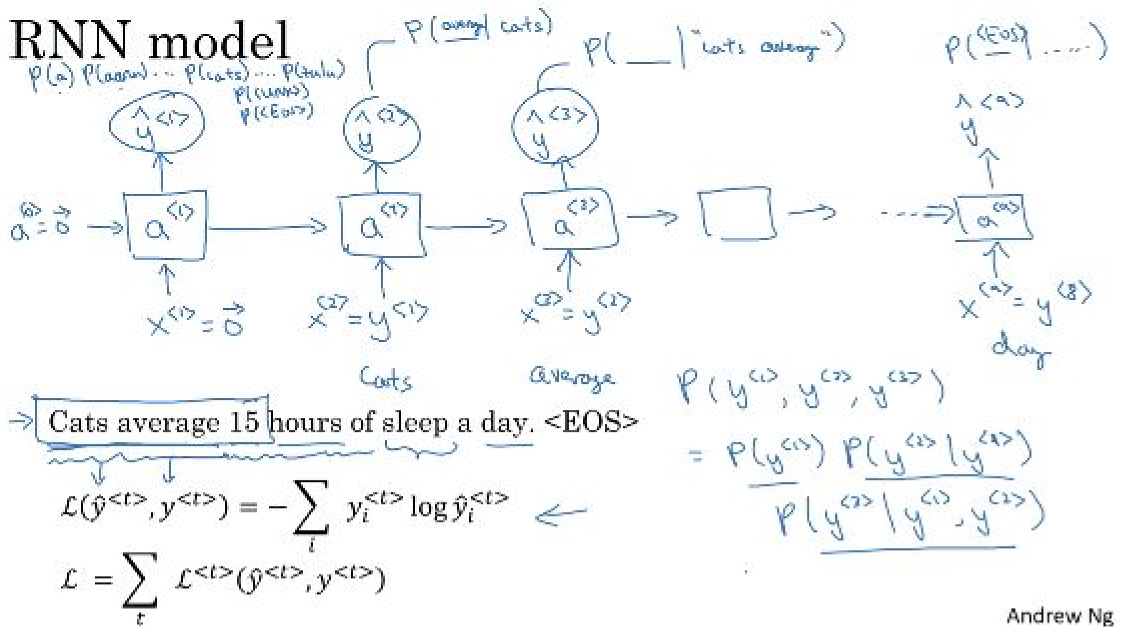
$x^{<t>} = y^{<t-1>}$

The Egyptian ~~Mau~~ is a bread of cat. <EOS>

10,000 <UNK>

Andrew Ng

- How do we build the neural network? We start by initializing $a^{<0>} and\, x^{<1>}$ to 0 and $a^{<1>}$ is a softmax activation that will output a probability for each of the 10,000 words in our dict: P(a), P(aaron)...P(cats)...P(<UNK>)...P(<EOS>). We then move on to the next layer and make $x^{<2>} = yhat^{<1>}$. We are telling the neural network what was the first word in our sequence. This layer then outputs $yhat^{<2>} = P(average \mid cats)$. We do the same in the next layer: $x^{<3>} = yhat^{<2>}$ and $yhat^{<3>} = P(... \mid "cats\ average")$. We repeat the same process until the end of our sequence which, in this case, is $x^{<9>} = yhat^{<8>}$ and $yhat^{<9>} = P(< EOS > \mid "cats\ average....")$.



RNN model

Cats average 15 hours of sleep a day. <EOS>

$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

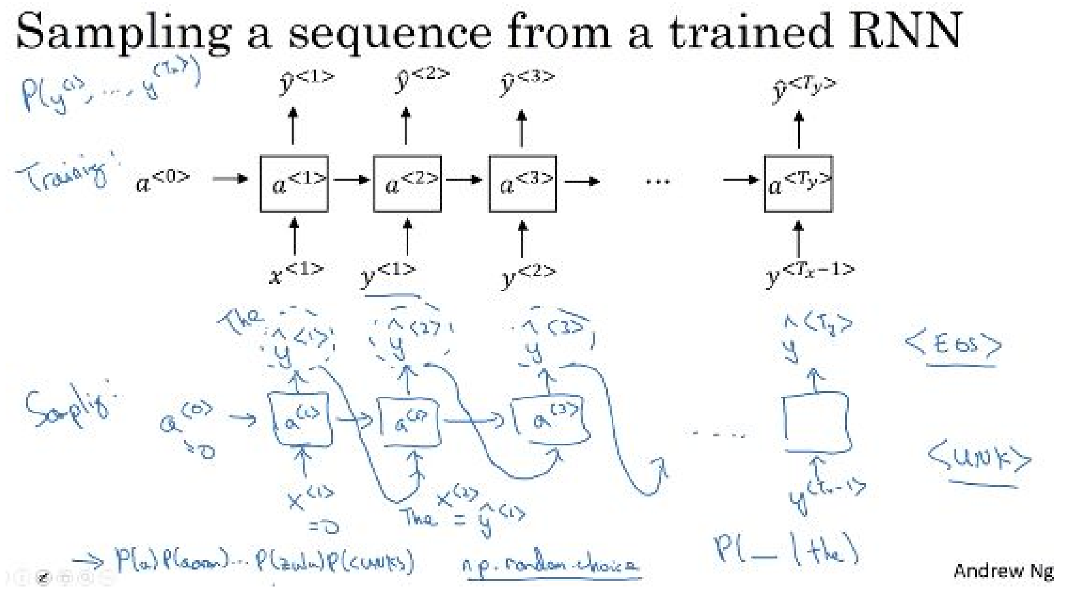
$P(y^{<1>}, y^{<2>}, y^{<3>})$
$= P(y^{<1>})\, P(y^{<2>} \mid y^{<1>})$
$P(y^{<3>} \mid y^{<1>}, y^{<2>})$

Andrew Ng

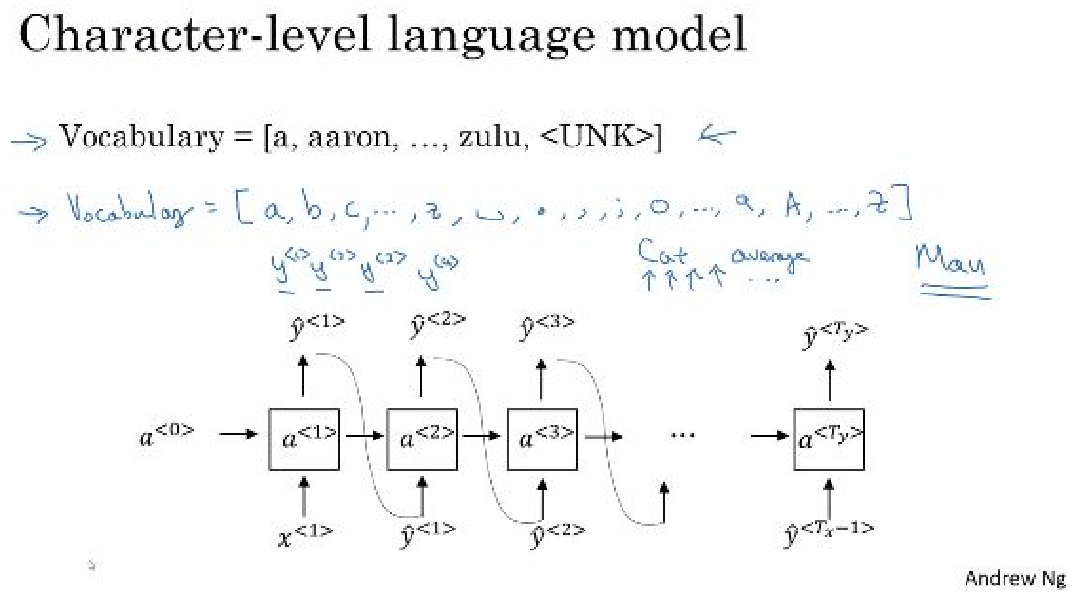
So $P(y^{<1>},\ y^{<2>},\ y^{<3>}) \ = P(y^{<1>}) \ *\ P(y^{<2>} \mid (y^{<1>}) \ * P(y^{<3>} \mid y^{<1>}, y^{<2>})$.

Sampling Novel Sequences

- Once we have trained a sequence model one of the ways to sense what it has learned is to have it sample novel sequences. Let's see what that means. To do sampling what we do is we randomly sample the output $yhat^{<1>}$ and we use something like $np.random.choice$ and we end up with one word. That word is then fed to the second layer and we do the same process again: we sample $yhat^{<2>}$ and end up with a word which is then fed to the next layer and so on. We stop when we get an <EOS> or sometimes you stop after 20, 40 words.



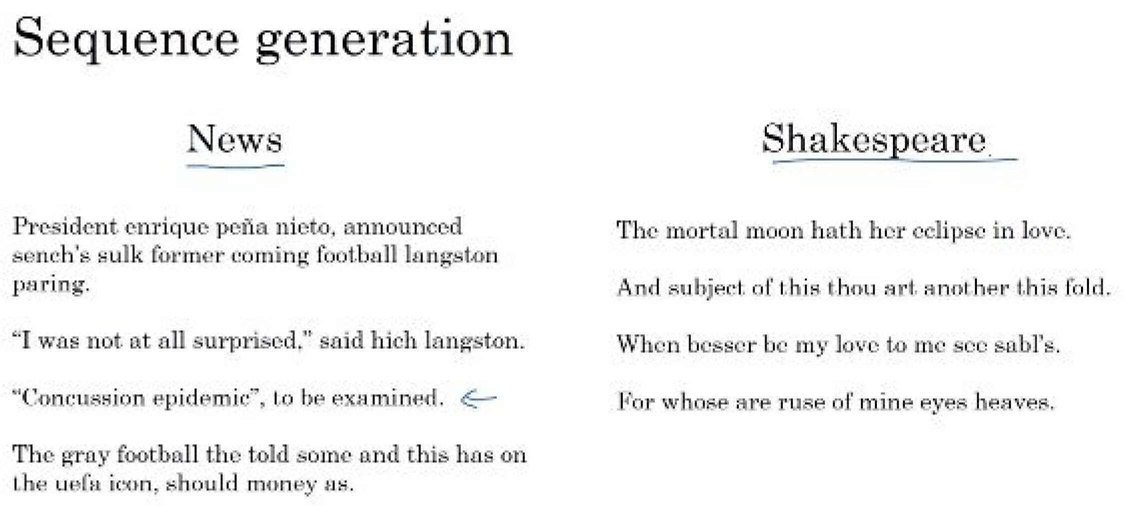
- We can also build character-level RNNs and for that we use as a vocabulary characters instead of words so our dictionary will look like this: $[a,\ b,\ c,\ ...,\ A,\ B,\ C, ...,\ Z]$. For our example sentence C would $y^{<1>}$, A would be $y^{<2>}$ and so on. We then do the same process as before.



Character-level language model
Vocabulary = [a, aaron, ..., zulu, <UNK>]
Vocabulary = [a, b, c, ..., z, ␣, ., ,, ;, 0, ..., 9, A, ..., Z]
y<1> y<2> y<3> y<4>
Cat average
Max
a<0>
a<1>
a<2>
a<3>
a<Ty>
x<1>
ŷ<1>
ŷ<2>
ŷ<3>
ŷ<Ty>
ŷ<Tx-1>
Andrew Ng

One advantage of a character-level language model are that it can deal with words that might not have been in your word-level language model so we never deal with unknown tokens. Disadvantages are that you end up with longer sequences and these kinds are models are more computationally expensive to train. We still see word-level language models as more common.

- Below are some examples of text that were sampled from character-level language model. One was trained with news and the other one was trained with Shakespeare.



Sequence generation

News

President enrique peña nieto, announced sench's sulk former coming football langston paring.

"I was not at all surprised," said hich langston.

"Concussion epidemic", to be examined.

The gray football the told some and this has on the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.

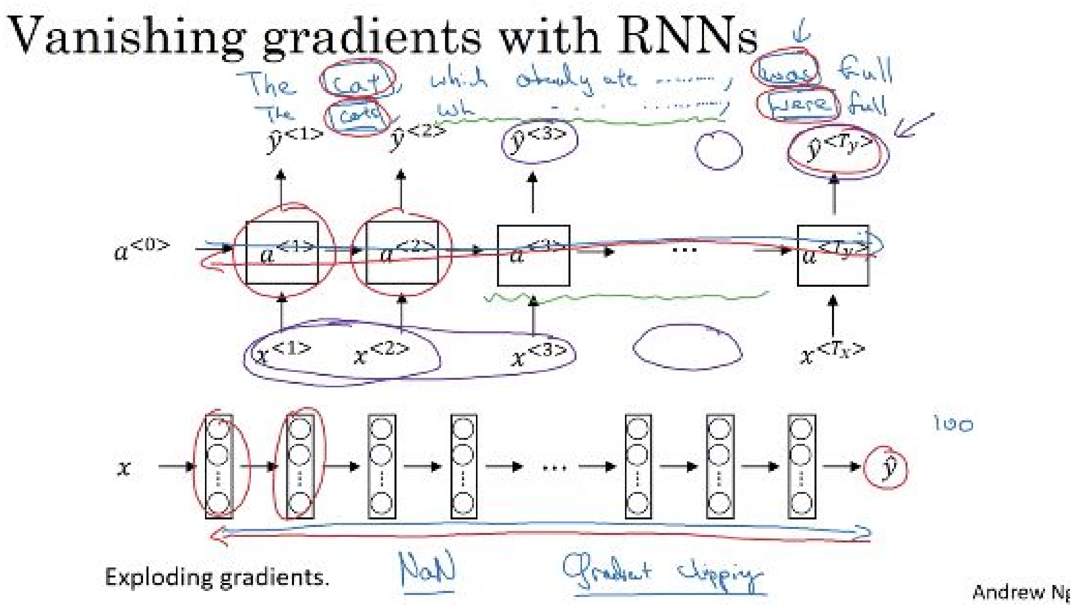
And subject of this thou art another this fold.

When besser be my love to me see sabl's.

For whose are ruse of mine eyes heaves.

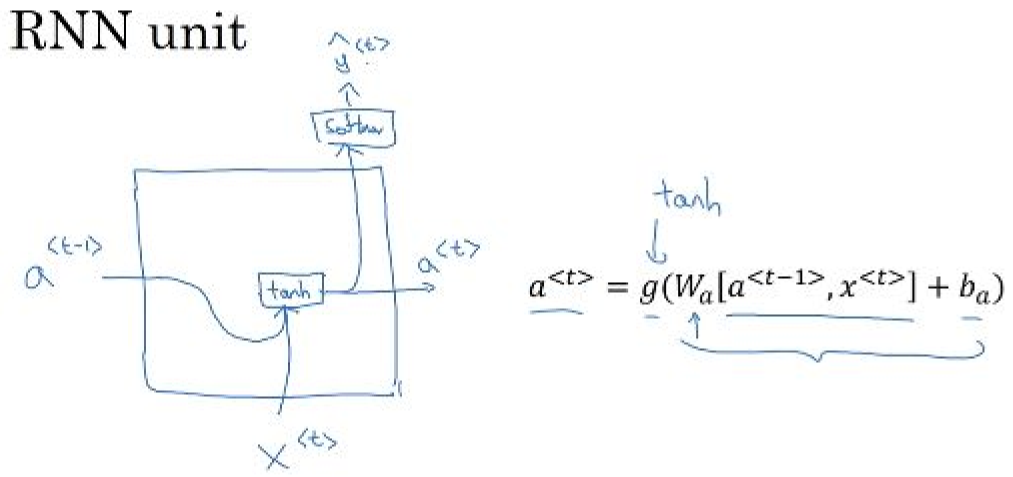
Vanishing Gradients with RNNs

- The RNNs we have seen so far is that it does not do a good job of dealing with long-term relationships. It doesn't because it runs into the vanishing gradient problem, a problem we have seen other NNs run into. To illustrate the problem consider the 2 sentences below: there is a relationship between the noun (cat or cats) and the verb (was or were). Our current RNNs will not be able to deal with this relationship because, as we just saw, input $x^{<3>}$ is dependent on $x^{<1>} and\, x^{<2>}$ but not on later inputs. RNNs also have the problem of exploding gradients but we can deal with that problem with gradient clipping (rescale your gradient vector if it is bigger than a threshold). However, vanishing gradient is a harder problem to solve. In future videos, we will look at GRUs which is a robust solution for the vanishing gradient problem and will allow the RNN to capture much longer range dependencies.


Vanishing gradients with RNNs
The cat, which already ate ......, was full
The cats wh ..... were full
Exploding gradients.
NaN
Gradient clipping
100
Andrew Ng

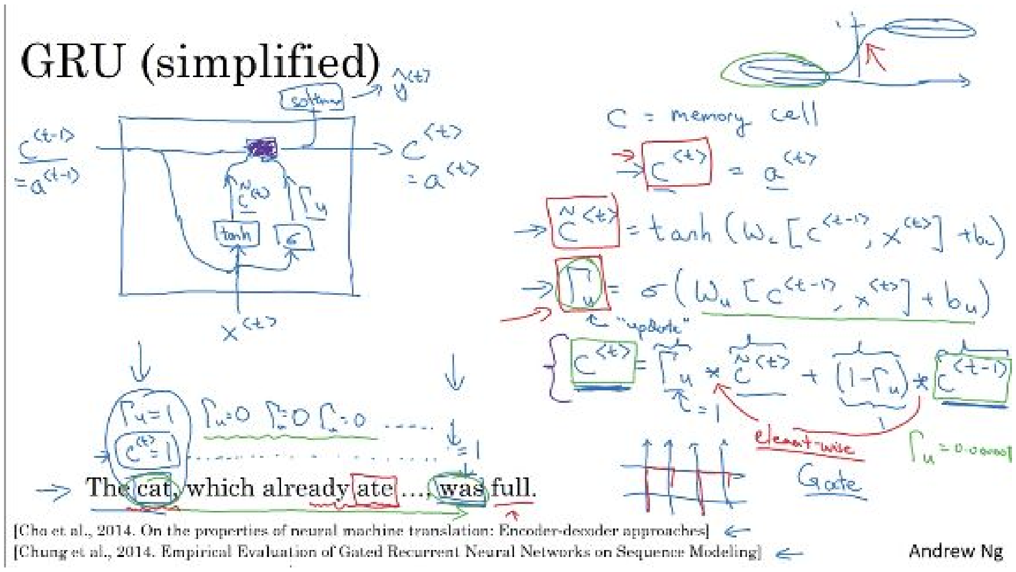
Gated Recurrent Unit (GRU)

- We are going to start looking at GRUs by reviewing a RNN unit for a hidden layer from a RNN. The RNN unit receives $a^{<t-1>}$ and $x^{<t>}$ as inputs to an activation tanh and computes the output activation $a^{<t>}$. $a^{<t>}$ could also be passed to a softmax which will use it to calculate $yhat^{<t>}$. We review this unit because GRU is a modification of it.



- Now let's take a look at the GRU. We are going to use the same sentence as before since it illustrates the purpose of the GRU: in this case we need to remember the noun 'cat' so we can check later that the verb agrees in number with the noun. OK, we start to see the U with a new concept, the memory cell which we will label C. We will have various C cells so we have $C^{<t>}$ which initially will be equal to $a^{<t>}$. We also have $\tilde{C}^{<t>}$ which is a candidate to replace $C^{<t>}$ and finally we have a gate which we will call gamma and denote as Γ_u. So what happens is that in every layer we have $\tilde{C}^{<t>}$ which will try to replace $C^{<t>}$ and Γ_u will decide if that is the case. The equation for each parameter is

- $\tilde{C}^{<t>} = tanh(W_c * [C^{<t-1>}, X^{<t>}] + \ b)$
- $\Gamma_u = \sigma(W_u * [C^{<t-1>}, x^{<t>}] + b_u)$
- $C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \ (1 - \Gamma_u) * C^{<t-1>}$



GRU (simplified)

The cat which already ate ..., was full.

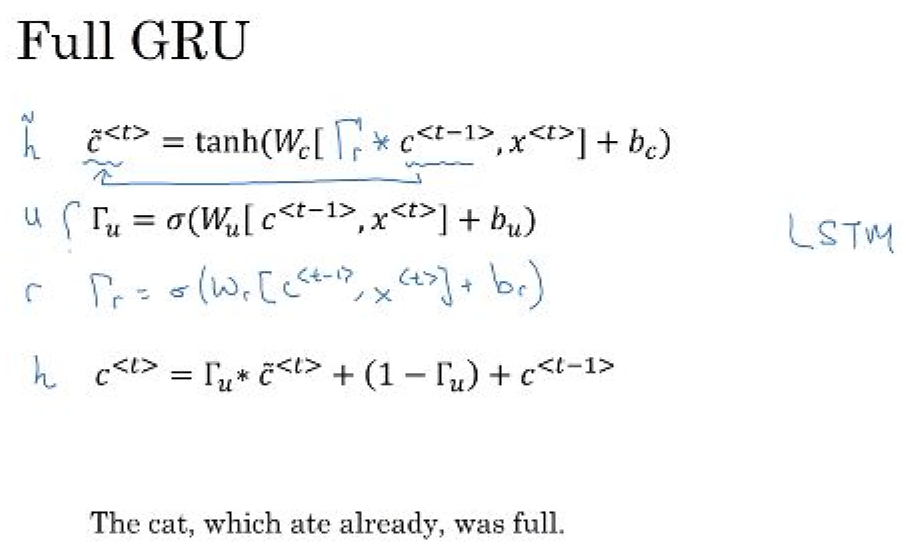
[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches]
[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Andrew Ng

$C^{<t-1>} = a^{<t-1>}$
softmax
$\hat{y}^{<t>}$
$C^{<t>} = a^{<t>}$
$\tilde{C}^{<t>}$
Γ_u
tanh
σ
$x^{<t>}$
C = memory cell
$C^{<t>} = a^{<t>}$
$\tilde{C}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$
$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$
"update"
$C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + (1-\Gamma_u) * c^{<t-1>}$
$\Gamma_u = 1$
$C^{<t>} = 1$
$\Gamma_u = 0$ $\Gamma_u = 0$ $\Gamma_u = 0$ ....
element-wise
Gate
$\Gamma_u = 0.000001$

$\tilde{C}^{<t>}$, $C^{<t>}$ and Γ_u are of the same dimension so the multiplication in the $C^{<t>}$ equation is an element-wise multiplication. So what happens, using our diagram, is that $C^{<t-1>}$ (which is initially $= a^{<t-1>}$) and $x^{<t>}$ will combine in a tanh activation to create $\tilde{C}^{<t>}$. These same parameters, combined with other parameters and a sigmoid function will give us Γ_u and $\tilde{C}^{<t>}$ and, finally, all these values combine to create the purple function above which will give us $C^{<t>}$. We could also send the output to a softmax to predict $yhat^{<t>}$. So that's GRU which is very good at going through the layers and deciding when is a good time to update the memory cell. Because of the sigmoid function it is very easy to set the gate to 0 and that will help us to carry the memory cell throughout the neural network's layers and this is what helps to minimize the vanishing gradients problem.

- What we saw above was the simplified version of a GRU. Now let's take a look at the full version. For a full GRU we copy the 3 equations from the previous slide and we will modify one equation and add a new equation. The equation for c tilde t $\tilde{C}^{<t>}$ will have another gate, the relevant gate denoted as Γ_r and its equation is $\Gamma_r = \sigma(w_r * [C^{<t-1>}, x^{<t>}] + b_r)$. Another variation of GRU is LSTM that we are going to look at in the next session. Another common notation used for these equations is shown below on the left but we will continue to use the notation we saw in the previous slide.



Full GRU

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

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

r $\quad \Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$

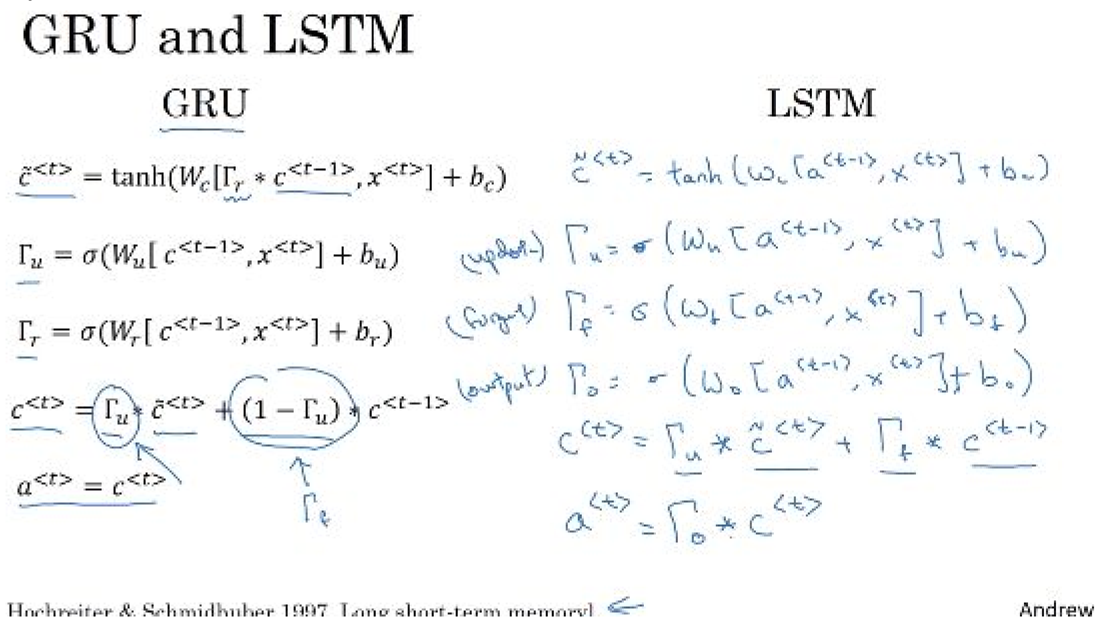
h $\quad c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$

LSTM

The cat, which ate already, was full.

LSTM (Long Short Term Memory) Unit

- In this session we are going to look at the LSTM unit which i more powerful than GRU. LSTM has 3 gates instead of 2. The gates are gamma update (Γ_u), gamma forget (Γ_f) and gamma output (Γ_o). In LSTM $a^{<t>} \neq c^{<t>}$ Instead $a^{<t>} = \Gamma_o * tanh\ C^{<t>}$. The rest of the equations are shown below.



GRU and LSTM

GRU

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

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

$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$

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

$a^{<t>} = c^{<t>}$

LSTM

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

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

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

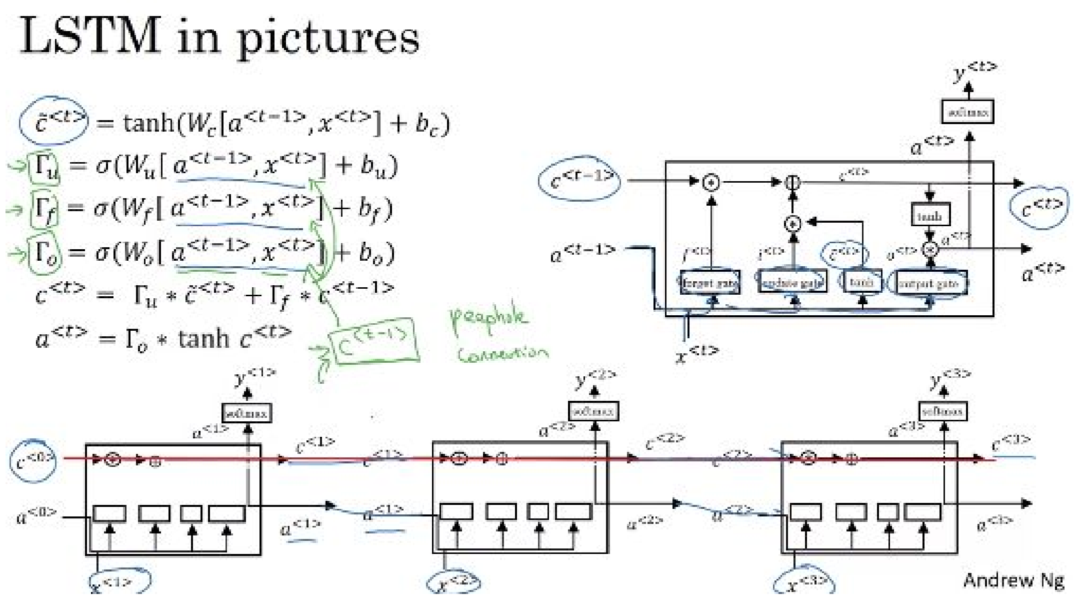
(output) $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

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

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

[Hochreiter & Schmidhuber 1997. Long short-term memory] ← Andrew

- Now let's look at a LSTM diagram. The diagram is shown below, notice how $a^{<t-1>} and\ x^{<t>}$ are used to calculate the update, forget and output gate. Some people also add $C^{<t-1>}$ to the gate equations and this is called a peephole connection. Even though we looked at GRU first LSTM came first and is still used more frequently. The advantage of GRU is that it is simpler and is faster. LSTM is more powerful and more effective because it has 3 gates instead of 2.



LSTM in pictures

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

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

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

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

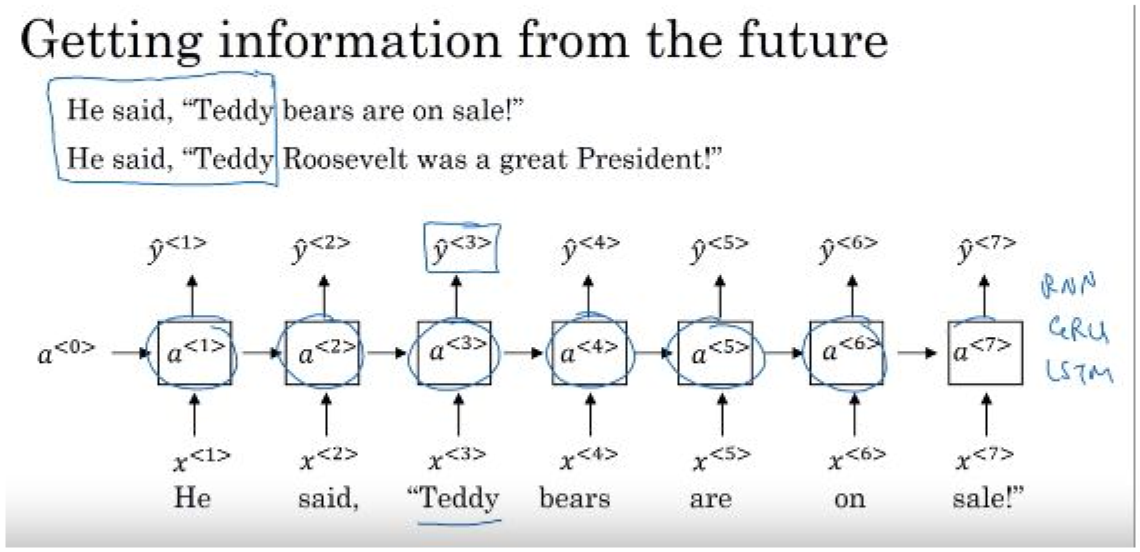
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$

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

$\boxed{c^{<t-1>}}$ peephole connection

Bidirectional RNNs

- Now we are going to look at bidirectional RNNs. Here's an example why they are needed. In the 2 sentences below we don't know if 'Teddy' is a name or something else. To know that, $yhat^{<3>}$ would need to look into the future so it can decide if its output is 0 or 1. How do we fix this problem? With bidirectional RNNs.



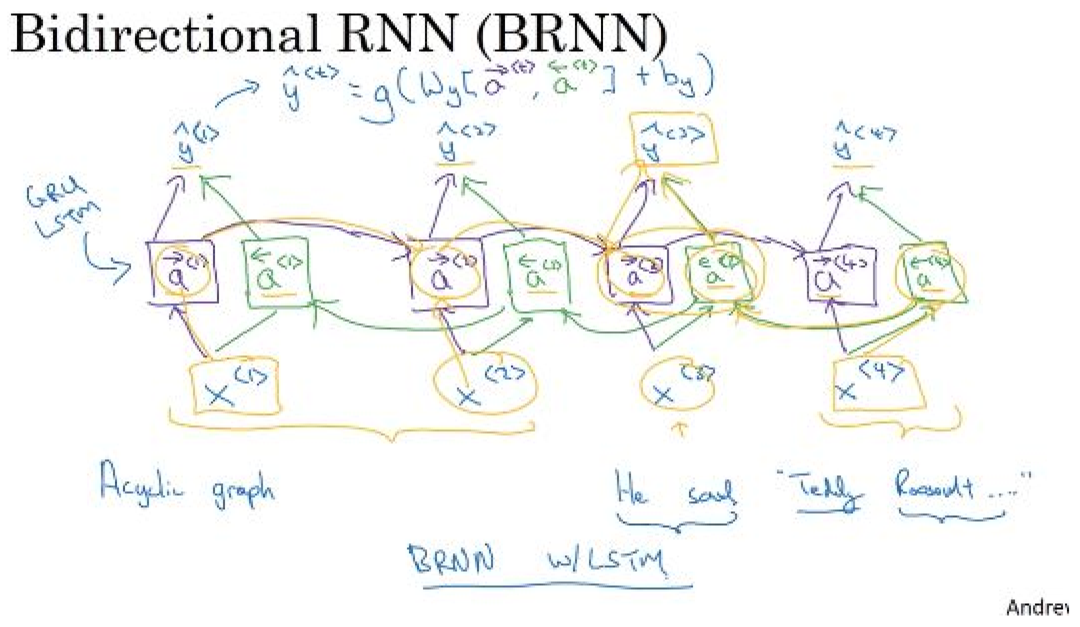
Getting information from the future

He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"

$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<3>}$ $\hat{y}^{<4>}$ $\hat{y}^{<5>}$ $\hat{y}^{<6>}$ $\hat{y}^{<7>}$
$a^{<0>}$
$a^{<1>}$ $a^{<2>}$ $a^{<3>}$ $a^{<4>}$ $a^{<5>}$ $a^{<6>}$ $a^{<7>}$
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<6>}$ $x^{<7>}$
He said, "Teddy bears are on sale!"
RNN
GRU
LSTM

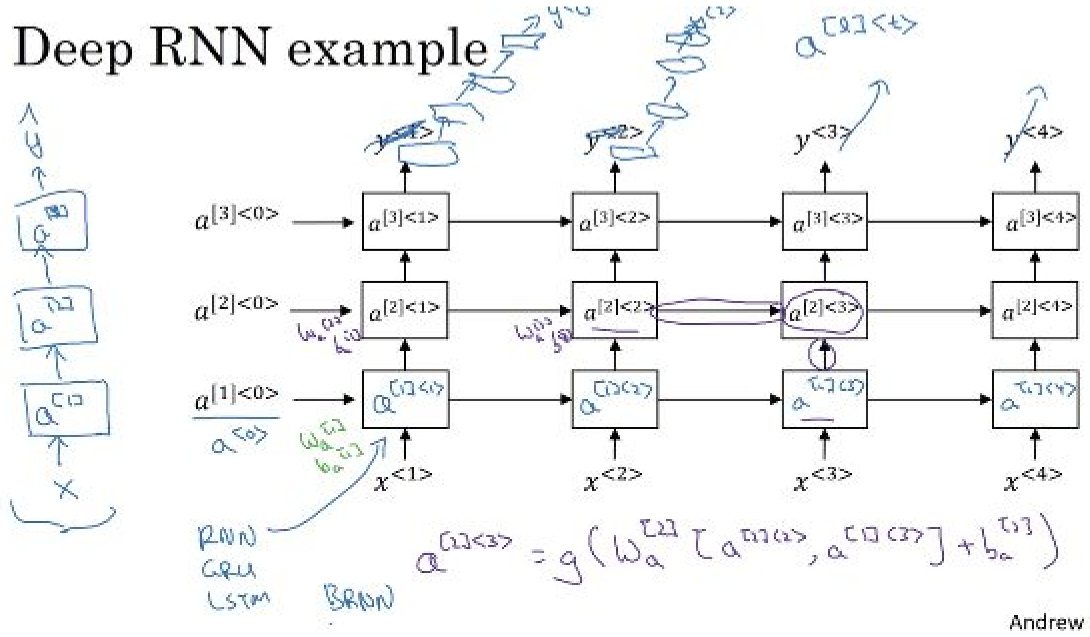
- Below we have the architecture of a BRNN. Notice there are forward and backward activations. So the predictions y use both the forward and backward information. Specifically, $yhat^{<t>} = g(w_y * [a^{\rightarrow<t>}, a^{\leftarrow<t>}] + b_y)$. This way, the yhat prediction is taking information from the past ($a^{\rightarrow<t>}$) and from the future ($a^{\leftarrow<t>}$). The diagram below is an acyclic graph so the activations will be computed both forward and backwards. Also notice the units can be GRU and LSTM too. A common architecture is a BRNN with LSTM units. The disadvantage of a BRNN is that you need the full sequence of speech before you can start making predictions. For real time speech recognition applications a BRNN is not the best choice but it is a good choice if you can get the whole speech at the same time.


Bidirectional RNN (BRNN)
$\hat{y}^{<t>} = g(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$
GRU
LSTM
Acyclic graph
He said "Teddy Roosevelt ..."
BRNN w/ LSTM
Andre

Deep RNNs

- In this session we are going to see how to build deep RNNs. The diagram on the left shows a traditional RNN. The diagram on the right shows a deep RNN where we stack various levels, one on top of each other. An activation is then computed as follows, let's ake for example $a^{[2]<3>}$ whose activation would be $g(w_a^{[2]} * [a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$. Notice the parameters w and b are the same for all the activations in the same layer. The deep RNN can have normal units or GRU or LSTM units too. In general we don't see 100 levels of deep RNNs, 3 is already too much.

Deep RNN example



$$a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

Andrew Ng