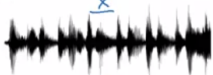

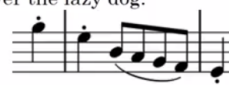
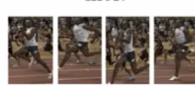


Sequence model

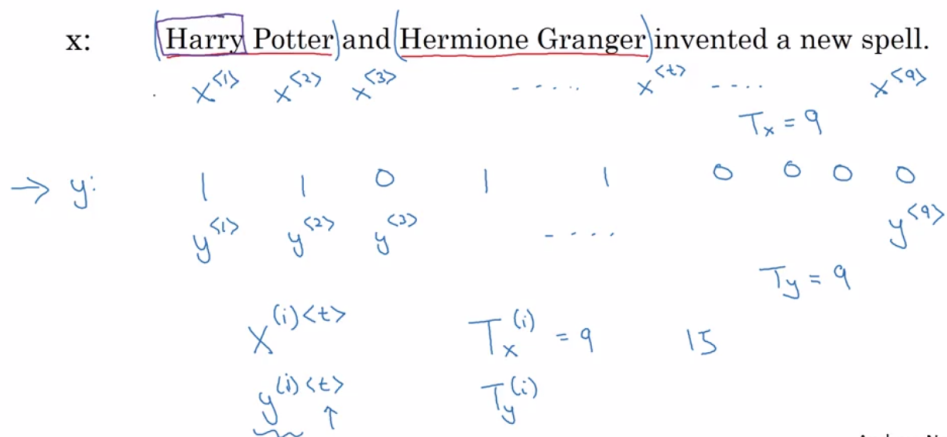
- see what it can do!

Examples of sequence data

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	AGCCCCTGTGAGGAAGTAG	→	AGCCCCTGTGAGGAAGTAG
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

Motivating example



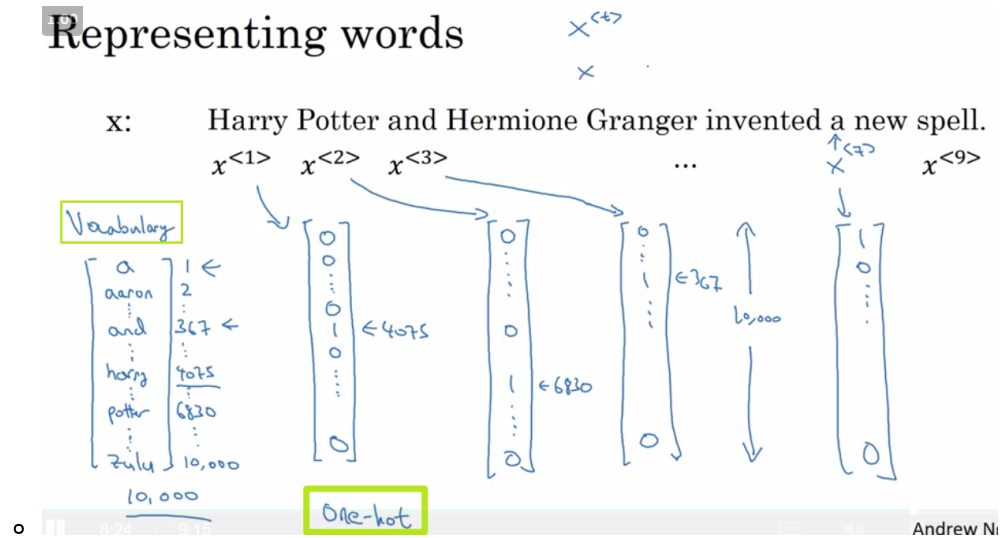
Andrew Ng

- T_x : the **size** of the input sequence, T_y : the size of the output sequence
- $x^{(i)(t)}$ is the element t of the sequence of input vector i .
- $T_x^{(i)}$ the input sequence length for training example i . It can be **different** across the examples.

- representing words

- use a vocabulary
- one-hot encoding

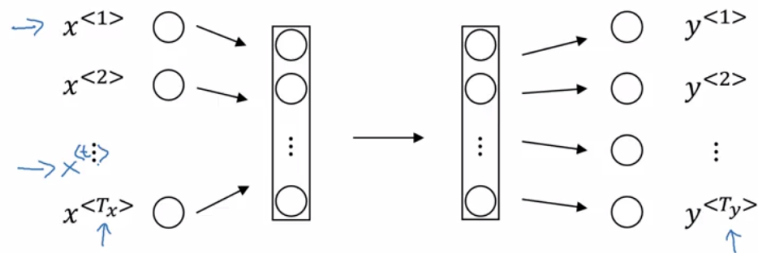
Representing words



RNN

- why use RNN to tackle with sequence tasks?

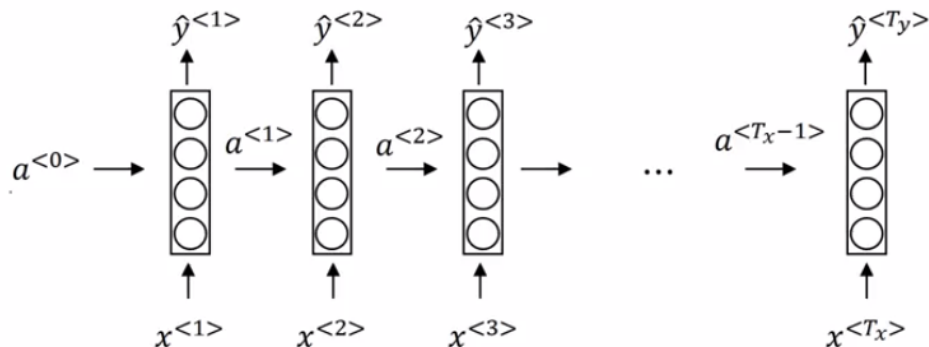
Why not a standard network?



Problems:

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

- basic RNN structure



- 3 kinds of matrices
 1. W_{ax} : (HiddenNeurons, n_x)
 2. W_{aa} : (HiddenNeurons, HiddenNeurons)
 - use to maintain memory

3. Wya: (ny, HiddenNeurons)

- weakness
 - can't catch the **long term dependencies** from previous and later.
- forward propagation

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

$(100, 100) \quad 100 \quad (100, 10,000)$

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

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

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

$$\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} = W_a \quad (100, 10100)$$

$$\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \quad \begin{matrix} \uparrow 100 \\ \uparrow 10000 \end{matrix} \quad \begin{matrix} \uparrow 100 \\ \uparrow 10100 \end{matrix}$$

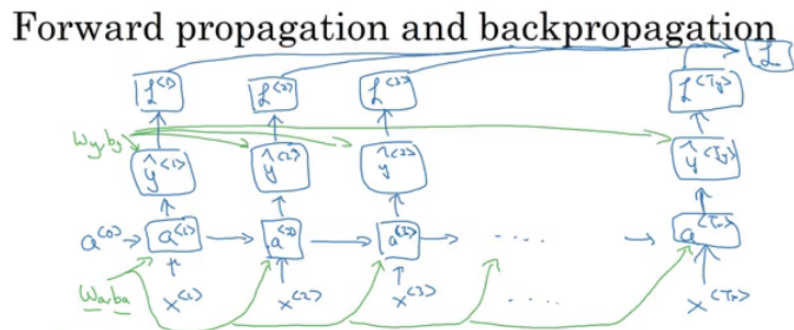
$$\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$$

-
- activation function: Relu or tanh
- details in slide
 - 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)
- backward propagation intuition
 - loss function

$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$

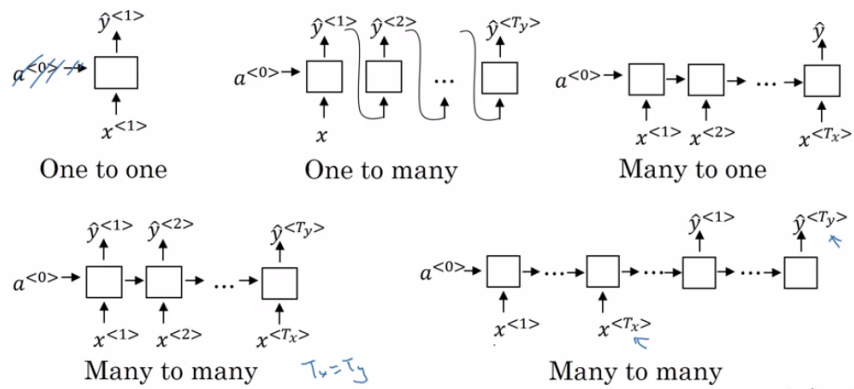
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^T \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

- diagram



- different kinds of RNN

Summary of RNN types



- one to many: music generation
- many to many:
 - equal length out put:
 - unequal: machine translation (encode then decode)

Language model

- what is it?
 - for speech recognition, some word sound **similar** and we have to judge which word is the speaker actually speaking.
 - essence: **conditional probability**

What is language modelling?

Speech recognition

The apple and pair salad.

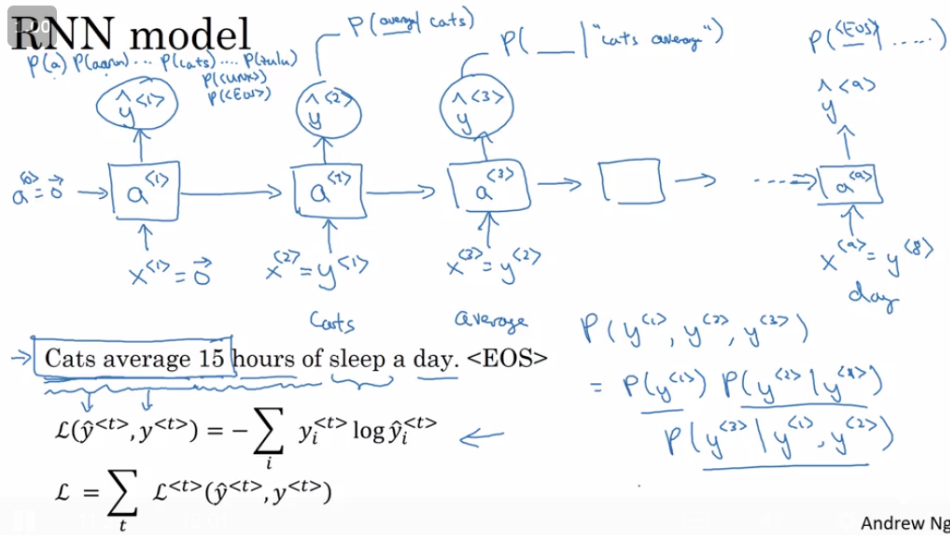
→ The apple and pear salad.

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

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

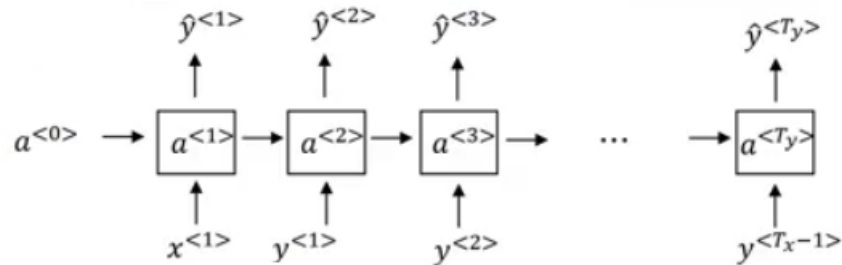
$$P(\text{Sentence}) = ? \quad 1 (y^{<1>}, y^{<2>}, \dots, y^{<T_y>})$$

- how to build?
 - get a **training** set: a large corpus of target language text.
 - tokenize** them by getting the vocabulary and then one-hot each word.
 - add token **<EOS>** at an end of sentence
- whole RNN:



• sampling novel sequences

- After a sequence model is trained on a language model, to check what the model has learned you can apply it to sample novel sequence.
- process:
 1. Given this model:



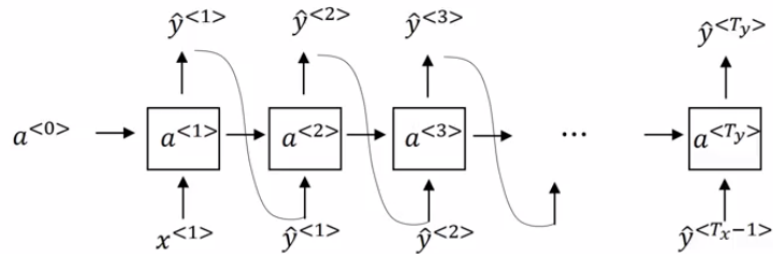
2. pass $a^{<0>}$ = **zeros** vector, and $x^{<1>}$ = **zeros** vector.
3. choose a **prediction randomly** from distribution obtained by $\hat{y}^{<1>}$. For example it could be "The".
 - In numpy this can be implemented using: `numpy.random.choice(...)`
 - This is the line where you get a random beginning of the sentence each time you sample run a novel sequence.
4. We pass the **last predicted** word with the calculated $a^{<1>}$
5. We keep doing 3 & 4 steps for a fixed length or until we get the $\langle \text{EOS} \rangle$ token.
6. You can reject any $\langle \text{UNK} \rangle$ token if you mind finding it in your output.

• character-level language model

Character-level language model

Vocabulary = [a, aaron, ..., zulu, <UNK>]

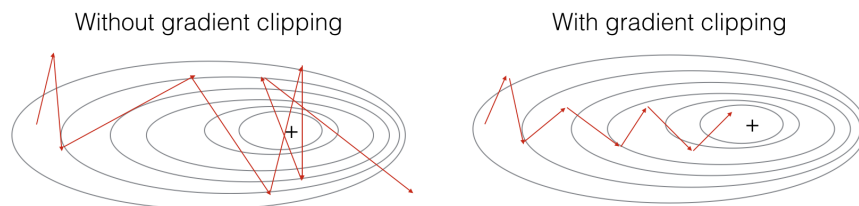
Vocabulary = [a, b, c, ..., z, 1, 2, ..., 9, 0, ..., A, ..., Z]



- Pros:
 - a. no <UNK> token
- Cons:
 - a. end up with much longer sequences.
 - b. not good at capture long-term relationship
 - c. more computationally expensive and harder to train.

. Vanishing gradient

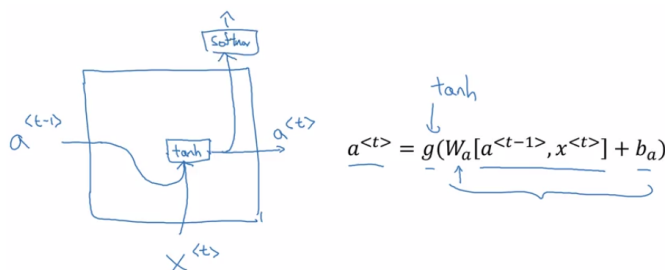
- exploding gradients



- can be easily seen when your weight values become NaN.
- apply **gradient clipping**
 - set threshold
 - if your gradient is more than some threshold - re-scale some of your gradient vector so that is not too big.
- vanishing gradients
 - use new architecture NN

Gated Recurrent Unit GRU

- can remember long-term dependencies
- basic RNN unit



- GRU unit
 - a new variable C which is the memory cell. It can tell to whether memorize something or not.—**Candidate**
 - update formula

Full GRU

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

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

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

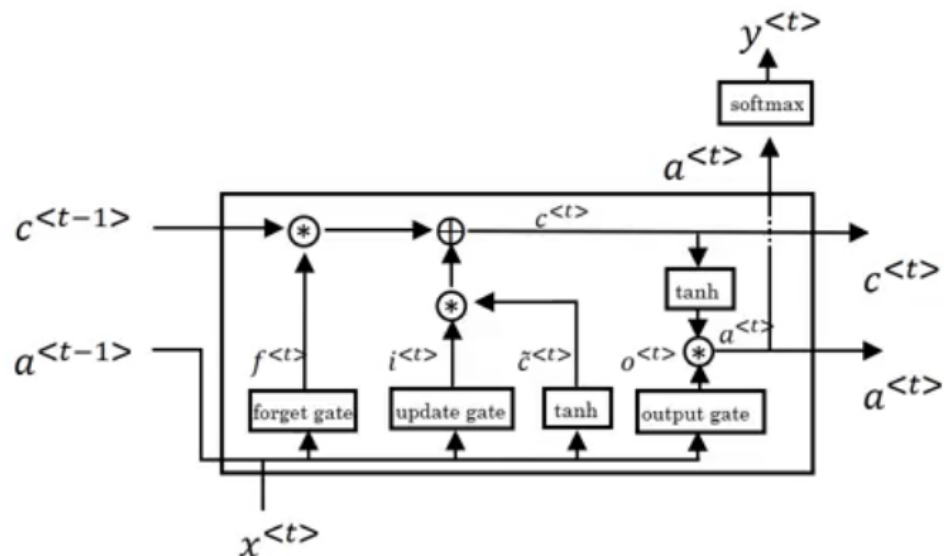
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

The cat, which ate already, was full.

- The full GRU contains a new gate that is used with to calculate the candidate C. The gate tells you how relevant is $C^{<t-1>}$ to $C^{<t>}$
 - Because the update gate U is usually a small number like 0.00001, GRUs doesn't suffer the vanishing gradient problem.

LSTM

- LSTM units



- update formula

$$\begin{aligned}\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \text{(update)} - \Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\ \text{(forget)} - \Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\ \text{(output)} - \Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>}\end{aligned}$$

- wrap up

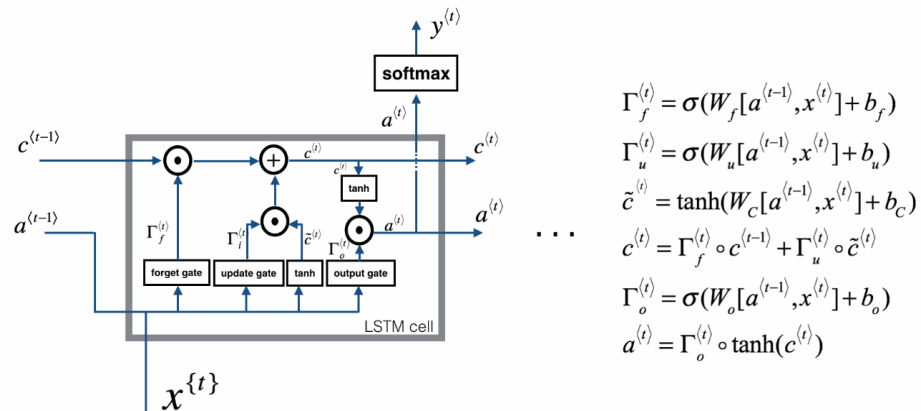
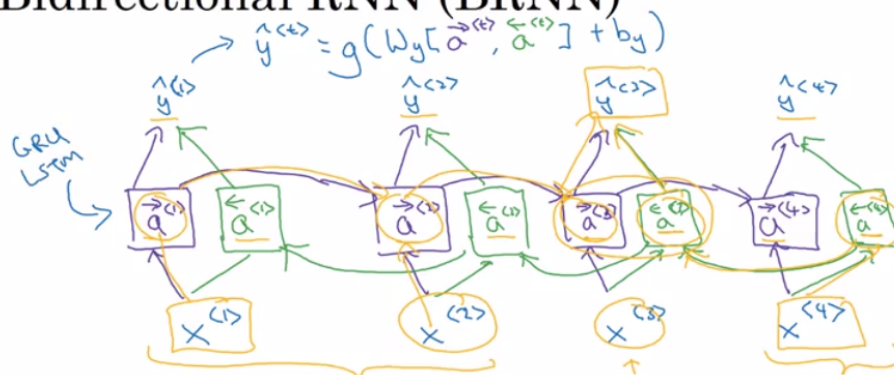


Figure 4: LSTM-cell. This tracks and updates a "cell state" or memory variable $c^{(t)}$ at every time-step, which can be different from $a^{(t)}$.

BiRNN

- **acyclic graph.**
- blocks can be basic RNN or GRU or LSTM

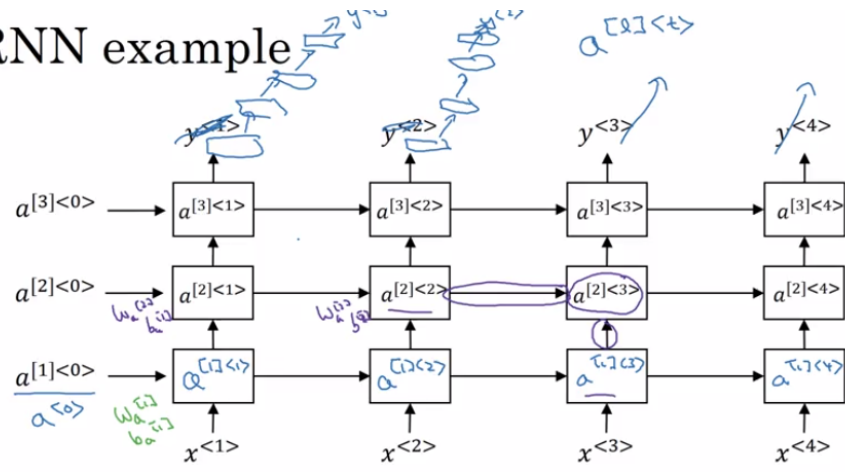
Bidirectional RNN (BRNN)



- pros
 - capture the dependencies from front and post
- cons
 - need entire sequence before process it

Deep RNN

Deep RNN example



$$a^{[2]}<3> = q(W_a^{[2]} [a^{[1]}<2>, a^{[0]}<3>] + b_a^{[2]})$$