Week 1 - Recurrent Neural Networks

Wednesday, September 30, 2020 3:07 PM

① Why Sequence Models

- Used for sequential data

- All of these can be addressed using supervised learning with label x, y as the training set

② Notation

- Motivating example

  → Named entity recognition

  - Outputs the names in a sentence

    X: Harry Potter and Hermoine Granger invented a new spell

    $x^{<1>}$ $x^{<2>}$ ... $x^{<t>}$ ... $x^{<9>}$

    y: 1 1 0 1 1 0 0 0 0

    $y^{<1>}$ $y^{<2>}$ ... $y^{<t>}$ ... $y^{<9>}$

    $T_x = 9$ → length of input

    $T_y = 9$ → length of output

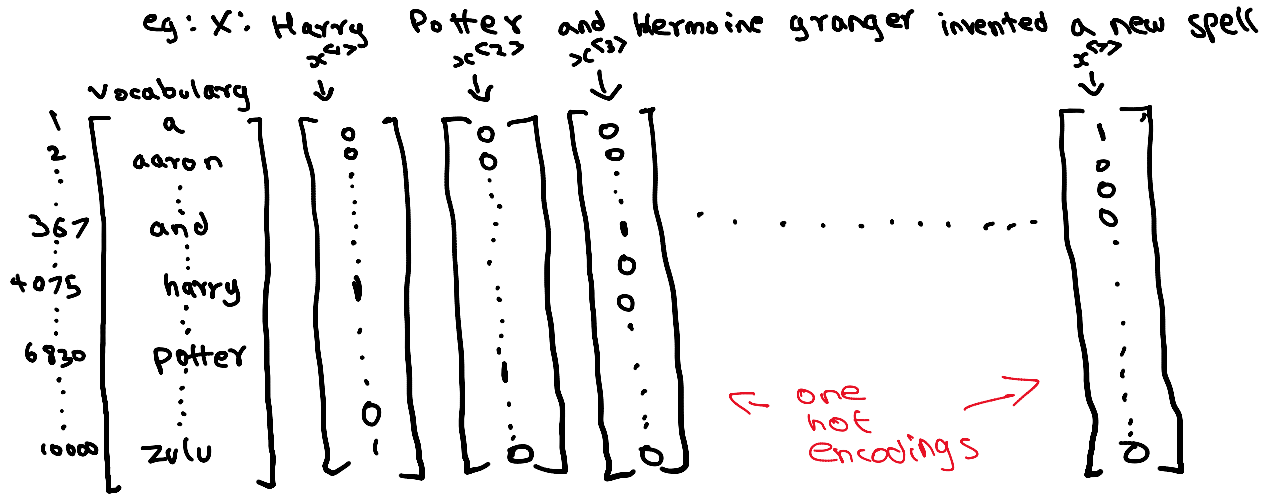
    $\{ x^{(i)<t>}, y^{(i)<t>}, T_x^{(i)}, T_y^{(i)} \}$ for i training examples

  - Representing words

    In NLP, we need a vocabulary list that contains all the words in our target sets

    eg: X: Harry Potter and Hermoine granger invented a new spell

    $x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... $x^{<5>}$ ... $x^{<7>}$

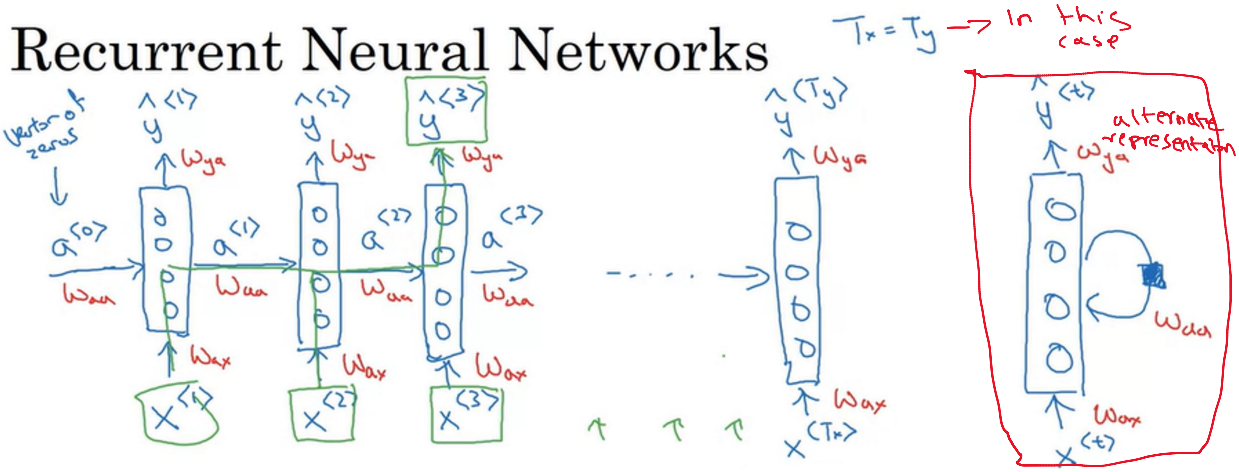


Unknown words could be represented by the same field in the vocabulary

(3) Recurrent Neural Network model

- Why not a standard network
  - → Inputs and outputs differ in length
  - → No shared features learned across different positions of text
  - → one-hot encoded vectors are high dimensional, requiring lot of parameters

- Recurrent Neural Networks



- → $a^{<0>}$ is initialized with zeros, but some may initialize randomly
- → The weights of previous layers are passed on to the subsequent layers
- → Three weight matrices are involved
  - W_{ax} → conventional input

→ Three weight matrices are involved

- W_{ax} → sequential input
  shape = (No. of hidden neurons, n_x)
- W_{aa} → computation of previous layers (memory of the NN)
  shape = (No. of hidden neurons, No. of hidden neurons)
- W_{ya} → sequential output
  shape = (n_y, No. of hidden neurons)

→ output $\hat{y}^{<t>}$ depends on previous inputs and activations

→ Downside is that it can't learn from the elements later in the sequence

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

He said, "Teddy bears are on sale!"

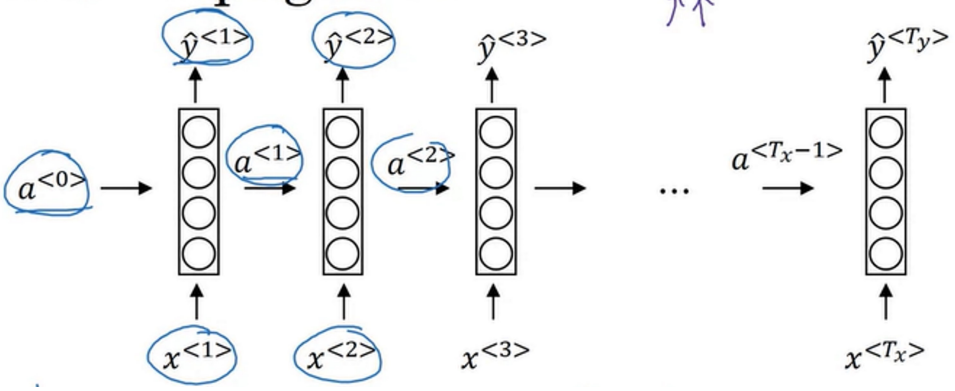
→ cannot output if Teddy is the name of the president

→ This can be addressed using Bidirectional RNN (BRNN)

- Forward propagation

Forward Propagation

$a \leftarrow W_{ax} x^{(i)}$



$a^{<0>} = \vec{0}$

$a^{<t>} = g_1(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a)$ → usually tanh / ReLU

$\hat{y}^{<t>} = g_2(W_{ya} a^{<1>} + b_y)$ → depends on output (sigmoid for entity recog → 2 o/p classes)

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

- Simplified RNN notation

$$a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a) \Rightarrow a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$
$$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$$
$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

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

eg: if $W_{aa} \rightarrow (100, 100)$, $a^{<t-1>} \rightarrow (100)$

$W_{ax} \rightarrow (100, 10000)$, $x^{<t>} \rightarrow (10000)$

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

eg: if $W_{aa} \to (100, 100)$ $\quad a^{<t-1>} \to (100)$

$W_{ax} \to (100, 10000)$ $\quad x^{<t>} \to (10000)$

$100 \updownarrow [\underset{\leftarrow 100}{W_{aa}} \quad \underset{\leftarrow 10000}{W_{ax}}] = W_a \quad (100, 10100)$

↳ stack both vectors

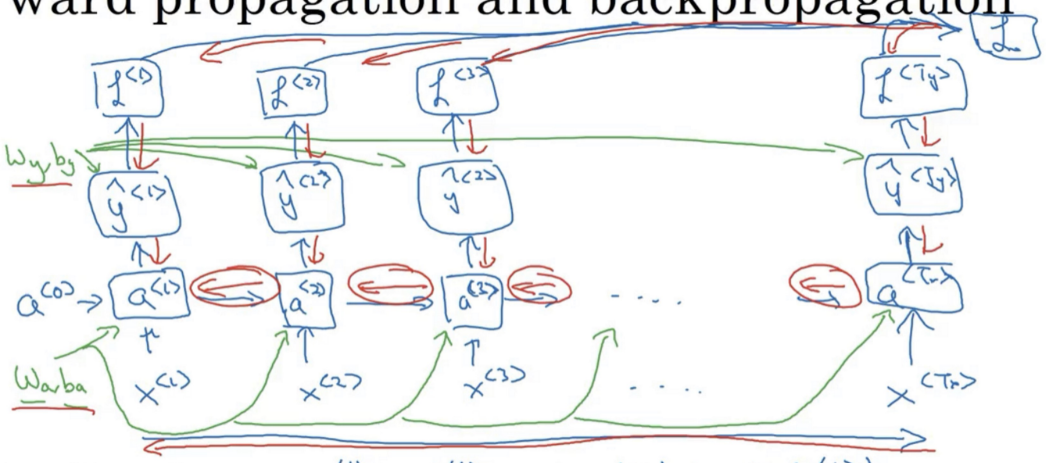
$\Rightarrow [W_{aa} \quad W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$

↳ original eqn.

④ Backpropagation through time

- Implemented by the framework automatically
- computation graph

Forward propagation and backpropagation



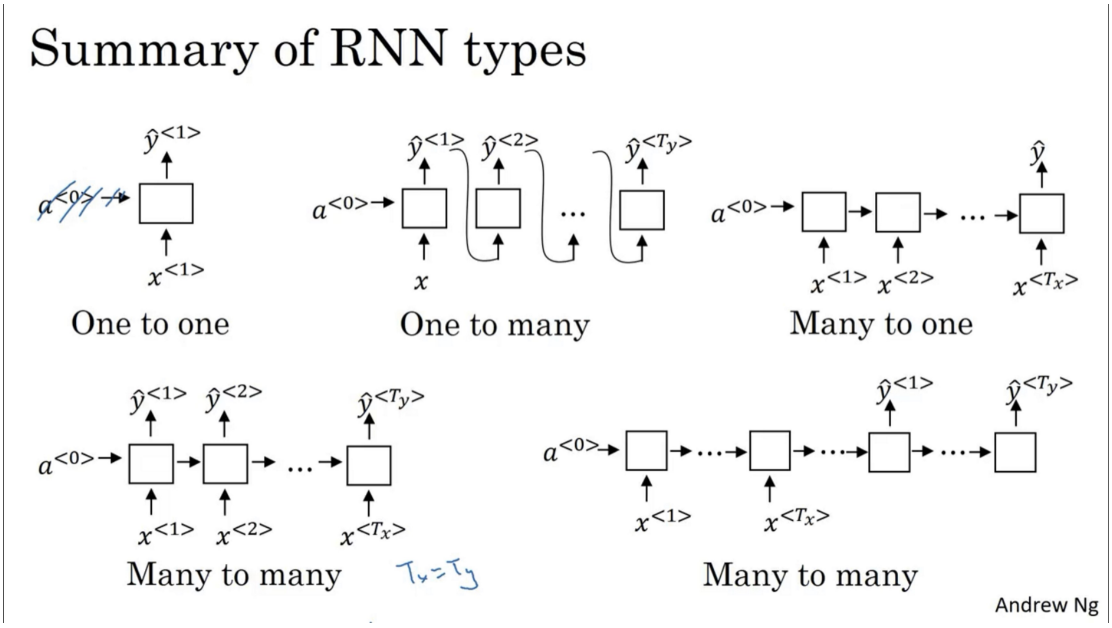
→ W_a, b_a, W_y, b_y are shared across each element in a sequence

→ Cross entropy function is used to compute loss

→ It's called 'backpropagation through time' because we pass activation a from one sequence element to another backwards in time.

⑤ Different types of RNNs

- we've seen RNN for $T_x = T_y$
- Other cases are



- Attention models also exist which is more complex than above architectures

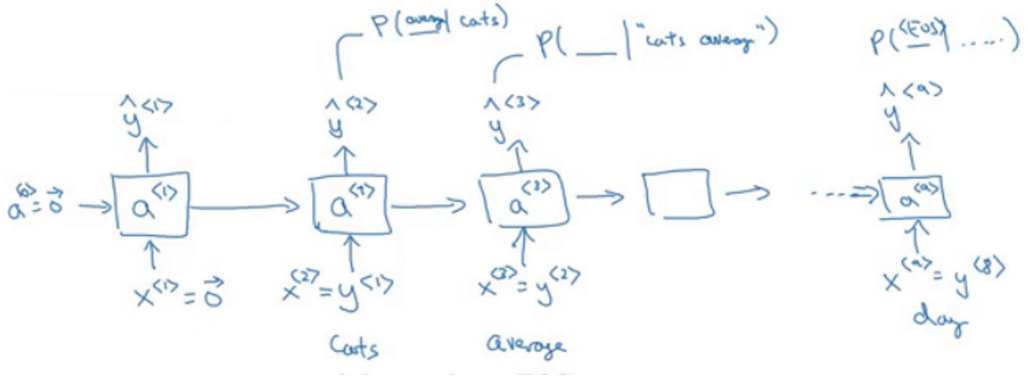
⑥ Language model and seaquence Generation

- Language model
  - → It estimates the probability of that particular seqeuence of words
  - → eg: In speech recognition

    P(The apple and pair salad) = 3.2×10^{-13}
    P(The apple and pear salad) = 5.7×10^{-10}

    Even though these sentences sound alike, the second sentence is 3 orders more probable to occur.
  - → P(sentence) = ? ⇒ $P(y^{<1>}, y^{<2>}, \ldots, y^{<T_y>})$
- Language modelling with an RNN
  - → Training set : Large corpus of english text
  - → We tokenize the input sentence by mapping words to vocabulary set (one-hot encoding)
  - → Add an <EOS> (end of sentence token and <UNK> token for unknown words
  - → we also set $x^{<t>} = y^{<t-1>}$
- RNN model

→ For the sentence "cats average 15 hours of sleep a day"



→ Loss function is the cross entropy loss

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

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

i → all elements in corpus
t → for the entire sequence

→ To use the model

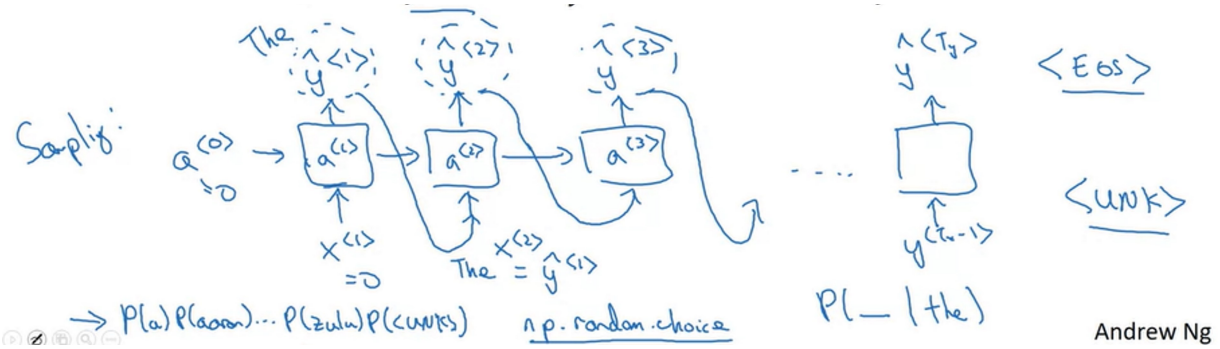
- To predict next word given the previous words, we feed the sentence and get the final $\hat{y}^{<t>}$ hot vector and sort by maximum probability
- To compute probability of a sentence with 3 words

$$P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>}) * P(y^{<2>} | y^{<1>}) * P(y^{<3>} | y^{<1>}, y^{<2>})$$

⇒ we multiply probability of occurence of each word to get probability of occurence of the sentence

⑦ Sampling novel sequences

- To check what a language model has learned, we can try to sample novel sequences.
- To sample a novel sequence

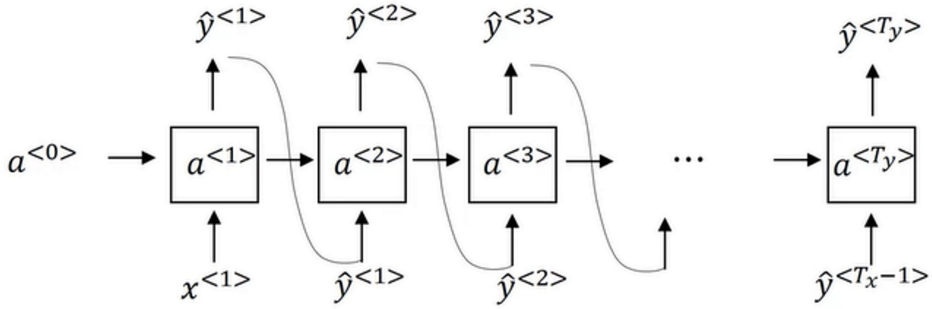


→ we pass $a^{<0>}$ = zeros & $x^{<1>}$ = zeros
→ Random word from output $\hat{y}^{<1>}$ is chosen as the first word
→ The predicted word is passed as $x^{<2>}$
→ The above 2 steps are repeated until we get <EOS> or after the required no. of iterations
→ <UNK>s can be removed

- Character-level language model

Vocabulary = [a, b, c, ..., z, ␣, ., ,, ;, 0, ..., 9, A, ..., Z]

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ Cat average ... May



→ Pros
- No unknown token

→ Cons
- much longer sequences
- Not good at long range dependencies in a sentence
- computationally more expensive and harder to train

⑧ Vanishing gradients with RNNs

- RNNs run into vanishing gradient problems because of extremely long term dependancies in NLP.

eg: The cat, which already ate ..... (arbitrary no. of words) was full
The cats, which already ate ..... were full

The rat/cats affects was/were over

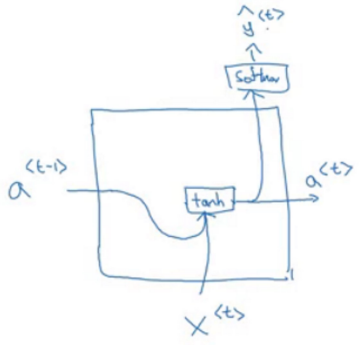
The cats, which already ate . . . , were full

The rat/cats affects was/were over a long range

- RNNs are not very good at learing long term dependencies
- similar to deep neural networks, we run into vanishing gradient problems with deeper networks or RNNs with large sequence size.
- Exploding gradients also happen but can be easily identified as they cause a lot of NaNs. They are easily fixed by gradient clipping

⑨ Gated Recurrent Unit (GRU)

- Basic RNN unit



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

(tanh)

- Each layer in a GRU has a **variable** c which is the memory cell

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

↳ memory cell

- $\tilde{c}^{<t>} = \tanh\left(w_c\left[c^{<t-1>}, x^{<t>}\right] + b_u\right)$

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

↳ update gate

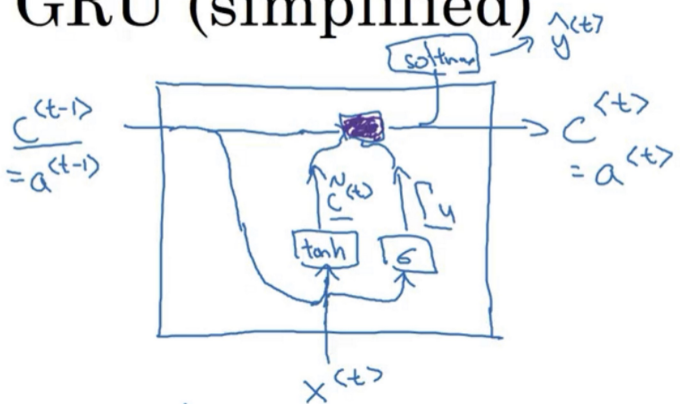
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$

→ update gate is from 0 to 1 (mostly 0 or 1)

→ we update c based on update cell and previous cell

→ It updates when the memory cell is no longer needed

GRU (simplified)



Shapes:

$a^{<t>} \rightarrow$ (No of hidden Neurons, 1)

$c^{<t>}$ & $\tilde{c}^{<t>}$ same as $a^{<t>}$

$u^{<t>}$ also same as $a^{<t>}$

- Full GRU

Full GRU

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

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

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

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

→ It contains an extra gate that is used to calculate candidate c

→ Tells how relevant $c^{<t-1>}$ is to compute $c^{<t>}$

(10) Long Short Term Memory (LSTM)

- more powerful and generalised than GRU
- In LSTM $c^{<t>} \neq a^{<t>}$
- Equations

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

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

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

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

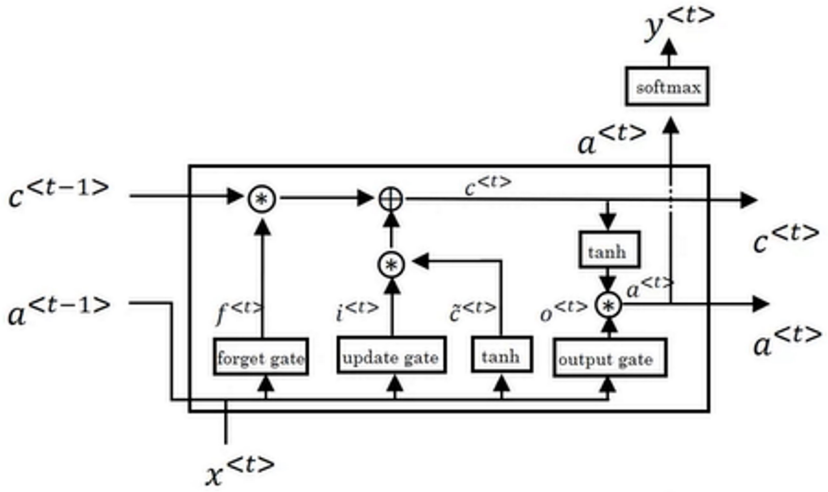
forget gate ← $\Gamma_f = \sigma(W_f [a^{<t-1>}, x^{<t>}] + b_f)$

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

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

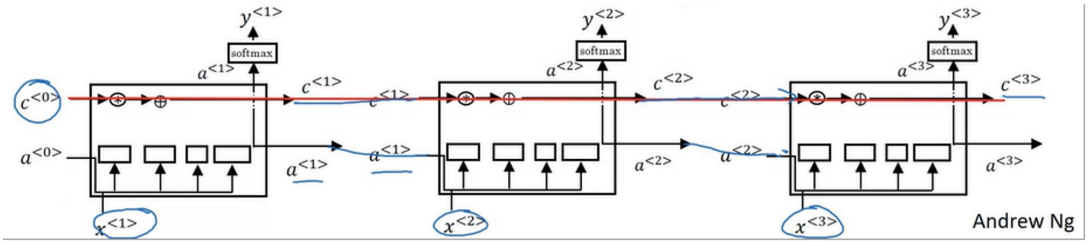
$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

- LSTM in picture



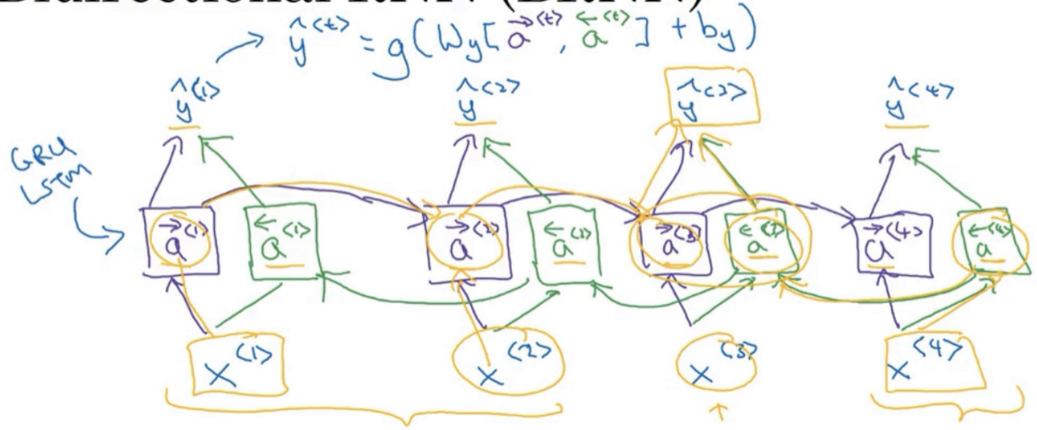
→ Some variations of LSTMs include 'peephole connections' which include $c^{<t-1>}$ in every gate instead of just $a^{<t-1>}$

- When LSTMs are chained together, a straight connection can still be made from $c^{<0>}$ to $c^{<3>}$. This allows it to remember long term dependencies



(11) Bi-directional RNN

Bidirectional RNN (BRNN)



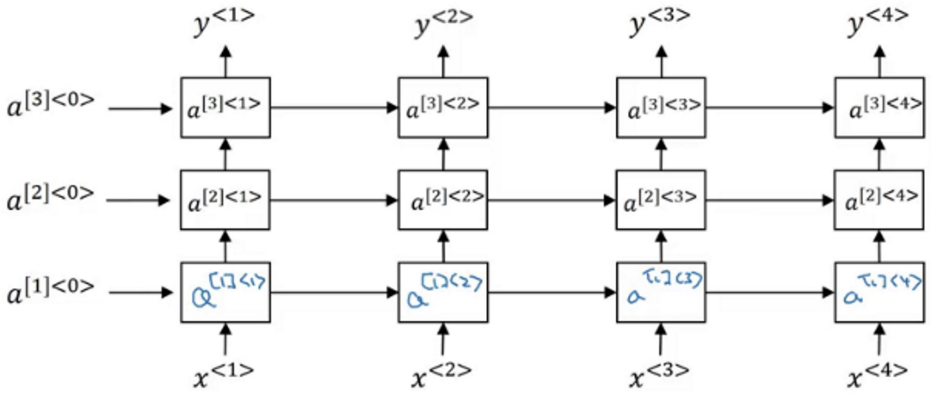
→ Acyclic graph
forward propagation goes left to right and right to left since it learns from both sides

→ The blocks can be basic RNN, GRU or LSTMs

→ Usually BRNN with LSTM is used for many NLP tasks

→ Disadvantage is that it requires entire sequence to make predictions. Hence, it cannot be used in real time tasks.

(12) Deep RNNs



→ RNN with 3 hidden layers

→ Unlike feed-forward neural networks, in deep RNNs stacking 3 layers is considered deep and expensive to train

→ In some models, feed forward neural networks are connected after recurrent

→ In some models, Feed forward neural networks are connected after recurrent cell