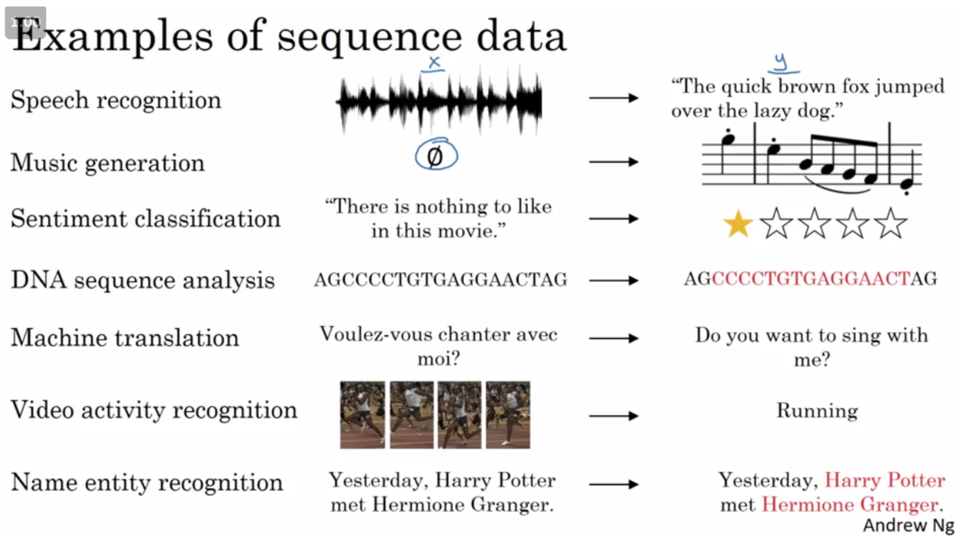
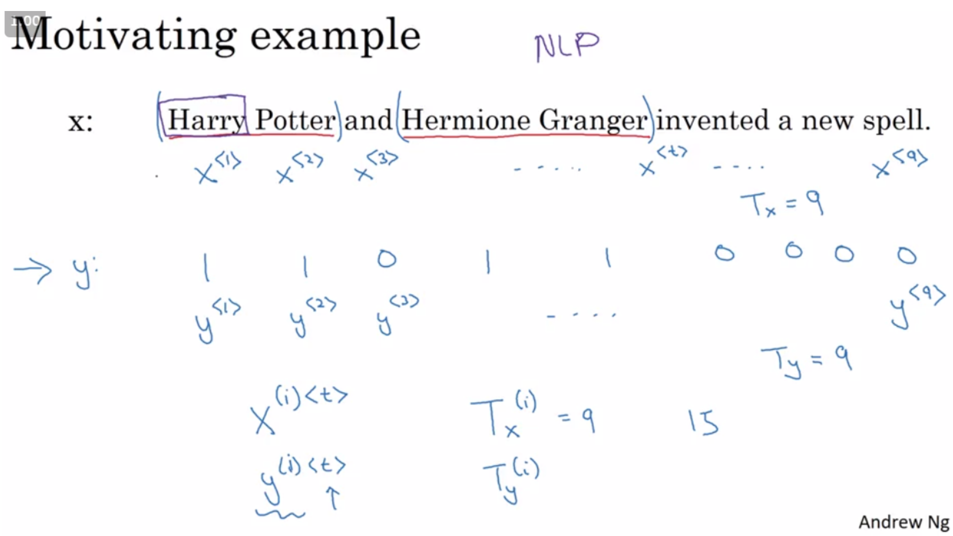
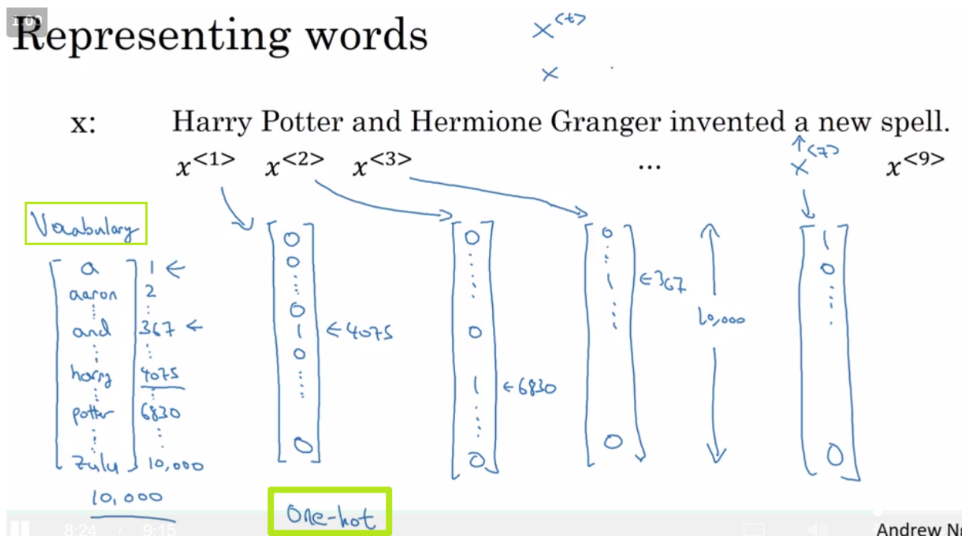
Created: 19/03/18 10:21 **Updated:** 19/03/18 21:43

Sequence model

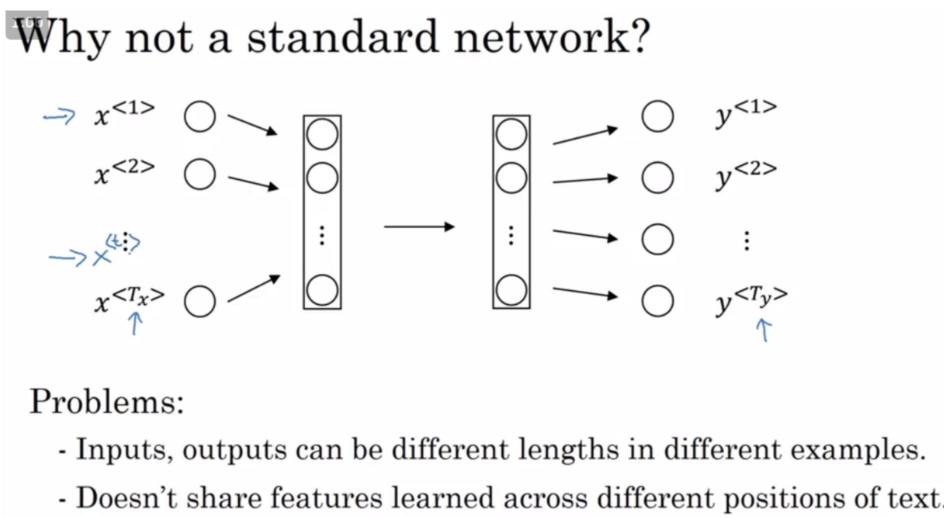
- see what it can do!
  - 

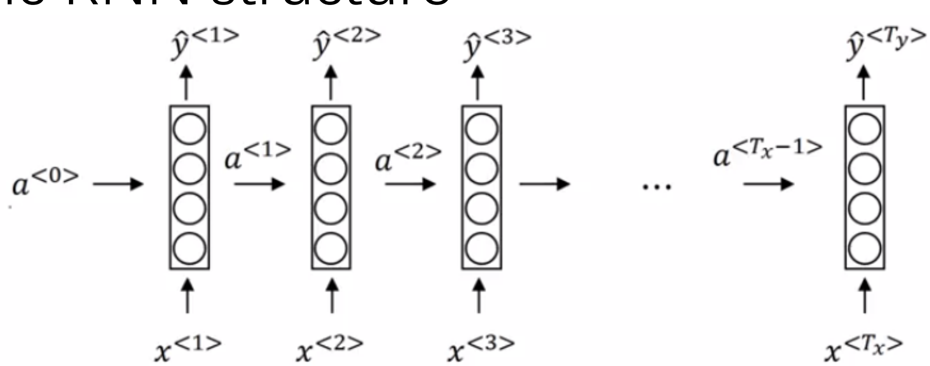
- notation
  - 

  - 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**

- 

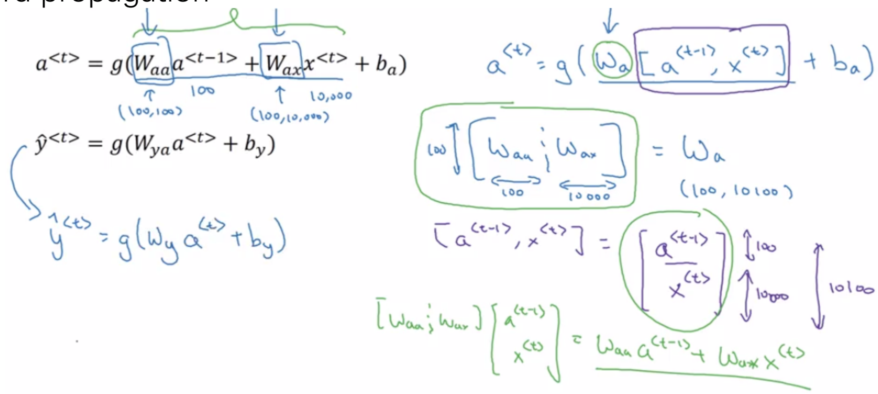
RNN

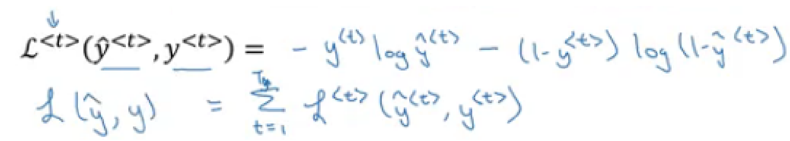
- why use RNN to tackle with sequence tasks?
  - 

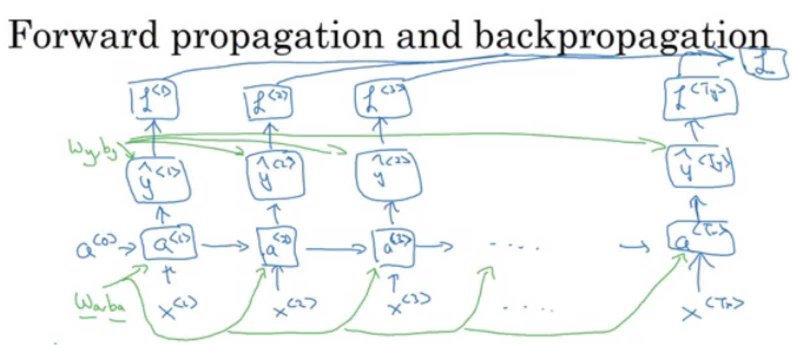
- basic RNN structure
  - 

  - 3 kinds of matrices
    1. Wax: (HiddenNeurons, nx)
    2. Waa: (HiddenNeurons, HiddenNeurons)
       - use to maintain memory

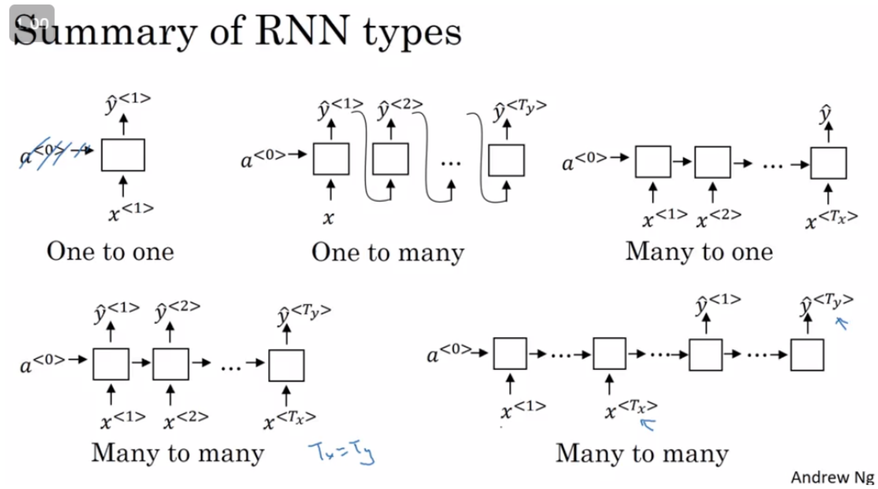
3. Wya: (ny, HiddenNeurons)

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

  - 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
    - 

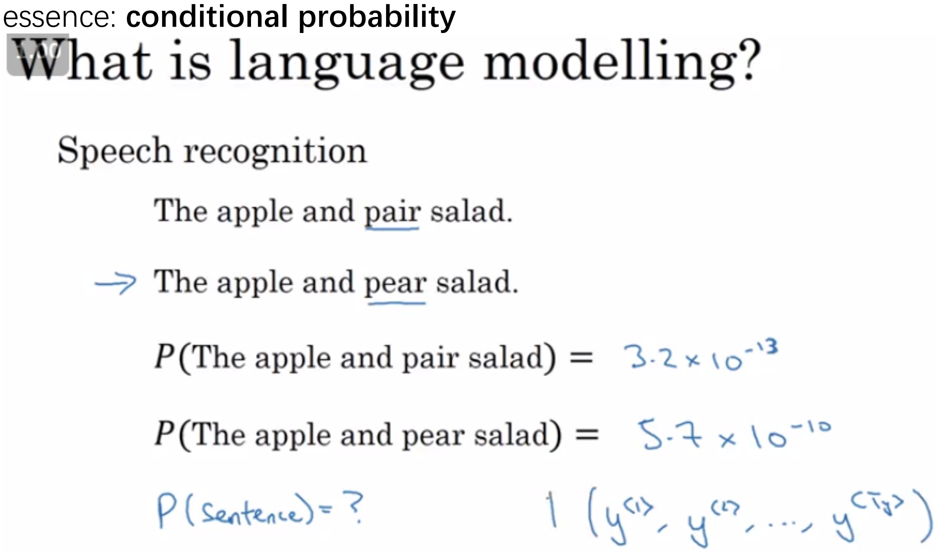
  - diagram
    - 

- # different kinds of RNN

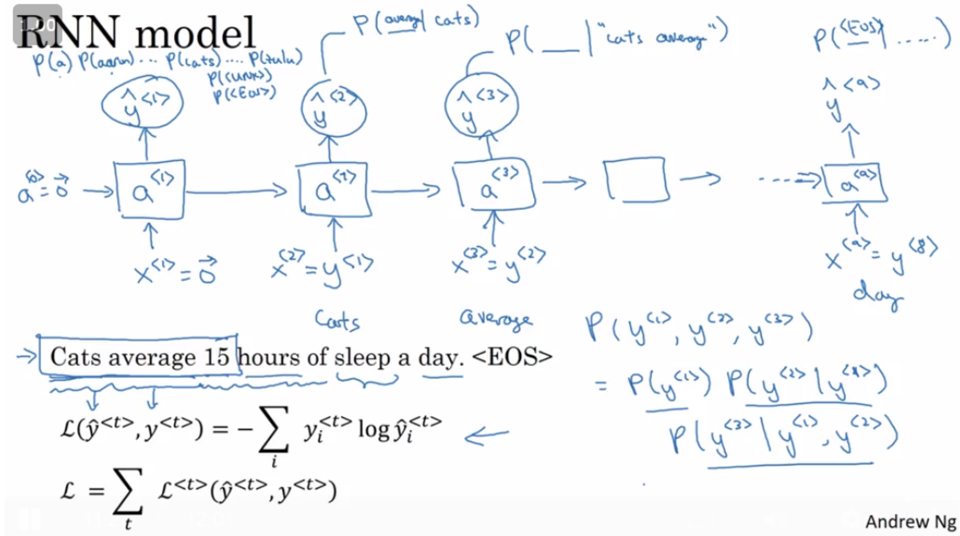
- 

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

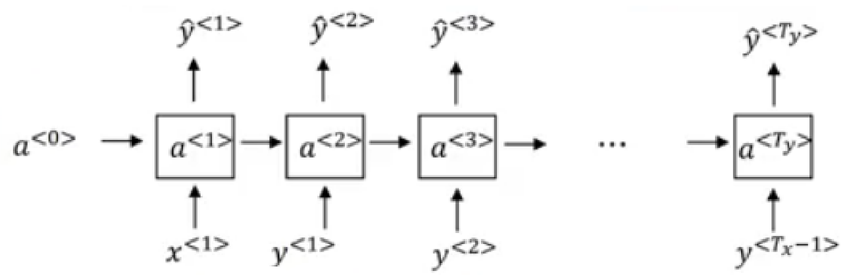
Language model

- 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**
    - 

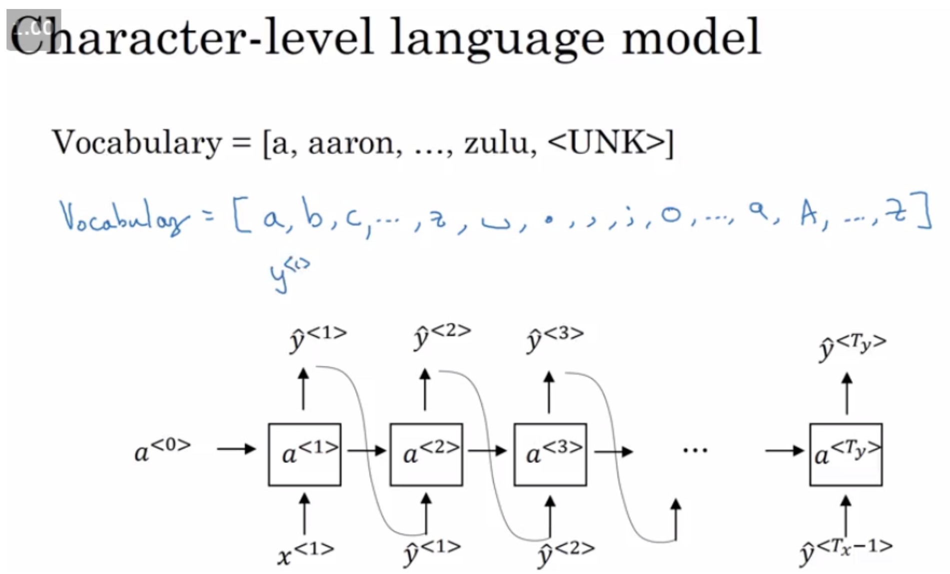
- how to build?
    - get a **training** set: a large corpus of target language text.
    - **tokenize** them by getting the vocabulary and then one-hot each word.
    - add token <EOS> at an end of sentence
- whole RNN:



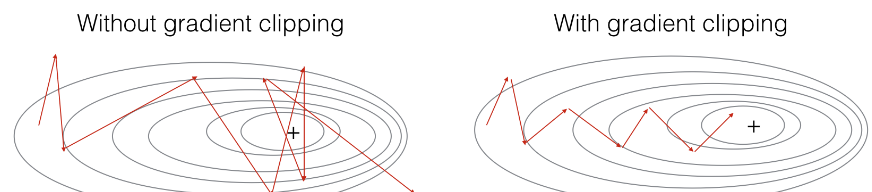
- sampling novel sequences
  - After a sequence model is trained on a language model, to check what the model has learned you can apply it to sample novel sequence.
  - process:
    1. Given this model:



    2. pass $a^{<0>}$ = **zeros** vector, and $x^{<1>}$ = **zeros** vector.
    3. choose a **prediction randomly** from distribution obtained by ŷ<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 <EOS> token.
    6. You can reject any <UNK> token if you mind finding it in your output.
- character-level language model

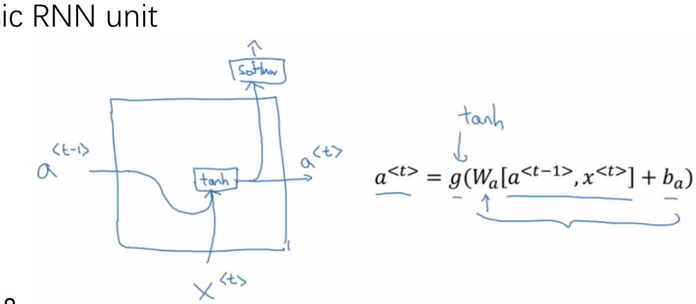
- 

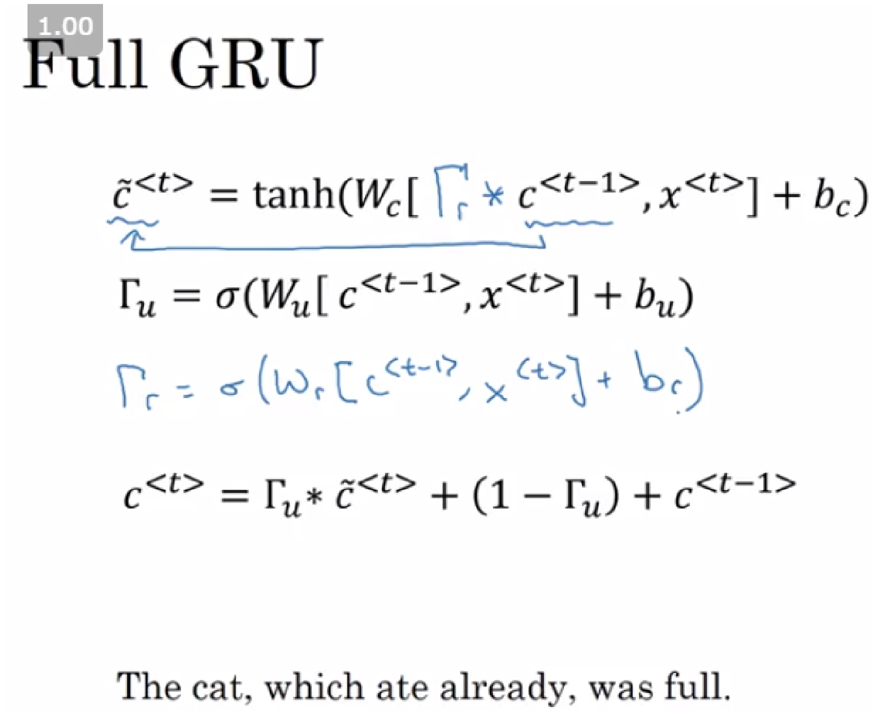
- Pros:
    a. no <UNK> token
- Cons:
    a. end up with much longer sequences.
    b. not good at capture long-term relationship
    c. more computationally expensive and harder to train.

- # Vanishing gradient
    - exploding gradients
        - 

        - can be easily seen when your weight values become NaN.
        - apply **gradient clipping**
            - set threshhold
            - 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

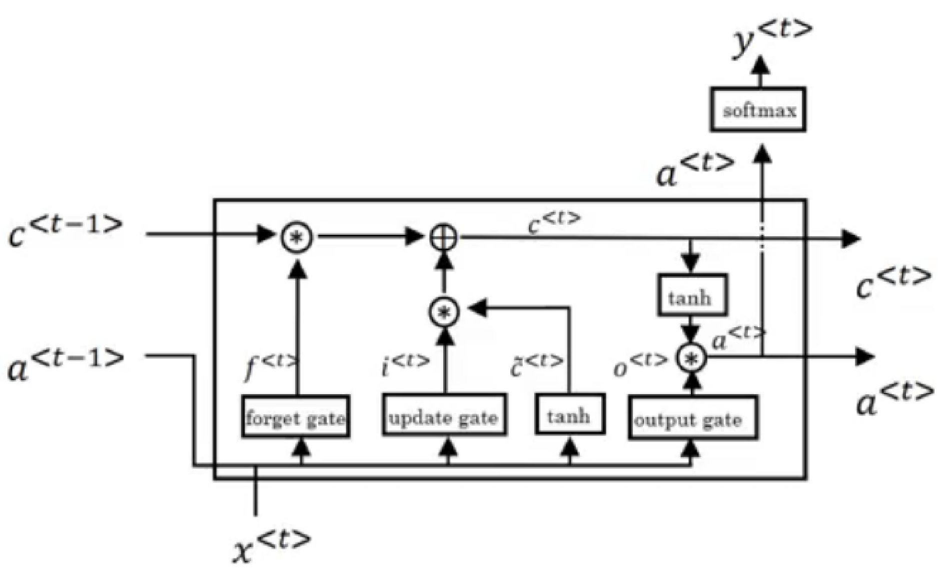
Gated Recurrent Unit GRU

- can remember long-term dependecies
- 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
    - 

    - 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

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

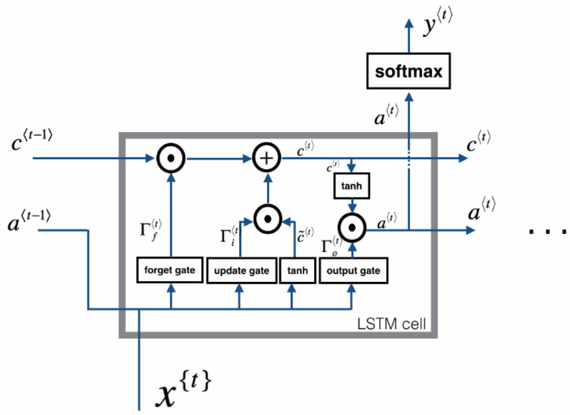
(update) $$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

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

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

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

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

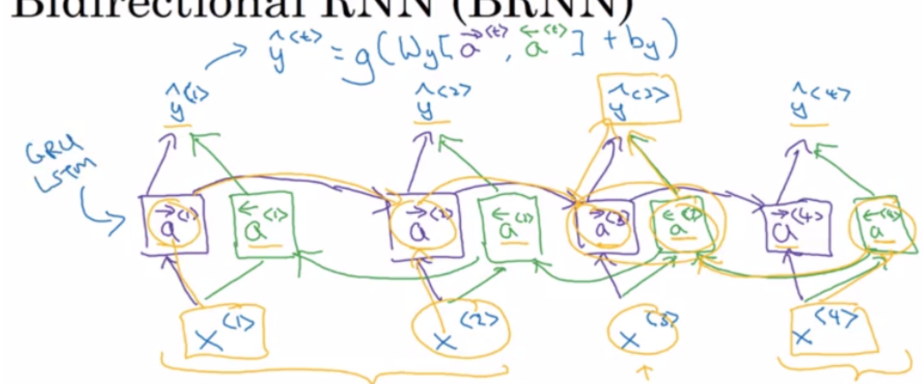
- wrap up
    - 

$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$
$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$
$$\tilde{c}^{\langle t \rangle} = \tanh(W_C[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_C)$$
$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} \circ \tilde{c}^{\langle t \rangle}$$
$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$
$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} \circ \tanh(c^{\langle t \rangle})$$

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

BiRNN

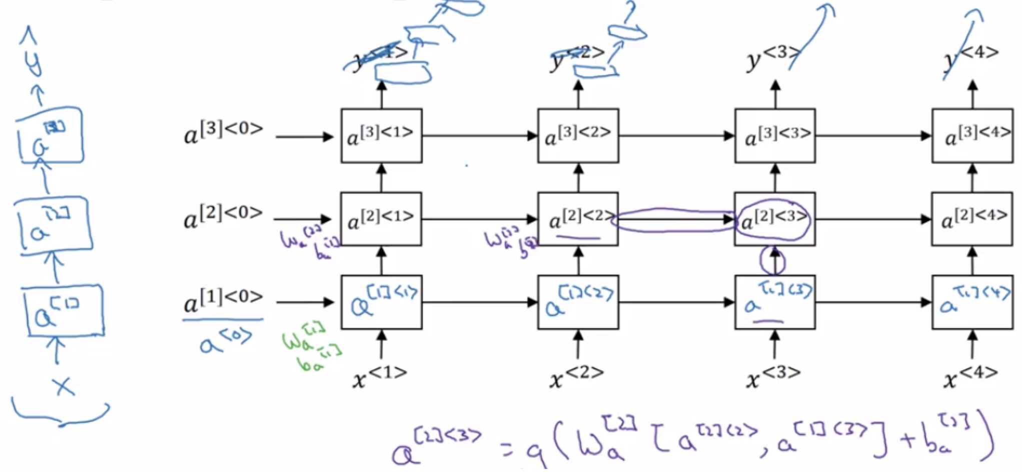
- **acyclic graph.**
- blocks can be basic RNN or GRU or LSTM
    - Bidirectional RNN (BRNN)

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

Deep RNN

Deep RNN example



-