Notes for Week1 Sequencial model

author: Lu KONG
course site: here
Professor: Andrew Ng

Outline:

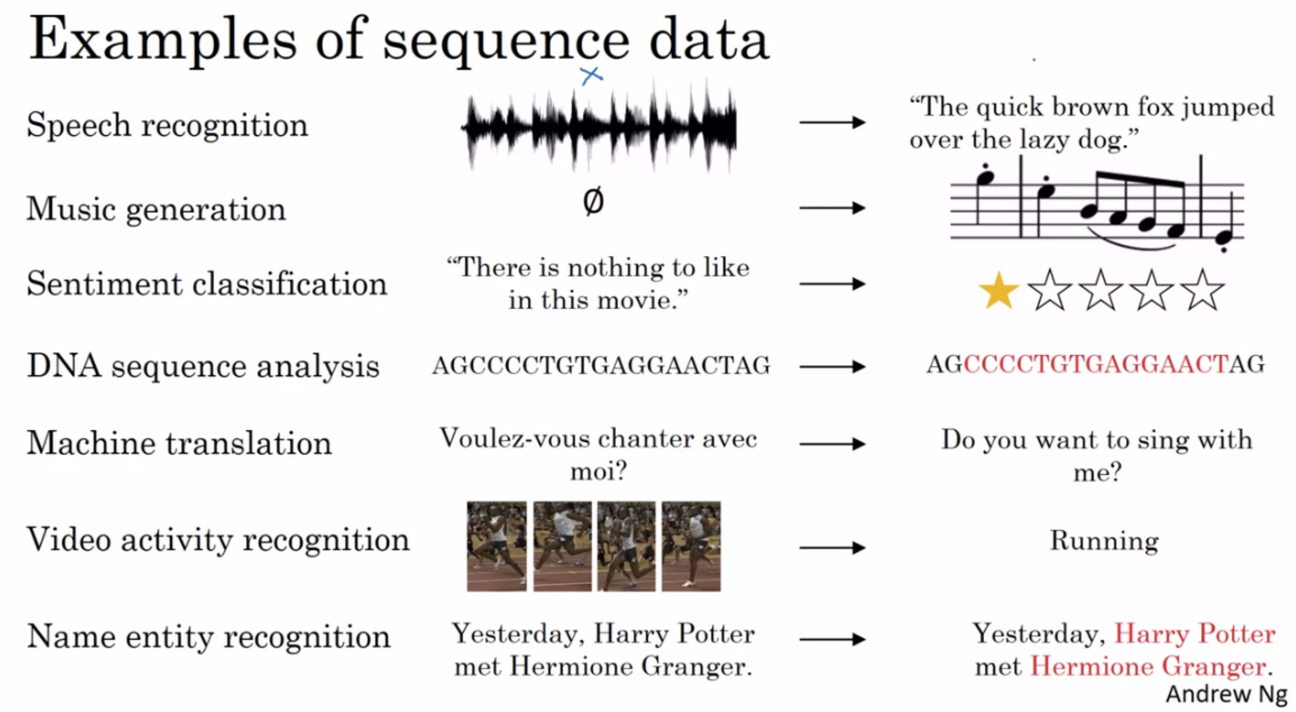
Recurrent Neural Networks

1. Introduction

1.1 Examples of sequence data:

input or(and) output in form of sequence

- speech recognition
- Music generation
- Sentiment classification
- DNA sequence analysis
- Machine translation
- Video activity recognition
- Name entity recognition



1.2 Notation

- to possition `i` of the sequence x, we denote:

$$x^{<i>}$$

- the same for the output y:

$$y^{<i>}$$

- for i^{th} training example, we denote the t^{th} position of the input and output as:

$$x^{(i)<t>}$$
$$y^{(i)<t>}$$

- We denote the length of the sequence as a stopping time:

$$T_x^{(i)}$$
$$T_y^{(i)}$$

  , which denote the stopping time of the i^{th} input x and output y

1.3 How to represent the words in the sentense

1. Firstly we need our `vocabulary` or `dictionary`
2. Use a `one-hot` presentation to represent a `word` in a vecter:
   `0` for all except `1` for the sequencial number of the `word`
3. `token` for unknown word

2. Standard network:

Network has $T_x^{(i)}$ inputs and $T_y^{(i)} = T_x^{(i)}$ outputs `correspond` with each other.

Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share feathers learned across differents of texts. (`too independent`)

  We hope that the model can share its experience: When first learnt a word as a `nom`, it will have some experience when it see it in other position that this word could be a `nom`. This is also for reduce the number of parameters

3. Recurrent Neural Network

3.1 Composition

`Instead of` only using just the input of this position, the RNN will take the `activations` of `previous position` as inputs together to predict the output of the current position.

Add vector of zeros as $a^{<0>}$ so that the first position works as laters.

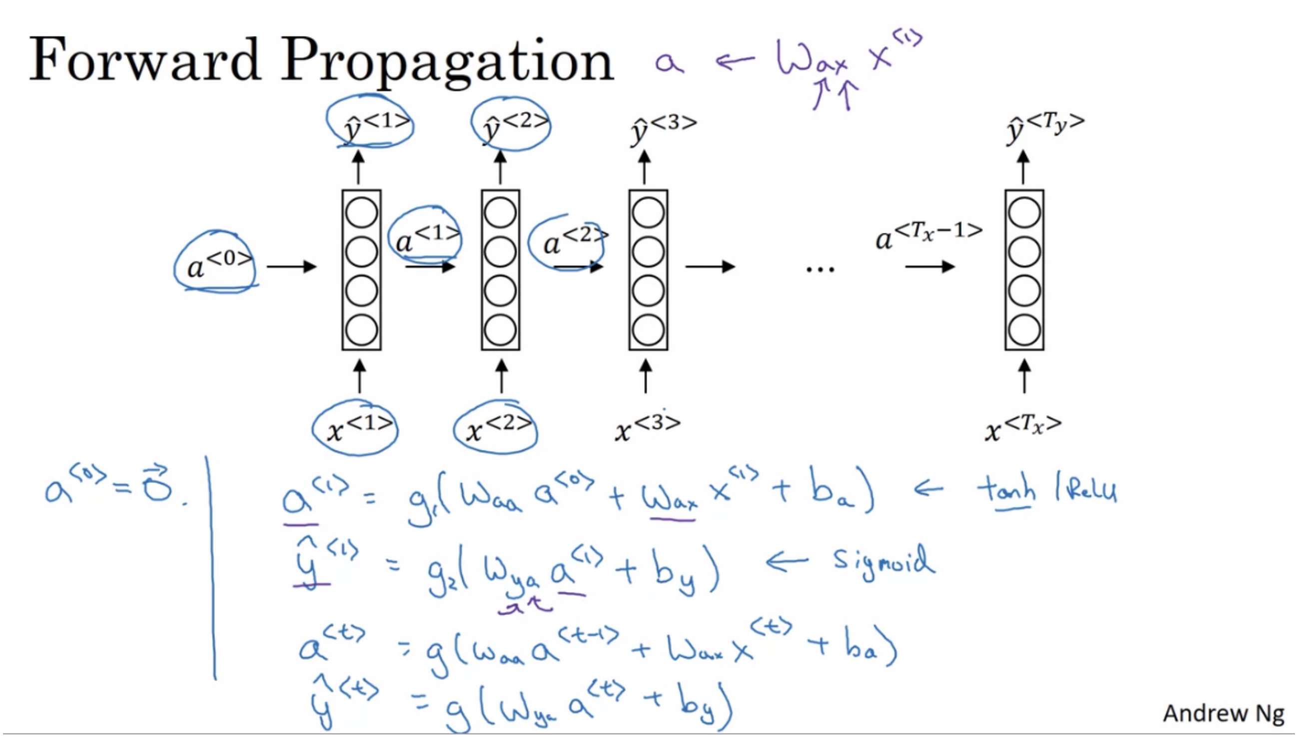
We can denote parameters of this network in three types:

- ω_{ax} : for parameters work with input $x^{<i>}$
- ω_{aa} : for parameters work with input $a^{<i-1>}$
- ω_{ya} : for parameters works with activation $a^{<i>}$ to get $\hat{y}^{<i>}$ as output.

3.2 Problem

- Problem: It looks only these parameters of previous positions but `not later positions`, which is also very important `linguisticly` in natural language.
- Solution: Bidirectional RNN (`BRNN`)

3.3 Forword Propagation



Compress parameters

Originally the fomules wrote as follow :

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

By stacking the inputs vectors $a^{<t-1>}$ and $x^{<t>}$ we get the input

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

note as $\frac{A}{X}^{<t>}$.

And the parameters W_{aa} and W_{ax} are transfered as:

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

So that,

$$W_a \frac{A}{X}^{<t>} = [W_{aa} \quad W_{ax}][\begin{matrix} a^{<t-1>} \\ x^{<t>} \end{matrix}]$$
$$= W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

and we rewrite the fomulations as :

$$a^{<t>} = g(W_a \frac{A}{X}^{<t>} + b_a)$$
$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

3.4 Backpropagation through time

important : former parameters infect all later activations by recurrenting

Loss function:

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

Backpropogation through time

3.5 Different types of RNNs

There existes cases where $T_x = T_y$.

- `Many to Many`
- `Many to One`
  ex : Sentiment classification
- `One to Many`
  ex: Music generation

`One to One`
Just the normal problem without sequence.

The most often used structure is of cause : `Many to Many`

For example, when we do `machine translation`, sentences in different languages can have different length.

4. Language model and sequence generation

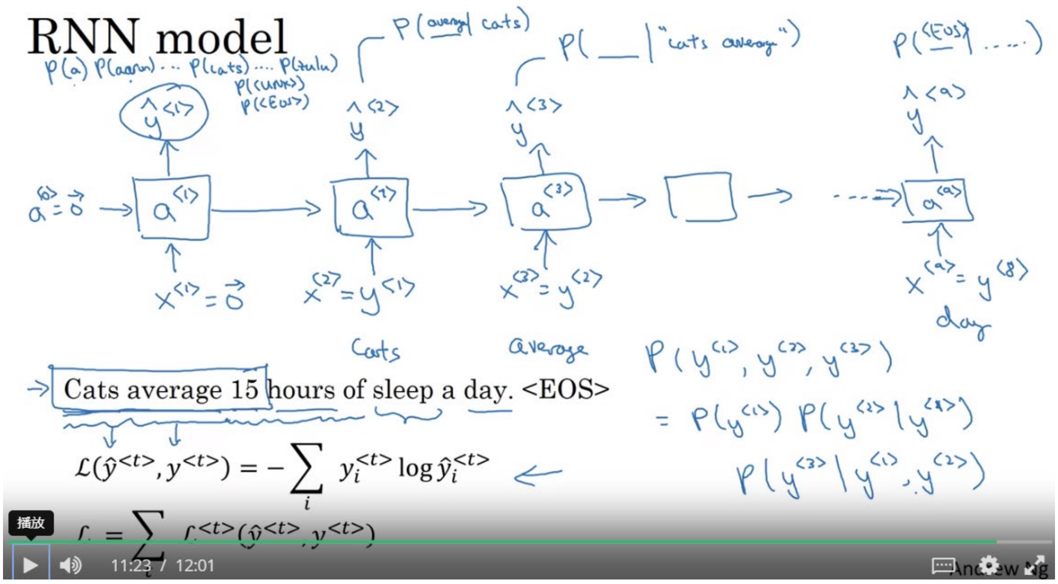
How to build a language model

Speech recognition: give every sentence a probability

Output only sentence that is likely right.

4.1 Language modelling with an RNN

- Traning set : Large corpus of English text. Tokenize
  - `<EOS>` token : End of Sentense
  - `<OOV>` token : `<UNK>` un known or `Out of Vocabulary`
- Model:
  - output softmax
  - 前瞻模型， 看到之前所有的序列。
  - That is:
    - Each time when we try to predict the probability at a position $\hat{y}^{<t>}$,
    - we get the activation of the former position $a^{<t-1>}$
    - and also the **`real value`** of the former position $y^{<t-1>}$



4.2 Sampling novel sequences

1. Randomly sample according to the `softmax` distribution
2. Take the `predicted output` $\hat{y}^{<t>}$ instead of the `real value` $y^{<t>}$ as the input of the next

position until the end of the sentence. (`<EOS>`)

The same time we `refuse` all the sample that contains `<OOV>`

3. So that we do the prediction of the full sentence as a `sample of predicion` based on our former predicions

Word-level = Caracter-level rnn

5. Problems with RNN

Vaninishing gradients!

Raison: fail to deel with long range depencency

5.1 Exploding gradients with RNN

- Exploding gradients : NaN
- Solution: `Gradient clippling`
  That means, look at the gradient vectors, if it's `too big` then we make a `rescedule` so that we can continue to propogate. There are `clips` according to some maximum value.
  This is a relatively robust solution to the problem of `exploding gradients`

5.2 Vaninshing gradients with RNN

Long term RNN model to envite Vanishing gradients

We will use `Gated Recurrent Unit` (GRU)

Which is introduced by Cho et al., 2014 and Junyoung Chung et al., 2014

memory cell c^T

Use some sigmoid-like Gama function Γ_u to denote the gate function which decide wether we will update our memory $c^{<t>}$

$$
\begin{aligned}
\tilde{c}^{<t>} &= \tanh(\Gamma_r * W_c{}_{X^{<t>}}^{C^{<t-1>}} + b_c) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \\
a^{<t>} &= c^{<t>}
\end{aligned}
$$

where,

$$
\{ \begin{aligned}
\Gamma_u &= \sigma(W_u{}_{X^{<t>}}^{C^{<t-1>}} + b_u) \\
\Gamma_r &= \sigma(W_r{}_{X^{<t>}}^{C^{<t-1>}} + b_r)
\end{aligned}
$$

6. Long Short Term Memory (LSTM)

Introduced by Hochreiter & Schmidhuber 1997

Difference between GRN and LSTM:

- Instead of using gate function

- update gamma : Γ_u
- relevent gamma : Γ_r

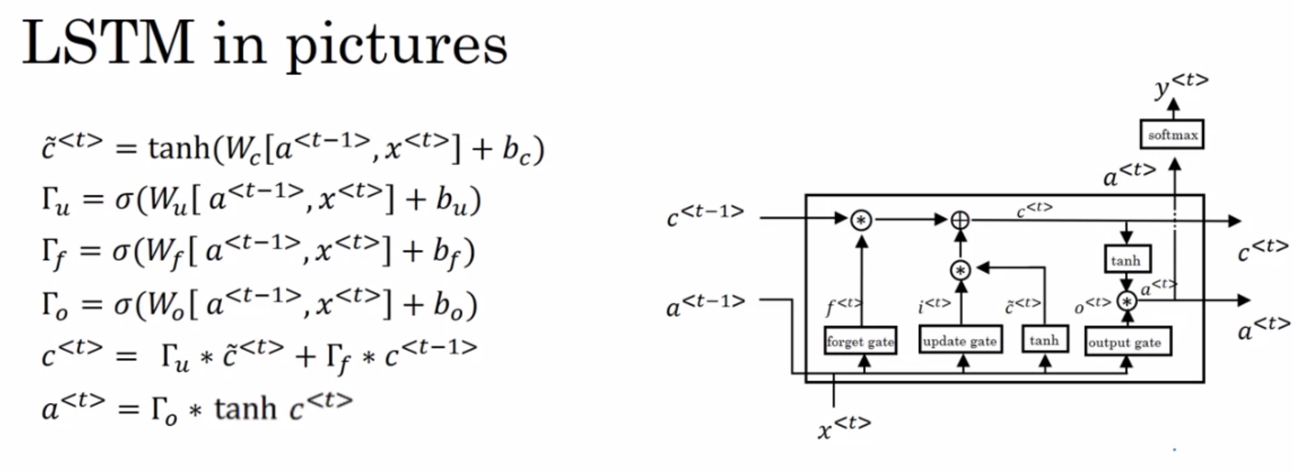
- We introduce in LSTM:
  - update gamma : Γ_u
  - `forget` gamma : Γ_u
  - output gate : Γ_o
- Then we deduce $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
  Instead of $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$
- Finally we deduce the activation with the output gate:

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

Instead of

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

Let's look at it in picture:



peephole connection: Use also $c^{<t-1>}$ as the input of

$$\Gamma_{u,f,o}$$

`GRU` use `smaller` model, and is usually much `easier` to train, but since is expericaly proved with high stability, we often try `LSTM` first as the default model for this kind of `long range memory problems`.

<!-- GRN	LSTM
\s	sf -->

7. Introduction to some other mentioned RNNs

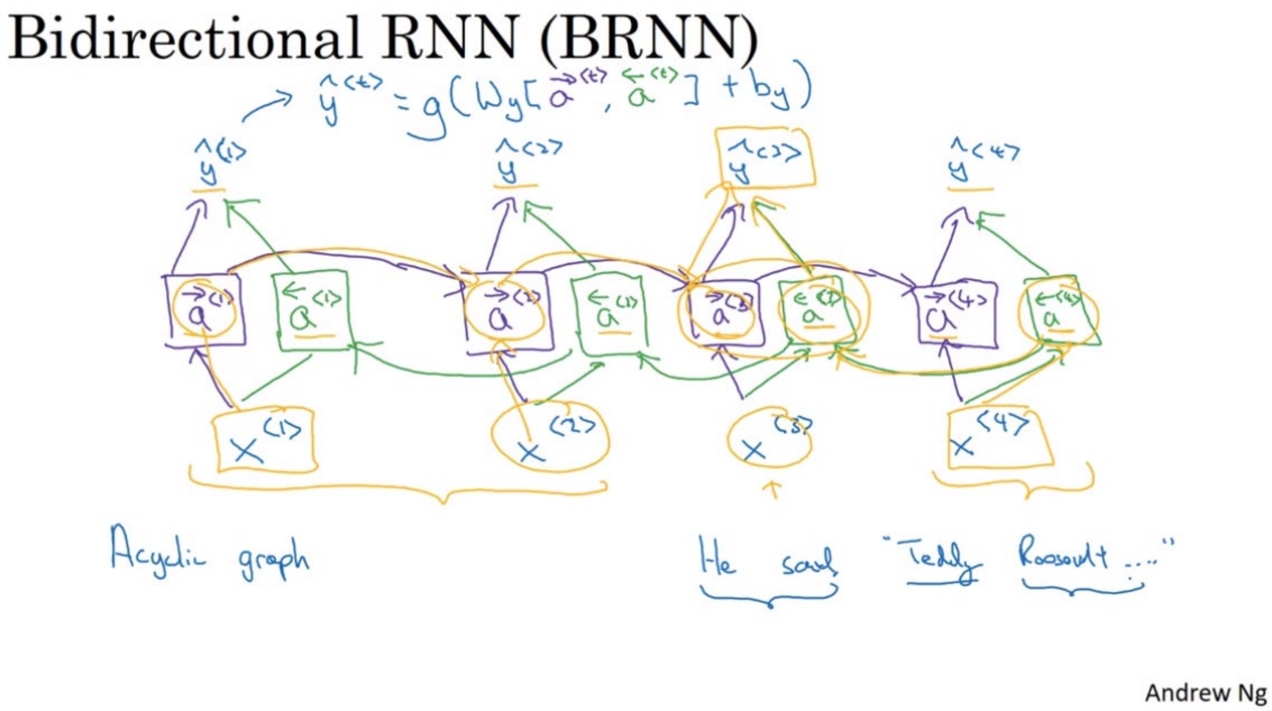
As we were seen, traditionnal RNNs take only one `forward direction` of positions to update. That works generally well `may` still meets some problems because in `natural language-like text`, later relations make sense also for the meaning.

7.1 Bidirectional RNN

We can get the infomation from the future!

In fact, we read a text of listen to some textse, by considering the context, that is, the text `before` and `after`.

Add in the network some backward activation
Acyclic graph :



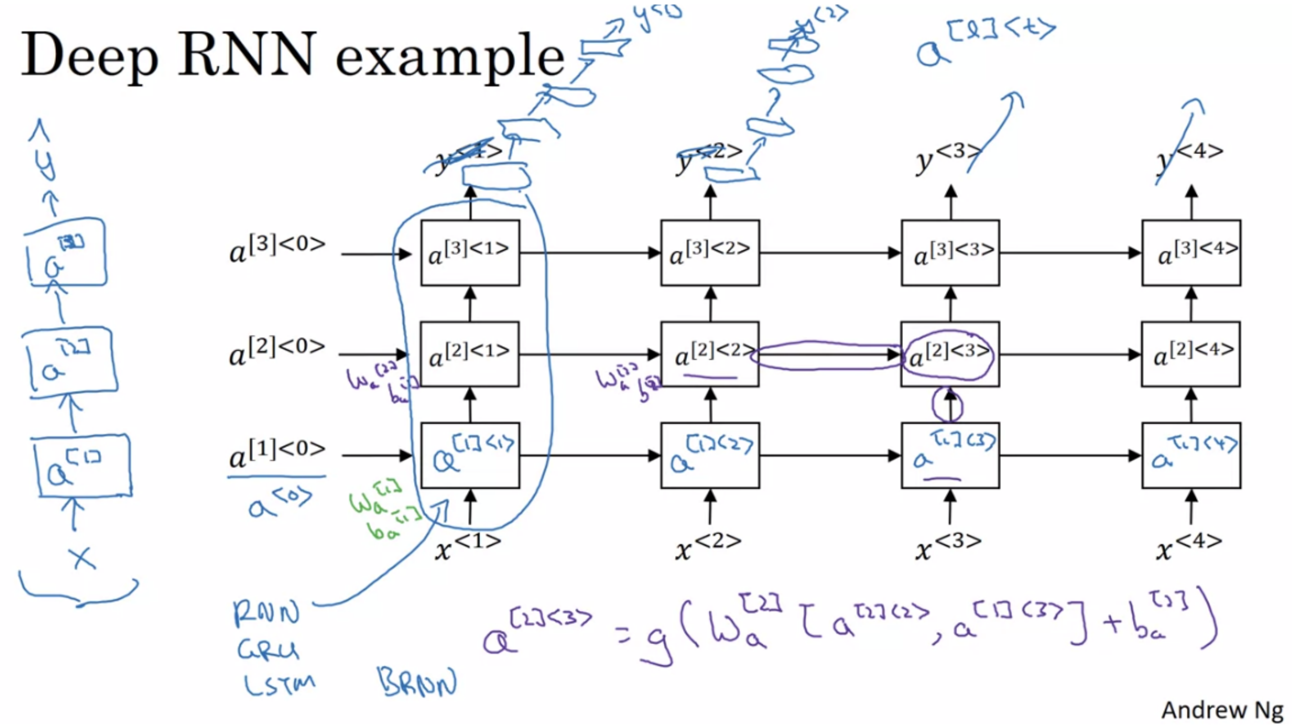
So that we have both `forward activation` and `backward activation`:

- we begin by going forward through the whole sentence with these traditional `forward activations`
- and then go back through all of the `backward activations`.
- In this way, we can then do the prediction at each position with both `forward info` and `backward info` if there existe.
- It works for RNN and also for GRU and LSTM

disadvantage: we need to konw the entire sentence before we do the text recognition.

7.2 Deep RNNs

We can use `multiple layers` of LSTM or RNN or GRU to contruct a deep RNN network,



BUT, Since a single RNN contains already many parameters, usually 3 layers is the maximum for the network structure.