Sequence Models

W1 - L1

X: $\overset{x^{<1>}}{\text{Harry}}$ $\overset{x^{<2>}}{\text{Potter}}$ and Hermione $\overset{x^{<t>}}{\text{Granger}}$ invented $\overset{x^{<7>}}{\text{a}}$ new $\overset{x^{<9>}}{\text{spell}}$

y: 1 1 0 1 1 0 0 0 0

1 → Name

0 → Not Name

$x^{(i)<t>}$: t^{th} word of i^{th} example

$T_x^{(i)}$: input sequence length of i^{th} training example

$y^{(i)<t>}$: t^{th} element of i^{th} output seq

$T_y^{(i)}$: Output sequence length of i^{th} training example

$x^{<t>}$ = feature for ~~the~~ t^{th} word

$T_x = 9$ (length of input)

$T_y = 9$ (" " output sequence)

Representing Words

Vocabulary

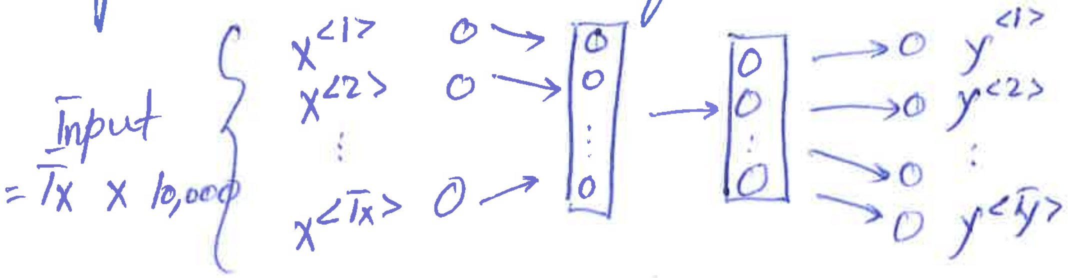
$$\begin{bmatrix} \text{a} \\ \text{aaron} \\ \vdots \\ \text{and} \\ \vdots \\ \text{harry} \\ \vdots \\ \text{potter} \\ \vdots \\ \text{zulu} \end{bmatrix} \begin{array}{l} 1 \\ 2 \\ \vdots \\ 367 \\ \vdots \\ 4075 \\ \vdots \\ 6830 \\ \vdots \\ 10{,}000 \end{array}$$

$$\underset{\text{HARRY}}{x^{<1>}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ \\ \\ \to 4075 \\ \\ \\ \\ \end{array} \quad (\mathbb{R}^{10{,}000})$$

$$x^{<7>} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (\mathbb{R}^{10{,}000})$$

Recurrent Neural Networks

Why not standard NN for NER?

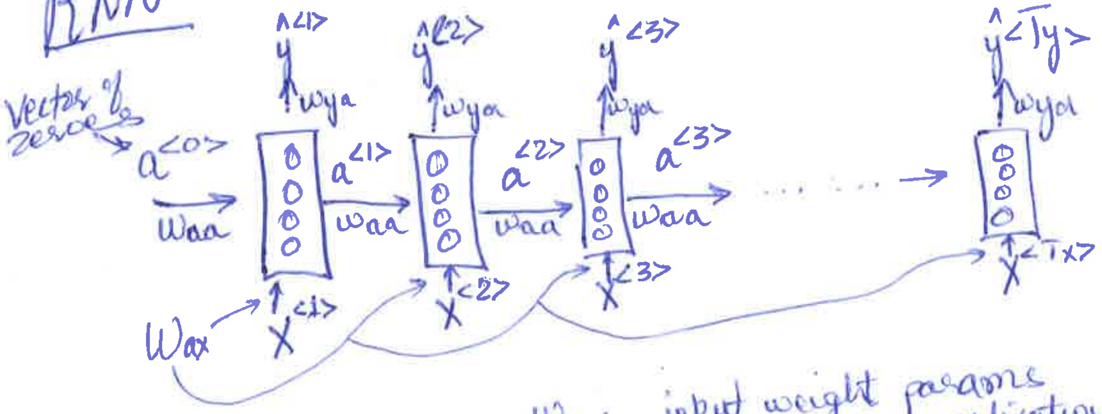


Problems

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

RNN

Here $T_x = T_y$

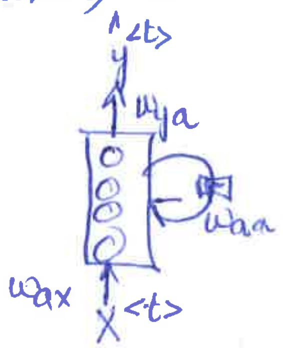


W_{ax}: input weight params
W_{aa}: params for activations
W_{ya}: Output params

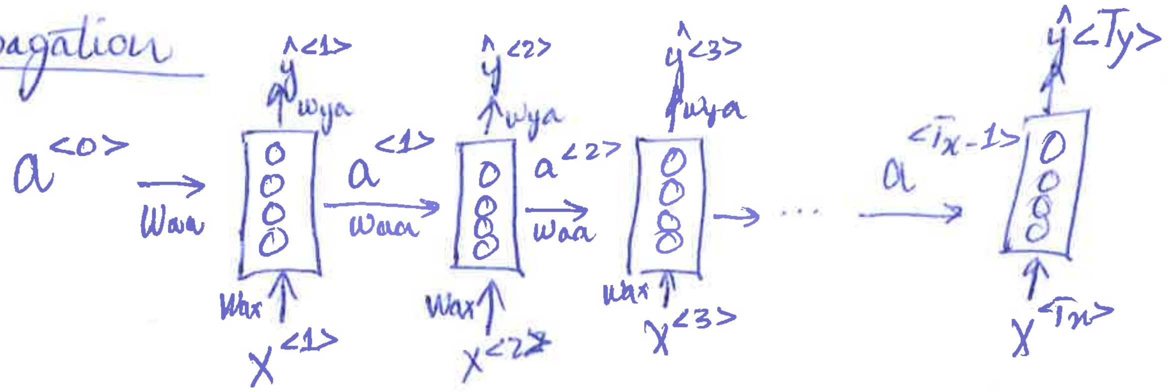
- ✱ In RNN, parameters are shared throughout the network.
- ✱ One disadvantage of RNN is that it does not take into account future inputs to make predictions on output at "t". Therefore, we will see Bidirectional RNN (BRNN) that does this job.

For instance: He said, "Teddy Roosevelt was a great President"
we can predict it as name if we take into account future words.

In some books, RNNs are shown like this:



Forward Propagation



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

$$a^{<1>} = g_1(W_{aa}\, a^{<0>} + W_{ax}\, x^{<1>} + b_a)$$

g_1: tanh / Relu Activation
(tanh: common in RNN)

$$\hat{y}^{<1>} = g_2(W_{ya}\, a^{<1>} + b_y)$$

g_2: Sigmoid activation (binary task)
or
g_2: softmax (for K class classification)

W_{ax}
2nd notation means it will be multiplied by quantity like "x"
1st notation means it will calculate quantity like "a".

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

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

Simplified RNN Notation

$$a^{<t>} = g(\underbrace{W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>}}_{\text{simplify this bit using stacking}} + b_a)$$

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

Simplifying

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

In our running example:

$x^{<t>} = R^{10,000}$ & lets say $a^{<t-1>} = R^{100}$

$\Rightarrow$ ~~Waax~~ $a^{<t>} = 100 \times 1)$

$W_{aa} = 100 \times 100$

$W_{ax} = 100 \times 10,000$

~~Then~~ And $W_a = \begin{array}{c} \\ 100 \end{array}\left[\begin{array}{c|c} W_{aa} & W_{ax} \\ \xleftrightarrow{100} & \xleftrightarrow{10,000} \end{array}\right] = (100 \times 10,100)$

(Stacking W_{aa}, W_{ax})

And $[a^{<t-1>}, x^{<t>}] = \underbrace{\begin{bmatrix} a^{<t-1>} \\ \\ x^{<t>} \end{bmatrix}}_{\text{Stacking } a^{<t-1>} \text{ \& } x^{<t>}} \begin{array}{l} \to R^{100} \\ \\ \to R^{1000} \end{array}$

$$\Rightarrow \boxed{W_a [a^{<t-1>}, x^{(t)}] = [W_{aa} \,\vdots\, W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}\, a^{<t-1>} + W_{ax}\, x^{<t>}}$$

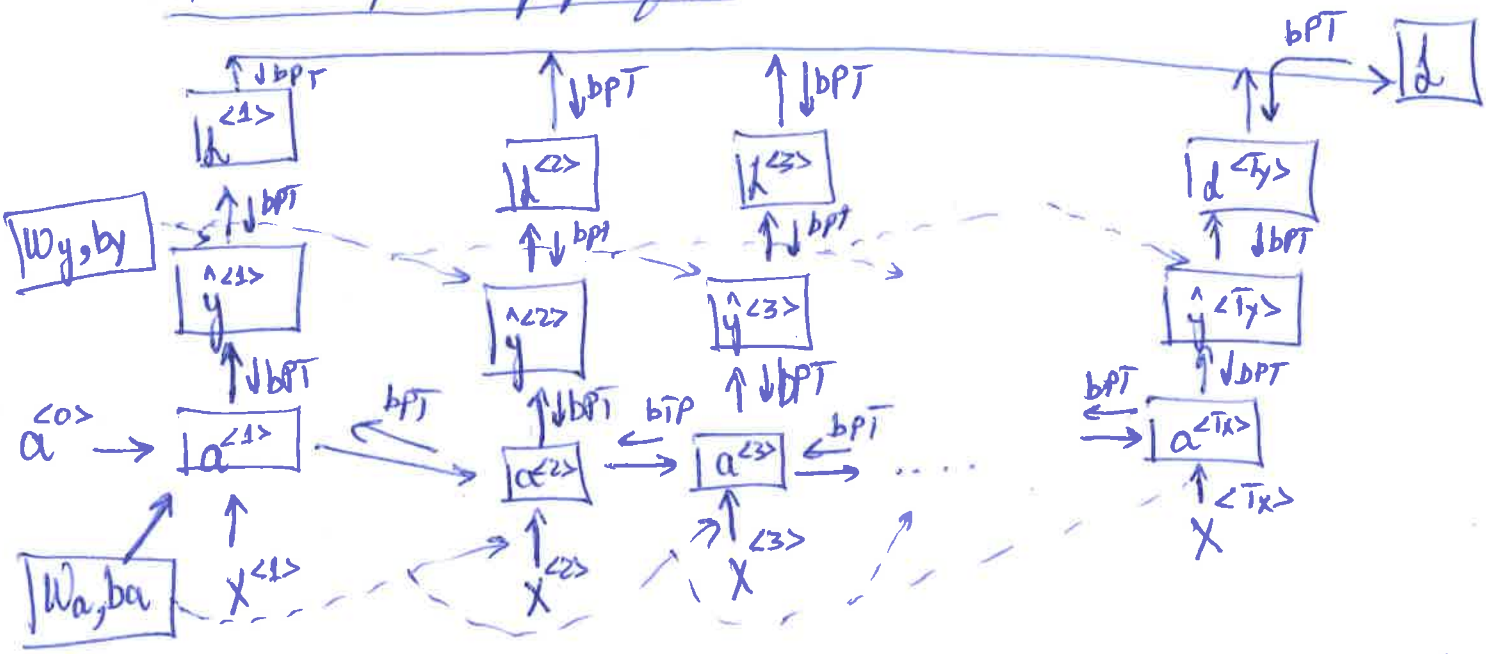
The advantage of this stacking is that instead of two parameters (W_{aa} & W_{ax}), we have just one parameter W_a — which will help us form complex networks.

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

subscripts now indicates we are calculating "y"

Backpropagation through time (BPT)

Lets see forwardprop of RNN:



Loss function

In our example of NER, we will define loss for each prediction of word.

$\Rightarrow$ loss for t^{th} word or t^{th} token (lets define logistic loss)

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

Prediction ($\hat{y}^{<t>}$), Ground truth ($y^{<t>}$)

For entire sequence:

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

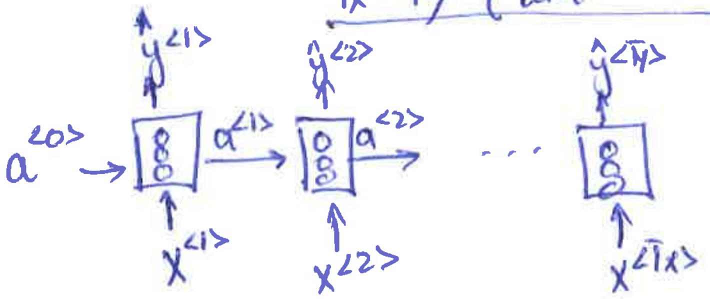
* BPT will update W_a, b_a, W_y, b_y by imputing partial derivatives.

Different types of RNNs

Presentation inspired by blogpost
Andrej Karpathy: "The unreasonable effectiveness of RNN".

1) Many-to-Many Architecture

$T_x = T_y$ (like our NER)



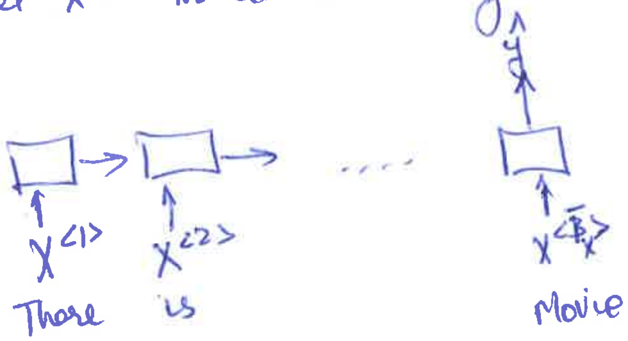
2) Many-to-One Architecture

Consider Sentiment analysis

x = text

y = 0/1 or five star rating

Let x = There is nothing to like in this movie



3) One-to-One Architecture

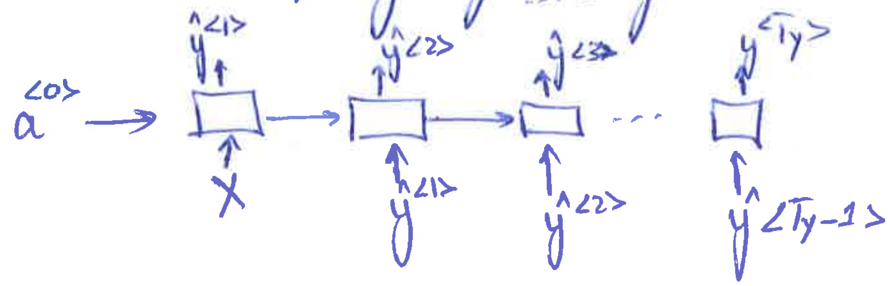
(Normal NN)

like in Course 1 & Course-2

4) ONE-to-Many Architecture

Example is Music generation

$X \rightarrow y^{<1>} \; y^{<2>} \; \cdots \; y^{<T_y>}$

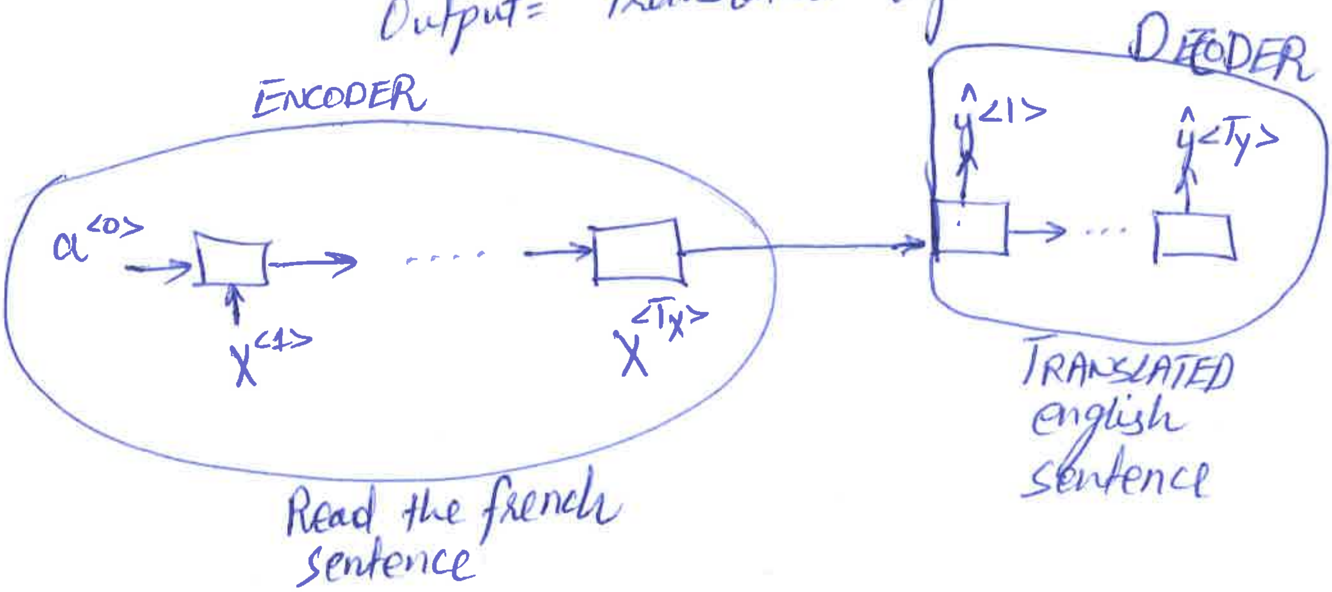


5) MANY-to-MANY Architecture

* $T_x \neq T_y$ like in Machine translation

Input = french sentence

Output = translated english sentence



Language Model

It takes input a text sequence,
Lets say an output of speech recognition $\hat{y} = \hat{y}^{<1>}, \hat{y}^{<2>}, \dots, \hat{y}^{<T_y>}$

$\boxed{\text{voice} \longrightarrow \text{text}(\hat{y})}$ (Apples and pears are delicious)

then language model will output the probability of $\hat{y}$ i.e $P(\hat{y})$

(Learning a Language Model using RNN)

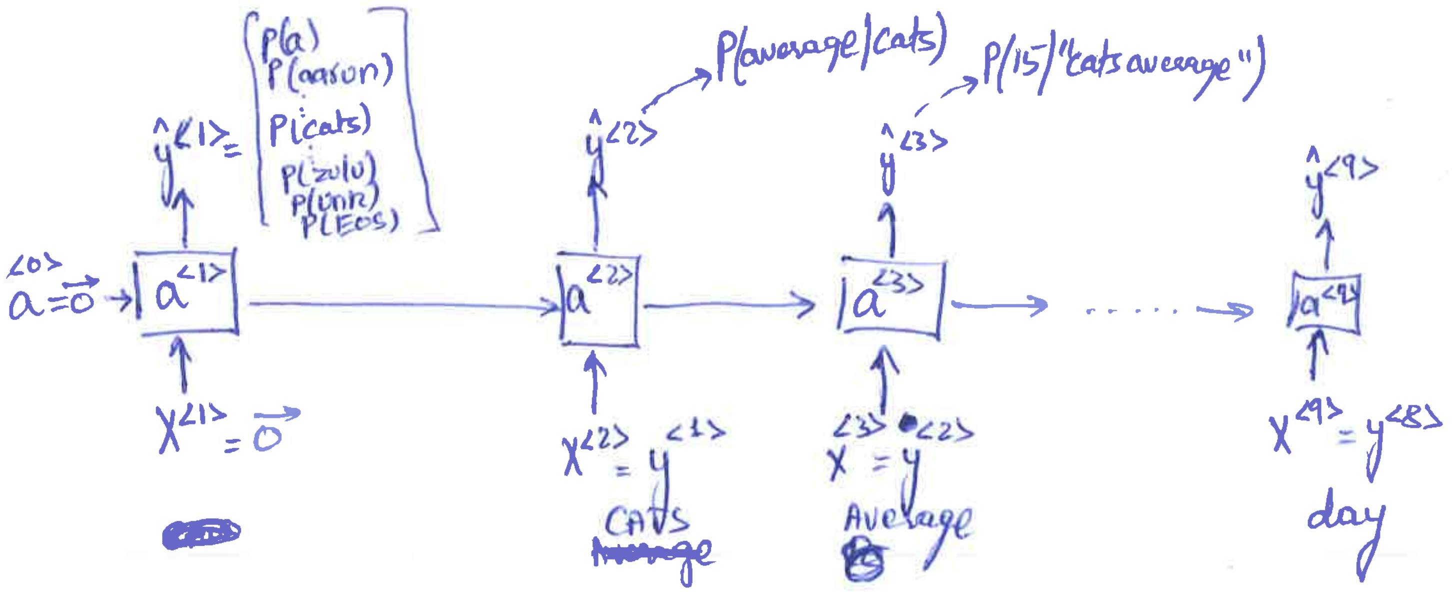
- Training Set: Large corpus of english text
- Running Example: CATs average 15 hours of sleep a day. <EOS>

$y^{<1>}$ (CATs), $y^{<2>}$ (average), $y^{<3>}$ (15) . . . . $y^{<8>}$ (day), $y^{<9>}$ (<EOS>)

- Lets imagine our dictionary is: $\begin{bmatrix} \text{a} \\ \text{aaron} \\ \vdots \\ \text{cats} \\ \vdots \\ \text{zulu} \end{bmatrix}$ 10,000 words

RNN Model

Initialize with zero vector: $a^{<0>}$ & $x^{<1>}$



Loss function (softmax):

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

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

After training, if we are given a sentence, then we can calculate its probability ~~after~~ using a language model.

Sentence : $y^{<1>}, y^{<2>}, y^{<3>}$

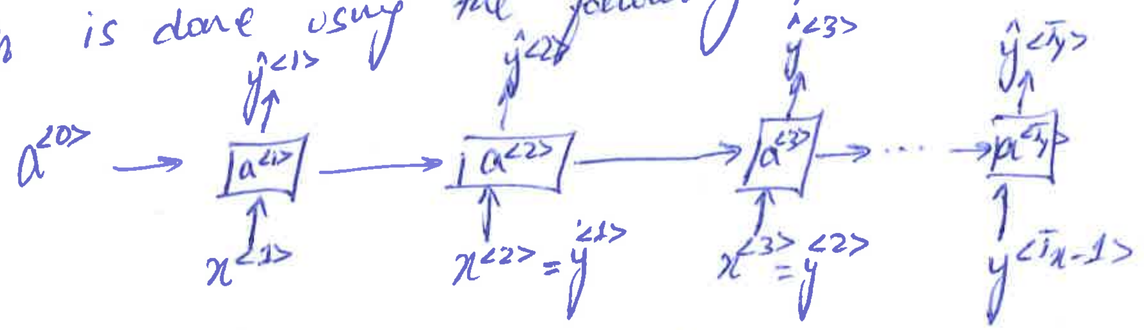
$$\Rightarrow P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>})\, P(y^{<2>} | y^{<1>})\, P(y^{<3>} | y^{<1>}, y^{<2>})$$

The advantage of language model is that we can sample sequences.

Sampling Novel Sequences

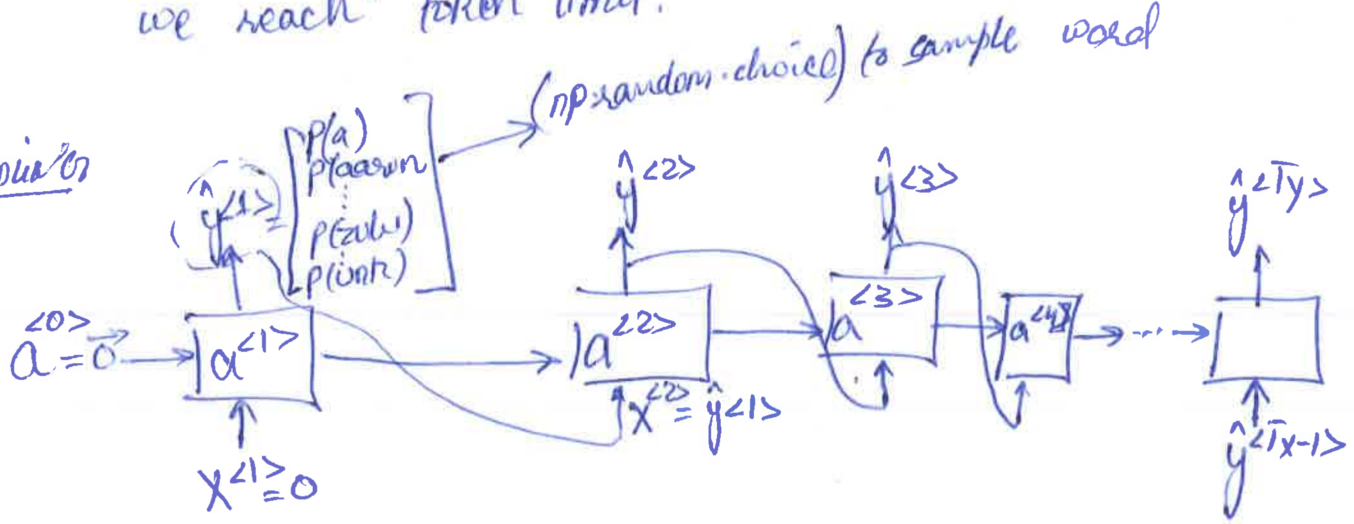
$$P(y^{<1>}, \dots y^{<T_y>}) = ??$$

Training is done using the following RNN:



For Sampling, we will do something different i.e our goal is to generate a new sequence of words. We will keep generating words until we get <EOS> token or we reach token limit.

Sampling

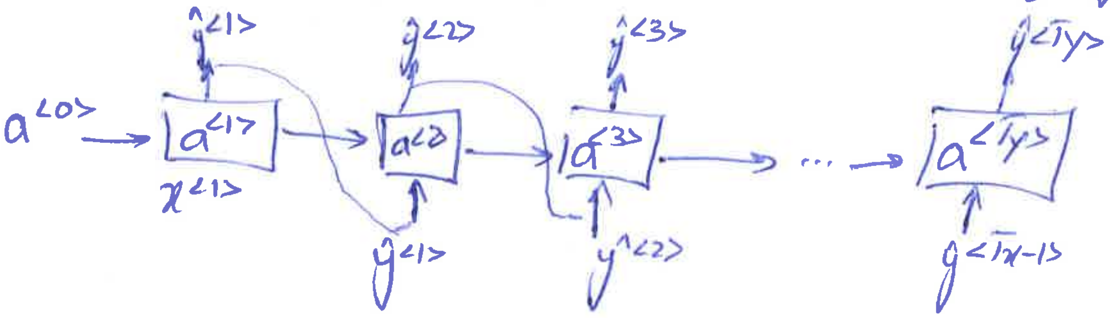


Uptil now, we have seen word-level models where vocabulary = [a, aaron, ..., zulu, <UNK>]

We can also build character-level model where:

$$\text{Vocabulary} = [a, b, c, \ldots, z, \underbrace{}_{\text{space}}, A, B \ldots Z, 0, 1 \ldots 9]$$

And now use RNN to learn character-level language model.



Here $\hat{y}^{\langle 1 \rangle} \ldots \hat{y}^{\langle T_y \rangle}$ are characters.

W1-28

VANISHING Gradient with RNNs

* RNNs are prone to vanishing gradient problem which will be solved using GRUs.
* There is one more issue, exploding gradient which is solved using gradient clipping (indication: you will see NaN values).

W1-29

GATED Recurrent Unit (GRU)

It is a modification to RNN's hidden layer that helps RNN capturing long-range dependencies

PAPER →
- Cho et al., 2014. On the properties of neural machine translation. Encoder-decoder approaches
- Chung et al. 2014. Empirical Evaluation of Gated Recurrent NN on Sequence Modeling

Intuition for GRUs

RNNs seems to have trouble in learning long-term dependencies.

Sent-1: The cat, which already ate ... was full.
Sent-2: The cats, which already ate ... were full

In both sentences, cat/cats depend on was/were. How to make RNNs learn these type of dependencies.

Notations (GRU variables)

c = memory cell (it will help memorize ~~sentence~~ if cat was singular or plural)

$c^{<t>}$ = memory cell value at location "t"

In GRU:

$c^{<t>} = a^{<t>}$: memory cell value is equal to output activation at time step "t". In LSTM, $c^{<t>}$ & $a^{<t>}$ will be different values.

$\tilde{c}^{<t>}$ = candidate to replace $c^{<t>}$

Γ_u = u stands for update "U"
Γ " " gate

Equations governing GRU unit:

At every timestep "t":

$\tilde{C}^{<t>}$ = candidate to replace $C^{<t>}$

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

Important idea:

$$\Gamma_U = \sigma(W_u[C^{<t-1>}, X^{<t>}] + b_u)$$

"U = update"

Γ_U: it is mostly very close to 0 or "1". Most of the times it is 0.

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

Illustration:

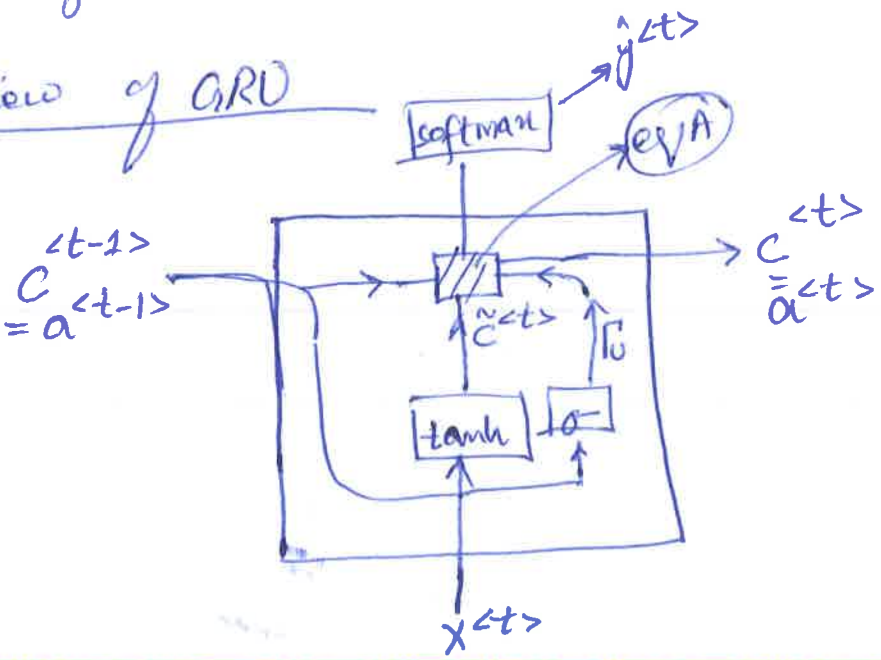
$\Gamma_U = 1$, $C^{<t>} = 1$; $\Gamma_U = 0$, $C^{<t>} = 1$; $\Gamma_U = 0$; ..... $C^{<t>} = 1$

Sent: The $\boxed{\text{cat}}$, which already ate, ... was full

Lets say we want to learn a concept (Singular ~~noun~~ subject). Thus we will set $\Gamma_U = 1$ when we have singular noun or singular subject in a sentence. Thus, the moment we detect "cat", we will set $\Gamma_U = 1$ & $C^{<t>} = 1$.

$\Gamma_U = 0$ means don't update $C^{<t>}$ & hang-on its old value. So until the end of sentence, you have successfully memorized that "cat" was singular

Pictorial View of GRU



How $c^{<t>}$ helps solving vanishing gradient problem?

Consider eq (A), most of the times Γ_u is close to zero (something like 0.000001) so as we scan from left to write, we maintain $c^{<t>} = 1$ as $c^{<t>} = c^{<t-1>}$ when $\Gamma_u \approx 0$. Thus ~~..~~ it makes possible to learn a concept even after long-term dependencies.

Dimensions

Lets assume $c^{<t>} = a^{<t>} = R^{100}$ (i.e 100 hidden units)

Then $\tilde{c}^{<t>}$ & Γ_u will also be 100-dimensional

Then in $c^{<t>}$ (eq A), $*$ will be element-wise multiplication

Full-GRU Unit

In academic literature

relevance

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

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

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

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

* GRU makes RNN able to capture long-term dependencies in input. ~~..~~ To capture long-term dependencies, GRU & LSTM are the two main ideas that make RNN achieve their goal.

LSTM

Seminal-Paper
Hochreiter & Schmidhuber 1997. LSTM

Blog Chris OALAH

LSTM lets RNN learn very long-term dependencies (just like GRU)

GRU (2-gates)

$\tilde{c}^{<t>} = \ldots$

$\Gamma_u = \ldots$

$\Gamma_r = \ldots$

$c^{<t>} = \ldots$

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

LSTM (3-gates)

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

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

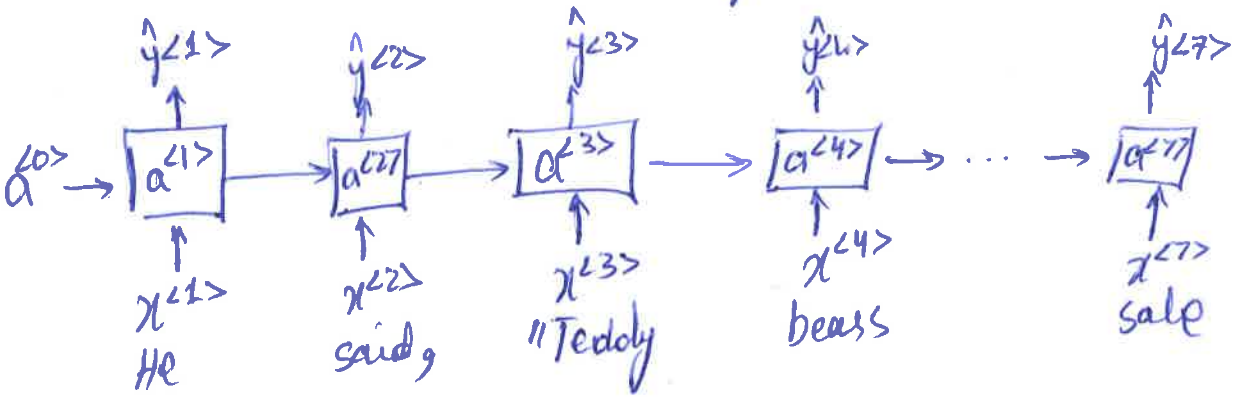
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$

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

Bi-directional RNN

~~In~~ ~~[illegible]~~ Getting information from the future

Sentence-1 : He said, "Teddy bears are on sale!"
Sentence-2 : " " , "Teddy Roosevelt was a great President!"

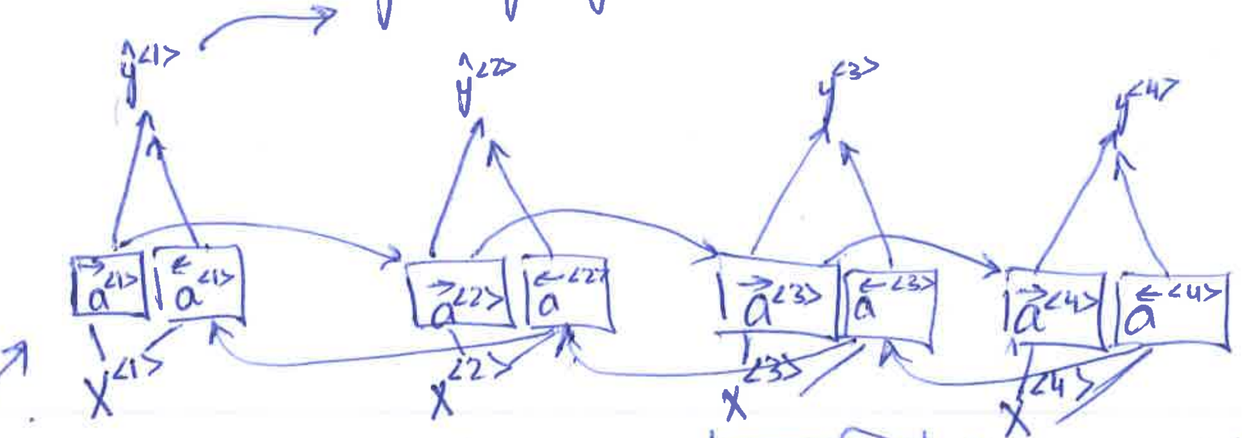


It is hard to know if $\hat{y}^{<3>} = 1$ (Name) or 0 (Not name) (i.e. Teddy) by just having a look on $x^{<1>}$ & $x^{<2>}$. This ~~sod~~ statement is true if above units are simple RNN, GRU or LSTM units. BRNN helps deciding whether $\hat{y}^{<3>}$ is name or not by taking into account entire sequence.

BRNN

Simplified 4-word sentence

$$\hat{y}^{<t>} = g(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$



Information from past is take { To predict $\hat{y}^{<3>}$ } ← information from future is taken

Acyclic graph

For many NLP task, bi-directional RNN with LSTM blocks is most common.

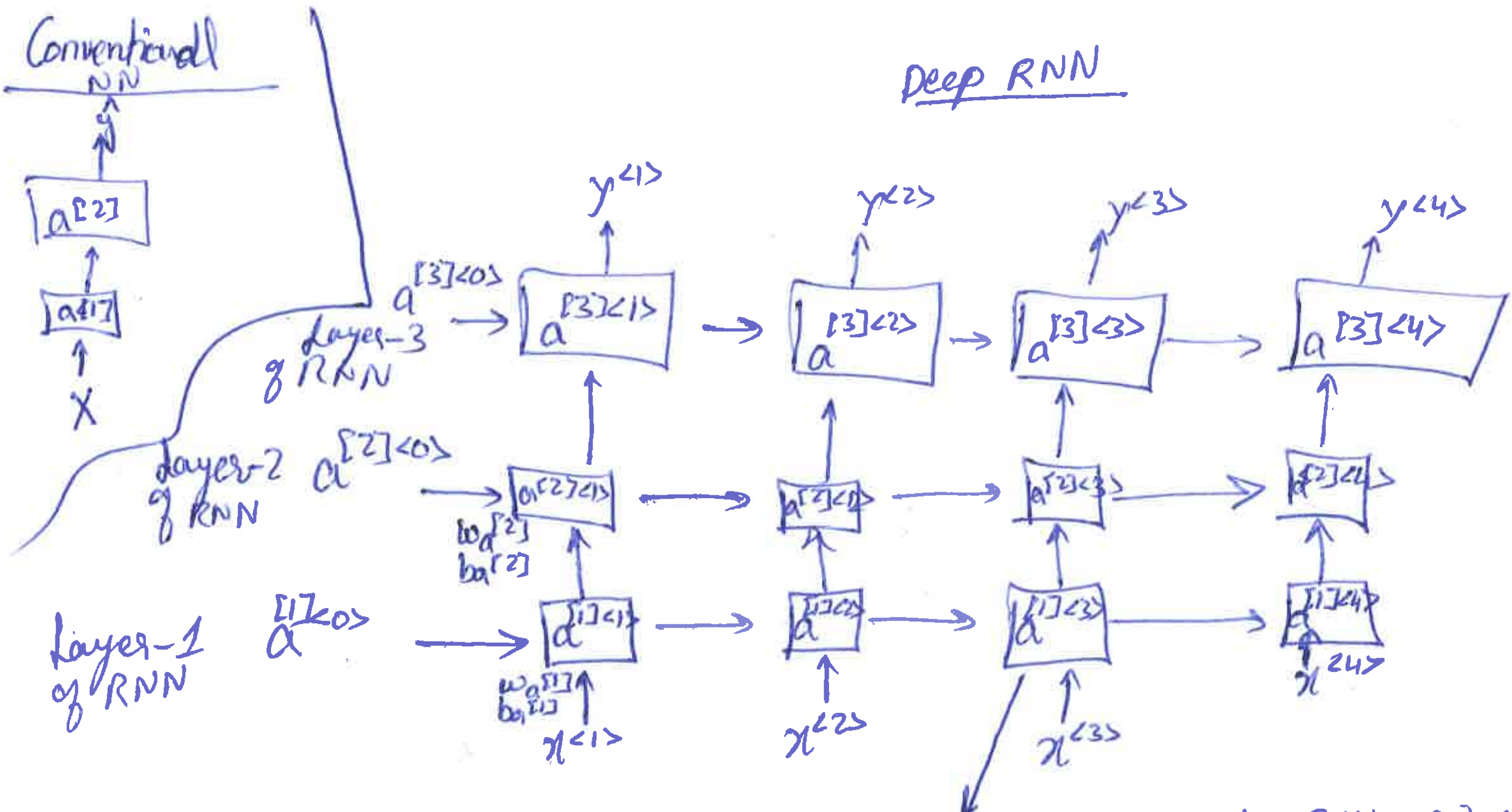
One disadvantage of BRNN is that we need entire sequence to make prediction.

<u>TAKEAWAY</u>

1) LSTM & GRU help RNN learn long-term dependencies
2) BiRNN helps making prediction by taking into account the entire sequence ~~i.e at time "t"~~ ($0 \to t$ & $t+1 \to t$ is taken into account for making prediction at "t")

W21-L12

Deep RNN Model



This unit can be RNN, GRU or LSTM unit

$$a^{[2]<3>} = g\left(w_a^{[2]}\left[a^{[2]<2>}, a^{[1]<3>}\right] + b_a^{[2]}\right)$$

<u>Modification</u> We can implement BRNN (for deep RNN) or append another N/w at the output with no-horizontal connection.