Sequence Models.

Examples of Sequence data

→ Speech recognition
→ Music generation
→ Sentiment classification
→ DNA sequence analysis.
→ Machine translation.
→ Video activity recognition
→ Name entity recognition

☆ Recurrent Neural Network.

Notation.

Motivating Example : Name entity recognition

x : Harry potter and Hermione Granger invented a new spell

y : 1 1 0 1 1
    0 0 0 0

Represent As :

$X : x^{<1>}\ x^{<2>}\ x^{<3>} \cdots x^{<t>} \cdots x^{<9>}$ $\quad T_x = 9$

$y : y^{<1>}\ y^{<2>}\ y^{<3>} \cdots y^{<t>} \cdots y^{<9>}$ $\quad T_y = 9$

Therefore,

$X^{(i)\langle t \rangle}$: t^{th} element in Sequence in i^{th} Example of training

$y^{(i)\langle t \rangle}$: t^{th} element in Sequence in i^{th} output of training training

$T_x^{(i)}$: length of Input Sequence in i^{th} example.

$T_y^{(i)}$: length of output Sequence in i^{th} Example.

~~Representing Words.~~

~~X: Harry potter and Hermione Granger~~

P.T.O →

Representing Words.

X: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<6>}$ $x^{<7>}$ $x^{<8>}$ $x^{<9>}$

Vocabulary

a	1	0		0		0		1	← 1
aaron	2	0		0		0		0	
⋮	⋮	⋮		⋮		⋮		⋮	
and	367	0		0		1	← 367	0	
⋮	⋮	⋮		⋮		⋮		⋮	
harry	4075	1	← 4075	0		0		0	
⋮	⋮	⋮		⋮		⋮		⋮	
Potter	6830	0		1	← 6830	0		0	
⋮	⋮	⋮		⋮		⋮		⋮	
zulu	10,000	0		0		0		0	

10,000

Size of dictionary = 10,000

we did "one-hot" encoding.

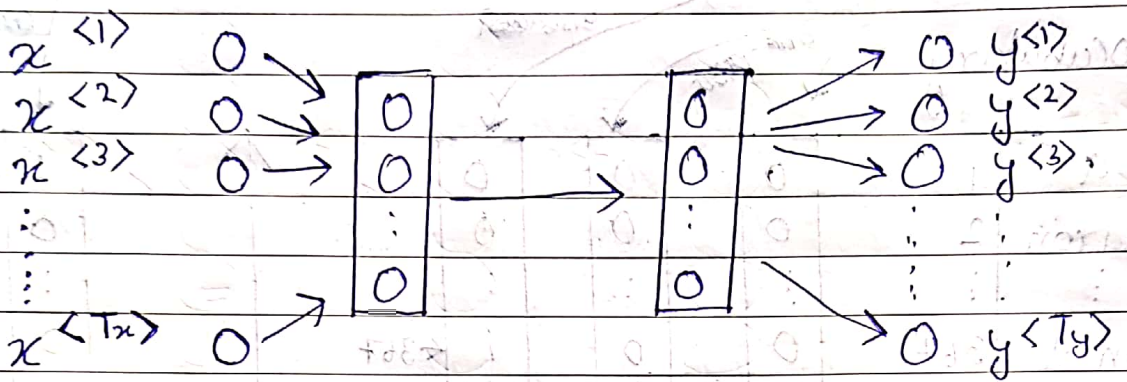
⇒ So, In this representation, $x^{<t>}$ will be one-hot vector.

⇒ will do this as it is Supervised learning method (X → Y)

RNN model.

Q Why not a Standard network?

⇒



problems:

⇒ Input, outputs can be different length in different examples.

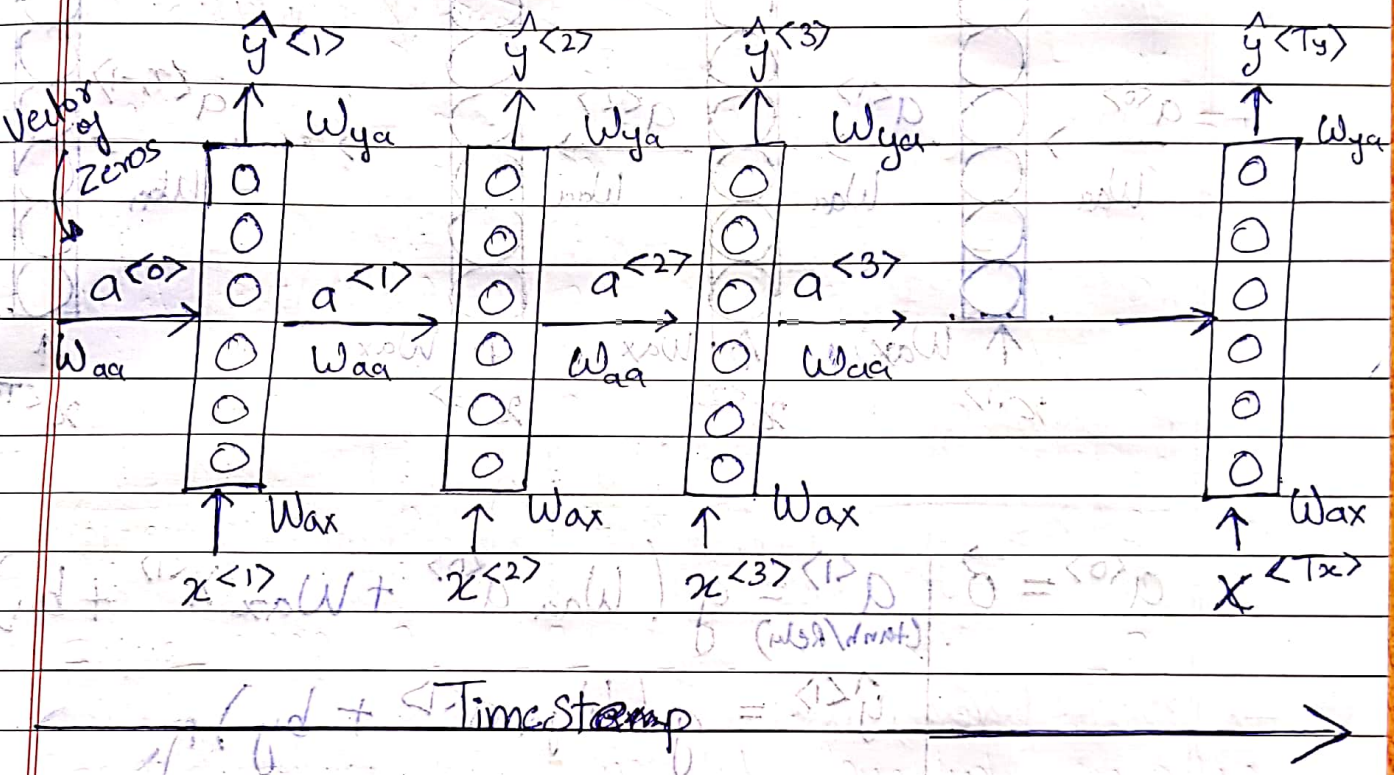
⇒ Doesn't share features learned across different positions of text.

⇒ Weight metrix will end-up have enormous number of parameters.

∴ Standard Network won't work.

Q what is Recurrent Neural Networks.

⇒ Let build one up:

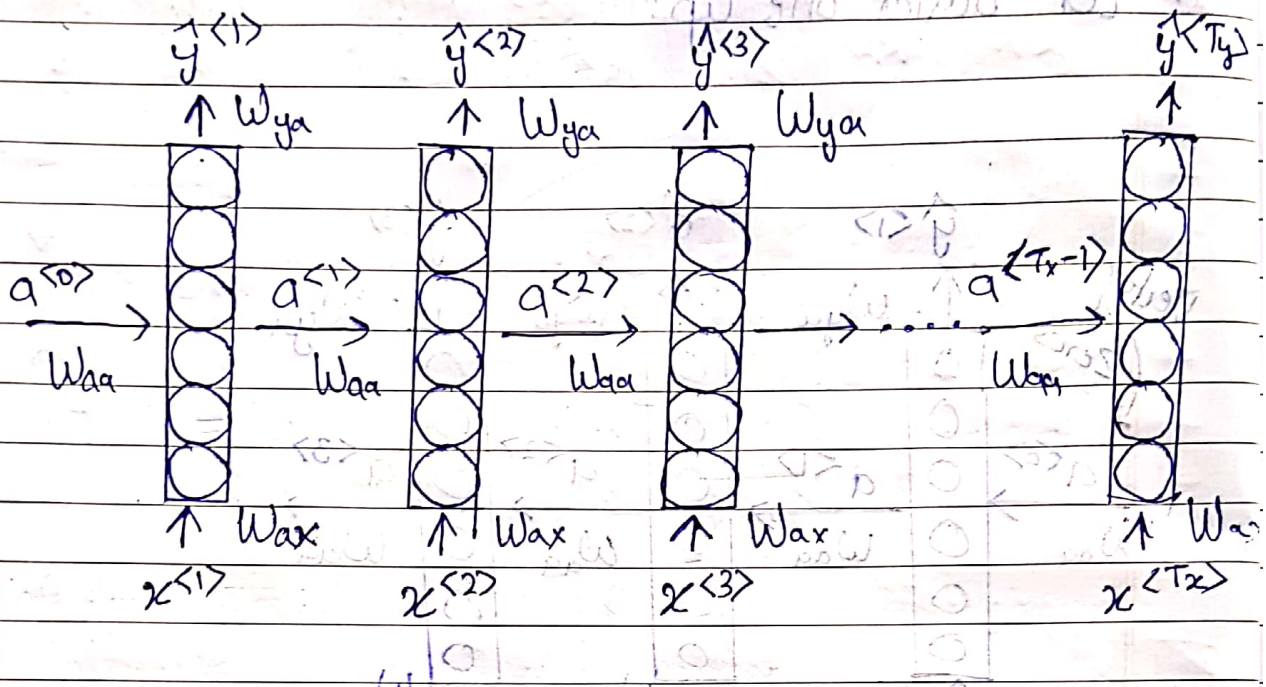


⇒ So, in this RNN what does it means is that when making the prediction of $\hat{y}^{<3>}$, it get information not only from $x^{<3>}$ but also information from $x^{<1>}$ and $x^{<2>}$ because information on $x^{<1>}$ can pass thru from $x^{<2>}$ to $x^{<3>}$ to help with prediction.

Limitation:

→ only uses information that earlier in the sequence, in particular, when predicting $\hat{y}^{<3>}$ It does not uses information from $x^{<4>}$, $x^{<5>}$, $x^{<6>}$ etc.

⇒ forward propagation



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

$a^{<1>} = g(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a)$ (tanh/Relu)

$\hat{y}^{<1>} = g(W_{ya} a^{<1>} + b_y)$ ← Sigmoid/Softmax

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

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

In $a^{<t>}$ activation could be, i.e $g()$.

⇒ tanh(), relu()

⇒ Simplified RNN Notation

$$a^{\langle t \rangle} = g(W_{aa} \cdot a^{\langle t-1 \rangle} + W_{ax} \cdot x^{\langle t \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = g(W_{ya} \cdot a^{\langle t \rangle} + b_y)$$

→ $$a^{\langle t \rangle} = g(W_{aa} \cdot a^{\langle t-1 \rangle} + W_{ax} \cdot x^{\langle t \rangle} + b_a)$$

$$a^{\langle t \rangle} = g(W_a [a^{\langle t-1 \rangle} + x^{\langle t \rangle}] + b_a)$$

i.e

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

take W_{aa} and W_{ax} and stack them side by side as shown above that we create W_a.

For Example: if a is 100 dim., and x is 10,000 dim.

then $W_{aa} \in \mathbb{R}^{100 \times 100}$ and $W_{ax} \in \mathbb{R}^{100 \times 10000}$

∴ Stacking this two matrix will be,

$$100 \updownarrow [W_{aa} \mid W_{ax}] = W_a \in \mathbb{R}^{100 \times 10,100}$$

(100 + 10,000)

And Same with Vectors.

§ Stacking $a^{\langle t-1 \rangle}$ and $X^{\langle t \rangle}$

$$[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] = \begin{bmatrix} a^{\langle t-1 \rangle} \\ \hline x^{\langle t \rangle} \end{bmatrix} \begin{matrix} \updownarrow 100 \\ \updownarrow 10{,}000 \end{matrix} \quad \updownarrow 10{,}100.$$

therefore,

$$\rightarrow \quad a^{\langle t \rangle} = g(W_a [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a)$$

$$a^{\langle t \rangle} = g\left([W_{aa} \mid W_{ax}] \begin{bmatrix} a^{\langle t-1 \rangle} \\ \hline x^{\langle t \rangle} \end{bmatrix} + b_a \right)$$

which equal to :

$$a^{\langle t \rangle} = g(W_{aa} \cdot a^{\langle t-1 \rangle} + W_{ax} \cdot x^{\langle t \rangle} + b_a)$$

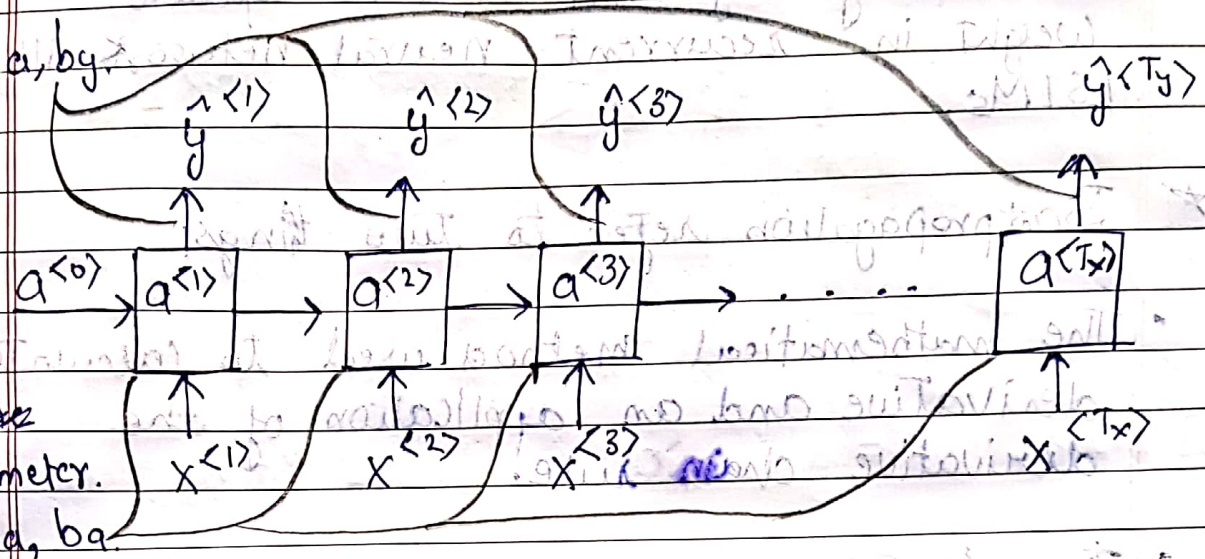
$\Rightarrow$ Now ~~for~~, No change need for this

$$\hat{y}^{\langle t \rangle} = g(W_{ya} \cdot a^{\langle t \rangle} + b_y)$$

- $\hat{y}^{\langle t \rangle}$: output at ~~time~~ step 't'
- W_{ya} : output weight metric
- b_y : output bias.

→ Backpropagation through time

Wya, by.



Parameter.
Wa, ba

★ Loss function (Cross-entropy)

Element wise:

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

Overall loss:

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

⇒ Backpropagation Through Time, or BPTT, is the Training algorithm used to update weight in recurrent neural network like LSTMs.

★ Backpropagation refer to two things:

- The mathematical method used to calculate derivative and an application of the derivative chain rule.
- The training algo. for updating network weights to minimize error.

→ It is a Supervised learning algo. that allows the network to be corrected with regard to the specific error made.

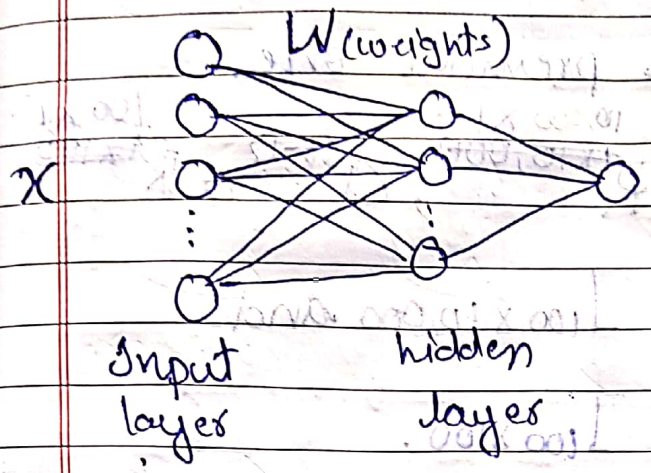
★ The general algorithm is as follows:

1) Present a training input pattern and propagate it through the network to get output.
2) Compare the prediction output to the expected and calculate the error.
3) Calculate the derivatives of the error with respect to the network weights.
4) Adjust the weights to minimize the error.
5) Repeat.

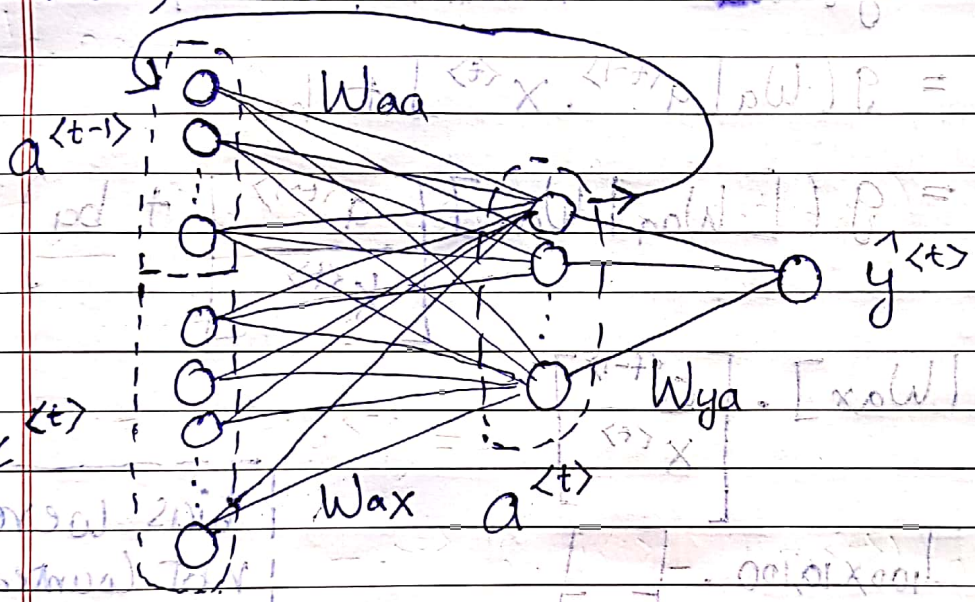
Self-Explanation

* What RNN look like according to me?

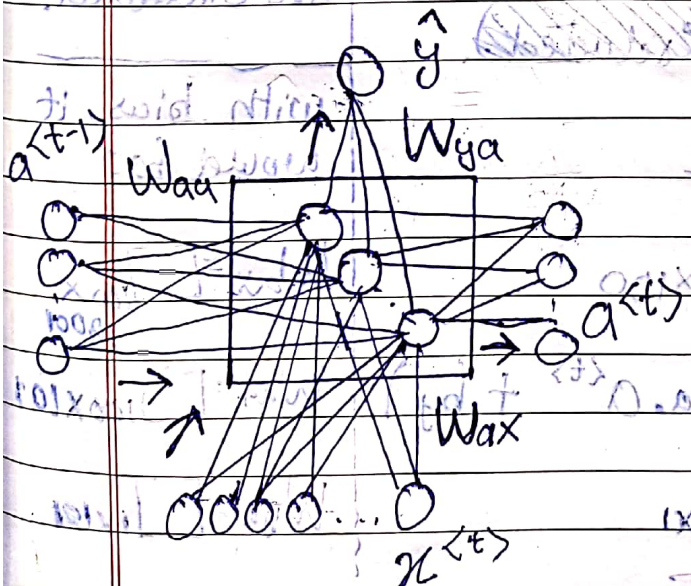
⇒ First, Standard NN.



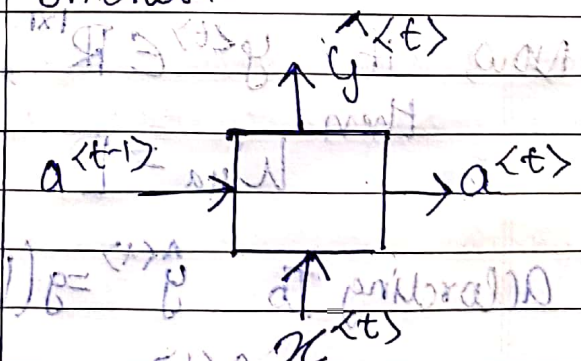
→ Now, RNN



→ This is how I think it looks like.



In short:



Say-Explanation

Q. How input, weight and bias pass thru to calculate $a^{<t>}$ and outputs?

⇒ for Example, Same as previous one.

Consider, if $x^{<t>} \in \mathbb{R}^{10,000 \times 1}$, $a^{<t-1>} \in \mathbb{R}^{100 \times 1}$

then,

$W_{ax} = [\ ]_{100 \times 10,000}$ and

$W_{aa} = [\ ]_{100 \times 100}$.

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

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

$$a^{<t>} = g\left([W_{aa} | W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_a\right)$$

$\therefore$ $[W_{aa} | W_{ax}] \cdot \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$

$= [\ ][\ ]_{100 \times 10,100} \cdot \begin{bmatrix} \\ \\ \end{bmatrix}_{10,100 \times 1}$

$a^{<t>} = [\ ]_{100 \times 1}$

bias were not counted ∴ b_a and b_y are excluded.

with bias, it would be:

$W_{ax} = [\ ]_{100 \times 10,001}$

$W_{aa} = [\ ]_{100 \times 101}$

$W_{ya} = [\ ]_{1 \times 101}$

Now, if $y^{<t>} \in \mathbb{R}^{1 \times 1}$,

then $W_{ya} = [\ ]_{1 \times 100}$

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

$\hat{y}^{<t>} = [\ ]_{1 \times 1}$

☆ Different types of RNNs

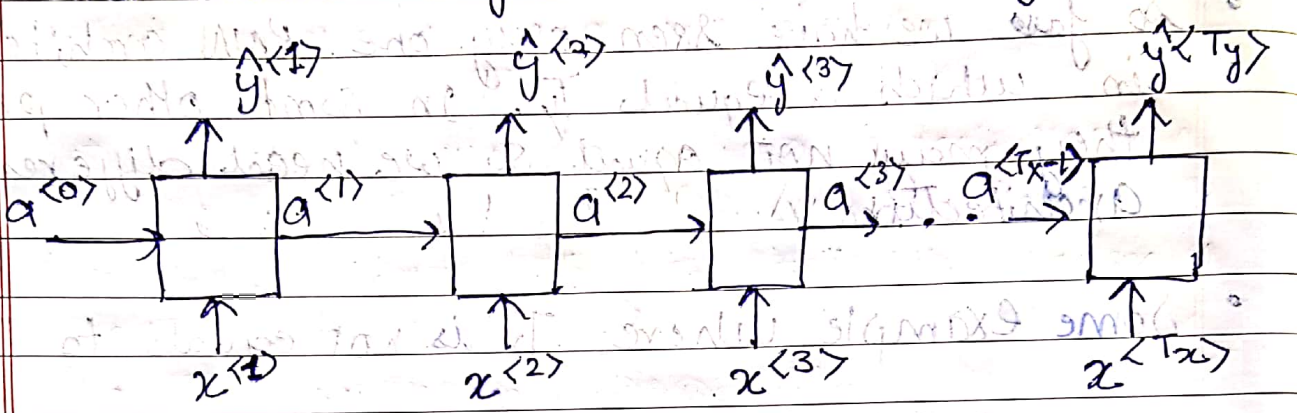
- So far we have seen only one RNN architecture in which T_x equals T_y. In some other problems, they may not equal so we need different architecture.

- Some example where T_x is not equals to T_y.

	X	Y
→ Sentiment classification	"There is nothing to like in this movie."	★☆☆☆☆ 1 to 5 rating
→ Machine translation (many-to-many but different length)	Voulez-vous chantar avec moi?	Do you want to sing with me?
→ Music generation	∅ Null	many words.
→ Name entity recognition	Yesterday, Harry potter met Ron	Yesterday, Harry potter met Ron
	many	many

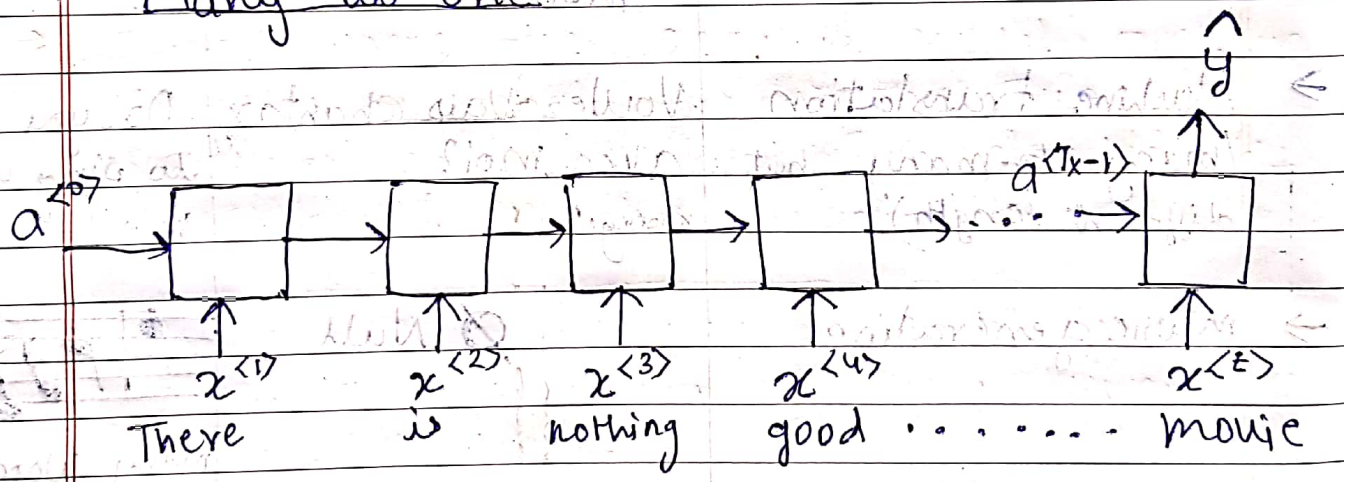
- **Examples of RNN architectures.**

→ when $T_x = T_y$, we have seen only this architecture so far. Many-to-Many



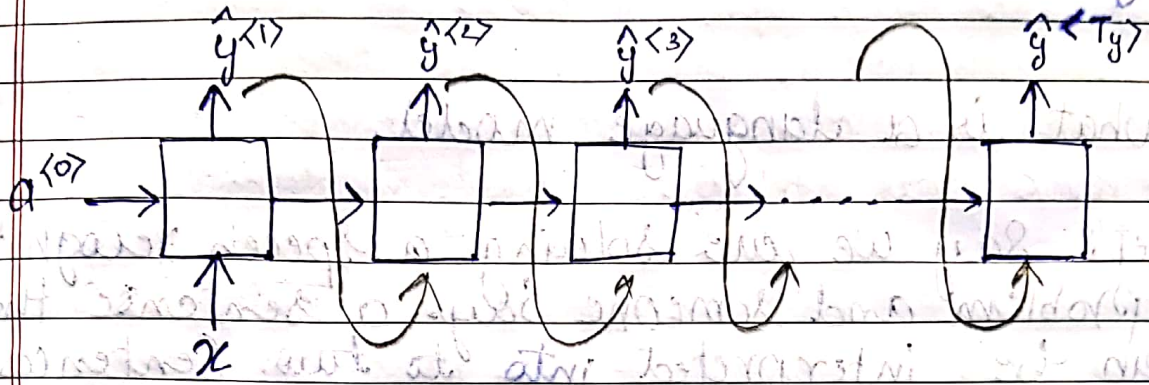
→ In Sentiment analysis problem, X is a text while Y is a Integer that ranges 1 to 5. The RNN architecture for that is:

→ Many-to-One



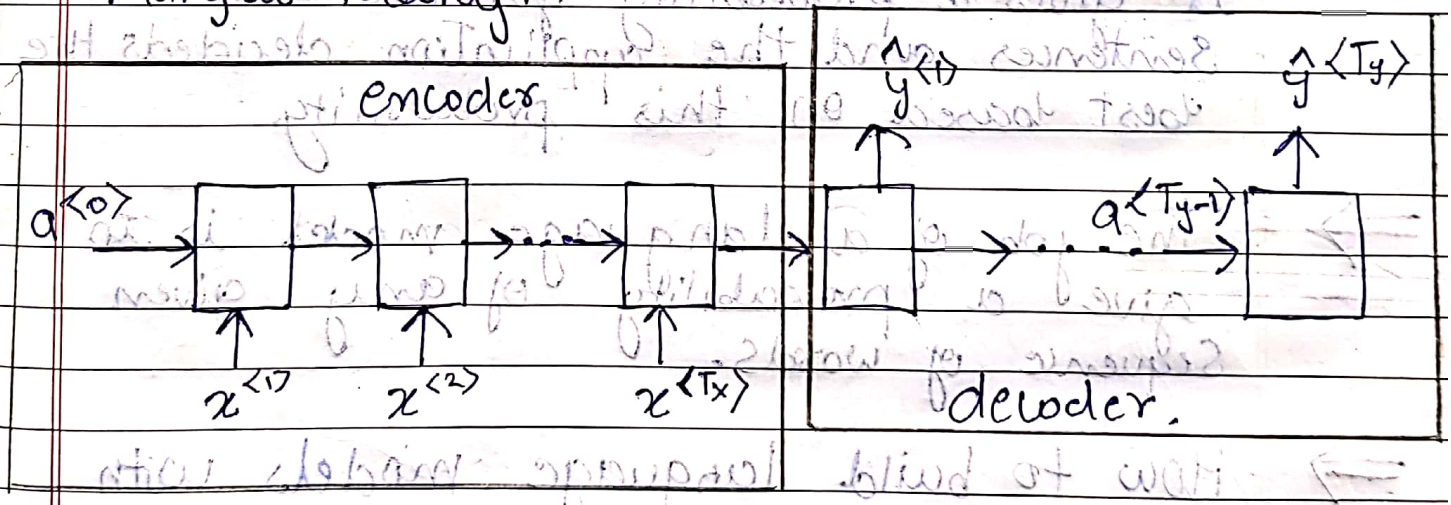
→ Classic, standart Neutral Network known as one-to-one RNN

→ A one-to-many architecture application would be music generation.



- Note that Starting the Second layer we are feeding the generated output back to the ~~next~~ Network.

→ There are another interesting architecture is Many-to-many. Application like machine translation inputs and outputs sequences have different lengths in most of the cases. ∴ alternative of Many-to-Many.



- There are an encoder and a decoder parts in this architure. The encodes the input ~~the~~ sequence into one metrix and feed it to the decoder to generate the outputs. Encoder and decoder have different weight matrices.

☆ Language model and sequence generation.

⇒ What is a language model.

→ Let's say we are solving a speech recognition problem and someone says a sentence that can be interpreted into to two sentences
- The apple and pair salad
- The apple and pear salad

→ Pair and pear sounds exactly the same, so how would a speech recognition application choose from the two.

→ That's where the language model comes in. it gives a probability for the two sentences and the application decideds the best based on this probability.

⇒ The job of a language model is to give a probability of any given sequence of words.

⇒ How to build language models with RNNs?

→ The first thing is to get a training set: a large corpus of target language text

→ Then tokenize this training set by

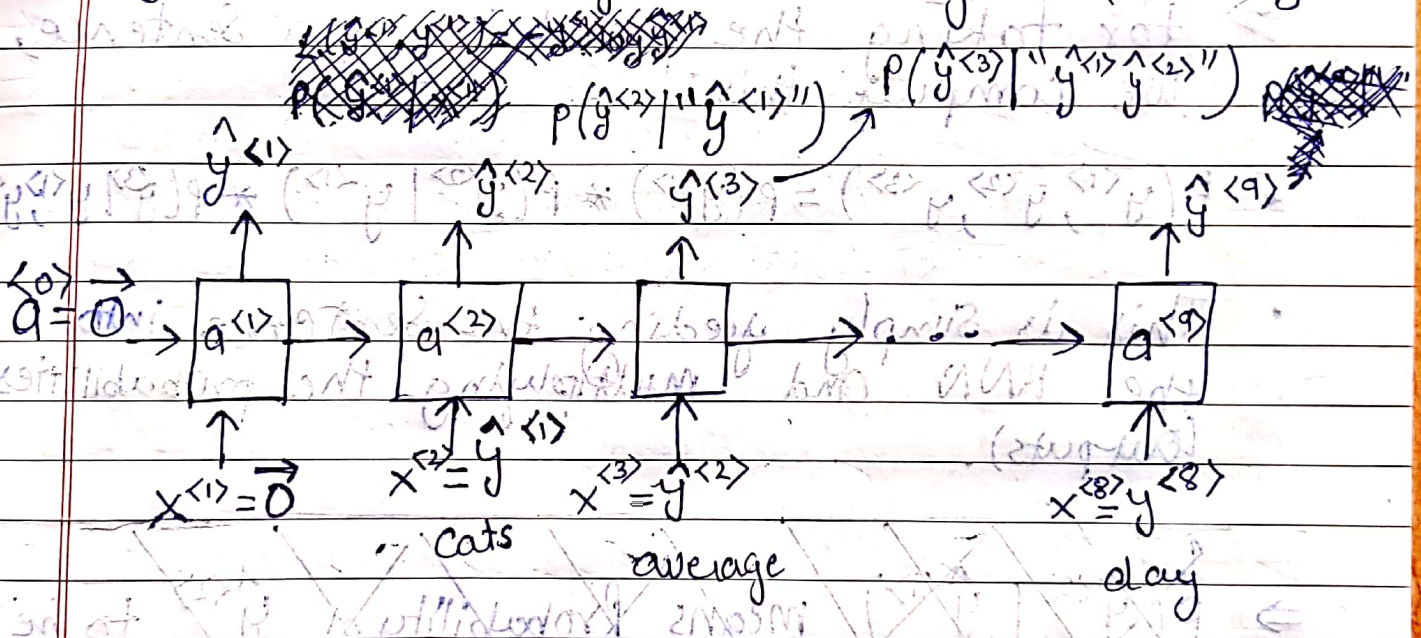
getting the vocabulary and then one-hot each word.

→ Put an 'end of sentence' token <EOS> with the vocabulary and include it with each converted sentence. Also, use the token <UNK> for unknown words.

⇒ Given the sentence.

$y^{<1>}\ y^{<2>}\ y^{<3>}\ y^{<4>}\ y^{<5>}\ y^{<6>}\ y^{<7>}\ y^{<8>}\ y^{<9>}$

y = "Cats average 15 hours of sleep a day. <EOS>"



→ Loss function is define by cross-entropy loss.

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

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

Note $\mathcal{L}(\hat{y}, y) = -\hat{y} \cdot \log(\hat{y})$
if $y = 0 = -0 \cdot \log(\hat{y})$
$= 0$

*enlivo
Date: ___/___/___
Page: ___

→ i is of all element in the corpus, t is for all timesteps.

⇒ To use this model:

→ for predicting the chance of next word, we feed the sentence to the RNN and then get the final $\hat{y}^{<t>}$ hot vector and sort it by maximum probability.

→ for taking the probability of a sentence, we compute this:

- $P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>}) * P(y^{<2>} | y^{<1>}) * P(y^{<3>} | y^{<1>}, y^{<2>})$
- This is simply feeding the sentence into the RNN and multiplying the probabilities (outputs).

⇒ $P(\hat{y}^{<2>} | \hat{y}^{<1>})$ means probability of $\hat{y}^{<2>}$ to be "average" due to $\hat{y}^{<1>}$ (cats) as input.

⇒ $P(\hat{y}^{<3>} | \hat{y}^{<1>} \hat{y}^{<2>})$ means probability of $\hat{y}^{<3>}$ to be "is" due to $\hat{y}^{<1>}$ and $\hat{y}^{<2>}$ "cats average"

⇒ like wise goes for all till $P(\hat{y}^{<9>} | \hat{y}^{<1>} \dots \hat{y}^{<8>})$

⇒ Probability of sentence will be:

$P(\hat{y}^{<1>}, \hat{y}^{<2>}, \dots \hat{y}^{<9>}) = P(\hat{y}^{<1>}) * P(\hat{y}^{<2>} | \hat{y}^{<1>}) * \dots P(\hat{y}^{<9>} | \hat{y}^{<1>} \dots \hat{y}^{<8>})$

*enlivo
Date: __/__/____
Page: _____

As mentioned :

y = "Cats average 15 hours of sleep a day. <EOS>"

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$ $y^{<7>}$ $y^{<8>}$ $y^{<9>}$

⇒ here, P( ) mean probability.

at first time step, $P(y^{<1>})$ mean what is the probability of $\hat{y}^{<1>}$ to be $y^{<1>}$ i.e "Cats", as $\hat{y}^{<1>}$ will be output Vector coming thru Softmax, which means what will be the probability of "Cats" in output vector when Input is is feed to Network. $x^{<1>} = \vec{0}$, $a^{<0>} = \vec{0}$

Same way, at second time step, $P(y^{<2>} | y^{<1>})$ means what is probability of $\hat{y}^{<2>}$ to be $y^{<2>}$ i.e "Average" when $y^{<1>}$ was feed into network as $x^{<2>}$. $x^{<2>} = y^{<1>}$.

Same way, at third time step, $P(y^{<3>} | y^{<1>} y^{<2>})$ means what is probability of $\hat{y}^{<3>}$ to be $y^{<3>}$ i.e "15" when $y^{<2>}$ was feed into network as $x^{<3>}$. $x^{<3>} = y^{<2>}$.

So. on, till the end $P(y^{<9>} | y^{<1>} y^{<2>} y^{<3>} \dots y^{<8>})$

→ To Calculate probability of Sentence.

$P(y^{<1>}, y^{<2>}, y^{<3>}, y^{<4>}, y^{<5>}, y^{<6>}, y^{<7>} y^{<8>}, y^{<9>}) =$

$P(y^{<1>}) * P(y^{<2>} | y^{<1>}) * P(y^{<3>} | y^{<1>} y^{<2>} y^{<3>}) * \dots$

$* P(y^{9} | y^{<1>} y^{<2>} y^{<3>} \dots y^{<8>})$

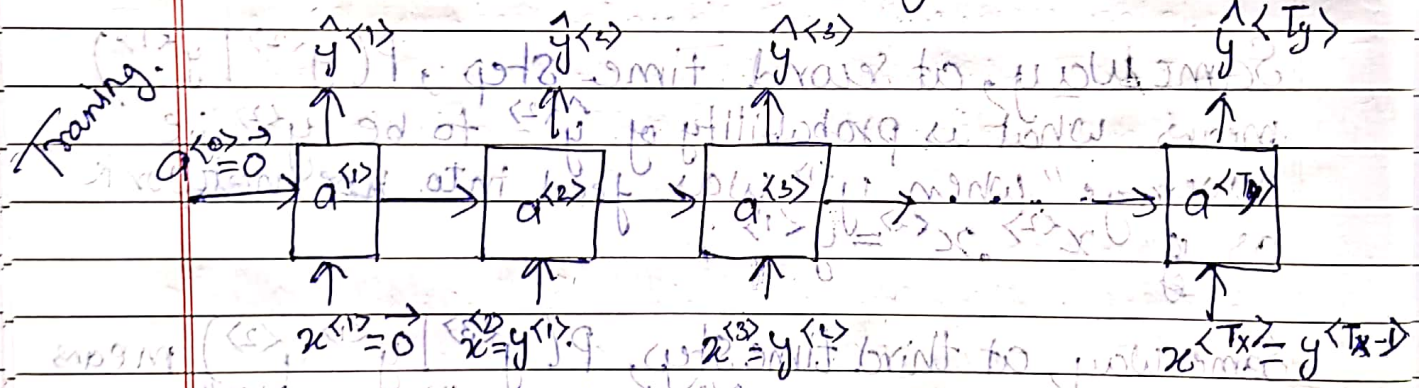
☆ Sampling noval Sequences.

- 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

- lets see the steps of how we can Sample a noval Sequence from trained Sequence language model:

First of all,

⇒ Network was trained using this Structure

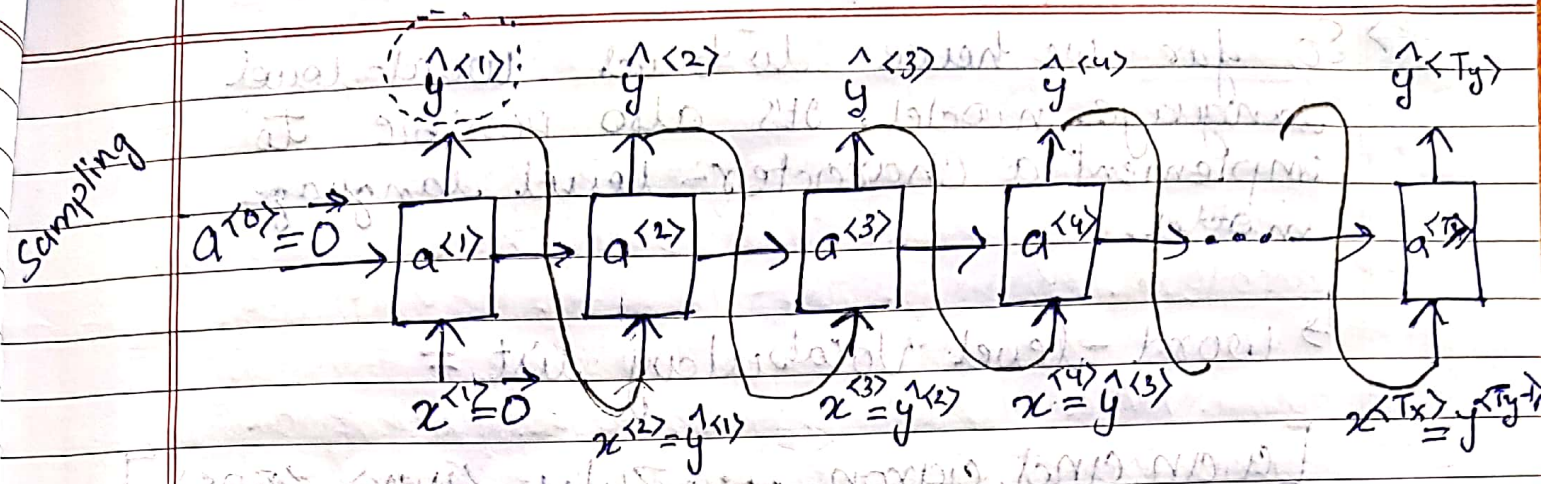


where probability was

$$P(y) = P(y^{\langle 1\rangle}), P(y^{\langle 2\rangle} | y^{\langle 1\rangle}), P(y^{\langle 3\rangle} | y^{\langle 1\rangle} y^{\langle 2\rangle}) \ldots$$

$$P(y^{\langle T_y\rangle} | y^{\langle 1\rangle} y^{\langle 2\rangle} \ldots y^{\langle T_x-1\rangle})$$

⇒ Now let See how Sampling is done on a trained language model.



→ We first pass $a^{\langle 0\rangle} = \vec{0}$, and $x^{\langle 1\rangle} = \vec{0}$

→ Then we choose a prediction randomly from distribution obtained by $\hat{y}^{\langle 1\rangle}$. for exampt it could be anything, les say "The".

↳ In numpy this can be implemented using: numpy.random.choice(...)

↳ This is the done where you get a random beginning of the Sentence each time you sample run a novel sequence.

→ We pass the last predicted word with the Calculated $a^{\langle 1\rangle}$ to next time step as input.

→ We keep doing 3 & 4 steps for a fixed length or until we get the <EOS> token.

→ you can reject <UNK> or <OOV> token if you mind finding it in your output.

→ So far we have to build word-level language model. It's also possible to implement a character-level language model.

↳ word-level Vocabulary list =

[a, an, and, aaron, ..., zulu, <UNK>, <EOS>]

↳ character-level Vocabulary list =

[a, b, c, d ... z, A, B, C, D ... Z, 1, 2, 3 ... 9, 0]

→ In character-level language model, the vocabulary will contain [a-zA-Z0-9], punctuation, special characters and possibly token.

→ In character-level language model, we give input as: "cats average 15 ... <EOS>"

$y^{<1>} y^{<2>} y^{<3>} y^{<4>} y^{<5>} \ldots \ldots y^{<T_y>}$

→ character-level language model has some pros and cons compared to the word-level language model.

↳ Pros:

→ There will be no <UNK> token; it can create any word.

*enlivo
Date: __/__/____
Page: _____

↳ Cons:

→ The main disadvange is that you end-up with much longer sequences

→ Character-level language model are not as good as word-level language models at capturing long range dependencies between how the earlier parts of the sentence also affect the later part of the sentence.

→ Also, more computationally expensive and harder to train.

⇒ The trend Andrew Ng Sir has seen in NLP is that for the most part, a word-level language model is still used, but as computers get faster there are more and more applications where people are, at least in some special cases, starting to look at more character-level models. Also, they are used in specialized applications where you might need to deal with unknown words or other vocabulary words a lot. Or they are also used in more specialized application where you have a more specialized vocabulary.

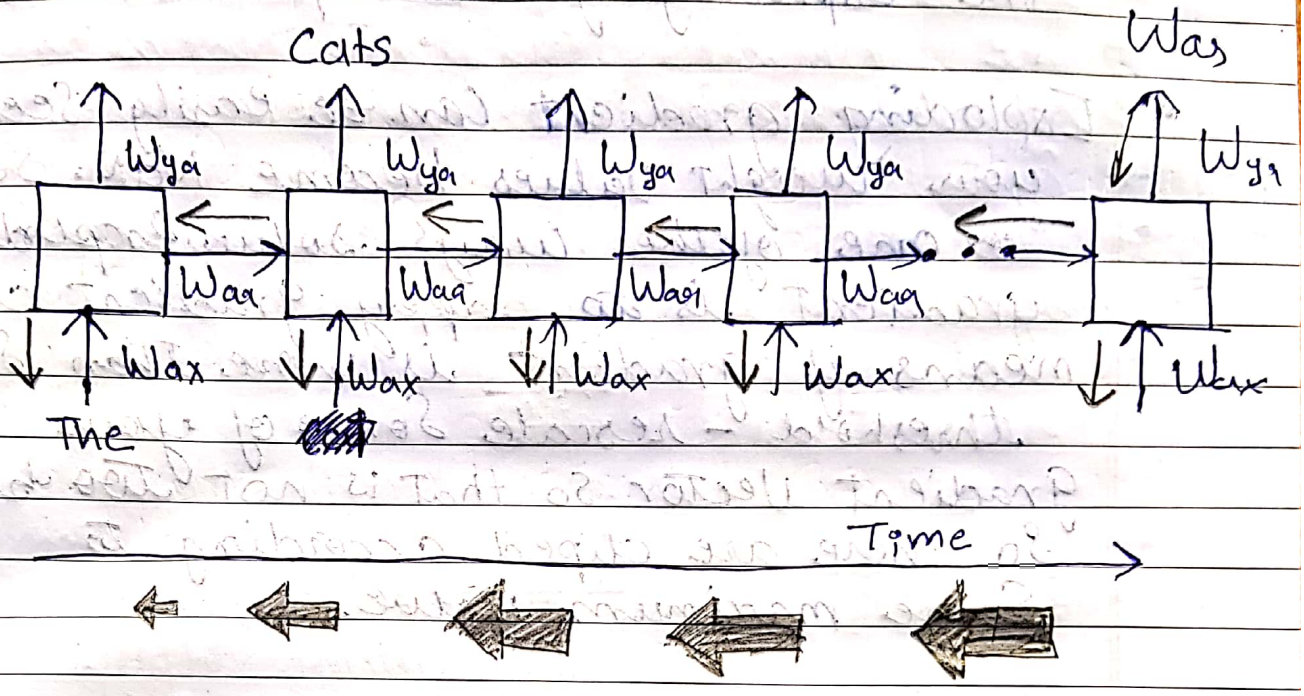
★ Vanishing gradients with RNNs

- One of the problems with a naive RNNs that they run into Vanishing gradient problem.
- An RNN that process a sequence data with the size of 10000 time steps, has 10,000 deep layers which is very hard to optimize.
- Let's take an example. Suppose we are working with language modeling problems and there are two sequences that model tries to learn:

→ "The cat which already ate ..., was full"
→ "The cats which already ate ..., were full."
⇒ Dots ~~repren~~ represent many words in between.

- What we need to learn here that "Was" came with "cat" and that "were" came with "cats". The naive RNN is not very good at capturing very long-term dependencies like this.
- As we have discussed in Deep neural networks, deeper networks are getting into Vanishing gradient problem.

That also happen with RNNs with a long sequence size.



→ For computing the word "Was", we need to compute the gradient for everything behind. Multiplying fraction tends to Vanish the gradient, which means no updation of weight at early time steps.

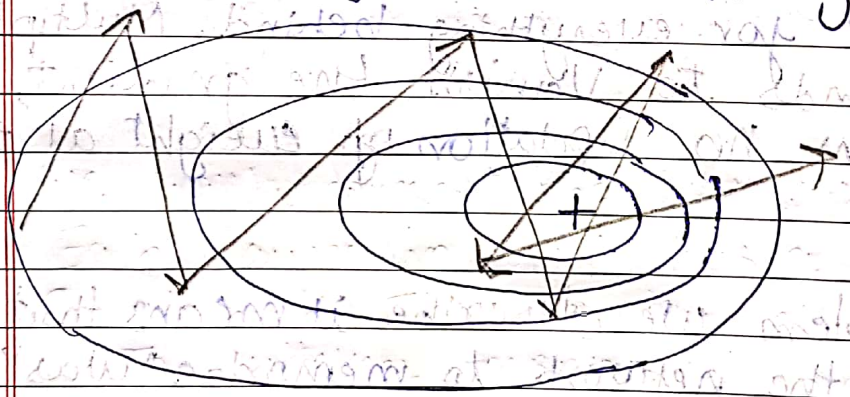
→ In the problem we describe it means that its hard for the network to memorize "Was" word all over back to "cats". so in this case, the network won't identify the singular/plural words so that it gives it the right grammar form of verb Was/were.

→ The conclusion is the RNN are not good in long-term dependecies because some of the weight may not be update properly.

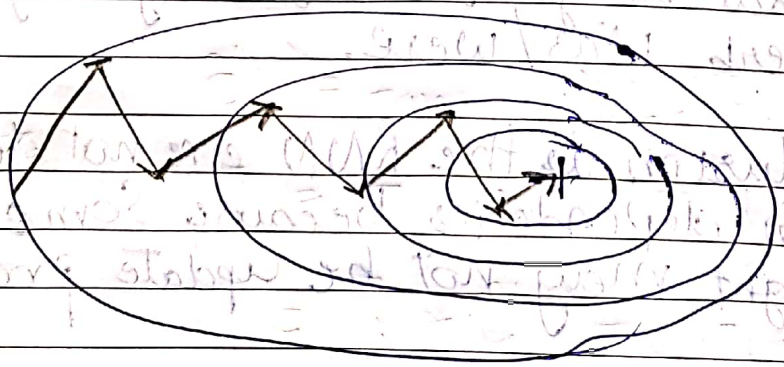
- Vanishing gradient problem tends to be the bigger problem with RNNs than the exploding gradients problem.

- Exploding gradient can be easily seen when your weight values become NaN. So one of the ways solve exploding gradient is to apply gradient clipping. means if gradient is more than some threshold - rescale some of your gradient vector so that is not too big. So there are cliped according to some maximum value.

Without gradient clipping.



With gradient ~~clipping~~ clippingclipping.



✯ Extra:

→ Solution for the Exploding gradient problem:

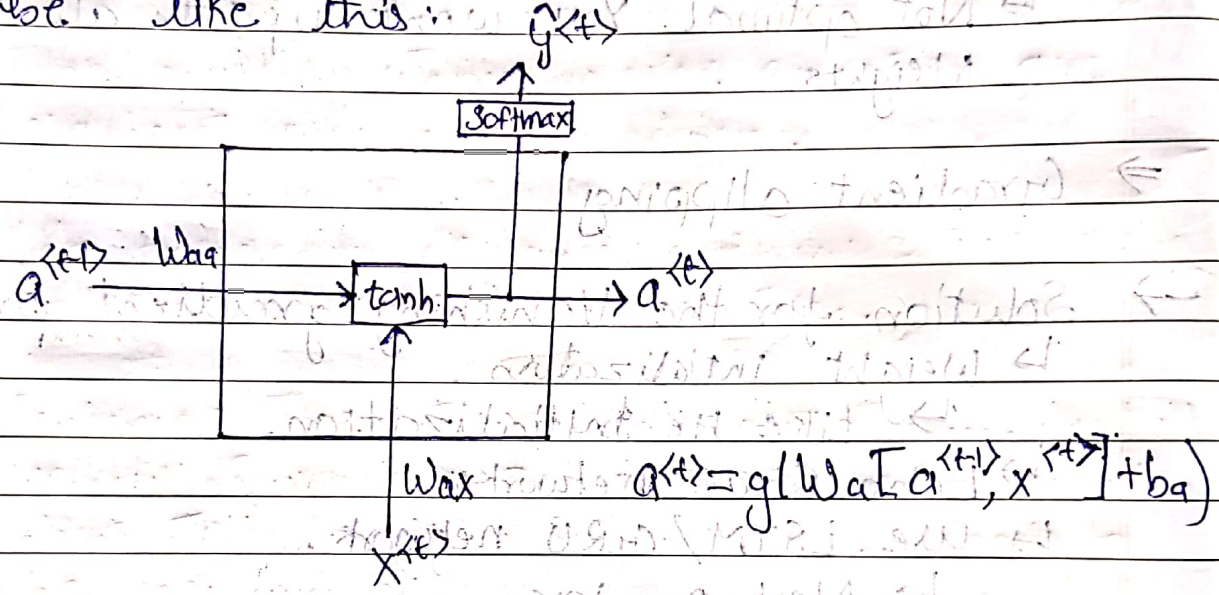
⇒ Truncated backpropagation.
- ↳ Not to update all the weights in the way back.
- ↳ Not optimal. You won't update all the weights.

⇒ Gradient clipping.

→ Solution for the Vanishing gradient problem.
- ↳ Weight initialization.
  - ↳ Like He Initialization.
- ↳ Echo State networks.
- ↳ Use LSTM / GRU network.
  - ↳ Most popular

☆ Gated Recurrent Unit (GRU)

- GRU is an RNN type that can help solve the vanishing gradient problem and can remember the long-term dependencies.
- The basic RNN can be visualized to be like this:



- We will represent the GRU with similar drawings.
- Each layer in GRUs has a new variable C which is the memory cell. it can tell to whether memorize something or not.
- In GRU, $C^{\langle t \rangle} = a^{\langle t \rangle}$.
- Equations

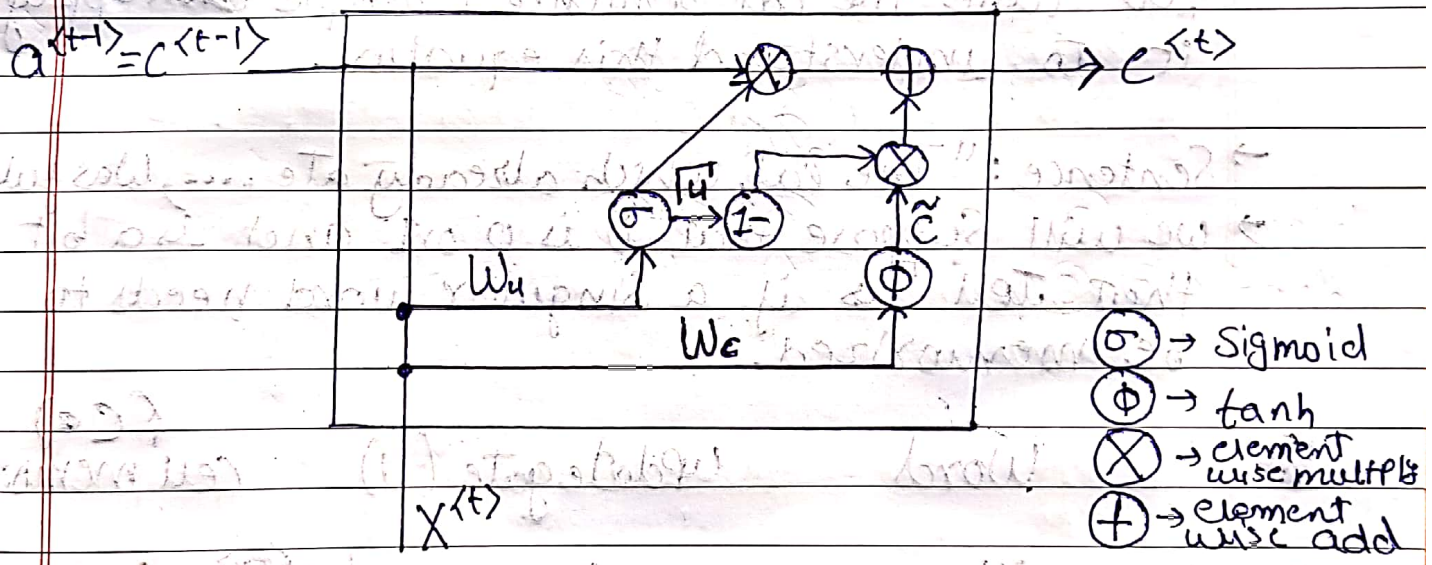
$$\tilde{C}^{\langle t \rangle} = \tanh\left(W_c [C^{\langle t-1 \rangle}, X^{\langle t \rangle}] + b_c\right)$$

update gate → $$\Gamma_u = \sigma\left(W_u [C^{\langle t-1 \rangle}, X^{\langle t \rangle}] + b_u\right)$$

$$C^{\langle t \rangle} = \Gamma_u * \tilde{C}^{\langle t \rangle} + (1 - \Gamma_u) * C^{\langle t-1 \rangle}$$

↳ The update gate is between 0 and 1.
↳ To understand GRUs, imagine that the update gate is either 0 or 1 most of the time.

- So we update the memory cell based on update cell and the previous cell.
- Drawing of the GRU (Simplified).



- Because the update gate Γ_u is usually a small number like 0.00001, GRUs doesn't suffer the vanishing gradient problem.

↳ In the equation this makes $c^{\langle t\rangle} = c^{\langle t-1\rangle}$ in a lot of ~~cases~~ cases.

Note : Update gate Γ_u can also be written as u

- Shapes :
  - → $a^{\langle t \rangle}$ shape is (No. of hidden Neurons)
  - → $c^{\langle t \rangle}$ is the same as $a^{\langle t \rangle}$
  - → $\tilde{c}^{\langle t \rangle}$ is the same as $a^{\langle t \rangle}$
  - → $u^{\langle t \rangle}$ is also same dimension as of $a^{\langle t \rangle}$
- The multiplications in the equation are element wise multiplication.
- Lets take the cat Sentence example and apply it to understand this equation.
  - → Sentence : "The cat, which already ate ...., Was full"
  - → we will Suppose that u is 0 or 1 and is a bit that tell us if a Singular word needs to be memorized.

Word	Update gate (u)	$\Gamma c^{\langle}$ cell memory
The	0	Val
Cat	1	new_value
which	0	new_value
already	0	new_value
....	0	new_value
Was	1	newer_value
full		

⇒ What has been describe so far is the Simplified GRU unit. Let's describe the full one:

- The full ~~GRU~~ GRU contains a new gate that is used with to calculate the candidate c. The gate tells you how relevent is $c^{\langle t-1 \rangle}$ to $c^{\langle t \rangle}$

→ Equations :

$$\Gamma_u = \sigma\left(W_u\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$$

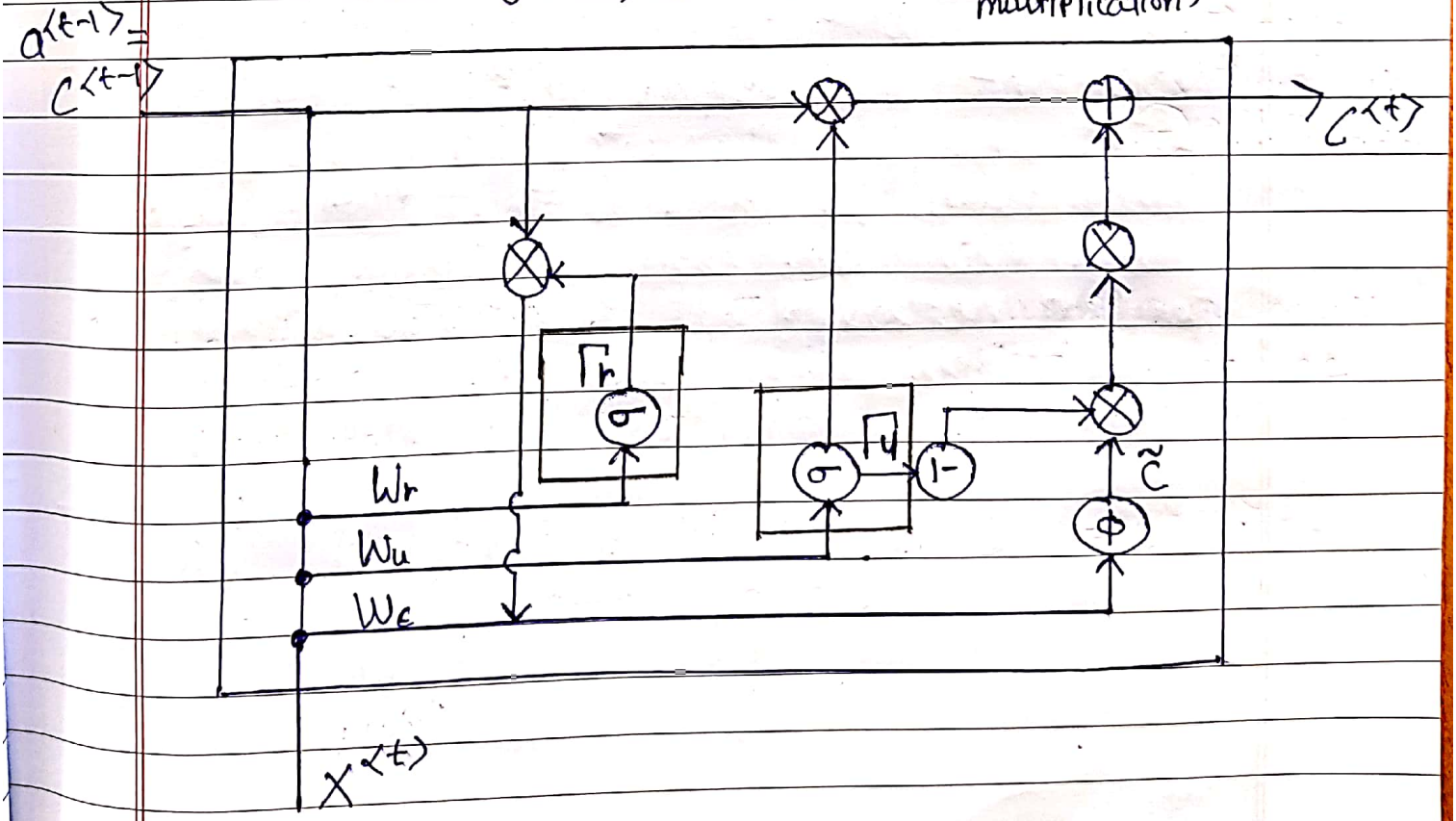
$$\Gamma_r = \sigma\left(W_r\left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_r\right)$$

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c\left[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$$

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

→ Diagram

(σ) → Sigmoid, (Φ) → tanh, (⊗) → element wise multiplication, (⊕) → element wise addition



- Shapes are the Same.
- So why we use these architectures why don't we change them, how we know they will work, why not add another gate, why not use the Simpler GRU instead of the full GRU well researchers has experimented over years all the various types of these architectures with many different version and also addressing the vanishing gradient problem. They have found that full GRUs are one of the best RNN architecture to be used for many different problems. you can make your design but put in mind that GRUs and LSTMs are Standards.