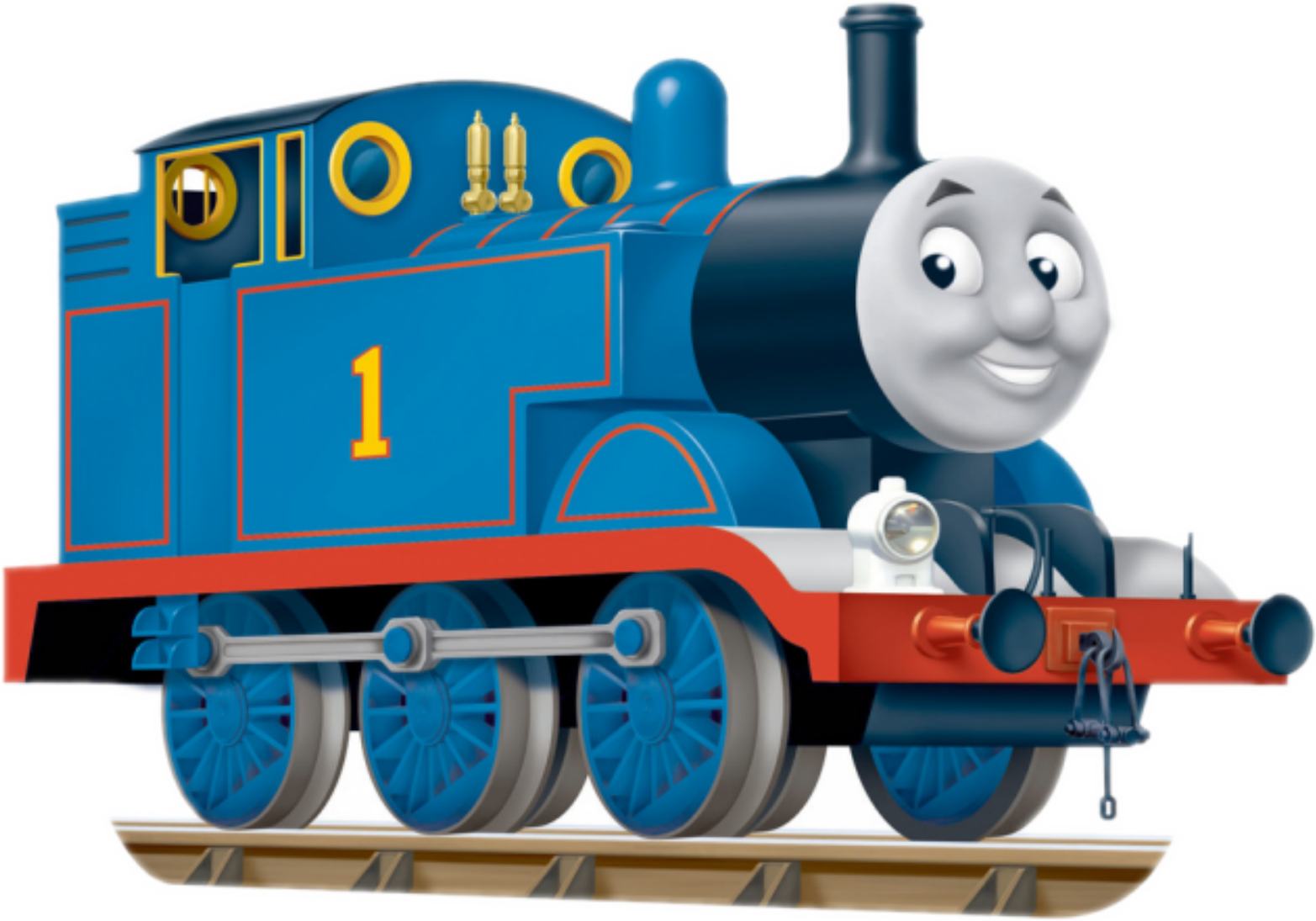
SEQUENCE MODELS

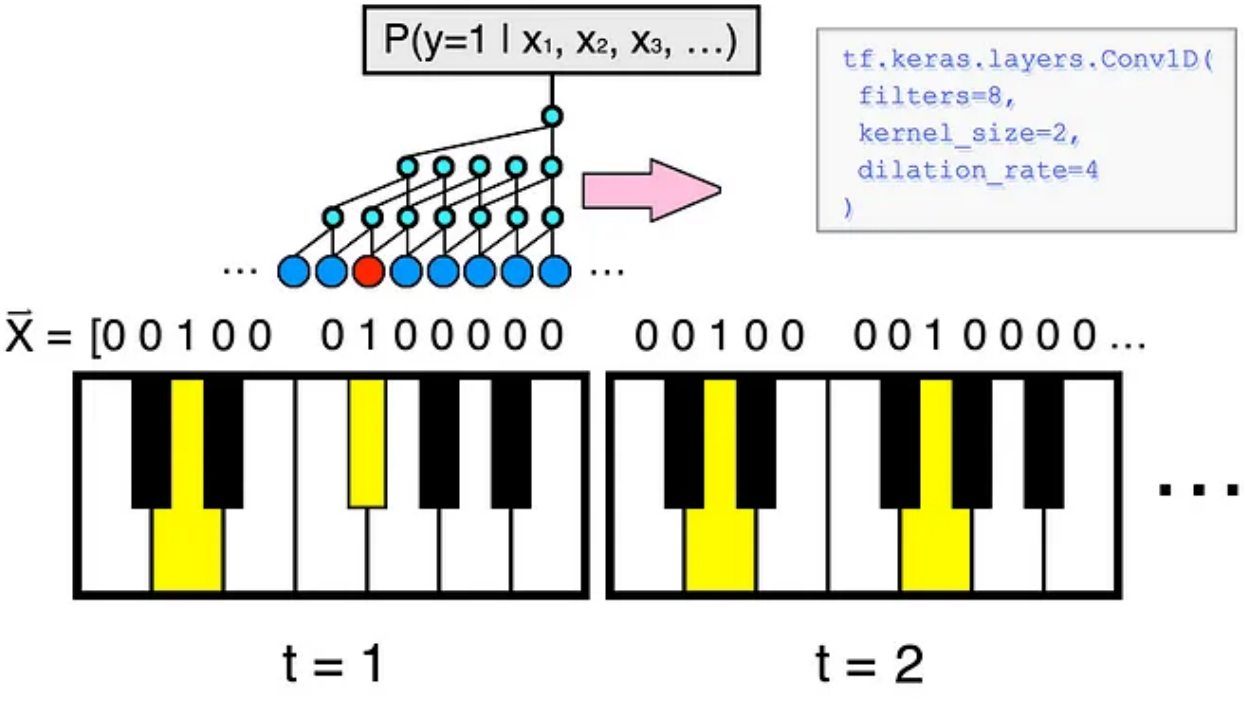
ARIHARASUDHAN



SEQUENCE MODELS

Models like Recurrent Neural Networks (RNNs) have changed speech recognition,natural language processing and other areas. In speech recognition you are given an input audio clip X and asked to map it to a text transcript Y. Both the input and the output here are sequence data, because X is an audio clip and so that plays out over time and Y, the output, is a sequence of words.

Music generation is another example of a problem with sequence data. In this case, only the output Y is a sequence,the input can be the empty set, or it can be a single integer, maybe referring to the genre of music you want to generate or maybe the first few notes of the piece of music you want. But here X can be nothing or maybe just an integer and output Y is a sequence.



In sentiment classification the input X is a sequence, so given the input phrase like, "There is nothing I like in this tweet", what could be the sentiment here ?

Sequence models are also very useful for DNA sequence analysis. Our DNA is represented via the four alphabets A, C, G and T. And so given a DNA sequence,

we can label which part of this DNA sequence say corresponds to a protein.

In machine translation you are given an input sentence, and you're asked to output the translation in a different language.

In video activity recognition you might be given a sequence of video frames and asked to recognize the activity. And in name entity recognition you might be given a sentence and asked to identify the people in that sentence. So all of these problems can be addressed as supervised learning with label data X, Y as the training set. But, as you can tell from this list of examples, there are a lot of different types of sequence problems. In some, both the input X and the output Y are sequences,and in that case, sometimes X and Y can have different lengths.

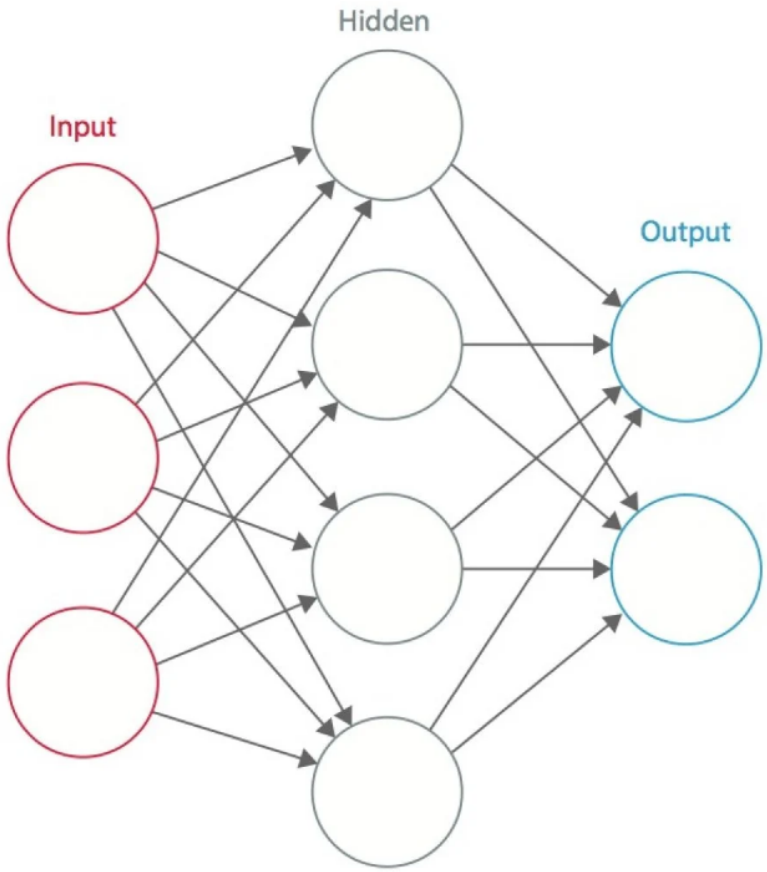
NAMED ENTITY TAGGING

Let’s say we have a sentence like , “I am Aravind and he is Vicky”. Here, we have to find out the named entities which are Aravind and Vicky. How can we do this ? We usually have a vocabulary / dictionary of size n. We look up at that vector and finds out whether the word in the given text is there or not in name category or parts of speech. The names can be given 1 and the else will be given 0.

[I am Aravind and he is Vicky ::: 0 0 1 0 0 0 1]

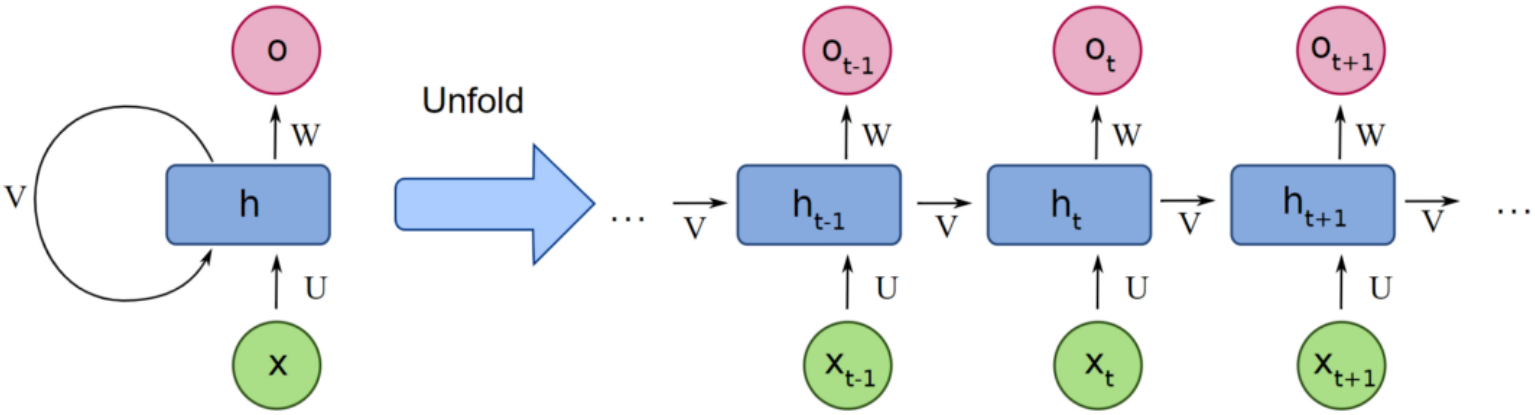
WHY NOT CONVENTIONAL ?

The conventional networks can not be used for working with sequential data as the sequential data may vary in length. We can’t alter the input and output layer size of the network each time. We want things learned for one part of the data to generalize quickly to other parts of the data.



THE SEQUENCE MODEL

The disadvantages in the conventional neural network can be overcome using the sequence networks such as RNN,LSTM and so on. In Recurrent Neural Network, the output activation is again fed for the further computation.

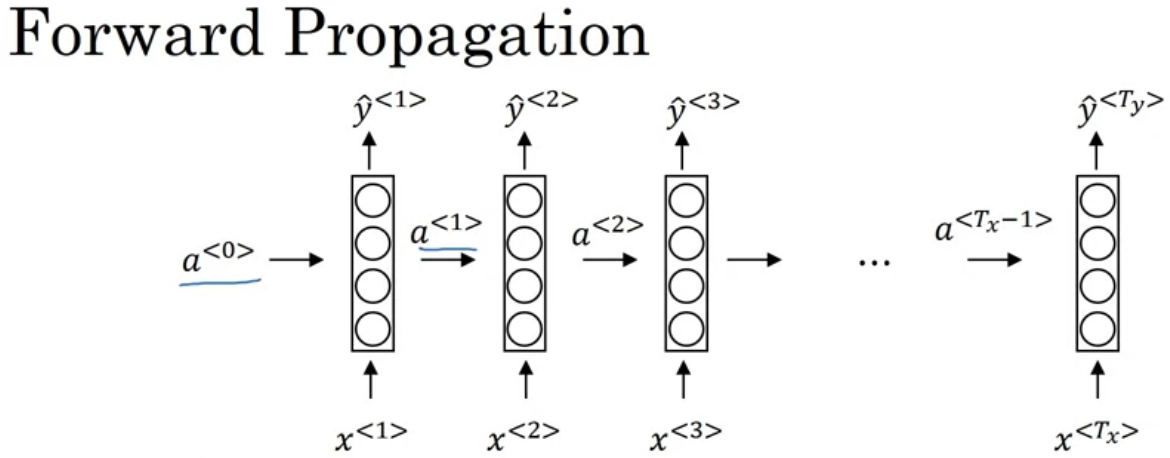


The network can be unrolled like the above. The problem with an Unidirectional RNN is that it only takes account of the earlier.

For an instance, if there is a sentence like “Teddy Young is a wrestler”, The RNN will take each word from the left and process it. It can’t guess what could be the next without processing the right parts. “Teddy Bear is on sale” is also a such a sentence. We can’t conclude that the Teddy in the first sentence is a person without processing it in the right part. To address this issue, there is the so-called BRNN or the Bidirectional Recurrent Neural Network which we’ll be addressing later in this book.

NOTATIONS IN RNN

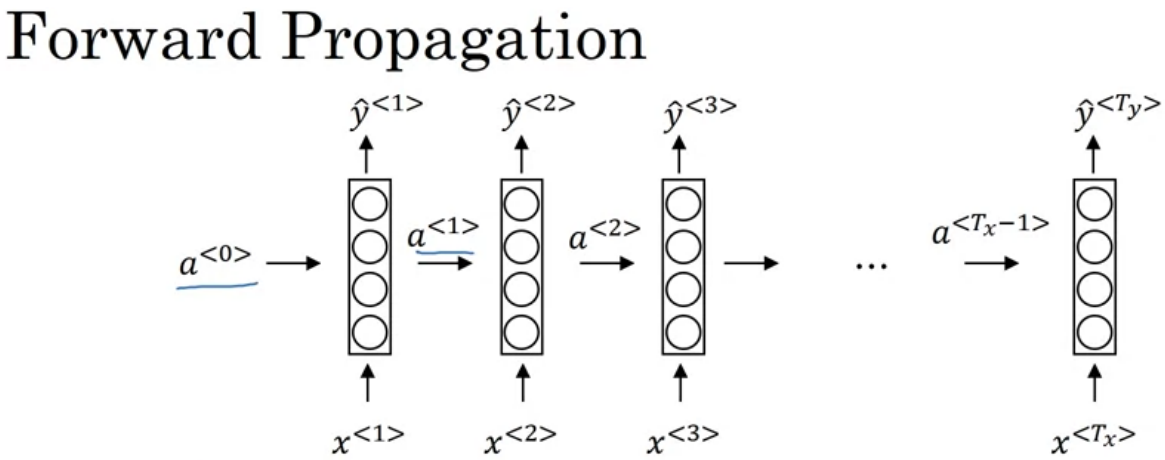
In an Unidirectional RNN, we have some terms to consider.



$a^{<0>}$ is the empty activation values or randomly initialized vector of activation. $x^{<1>}$ is the input to the network which is a part of a sequential data. $y^{hat<1>}$ is the output of the network which is fed the $x^{<1>}$. $a^{<1>}$ is the activation vector from the network which is given to the next network.

EXPRESSIONS IN RNN

As in the conventional neural networks, here also we have some expressions for the activation and the output values.



$$a^{<1>} = g(w_{ax}x^{<1>}+w_{aa}a^{<0>}+b_a)$$
$$y^{hat<1>} = g(w_{ay}a^{<1>}+b_y)$$
$$a^{<2>} = g(w_{ax}x^{<2>}+w_{aa}a^{<1>}+b_a)$$
$$y^{hat<2>} = g(w_{ya}a^{<2>}+b_y)$$

It is obvious from these expressions that these expressions take in account of the previous expression.

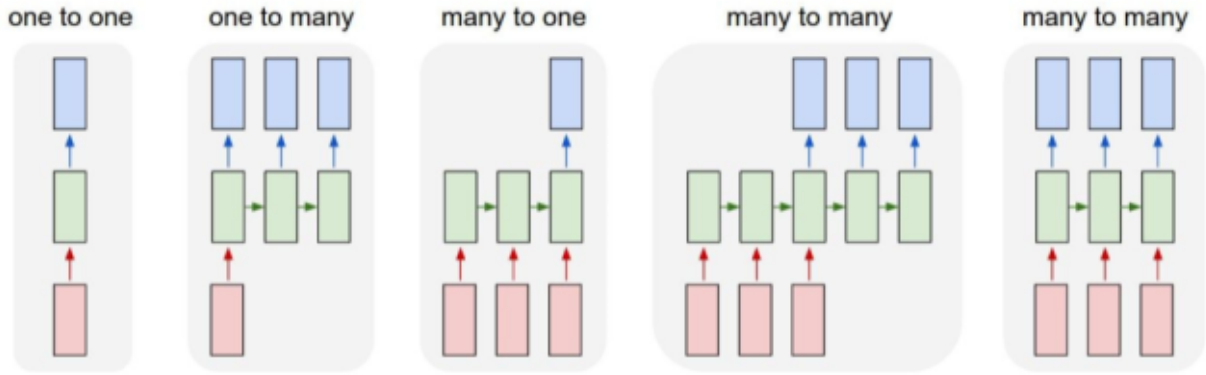
tanh and sigmoid are used respectively for the a and y expressions. tanh lasts for a long and sigmoid is for the binary classification.

Simplified RNN notation

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

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

DIFFERENT RNN STYLES



SOME ARCHITECTURAL USES :

One To One : Vanilla Neural Network

One To Many : Image Captioning

Many To One : Sentiment Analysis

Many To Many : Machine Translation

BACK-PROPAGATION IN RNN

Backpropagation Through Time, or BPTT, is the application of the Backpropagation training algorithm to recurrent neural network applied to sequence data like a time series.

We can summarize the algorithm as follows:

1. Present a sequence of timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across each timestep
3. Roll-up the network and update weights.
4. Repeat.

BPTT can be computationally expensive as the number of timesteps increases.

LANGUAGE MODELLING

If there are two sentences like, “I am here” and “I am hear”, which one should be assigned more probability? i.e., which one is the most correct ? We, humans can easily say that the first one is correct. But, the network should be able to pick the correct sentence.

A Language Model helps us here. It gives the sentence with high probability.

P(“I am here”) = 0.9

P(“I am hear”) = 0.09

HOW TO BUILD SUCH MODEL?

>> We need a large corpus of text (Dataset)

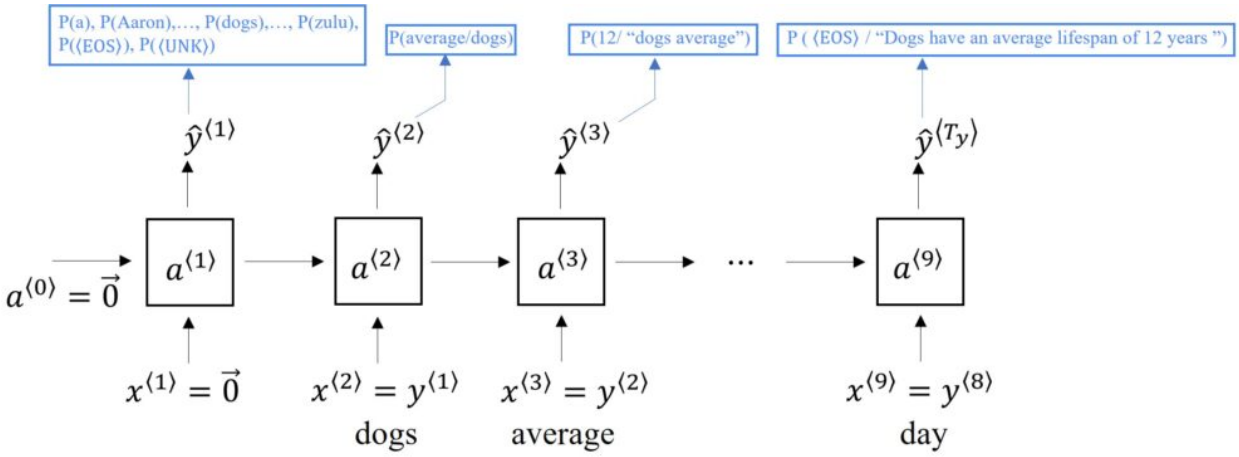
>>Tokenize (Vocab-Mapping of each words in the sentence)

>> EOS token should also be added in the end of the sentence to indicate the end
>> If the word in the given sentence is not there in our vocabulary, then the token for it should be UNK. In such case, we will model the chances for the unknown word and not for the actual word.
>> Then, build an RNN to get the sentence probabilities.

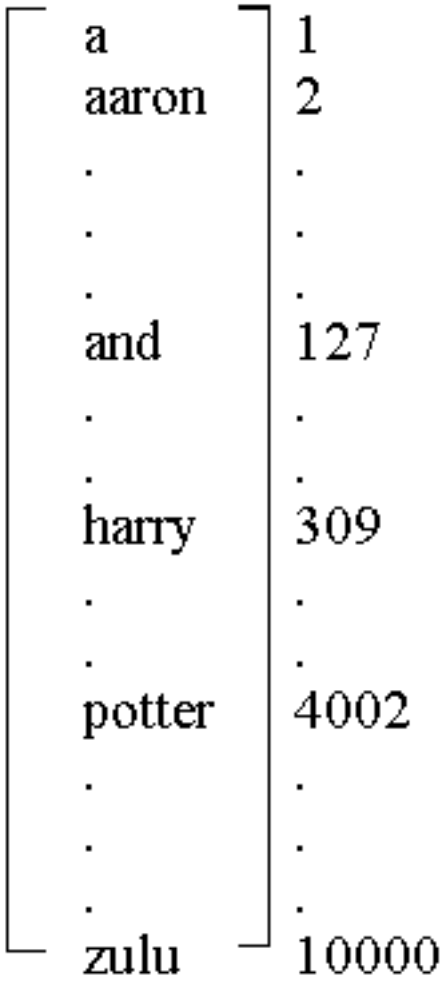
THE RNN

After understanding how tokenization of input sequence is done, we can now get on with building the RNN architecture for our training example sentence: “Dogs have an average lifespan of 12 years.”

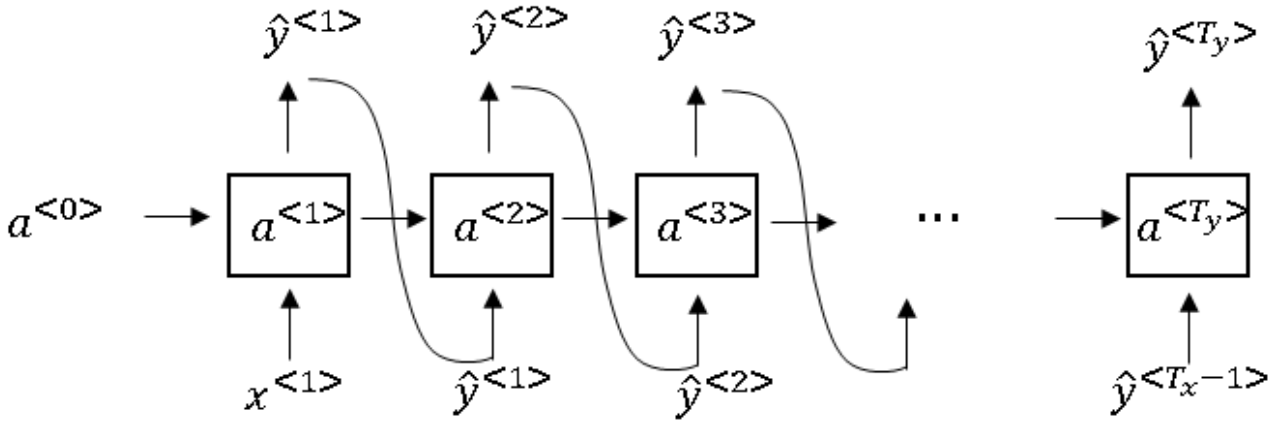
This is what the RNN architecture looks like:

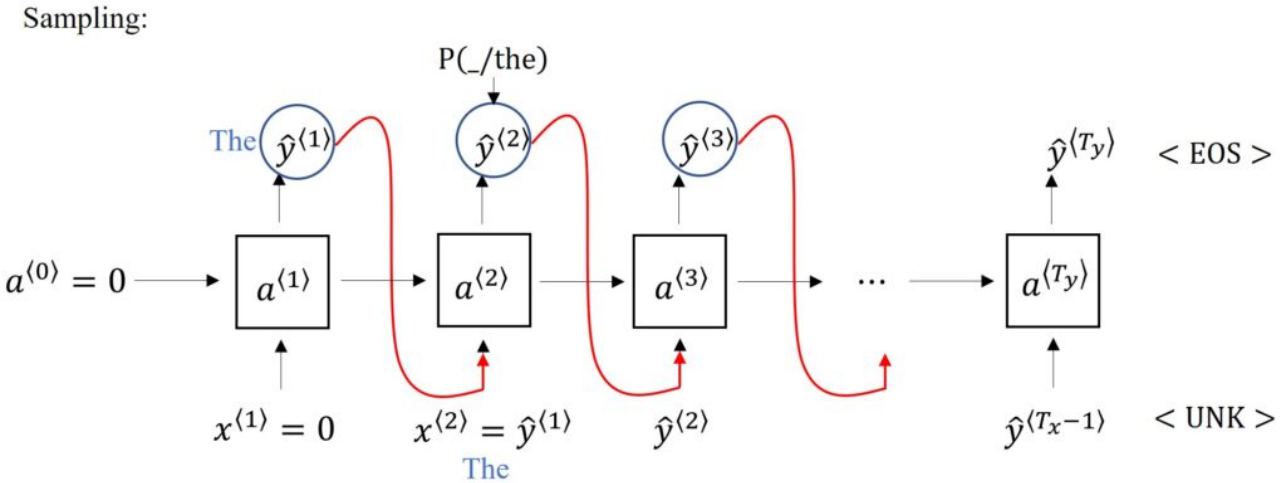


SAMPLING NOVEL SENTENCE



After trained an RNN, let’s check whether it generates a novel-meaningful sentence or not. So, if the first word given is “The”, the model is expected to generate next word based on the given first word.





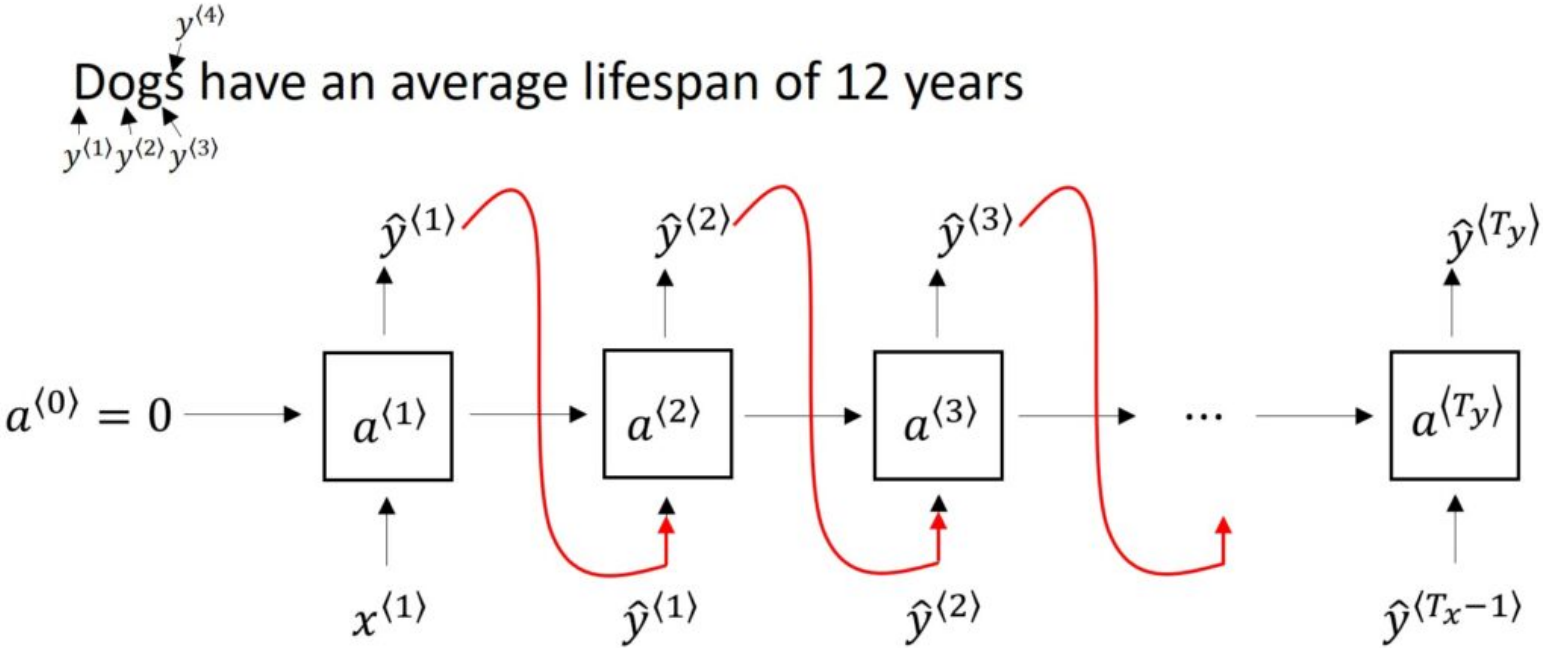
CHARACTER LEVEL MODEL

An alternative to a Words-level Language Model uses characters instead of words to form a dictionary. This means that the vocabulary only contains alphabets from “a” to “z” along with space, punctuation, and digits from “0” to “9”. We could also distinguish our alphabets by segregating them according to Uppercase and Lowercase within our dictionary. We can decide what to include in this character-level dictionary based on the kind of characters that appear in our training set.

Vocabulary = [a, Aaron, ... zulu, < UNK]

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

$y^{\langle 1 \rangle}$ $y^{\langle 2 \rangle}$ $y^{\langle 3 \rangle}$ -individual characters



VANISHING GRADIENT IN RNN

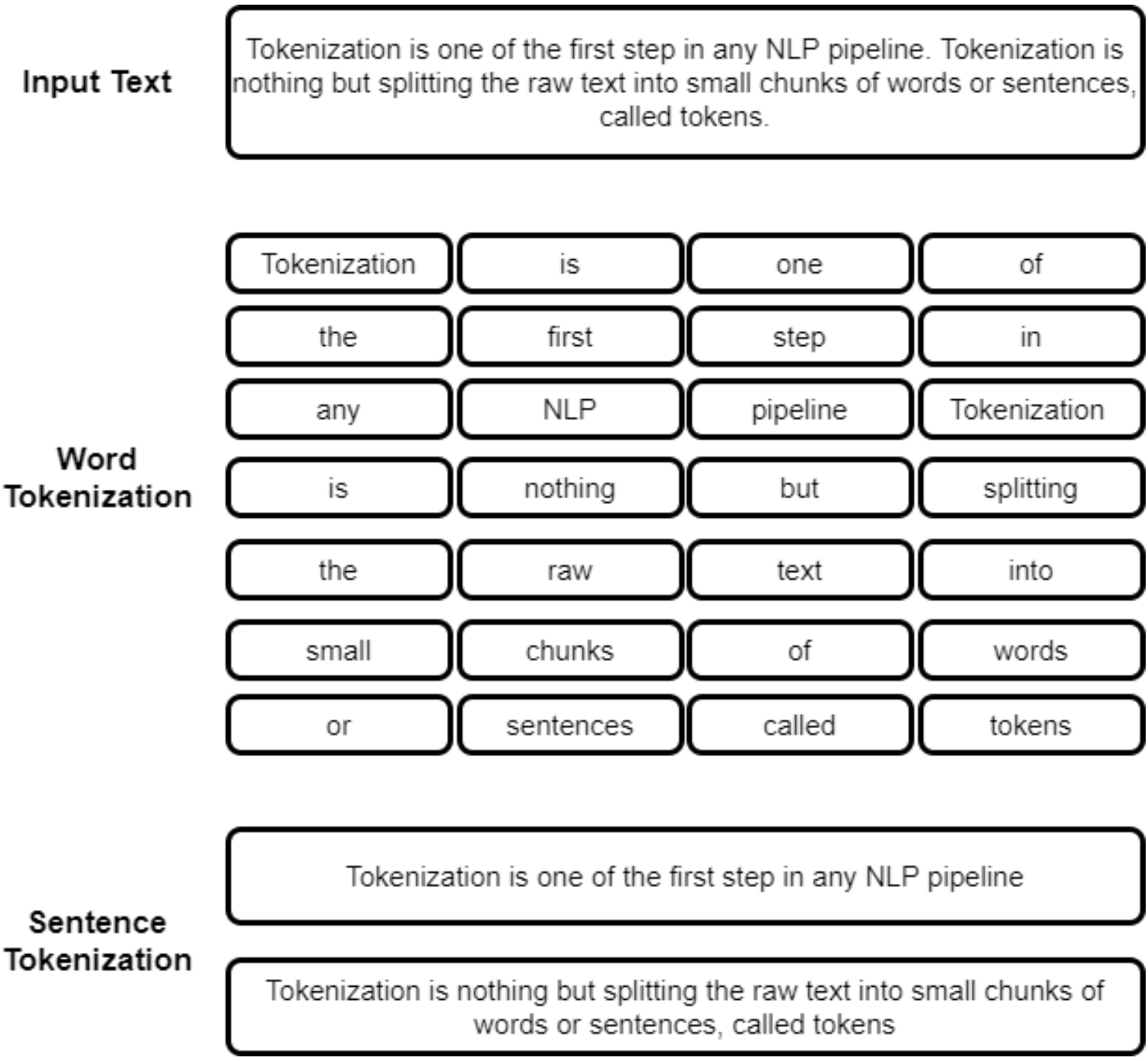
The vanishing gradient problem occurs when the backpropagation algorithm moves back through all of the neurons of the neural net to update their weights. The nature of recurrent neural networks means that the cost function computed at a deep

layer of the neural net will be used to change the weights of neurons at shallower layers. This change is multiplicative, which means that the gradient calculated in a step that is deep in the neural network will be multiplied back through the weights earlier in the network. Said differently, the gradient calculated deep in the network is "diluted" as it moves back through the net, which can cause the gradient to vanish - giving the name to the vanishing gradient problem! We go for LSTM to get rid of this problem. Weight Initialization is important in avoiding this problem.

TOKENIZATION

Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens. If the text is split into words, then its called as 'Word Tokenization' and if it's split into sentences then its called as 'Sentence Tokenization'. Generally 'space' is used to perform the word tokenization and characters like 'periods, exclamation point and newline char are used for Sentence Tokenization. We have to choose the appropriate method as per the task in hand.

While performing the tokenization few characters like spaces, punctuations are ignored and will not be the part of final list of tokens.



WHY TOKENIZATION

Every sentence gets its meaning by the words present in it. So by analyzing the words present in the text we can easily interpret the meaning of the text. Once we have a list of words we can also use statistical tools and methods to get more insights into the text. For example, we can use word count and word frequency to find out importance of a word in that sentence or document.

SUB-WORD TOKENIZATION

Sub word tokenization is similar to word tokenization, but it breaks individual words down a little bit further using specific linguistic rules. One of the main tools they utilize is breaking off affixes. Because prefixes, suffixes, and infixes change the inherent meaning of words, they can also help programs understand a word’s function.This can be especially valuable for out of vocabulary words, as identifying an affix can give a program additional insight into how unknown words function.

The sub word model will search for these sub words and break down words that include them into distinct parts. For example, the query “What is the tallest building?” would be broken down into ‘what’ ‘is’ ‘the’ ‘tall’ ‘est’ ‘build’ ‘ing’ How does this method help the issue of OOV words? Let’s look at an example: Perhaps a machine receives a more complicated word, like ‘machinating’ (the present tense of verb ‘machinate’ which means to scheme or engage in plots). It’s unlikely that machinating is a word included in many basic vocabularies. If the NLP model was using word

tokenization, this word would just be converted into just an unknown token. However, if the NLP model was using sub word tokenization, it would be able to separate the word into an ‘unknown’ token and an ‘ing’ token. From there it can make valuable inferences about how the word functions in the sentence.

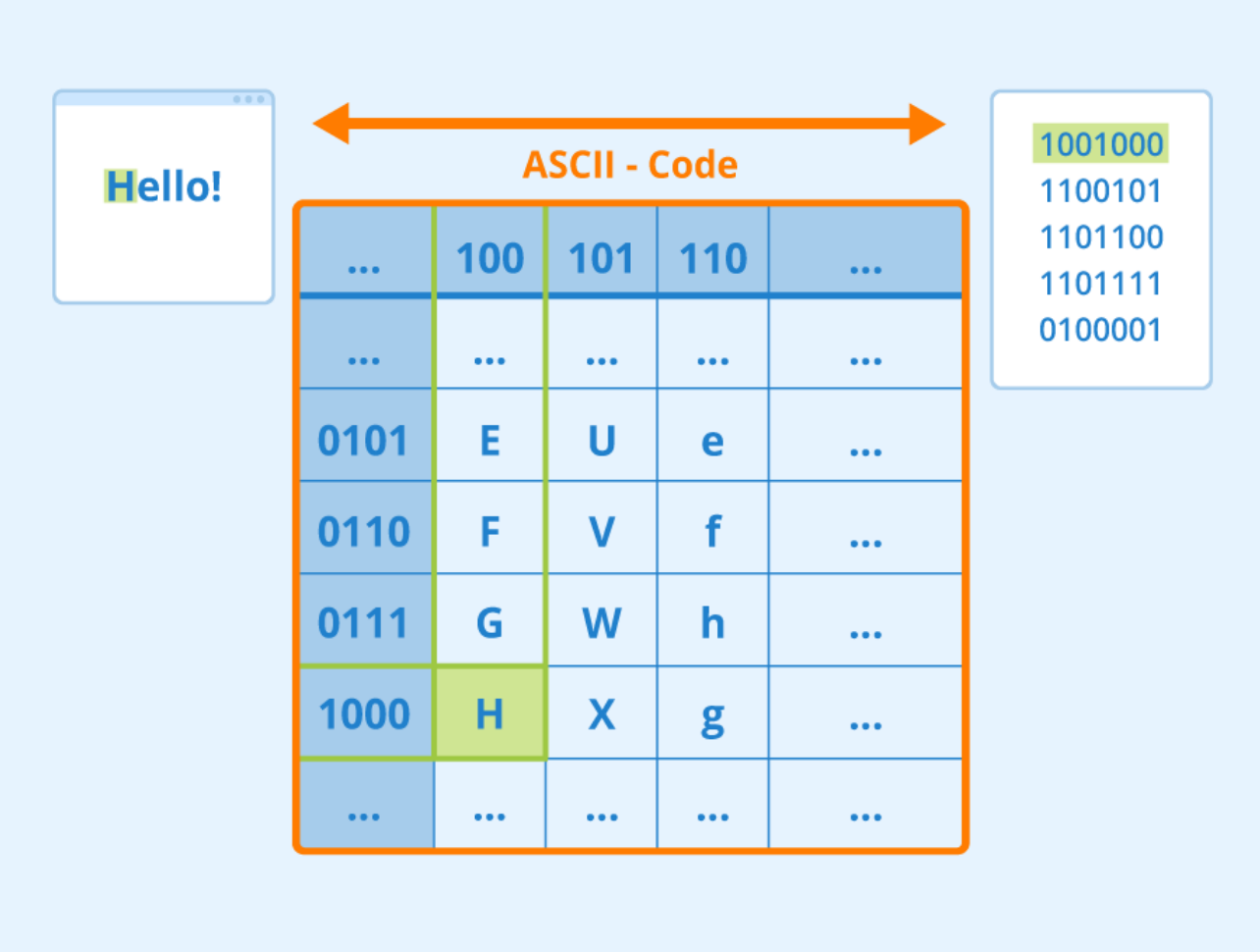
CHARACTER TOKENIZATION

While character tokenization solves OOV issues, it isn‘t without its own complications. By breaking even simple sentences into characters instead of words, the length of the output is increased dramatically.

With word tokenization, our example “what restaurants are nearby” is broken down into four tokens. By contrast, character tokenization breaks this down into 24 tokens, a 6X increase in tokens to work with. Character tokenization also adds an additional step of understanding the relationship between the characters and the meaning of the words. Sure, character tokenization can make additional inferences, like the fact that there are 5 “a” tokens in the above sentence. However, this tokenization method moves an additional step away from the purpose of NLP, interpreting meaning.

REPRESENTING TEXT

If we want to solve Natural Language Processing (NLP) tasks with neural networks, we need some way to represent text as tensors. Computers already represent textual characters as numbers that map to fonts on your screen using encodings such as ASCII or UTF-8.



We understand what each letter **represents**, and how all characters come together to form the words of a sentence. However, computers by themselves do not have such an understanding, and a neural network has to learn the meaning during training. Therefore, we can use different approaches when representing text:

CharacterLevel Representation

when we represent text by treating each character as a number. Given that we have C different characters in our text corpus, the word *Hello* would be represented by 5×C tensor.

WordLevel representation

When we create a **vocabulary** of all words in our text sequence or sentence(s), and then represent each word using one-hot encoding. This approach is somehow better, because each letter by itself does not have much meaning, and thus by using higher-level semantic concepts - words - we simplify the task for the neural network. However, given a large dictionary size, we need to deal with high-dimensional sparse tensors. For example, if we have a vocabulary size of 10,000 different words. Then each word would have an one-hot encoding length of 10,000; hence the high-

dimensional. To unify those approaches, we typically call an atomic piece of text **a token**. In some cases tokens can be letters, in other cases - words, or parts of words.

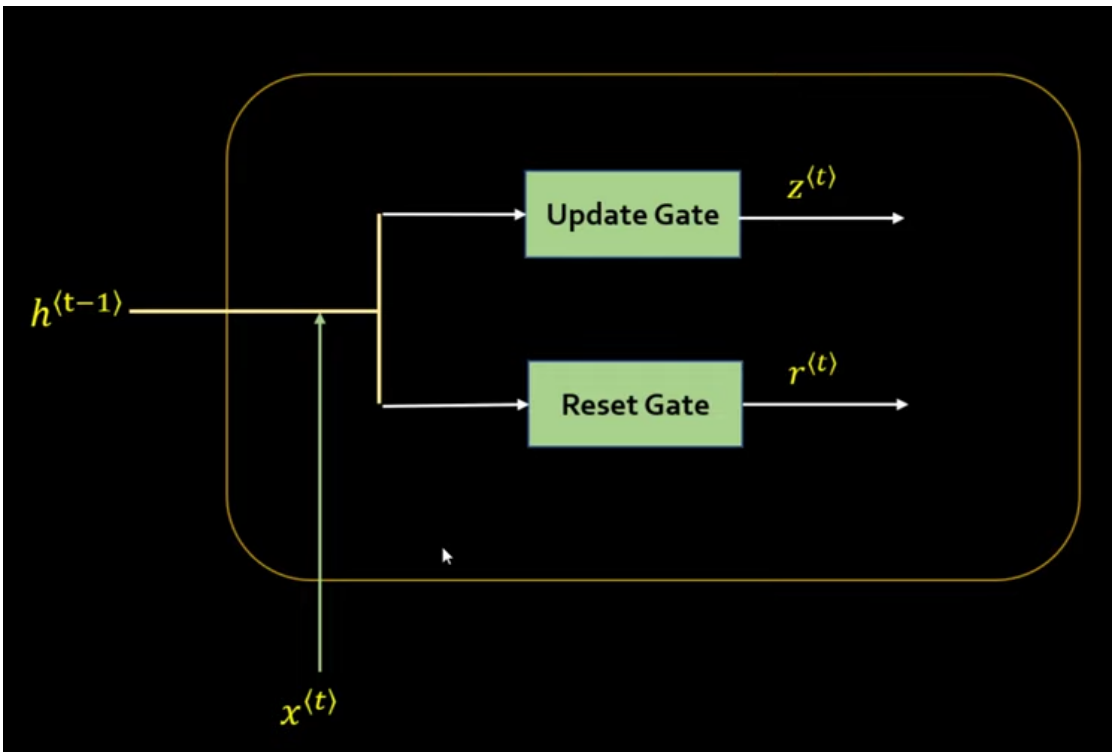
THE GATED RECURRENT UNIT

The Gated Recurrent Unit or GRU as short is a network with a memory which remembers a term for a long time. Assume, we have a sentence like the following:

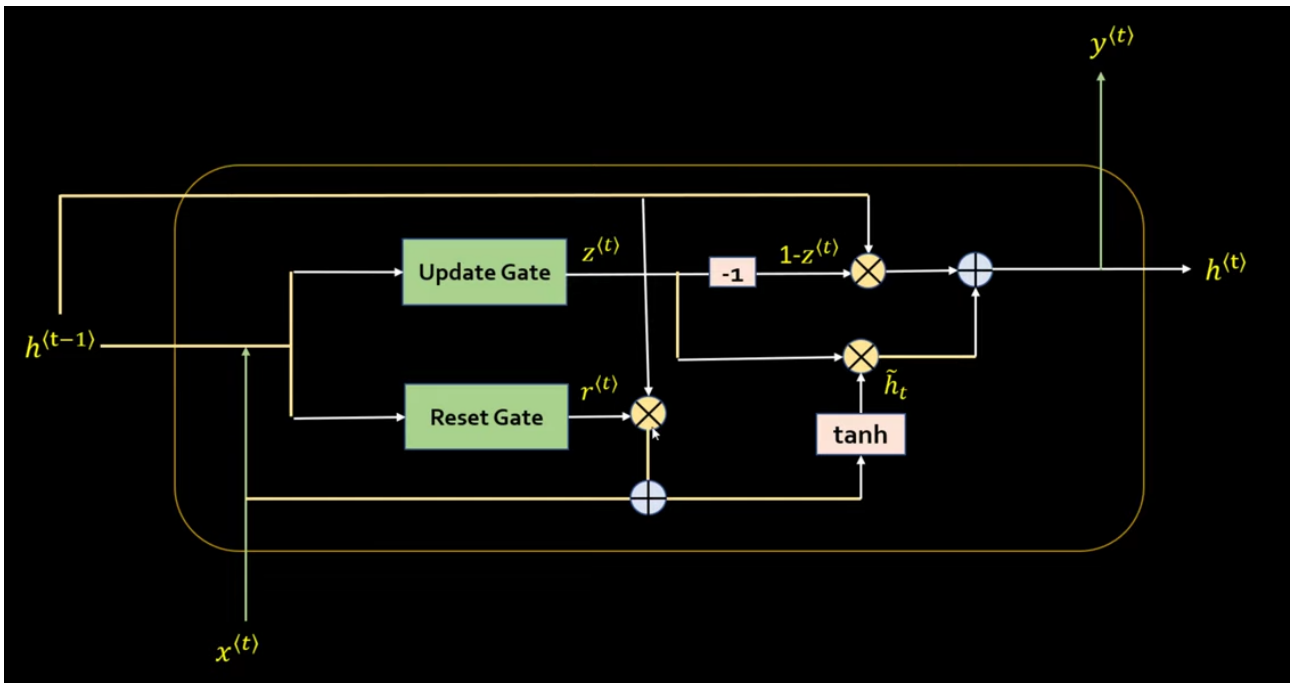
“Mangifera indica is the scientific name of the mango, so clearly, it belongs to ___”

The answer is India.

Here, the network must keep track of the word indica to deduce the answer India. But, in the case of RNN, there is the so-called Short Term Memory Problem which makes the network impossible to remember the word and deduce the answer as India. To overcome this problem, we have to introduce a new sort of network namely, Gated Recurrent Unit or the GRU. It is a bit similar to LSTM Network.



The actual GRU Network looks like the following :



UPDATE GATE :

The Update Gate determines the portions to be remembered from the past data.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

When x_t is plugged into the network unit, it is multiplied by its own weight $W^{(z)}$. The same goes for h_{t-1} which holds the information for the previous

t-1 units and is multiplied by its own weight $U^{(z)}$. Both results are added together and a sigmoid activation function is applied to squash the result between 0 and 1.

RESET GATE :

The Reset Gate determines the portions to be forgotten from the past data.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

This formula is the same as the one for the update gate. The difference comes in the weights and the gate’s usage.

CURRENT CONTENT ON RESET :

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

We introduce a new memory content which will use the reset gate to store the relevant information from the past. It is calculated as shown above.

CURRENT CONTENT ON UPDATE :

As the last step, the network needs to calculate h_t vector which holds information for the current unit and passes it down to the network. In order to do that the update gate is needed. It determines what to collect from the current memory content h'_t and what

from the previous steps $h^{(t-1)}$. That is done as follows:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

LSTM

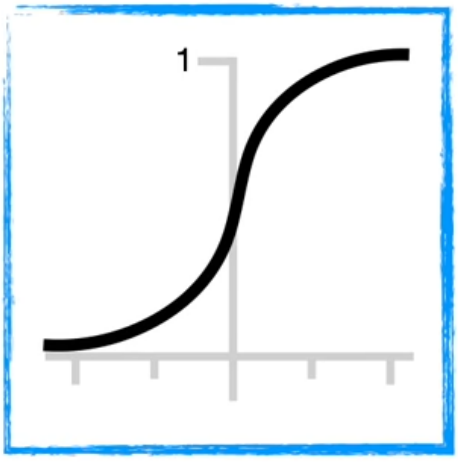
LSTM, abbreviating Long Short Term Memory is a recurrent neural network (RNN) architecture that **REMEMBERS** values over arbitrary intervals. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. One of the advantage with LSTM is insensitivity to gap length.

WHAT'S SPECIAL WITH LSTM ?

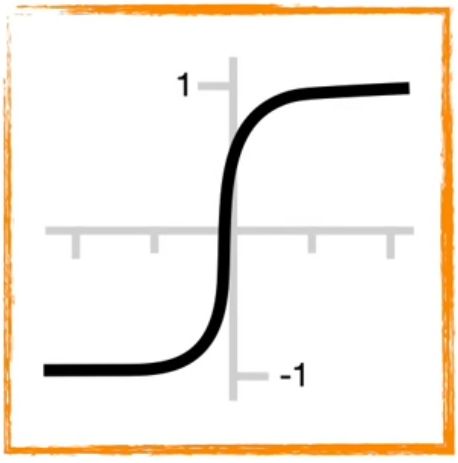
LSTM follows an approach of incorporating paths within. One path is for the Long Term Memory and another is for the Short Term Memory. The basic unit of the LSTM is a bit complicated but easily understandable.

ACTIVATION FUNCTIONS :

So, now that we know that the **Sigmoid Activation Function** turns any input into a number between **0** and **1**...

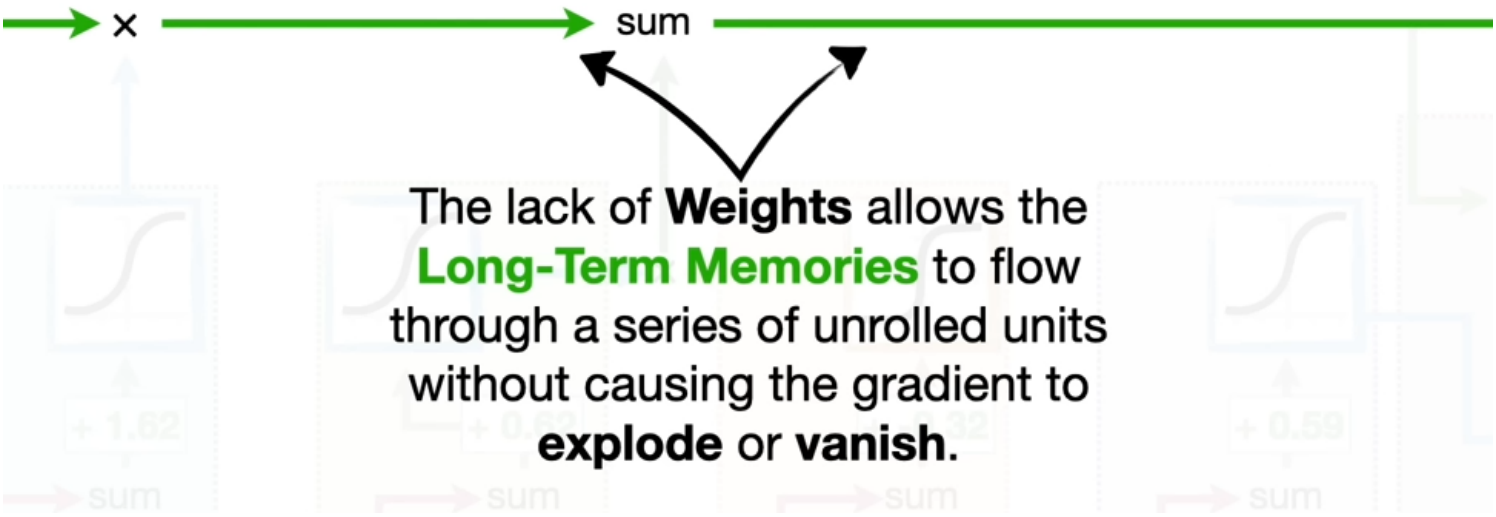


...and the **Tanh Activation Function** turns any input into a number between **-1** and **1**...

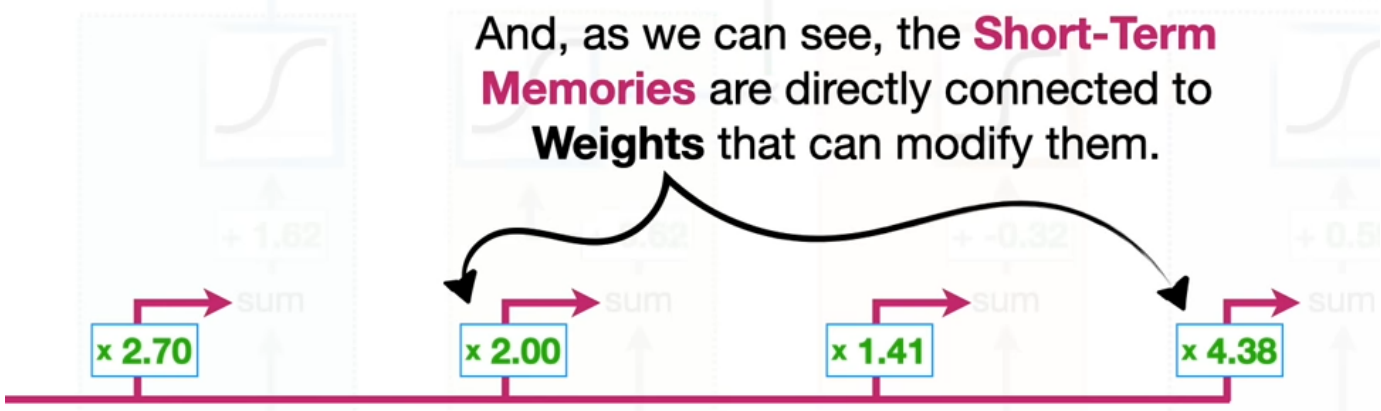


LSTM has two paths of which one is for Long Term Memory and the another one is for Short Term Memory.

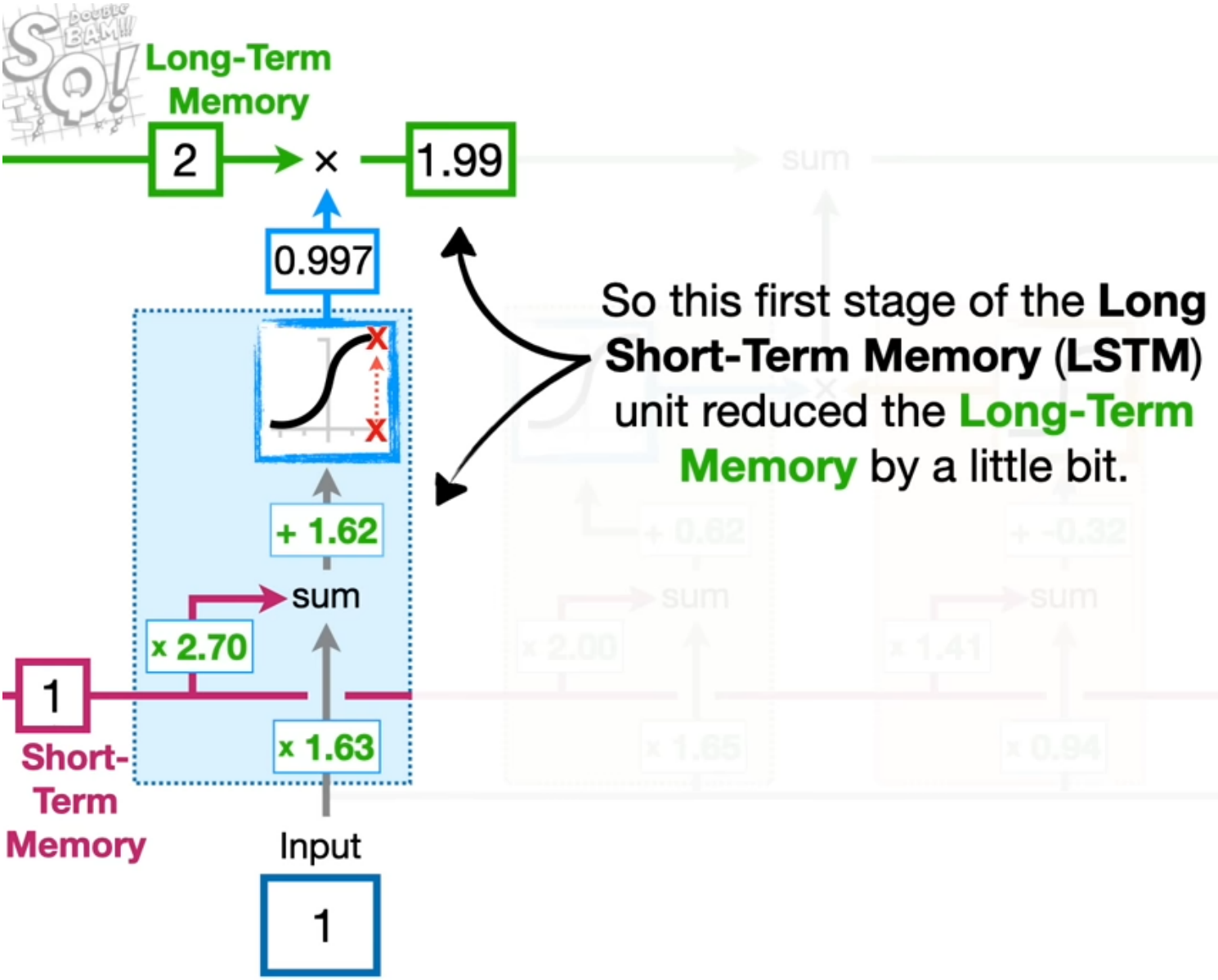
THE LONG TERM MEMORY PATH:



THE SHORT TERM MEMORY PATH:



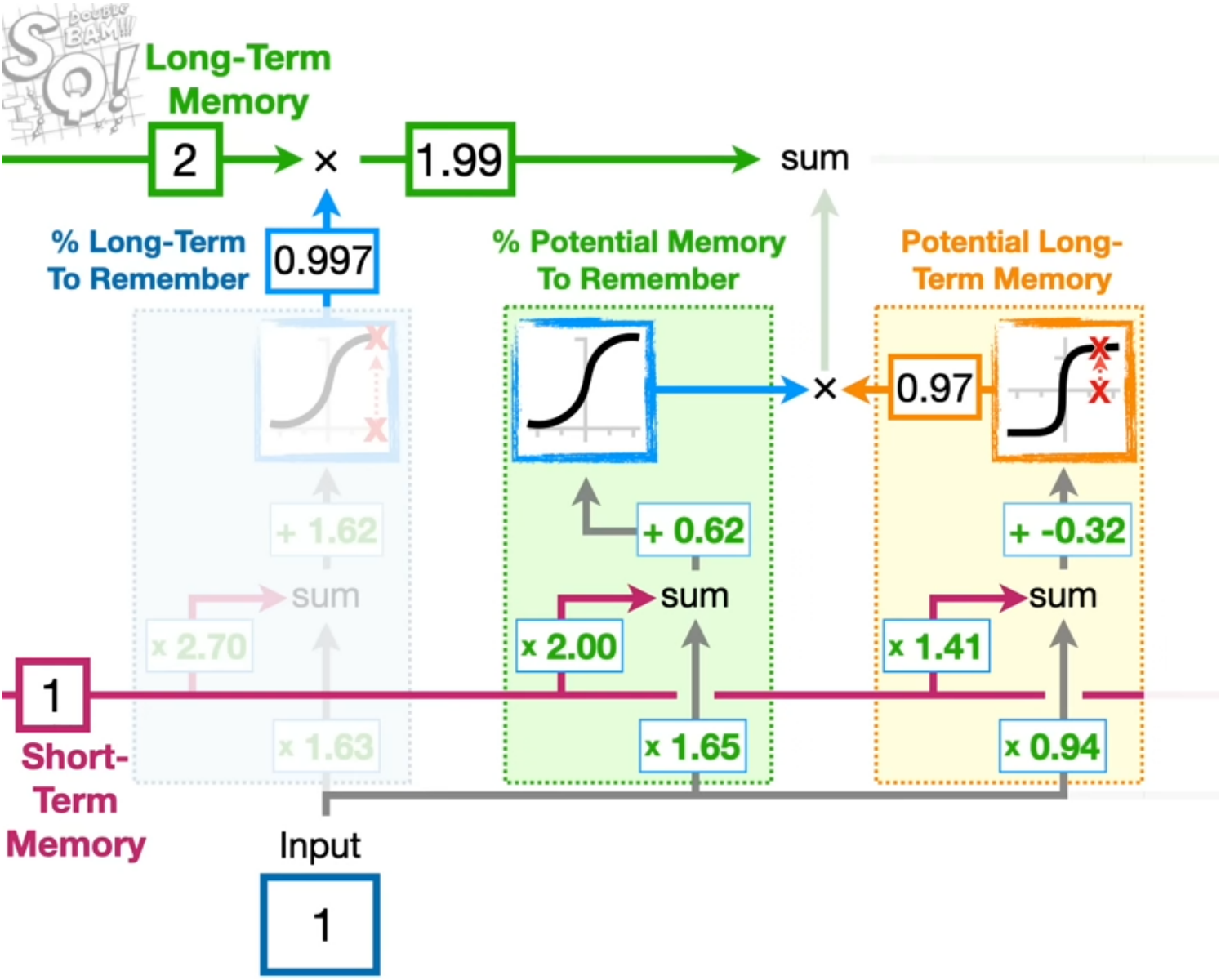
THE FORGET GATE



The short term memory is multiplied by it’s weight and the input is also multiplied by it’s weight which is later added and squished using the sigmoid function.

It determines what percentage of the long term memory should be remembered.

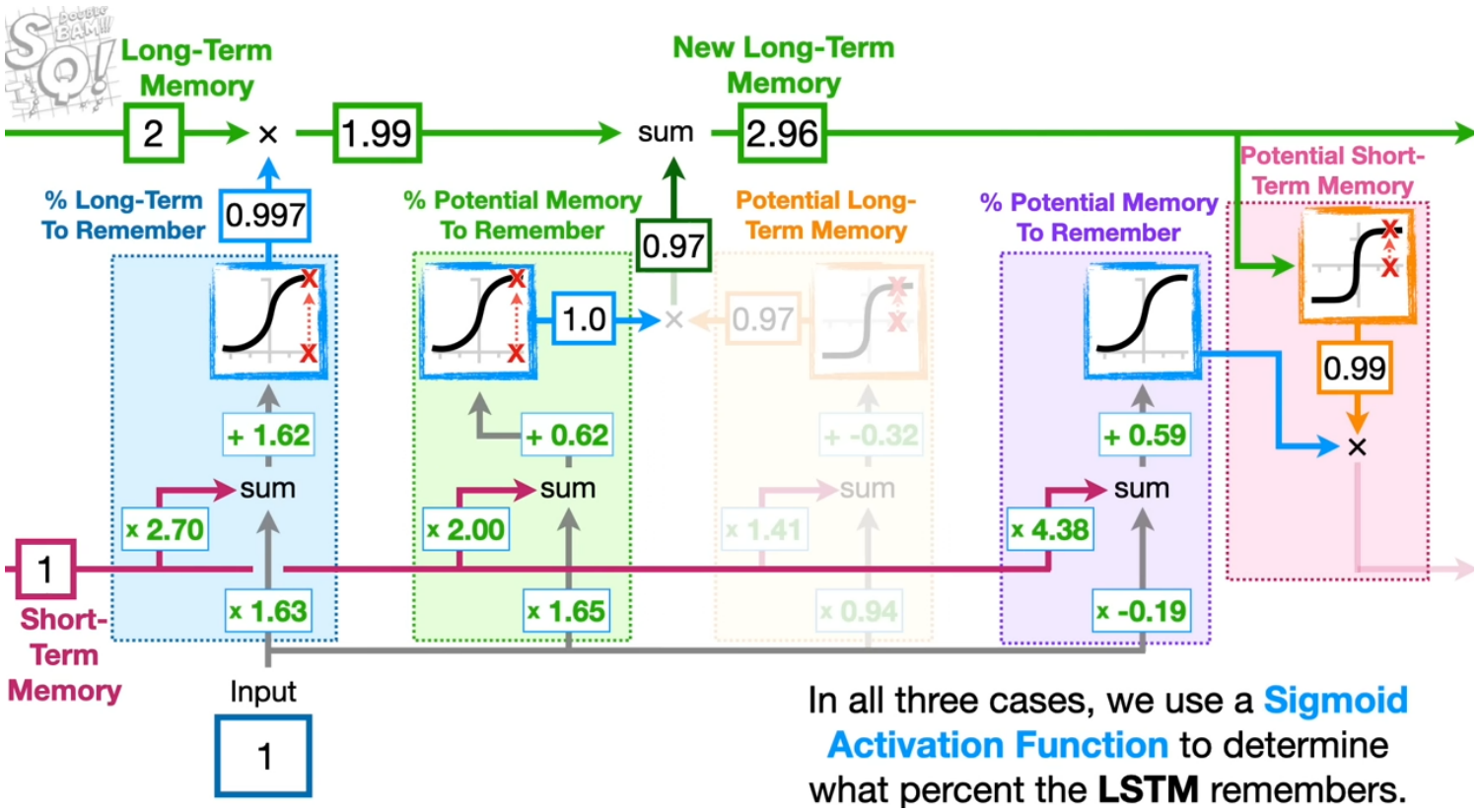
THE INPUT GATE



Here, The Potential Long Term memory to be remembered is computed using tanh activation

function and the exact same thing performed in the previous step.

THE OUTPUT GATE :

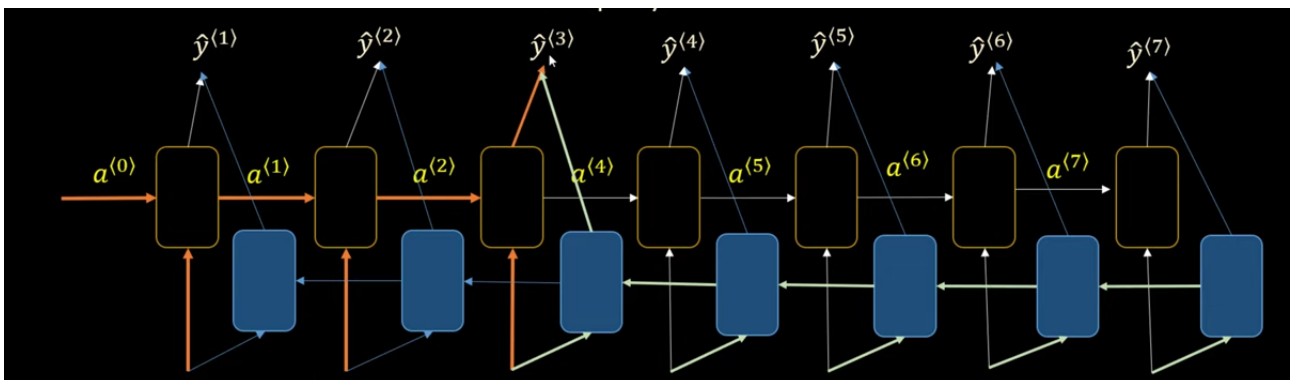


The last arrangements determines what percentage of the potential short term memory should be remembered. LSTM is great tool for anything that has a sequence .

Since the meaning of a word depends on the ones that preceded it. This paved the way for NLP and narrative analysis to leverage Neural Networks.

BIDIRECTIONAL RNN

Bidirectional RNN solves the problem of lacking the future features due to learning the earlier parts only. It is a simple RNN Network connected with another RNN Network which processes from the right and provides features of the future texts for computations.



Let’s say we have a sentences like the following :

“Ari likes kiwi which is a bird” and “Ari likes kiwi which is a citrus fruit”.

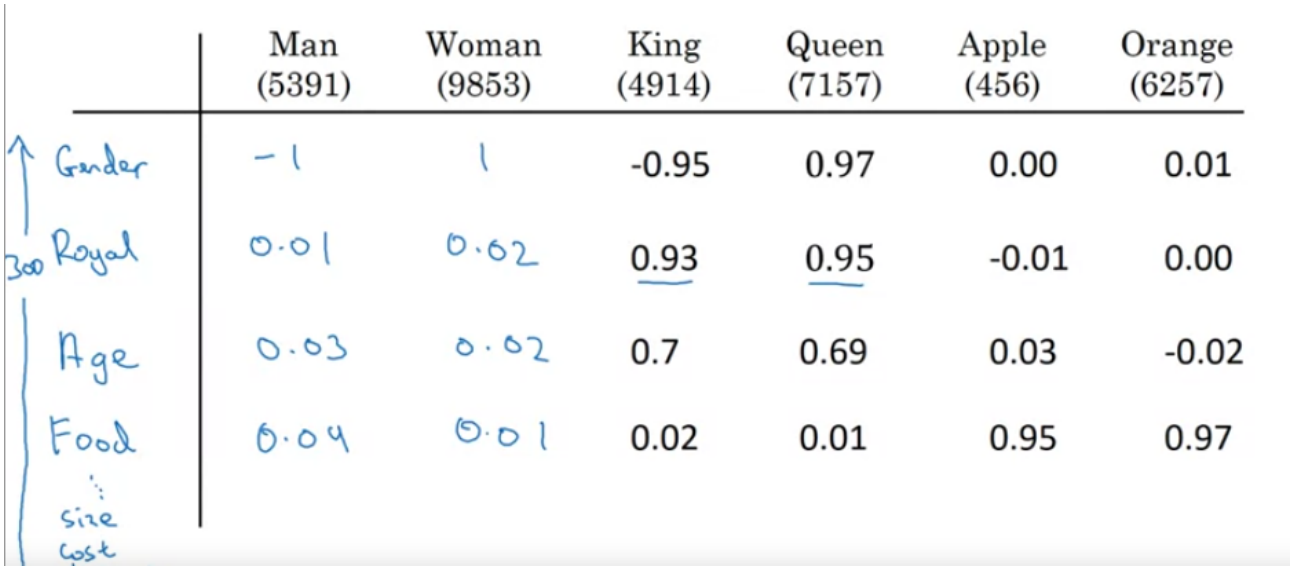
Now, the problem here is, whether the kiwi is meant to be a bird or a fruit while scanning from the left. If the network has scanned till kiwi, can the network identify whether it is a bird or a fruit? Definitely NO! It requires the insight from the right to proceed and that’s when the Bi-RNN plays a role.

WORD REPRESENTATION

Till now, we were using the one-hot encoding. i.e., a vector representing all the vocabulary words is used. If the word we search is there, it will be denoted as 1 and the else will be 0. If the word we search is in 0^{th} position, then the vector will be like, $[1\ 0\ 0\ 0\ 0.....]^T$. But, it's not good when we want to generalize for some inputs. For an instance, if we have , I want to have a cup of apple _____, If the model is able to predict that the word is juice, will it be able to

predict the following ? I want to have a cup of orange _____.

Definitely, it can't on using the one-hot encoding. Because, one hot encoding is not apt for generalization. So, what we can do is, we can use something else like the Featurized Representation.



	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size						
cost						

300

We can use more features for a generalization of results.

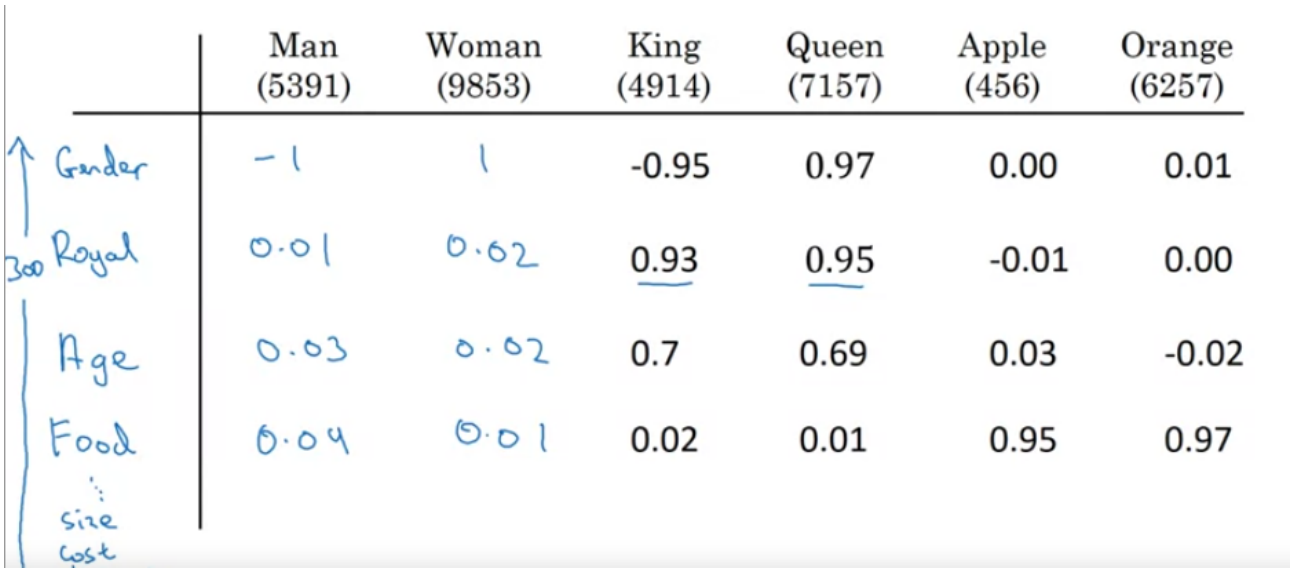
WORD EMBEDDINGS

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meaning to have a similar representation. They can also approximate meaning. A word vector with 50 values can represent 50 unique features. Anything that relates words to one another. Eg: Age, Sports, Fitness, Employed etc. Each word vector has values corresponding to these features. Using the featurized representations, the vectors are generated.

More the closeness between the vectors, more the THAT between the words.

PPTIES OF WORD EMBEDDINGS

First, let's talk of the analogical reasoning. If we are given something like , Man→Woman, what could be King→? The problem comes into the category of analogical reasoning. It can be computed as the following. These



	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size						
cost						

300

columns are our embedding vectors.

Now, we are given, Man→Woman. So, as a first step, let's find the difference between them.

$$e_{man} - e_{woman} = [-2\ 0\ 0\ 0\ ]^T$$

Now, we have to find out the similar matching by subtracting the King vector with each.

$$e_{king} - e_x = [-2\ 0\ 0\ 0\ ]^T$$

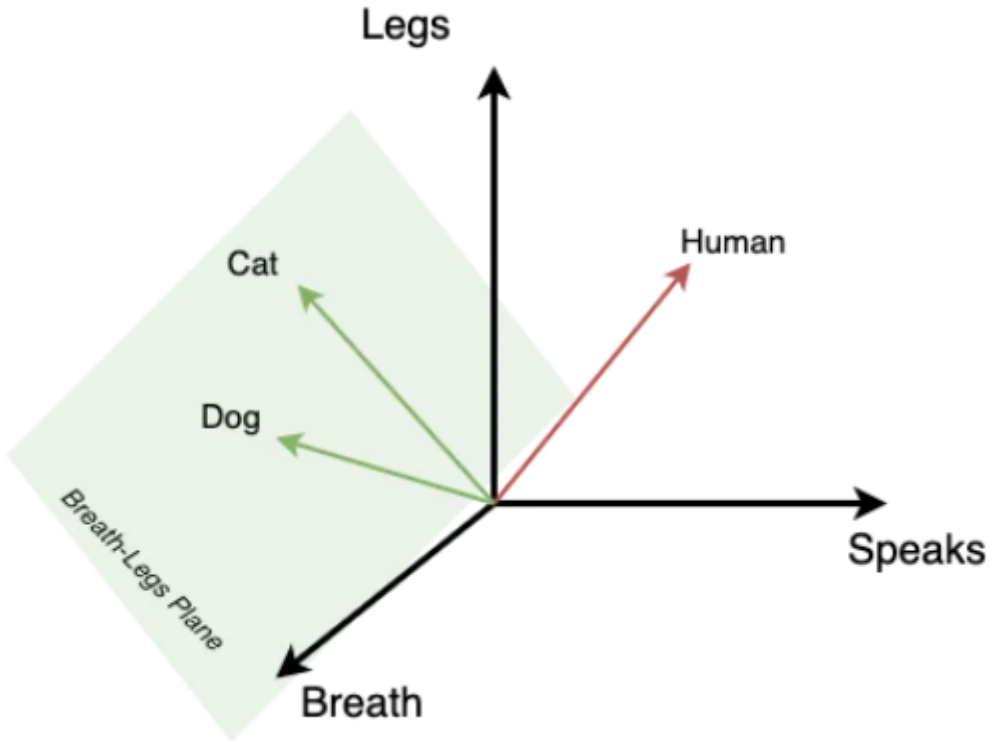
The one with which we have the result vector value equivalent to the one we got on subtracting man embedding with woman is the answer. i.e., King→ANSWER. STRIVE FOR

$$e_{king} - e_x =_{approx} e_{man} - e_{woman}$$

We can formally speak like, we have to find out e_x that maximizes $sim(e_x, e_{king}-e_{man}+e_{woman})$.

COSINE SIMILARITY

The embedding vectors may be in n Dimension.



If we want to find out the similarity between any two vectors, we can go for the usage of the so-called cosine similarity which will give the cosine angle between the vectors and that’s indeed the relationship between the vectors.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

TEXT CORPUS REQUIRED!

In the case of Named Entity Tagging, if we are given sentences as, Jack is an Apple farmer, the model can conclude Jack as a Human as Apple farmer can be processed so. What if we are given a sentence like Jacob is a Camachile Cultivator and those two words at the end are not there in our dictionary ? TRAGEDY! But, camachile is also fruit and cultivator is also a sort of farmer. For such relationship mapping, a large unlabelled text-corpus is required for our model for a broad learning. Such text-corpus are available in net.

EMBEDDING MATRIX

Depends on the way of embedding we incorporate, there are two types of embedding matrix. While we use one-hot encoding, it will be like the following.

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

If we use featurized vectors, it will be like the following.

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1

Each word gets a 1x3 vector

Similar words... similar vectors

PREDICTING NEXT WORD

On having a sentence like the following, we need to predict.

Tiger is my favorite _____.

The prediction should be animal. What is done conventionally is the following : For each word, an embedding vector is found. If Tiger is represented as a one hot vector as O_{2345}, it is multiplied with the Embedding vector and the result is E_{2345}. This operation is performed on each word. Eventually, the E vectors are given to a neural network for which a soft-max is applied at the end which outputs the probabilities.

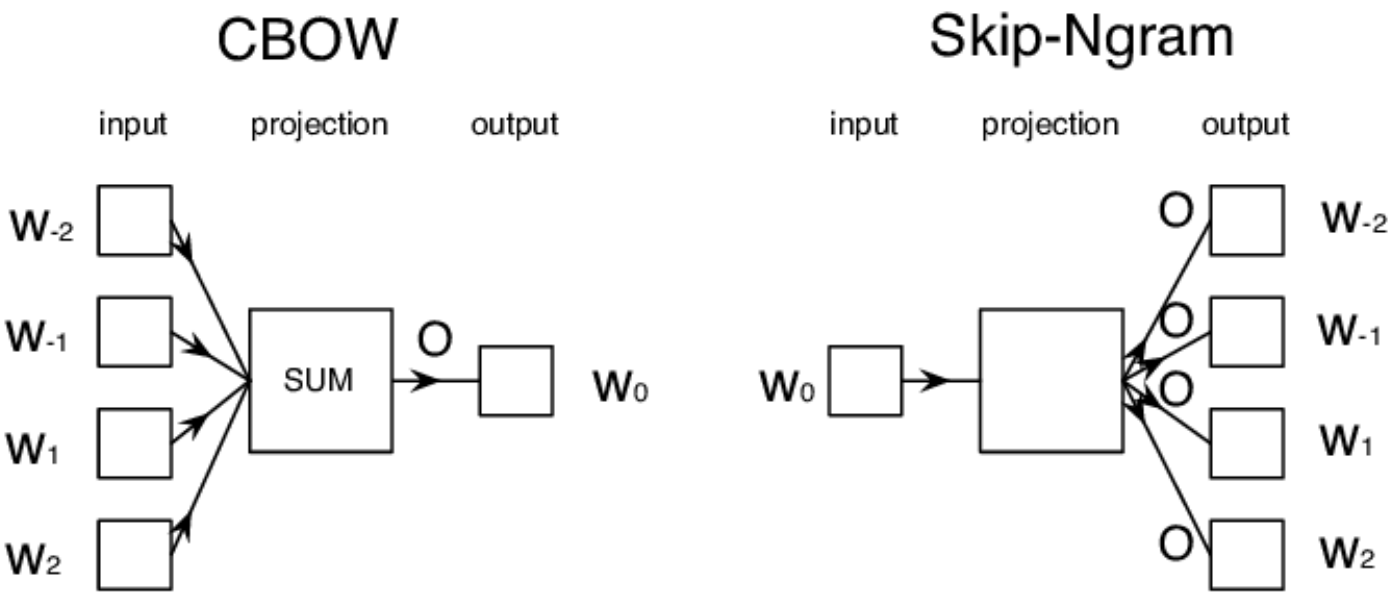
It is followed to keep track of the n (Hyperparameter) words on the left of the target always.

WORD2VEC

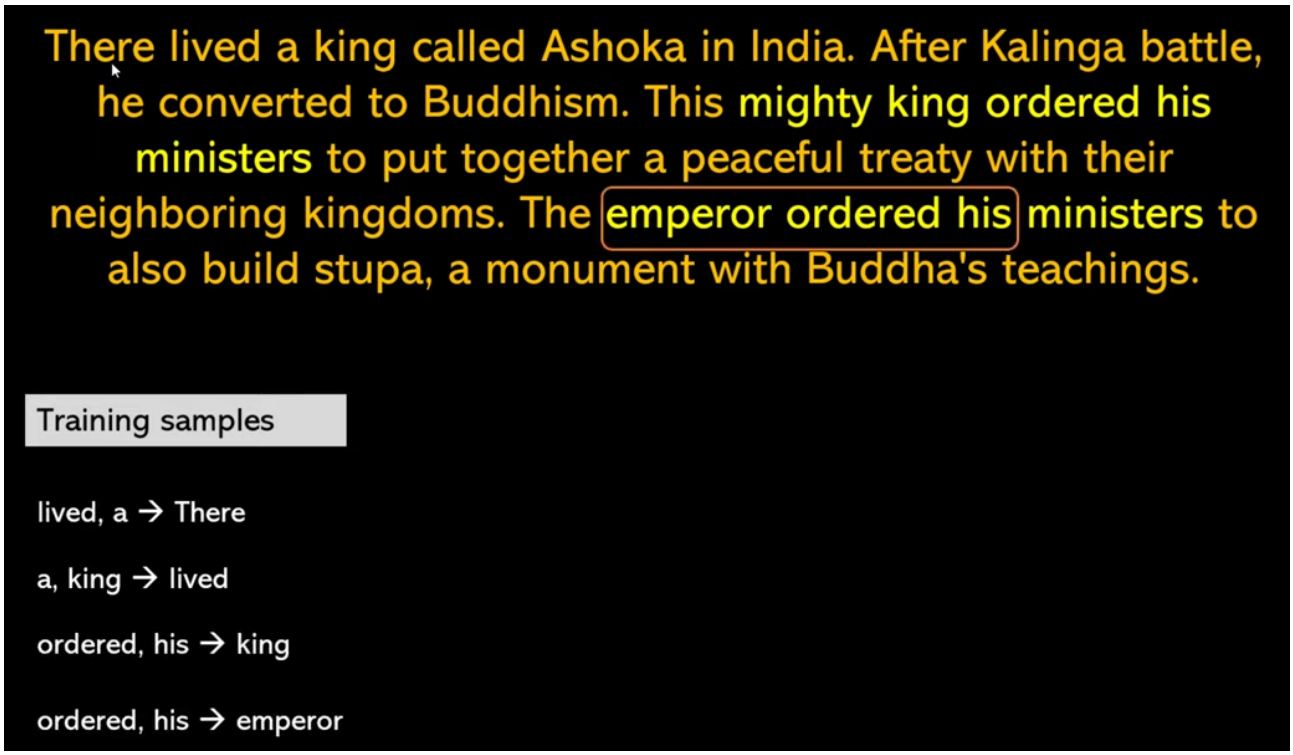
The basic intuition behind the word2vec technique is, “We can get useful information about a word by observing its context/neighbors.” Word2vec represents words in vector space representation. Words are represented in the form of vectors and placement is done in such a way that similar meaning words appear together and dissimilar words are located far away. The features are not hand-coded rather they are found using a neural-net.

The Bag Of Words is the approach which was used word embeddings. It used the concept where words are represented in the form of encoded vectors. It is a sparse vector representation where the dimension is equal to the size of vocabulary. If the word occurs in the dictionary, it is counted, else not. There are mainly tow architectures in Word2Vec which are, CBOW (Contiguous Bag Of Words) and SkipGram. In CBOW, the current word is predicted using the window of surrounding context windows. For example, if $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$ are given words or context, this model will provide w_i.

Skip-Gram performs opposite of CBOW which implies that it predicts the given sequence or context from the word. You can reverse the example to understand it. If w_i is given, this will predict the context or $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$.



THE TRAINING PROCESS



In the example given above, three words are grouped. The first word is said to be y and the trailing two words are said to be x. The x is fed to the neural network to predict the correct sentence. The cost is computed and back-propagated.

THE GloVe MODEL

The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics. For large corpora, this pass can be computationally expensive, but it is a one-time up-front cost. Subsequent training iterations are much faster because the number of non-zero matrix entries is typically much smaller than the total number of words in the corpus.

Consider, we have a set of sentences as, “Ari loves birds” and “Ari loves reptiles”. We can create a co-occurence matrix with window of value 1 as shown below:

	Ari	Loves	Birds	Reptiles
Ari	0	2	0	0
Loves	2	0	1	1
Birds	0	1	0	0
Reptiles	0	1	0	0

The table can be called as X_{ij} where i and j represents the horizontal and vertical records.

It is obvious that, Ari co-occurs with the word Loves more times and the vice versa. Loves co-occurs with the word Reptiles and Birds one time and the vice versa. If we plot probabilities,

P(k/Ari) =>

HIGH if k=Loves

P(k/Loves) =>

HIGH if k=Birds / Reptiles

P(k/Reptiles) =>

HIGH if k=Loves

and we can list out all the probabilities----possibilities like the above given. Mean square error function is used as the cost function here.

THE SENTIMENT CLASSIFICATION

The sentiment classification is a use-case where for a text (majorly tweets) given as input, the output is the sentiment (star rating).

Excellent

Good

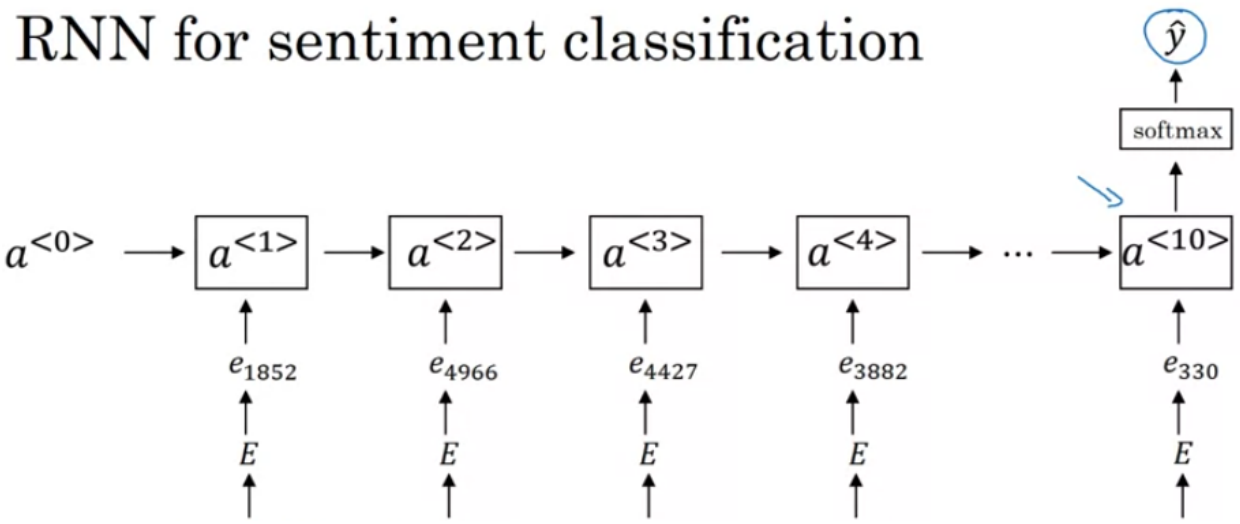
Average

Poor

Avoid

It is done by using normal embeddings.

If the text to feed is, “I am good”, the equivalent embedding is found for each words, averaged and given to a softmax classifier. It will give us the number of stars to be activated. Okay! Now, if the text is, “This seems so good, really awesome, wonderful and in my dream”. The text is seemed to be full of positive words but that positiveness is no more when we realize that the guy is mocking us. But, due to the repeated occurrence of the positive words, there is a chance for a good review rate although it isn’t. To overcome this, indeed we need to train a model.

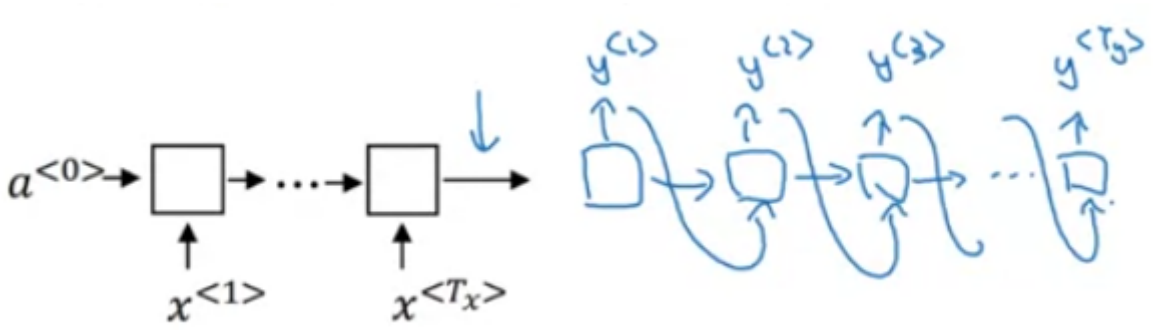


SEQ2SEQ MODELS

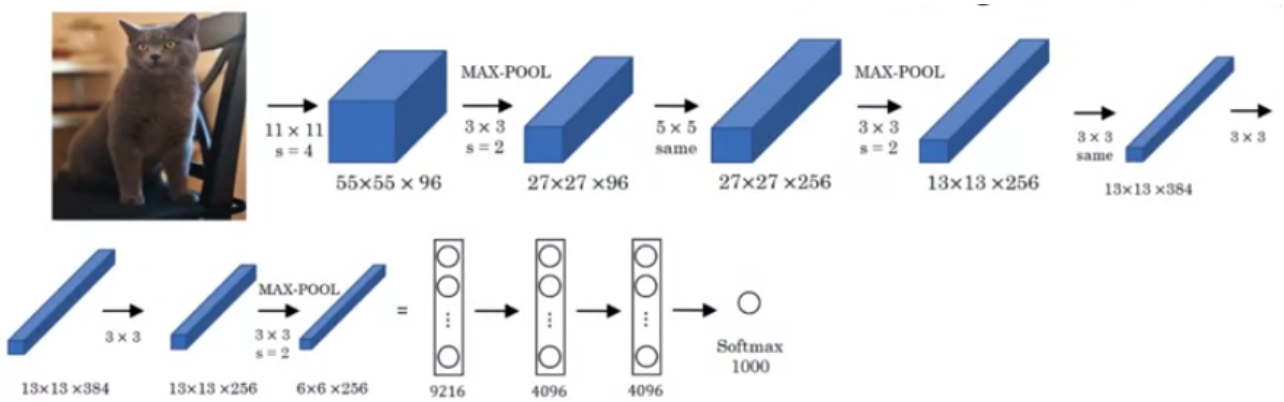
If both our input and the output data is a sequence, we need sequence to sequence models that can handle it. For example, if we have to deal with the problem of language translation, on having a text in Malayalam like, "അരി പഠിക്കുന്നു" we have to translate it into Tamizh as, "அரி படிக்கிறான்".

$x^{<1>}$ $x^{<2>}$ → $y^{<1>}$ $y^{<2>}$

അരി പഠിക്കുന്നു → அரி படிக்கிறான்



The model uses an Encoder Network on Malayalam sentence and a Decoder Network on that encoding to convert it into Tamizh. The network works very well. It is applicable for Image Captioning too. The pre-trained AlexNet can be used as an Encoder for the image and a sequence model to generate the caption (one word at a time).

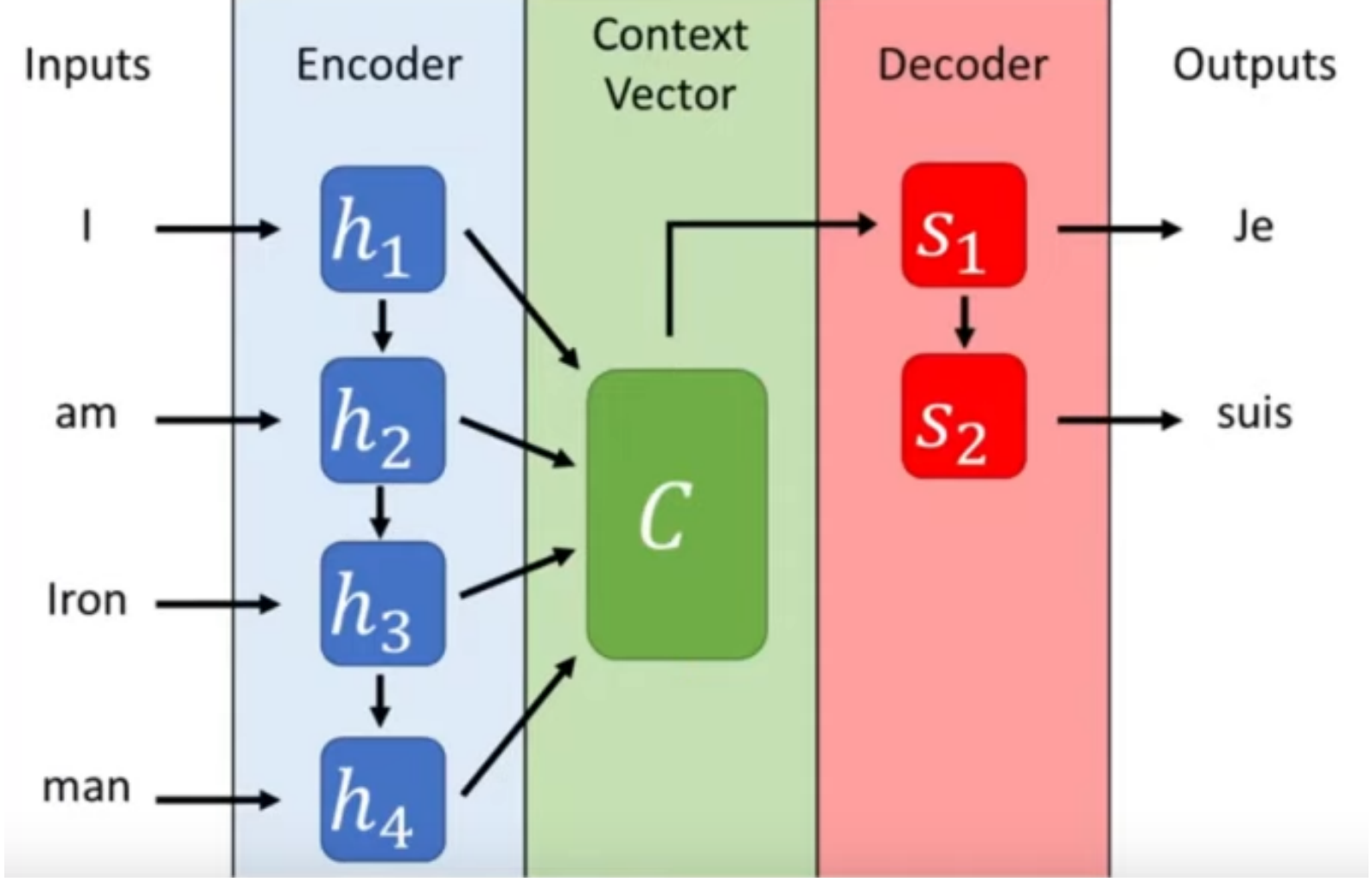


THE MOST LIKELY SENTENCE

It is quite important to pick the most likely sentence from the possibilities generated by a Encoder-Decoder network the so-called Machine Translation Network. So, we need a Language model that predicts the most likely sentence given the input sentence. It is a conditional language model. What we are talking about so far is,

$$P(y^{<1>},y^{<2>}..,y^{<len(x)-1>}/x)$$

On having "അരി പഠിക്കുന്നു ", we may get, அரி படிக்கிறான், அரிசி படிக்கிறான் and some other possibilities.



Besides, we have a meaning of “to know” for the word, "അറി" in Malayalam. So, it is quite important to have a conditional language model which picks the most likely sentence out of the possibilities. In our words, argmax(P(y<1>,y<2>..,y<len(x)-1>/x)) is what we have to find. The most common method

followed for this is the so-called Beam Search. There is another mechanism called, Greedy Search which picks one word and at a step and find the probability of having that at first and based on the results , it finds out the next word to place-right. It may be expensive.

THE BEAM SEARCH

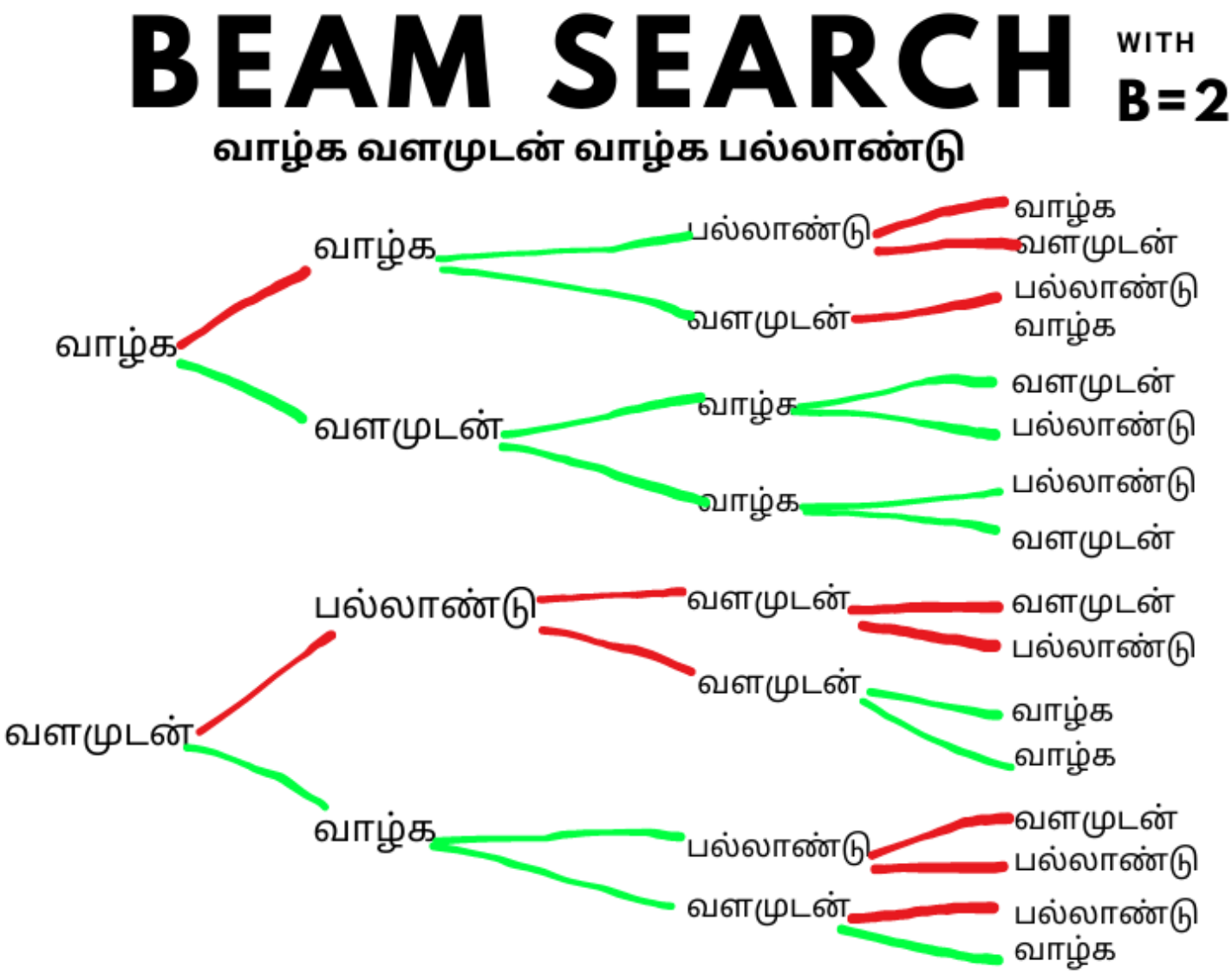
In Beam search, instead of picking just one word at a time, n number of possible words are picked. The number n is valued by B which is the beam width in Beam Search. If B==2, then initially two words will be chosen. For each word, furtherly two words will be chosen to proceed.

If the sentence predicted is, வாழ்க வளமுடன் வாழ்க பல்லாண்டு, on applying the beam search, it would be like the image given below. Here, red-lines denote the words with low probabilities and green with H which is found with a neural net.

Suppose, if we have found the sentence upto, வாழ்க வளமுடன் வாழ்க, and the word we take as a next word is பல்லாண்டு, then, the probability expression is,

P(பல்லாண்டு/வாழ்க வளமுடன் வாழ்க பல்லாண்டு, வாழ்க, வளமுடன், வாழ்க)

It is $P(y^{<t>}/x,y^{<1>},y^{<2>},..,y^{<t-1>})$. It is for one single sentence



generated by the Beam Search. But, we know that it will generate more sentences like this with different probabilities.

Say,

வாழ்க வளமுடன் வாழ்க பல்லாண்டு 0.98

வாழ்க வாழ்க வளமுடன் பல்லாண்டு 0.07

வளமுடன் பல்லாண்டு வாழ்க வாழ்க 0.05

வளமுடன் வாழ்க பல்லாண்டு வாழ்க 0.09

வாழ்க பல்லாண்டு வளமுடன் வாழ்க 0.32

வாழ்க பல்லாண்டு வாழ்க வளமுடன் 0.72

What we need is, the one with HIGH Probability. i.e., [வாழ்க வளமுடன் வாழ்க பல்லாண்டு, 0.98]

In our language,

$$\arg\max_{y} \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

Some refinements are required here. The values can be of tiny extent. Multiplying the probabilities can lead to a very tiny extent of values which maybe so hard to process. So, what we are about to here is, taking log!

$$\arg\max_{y} \sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

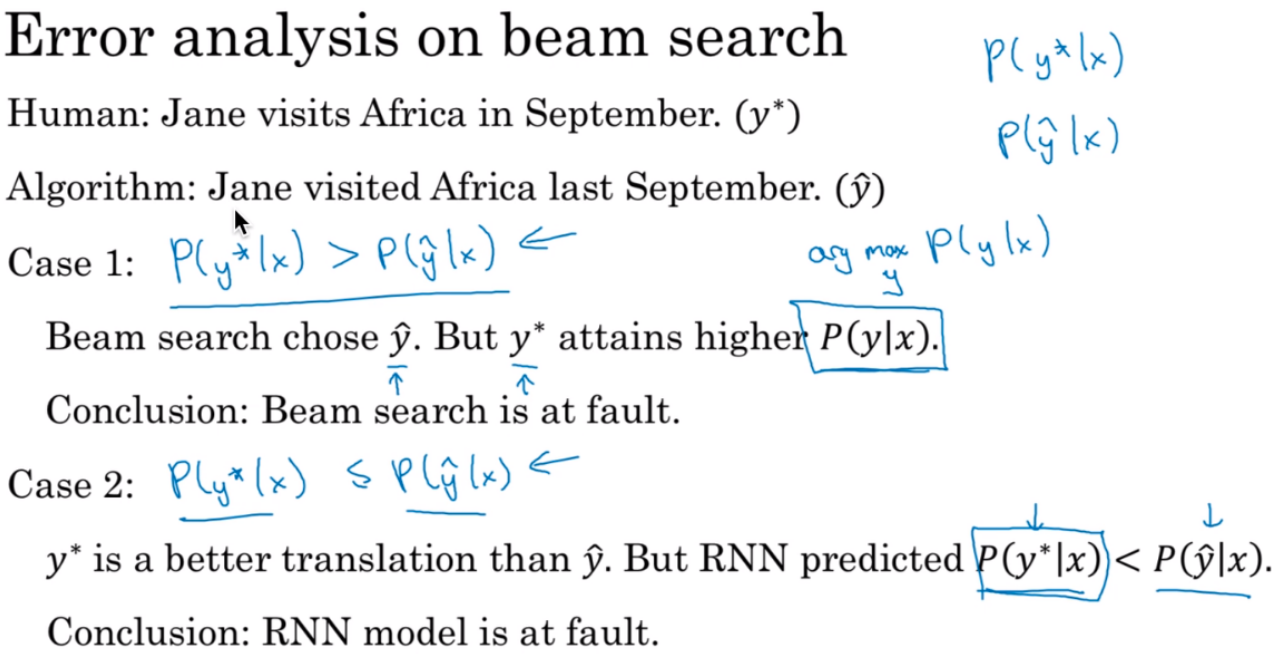
Now, the extents will be the same even though the value changes. It can be normalized by dividing by the length of words (T_y). It is used as (T_y^a).

If Beam Width, B is larger, more possibilities and better result can be obtained with slow computation. If it is

small, the result will be not desirable.

ERROR ANALYSIS

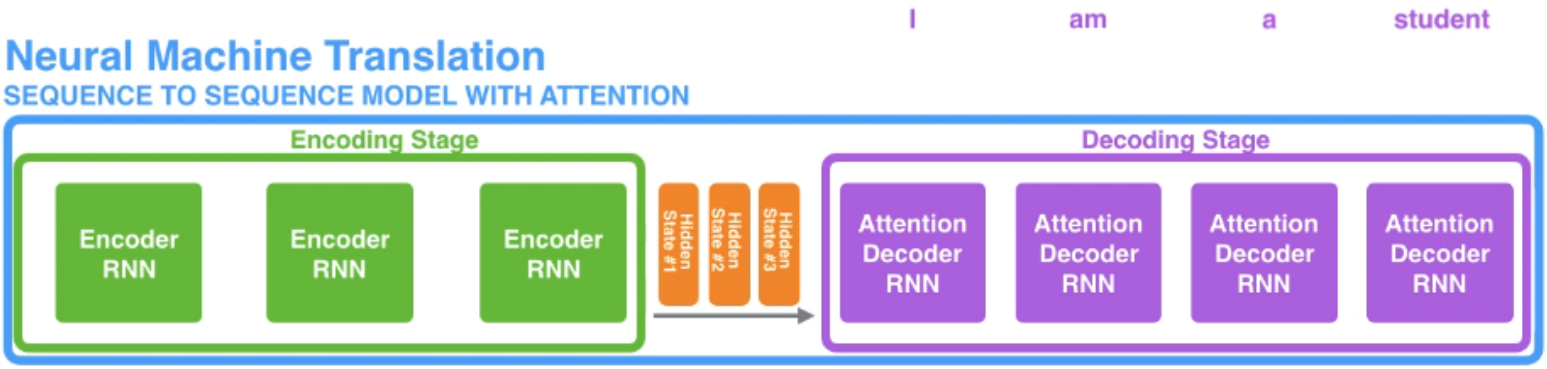
In Beam search, if a fault occurs, it is not obvious by whom the fault occurred whether because of the algorithm or of the RNN incorporated in it. To analyze it, here we go.



ATTENTION IS ALL WE NEED

If we feed the neural network a long sentence, it is quite difficult for it to memorize it and give the intended translated sentence. To achieve such a thing, we need the so called attention mechanism. How do we the humans translate long sentence ? We won’t read the entire sentence. We’ll look at the important words or a portion of the given sentence. Based on that, we do translation. It is a very influential idea. While processing long sentences, the context vector has to hold more contents.

An attention model differs from a classic sequence-to-sequence model in two main ways : First, the encoder passes a lot more data to the decoder. Instead of passing the last hidden state of the encoding stage, the encoder passes *all* the hidden states to the decoder.



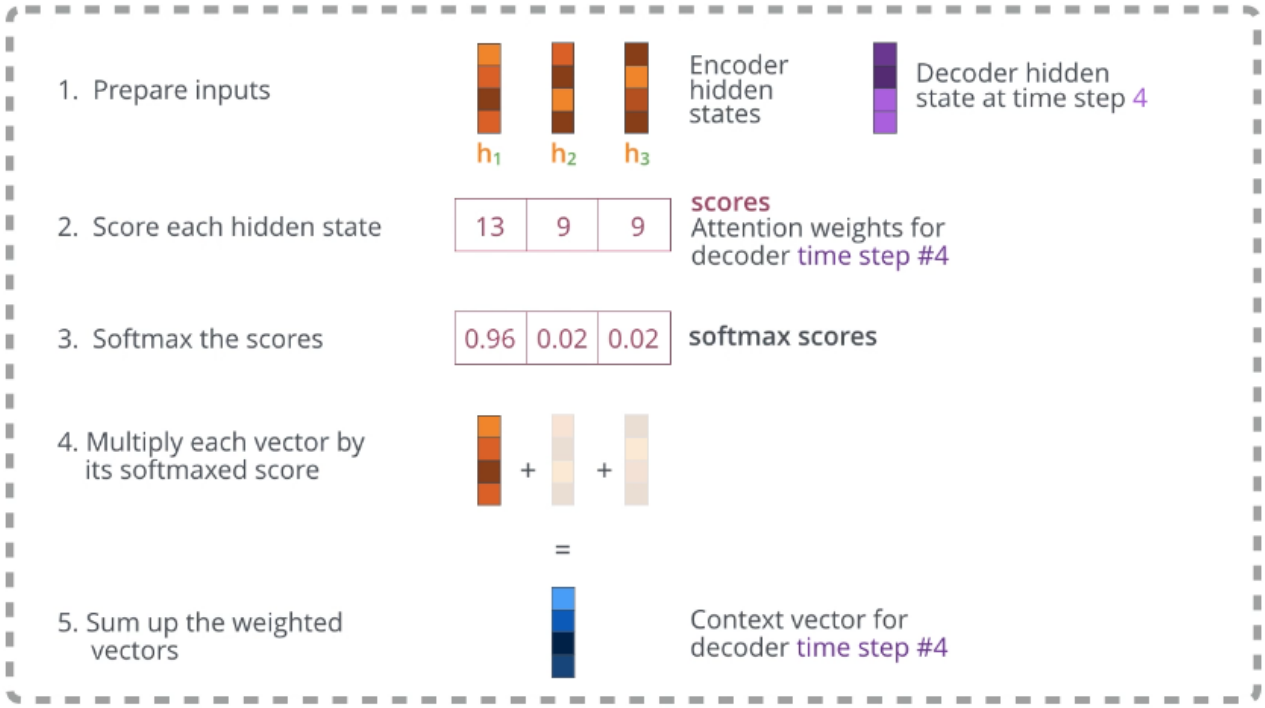
Second, an attention decoder does an extra step before producing its output. In order to focus on the parts of the input that are relevant to

this decoding time step, the decoder does the following :

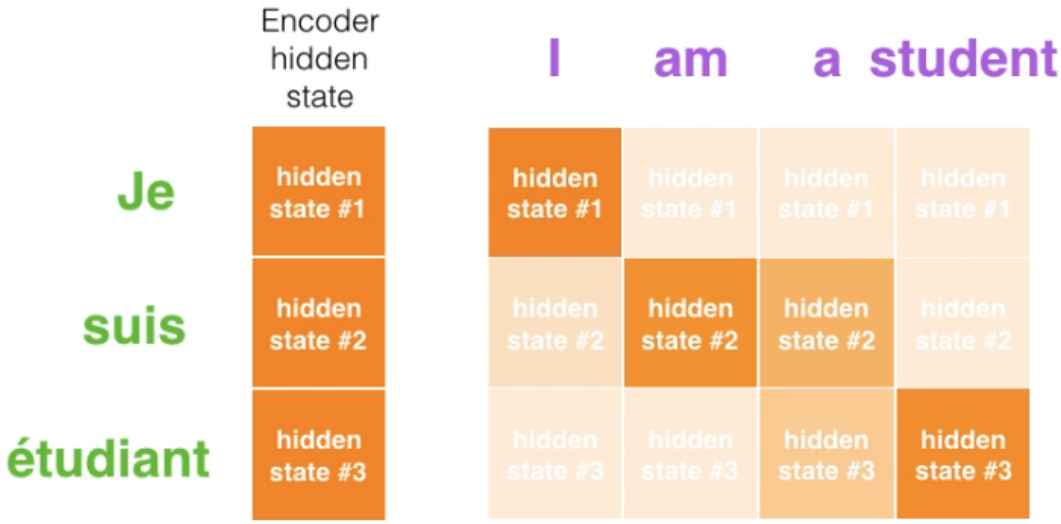
@ Look at the set of encoder hidden states it received – each encoder hidden state is most associated with a certain word in the input sentence

@ Give each hidden state a score

@ Multiply each hidden state by its softmaxed score



For each inputs, different parts are paid attention.



TRANSFORMERS ARE TO BE DISCUSSED IN THE UPCOMING, “TRANSFORMERS” BOOK.

MERCI BEAUCOUP