

Natural Language Processing (NLP)

Objective :-

- Understand the Philosophical foundation of NLP
- Understand sequence models
- Understand the application of NLP
- Understand the implementation of NLP with TensorFlow

Natural Language Processing and It's philosophy

Sequence Models

- 1) What does it mean by a sequence ?
- 2) What does it mean by Recurrent Neural Networks ?
- 3) Why do we choose RNN for sequence models ?

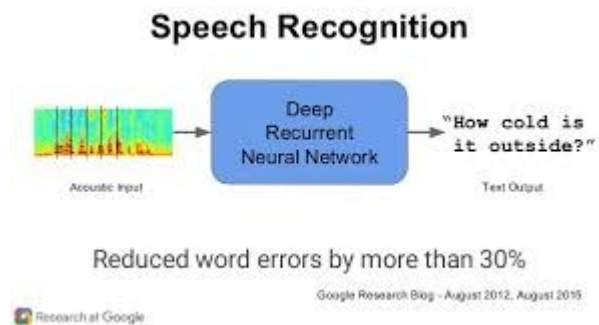
There are some problems that are in a sequence form , existing one after the other over a period of time.

Sequential data means whenever the points in the dataset are dependent on the other points in the dataset the data is said to be in a sequential form.

Some of the problems of sequence model

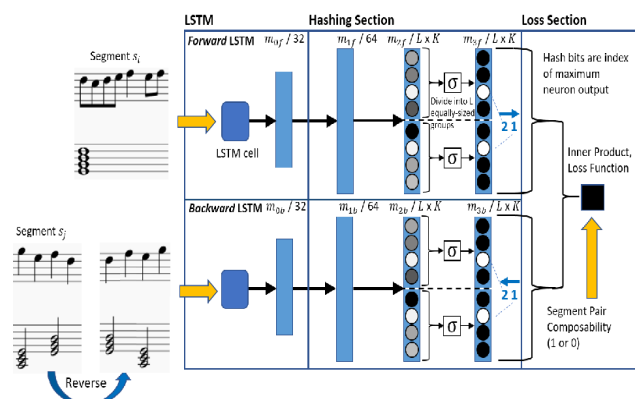
- 1) **Speech Recognition** :- Given the input x which is the speech and then translated in to a word y as an output. The speech is a sequence of audio clip plays overtime. In these case both the input and the output are a sequence

data , the input is a sequence of input overtime and the output y is a sequence of words.



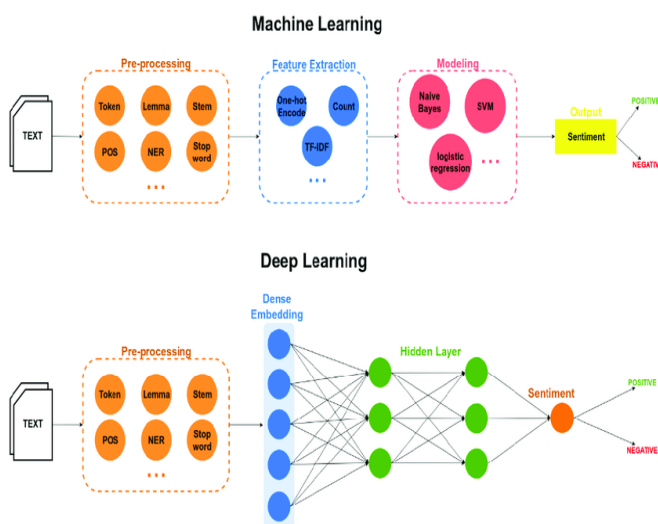
How does a speech recognition works ?

- 2) **Music Generation** :- This is another problem of a sequence model, the input x is not a sequence model , it is an empty set(\emptyset) or a list of notes and the output is a generated music.



- 3) **Sentiment Classification** :- The word sentiment means view or opinion. sentiment analysis is a powerful text analysis tool that automatically mines unstructured data (social media , emails , customer

feedback and more) for opinion and emotion and then classify it with one of the labels that we provide. or example, you could analyze a tweet to determine whether it is positive or negative, or analyze an email to determine whether it is happy, frustrated, or sad. or if you have a hotel owner , the customers will give you a text based feedback then the algorithm will classify it in to rates , How many stars these review would be ?



4) **DNA Sequence analysis** :- This is also another problem of a sequence problem. The input X is a DNA sequence and the output y , which says , which part of these DNA sequence corresponds to a protein.

Example:-Input-X:-

AGCCCCTGTGAGGAACTAG

Output:-y

AG**CCCCTGTGAGGAACT**AG

:-

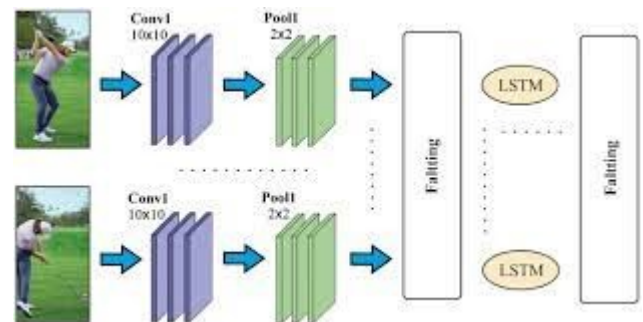
- DNA is represented using alphabet A C G. In the output the red part of the DNA refers to protein.

5) **Machine Translation** :- This is another sequence problem where you input a sentence and then expect translation in another language.

Example:- French Input :- Voulez vous chanter avec moi

English output :- Do you want to sing with me ?

6) **Video Activity Recognition** :- We will have a sequence of video frames as input and then the output y , is to recognize the activity.



7) **Name Entity Recognition** :- We will give a sequence of sentence as input X and ask the sentence the name of the people in that sentence.

Example:- Input X :- Yesterday , Harry potter met Hermione Granger

Output y :- Yesterday **Harry potter** met **Hermione Granger**

Why do we need name entity recognition ? because sometimes we may need to index(to index means to

organize them in a certain schema) all of the all the people mentioned in the last 24 hours of news, articles. but not only people's name but also company names , location names , country names , currency names(Birr , dollar , pound).

- So all of these supervised machine learning problems were there is a mapping between an input x and a label y . so in these problems may be both of the input x and the output y are sequences or one of them might be sequences and one might not. sometimes the length of the input sequence and the output sequence might be equal or sometimes not.

Notations used to build sequence models

X : Harry potter and Hermione Granger invented new spell
y:- 1 1 0 1 1
0 0 0

We have one output per input word and the target output to the desire y tells you for each of the input words is the part of the person's name.

X:- (Harry potter) and (Hermione Granger) invented new spell
y:- 1 1 0 1
1 0 0 0

These is another sophisticated way of representation which tells you not only

is a word is a part of a person's name but tells you where the start and the end of people's names in the sentence.

So the input is a sequence of nine words , so we are going to have nine features to represent this nine words and index in to the position and the sequence as a superscript for each of the features.

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|--------------------|-----------|
| $x^{<1>}$ | $x^{<2>}$ | $x^{<3>}$ | $x^{<4>}$ | $x^{<5>}$ | $x^{<6>}$ | $x^{<7>}$ | $x^{<8>}$ |
| X:Harry | potter | and | Hermione | Granger | invented | new | spell |
| y:- 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $y^{<1>}$ | $y^{<2>}$ | $y^{<3>}$ | $y^{<4>}$ | $y^{<5>}$ | | ... $y^{<7>}$ | $y^{<8>}$ |

small t represents the time step. **What does it mean by time steps ?** In Natural Language Processing (NLP), a time step may be associated with a character, a word, or a sentence, depending on the setup.

To know the length of the words in the input sequence $T_x = 8$, to know the length of the words in the output sequence we will represented as $T_y = 8$. In these case(in the name entity recognition problem) both the inputs and the output length is equal.

In the notation that we saw $x^{(i)}$ - to denote the i^{th} training example so, to refer the t^{th} element or the t^{th} element of the sequence of the training example i , will use this notation $x^{(i)<t>}$.

and if T_x is the length of the sequence then $T_x^{(i)}$ is the length of the sequence in that training example.

Similarly $y^{(i)<t>}$ means the t^{th} element of the sequence of the output label for the i^{th} sample. and if T_y is the length of the sequence then $T_y^{(i)}$ is the length of the sequence in that i^{th} sample.

How to represent each words in the sequence (How to convert the problem in a Digitalized Form)

Representing the words

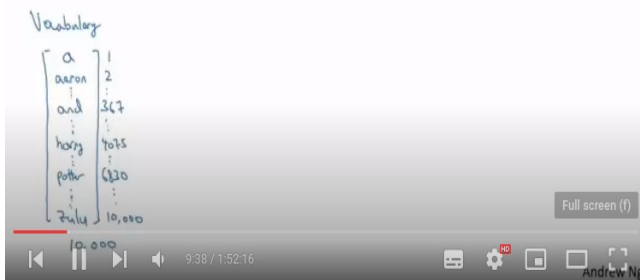
Harry Potter and Hermione Granger invented new spell.
 $x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<6>}$ $x^{<7>}$ $x^{<8>}$

We are now working on a Natural Language Processing and one thing that we need to decide is how to represent individual words in the sequence like how do we represent the word harry and what should $x^{<1>}$ really be.

So to represent (changing a certain problem in to a number) the words in the sentence the first thing you do is come up with a vocabulary or sometimes dictionary that means making a list of the words that you will use in a representation.

Representing words

x: Harry Potter and Hermione Granger invented a new spell.
 $x^{<1>}$ $x^{<2>}$ $x^{<3>}$... $x^{<9>}$

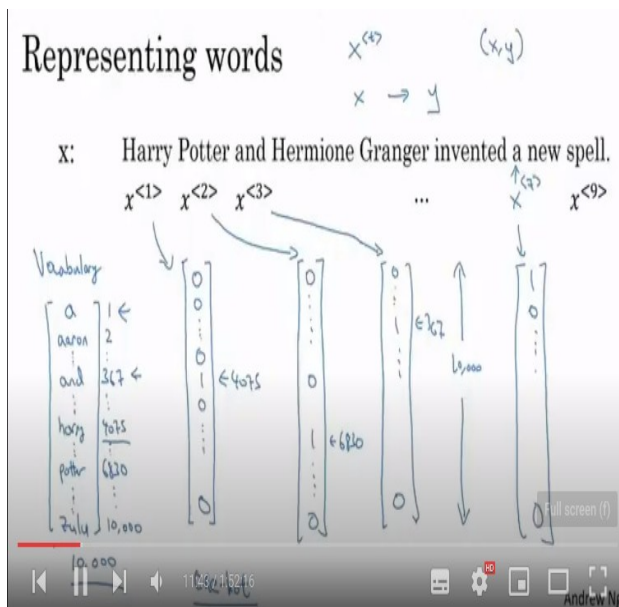


The dictionary starts with “a” , and the numbers in the left shows the indexes or the position of the words in the vocabulary.

In this example we are using a dictionary with the size of 10 , 000 words this is quite small by the modern NLP applications. For commercial applications dictionary size of 30 - 50 thousand words are more common and 100 thousand is not uncommon and some the large internet companies will use a dictionary size of a million words may be bigger than that.

So After deciding the size of our dictionary , one way to build these vocabulary would be to look through your training set and find the top 10,000 occurring words to look through some of the online dictionaries that tells you what are the most common 10,000 words in the english.

So in order to represent the words in our training corpus , which means in order to change our problem in to numbers in order to be computed by our algorithm , one way to do is through one hot encoding.



again , Recurrence is a new occurrence of something that happened or appeared before.

Question:- The output of the previous network will feed in to the input of the next layer during standard neural networks too, so why are we calling them as a recurrent neural networks ?

Answer:- There is also another definition of “Recurrent” which is a running or turning back in a direction opposite to a former course , these mean the Architecture of RNN’S is Bidirectional , but on the standard neural networks we only can go in the forward direction (Back prop is just a way of updating the weights not because it’s Bidirectional)

We are doing these using supervised machine learning by giving a pair of (x,y)

What if we encounter a word that is not in our vocabulary , well the answer is we create a new token or a new fake word unknown word to represent words that are not in our vocabulary.

Recurrent Neural Networks Model

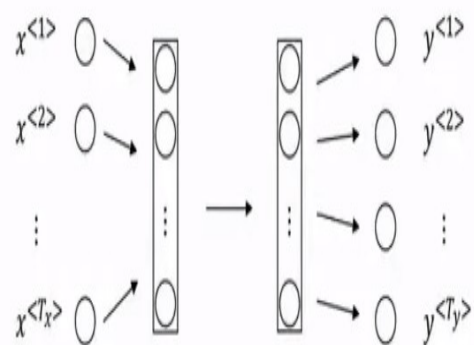
- Objective :-

- ✓ What does the name Recurrent refers to ?
- ✓ Why do we choose these Model for sequential problems
- ✓ What are the problems with the standard neural networks or convolutional networks

What does the name Recurrent refers to in Recurrent Neural Network

Recurrent means the previous output is fed in to the model as a part of input. Reoccurring or appearing

Why not a standard network?



The output of the values is zero/one that tells you whether each word is a part of the person’s name.

what is the problem with the standard neural network ?

1) Input and outputs can be different length in different training examples , so there is no defined input and output shape. So it's not as if every single example has same input length T_x or same output length T_y .

But one solution for this is to set a maximum length and work in according to that but this is still doesn't seem a good solution.

2) Doesn't share features learned across different positions of the text. suppose we have an input word $x^{<1>}$, these input text $x^{<1>}$ doesn't learn features from the top or the bottom features. **why do we need to learn features from other positions of the text ?**

Example :- 1) He said , "Teddy Roosevelt was a great president"

2) He said , "Teddy bears are on the sale"

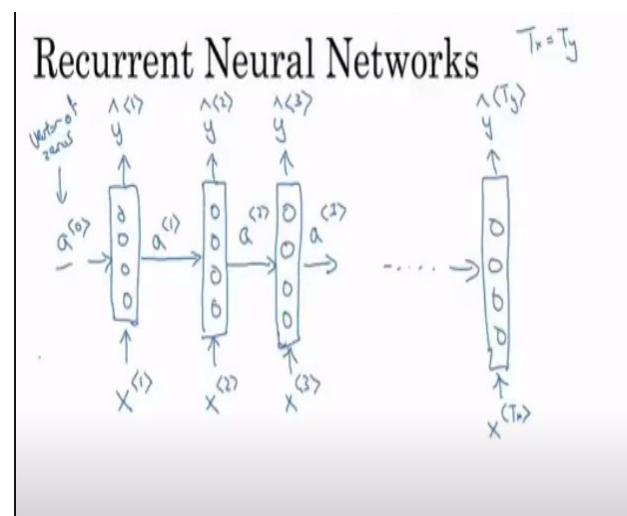
In the first example , if $x^{<3>}$ can't learn features from $x^{<4>}$ it doesn't be sure that teddy is a name or not because in the second example we can see that Teddy is not a person's name.

3) The number of parameters will be high (this is basically the problem of a standard neural network) , previously we said that each of these is a 10,000 dimensional one hot vector , so this is just a very large input layer , and if the total input size of the maximum

number of words times 10,000 and the weight matrix of this first layer would end up having an enormous number of parameters.

so recurrent neural networks solve all of these problems. How recurrent neural networks solve these problems and how it suits for a sequential problems.

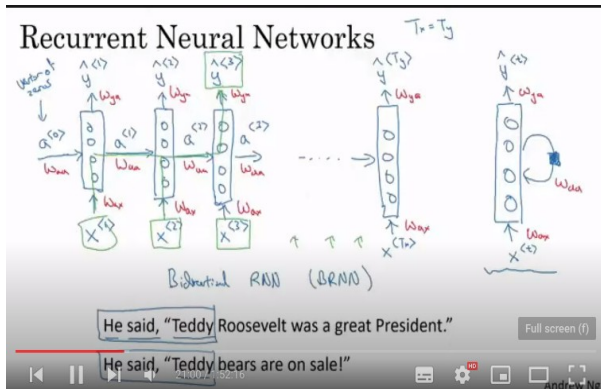
Recurrent Neural Networks :- so if we are reading a sentence from left to right , the first word you read say $X^{<1>}$ and take this first word and feed it to the neural network layer and make a prediction that it's part of a person's name or not.



What a neural network does is when it then goes on to read the second word in the sentence $x^{<2>}$, instead of just predicting $y^{<2>}$ using $x^{<2>}$, it also gets to input some information from the earlier network. In our example $T_x = T_y$, the length sequence is equal to the output sequence , this is no always

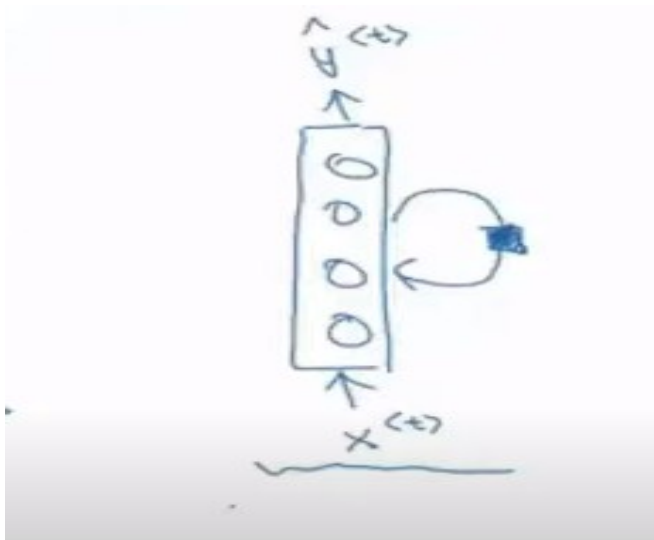
happened so that the Architecture will also change.

And to kick off the whole thing we'll also have some made up activation with a time step of zero, this is usually



a vector of zeros and some researchers initialize $a^{<0>}$ randomly. **why do we need to kick off ?**

On some books or research papers , we will find a neural network architecture like this



It's difficult to interpret this but it's the same as the one in the above (we just need to unroll it)

The parameters which governs the connection from $x^{<t>}$ to the hidden layer is gonna be denoted as w_{ax} , We are using the same parameter for all of the time steps. The horizontal connections will be governed by some set of parameters w_{aa} (the same parameters for all of the time steps). The output predictions will be governed by some set of parameters called w_{ya} (same parameters for all of the time steps). **why are we using the same parameters in each of the time steps ?**

Note in the Recurrent Neural Network ,when making a prediction for $y^{<3>}$, it gets information not only from the $x^{<3>}$ but also from $x^{<2>}$ and $x^{<1>}$ because the information from $x^{<1>}$ pass through on the way to help the prediction with $y^{<3>}$

Now the weakness of this RNN is that , it only uses the information that is earlier in the sequence to make prediction , in particular when predicting $y^{<3>}$, it doesn't use information from the words $x^{<4>}$, $x^{<5>}$, $x^{<6>}$, and so on.

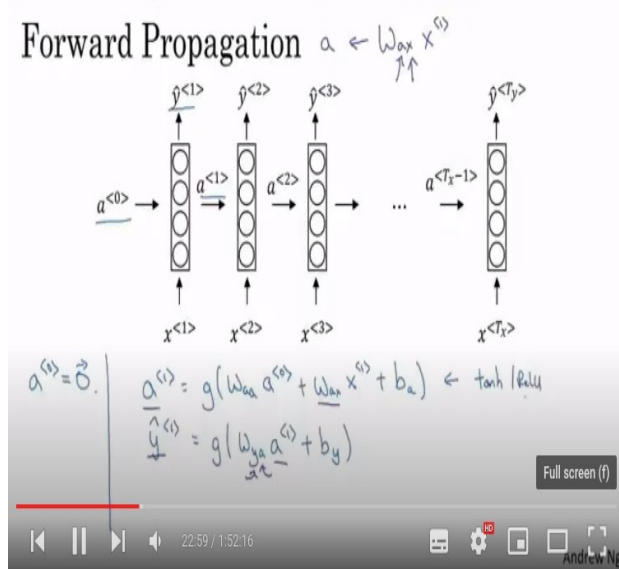
Example:- He said , “Teddy Roosevelt was a great president ”.

- In order to decide whether not the word teddy is a part of a person's name , it'd be really useful to know not just information from the first two words (He said) but to know

information from the later words in the sentence as well.

Example :- He said , “Teddy bears are on sale” - sentences like these could also happen , that given just the first three words , is not possible to know for sure whether the word teddy is a part of a person’s name or not. In the first example it is but on the second example it’s not but you can’t tell the difference if you look only at the first three words. We will address these problem using Bidirectional Recurrent Neural Networks (BRNN) but this is a uni-directional neural network architecture.

Forward Propagation In Time Of RNN



What is the reason for the name “Forward propagation in time ?” because Our RNN will act at each and every time steps like at $t = 1$ will preprocess one word and at $t=2$ will preprocess another word.

What does the $a^{<1>}$, $a^{<2>}$, $a^{<3>}$... $a^{<t>}$ refers to ? are we computing the total or sum of each neuron in that RNN’s layer ?

$$a^{<t>} = w_{aa} a^{<t-1>} + w_{ax} x^{<t>} + b_a$$

This is how we are computing the $a^{<t>}$ for each time step. **so what is the application of each of the neuron’s on each block of the RNN , if they are not useful for computing $a^{<1>}$?** Look it consciously and compare it with the way of computing the standard Neural Networks.

$$Z^{<1>} = (w_1 x_1 + w_2 x_2 + w_3 x_3) + b$$

$$a^{<1>} = g(z^{<1>})$$

But now on RNN’s case :-

$$a^{<t>} = w_{aa} a^{<t-1>} + w_{ax} x^{<t>} + b_a$$

We are not computing only $x^{<t>}$ but also $a^{<t-1>}$ this means in Recurrent Neural Network when making a prediction for $y^{<t>}$, it gets information not only from the $x^{<t>}$ but also from $x^{<t-1>}$ and $x^{<t-2>}$ and so on because the information pass through on the way to help the prediction with $y^{<t>}$ But on both direction.

Why do we need weights for every connection ? for the feature , for the activation and for the output ?

Why is it the weights for the entire RNN is the same ?

The Notation w_{ax} means , the second index(x) means that this w_{ax} going to be multiplied by some x like quantity and the first index(a) means that this is used to compute a like quantity($a^{<t>}$).

Similarly we will notice that w_{ya} is multiplied by 'a' like quantity to compute y type quantity ($y^{<t>}$)

The activation function used in to compute the activations will often be a tanh in the choice of RNN and sometimes ReLu are also used , but tanh is the common choice. Activation mean the activation for each neuron in each of the RNN blocks.

Depending on what the output y is , if it is binary classification problem or Multi label classification problem we will use sigmoid activation function or it could be a softmax if it is a multi class classification problem but generally the choice of the activation function would depend on what type of output y we have.

In our current problem which is named entity recognition our output is binary classification(The word is a person name or not) so we will use sigmoid activation function for each of our output.

More generally at times $<t>$:-

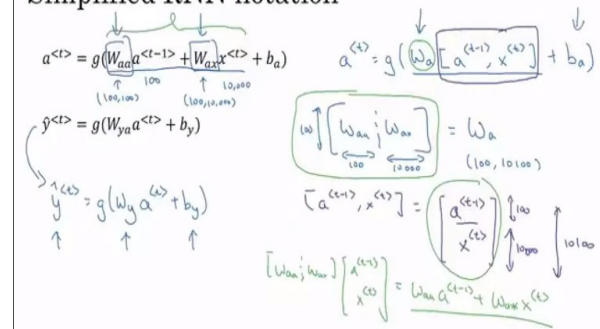
$$a^{<t>} = g(w_{aa} * a^{<t-1>} + w_{ax} * x^{<t>} + b_a)$$

$$y^{<t>} = g(w_{ya} * a^{<t>} + b_y)$$

In order to help us develop the more complex neural network , we can actually take the above notation and simplify it a little bit.

Simplified RNN notation

Simplified RNN notation



From the previous we have this equation:-

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

$$y^{<t>} = g(w_{ya} * a^{<t>} + b_y)$$

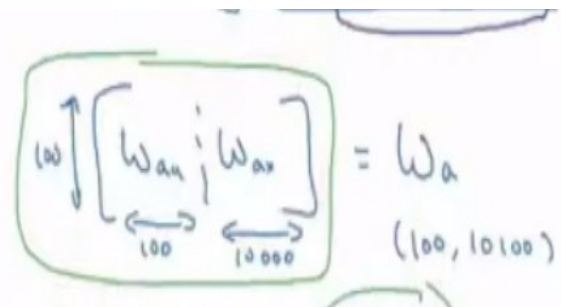
now the simplified version is

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

$$y^{<t>} = g(w_y a^{<t>} + b_y)$$

- So if we take the matrix w_{aa} and w_{ax} and put them side by side to be horizontally stacked $[w_{aa} | w_{ax}] = w_a$

For example :- if $a^{<t-1>}$ is a 100 dimensional vector and $x^{<t>}$ was a 10,000 dimensional one hot vector , so w_{aa} will be (100*100) dimensional matrix and w_{ax} will be (100 , 10,000) dimensional matrix , so stacking them these matrices together horizontally



Why do we make 10,000 in the horizontal and 100 in the vertical ?
 because of the rule (row , column).
 and now if we stack $a^{<t-1>}$ and $x^{<t>}$ vertically

$$\begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix}$$

Dimensions: 100, 10,000, 10,100

$$\begin{bmatrix} w_{aa} & w_{ax} \end{bmatrix} \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = w_{aa} a^{(t-1)} + w_{ax} x^{(t)}$$

Now by multiplying the horizontally stacked (w_a) weights by the vertically stacked ones we can have our original formula (not the simplified one).

so if they are equal , why do we need to simplify the notation , the advantage of the simplified notation is that , rather than carrying around two parameters (w_{aa} and w_{ax}), we can compress them in to one parameter matrix w_a and this will simplify our notation when we develop more complex models.

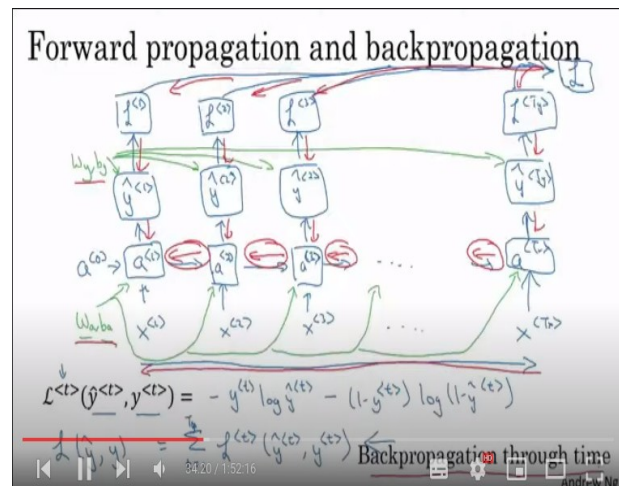
Similarly :-

$$y^{<t>} = g(w_y a^{<t>} + b_y)$$

$$y^{<t>} = g(w_y a^{<t>} + b_y)$$

The subscript w_y and b_y denotes the type of output quantity we are computing which is $y^{<t>}$. w_a and b_a denotes our output quantity we are computing which is $a^{<t>}$

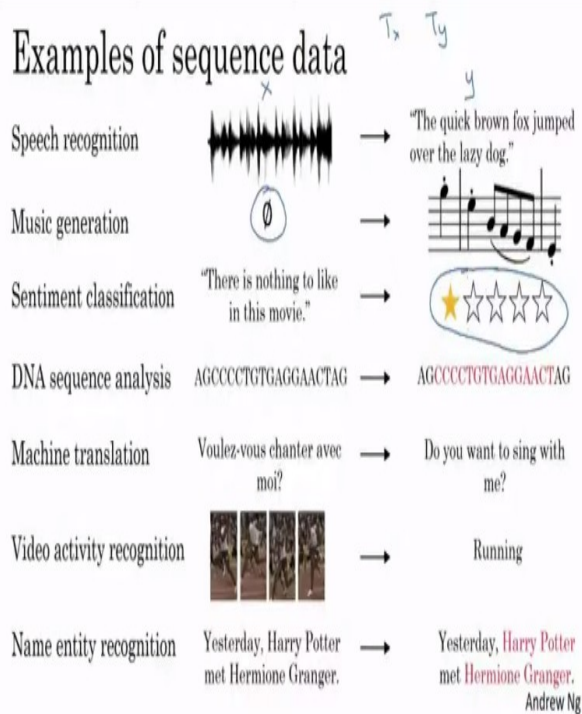
Back Propagation Through Time



Why is the name called Back propagation through time ? Because as we were going from the left to the right forwardly at each indices time step t , now we are going from the right to the left for each word(in each time step) to update the weights by back propagating. That is why we called it Back-Propagation thorough time.

Different Types of Recurrent Neural Networks

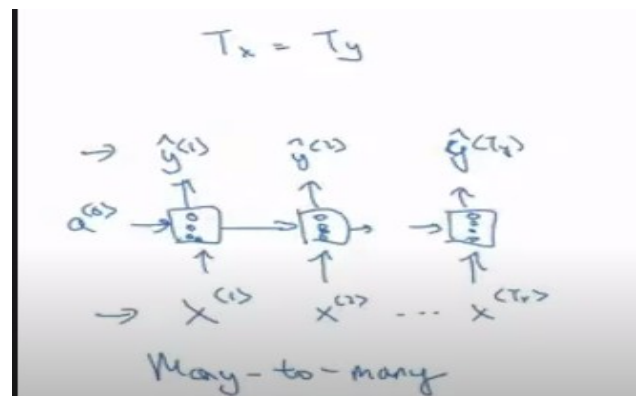
so far we've seen an RNN architecture where the number of inputs T_x is equal to the number of outputs T_y . It turns out that for other applications T_x and T_y may not always be the same.



In the second example Music Generation T_x can be length one or empty set or in example like sentiment classification the output y could be just integer from 1-5 where the input is a sequence.

In the name Entity recognition both the length of the input x and the output y are in equal size where there are some problems like machine translation where both of them are sequences but the length of the input and the output are not equal.

So we could modify the basic RNN

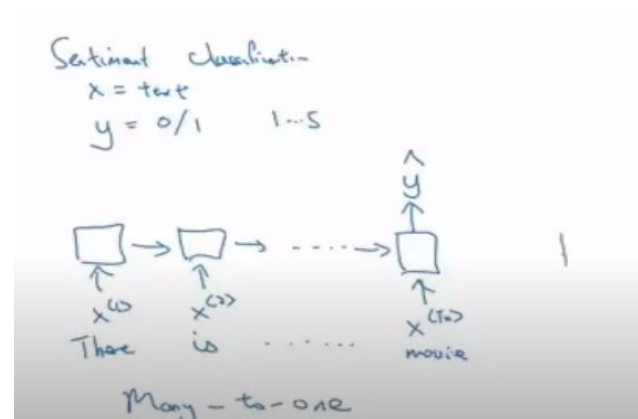


Architecture to address all of these problems.

Examples Of RNN Architecture :- The example we've seen so far worked as follows ($T_x = T_y$)

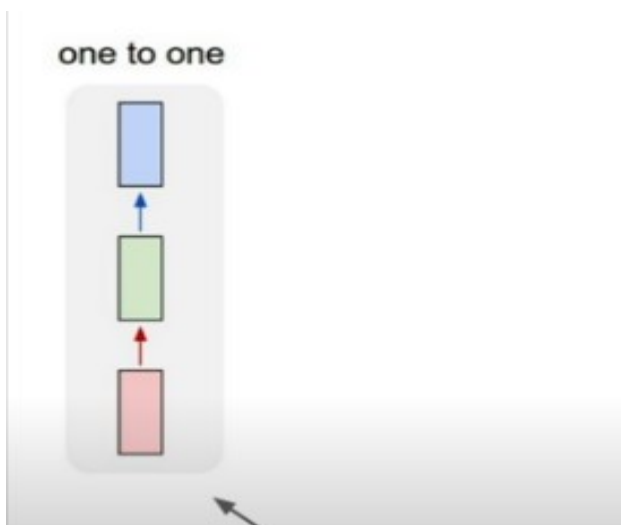
This is an architecture called many to many (The input sequence has many inputs and the output sequence has many outputs)

- Let's say we want to address a sentiment classification, Here x is a piece of text, it might be a movie review and y is a number from 1 to 5 or may be 0/1 (Either it is a positive review or a negative review), so for this kind of problem we will simplify our RNN Architecture like this



Rather than to have an output in every single time step, we can just have the RNN read the entire sentence and have an output y at the last time step y , this architecture called many to one, which is many input words and one output.

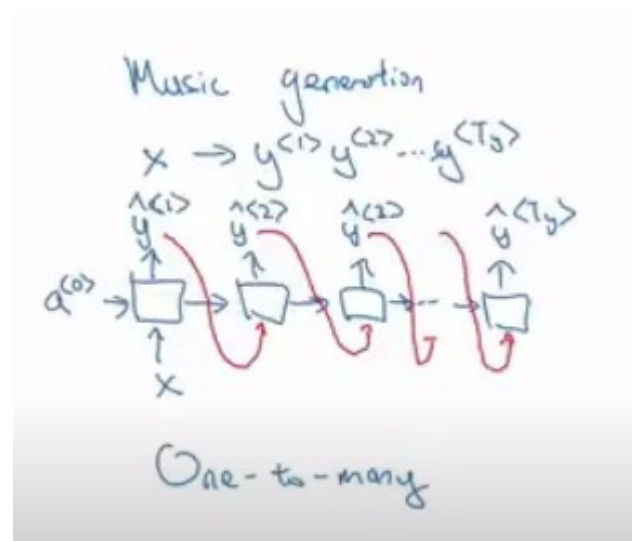
There is also an architecture called one to one but these is not actually different from the standard neural network(input x and just output y).



All of our network architectures have this flavor where we receive some input and that input is fixed size object(a fixed set of features which does not differ by different training examples) like an image or a vector and that input is fed in to some set of hidden layers and produces a single output like a set of classification scores over a set of categories.

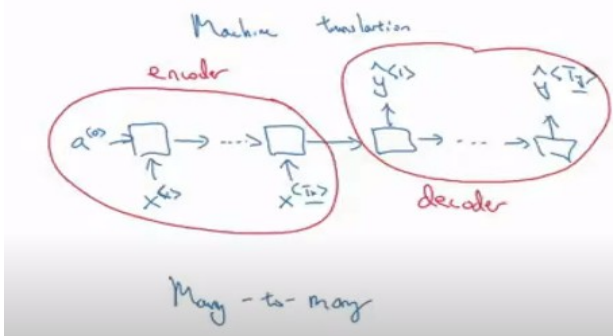
Let's say we want to address a music generation problem, Here x is may be an integer to tell the genre of the

music or empty set or a vector of zeros as input and y is a set of notes.



In these case the input is empty set or any integer which the Genre of the music or a vector of zeros.

It turns out that there is also more interesting example of many to many which is the length of the input and the output is different, so in the many to many example we saw for Name Entity Recognition, the input length and the output length have to be exactly the same, but for application like machine translation the number of words in the input sentence (say French sentence) is not equal to the output sentence(say english sentence).



The Architecture has two different distinct parts the Encoder(which takes input , the french sentence) and then there is a decoder (which having read the sentence outputs the translation in to a different language).

Language Model

Objective :-

- ➔ What is Language Model
- ➔ Why do i need to study a language model
- ➔ What are some application of language models
- ➔ How language model works
- ➔ How to implement language model

Language Modeling is a task of predicting what word comes next

Example :- The students opened their _____

More formally given a sequence of words $x^{<1>}$, $x^{<2>}$, $x^{<3>}$,, $x^{<t>}$, Then compute the probability distribution of the next word $x^{<t+1>}$, a language model is something that computes the probability distribution of the next word $x^{<t+1>}$. $p(x^{<t+1>}|x^{<t>}.....x^{<1>})$ where $x^{<t+1>}$ can be any word in the vocabulary $v = \{w_1 ,, w_{|v|}\}$

There is an alternative way of thinking about a language model

- we can think of a language model as a system that assigns probability to a piece of text.

For example :- If we have some text $x^{<1>}$, , $x^{<T>}$ then the probability of this text (according to the language model) is

$$P(x^{<1>} ,, x^{<T>}) = p(x^{<1>}) * p(x^{<2>} | x^{<1>}) * * p(x^{<t>} | x^{<t-1>} x^{<1>})$$

so we can think these as something equivalent , predicting the next word gives you a system that can give the probability of a given piece of text.

Example :- Let's say we are building a speech recognition and you hear the sentence "The apple and pear salad was delicious" so what did you hear ? did I say 1) The apple and pair salad or 2) The apple and pear salad

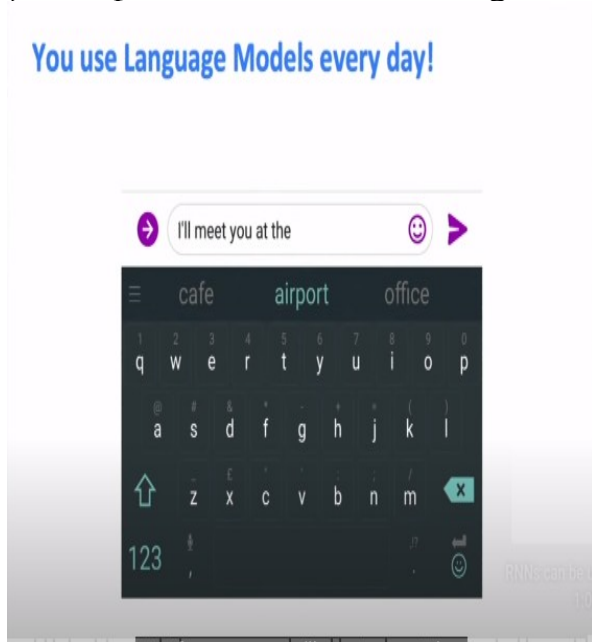
Humans probably think the second sentence is more likely and that's what a good speech recognition system would output even though these two sentences sound exactly the same. and the way the speech recognition system picks the second sentence is by using a language model which tells it what the probability of either of these sentence probabilities

$$p(\text{The apple and pair salad}) = 3.2 * 10^{-13}$$

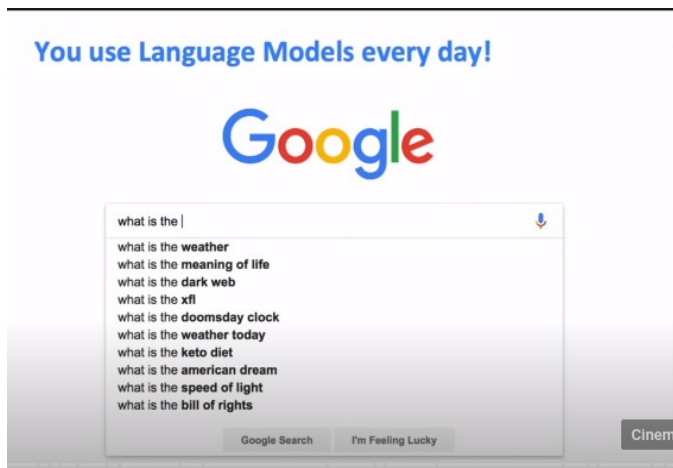
$$p(\text{The apple and pear salad}) = 5.7 * 10^{-10} , \text{The second sentence is much more likely over a factor of } 10^3.$$

We use language models everyday :-

1) Text prediction in our messages



2) When you search for something on the internet , for example Google , and we start typing a query and then google will try to complete your query for you.



How would you learn a language model

Before the deep learning era we were using N-grams and now we are using Recurrent Neural networks.

n-gram language model :-

The students opened their_____

Definition of n-gram:- A n-gram is a chunk of n-consecutive words

- Unigrams :- “the” , “students” , “opened” , “their”

- bigrams :- “the students” , “students opened” , “opened their”

- trigrams :- “the students opened” , “the students opened their”

- 4 grams :- “The students opened their”

what is the idea of n-gram language model ?

So the ideas of n-gram language model is that , in order to predict what word comes next , we are going to collect a bunch of statistics about **how frequent** d/t n-grams are from our training data and use those statistics to predict what next word may be likely.

So now how do we calculate the conditional probability in order to predict the next word.

Example:- Suppose we are learning a 3 gram model

➤ “as the proctor started the clock, the students opened their_____”

so because we are learning a three gram language model our simplifying assumption is that the next word depends only to the last three words.

$$p(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

We are now calculating the conditional probability of a certain word w from the vocabulary and this is calculated by counting the 'students opened their w ', from the training corpus divided up the entire count of the 'students opened their'.

For example :- Suppose that in our corpus :-

"students opened their " -- occurred 1000 times

"students opened their books"-- occurred 400 times

-> $p(\text{books}|\text{students opened their}) = 0.4$

"students opened their exam " -- occurred 100 times

-> $p(\text{exam}|\text{students opened their}) = 0.1$

Now the question is what about the context ?

Now the question is , was it a good idea for us to discard the proctor context ?

- if we look our actual example that we had "as the proctor started the clock the students opened their -----",

so do we think that books or exams is more likely given the actual context ? It's exams , exams is more likely because the proctor is heavily implies that it's an exam scenario , than more likely to be books , unless it's an open book exam.

So the problem that we are seeing here is that in the training corpus the fact that students were opening something means it's more likely to be books than exams because overall books are more common than exams. But if we know the context "the proctor and the clock " then it should be exams , so what we are highlighting here is the problem with our simplifying assumption , of we through too much context then we are not as good as predicting the words as we would be if we kept the context. **So how do we solve such kind of problem ?**

What if our count is zero or that kind of tri-gram combination isn't in our training corpus ?

There is another problem called the sparsity problem , what happens if the number on the top , the numerator , what if the count is equal to zero ? what if some particular word w , in the phrase "students opened their w " is never occurred in the data. For example suppose students opened their petri dishes , is fairly uncommon and it never happens in the data then that means our probability of the next word being petri dishes will be zero

and this is bad because it might be uncommon but it's a valid scenario. This is a problem called sparsity problem, so one partial solution to this problem is that we may be should add a small delta to the count of every word in the vocabulary and then this way every possible word that comes has at least a small probability, so petri dishes will have some small probability, so this technique called smoothing, because the idea is we're going from very sparse probability distribution, which is zero almost everywhere with a few spikes where the vocabulary in the training corpus(count(students opened their w)), so smoothing is all about going from that to being a more smooth probability distribution where everything has a small probability on it.

The second problem is what happens if the number in the denominator is zero? in our example that would mean, what if we never saw the trigram "students opened their" in the training corpus, if that happens we can't even calculate the probability distribution at all. Then what will be the possible solution? so a possible solution for this is that if we can't find "students opened their" in the corpus, then we should back off to just conditioning on the last two words rather than the last three words, see "open their" this is called **back off**, when you have no data for your 4 gram language model,

you are backing off to a trigram language model, if there is no a trigram in our training corpus we will back off to the bi-gram language model and so on.

Another problem is if we increase N very much, if we make N larger in our n-gram language model, we might want to do this because to have a larger context and pay attention what happens longer ago for a better predictor, we might think getting N bigger might be a bigger idea, but the problem is if we do that, **the sparsity problem gets worse**, let's say you want a 10 gram or a 9 gram language model, the problem is we are going to counting 9 grams or 10 grams but this will probably never occurred in our training corpus 9 grams or 10 grams but this will probably never occur in our training corpus which means the whole thing becomes dis-functional, In practice we usually can't make N much bigger than 5.

let's say we train out language model with 1.7 million training corpus with 2-grams

What to do if we have words that have the same probability?

Example :- today the _____

Here are some of the probability distributions

- Company :- 0.153
- Bank :- 0.153
- Price :- 0.077
- Italian :- 0.03

But we have a problem , the top two most likely words have the exact same probability , and the reason for that is , this number is 4/26 for both cases , we saw “today the company” and “today the bank” 4 times each , the reason for this is because we haven’t seen that many variety in our training corpus , so this will be solved by having large number of training corpus , if we have large number of training corpus it’s almost impossible to have such kind of cases.

A Language model to generate a text

Artificial Intelligence is now capable of writing scripts for a movie with the help of language models

```
THE JOKER
I am such a freak. Society is bad.
You drink water, I drink anarchy.

BATMAN
I drink bats just like a bat would!

Batman looks around for his parents, but they are still dead.
This makes him have anger. He fires a batrocket. The Joker
deflects it with his sick sense of humor. A clownly power.

THE JOKER
I have never followed a rule. That
is my rule. Do you follow? I don't.

BATMAN
Alfred, give birth to Robin.

Alfred begins the process since it is his job. The Joker now
has a present in his hand. He juggles it over to Batman.

THE JOKER
Happy batday, Birthman.

Batman opens the present since he's a good guy. It contains a
coupon for new parents, but is expired. This is a Joker joke.
```

Example :- today the _____

Here are some of the probability distributions

- Company :- 0.153
- Bank :- 0.153
- Price :- 0.077
- Italian :- 0.03

- Now we can sample from the above probability distributions , let’s say we sampled ‘price’ is the next word , now we can condition on the last two words

today the price _____

-now our new probability distributions is

- of :- 0.308
- for :- 0.056
- it :- 0.046
- to :- 0.046
- is :- 0.031

- Now let’s say we randomly sampled “of ” from the probability distribution

* Now sampling :- today the price of _____

so we can continue like this , sampling , conditioning again and then sampling again , so if we continue like this , we can go long enough , we can have a long text like this :-

“today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected”

Even though it’s good , it doesn’t make any sense , it’s incoherent , we need to consider more than these words if we want our model language well , but increasing N have the

sparsity problem and increases the model size.

Building a language Model with RNN

➔ what is the advantage of building a language model using RNN than n-grams?

Training set :- large corpus of english text

The RNN model learns from the training data , then it learns dependency (short term and long term dependency) , learn the connection between the words , map the words to their labels and extract feature from them.

Let's say , we see these sentence :-

- The cat , which already ate a bunch of foods, was full.
- The cat,which already ate a bunch of foods, were full.

Which one of the sentence is correct do we need to use was or were ? These kind of problem can not be solved by n-grams , The RNN model must hold long term dependencies and must also extract information between the words ,just focusing on the last n words doesn't help us to predict what the next word will be.

To build such a model using RNN you will first need a training set comprising a large corpus of english text or text from whatever language

you want to build a language model and the word corpus is an NLP terminology that means a large body of a large set of an english text

Let's say you found an english sentence say

:- Cats average 15 hours of a sleep a day

We are now finding the probability of the entire text , so the first thing we would do is tokenize this sentence (means a process of taking the input sentence and mapping it to the individual token or the individual words in the vocabulary)

| | | | | | |
|------------|-----------|-----------|-----------|-----------|---|
| cats | average | 15 | hours | of | a |
| sleep | a | day | <EOS> | | |
| $y^{<1>}$ | $y^{<2>}$ | $y^{<3>}$ | $y^{<4>}$ | $y^{<5>}$ | |
| $y^{<6>}$ | | $y^{<7>}$ | $y^{<8>}$ | $y^{<9>}$ | |
| $y^{<10>}$ | | | | | |

One thing we might also want to do is model when sentences end , which is to add an extra token called EOS that stands for end of sentence that can help you figure out when a sentence ends , the EOS taken can be appended to the end of every sentence in your training set and tokenize the EOS by also adding it in to our vocabulary then our model will explicitly capture when the sentences end.

What if some of the words in your training set are not from the vocabulary ?

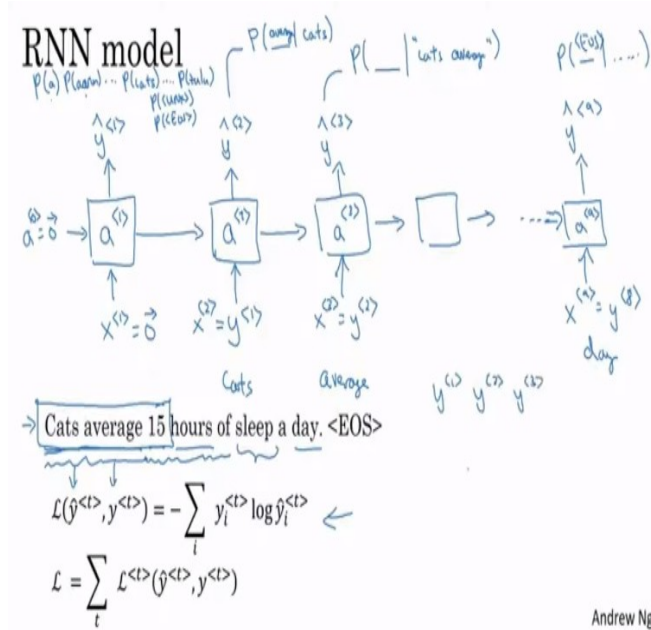
Example :- The Egyptian Mau is a bread of cat <EOS>

so in these sentence we could take the word MAU and replace it with a unique token called <unk> and just model the chance of the unknown word instead of a specific word mau.

one of the setting we will end up is $x^{<1>}$ is equal to $y^{<t-1>}$

Now let's build the RNN model :-

Example :- cats average 15 hours a sleep a day <EOS>.



$x^{<1>}$ is a zero vector , what is the input that makes the first $y^{<1>}$ prediction ? It's just the probability of that word given anything. So in the training set the word which has the highest probability will be selected ? But is this means that every time we want to generate a text it always starts with the

word which has the highest probability ? But if we look real world applications we will probably give it the first word and the machine will continue giving me the next word or continue to give a full text.

what makes a $a^{<1>}$ does is it will make a soft max prediction to try to figure out what is the probability of the first word $y^{<1>}$, it's trying to predict the probability of the first word from any word in the dictionary.

The job of a $a^{<2>}$ at this step is to try to figure out what is the second word but now we will also give it the correct first word and at this step the output is predicted by the soft-max the RNN'S job is to predict what is the chance of being whatever the word it is , a , Aron , cats , or Zulu given the previous words.

In each step in the RNN , we'll look at the some set of the preceding words such as given the first three words what is the distribution of the next word and these RNN learns to predict one word at a time going from left to right.

And if we train these RNN on large training set , given any initial set of words , it will predict what is the chances of the next word.

And given a new sentence say $y^{<1>}$, $y^{<2>}$, $y^{<3>}$,, $y^{<Ty>}$ then we will do $p(y^{<1>} , y^{<2>} , y^{<3>} , \dots , y^{<Ty>}) =$

$$p(y^{<1>}) * p(y^{<2>}|y^{<1>}) * p(y^{<3>}|y^{<2>}y^{<1>}) * \dots * p(y^{<t>}|y^{<1>} \dots y^{<t-1>}).$$

How do we calculate the probability ?

I think the way we are finding the probability of the next word is using the formula.

$$p(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

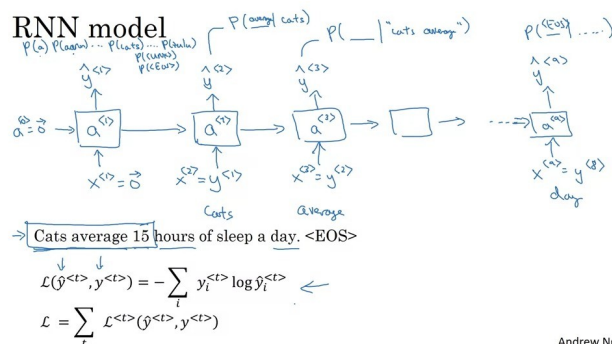
If we are calculating the probability this way , why are we not using N-grams simply ? and what is the advantage of the RNN blocks , did they even learn anything ?

One of the advantage of building a language model using RNN is that in the case of the n-gram model sparsity is a problem , the algorithm don't know what to do if there comes a word that is not from the training data , but in the case of the RNN , even if there is a vocab that is no from the training corpus the network will give relatively good predicted one.

Plus the Recurrent Neural Network enables us to have a long term dependency using concepts like GRU and LSTM.

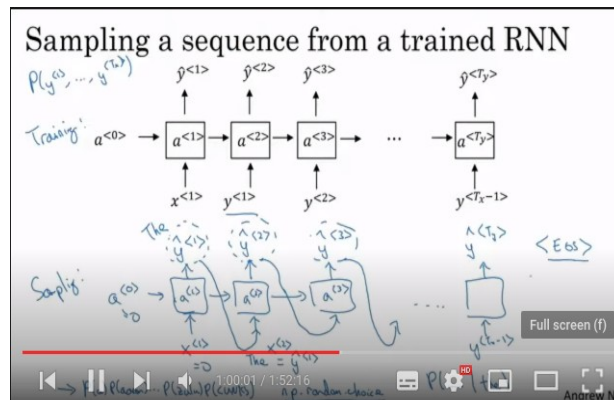
To train this neural network we're going to define the cost function. At a certain time 't' , if the word was $y^{<t>}$ and our network soft-max predicted

some $y^{<t>}$ this will be the soft max loss function and the over all loss is just the sum of all time steps of the losses associated with the individual predictions.



So if we train this RNN on a large training set what it will be able to do is given any initial set of words , it can predict what if the chance of the next word.

Sampling Novel sequences from a trained RNN'S (Randomly sampling a language model)



Questions :-

- ✓ What does it mean by a novel sequence
- ✓ Why do i need to choose it randomly among the probability distribution ? Why didn't i just choose the highest probability ?

A novel(ἄνωμα) is a relatively long work of narrative fiction, typically written in prose and published as a book. So a novel sequence is mean by a sequence of narrative words.

From the sequence model , models the chance of any particular sequence of words $p(y^{<1>} , y^{<2>} , \dots , y^{<T_y>})$ and now what we like to do is sample from this distribution to generate novel sequences of words.

so the first time step will have some soft-max probability over possible outputs ($y^{<1>}$) , so what we do is we then randomly sample according to these soft-max distribution , so what the soft-max distribution gives you is , it tells you what is the chance of the first is 'a' or what is the chance of the first

aron...., $p(a)$
 $p(aron)....p(cat)....p(zulu)....p(unk)$

and then we take these vector to randomly choose one of the distribution , for example using the numpy command ->
`np.random.choice()` to sample according from our distribution of vector probabilities.

In the second time step what ever word randomly we chose for the first time step we pass it as input in the second position (suppose let's say the first randomly selected word is "The " so this will be input for the second position)

so the $y^{<2>}$, we will try to figure out what is the chance of whatever the second word is given the first word is "the"

$p(__|The)$ and then randomly choose from the distribution.

How do we know when the sequence is end ? one thing we can do is if EOS token is part of our vocabulary we can keep sampling until we generate EOS token , alternatively if we don't include EOS in our vocabulary then decide to sample 20 words or 100 words or something and then keep going until you reached that number of time steps.

And this procedure will sometimes generate unknown word token if you want to make sure that your algorithm never generates these token one thing we can do is reject any sample that come out this unknown word token and just keep resampling from the rest of the vocabulary until there is no unknown word.

So far we have been building a word level RNN by which I mean the vocabulary are words from english. Depending on our application one thing we can do is also build a character level RNN so in these case our vocabulary would be just alphabets.

Vocabulary = [a,b,c,....z, . , !,?, A,B,....,Z , 0,1,2,....9]

The vocabulary may be combination of lower case, punctuation , upper case ,digits. beside these one thing we also could do is actually look at your training set corpus and look the characters that appear there and use that to define your vocabulary.

And if we build a character level language model rather than a word level language model then your sequence $y_{<1>}$, $y_{<2>}$, $y_{<3>}$, $y_{<t>}$ would be individual characters in our training data , so far in our previous example the sentence “cats averaged 15 hours of a sleep a day” , in these example ‘c’ would be $y_{<1>}$, ‘a’ would be $y_{<2>}$, ‘t’ would be $y_{<3>}$, the space would be $y_{<4>}$ and so on.

using a character level language model has some ‘pros’ and ‘cons’ , one advantage is that we don’t ever to have to worry about unknown tokens in particular in character level language model is able to assign a sequence like “mou” a non zero probability , where as if “mou” was not in your vocabulary for the word level language model , you just have to assign it the unknown word token.

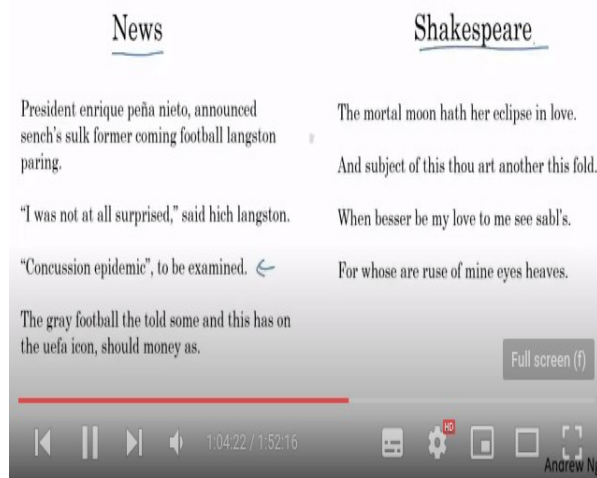
But the main disadvantage of a character level language model is that we end up with a much longer sequences , many english sentences may have 10 to 20 words but may have many many dozens of characters

and character level language models are not as good as word level language models at capturing long range dependencies between how the earlier part of the sentence also affect the later part of the sentence. Also character level language models have more computationally expensive to train.

For the most part of word level language models are still used but as computers get faster more and more applications where peoples are used in the specialized applications starting to look at more character level models. But they tend to be much hardware, much more computationally expensive to train, so they are not in widespread use today. Except for maybe 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 applications where you have a more specialized vocabulary(may be punctuations , specialized characters).

For example:- Here is an example , there was sample from a language model actually from a character level language model , if the model was trained on news articles and generate text shown on the left and it was trained on Shakespeare in text and generate texts that sounds Shakespeare would written it.

Sequence generation



Bi-Directional Recurrent Neural Networks

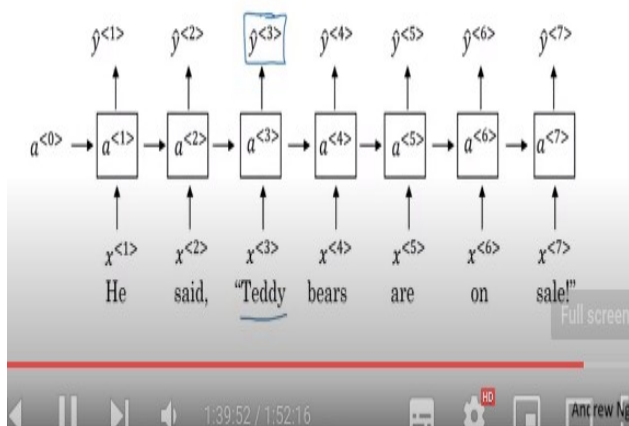
Getting information from the future :-

- He said , “Teddy bears are on sale !”.
- He said , “Teddy Roosevelt was a great president ”.

Getting information from the future

He said, “Teddy bears are on sale!”

He said, “Teddy Roosevelt was a great President!”



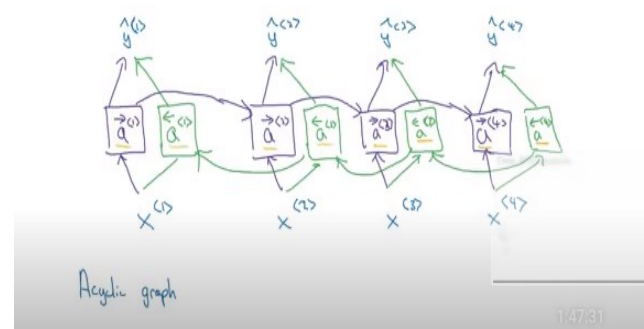
But one of the problems of this network is that to figure out whether the third word teddy is a part of a person's name is not enough to look at the first part of the sentence , so to tell $y^{<3>}$ is 0 or 1 , we need more information than just the first three words because the first three words doesn't tell if they're talking about teddy bears or the former president Teddy Roosevelt.

So this is a uni-directional or a forward directional only RNN and in our above example , we just made a standard RNN but it can be GRU or LSTM. so what is a bi-directional RNN , how does it fix this issue ? .

Bi-Directional RNN :- These networks hidden layer will have a forward recurrent component and a backward recurrent component.

These networks has a forward component , so we will put a right arrow on their top of the activations to denote that it is a forward component.

Bidirectional RNN (BRNN)



so the prediction $y^{<t>}$ will be calculated as :-

$$y^{<t>} = g(w_y[a^{<t>}, a^{<t>}], b_y)$$

So if we look at the prediction at time step 3 then information from $x^{<1>}$ can flow thorough $a^{<3>}$ then $y^{<3>}$, so information from $x^{<1>}$, $x^{<2>}$, and $x^{<3>}$ they are all taken to account whereas information from $x^{<4>}$ flow through a backward $a^{<4>}$, to $a^{<3>}$, then to $y^{<3>}$, so the prediction at time 3, to take the input both information from the past as well as information from the present ($x^{<3>}$) which both the forward and back ward goes to the same step.

Notice that this network defines a cyclic graph then given an input sequence $x^{<1>}$ to $x^{<4>}$ then go to the forward to compute $a^{<1>}$ $a^{<2>}$ $a^{<3>}$ and backward sequence will start to computing a backward and then go back and compute $a^{<4>}$, $a^{<3>}$ and notice that we are computing Network activations, this is not back propagation, this is forward propagation but bi directional, it goes left to right and part of the computation goes from the right to the left.

For this example we just used a standard RNN but they can also be GRU and LSTM.

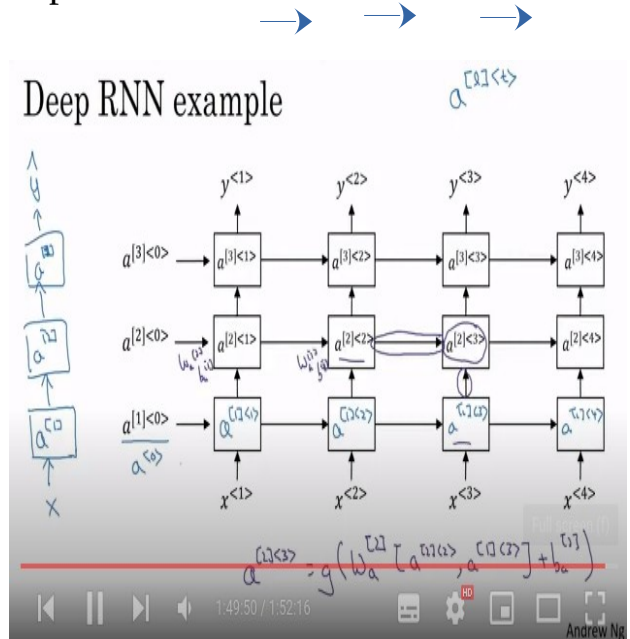
The disadvantage of Bi-directional RNN is that, you do need the entire sequence of data before you can make

a prediction, so for example, if we are building a speech recognition system, so BRNN will let you take in to account the entire speech utterance, so if we use BRNN we need to wait for the person to stop talking to get the entire utterance before we actually process it and make a speech recognition prediction.

So is that mean, we don't use Bi directional for Language Models?

Deep RNN

For learning very complex functions, it's sometimes useful to stack multiple layer's of RNN together to build even deeper versions of these models.



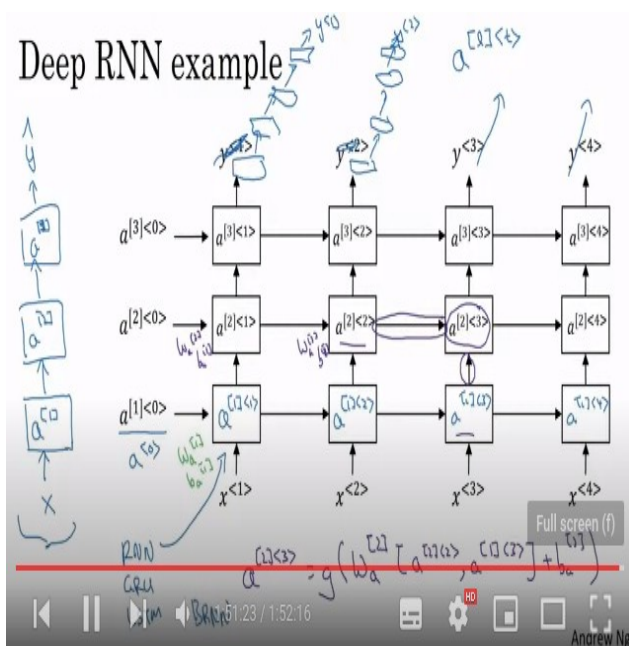
we changed a little bit of notation, instead of only saying $a^{<0>}$ we just added a square bracket notation to denote the layer $a^{[l]<t>}$, which means an activation with a layer l and a time step t .

so let's look an example :- to see how these value is computed , let's select $a^{[2]<3>}$, has two inputs from the $a^{[2]<2>}$ (input from the left) and an input coming from the bottom ($a^{[1]<3>}$) so to compute an activation function G , applied to a weight matrix.

$$a^{[2]<3>} = g(wa^{[2]<2>} , a^{[1]<3>} + ba^{[2]<2>})$$

for RNN's having three layer is already quite a lot because of the temporal dimension these networks can already get is quite big , even if you have just a small handful of layers and we don't usually see these stacked up (people's don't use it that much).
How does a three layer recurrent neural network be quite a lot dimension ?

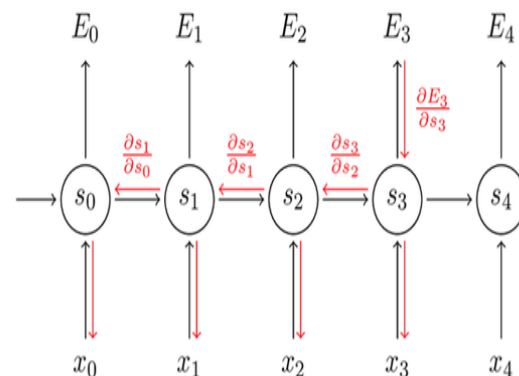
what we see that mostly people are using these one



A deep network but they are not connected horizontally. In these example we just saw a standard RNN but we can also use GRU , LSTM or we can build a deep version of RNN. But generally Deep RNN's are computationally expensive.

Vanishing and Exploding Gradient Problem in RNN

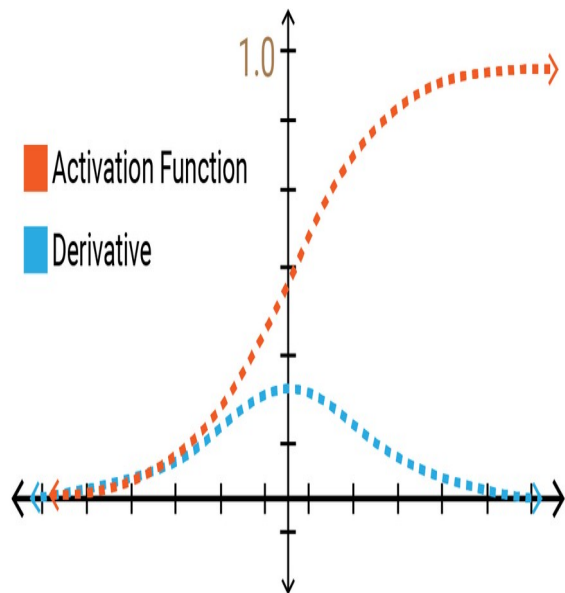
- what is Vanishing Gradient Problem
- How does Vanishing gradient problem occur ?
- what about Exploding gradient ?
- Why i am facing a vanishing gradient problem in the Recurrent neural networks ?



Backpropagation Through Time

These vanishing problem can be avoided when we use ReLu activation function, but mostly RNN uses tanh as an activation for the hidden layer.

Now vanishing gradient problem occurs if we use sigmoid or tanh activation function, because the derivative of these activation functions is very small, suppose in the sigmoid activation function our derivative range between $0 \leq \text{gradient} \leq 0.25$.



so in order to update the weight of the lower layers we need to use chain rule and multiply so many numbers layer by layer until we reach the weight for the lower layers.

- let's say our network is standard neural network :-

$$\begin{aligned} \frac{d(\text{loss})}{dw_1} &= \frac{d(\text{loss})}{d(\text{output})} * \\ &\frac{d(\text{output})}{d(\text{dlayer2})} * \\ &\frac{d(\text{dlayer2})}{d(\text{layer1})} \\ &= 0.20 * 0.1 * 0.05 \end{aligned}$$

so multiplying so many fraction numbers gives you even smaller number, so if we go deeper in the network, the lower layer gradient will vanish and if we subtract to the gradient descent, it won't update

$$w_j := w_j - \alpha \left(\frac{d(\text{loss})}{dw_j} \right)$$

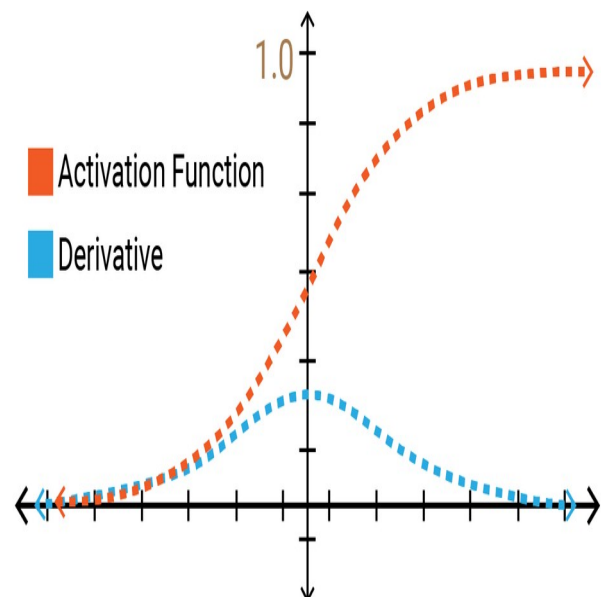
so if our gradient is not making any change on updating our weights, the old and the new weights will be almost the same, then it will be too slow to converge to the optimal.

Derivative of a sigmoid activation function

As we studied the derivative of a sigmoid activation function for a back propagation is

$$\begin{aligned} g'(z) &= g(z)(1 - g(z)) \\ g'(z) &= a(1 - a) \end{aligned}$$

so let's say $g(z)$ or a is 0.5 then the derivative will be 0.25

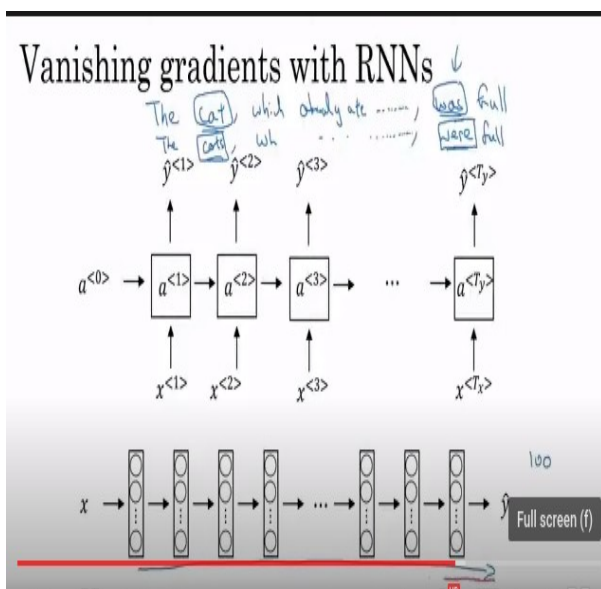


What about Exploding Gradient Problem ?

When building a deep networks , the lower layers will get explode instead of vanishing. so in every step of the chain rule to compute the gradient of the lower layers , the multiplication in every step will become larger and larger , then when we update the weights the old value and the new value of the weights will be very much varied , so these makes the gradient descent to jump and diverge.

Vanishing Gradient Problem In RNN

It turns out that one of the problems in the basic RNN is that it runs in to vanishing gradient problem.



Let's say we see these sentence :-

:- The **cat** , which already ate a bunch of foods , , **was** full.

:- The **cats** , which already ate a bunch of foods , , **were** full.

These is one example of when language can have a very long term dependencies where a word that is much earlier (The cat , The cats) can affect what needs to come much later(was , were).

But it turns out that the Basic RNN we've seen so far are not very good at capturing a very long term dependencies , the reason is vanishing gradient problem , that training a very deep neural networks than the gradient form just output y would have a very hard time propagating Back to effect the weights of the earlier layers.

And for RNN , we have the same problem , we have forward propagating going from left to right and Back prop going from right to left and it can be very much difficult because of vanishing gradient problem for the outputs of the errors associated with the later time steps to affect the computations that are earlier.

So In practice what this means is it might be difficult in your network to realize that it needs to memorize, did you see a singular noun or a plural noun so that later on the sequence you can generate either "was" or "were" depending on whether it was singular or plural and in English the staff in the middle(which already ate a bunch of foods.....) could be arbitrarily long , so we might need to memorize the singular plural for a very long time

before we get to use that bit of information.

Language Model Task :- The Writer of the books _____

correct answer :- The writer of the books _is_ planning a sequel.

Syntactic Recency :- The **writer** of the books **is**

:- we are considering what is the syntactically close word in here

Sequential Recency :- The **writer** of the book **are**

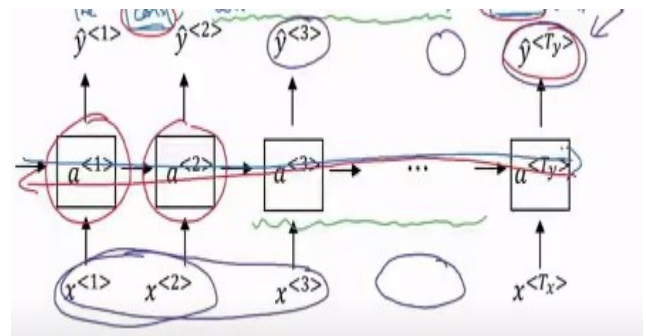
:- we are considering the simpler concepts how close words are just in the sentence as a sequence of words. In this example books are very sequential recent because they are next to each other

note :- the word recency means :- The act of being Recent

So the problem is the basic RNN language model we've seen so far is better in learning from sequential recency than from a syntactic recency, and this is due to vanishing gradient problem because if you syntactically related word is actually far away and it might get really hard to use information from the syntactically word.

Because of the problem with the Basic RNN model has many local influences, meaning that the output $y_{<3>}$ is

mainly influence by values close to $y_{<3>}$



So in the Basic RNN the values of some prediction y is influenced by inputs that are somewhat close and it's difficult to be strongly influenced by an input that was early in the sequence.

Even though our discussion is focused in vanishing gradient problem there are also, Exploding gradient problems, where doing Back prop the gradients should not just decrease exponentially with the number of layers we go through, It turns out that vanishing gradients tends to be a bigger problem with training RNN's although when exploding gradients happens, it can be catastrophic because exponentially large gradients can cause your parameters to become so large than your neural networks parameters and get really messed up, so it turns out exploding gradient easily to spot because the parameters will be just blow up and we might often see NAN(not numbers) or inf (infinity), meaning your results of numerical overflow in your neural network

computation. The solution to Exploding gradient problem is Gradient Clipping.

Gradient clipping: solution for exploding gradient

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

```

 $\hat{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq \text{threshold}$  then
   $\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$ 
end if

```

- The idea of Gradient Clipping is pretty clear , we will compute the norm of the gradient and check if it is greater than a certain threshold , if it is use some techniques to scale it down before applying gradient descent.

So we saw on how training a very deep neural network can ran in to a vanishing gradient or exploding gradient problems with the derivative either increases or decreases exponentially as a function of the number of layers and RNN processing data over a 10 ,000 time step or a 10 ,000 time step , that is basically a 1000 layer or a 10,000 layer neural network , so it runs in to the above problems.

So in order to solve this problems we will use 1) GRU(Gated Recurrent Unit

) 2) LSTM (Long Short Term Memory)

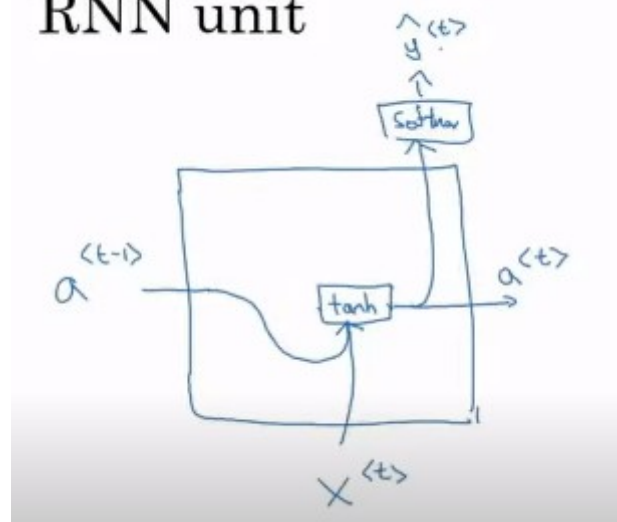
GRU'S (Gated Recurrent Units)

This is a modification to the recurrent network and hidden layer that makes it much better at capturing long range connections and helps a lot with the vanishing gradient problem.

In the RNN UNIT :-

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

RNN unit

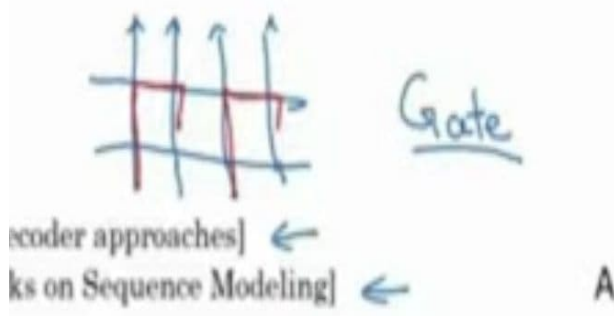


Example :- The cat , which already atewas full

- So as we read in these sentence from left to right , the GRU unit is going to have a new variable c , which stands for cell for memory cell.

And what the memory cell do is , it will provide a bit of memory to remember for example whether a cat was singular or plural , it gets much further in to the sentence it can still worth under consideration whether the subject of the sentence is singular or

— poor



And at time t , the memory cell will have

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

So the memory cell will have some value $c^{<t>}$ and the GRU unit actually output an activation value $a^{<t>}$ which is equal to $c^{<t>}$. And for now , let's use different symbols $c^{<t>}$ and $a^{<t>}$ to denote the memory cell value and the output activation value , even though they are the same , we are using these notations is because when we talk about LSTM's , these will be two different things.

In every time step we are going to consider over writing the memory cell with a value $\tilde{c}^{<t>}$:- \tilde{c} tilde of t (tilde

means an english notation for the approximation \sim), which is gonna be a candidate for replacing the memory cell $c^{<t>}$ and we're gonna compute this like this

$\tilde{c}^{<t>} = \tanh(w_c[c^{<t-1>}, x^{<t>}] + b_c)$, this is a candidate for replacing $c^{<t>}$

And the important idea of GRU is the Gate ,

$$\Gamma_u = \text{sigmoid}(w_u[c^{<t-1>}, x^{<t>}] + b_u)$$

This notation is the capital Greek alphabet gamma , and 'u' stands for update gate and as you saw in the above it's a sigmoid function , which means it's value is between 0 and 1.

The reason we chose the alphabet Gamma is because if we look to the Gate in our house there are a lot of Gamma's.

and also Gate is G and Gamma is G , so G for gate and G for Gamma.

We have come up with a candidate weight ($\tilde{c}^{<t>}$) , we're thinking of updating $c^{<t>}$, using $\tilde{c}^{<t>}$ and the gate gamma will decide whether or not actually are updating.

The way to think about it is may be this memory cell is going to set to either 0 or 1 , depending on whether the word we are considering really

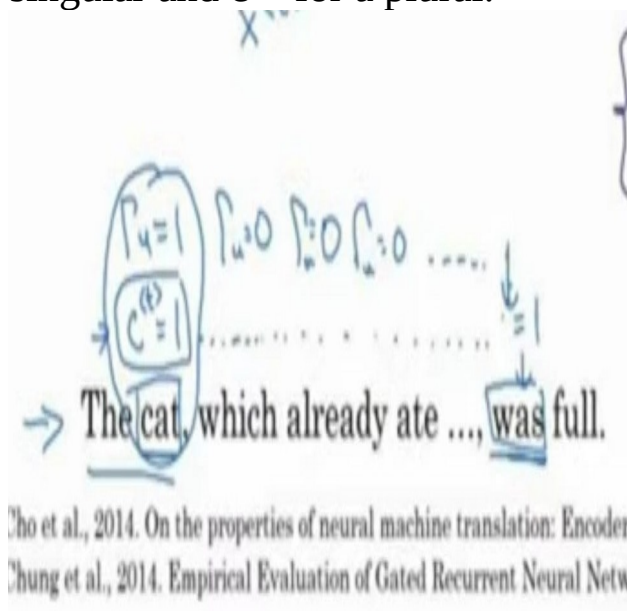
is the subject of the sentence , singular or plural.

So in our example since it's the subject of the sentence (so for now our memory cell is considering whether the word is subject or not), so to decide that this memory cell is a subject of a sentence , say

:- Update Gamma = 1 , if the word is a subject

:- Update Gamma = 0 , if the word is not a subject

and also we need to give some representation for making a distinction that subject is a singular or plural , let's say $c^{<t>} = 1$ for a singular and $c^{<t>} = 0$ for a plural.



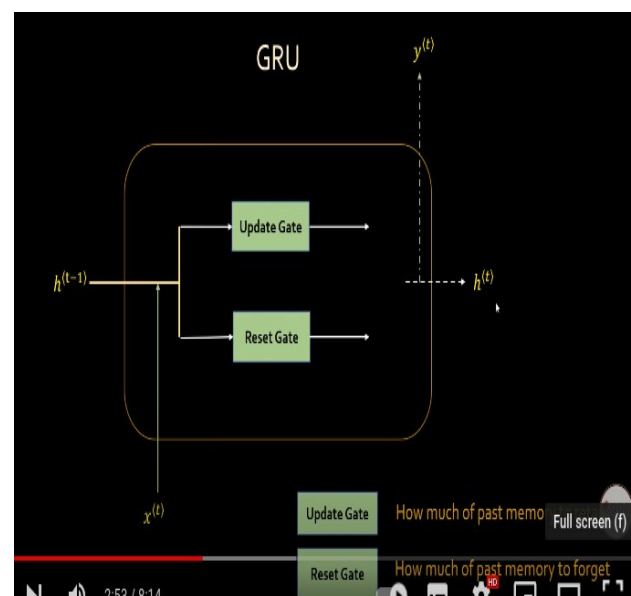
These will be candidates when we go from the left to the right and using Gamma to decide that they will replace $c^{<t>}$ or not. So in our example

the value of $c^{<t>}$ will be memorized all the way until it says “oh it is singular so it will be was”.

So the specific equation we will use for GRU is the following :-

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

So notice in the formula , if the gate or the update value is 1 , then the candidate value $c^{<t>}$ will replace the $c^{<t-1>}$ but if our gate or update value is 0 , then $1 - \Gamma_u \rightarrow 1 - 0$, which is $c^{<t-1>}$ and $\Gamma_u * c^{<t-1>}$ will be zero , so the candidate will not replace $c^{<t-1>}$, so $c^{<t>}$ will be equal to $c^{<t-1>}$ preserving the old value . In our example $c^{<t>}$ is “which ” and $c^{<t-1>}$ is “cat” so the candidate word is not a subject so the new memory cell will preserve the previous memory cell $c^{<t>} = c^{<t-1>}$, so it will go all along without updating so it will memorize the cat was singular.



Example :- Dhaval eats **Samosa** almost ever day , it shouldn't be hard to guess that his favorite cuisine is _____.

These is something the machine needs to predict , suppose we are doing email Auto Completion.



Example :- The **dog** , which ran our**has** come back.

If you use a bit memory cell for defining dog is singular or plural , we can use another bit to tell the GRU that the sentence is “talking about going out the house or entering the house ” and make subsequent prediction accordingly. so this means we will have so many memory cell for different scenarios.

LSTM(Long Short Term Memory)

- What is the meaning of Long Short Term Memory
- Why would I use it ?

- We are finding a way that pertains long term dependencies in the forward direction of our model , but vanishing gradients happens during back prop so , if that so how does vanishing problem affect our language models ?
- What makes LSTM differ from GRU ?

so on each step t , instead of having a hidden state $a_{<t>}$, we have both the hidden state $a_{<t>}$ and the cell state $c_{<t>}$ and both of these are vectors of the same length ? how ?

The important thing is LSTM can erase , read and write information from the cell , so the way the LSTM decides whether it wants to erase , write or read is controlled by the corresponding gates. So in each time step , each element of this gates which are vectors somewhere between zero and one. 1 represents an open gate and 0 represents a closed gate. and we can have values in anywhere between them. If the gates are open , some kind of information being passed through and if the gate is closed it means information does not pass through.

We have a sequence of inputs $x_{<t>}$ and we will compute a sequence of hidden states $a_{<t>}$ and cell states $c_{<t>}$.

on each time step t :-

Forget Gate :- $\text{sigmoid}(w_f[a^{<t-1>}, x^{<t>}] + b_f)$, these controls what is kept

versus what is forgotten from the previous memory

Input Gate :- $\text{sigmoid}(\mathbf{w}_i[\mathbf{a}^{<t-1>} , \mathbf{x}^{<t>}] + \mathbf{b}_i)$, these controls what part of the new cell content are written to the cell

Output Gate :- $\text{sigmoid}(\mathbf{w}_o[\mathbf{a}^{<t-1>} , \mathbf{x}^{<t>}] + \mathbf{b}_o)$, these controls which part of the cell is outputs by the hidden state (or which part of the cell is read by the hidden state) , this means we are reading some information from our memory cell and that's gonna get put in to our hidden state.

Question :- In the forget gate , if i am calculating the amount of the forget or kept from the previous memory cell , why am i dealing with the previous activation function $\mathbf{a}^{<t-1>}$? .

- Now let's calculate the candidate cell content , which is gonna replace the old memory cell

$$\begin{aligned}\tilde{\mathbf{c}}^{<t>} &= \tanh(\mathbf{w}_c[\mathbf{a}^{<t-1>} , \mathbf{x}^{<t>}] + \mathbf{b}_c) \\ \mathbf{c}^{<t>} &= \mathbf{f}^{<t>} \circ \mathbf{c}^{<t-1>} + \mathbf{i}^{<t>} \circ \tilde{\mathbf{c}}^{<t>}\end{aligned}$$

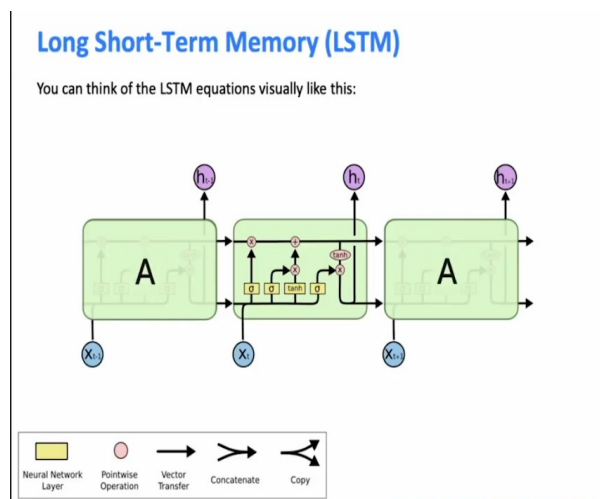
so in these case we are using the forget gate to selectively forget some of the information from the previous memory cell, if we set to zero, we will completely ignore the information from the previous memory cell . we also used the input get to selectively control what amount of information are we writing from the candidate memory cell ? if we set it to 1 it means we are completely writing the

candidate memory cell to our new cell , but if we set it to zero then the previous memory cell will be our new memory cell , which is holding a long range dependency.

$$\mathbf{a}^{<t>} = \mathbf{o}^{<t>} \circ \mathbf{c}^{<t>}$$

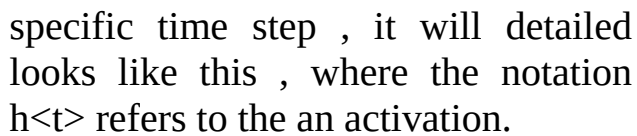
The result of $\mathbf{c}^{<t>}$ is filtered out by $\mathbf{a}^{<t>}$, so in LSTM, we often think the hidden states as being like the outputs of the RNN , because we kinda view the cell states as being internal memory , that's not generally accessible to the outside , but the hidden states are gonna pass to the next part of the model.

In the last operations the 'o' represents an element wise operation , which is because all of the notations up there are vectors of the same length n , how ?



on each of the time steps , LSTM looks like this , but if we look one

You can think of the LSTM equations visually like this:



- ➔ What does it mean by word embedding
- ➔ why do we need to use word embedding ?
- ➔ Why is better than the one hot encoding word representation ?
- ➔ What is the advantage of analogy relationships in the natural language processing ?
- ➔ What are some application of word embedding ?

So far we've been representing words using vocabulary of words and vocabulary in our previous example was $|V| = 10,000$ words

$v = [a, aaron, \dots, Zulu, unk]$

we are representing the words using one hot vectors

One Hot Representation :- means representing words by setting one in one position and zero in the rest position. For Example :- if man is word number 5391 , in our dictionary then we will represent with a vector one in the position 5391

One of the weakness of this representation is that it treats each word as a thing until itself and it doesn't allow to generalize easily across words for example :- let's say you have a language model that has learned " I want a glass of orange _____ ", so very likely to be juice , so

even if the learning algorithm has learned that i want a glass of orange juice is a likely sentence but if we say :-

I want a glass of Apple_____, as far it knows the relationship between Apple and Orange is not any closer as a relationship between any of the other words , like man-woman, king-queen and orange. The relation ship between the word orange and the word Apple is no any closer in the vocabulary, orange is indexed in the position 6257 and Apple is indexed in the position 456. so this means it just doesn't know that somehow Apple and orange are much more similar than king and orange or queen and orange. This is the one thing that learning language model with RNN model is better than the n-grams because even if the nominator is zero (because $p(I \text{ want a glass of Apple } | w)$ is zero) , the RNN tries to learn relationship among the words , so if we represent our words using one hot encoding , it tries to learn the relationships by the index in the vocabulary.

But the problem is representing words using One Hot representation has problems like that. This means one hot vector representation just does not know somehow Apple and orange are more similar than king and orange or queen and orange. so it would be nice if we could use Featurized Representation.

Featurized Representation :- Word Embedding

one hot position of 6392

| | Man (6391) | Woman (9253) | 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 | 0.01 |
| 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
- core
- alive
- action
- noun
- verb

→ We can imagine
Coming out with many features

Man & Woman doesn't tell anything about age

King and Queen are almost always Adult

REDMI NOTE 6 PRO
MI DUAL CAMERA

so the numbers in the featurized representation shows that the effect of that feature on that specific word in the vocabulary, like how the word “man” is affected by the feature “Food”, so as we know we are converting our problem in to numbers so that it will be easy for the algorithm to understand this numbers and extract pattern out of it , to the lower number shows that this feature has almost no effect to that word and a high word shows that this feature has effect to that word.

So for now , let's say 300 different features and what that does is it allows you to take is a list of 300 dimensional vector so each word will have this 300 dimensional vector

If we use this representation to represent the words orange and Apple , then notice that the representation for orange and apple is quite similar. **This means most of the features not all of the features** , some of the features may be differ because may be the color of an orange and the color of an apple , the taste order , or some other feature would differ but by in large a lot of feature of apple and orange are actually the same , so this increases the odds of the algorithm that has figured out that orange juice is a thing to also quickly figure out apple juice is also a thing. so this representation helps us to generalize better across different words.

So what is the use of word embedding ? It helps us to generalize different words.

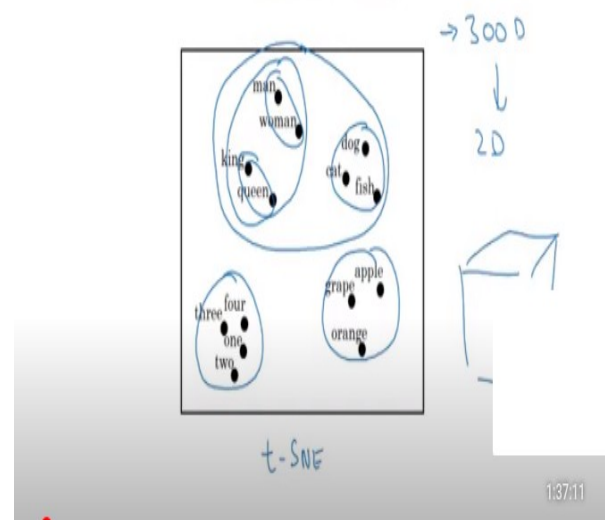
So we will find a way to learn a word embeddings which is basically seem to learn high dimensional feature vectors like this , that gives a better representation than one hot vectors. So Featurized representing will allow an algorithm to quickly figure out that Apple and Orange are more similar than king and orange or Queen and orange.

Visualizing Word Embeddings

- ➔ Why do we need to visualize word embeddings ?
- ➔ How this t-SNE algorithm works ?

- ➔ What does it mean by visualizing word embeddings?
- ➔ Why do i need to visualize it ?
- ➔ What is t-SNE algorithm
- ➔ Why do i need to use a t-SNE algorithm ?
- ➔ What does it mean by the word Embedding ?

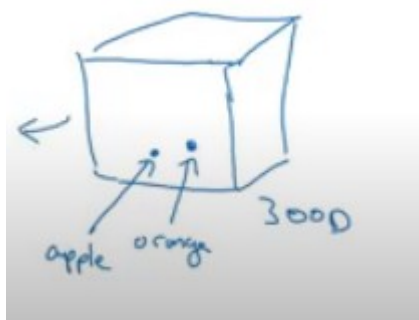
Visualizing word embeddings



If we're able to learn a 300 dimensional feature vector or 300 dimensional embedding for each word one of the popular things to do is also to take this 300 dimensional data and embedded say in to two dimensional space so that we can visualize them and one common algorithm for this is t-SNE algorithm (Distributed Stochastic Neighbor Embeddings).

So the point of reducing the 300 dimension in to two dimension is for the purpose of the visualizing because it's hard to visualize multi dimensional data using scatter plot or any plot.

The input for the t-SNE algorithm is a high dimensional data-set \mathbb{R}^n and the output is a 2 dimensional embedding \mathbb{R}^2 . So the principle of a t-SNE algorithm is reduce a certain dimension to a 2D and helps us to keep similar data points close together in the 2-D embedding. so if we use this algorithm we will find that words like man and woman will be grouped together , fruits will be grouped to each other and numbers will be grouped together, humans and animals can be grouped as animate objects. and these featurized representation , may be 300 dimensional space, these are called embeddings and **the reason we called them embeddings is we can think each and every word and get embedded to a certain point in these 300 dim space.** Example :- the word orange is embedded like in the image and also the word apple is embedded in this 300 dimensional space.

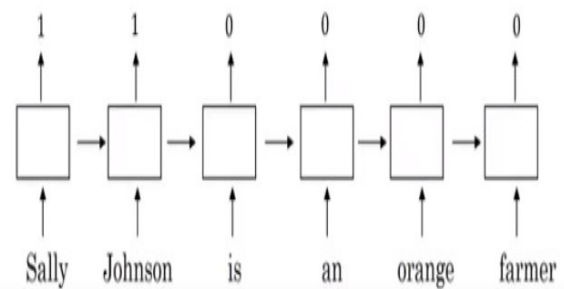


And to visualize it algorithms like t-SNE map this 300 dimensional vector space to a lower dimension for the purpose of visualization (you can plot the 2D data)

Using Word Embeddings

Now let's take these featurized representation and plug them in to NLP applications

Named entity recognition example



Example :- continuing the Name Entity Recognition

If we are trying to detect people's name given a sentence like "sally Johnson is an orange farmer " , hopefully we figure out that "sally Johnson" is a person's name , the output y is one and one way to be sure that sally Johnson is a person's name rather than it's a corporation name is that "orange farmer" is a person.

Previously we've seen one-hot representation to represent $x^{<1>}$, $x^{<2>}$, $x^{<3>}$, ..., $x^{<Tx>}$, but if we can now use a featurized representation, the embedding vectors after we trained a model that uses word embedding as the inputs, if we now see a new input in the test set :- “Robin Linn is an apple farmer”, so from the training set knowing the orange and the apple are very similar, so it will make it easier for our learning algorithm to generalize that “Robin Linn” is also a person's name.

But what if in our test set, we see not Robin Lin is an Apple farmer but we see much less common words, what if we see, “Robin Lin is a durian cultivator”, where durian is a rare kinda fruit which mainly exists in singapoure.

- But if we have a small label training set for the named entity recognition task you might not even see the durian or cultivator in our training set, but if we learned a word embedding that tells you durian is a fruit and cultivator is someone cultivates is a farmer then we might still can generalize from having seen an orange farmer in our training set to knowing that durian cultivator is also probably a person, so one of the reasons where the embeddings will be able to do this is the algorithm for learning word embeddings can examine very large text corpses may be found off the

internet so we can examine very large datasets may be a million words, may be even up to 100 billion words this could be quite reasonable so very large training sets are just unlabeled text and by examining tons of unlabeled text which you can download or for free, so that you can figure out that durian and orange are kinda similar and therefore learn embeddings that can group them together.

so we discovered that orange and durian are both fruits by reading massive amount of internet text, what we can do is take this word embedding and apply in to our named entity recognition system, for which we might have a much smaller training set may be even just a hundred thousand words in our training set or even much smaller, so this allows you to carry out **transfer learning**, where we take information we learned from huge amount of unlabeled text, essentially from the internet to figure out that orange, apple, durian are fruits and then transfer that knowledge to a task such as named entity recognition for which we might have relatively small labeled training set.

Transfer Learning and Word Embedding

1) Learn word embeddings from large text corpus (1-100B words) or we can

download pre-trained embedding online under very permissive licenses.

2) and we can take those words and embeddings to transfer embedding to new task with smaller training set (say 100k words)

3) Optional :- continue to fine tune the word embeddings with new data, optionally we can continue to fine tune to adjust the word embeddings with the new data , on practice we would do this only if our task (number-2) has a pretty big dataset. we will fine tune if our task has larger data set then we will fine tune it (may be train earlier layers).

so word embedding tend to make the biggest difference when the task we are trying to carry out has a relatively small training set , so it has been useful for many NLP tasks such as

- Name Entity Recognition
- Text Summarization for co-reference resolution

and has been useful for the task such as

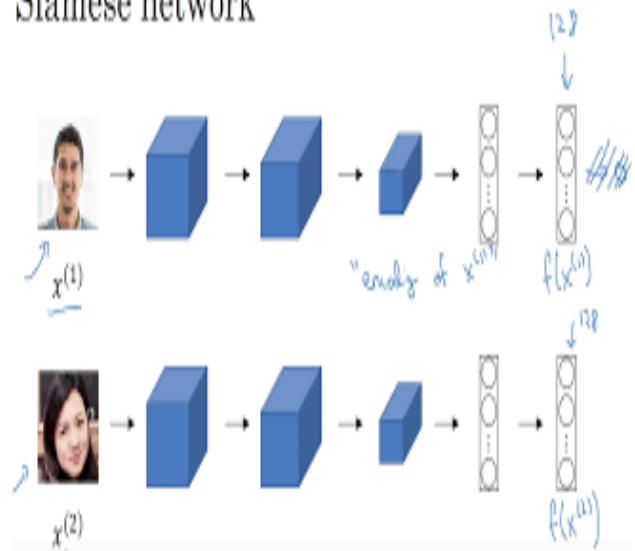
- Language Model
- Machine Translation Task

so transfer from some task A to some task B , the process of transfer learning is most useful when you happen to have a ton of data for A and relatively smaller data-set for B, and that is true for a lot of NLP task and

less true for some tasks such as language modeling and machine translation , but why transfer learning is not working for language modeling and machine translation?

Word Embedding In Relation with Face Encoding

Siamese network



The Siamese network from computer vision so we train these architecture that will learn say 128D representation for different faces and then we could compare these encodings in order to figure out if these two pictures are of the same face or not ?

The word embedding and encoding mean fairly similar things so in the face recognition literature people also use the term embedding to refer to these 128D vectors.

One basic difference between the face recognition and in word embedding is that for face recognition we wanted to

train a neural network that can take any input , any face picture even a picture we've never see before and have a neural network computing encoding for the new picture where as learning word embeddings is that we have a fix vocabulary say 10,000words and we will learn a vector $e_1, e_2, e_3, \dots, e_{10,000}$ that just learns a fix encoding for each of the words in the vocabulary.

The terms encoding and embedding are used somewhat interchangeably so the difference we just described is not represented by the difference in terminologies , it's just a difference in how we need to use these algorithms in Face Recognition , where there is unlimited see of pictures we could see in the future versus Natural Language Processing where there might be just a fixed vocabulary and everything else like that will declare as unknown word.

Properties of Word Embedding

Word embedding can help with reasoning about analogies , and reasoning about analogies by itself , may not be the most important NLP application , but they might give more sense of what word embedding can do.

Analogies

| | 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 | 0.70 | 0.69 | 0.03 | -0.02 |
| Food | 0.09 | 0.01 | 0.02 | 0.01 | 0.95 | 0.97 |

$e_{\text{man}} - e_{\text{woman}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
 $e_{\text{king}} - e_{\text{queen}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Let's say i posed the question :-
man is to woman as king is to ?

Many of you would say man is to woman as king is to queen , but it is possible for an algorithm to figure this out automatically.

let's say we are using the four dimensional vector to represent :-

| | |
|--|---|
| Man | and woman |
| $\begin{bmatrix} -1 \\ 0.01 \\ 0.09 \end{bmatrix}$ | $\begin{bmatrix} 1 \\ 0.02 \\ 0.01 \end{bmatrix}$ |
| e_{5391} | e_{9853} |

and similarly we will do for king and queen

$\begin{bmatrix} -2 \end{bmatrix}$

Then :- $e_{\text{man}} - e_{\text{woman}} \approx \begin{bmatrix} 0 \end{bmatrix}$:-
The main difference between man & woman

$\begin{bmatrix} 0 \end{bmatrix}$

is Gender

$\begin{bmatrix} 0 \end{bmatrix}$

:- $e_{\text{king}} - e_{\text{queen}} \approx \begin{bmatrix} -2 \end{bmatrix}$

$\begin{bmatrix} 0 \end{bmatrix}$:-The

main difference between king and queen is

also Gender(They are equally royal)
 $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is

One way to carry out this analogy reasoning is if the algorithm asks man is to woman as king is to what ?

- What we can do is compute $e_{\text{man}} - e_{\text{woman}}$ and try to find a word that $e_{\text{man}} - e_{\text{woman}}$ is close to $e_{\text{king}} - e_{\text{word}}$

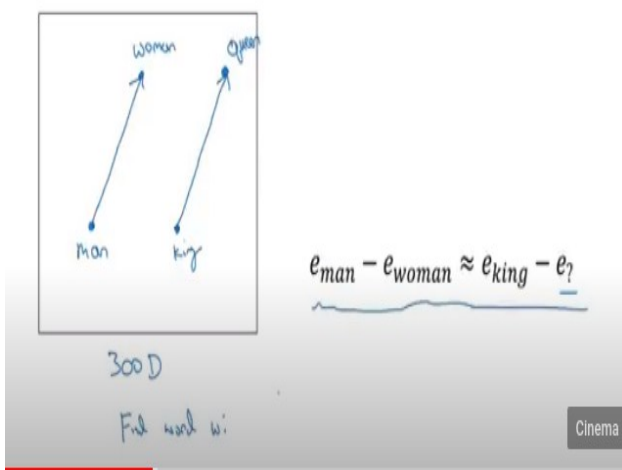
$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{word}} ?$:- so the left hand side is close to the right hand side , now let's formulate how we can turn these in to an algorithm.

Analogies using word vectors

➔ What are word vectors ?

➔ How does they help us for learning analogies between words ?

Analogies using word vectors



$$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{word}}$$

The vector difference between man and woman is very similar between king and queen.

What we can do is try to find a word w , so that :-

$$\text{argmax sim}(e_{\text{word}} , e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$$

we need a high degree of similarity between e_w and $e_{\text{king}} - e_{\text{man}} + e_{\text{woman}}$, we just make one side of the equation and other three terms to the right hand side. so if we have some appropriate similarity function for measuring how similar is the embedding of some word w to this quantity of the right and finding the word that maximizes the similarity should hopefully let you pick out the word queen. So the most commonly similarity function is called cosine similarity.

$$\text{sim}(e_w , e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$$

$$\text{sim}(u,v) = \frac{u^T v}{\|u\| \|v\|}$$

The nominator is basically an inner product and if u and v are very similar the inner product will be large , we will dive in to this more.

To measure the similarity we can also use Euclidean distance , although it's mainly used to measure dissimilarity rather than similarity and this will

work okay as well but cosine similarity used more often.

So one of the remarkable results about word embedding is the generality analogy relationship it can learn :-

For example :- it can learn that

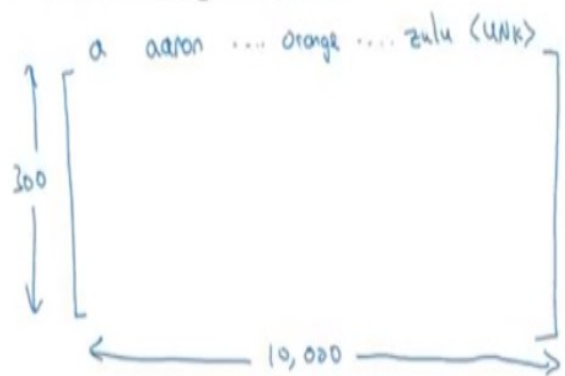
- man is to women as boys to girl
- Ottawa for Canada as Nairobi to Kenya (city capital to the name of the country relationship)
- Big:bigger as Tall to taller
- Yen:Japan as Ruble to Russia (currency to country relationship)

Embedding Matrix

- What does it mean by Embedding matrix ?
- Why would i need to study word embeddings ?
- What are some of it's applications in real world NLP ?

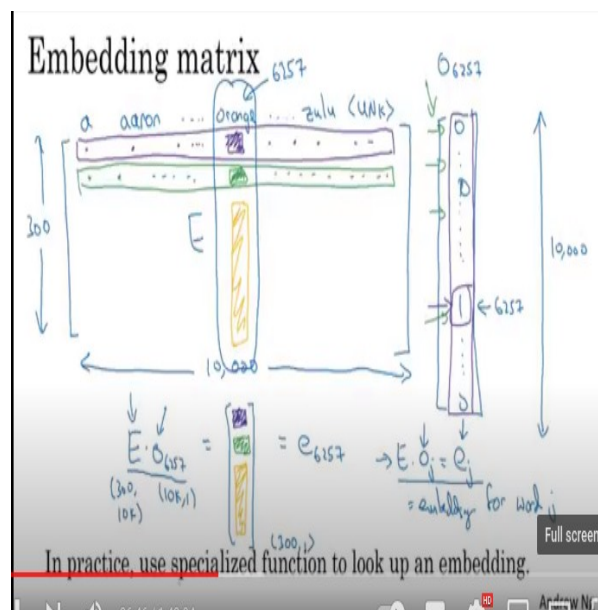
When you implement an algorithm to learn a word embedding what you end up learning is embedding matrix.

Embedding matrix



300 * by 10,000 word in vocabulary , adding the unknown word 10,001 , one extra token.

and the columns of these matrix would be different embeddings for the 10,000 different words we have in our vocabulary. so orange was word number 6257 in our vocabulary of 10,000 words , so one piece of notation we will use is that O_{6257} was



In practice, use specialized function to look up an embedding.

the one Hot vector , with zeros everywhere and a 1 in position 6257 and it will be a 10,000 dimensional vector with a 1 in one position.

So if we called the Embedding matrix E , then notice that , if we take E and multiply it by O_{6257}

The diagram shows a matrix E with dimensions $(300, 10K)$ and a vector O_{6257} with dimensions $(10K, 1)$. An arrow points to the result, which is a vector e_{6257} with dimensions $(300, 1)$. The equation is written as $E \cdot O_{6257} = e_{6257}$.

and notice that to compute the first element of this 300 Dimensional vector , multiply the first row of E with the O_{6257} and end up like in the image above and all of these elements in the E will be zero except the element E_{6257} .

$$E \cdot O_{6257} = e_{6257} \rightarrow (300, 1)$$

so e for a certain specific word w will be e_w - the embedding for the word w , so more generally :-

$$E \cdot O_j = e_j$$

O_j :- one hot vector with a 1 at the position j , where e_j is the embedding vector for the word j in the vocabulary.

:- Our goal is to learn an embedding matrix E but when we are implementing this it's not efficient to actually implement this as a matrix

vector multiplication , because the one hot vectors are relatively high dimensional vector and most of the elements are zero , so it's not efficient to use a matrix vector implementation.

- So in practice we will actually use a specialized function that just look up a column of the matrix ' e ' rather than do this with the matrix multiplication but when we write it out in math its convenient to write out $E \cdot O_j = e_j$

- In keras , there is an embedding layer , then it more efficiently pulls out the column from the embedding matrix E rather than does it with a slower matrix vector multiplication.

Learning Word Embeddings

As far as we know , we just saw the representation of the words using featurized representation , this helps us to convert our problem in to numbers which will be easy to feed to our learning model , to extract patterns.

- Now we will learn some concrete algorithms for learning word embeddings

- 1) Natural Language Model
- 2) other contexts & target pairs (Learning different contexts for word embedding)
- 3) Word2Vec
- 4) Negative Sampling
- 5) Glove word vector algorithm

Natural Language Model

Let's say in our training set we have these longer sentence

I want a glass of
orange _____?
4343 9665 1 3852 6163
6257

Let's say you're building a language model and we do it with a neural network, so during training we might want our neural network to do something like Input "I want a glass of orange" and then predict the next word in the sequence.

I O_{4343} -----> E
-----> e_{4343}

Want O_{9665} -----> E
-----> e_{9665}

a O_1 -----> E
-----> e_1

Softmax
glass O_{3852} -----> E
-----> e_{3852}

of O_{6163} -----> E
-----> e_{6257}

$w^{[1]}, b^{[1]}$

orange O_{6257} -----> E
-----> e_{6257}

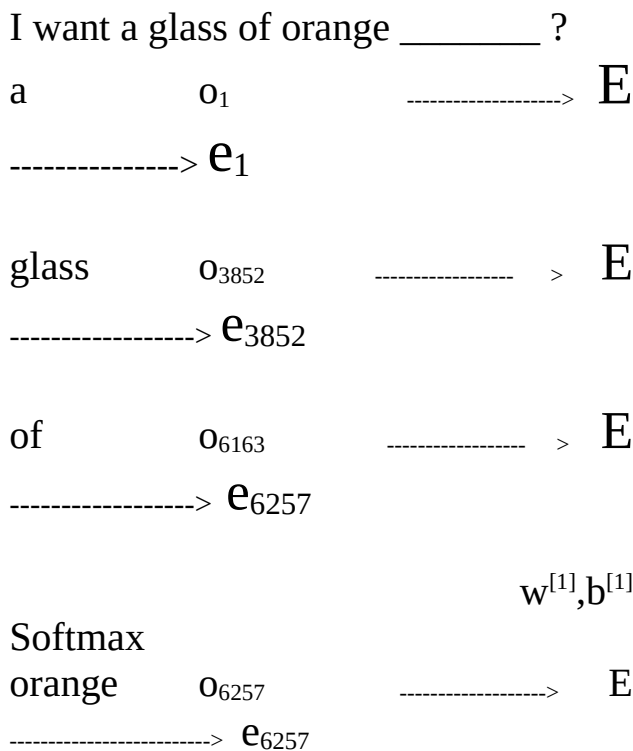
The symbol O represents a one hot vector to that given word, which is a 1 in position 4343, where E represents the embedding matrix and where e represents the embedding vector to that specific word.

Now we have a bunch of 300 dimensional embedding vectors and we will do to feed them in to a neural network. So the soft-max will classifies among from the 10,000 possible outputs in the vocabulary for this final word we are trying to predict. In the training set we saw the word juice then the target for the soft-max during training would be that is should predict the word juice is what came after this, if not, we will calculate the loss and back propagate, update the weight and try to predict again.

The hidden layer has its own parameters and also the soft-max layer has its own parameters $w[2], b[2]$.

The embedding vectors in this example are 300 dimensional embedding word and here we have 6 words so these would be 6×300 , so our input will be 1800 dimensional vector obtained by taking our 6 embedding vectors and stacking them together.

What's actually more commonly done is to have a fixed historical window , for example we always went to predict the next word , given say the previous four words , where 4 here is the hyper parameter of the algorithm.



$$4 * 300 = 1200$$

The parameters of these model will be this matrix E , and we use the same matrix E for all of the words , we don't have different matrices for different position.

The weight are also parameters of the algorithm and we can use back prop to perform gradient descent to maximize the likelihood of the

training set (to fit the training set very well)

In the algorithms incentive to learn pretty similar word embeddings for like orange and apple because doing so will allow it to fit the training set better by going to see orange juice is Apple juice sometimes a we have only 300 feature vectors to represent all of these words with which fits the training if oranges , and and may be ans which is also a fruit.

so these one of the successful algorithm for learning embedding , for learning the matrix E .

Now let's illustrate other algorithms using a complex sentence as our example.

Other Context and Target Pairs

Let's say in our training set we have a longer sentence :-

I want a glass of orange juice to go along with my cereal

The job of the algorithm is to predict something called juice , in our previous example our context was the last 4 words before the target word. and so if our goal is to

learn the embeddings , researchers of experimented with many different types of context , if we go to build a language model , it's natural for the context to be few words before the target word but if our goal isn't to learn a language model then we can choose other context.

For example :- we can pose a learning algorithm problem where the context is the four words on the left and four words on the right and we can take the first four words on the left and right as a context.

I want a **glass of orange juice to go along with** my cereal.

- So we are posing a learning algorithm where the algorithm is given four words on the left and four words on the right to go along with asked to predict the word in the middle. But how is this work in the case of language model , we want to predict the next word how can we extract words even after the target word?

And posing a learning problem like this ,where you have the embedding of the left four words and the right four words feed in to a neural network to predict the target word in the middle.

We can also use a simpler context may be use the last 1 word , so given the word orange what comes after orange ? so this is a different learning problem where you tell it one word , orange and then also say “what do you think the next word and we can construct a neural network that just feeds the word , the one previous word to a neural network and try to predict the next word”.

And also one thing that surprisingly works is that to take “Near By 1 Word” , which means near by 1 word as a context.

Example :- I want a glass of orange **juice** to go along with cereal.

We might tell , that well the word glass is somewhere close by , so i saw the word glass and there must be another word somewhere close to glass , what do you think that word is ?

We talked about language modeling problem which causes the poses of a machine learning problem where you input the context like the last four words and predict some target word , how posing that problem allows you to learn a good word embeddings.

Next we will even see simpler contexts and simpler learning algorithms to map from context to target word can also allow you to learn a good word embedding.

Word2Vec Algorithm

What does it mean by word2vec algorithm ?

What does the name Word2Vec refers to ?

Why we would I use word2vec algorithm ?

a Word2Vec algorithm which is a simpler and computationally more efficient way to learn this types of embeddings.

Skip Grams :- Let's say you are given this sentence in your training set I want a glass of orange juice to go along with my cereal.

* In the skip-gram model what we're going to do is come up with a few context to target pairs to create our supervised learning problem.

- Rather than having the context be always the last four words , or the last n words , immediately before the target word , what we gonna do is , randomly pick a word to be the context word , let's say we choose the word orange and what we gonna do is

randomly pick another word with in some window , say ± 5 words or ± 10 words from the context word and we choose that to be our target word , so may be by just a chance , we peaked a juice as our target word. That's just one word later , we might choose another pair where the target could be a "glass" or may be just by chance we choose the word "my".

| | <u>Context</u> |
|---------------|----------------|
| <u>Target</u> | |
| | orange |
| juice | orange |
| glass | orange |
| my | |

We set up a supervised learning problem were given the context word , we're asked to predict what is a randomly chosen word in say ± 10 or ± 5 word window. And this is not an easy learning problem because you know with in ± 10 words of the word orange it could be a lot of different words.

Detail Of The Model

Let's say still our vocab size is 10,000 and some have been trained on vocab sizes that exceed a million words. so the basic supervised learning problem we're gonna solve is that we want to learn a mapping from some context 'c' such as the word "orange" to some

target “t” , which might be the word “juice” or the word “glass” or “my”.

context c (“orange”) -----> Target t (“Juice”).

The word orange is in index 6257 from the vocabulary and also the word juice is in the index 4834 from our vocabulary.

- To **represent** the input , such as the word orange , we can start out with some one hot vector for the context word and then



Then in these little neural network that we form we can take these vector e_c and feed it to a soft-max unit and the job of the soft-max unit is to output y^{\wedge} .

To write out this model in detail :- the soft-max model will be :-

soft-max model :- $p(t|c)$, it's the probability of different target words given the context

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

In the nominator where theta is the parameter choice associated with the output t. and the denominator is just a process of finding the target from the vocabulary. I didn't understand this formula.

The loss $(y^{\wedge}, y) =$

Sentiment Classification

- ➔ What is sentiment classification
- ➔ Why do i need sentiment classification
- ➔ How sentiment classification works
- ➔ How to implement sentiment classification

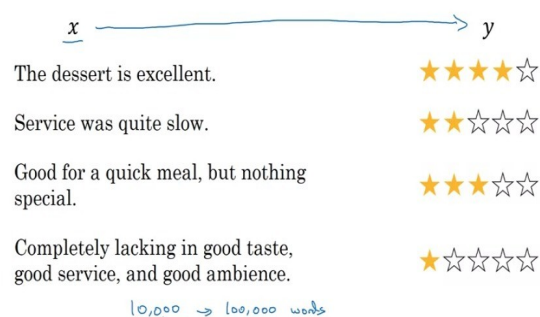
→ How is this it be an application of word embedding ?

The word sentiment means view or opinion. sentiment analysis is a powerful text analysis tool that automatically mines unstructured data (social media , emails , customer feedback and more) for opinion and emotion and then classify it with one of the labels that we provide. or example, you could analyze a tweet to determine whether it is positive or negative, or analyze an email to determine whether it is happy, frustrated, or sad. or if you have a hotel owner , the customers will give you a text based feedback then the algorithm will classify it in to rates , How many stars these review would be ?

It's a task of looking at a pice of text and telling if someone likes or dislikes the things that they are talking about. It's one of the building blocks if NLP.

The input X is a piece of text and the output Y that you want to predict is what is the sentiment, such as the star rating of, let's say, a restaurant review.

Sentiment classification problem



The diagram illustrates the sentiment classification problem. It shows a mapping from input text (x) to sentiment output (y). The input text is represented by four examples, and the output is represented by star ratings. The examples are:

| x | y |
|--|-------|
| The dessert is excellent. | ★★★★☆ |
| Service was quite slow. | ★★☆☆☆ |
| Good for a quick meal, but nothing special. | ★★★☆☆ |
| Completely lacking in good taste, good service, and good ambience. | ★☆☆☆☆ |

Below the examples, there is a note: $10,000 \rightarrow 100,000$ words.

Andrew Ng

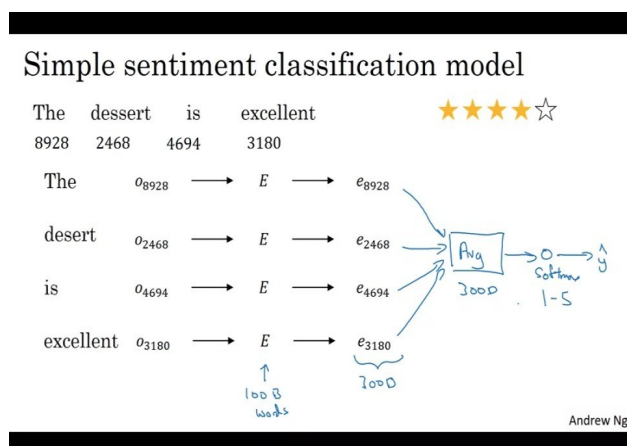
So if we can train a system to map from x to y based on a label dateset like this then you can use it to monitor comments that people are saying about may be a restaurant on a social media , on Twitter or Facebook or Instagram or other forms of social media and if we have a sentiment classifier they can just look a piece of text and figure out the positive or negative is this sentiment of the poster toward our restaurant , then automatically keep track of weather or not there are any problems or if our restaurant is getting better or worse all the time.

One of the challenges of sentiment classification is that we might not have a huge label dateset , so for sentiment classification problems training set work may be anywhere from 10,000 or 100,000 words would not be uncommon , sometimes even smaller than 10,000 words and word embeddings can help us do better on this task , specially we have a small training set. **How does word embedding enable us to do this ?**

Simple Sentiment Classification Model

The dessert is excellent
8928 2468 4695 3180

Now let's build a classifier to map it to the output y , given these four words as usual we can take these four words and look up the one-hot vector.



The embedding E is learned from a large corpus say a billion words (Transfer Learning) then this allows you to take knowledge even from infrequent words and apply them to our problem even words that were not in our labeled training set, and we use that to extract the embedding vector for each word (e_j).

Now one way to build a classifier is to take these vectors and we can just sum or Average their meaning and pass it to the softmax classifier.

The softmax can output what are the probabilities of five possible outcomes (1 star - 5 star). Now by using the average operation here this particular

algorithm works for reviews that are short or long because even the review is a hundred words long, we can just average all hundred words and if that gives you a representation, a 300D representation can be passed in to a softmax classifier (averages the meaning of all the words). **How averaging the words enables us to average the meaning? Is that make sense even, averaging the meaning, even if we are averaging numbers?**

One of the problems with these Algorithms is that it ignores words order. **What do you mean by a word order, do you mean by the context or the meaning sequence of the word passes one after the other?**

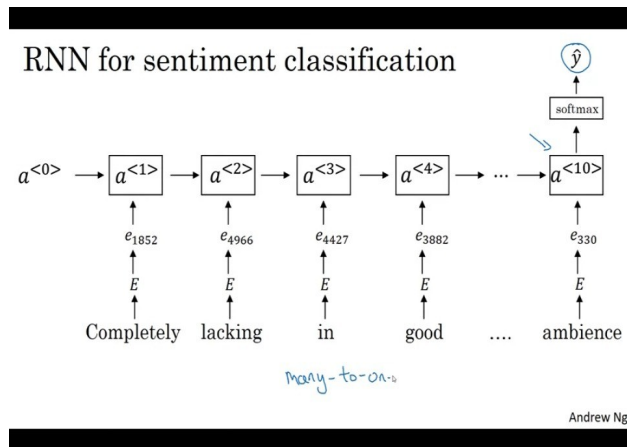
Example :- In particular this is a negative review "completely lacking in a good taste, good service and good ambiance".

The word good appears a lot so if we use an algorithm like this that ignores the word order and just sums or averages the embedding for the different words then we will end up having a lot of representation of Good, so our final feature vector and our classifier will probably think this is a good review even though it's a harsh one star review.

So instead of summing all of our word embeddings we can instead use RNN for sentiment classification.

Recurrent Neural Networks For Sentiment Classification

➔ How does this way takes account word sequence?



This is an example of many to one RNN. So with in algorithm like this it will be much better at taking word sequence (Because recurrent neural networks are models used for a sequential concept)in to account and realize that things like lacking in a good taste as a negative review and not good as a negative review.

Because our word embedding could be trained from a much larger dataset this will do a better job generalizing , may be a new words which is not in our senior labeled training set such as some else says “completely absent of good taste” , instead of completely of lacking , then even if the word absent is not in our label training set but if it was in our 1 billion word corpus used to train the word embeddings , it might still get this right and generalized much better.

De biasing Word Embedding

- ✓ what does it mean by De-biasing word embedding
- ✓ Why do i need to de bias it ?
- ✓ How is this it be an application of word embedding ?
- ✓ How to address bias in word embeddings

So machine learning algorithms are really good at decisions and so we would like to make sure that as much as possible that they are free of undesirable forms of bias such as Gender bias , Ethnicity bias and so on.

so we will see some of the ideas in diminishing /eliminating these forms of bias in the word embeddings.

The problem of bias in word embedding

In word embedding we learned on how analogies like man is to woman where as king is to queen but what if we asked “Man is to computer programmer as women is to _____ ? ” and we found a horrifying result where learned a word embedding might output “Man is to computer programmer as women is to house maker” , this is a gender stereotype. But the output must be “Man is to computer programmer as woman is to computer programmer” . Researchers also found “Father is to Doctor as Mother is to Nurse”.

Word embedding can reflect gender , ethnicity , age , sexual orientation and other biases of the text used to train the model , so every person whether you

come from a wealthy family or lower income family or anywhere in between i think everyone should have creating opportunities. **Why do i need to de bias word embeddings ?** Because machine learning algorithms are being used to make very important decisions influencing everything ranging from college admissions to the way people find jobs to loan applications whether your application for loan gets approved model to in the criminal justice system even sentencing guide lines. So it is important if we try to change learning algorithms to diminish as much as possible or ideally eliminate these types of undesirable biases.

Over many decades humanity has made progress in reducing these types of bias and i think may be for AI i think we actually have better ideas for quickly reducing the bias in Ai than quickly reducing the bias in the Human race.

Addressing Bias In Word Embeddings

→ How to address the bias in word embedding

Let's say we have already learned a word embeddings

Step 1:- Identify the bias direction

For the case of gender what we will do is take the embedding vector of the he and subtract to the embedding vector she.

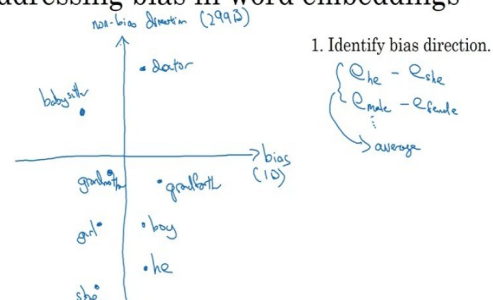
$$e_{he} - e_{she}$$

$$e_{male} - e_{female}$$

Take a few of these and average them , this will allow us to figure out that what

looks like bias and the non bias direction , and in these case think of this bias direction as a 1D subspace where as a non biased direction this will be a 299 dimension subspace. **How does averaging the gender differences enables us to figure out the bias and the non bias direction ?** In our case we just simplified the description a little bit but on the original paper the bias direction can be a bit more than one dimension and rather than taking the average as we are describing here , it actually found using a more complicated algorithm called SVU , singular value algorithm decomposition which is closely related to principal component analysis(PCA).

Addressing bias in word embeddings



[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]

Andrew Ng

2) Neutralize :- For every word that is not definitional project it to the non - bias direction to get rid of the bias.

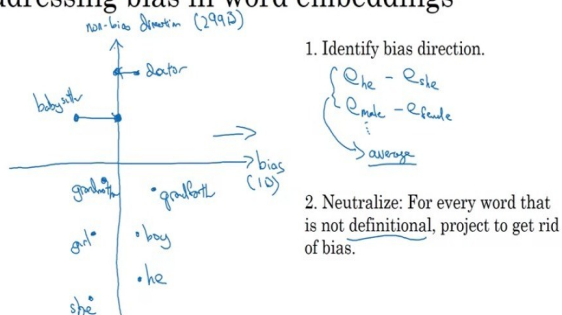
These are some words that are intrinsically(By nature) capture gender words like Grandfather , Grandmother ,girl , boy , she , he and for this gender is intrinsic in the definition where as there are other words like Doctor and baby sitter that we want to be gender

neutral and ethnicity neutral or sexual orientation neutral and so on.

So for every word that is not definitional this basically means not words like grandmother and grandfather which really have a very legitimate gender component because by definition grandmothers are female and grandfathers are male but for the words like doctor and baby sitter we will project the non - bias direction.

so we will project words like doctor and baby sitter in to non bias direction to eliminate the component in the bias direction.

Addressing bias in word embeddings



[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings] Andrew Ng

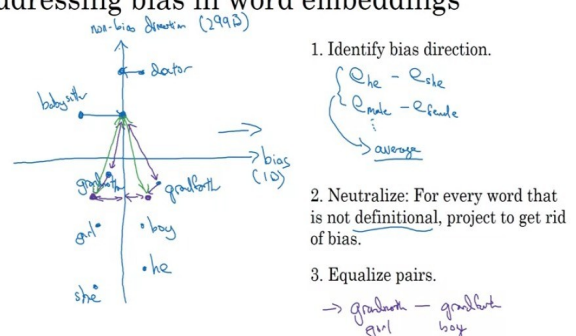
3) Equalization of pairs :- Which we might have pairs of words such as grandmother and grandfather or girl and boy. when we want the only difference in the embedding to be the gender.

In our example the distance or the similarity between baby sitter and grandmother is actually smaller than the distance between the distance of baby sitter and grandfather and these may be reinforces unhealthy or undesirable bias

that grandmothers end up baby sitting than grandfathers.

So in the final equalization step what we would like to do is to make sure that words like grandmother and grandfather are both exactly the same similarity or exactly the same distance from the words that should be gender neutrals such as baby sitter or a doctor(These has a few linear algebra steps).

Addressing bias in word embeddings



[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]

Andrew Ng

In general there are so many pairs of words like grandmother , grandfather , boy , girl , sorority , fraternity , girl hood , boy hood , sister , brother , niece , nephew , daughter , son , so then we might to carry out through this equalization step.

How do we decide what to neutralize

For example :- The word doctor seems like the word we neutralize , make it non gender specific or non ethnicity specific. Where as the word grandmother, grandfather should not be made non gender specific and there are also words like beard that it's fact that men are much more likely to have beers than woman so may be beard should be closer to male than female , so it must not be neutralize.

So how do we decide what words to neutralize, are we gonna classify each of them manually as functional and non-functional? So what the authors did is train a classifier to try to figure out what words are definitional (what words are gender-specific) and what words are non-definitional, so it turns out most words in the English language are not definitional, meaning that gender is not part of the definition. But there are small subsets of words like grandmother and grandfather, boy, girl and so on that should not be neutralized, so sometimes it's quite feasible to hand-pick most of the pairs we want to equalize.

Sequence to sequence models :-

- ✓ **Machine Translation**
- ✓ **Beam Search**
- ✓ **Attention Models**
- ✓ **Speech Recognition**

Machine Translation(sequence to sequence)

Let's say we want to input a French sentence and translate it to English sentence.

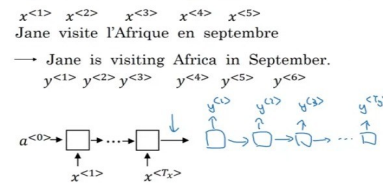
Example:-

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

Jane Visite l'Africa en Septembre
Jane is visiting Africa in September.

How can we train a neural network to input the sequence x and output the sequence y .

Sequence to sequence model



[Sutskever et al., 2014. Sequence to sequence learning with neural networks]

[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation]

Andrew Ng

The Encoder learns a set of features, this could be LSTM or GRU and feed one French word at a time and after ingesting the input sequence the RNN then outputs a vector that represents the input sentence.

The Decoder takes this input from the encoding output by the encoder network and translates it into the equivalent English translation one word at a time.

Why are we not using a kind of architecture like named entity recognition where we input one French word at a time and translate it into an equivalent English word at a time without needing an encoder and decoder? The reason the architecture is changed is because we may not have an equal number of input and output lengths even if it is many-to-many type Recurrent Neural Network.

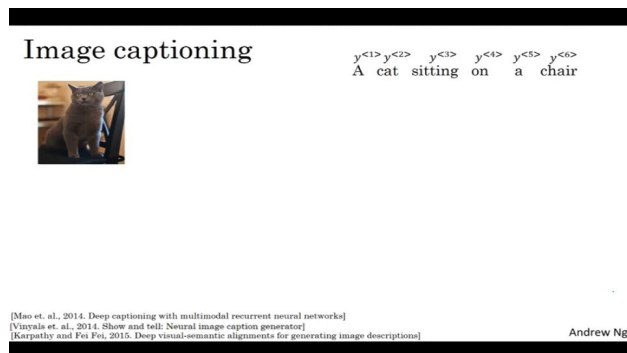
Given enough pairs of French and English sentences, if we train a

model to input a french sentence and output the corresponding english translation , this will actually work decently well because machine translation is a kind of End to End deep learning.

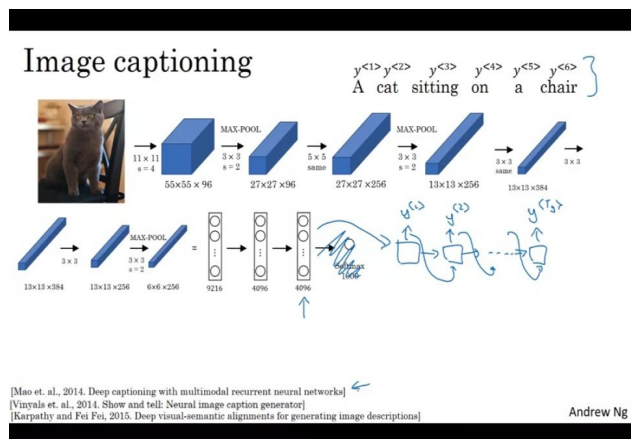
Image Captioning (Image to sequence)

➔ What is image Captioning ?

An architecture which is very similar to machine translation also works for Image Captioning.



How do we train a neural network that input an image and output a caption. Here is what we can do , as we know from computer vision ConvNet , we've seen how we can input an image in to a convolutional neural network may be a pre-trained Alex Net and have that learn an encoding (learn a set of features) of the input image.



We will get rid of the softmax from our Alex Net Architecture and take the encoding vector and feed it in to the decoder Recurrent Neural Network. The pre-trained Alex Net can give us 4096 dimensional feature vector to represent the picture of a cat (This will be our encoder RNN) and the decoder will generate a caption one word at a time.

Question :- How do we feed the encoding vectors to the decoder ?

Picking the most likely sentence

There are some differences between how we run a model like the generate a sequence compared to how we synthesize Novel text using a language model , One key difference is we don't want a randomly chosen translation , we don't want a randomly chosen caption but the most likely caption and the most likely translation.

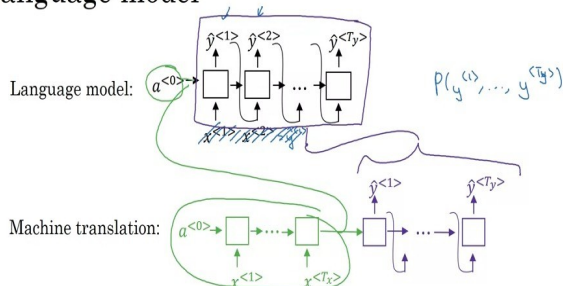
There are some similarities between sequence to sequence machine translation model and the language model.

Machine Translation as building a conditional language

- ➔ How language model and machine translation model has similarity
- ➔ Why are we calling it a machine learning model as a conditional language model

In language modeling this was the network we had built and this model allow us to estimate the probability of a sentence , predicting the next likely sentence and generating a novel sentence.

Machine translation as building a conditional language model



Andrew Ng

As we saw in here the decoder network looks very much similar to the language model that we had above.

So what is the difference ?

The difference is instead of starting with $a_{<0>}$ like the language model but

instead has an encoder network that figures out some representation for the input sentence and it takes that input sentence and starts off the decoding network with the representation of an input sentence instead of representation of all zeros.

So will call this as a conditional language model and instead of modeling the probability of **any sentence** $p(y^{<1>}, y^{<2>}, \dots, y^{<Tx>})$ **the probability of novel sentences**, we will now model the probability of out put english translation conditioned on some input French sentence ($p(y^{<1>}, y^{<2>}, y^{<3>}, \dots, y^{<Ty>} | x^{<1>}, \dots, x^{<Tx>})$) or we want to estimate the probability of english translation , what is the chance that the translation is “Jane is visiting Africa in September” but conditions on the input french sentences.

Finding/Searching the most likely Translation

- ➔ This one is finding not picking
- ➔ How to find the most likely translation
- ➔ what is the problem of language models on generating a random words instead of the most likely words
- ➔ why are we not using a Greedy search

Given the input french sentence “Jane visite l’Afrique en septembre”

:- The model might tell you what is the probability of different corresponding english translations.
 $P(y^{<1>}, \dots, y^{<Ty>} | x)$

Beam Search Algorithm

Refinements to Beam Search

Error Analysis on beam search

Bleu Score

- ➔ What are bleu scores
- ➔ Why do we need them
- ➔ what is Bleu score stands for and what we can understand from the name
- ➔ why are we making human references as our best evaluation metrics

One of the challenges of machine translation is that given a french sentence there could be multiple english translations that are equally good translations for that french sentence.

How do evaluate a machine translation system if there are multiple equally good answers unlike say image recognition where there's one right answer then just measure accuracy but

if there are multiple great answers how do we measure accuracy ?

The way this is done conventionally is using something called bleu score.

Evaluating Machine Translation

Let's say we have a french sentence
French:- Le chat est sur le tapis

:- and we are given a human generated reference for the translation of this which is

Reference 1 :- The cat is on the mat

Reference 2:- There is a cat on the mat

Different humans translate a french sentence. Both of these references are just fine perfectly , now what the bleu score does is given a machine generated translation , it allows you to compute a score that measures how good is that machine translation.

The intuition is so long as the machine generated translation is pretty close to any of the references provided by the humans then it will get high bleu score,

Bleu stands for Bi-lingual evaluation understudy. The intuition behind a bleu score is we are going to look at the machine generated output and see if the types of words it generates appear in at least in one of the human generated references and so these human generated references would be

provided as part of the dev-set or as a part of the test set. **What does this mean ?**

Let's say that machine translation system abbreviating as machine translation as MT.

MT output:- the the the the the the the
This is a terrible translation , so one way to measure how good the machine translation output is to look each of the words in the output and see if it appears in the human references and this will be called Precision. **Why are we calling it precision ? How is the name precision related with our problem.**

precision = out of the machine translation output how many of them are in the human reference ? / how many words are in the machine translation output

- So in our case the machine translation output has seven words and each of the words appeared either in the reference 1 or reference 2 , so the word "the" appeared in both of the references "

precision = 7/7

Do you think this precision measure has a problem ?

This precision enables us to calculate what fraction of the words in the

machine translation output also appear in the human reference and this is not particularly useful measure because it seems to imply that this empty output has a very high precision.

So instead what we are going to use is a modified precision measure , in which we will give each word credit only up to the maximum number of times it appears in the reference sentence.

So in the reference 1 , the word "The" appears twice and in the reference 2 , the word "the" appears just once , so 2 is bigger than 1 and so we gonna say that the word "the" get credited up to twice. so with this the modified precision we will say we can take the word "the"

Modified precision = The maximum happened from the human reference / total number of that word from the machine translation.

modified-precision=

$\text{count_clip}(\text{"The"}) / \text{count}(\text{"the"})$

What does it mean when we say count_clip ? because we clip or take the maximum appearance of the machine translation output word in the either of the human generated references.

The modified precision shows out of the seven words we will give it two

credits for appearing in the human reference

| | | |
|---------------|---|---|
| on the ----- | 1 | 1 |
| the mat ----- | 1 | 1 |

Modified precision = $2/7$

So far we've been looking words in isolated form , but in the bleu score we don't wanna just look for isolated words may be we want to look at pairs of word as well.

Bleu Score On Diagrams

Example :-

Reference 1 :- The cat is on the mat

Reference 2 :- There is a cat on the mat

MT output :- The cat the cat on the mat

Bi-grams just means a pairs of words appearing one after the other. So the machine translation output slightly better output but still not a great translation.

Here the possible Bigrams from machine translation output are :-

"The cat" , "cat the " , "the cat" , "cat on " , "on the " , "the mat" , These are the bigrams in the machine translation output , there is actually a duplicated bigram "the cat" , we can remove one.

Now let's count up how many times each of these bigrams has appeared

| | count | count_clip |
|---------------|-------|------------|
| the cat ----- | 2 | 1 |
| cat the ----- | 1 | 0 |
| cat on ----- | 1 | 1 |

- The count shows how many number of times that diagram appears in the machine translation output

-Where us the count_clip shows the maximum number of times that a diagram is credited by our algorithm in either of the human references.

Modified bigram precision will be the sum of the count clips divided by the total number of biagrams in the machine translation output.

Modified precision = $4/6 = 2/3 //$

so this measuring will allows you to measure the **degree** to which the machine translation output is similar to the human references or may be overlaps with the references.

If the output is exactly the same as either reference 1 or reference 2 , then all of these values $p_1(\text{unigram})$, $p_2(\text{bigram})$,, $p_n(\text{ngram})$ all of them will be equal to 1.0

Bleu Details :- Let's put these together to score this final bleu score.

p_n = Bleu Score of n-grams only and by convention by combining say p_1 , p_2 , p_3 and p_4 and average them using the following formula

Bleu details

p_n = Bleu score on n-grams only

Combined Bleu score:

$$\frac{1}{4} \sum_{n=1}^4 p_n$$

p_1, p_2, p_3, p_4

[Papineni et. al., 2002. Bleu: A method for automatic evaluation of machine translation]

Andrew Ng

Brevity penalty

Bp stands for Brevity penalty, it turns out that if we output very short translation, it is easier to get high precision because probably most of the words of the output appear in the reference, so we don't want translation to be very short so the BP is an adjustment factor that penalizes the machine-translation system that outputs translations that are too short.

The formula for BP is :-

Bleu details

p_n = Bleu score on n-grams only

Combined Bleu score: $BP \times \left(\frac{1}{4} \sum_{n=1}^4 p_n \right)$

BP = brevity penalty

$$BP = \begin{cases} 1 & \text{if } MT_output_length \geq reference_output_length \\ \exp(1 - MT_output_length/reference_output_length) & \text{otherwise} \end{cases}$$

[Papineni et. al., 2002. Bleu: A method for automatic evaluation of machine translation]

Andrew Ng

So $BP = 1$ if the machine translation output are longer than the human generated references otherwise penalizes using the second formula for

a shorter machine translation output. and since brevity penalty is an adjustment factor, if the machine translation output length is longer than the human reference output, the entire bleu's formula will be multiplied by an adjustment factor of 1, which means no penalty but the question is, is there any problem when the machine translation output length greater than the length of the human generated reference?

Bleu Score as a single evaluation metrics

The Bleu Score is a useful single real number evaluation metric which Bleu Score allows us to try out two ideas and see which one has the highest Bleu Score and stick with one that achieved the highest score.

Speech Recognition Doesn't use Bleu Score

Bleu Score is not useful for speech recognition because speech recognition does usually have one ground truth, we just use other measures to see if we've got the transcription exactly word by word correct but for the things like Image Captioning there will be equally good or for machine translation there may be multiple translations which are equally good, the bleu score gives you a way evaluate that automatically.

Attention Model Intuition

- ➔ What are Attention Models
- ➔ Why do i need to study attention models
- ➔ What are the limitations of encoder and decoder network
- ➔ What are some application of the Attention model

We've been using Encoder and decoder architecture for machine translation , one Recurrent Neural network reads the sentence and then the different one outputs a sentence , there is a modification to this called the Attention Model that make all this better. **How does the Attention model makes a better machine translation output ?**

The problem of long sequences

- ➔ How does the Attention model makes us solve the problem of long sentences

So given very much long french sentence like “Jane s’est rendue en Afrique en septembre dernier a appercie la culture et a rencontre beaucoup de gens merveilleux ; elle est revenue en parlant comment son voyage etait merveilleux , et elle me tente d’y aller aussi”

Now what we are asking the encoder Neural Network to do is to read the whole sentence and then memorize the whole sentence and then store it to the

activations and then the decoder neural network then generate the English Translation.

“Jane went to Africa last September and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was and is tempting me to go too”

Now the way human translator would translate the sentence is not to first read the whole french sentence and memorize the whole thing and then regurgitate an english statement from scratch but instead what a human translator would do is read the first part of it may be generate part of the translation and look the second part , generate a few more words , look a few more words , generate a few more words and so on. We kinda work part by part through the sentence because it is just really difficult to memorize the whole long sentence, and so what we see for the encoder and decoder architecture is that it works quite well for short sentences but for every long sentences may be longer than 30 or 40 words , the performance comes down.

Why Encoder and Decoder is not working for longer sentences ?

Attention Model Intuition

Let's illustrate this with a short sentence

- Let's use a bidirectional RNN in order to input compute some set of

features for each of the input words here we drawn the standard bidirectional Recurrent Neural Network with the outputs $y^{<1>}$, ... $y^{<5>}$ but now we are not going to do a word to word translation so we will get rid of the y 's on the Top. Why are we getting rid of the y 's from the top of our bidirectional Recurrent Neural Network ?

But using a Bi Directional Recurrent Neural Network what we have done is for each of the words(Five position words) in the sentence we can compute a very rich set of features about the word in the sentence or may be surrounding words.

We're going to use another Recurrent Neural Network to generate the english translation.

and instead of using "a" to denote the activation , in order to avoid confusion with the activation we are going to use different notations , let's call it "s".

Now the question is when we are trying to generate the first word ("jane") , what part of the french input sentence should we be looking at , It seems that we should be looking at primarily the first word , may be few other words close by but we don't need to be looking way at the end of the sentence. Why , what if the word

in the end of the sentence might tell us some information about our translation ?

So what the attention model would be computing is a set of attention weights and we're going to use $\alpha^{<1,1>}$ to denote to generate the first word how much should we pay attention to the piece of information that we get from the first word $x^{<1>}$ from our feature vector.

And we also come up with a second attention weight , let's call it $\alpha^{(1,2)}$ which tells us while we are trying to compute the first word "Jane" how much attention should we paying to this second word from the input and so on and together this will tell us , what is exactly the context $c[1]$ we should we should be paying attention to and that is input to our first Recurrent Neural Network unit and then try to generate the first word , so this is one step of the RNN.

Question :- How exactly does this context defined ?

- How do we compute this attention weight α ?
- How does the attention model solves the problem of encoder and decoder , in the translation of longer sentences.
- What is the application if "s" ? The second recurrent neural network ?

And the $\alpha_{<t,t'>}$ allows it on every time step to look only may be with in a local window of the french sentence to pay attention to when generating a specific english word.

Formalize Attention Models

Now let's formalize the attention model.

We have an input sentence and we use a bi directional Recurrent Neural Network but we can also use a Bi Directional GRU or a bi directional LSTM to compute features on every word.

And so for the forward recurrence we would have \vec{a} and \overleftarrow{a} for the activation backward recurrence. Technically like $\vec{a}_{<0>}$ in the forward step we have also $\overleftarrow{a}_{<0>}$ for the backward step which is a vector of zeros and at every time step even though we have the features computed from the forward recurrence and from the backward recurrence in the bi directional RNN ($\vec{a}_{<0>}$, $\overleftarrow{a}_{<0>}$) and we are going to use $a_{<t>}$ for both of these concatenated together

$$a_{<t'>} = (\vec{a}_{<t>}, \overleftarrow{a}_{<t'>})$$

We are saying t' to denote that it is a french sentence, and this is going to be a feature vector for the time step t' in both forward and backward direction.

So the way we define the context is actually to sum the features from the different time steps weighted by this attention weights.

So more formally the attention weights will satisfy this

$$\sum_{t'} \alpha_{<t,t'>}$$

The alpha must be non-negative and they are summed to 1 and we will have the context at time step 1 like this :- $C[1] = \sum_{t'} \alpha_{<t,t'>} a_{<t'>}$

The context is going to be the sum over t' all the values of t' of this weighted sum of this activations $a_{<t'>}$

$\alpha_{<t,t'>}$ This term is the attention weights and this term $a_{<t'>} = (\vec{a}_{<t>}, \overleftarrow{a}_{<t'>})$

In other word when we are generating the output word how much should we paying attention to the t' input word.

So using the context vector the above network "s" looks like a standard Recurrent Neural Network with the context vector as output and we can just generate the translation one word at a time.

How do we calculate the attention weights

Now if we want to generate $y^{<t>}$ then $s^{<t-1>}$ was the hidden state from the previous step, that fed in to the $s^{<t>}$ and then $a^{<t'>}$ the feature from time step t' is the other input and the intuition is if we want to decide how much attention to pay to the activation of t' well that should depend on what our own hidden state activation from the previous time step and we don't have the current state activation because the context feeds in to the $s^{<t>}$ and we haven't computed that but look at whatever the hidden state is of this Recurrent Neural Network and then for each of the positions (word positions) look their features.

What is the downside of Attention Models

Now one downside of these algorithm is that, it does take a quadratic cost to run this algorithm, if we have T_x word inputs, T_y words in the output then the total number of these attention parameters is gonna be T_x times T_y , therefore these algorithm runs in a quadratic cost.

Attention Models for Image Captioning

→ **How does the attention models enables us for the image captioning purpose**

These algorithm also applied to other problems as well such as Image Captioning, the task is to look at the

picture and write any caption for that picture.

We could have a very similar architecture, look the picture and pay attention only to part of the picture at a time while you're writing a caption for the picture. **How do we know which part of the image is helpful for giving a caption about that image?** We need to read more research papers on this.

Attention Examples :-

1) Date Normalization Problem:- The problem of inputting date like july 20th 1969 and normalize it in to standard format. 1969 - 07 - 20.

A date like this 23 April 1564 -----> 1564 - 04 - 23

- How does attention model helps us to do this ?
- What is the advantage of solving this problem ?

Audio Data

- ✓ How audio data is formed ?
- ✓ How humans hear to sounds ?
- ✓ How to model speech in to numbers ?
- ✓ why do we need to translate in to a text transcript ?
- ✓ How Attention model helps for speech recognition ?
- ✓ What is the difference between speech recognition and trigger word detection ?

Once upon a time speech recognition systems used to be built using “Phonemes” and this was hand engineered basic units of sound.

Example:- The quick brown fox, be represented as phonemes like “the” has a ‘d’ and ‘a’ sound and quick has ‘k’ and ‘w’, ‘wi’, ‘k’ sound and linguists used to write out these basic units of sound and try to bring language down in to this basic units of sound. Linguists hypothesize that writing down audio in terms of this basic units of sound called “phonemes” would be the best way to do speech recognition. But with the end to end deep learning we are finding that “phonemes” representations are no longer necessary but instead we can build systems that input an audio clip and directly output a transcript without needing to use hand engineered representations.

One of the things that made this possible was going to much larger datasets, so academic datasets on speech recognition might be as 300 hours and in academia 3000 hour dataset of transcribe audio would be considered reasonable size. Time is the way we scale the size of a certain speech.

Attention Model For Speech Recognition

On the horizontal axis we take different time frames of the audio input and then we have an attention model try to output the transcript “The quick brown fox”. One of the method that seems to work well is the use of CTC cost.

Connectionist Temporal Classification (CTC)

Here is the idea, let’s say the audio clip was someone saying “The quick brown fox”, we are going to use a neural network structure like this, with equal number of input x’s and output y’s.

Notice that the number of time steps is very large and in speech recognition usually the number of input time steps is much bigger than the number of output time steps, for example :- if we have 10 second of audio and our features come at a 100hz so 100 samples per second and a 10 second audio clip would end up with a 1000 inputs but our output might not have a 1000 alphabets might not have a 1000 characters, so what do we do?

The CTC cost function allows the recurrent neural network to generate an output like this

t t t _ h _ e e e _ _ _ | _ _ _ | _ _ _ q q q _ _ _

And this is considered as a correct output for the first part of “the quick” and the basic rule for the CTC cost function is Basic Rule :- Collapse repeated characters that is separated by “blank”, so by collapsing repeated characters that is separated by blank, it’s actually collapse the sequence in to “the_q”. This allows the neural networks to have a thousand outputs by repeating characters a lot of times

Trigger Word Detection

so inserting a bunch of blank characters and still end up with a much shorter output text transcript.

so this phrase “The quick brown fox” including spaces actually has 19 characters and if somehow the neural network is forced to output a 1000 characters by allowing the network to insert blank and repeated characters we can still represent this 19 characters output with this 1000 outputs values of y .

Now today building effective or production scale speech recognition system is pretty significant effort and requires a very large dataset.

Let's see on how we can build a trigger words detection system or key word detection system , which is actually much easier and we can do it with a smaller or more reasonable amount of data.

