A Project Report On

Build an Auto-completion System (Text Generation) using Statistical and Neural model approaches

B.Tech 4rth year

IN

ELECTTRONICS AND COMMUNICATION ENGINEERING

Submitted by

MUTYALAPALLI KRANTHI KUMAR - N150596

Under supervision of

Ms. LAVANYA KOVVURU

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE AND TECHNOLOGIES,

NUZVID-521202, KRISHNA (DT.,), ANDHRA PRADESH

DEPARTMENT OF MECHANICAL ENGINEERING

CERTIFICATE

This is to certify that the report entitled “Statistical and Neural model approach of
Language Modeling” submitted by

MUTYALAPALLI KRANTHI KUMAR - N150596

To the department of electronics and communication engineering, RGUKT Nuzvid, for the project report B.Tech E4 in electronics and communication engineering is a bonafide work carried out by them under our personal supervision and guidance during the year 2020-2021.

PROJECT GUIDE	HEAD OF THE DEPARTMENT
LAVANYA KOVVURU	Mr. SK. Irfan Ali
Dept. of ECE	Dept. of ECE
RGUKT-Nuzvid	RGUKT-Nuzvid

TABLE OF CONTENTS

TITLE	PAGE NO.
Abstract	i
1. Language Modeling	1-4
1.1 Intuition behind Language Modeling	
1.2 What is Language Modelling?	
1.3 Application of Language Modelling	
2. Probabilistic Language Model (**Statistical Approach**)	5-12
2.1 Introduction	
2.2 Markov Assumption	
2.3 N-Gram Language Model	
2.4 Evaluation of Language Modelling	
2.4.1 Extrinsic Evaluation	
2.4.2 Intrinsic Evaluation	
2.5 perplexity	
2.6 Project details (**Statistical Approach**)	
3. Challenges in N-Gram Language Modelling	13
4. Neural Language Modelling (**Neural Approach**)	14-19
4.1 Introduction	
4.1.1 Neural Language Model	
4.1.2 LSTM based Language Model	
4.1.3Training Phase	
4.2 Generating Text using Language Model	
5. Project Details (**Neural Approach**)	20
6. Conclusions	21
7. References	21

ABSTRACT

In this project, I try to build a Language Modeling on the English Language. It is basically one of the major applications of NATURAL LANGUAGE PROCESSING technology.
The main aim of this model is to

1. Find out the best-meaning full sentences with their probabilities.
2. Try to predict the upcoming words in an incomplete sentence.

So here, I particularly focus on the 'Predict the upcoming words in an incomplete sentence by considering their long-term dependencies in between the words'. Firstly, I try to build the language model by using statistical approach and find out if there are any drawbacks in that particular approach or not. Then after, I try build go with the Neural approach by using deep learning techniques to predict the required number of upcoming words by considering the long-term dependencies. I decided to use the one of the popular architectures called Long Short-Term Memory Architecture because it is accurately worked on text data.
Finally, build a pipeline with these approaches make a test on the implemented Statistical and Neural Models.

1. Language Modeling

1.1 Intuition behind Language Modeling

We'll understand an interesting application of natural language processing, which is language modelling. So, let's first understand intuitively what language modelling is.
Let's say that we have to predict the next word in the sentence,

This cap is too ____

And here are some of the candidate words that can fill up the space.
[Small, Can, Low, It, Big]
What do you think which of these words are suitable to complete the sentence?
So, after filling all the options in that sentence, we can conclude that the sentence will make more sense when filled with the words small or big. That is, when we get these two sentences,

This cap is too **small** This cap is too **big**

So, this is actually what language model is.

1.2 What is Language Modelling?

Language modelling is the task of assigning probability to a sentence or phrase. And it's generally denoted as P(S), which is the probability of sequence of words.

$$P(S) = P(w_1, w_2, w_3, ..., w_n)$$

And what does this probability mean? So, this probability of sentence or phrase tells us how likely that sentence or phrase is to occur in natural language. Higher the probability of a sentence more likely will it occur in the natural language.
And the task of the language modelling is not limited to just assigning the probability to a sentence or a phrase, we can use language modelling to compute the probability of the coming words

as well. So given a sequence of words, language modelling can be used to predict the next word. And we can denote this as following.

$$P(w_5 \mid w_1, w_2, w_3, w_4)$$

So, what would be the probability of word five or w5, given a set of previous words w1, W2, W3 and W4. This can also be solved using language modelling.

Combining these two tasks, a language model can be defined as a model that computes the probability of a sentence P(S), or the probability of upcoming word Wn, given the set of previous words W1, W2, W3 so on Wn-1.

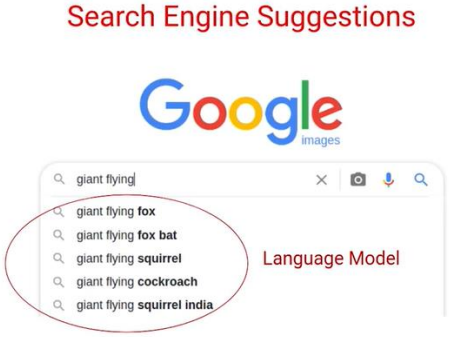
So that's a brief overview of what language modelling and language models are.

1.3 Application of Language Modelling:

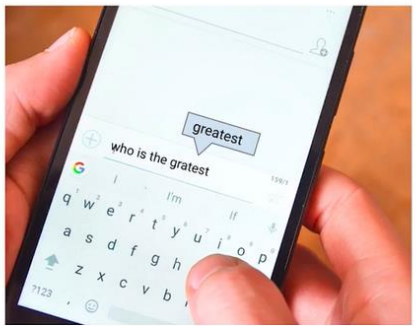
Let's look at some of the applications of language modelling in the field of natural language processing.

1. Search Engine Suggestions are most widely used example of language models.

Let's say that you type a few words in the search bar, 'Giant flying'. And then the search engine automatically gives us a bunch of suggestions. This is done using language models, were given a set of words we suggest the next word.



2. Language models are also used in the Autocorrect features. Let's say someone typed the sentence who is the greatest. And as you would have noticed, the spelling of the greatest word is incorrect.
   So, the autocorrect feature will suggest the correct spelling of the word.



Well, how is language modelling being used here? Let me explain that. So, the language model will calculate the probability of the sentence with incorrect as well as the correct spelling of the word. And it will assign higher probability to the sentence that will have the correct spelling than the sentence which is having incorrect spelling.

P(who is the greatest)

>

P(who is the gratest)

And that's how using language models performance of autocorrect feature can be enhanced.

3. Language models are also widely used in Machine Translation, where the task is to convert a sentence from one language to another. Let me illustrate this with an example. Let's say that we have a machine translation model that translates English sentences to Hindi, we passed a sentence ‘Have a good day’ to the model.
   This is where we can use the language modelling part. So, we'll find the probabilities of both these sentences and our language model will tell us the probability of sentence is more. And considering this output from the language model, the final output of the machine translation model will be

आपका दिन शुभ ___

हो (ho)

है (hai)

P(आपका दिन शुभ हो) > P(आपका दिन शुभ है)

So, we can integrate a language model with the machine translation model and make it better.

4. Language modelling can also be used in speech to text models. The

task of speech to text models is to take an audio input and convert that to text. So, let's say that we have a speech to text model, and we pass an audio sample to it. And as per our speech to text model, these are some of the candidate outputs for the input audio.

- Eye know him
- I no him
- Eye no him
- I know him

In spoken language, all these sentences sound similar. So, this is where we'll use language modelling to find the most suitable output sentence. Language models will assign probabilities to each of these sentences. And then we can find out the first three sentence are fewer probable sentences, as they do not make so much sense.

And once using language models in combination with the speech to text model, we can have better results. So, these are some of the common applications of language modelling. And the applications are not limited.

2. Probabilistic Language Modelling

2.1 Introduction

Till now we look at what language modelling is. And we also looked at different applications of language modelling. Now, we'll understand about the probabilistic language models. to recall the task of language model is to predict or assign probabilities to a sentence or phrase, and it can be used to predict the probability of the next word or the upcoming words, we will use the concept of conditional probability to assign probability to a sentence or predict the next word.
So, this is the formula for conditional probability of A given B.

$$P(A|B) = P(A,B) / P(B)$$

The probability of A given B is equal to the combined probability of A and B divided by the probability of B. Re-structuring the current formula will give us the following. So, the probability of the occurrence of a and b together is equal to the conditional probability of a given b multiplied with the probability of B.

$$P(A,B) = P(A|B) * P(B)$$

Using this formula and chain rule, we can find the probability of occurrence of multiple terms as well. So the probability of occurrence of terms

$$P(X_1,X_2,X_3,...X_n) = P(X_1) * P(X_2|X_1) * P(X_3|X_1,X_2) * ... * P(X_n|X_1,X_2,...,X_{n-1})$$

Now let's understand this with an example. Let's say I have the sentence ‘can you come here’, what will be the probability of the sentence using the chain rule, probability of occurrence of the words ‘can, you, come, here’ together will be equal to

$$P(can,you,come,here) = P(can) * P(you|can) * P(come|can,you) * P(here|can,you,come)$$

This is how we can represent the probability of any sentence using the conditional probability and the chain rule.
Now we have multiple terms here. So, let's see how can we calculate these terms individually. For now, let's see how can we calculate the conditional probability of the word ‘here’ Given the words ‘can, you, come’ this will be equal to the count of times you can come here occurred in the corpus divided by the count of times ‘Can you come’ occurred in the corpus.

$$P(here|can,you,come) = \text{Count of (“can you come here”)} / \text{Count of (“can you come”)}$$

But this approach has a challenge. What if there is no sentence, ‘Can you come here’ in my corpus. In that case, the count of occurrence of ‘can you come here’ will be zero, which means that the probability of this term, which is the conditional probability of ‘here’, given the word ‘can, you, come’, will also be zero.

What if there is no sentence "can you come here"

- Count of ("can you come here) = 0
- then P(here | can, you, come) = 0

And hence, this will assign the zero probability to the sentence. And the chance of getting zero probability for a sentence gets higher and higher as the length of the sentence increases.
So, what is the solution here? Currently, if we have to calculate the probability of word five, given the first four words, it's equal to the count of occurrence of these five words together, divided by the count of occurrence of the first four words in my corpus.

$$P(w5 \mid w1, w2, w3, w4) = \frac{\text{Count of } (w1, w2, w3, w4, w5)}{\text{Count of } (w1, w2, w3, w4)}$$

And there is a chance that these five words will not occur in my corpus. So, we apply the Markov assumption to these probabilistic language models.

2.2 Markov Assumption:

The Markov assumption says that the probability of let's say word five, given the first four words, will be equal to the probability of word five, given just the last two words, which is the word 3 and 4.

$$P(w5 \mid w1, w2, w3, w4) \approx P(w5 \mid w3, w4)$$

A simpler assumption that it will be equal to the probability of word five, given just the last word, which is the fourth word.

$$P(w5 \mid w1, w2, w3, w4) \approx P(w5 \mid w4)$$

So basically, it says that instead of looking at the entire context, let's just look at the past few words, which will be my most relevant context for that particular word. So for our current example, the probability of the word 'here', given the words 'can, you, come' Will nearly be equal to the probability of the word 'here', given the last word which is 'come'.

$$P(here \mid can, you, come) \approx P(here \mid come)$$

This will be equal to the count of times 'come here' occurred together, divided by the count of times 'come' occurred in my corpus,

$$P(\text{here} \mid \text{can, you, come}) \approx P(\text{here} \mid \text{come}) = \frac{\text{Count of (come here)}}{\text{Count of (come)}}$$

or if you want to take the last two words as the context, the probability of the word 'here', the words can you come will be equal to the probability of the word here, given just the past two words, which are, 'you, come' and this will be equal to the count of times 'You, come, here', divided by the count of times 'you, come' appeared in the corpus.

$$P(\text{here} \mid \text{can, you, come}) \approx P(\text{here} \mid \text{you, come}) = \frac{\text{Count of (you come here)}}{\text{Count of (you come)}}$$

So, this is how using Markov's assumption, we can simplify our formula to calculate the probability of a sentence.
Depending on the number of words that we want to take as our context. We have engram language models.

2.3 N-Gram Language Model:

As the name suggests, n-gram is a sequence of n tokens. Let's understand these n grams with an example.
Let's say that we have the same example. 'Can you come here'. So, when n is equal to one, we get unigrams. And as you can see here, each token is a separate unigram.

Unigrams:
N = 1, ["can", "you", "come", "here"]

So, the number of uni grames in a sentence is equal to the number of tokens in that sentence. And if you want to calculate the probability of a word given a set of previous words, in this case, the unigrams will be equal to the probability of that particular word itself.

$$P(w5 \mid w1, w2, w3, w4) \approx P(w5)$$

So, we are not considering any context in case of a unigram.
When n=2, we get Bigrams. A Bigrams is a sequence of two tokens. And as you can see here, the sentence can you come here has been converted into Bigrams,

Bigrams:
N = 2, ["can you", "you come", "come here"]

And as you might have guessed, the number of Bigrams in a sentence will be one less than the number of tokens in that sentence. So, in this case, the probability of a word given a set of previous words will be equal to the probability of that particular word given just the previous word.

$$P(w5 \mid w1, w2, w3, w4) \approx P(w5 \mid w4)$$

And similarly, a Trigram is when we take three tokens together, and for this example, the trigrams are,

Trigrams:
N = 3, ["can you come", "you come here"]

So now the number of trigrams in a sentence will be two less than the number of tokens in the sentence, and in the case of trigrams, while calculating the probability of a particular word, we take the last two words as the context. As you can see here, the probability of word five, given the previous four words is equal to the probability of word five, given the last two words, which are word three and word four.

$$P(w5 \mid w1, w2, w3, w4) \approx P(w5 \mid w3, w4)$$

So, we can pick different values of n in an n-gram model, which will give us different language models.

2.4 Evaluation of Language Modelling:

Till now we have understood what language modelling is and how we can make probabilistic language models.
Now we will look at the different evaluation metrics that can be used to evaluate the performance of a language model. Now what are the different aspects, on which we can evaluate a language model. So, we can look at,

- Does the language model prefer good sentences over bad ones?
- Does it assign higher probability to the real or frequently observed sentences?
- Does it assign lower probability to grammatically incorrect or less observed sentences?

So, these are some of the questions, which you can ask about your language model and decide whether it is a good language model or a bad one.

We have different ways to evaluate a language model, and the first one is

2.4.1 Extrinsic Evaluation:

- As the name suggests, in extrinsic evaluation, we use some external sources to evaluate our model.
- We train our language model and use it for some different tasks like machine translation or speech recognition or spelling correction for example. Now depending on how well these models will perform with and without a language model, we can decide whether our language model is good enough or not.
- So, this is how in extrinsic evaluation. We use some external source to evaluate the performance of a language model.
- But there is a challenge with this extrinsic approach. This approach is time consuming, and it can take days or even months to evaluate a single language model, since we're using this model on a different task.

And hence we move to the Intrinsic Evaluation.

2.4.2 Intrinsic Evaluation:

- The intrinsic evaluation actually measures how well a model is, or how well we are modelling natural language.
- And since we are not using any external source here, it gives us quick results.
- One of the commonly used evaluation metric for such intrinsic evaluation is perplexity evaluation.

2.5 Perplexity:

Just to recall the best language model will be the one which assigns highest probability to the test set. Let's say that we have these sentences in our test set.

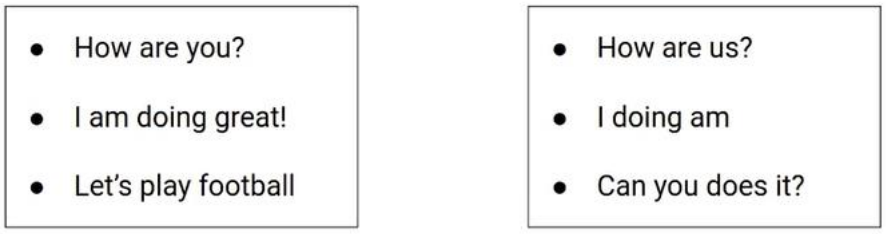
{How are you? I am doing great, Let's play football}.

Our language model should give higher probability to such sentences, since they are correct, and our common sentences.

The model is assigning higher probability, it means that the model is less confused about these sentences.

Perplexity means confusion. Since the model will less confused for these sentences, hence the perplexity is low.

Similarly, if you have below type sentences,



- High Probability
- Low Perplexity

- Low Probability
- High Perplexity

Formally say that 'perplexity is the inverse probability, on the test'. And here is the formula for that.

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

Note that we took the Nth root of inverse of probability. The intuition behind is that, the dataset can have varying number of sentences and those sentences can have varying number of words. So, to normalise the effect we normalise the probability of the test set by the total number of words, which is n.
This gives us per word measure of perplexity.
And as you might have already guessed, lower the perplexity, better would be the model. So that's all about the valuation metrics for language models, it's now time to apply all the learnings that we have gathered in this module so far on a real-world data set. We will now take a project where we will build a model that can predict the next word given some set of Words.

2.6 Project Details

The project is the next word recommendation, using language models. Here's the problem statement. "We want to build a next word recommender system, which will take a sequence of words as input and predict the next relevant word."
Let's say that the input words are, "How are". So, the task of the model will be to predict the next relevant word that could be, "How are you".

About the Dataset

- Taskmaster Dataset, published in 2019
- 64,777 conversational dialogue
- Six domains:
  - ordering pizza,
  - creating auto repair appointments,
  - setting up ride service,
  - ordering movie tickets,
  - ordering coffee drinks and
  - making restaurant reservations

Steps to build the next word recommender system

1. Loading and exploring the dataset
2. Creating N-grams of the dialogue
3. Building the N-gram Language Model
4. Predicting the next word using N-gram Language Model

3 Challenges in N-Gram Language Modelling

So far, we have seen how to perform language modelling, using heuristic methods. Let's look at how we can perform the same using deep learning. More specifically, using neural language models. But why do we even need a neural language model in the first place?
Well, there are certain challenges with The Statistical language models or the N-gram Language models. And that's why neural language models are preferable.

- Let's say we have a Trigram language model. And we want to find the probability of the sentence ‘Jag build that house’, using the chain rule, we will find the probability of the sentence.

N = 3 (Tri-gram model)

P("Jack built that house") =

P("Jack"). P("built"|"Jack"). P("that"|"Jack built"). P("house"|"built that")

Consider the first term 'Jack', probability of 'Jack', in this chain rule will be misleading.
Let me explain it to you with the help of an example,
Suppose that in our document, The word Jack appears 50 times, and the total number of words in the document are 500.
So, the probability of 'Jack' would be 0.1. In this case, 'Jack' can appear anywhere in the document. For example, it can appear in the middle of a sentence, or at the end of a sentence, it can even appear at the beginning of some sentences.
Now consider this scenario: Over here we are considering only those appearances of the term 'Jack' where it is at the beginning of the sentence.
And you can see that there are 10 such cases. So, the probability of Jack, now would be 0.02. And this probability is more appropriate in this case. So, this is an issue with the N-gram language model that one should know about.

- Moving on, imagine a different scenario, where we would use an N-gram language model to generate a sequence of texts, given an input text "Jack build that house".

So, let's say the model generates three words or three tokens after the input text, and the three words are 'in', 'two', 'years'.
Alright, it further generates another word 'in' after the term 'year', and another term that is 'India'. And one more term 'is'.

Input = "Jack built that house"

Model output = "Jack built that house in two years in India is..."

So, the problem here is, the N-gram language models do not know when to stop generating the text. Alright, so unless you specify how many words you want to generate the model will keep on generating.

- Moving on, the third problem with the N-gram language model is that, we cannot use a very long context, because then the computational overhead would become too expensive. That means the bigger the context, the more competition power it will

need.

And there is one more problem, which is known as ‘Sparsity Problem’, where the model will give zero probability, whenever it encounters an unseen world in a new data set.

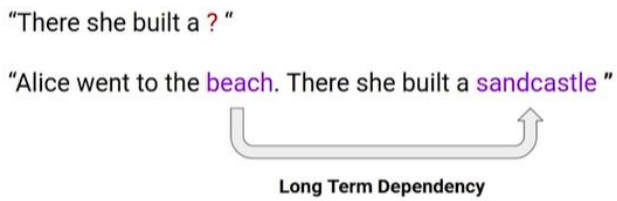
For example, if the training set for an N-gram language model contains these phrases, ‘find the differences’, ‘find the station’, ‘find the jacket’. And when the model comes across the phrase ‘find the cat’ in a new data set, then it will give zero to the probability of ‘cat’ given the previous two tokens, as ‘find’ and ‘the’.So, if this expression is part of a probability chain of a sentence, then the entire probability for that sentence would become zero. And that's not all.

There is one more issue with N-gram language modelling. Consider this thing complete sentence:

“There she built a ___”. Now, can you guess the next word. Let me give you some more context. I have added this extra sentence,

“Alice went to the beach, period. There she built a ____”. Now guessing the next term becomes slightly easier.

The correct next term is “sandcastle”. And it has a strong dependency with a term “beach”. Now, this is known as Long Term Dependency, or Unbounded Dependency.



An N-gram model cannot capture such dependencies, because they consider only the adjacent previous words as the context, to predict the next word.

Now this is where deep learning shines, it allows us to build a model that is able to handle the issues mentioned above so far,

whether it is related to dealing with unseen words in the test data set or understanding the context behind certain words or capturing the unbounded dependencies.

Neural language models, especially those that use RNN or LSTM minimise this issue, and provide a better alternative for language modelling.

4. Neural Language Modelling

4.1 Introduction to Neural Language Modelling

Now let’s learn, how neural language modelling can help overcome those challenges, and we will also understand how a neural language model works.

4.1.1 Neural Language Model:

It has been empirically proven that neural language models outperform all the previous language modelling techniques and approaches, especially the N-gram language model. These language models generalize better on unseen, or new data, because they are able to capture the semantic information in the text data, unlike the statistical language models, and hence these models are able to solve the sparsity problem, to a large extent. We can make neural language models with the help of

1.feed forward neural network

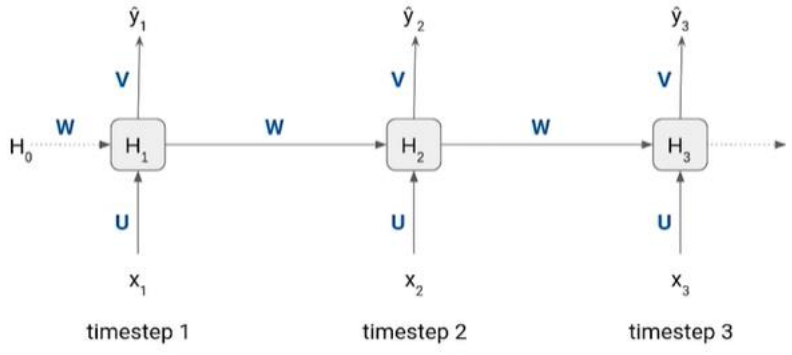
2. NN Based network

3.RNN/ LSTM base models usually perform better, because of their tendency to take into account the information from previous time steps.

So, we will focus LSTM based language models only.

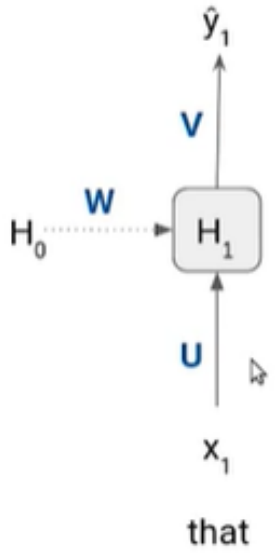
4.1.2 LSTM based Language Model:

Let's consider a minimalistic diagram of an LSTM Model:

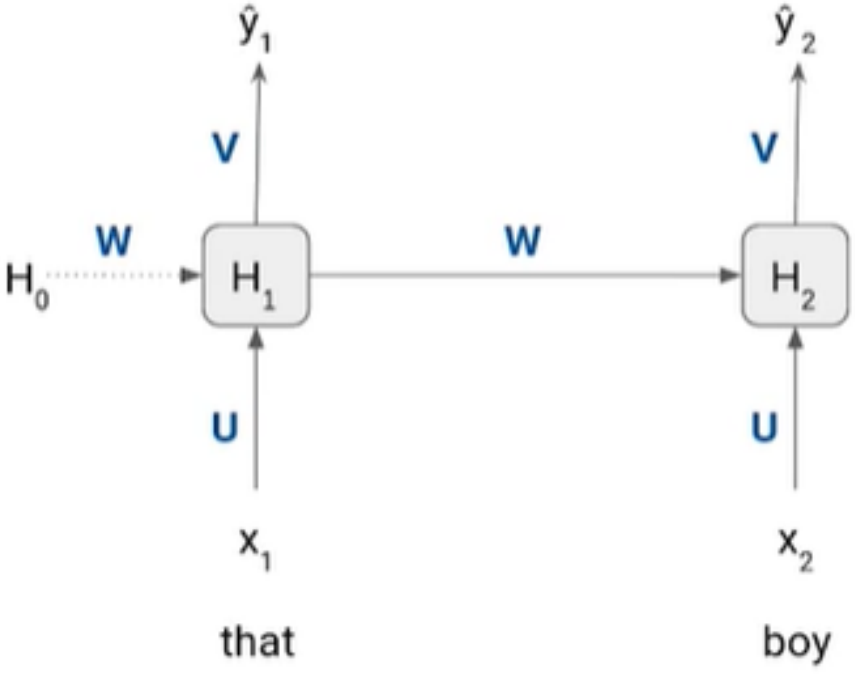


x1, x2 and x3 are the three different inputs passed to the model at three different time steps, and y1hat, y2hat and y3hat are the outputs of this model. So, it's a many to many architectures right. Multiple inputs, and multiple outputs. W, U and V are the weight matrices of this network

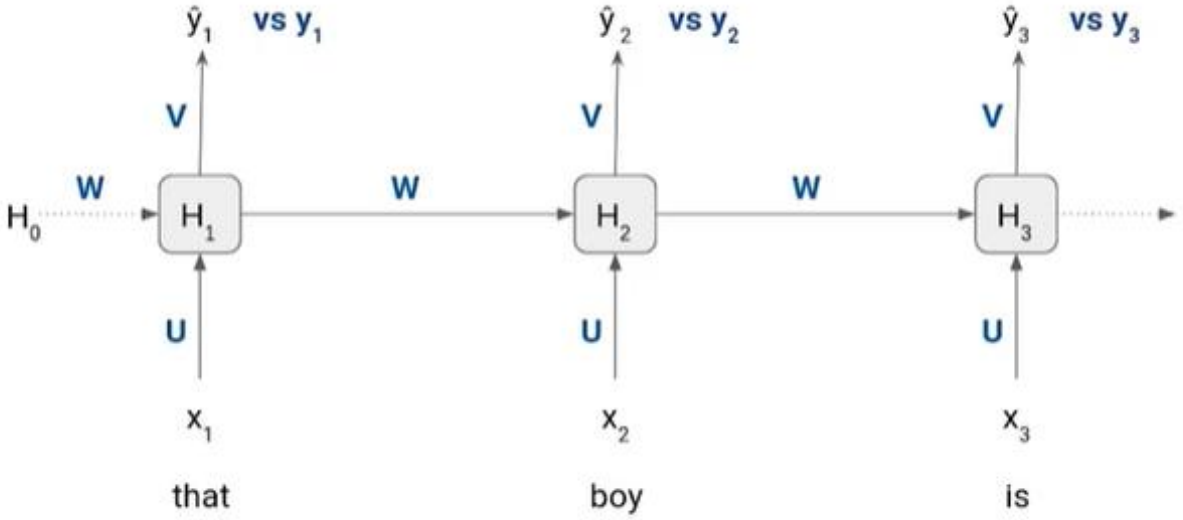
Let's understand how we can use this architecture to perform language modelling.
So first of all, we will see what happens during the training phase, when we train this model, you pass the first token of the input sequence to the first-time step of the model.



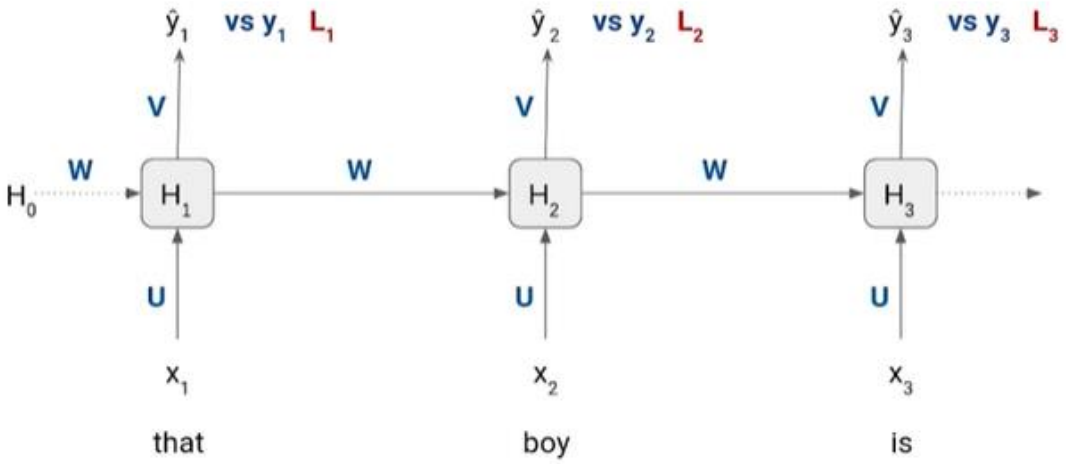
This x1 is the word embedding of the token, that along with x1, Ho is also passed to this time step, the output at this time step, is y1hat, and H1 is the hidden state computed at this
time step.
In the time step to the second token of the input sequence will be passed to the network, along with H1.



Which will give the output, y2hat and hidden state H2 .We have passed two input tokens to the model, and we have got two outputs, y1hat and y2hat.
Finally, we will pass the third token of the input sequence. In the third time step, and we will get the third output that is y3hat.



Now, these outputs y1hat, y2hat and y3hat will be compared with the actual values in the data set.



That is y1, y2 and y3. And based on that, the losses will be calculated for each of these time steps, and the final loss for this input sequence is going to be the average of L1, L2 and L3. And based on that loss, the weight matrices of this network will be updated.

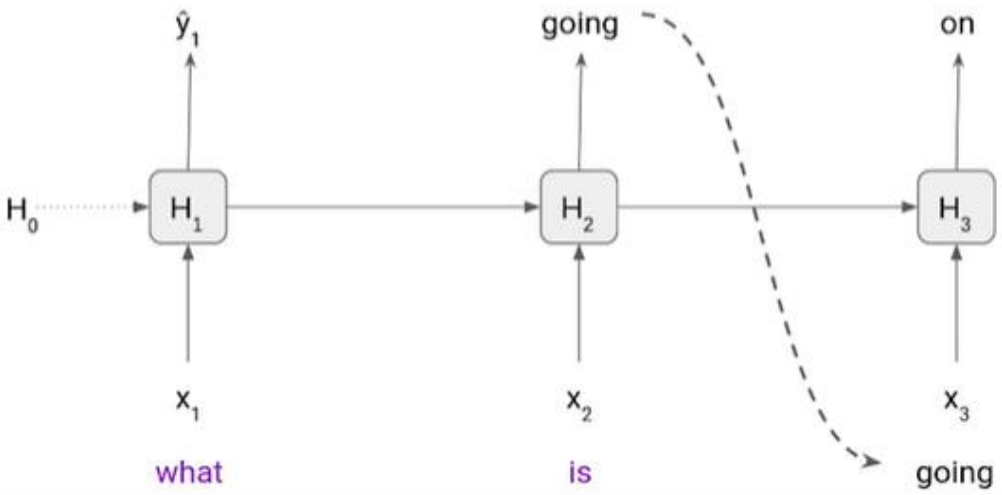
4.1.3Training Phase:

The output generated at each time step is passed to a fully connected layer, and a SoftMax activation function is applied on that,
Then the output will be a probability distribution of all the words in the vocabulary at each time step, the vocabulary is nothing but the collection of unique words or distinct words in the training set.

The loss function at each time step is the cross entropy between the model output and the actual values. And the final loss is going to be the average of all the losses at each time step.

4.2 Generating Text using Language Model:

Now once the model is trained. Let's see how we can use this model to generate some text.
Suppose we pass this input text string “what is” to the model, and we want the model to generate the next two words.
So, the first token of our input string that is ‘what’ will be passed to the first-time step, along with Ho and the model will give y 1hat as output.
This output is a probability distribution, in which the highest score will be given to the token or word. That should be the next token, after ‘what’ according to the model, but we won't consider this output right, because we already know what is the second word. And it is ‘is’.
So, the second word will be fast to the model at time step two, along with hidden state H1.
Now the model will give the output for the next word after the input string “what is”.
Let's say the output score at time step two, corresponds to the word “going”, but we need one more word, right.
As you can see, we have run out of the input text string. So, what should be the input at the third time step? Well, we will pass the output of the second time step, as the input for the third time step. So, we will pass, going as the input to the third time step, along with the hidden state H2 and it will give us the next word after “what is going”.



So, the generated text by the model is “going on”, and the complete sequences is “What is going on”.
So, this is how an LSTM based model performs text generation, and whatever the text the model generates depends on the data, it has been trained on.
For example, it could have generated “What is your name”, or “what is your native place” instead of “what is going on”.

Benefits – Neural Language Model

Just to summarise, these are some important benefits of using a neural language model:
They can represent Unbonded Dependencies, unlike N-gram language models.
The number of parameters does not grow with the length of the dependencies captured, and if you try to do the same with the N-gram language models, the computational costs will increase exponentially.
Now we know, what are the benefits of using neural language models and how our typical LSTM base language model works, both during the training phase, and while generating that text.

5. Project: Build an Auto-completion System (Text Generation) using Neural Language Model

Till now we covered Neural Language modelling in detail, such as, what are the benefits of neural language modelling, how our neural language model is trained and how it performs text generation.
Now we will train a neural language model of our own to perform text generation. But before that, let's discuss the problem statement of the project.

5.1PROBLEM STATEMENT:

The problem statement for this project is that "We have to develop our text generation system that should take in a seed text or an input text string from the user and generate a sequence of text, the number of tokens to generate will also be specified by the user."

For example,

Seed text = "how are", number of words to generate = 2

Output = "how are you doing"

5.2 About the Dataset:

The data set that we will be working with for this project,

- Taskmaster Dataset, published in 2019
- 64,777 conversational dialogs
- Six domains:
  - ordering pizza,
  - creating auto repair appointments,
  - setting up ride service,
  - ordering movie tickets,
  - ordering coffee drinks and
  - making restaurant reservations

Basically, this data set is a list of sentences or list of sequences.

4.3 Data Preparation:

It is quite crucial, task in this project. Let's say we have a data set, and there are only three sentences or sequences.

```
[ 'alright that is perfect',
  'sounds great',
  'what is the price difference' ]
```

The first sentence has 4 tokens, the second sentence has only two tokens, and the third sentence has 5 tokens. We can see that all these sequences have different lengths, and they also have different number of tokens, and we know that if we are using an LSTM model then all the sequences should have equal length.

So, we will have to make these sequences have equal length. For that, we will split the sequences to the length of three tokens each.

```
[ 'alright that is',
  'that is perfect',
  'sounds great',
  'what is the',
  'is the price',
  'the price difference' ]
```

The second sequence 'sounds great', is already less than three tokens in length, so it cannot be split further.

Now all these sequences have same length, except 'sounds great'. Its length is 2. So still, our sequences do not have same length. Now what should we do? we can use padding over here

We have to padded the shorter sequence of 'sounds great', with the <pad> token.

```
[ 'alright that is',
  'that is perfect',
  'sounds great <pad>',
  'what is the',
  'is the price',
  'the price difference' ]
```

Now all the sequences have same length, and that is three. So over here in this case, I have considered the sequence length of three, but you can try different lengths as well.

The next step is to convert these text sequences into integer sequences. For that, we will first have to prepare the vocabulary from our data set.

A vocabulary is nothing but a collection of unique words or distinct

words in the text data set and next, we will assign a unique integer to each of these tokens, so our token to end user mapping is ready and we can use this mapping to convert our text sequences into integer sequences.

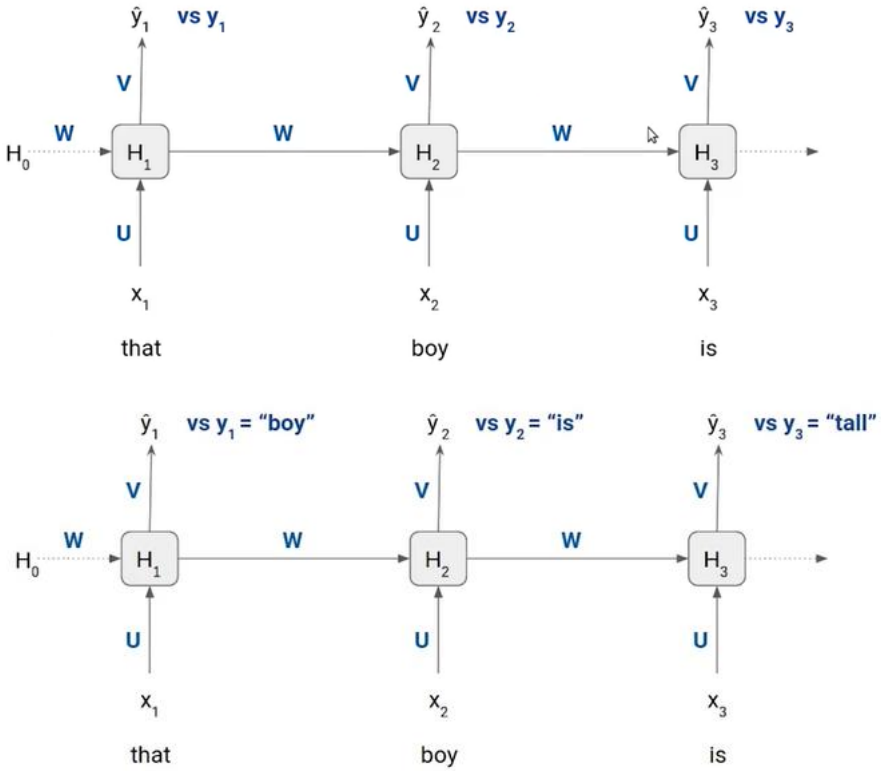
"<pad>"	- 0
"alright"	- 1
"difference"	- 2
"great"	- 3
"is"	- 4
"perfect"	- 5
"price"	- 6
"sounds"	- 7
"that"	- 8
"the"	- 9

```
[ [1, 8, 4],
  [8, 4, 5],
  [7, 3, 0],
  [10 4 9],
  [4 9 6],
  [9 6 2 ] ]
```

Now we can use these integer sequences as training data for our model.

5.4 LSTM based Language Model:

Let's have a look at the LSTM architecture during the training phase,

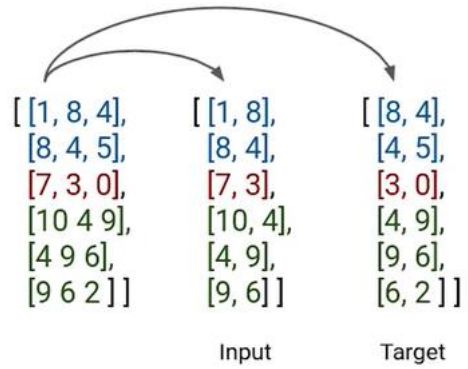


Here, you can see the target sequence is shifted towards left by one token. And this is how we have to prepare the input sequences, and the target sequences.

Now let's see how it can be done. Since our task is to predict the next word. We can create input and target sequences from the integer sequences.

We can consider the sequence from the first element, till the second

last element as the input sequence and the sequence from the second element, till the last element as the target sequence.



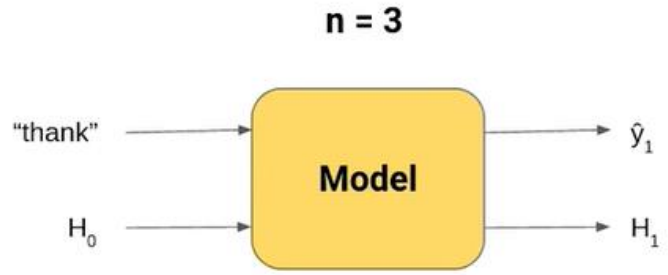
Over here, all that we have done is, shift the initial integer sequences towards either side by one token. So, these are the techniques that we will use while implementing this project.

5.5 Text Generation:

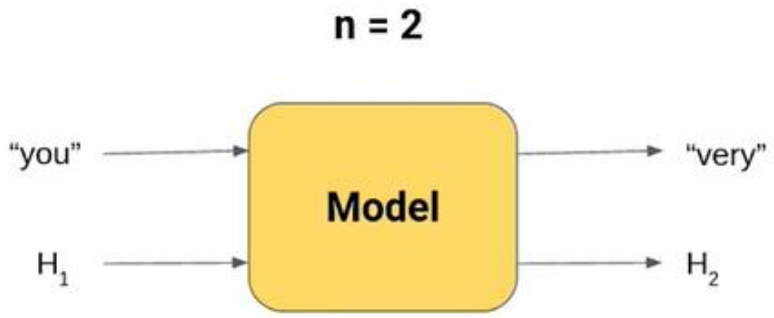
Now, coming on to the Text Generation Part:

- **User input**:
  - A seed text
  - number of tokens to generate
- **Output**: seed text + model generated tokens
  - Input seed text = “thank you”, n = 3
  - Output = “thank you very much sir”

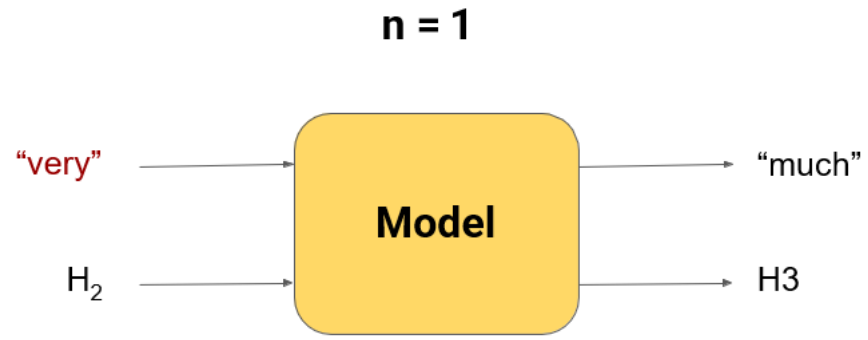
Let's try to understand this generation process graphically,
The seed text is ‘thank you’, and the number of tokens to be generated is three, at time step one, the first token of the seed text, that is ‘thank’ will be passed to the model with the initial hidden state [Ho] that is initialised by zeros. Model will give two outputs y-one-hat and H1 that is hidden state one.



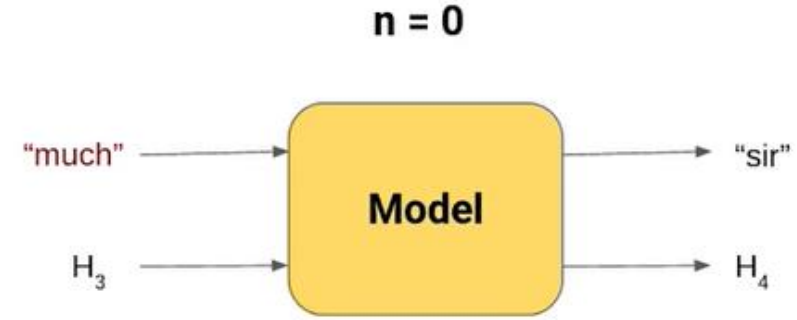
y-one-hat represents the next token as per the model, but we won't use this output, because we already have the next token in the seed text ‘you’.
At time step two, we will pass the second token ‘you’ to the model along with H1.



Now in this case we will use the output of the model, and it is the word very, so this is the first word generated by the model. The model also gives out hidden state two [H2} that we will use in the next time step.
Here in time step three, we will use the word generated in the previous time step, as the input. Along with that we will also pass H2 to the model, and the model generates the second word, which is ‘much’.



Again, in this timestamp. We will pass the word generated in the previous time step, and H3 as input to the model and the model will generate the final required token.



So now that we know what is the process behind training a language model. And while using that train model for text generation, we can now proceed to implement this project in PyTorch.

6. Conclusions

Build the language model by using probabilistic techniques and successfully predict the upcoming word by giving two previous words as inputs. Point out the drawbacks by using this probabilistic approach like Sparsity problems, Long-term dependencies and also trade of between computational resources and performance with respect to the N-grams used in the probabilistic language model. Then, to overcome these drawbacks build a Neural Language model, particularly with the LSTM architecture. Resolve all the issues and success fully predict required number of upcoming words by giving N-words as inputs.
Benefits we get by using Neural Language Model are,
1. They can represent Unbonded Dependencies, unlike N-gram language models.
2. The number of parameters does not grow with the length of the dependencies captured, and if you try to do the same with the N-gram language models, the computational costs will increase exponentially.

7. References

- Probability: https://www.khanacademy.org/math/statistics-probability/probability-library/conditional-probability-independence/v/calculating-conditional-probability

- LM with DL: https://medium.com/@shivambansal36/language-modelling-text-generation-using-lstms-deep-learning-for-nlp-ed36b224b275

- LM with N-grams: https://medium.com/analytics-vidhya/n-gram-language-models-9021b4a3b6b

- Tokenization: https://medium.com/bware-labs/tokenomics-explained-74d9bb8ff026

- Neural Networks: https://medium.com/@johnolafenwa/introduction-to-neural-networks-ca7eab1d27d7

- LSTM: https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

- RNN: https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf