Workshop on Natural Language Processing (Day 2)

Ramaseshan Ramachandran

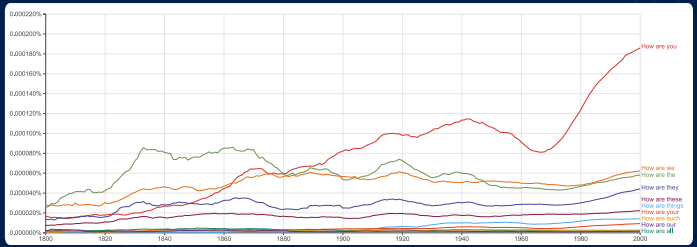
Introduction to Language Models

INTRODUCTION

How are ____? Can you guess the missing word?

INTRODUCTION

How are ____? Can you guess the missing word?



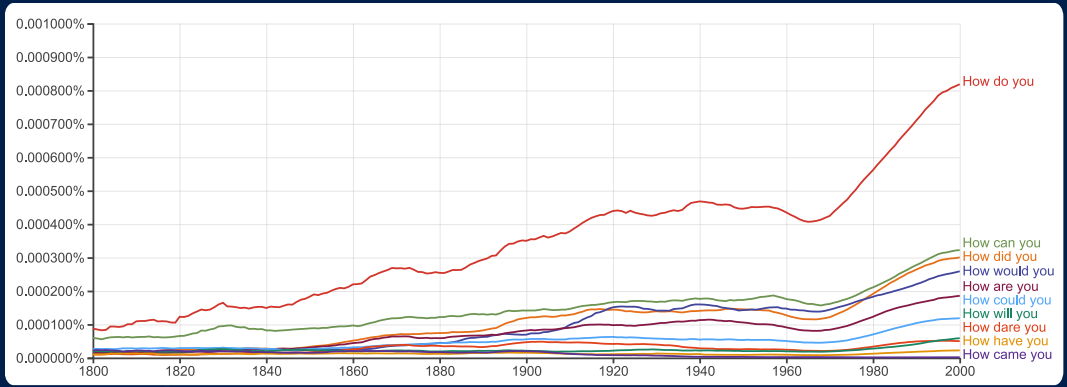
Source:Google NGram Viewer

INTRODUCTION

How ____ you? Can you guess the missing word?

INTRODUCTION

How ____ you? Can you guess the missing word?



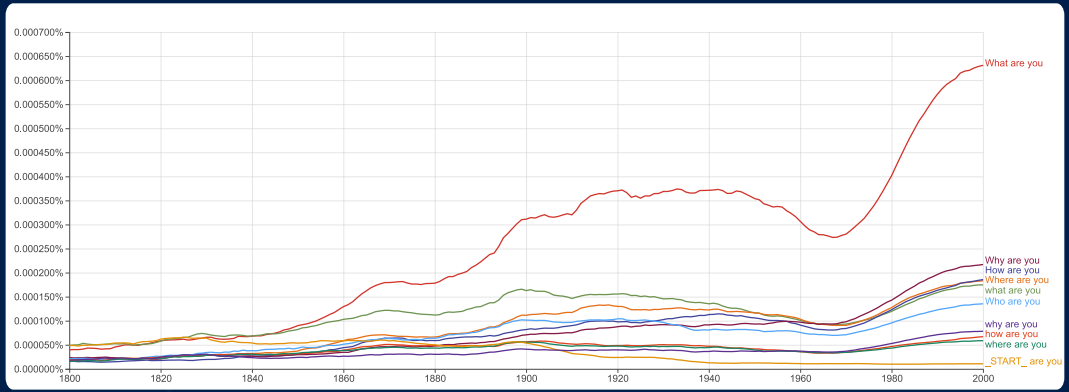
Source:Google NGram Viewer

INTRODUCTION

____ are you?

INTRODUCTION

____ are you?



Source:Google NGram Viewer

HOW ABOUT CLOZE TEST?

I am ___ ___ ___ ___. Nevertheless, I am ___ ___ ___ ___

HOW ABOUT CLOZE TEST?

I am __ __ __ __. Nevertheless, I am __ __ __ __

I am tired. Nevertheless, I am moving forward

I am uncertain. Nevertheless, I am hopeful

I am nervous. Nevertheless, I am prepared

I am afraid. Nevertheless, I am ready to take the leap

I am struggling. Nevertheless, I am not giving up

I am struggling. Nevertheless, I am determined

I am overwhelmed by the workload. Nevertheless, I am motivated to achieve my goals

INTRODUCTION

How do humans predict the next word?

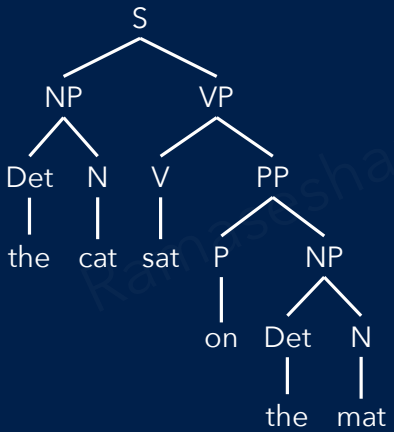
- Domain knowledge
- Syntactic knowledge
- Lexical knowledge
- Knowledge about the sentence structure
- Some words are hard to find. Why?
- Natural language is not deterministic in general
- Some sentences are familiar or had been heard/seen/used several times
- They are more likely to happen than others, hence we could guess

LEXICAL KNOWLEDGE

Lexical knowledge encompasses the nature of language, specifically the words and their meanings. Words are acquired through associative learning, such as colocations and expressions, without explicitly comprehending their underlying constituents.

SYNTACTIC RULES

We analyze the sentence structure to understand the grammatical role needed for the missing word



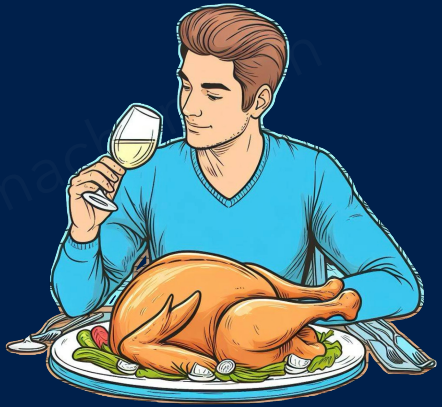
SEMANTIC CUES

We use our understanding of word meaning and context to infer the most likely word that fits the overall meaning of the sentence

WORLD KNOWLEDGE

We draw on our knowledge of the world to make educated guesses about the missing word.

PRAGMATIC CONSIDERATIONS



Chicken is ready to eat

Probabilitic Language Models

IDEAL LANGUAGE MODEL

A model that captures syntax and semantic patterns to predict the likelihood of specific sequences of words

THE LANGUAGE MODEL

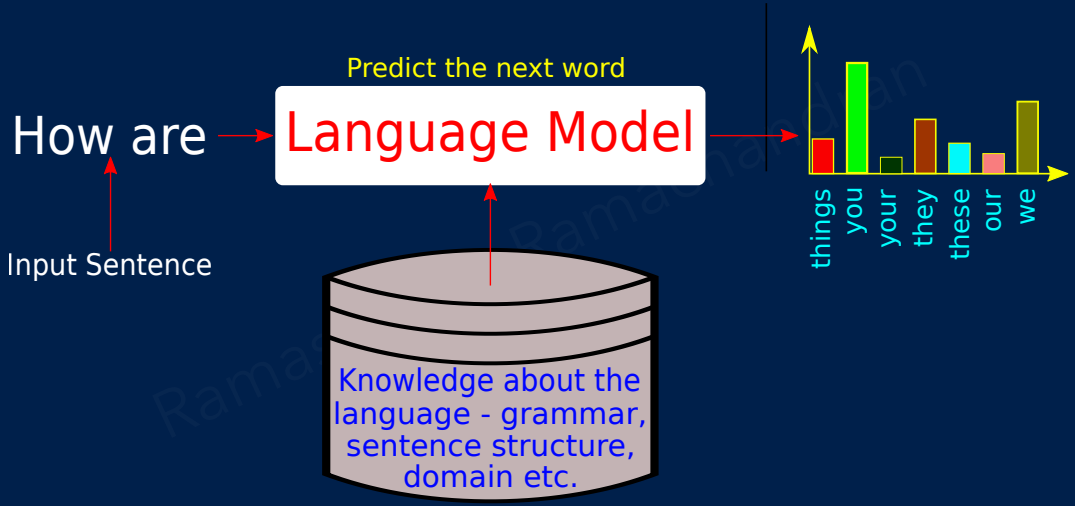
- Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- In probabilistic world, the Language model is used to assign a probability $P(W)$ to every possible word sequence W.

The current research in Language models focuses more on building the model from the huge corpus of text

APPLICATIONS

Application	Sample Sentences
Speech Recognition	Did you hear ***Recognize speech*** or Wreck a nice beach?
Context sensitive Spelling	One upon a ***tie***, ***Their*** lived aking
Machine translation	artwork is good → l'oeuvre est bonne
Sentence Completion	Complete a sentence as the previous word is given - GMail

A SIMPLE LANGUAGE MODEL IMPLEMENTATION



WHY PROBABILISTIC MODEL? I

- Uncertainty and stochasticity can arise from many sources
  - Speech recognition systems cannot depend on the processed speech signals.
  - Context understanding is important to transcribe incoherent speech signals
- No single method exists for generating natural language or performing translation
  - Almost for every given word, there are multiple possible next words
- Incomplete observability
  - Have we captured all possible English sentences in the corpus?
- Incomplete modeling
  - Use of simple a simple rule rather than a complex but dererministic one
- How do model systems that handle Inherent stochasticity?
- We need models to represent and reason under conditions of uncertainty

WHY PROBABILISTIC MODEL? II

- Propositional Logic provides a set of formal rules to determing whether a proposition is true or false
- Probability theory offers formal rules to determine the likelihood of a proposition based on the likelihood of others
- It provide a powerful framework for reasoning under uncertainty

---

- It enables the model to predict the most probable next set of words for a given word or phrase, enhancing applications like text generation, machine translation, and speech recognition
- It moves beyond rigid rule-based systems to more adaptive and context-aware mechanisms

WHY PROBABILISTIC MODEL? III

- A probabilistic model continuously assesses the rank of words in a sequence, phrase, or sentence based on their frequency of occurrence.

- Identify the most likely next set of words for a given word
- Determine the probability of the next word given the previous word using conditional probability - $\mathrm{p}(word|next) = \frac{\mathrm{p}(next, word)}{\mathrm{p}(next)}$

Evaluation of Language Models

PROBABILISTIC ASSERTIONS

In the logical assertions, we strictly rule out possibilities - if-else-endif

- Probabilistic assertions are about possible worlds

  - how probable the various worlds are and the set of all possible worlds is called the sample space

FORMAL DEFINITION

A probability model is a mathematical framework that describes the likelihood of different outcomes occurring in a certain event or situation.

Let V be the vocabulary, a finite set of symbols or words. Let us use $\triangleleft$ and $\triangleright$ as the start and stop symbols and let them be the part of V. Let $|V|$ denote the size of V.

Let W be infinite sequences of words from the collection of V. Every sequence in W starts with $\triangleleft$ and ends with $\triangleright$. Then a language model is a probability distribution of a random variable X which takes values from W.

Or p: $W \rightarrow \mathbb{R}$ such that

$$\forall x \in W,\ p(x) \geq 0 \text{ and } \sum_{x \in W} p(X = x) = 1 \tag{1}$$

PROBABILISTIC LANGUAGE MODEL

Goal: Compute the probability of a sequence of words

$$P(W) = P(w_1, w_2, w_3, ...w_n) \tag{2}$$

Task: To predict the next word using probability. Given the context, find the next word using

$$P(w_n | w_1, w_2, w_3, \ldots, w_{n-1}) \tag{3}$$

A model which computes the probability for (2) or predicting the next word (3) or complete the partial sentence is called as Probabilistic Language Model.

The goal is to learn the joint probability function of sequences of words in a language.
The probability of P(The cat roars) is less likely to happen than
P(The cat meows)

CHAIN RULE

The chain rule states that the probability of a sequence of events occurring is the product of the probabilities of each event occurring. In the context of NLP, The sentence "I am a large language model" occurring is the product of the probabilities of the words "I", "am", "a", "large", "language", and "model" occurring in the corpus.

- Chain rule models joint probability of multiple events in NLP.
- For word sequence $(w_1, w_2, \ldots, w_n)$:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2|w_1) \cdot \ldots \cdot P(w_n|w_1, w_2, \ldots, w_{n-1})$$
$$= \prod_{i=1}^{n} p(w_i|w_{1:i-1})$$

- Breaks down into product of conditional probabilities.
- Essential for language models (n-grams, Markov models, RNNs, transformers).
- Enables language sequence modeling and likelihood estimation.

CHAIN RULE - EXAMPLE

Any sentence can be decomposed using chain rule of probability (conditioned on the context) as

$$\begin{aligned} P(w_1, \ldots, w_n) &= p(w_1)p(w_2|w_1)\cdots p(w_n|w_{1:n-1}) \\ &= \prod_{i=1}^{n} p(w_i|w_{1:i-1}) \end{aligned}$$

The probability of the sentence `<start>I am a large language model<end>` is

$$\begin{aligned} P(<start>\ldots, model<end>) = & P(I|<start>) \\ & P(am|<start>, I) \\ & P(a|<star8)|<start>, I, am, \ldots, language, model) \end{aligned} \quad (4)$$

GENERATIVE MODEL

- A Generative model (GM) generates new sentences
- Produces sequences of words, sentences, or paragraphs that resemble any natural language
- Captures the underlying patterns and statistics of a corpus to create create new sentences
- Learns the underlying structure of language - includes grammar, syntax, semantics(?), and context

GENERATIVE LANGUAGE MODEL

- Ability to synthesize new sentences that are statistically indistinguishable from the training data (Ideal to pass the Turing Test)
- Describes the likelihood of any string (word or a sentence) using probability distribution
- Evaluates sentences - "The cat sat on the mat" is more likely than "The mat cat on the sat"
- Assists in predicting the next word or help in completing the sentence
- Provides alternate construct to a sentence
- Corrects spelling mistakes or grammar
- Provides translations or answers a question, if pairs of models are developed

Since natural languages are not like std-20 version of C++, the language models are not perfect :)

GENERATIVE LANGUAGE MODEL

- Rely on large amounts of training data to learn the patterns and statistics
- Depends on the diversity of the training data to have a good quality output

GENERATIVE MODEL

Here is the mathematical description of a generative model in NLP is the following:

$$P(w_1, w_2, ..., w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_n|w_1, w_2, ..., w_{n-1}) \quad (5)$$

where $P(w_1, w_2, ..., w_n)$ is the probability of the sequence of words $w_1, w_2, \ldots, w_n$ and
$P(w_i|w_1, w_2, \ldots, w_{i-1})$ is the probability of the word w_i given the previous words, or its context$w_1, w_2, \ldots, w_{i-1}$
What is the probability of the sentence ***The cat sat on the mat***?

$$\begin{aligned} P(\text{The cat sat on the mat}) =& P(\text{The})P(\text{cat}|\text{The}) \times P(\text{sat}|\text{The cat}) \\ & \times P(\text{on}|\text{The cat sat}) \times P(\text{the}|\text{The cat sat on}) \\ & \times P(\text{mat}|\text{The cat sat on the}) \end{aligned} \quad (6)$$

The probability distribution of words in a language is used as the knowledge to generate new text that appears lexically similar to the text from the corpus.

MARKOV ASSUMPTION I

Markov Assumption:The future behavior of a dynamic system depends on its recent history and not on the entire history
The product of the conditional probabilities can be written approximately for a bigram as

$$P(w_k|w_{1:k-1}) \approx P(w_k|w_{k-1}) \tag{7}$$

Equation (7) can be generalized for an *n-gram* as

$$P(w_k|w_1^{k-1}) \approx P(w_k|w_{k-K+1:k-1}) \tag{8}$$

MARKOV ASSUMPTION II

Now, the joint probability of a sequence can be re-written as

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n) = P(w_{1:n}) \tag{9}$$

$$= P(w_1)P(w2|w_1)P(w3|w_2, w_1)\ldots P(w_n|w_{n-1}, w_{n-2}, w_{n-3}, \ldots, w_1) \tag{10}$$

$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \tag{11}$$

$$\approx \prod_{k=1}^{n} P(w_k|w_{k-K+1:k-1}) \tag{12}$$

MOST LIKELIHOOD ESTIMATION

Given a training corpus of sentences, we can calculate the MLE for probabilities.

Let w_i represent the i-th word in a sentence

w_1^{i-1} denote the context words before w_i

$$P(w_i|w_{1:i-1}) = \frac{\text{count}(w_{1:i-1}, w_i)}{\text{count}(w_{1:i-1})} \tag{13}$$

The MLE for probability $P(w_i|w_{1:i-1})$ is estimated by counting the occurrences of w_i with the specific context and normalizing by the total count of that context: Here, $\text{count}(w_{1:i-1}, w_i)$ represents the number of times the context $w_{1:i-1}$ is followed by the word w_i, and $\text{count}(w_{1:i-1})$ is the count of the context $w_{1:i-1}$.

TARGET AND CONTEXT WORDS

Next word in the sentence depends on its immediate past words, known as context words

$$P(w_{k+1}|\underbrace{w_{i-k}, w_{i-k+1}, \ldots, w_k}_{\text{Context words}})$$

n-grams

- unigram - $P(w_{k+1})$
- bigram - $P(w_{k+1}|w_k)$
- trigram - $P(w_{k+1}|w_{k-1}, w_k)$
- 4-gram - $P(w_{k+1}|w_{k-2}, w_{k-1}, w_k)$

```
sentence_ngrams = list(ngrams(word_tokenize(sentence.lower()),
                              n=ngram_count,
                              pad_left=True,
                              pad_right=True,
                              left_pad_symbol='<start>',
                              right_pad_symbol='<end>'))
```

NLTK function to pad start and end of a sentencee with symbols

UNKNOWN WORDS

- In a closed vocabulary language model, there is no unknown words or *out of vocabulary words (OOV)*
- In an open vocabulary system, you will find new words that are not present in the trained model
- Pick words below certain frequency and replace them as OOV.
- Treat every OOV as a regular word
- During testing, the new words would be treated as OOV and the corresponding frequency will be used for computation
- This eliminates zero probability for sentences containing OOV

SMOOTHING

- One of the methods to find the unknown parameter(s) is the use of Maximum Likelihood Estimate
- Estimate the parameter value for which the observed data has the highest probability
- Training data may not have all the words in the vocabulary
- If a sentence with an unknown word is presented, then the MLE is zero.
- Add a smoothing parameter to the equation without affecting the overall probability requirements

$$P(\mathbf{W}) = \frac{C_{w_i} + \alpha}{C_W + \alpha|V|} \tag{14}$$

$$\text{If } \alpha = 1, \text{then it is called as Laplace smoothing} \tag{15}$$

$$P(\mathbf{W}) = \frac{C_{w_i} + 1}{C_W + |V|} \tag{16}$$

LANGUAGE MODELING USING UNIGRAMS

- All words are generated independent of its history $w, w_2, w_3, \ldots . w_n$ and none of them depend on the other
- Not a good model for language generation
- It will have $|V|$ parameters
- $\theta_i = p(w_i) = \frac{c_{w_i}}{N}$, where c_{w_i} if the count of the word w_i and N is the total number of words in the vocabulary
- It may not be able to pick up regularities present tin the corpus
- It is more likely to generate the the the the as a sentence than a grammatically valid sentence

BIGRAM LANGUAGE MODEL

- This model generates a sequence one word at a time, starting with the first word and then generating each succeeding word conditioned on the previous one or its predecessor
- A bigram language model or the Markov model (first order)is defined as follows:

$$P(\mathbf{W}) = \prod_{i=1}^{n+1} P(w_i|w_{i-1}) \tag{17}$$

where $\mathbf{W} = w_1, w_2, w_3, \ldots, w_n$

BIGRAM LANGUAGE MODEL

- Estimate the parameter $P(w_i|w_{i-1})$ for all bigrams
- The parameter estimation does not depend on the location of the word
- If we consider the sentence as a sequence in time, then they are time-invariant
- The next word is selected based on $\frac{n_{w_c,w_i}}{n_{w_c}}$
  - n_{w_c,w_i} is the number of times the words w_c, w_i occur together
  - n_{w_c} is the number of times the word/context w_c appears in the bigram sequence with any other word
- If V is the vocabulary of a corpus and V_c is the number of words in the vocabulary, what is the to number of parameters to be estimated?

BIGRAM LANGUAGE MODEL

Let's define the Bigram Language Model equation:

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \ldots \cdot P(w_n|w_{n-1}) \tag{18}$$

where

- $P(W)$: Probability of observing the entire sequence W.
- $P(w_i)$: Probability of the ith word in the sequence.
- $P(w_i|w_{i-1})$: Conditional probability of the ith word given the $(i-1)$th word.

BIGRAM LANGUAGE MODEL EXAMPLE

Now, let's consider a small corpus of sentences:

1. "I love cats."
2. "Cats love playing."
3. "Dogs chase cats."
4. "Many love cats."

We will build a bigram language model to calculate the probabilities of these sentences.

The probabilities for each sentence can be calculated as follows:

$$P(\text{"I love cats."}) = P(\text{I}) \cdot P(\text{love} \mid \text{I}) \cdot P(\text{cats} \mid \text{love}) \cdot P(\text{.} \mid \text{cats}) \tag{19}$$

$$P(\text{"Cats love playing."}) = P(\text{cats}) \cdot P(\text{love} \mid \text{cats}) \cdot P(\text{playing} \mid \text{love}) \cdot P(\text{.} \mid \text{playing}) \tag{20}$$

$$P(\text{"Dogs chase cats."}) = P(\text{dogs}) \cdot P(\text{chase} \mid \text{dogs}) \cdot P(\text{cats} \mid \text{chase}) \cdot P(\text{.} \mid \text{cats}) \tag{21}$$

COMPUTING PROBABILITIES

To calculate the probabilities, we need to count the occurrences of each word and each bigram in the corpus. For example, for the sentence "I love cats.":

- Count(I) = 1 ; Count(love| I) = 1 ;Count(cats| love) = 2 ; Count(.| cats) = 1

We calculate the probabilities as follows:

$$P(I) = \frac{\text{Count}(I)}{\text{Total number of words in the corpus}} \tag{22}$$

$$P(\text{love}|\ I) = \frac{\text{Count}(\text{love}|\ I)}{\text{Count}(I)} \tag{23}$$

$$P(\text{cats}|\ \text{love}) = \frac{\text{Count}(\text{cats}|\ \text{love})}{\text{Count}(\text{love})} \tag{24}$$

$$P(.|\ \text{cats}) = \frac{\text{Count}(.|\ \text{cats})}{\text{Count}(\text{cats})} \tag{25}$$

Finally, we can plug these values into the bigram language model equation to estimate the probability of each sentence.

BIGRAM LANGUAGE MODEL - EXAMPLE

Context - $\langle s \rangle$	
$p(the\|\langle s \rangle)$	0.13
$p(in\|\langle s \rangle)$	0.05
$p(we\|\langle s \rangle)$	0.05
$p(this\|\langle s \rangle)$	0.03
$p(a\|\langle s \rangle)$	0.02
$p(however\|\langle s \rangle)$	0.02
$p(these\|\langle s \rangle)$	0.02
$p(to\|\langle s \rangle)$	0.01
$p(our\|\langle s \rangle)$	0.01
$p(it\|\langle s \rangle)$	0.01

Context - *we*	
$p(have\|we)$	0.04
$p(also\|we)$	0.04
$p(found\|we)$	0.04
$p(show\|we)$	0.03
$p(present\|we)$	0.03
$p(propose\|we)$	0.03
$p(report\|we)$	0.03
$p(describe\|we)$	0.02
$p(used\|we)$	0.02
$p(identified\|we)$	0.02

Context - *report*	
$p(the\|report)$	0.18
$p(a\|report)$	0.1
$p(of\|report)$	0.07
$p(on\|report)$	0.06
$p(that\|report)$	0.06
$p(we\|report)$	0.05
$p(here\|report)$	0.02
$p(.\|report)$	0.02
$p(describes\|report)$	0.02
$p(is\|report)$	0.02

The bigram probabilities are computed as follows:

$$P(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)}{Counr(w_{n-1})} \tag{26}$$

$$P_L(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)+1}{Count(w_{n-1})+|V|} \tag{27}$$

where P_L is the Laplacian smoothing and $|V|$ is the size of the vocabulary.

BUILDING A BIGRAM MODEL - CODE I

```
#compute the bigram model
def build_bigram_model():
    bigram_model = collections.defaultdict(
        lambda: collections.defaultdict(lambda: 0))
    for sentence in kinematics_corpus.sents():
        sentence = [word.lower() for word in sentence
                    if word.isalpha()]  # get alpha only
        #Collect all bigrams counts for (w1,w2)
        for w1, w2 in bigrams(sentence):
            bigram_model[w1][w2] += 1
        #compute the probability for the bigram containing w1
        for w1 in bigram_model:
            #total count of bigrams conaining w1
```

BUILDING A BIGRAM MODEL - CODE II

```
            total_count = float(sum(bigram_model[w1].values()))
            #distribute the probability mass for all bigrams star
            for w2 in bigram_model[w1]:
                bigram_model[w1][w2] /= total_count
    return bigram_model

def predict_next_word(first_word):
    #build the model
    model = build_bigram_model()
    #get the next for the bigram starting with 'word'
    second_word = model[first_word]
    #get the top 10 words whose first word is 'first_word'
    top10words = Counter(second_word).most_common(10)
```

BUILDING A BIGRAM MODEL - CODE III

```
    predicted_words = list(zip(*top10words))[0]
    probability_score = list(zip(*top10words))[1]
    x_pos = np.arange(len(predicted_words))

    plt.bar(x_pos, probability_score,align='center')
    plt.xticks(x_pos, predicted_words)
    plt.ylabel('Probability Score')
    plt.xlabel('Predicted Words')
    plt.title('Predicted words for ' + first_word)
    plt.show()

if __name__ == '__main__':
    predict_next_word('how')
```

MODEL PARAMETERS - BIGRAM EXAMPLE

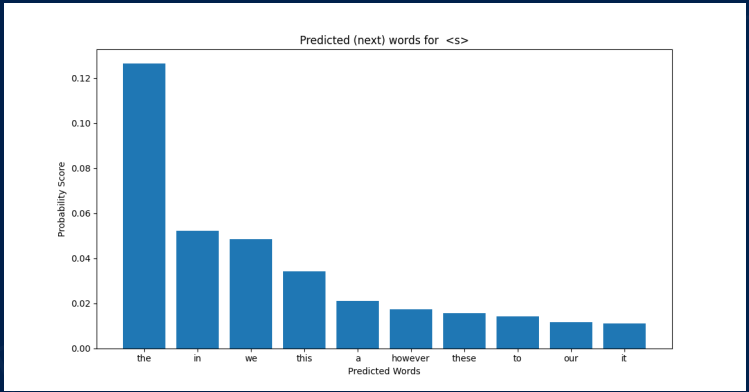
```
65  <defaultdict, len() = 226126>
66    > special variables
67    > function variables
67    > '<s>': <defaultdict, len() = 27215>
68    > 'bovine': <defaultdict, len() = 209>
69    > 'viral': <defaultdict, len() = 1541>
70    > 'diarrhoea': <defaultdict, len() = 121>
70    > 'bvd': defaultdict(<function build_bigram_model.<locals>.
71    v 'has': <defaultdict, len() = 1423>
72      > special variables
73      > function variables
74        'been': 0.31419770372444694
75        'made': 0.004620554466535984
76        'shown': 0.013114907122187996
77        'declared': 0.0014001680201624195
78        'achieved': 0.0009334453467749463
79        'proved': 0.0018668906935498926
80        'consequences': 0.00023336133669373658
          'had': 0.00695416783347335
    Hold Alt key to switch to editor language hover

    ct(lambda: collections.defaultdict(lambda: 0
    , "rb")) as openfile:

    l.load(openfile))
    (openfile)

    d19.txt')
79  model = read_model()
80
```

BIGRAM MODEL - NEXT WORD PREDICTION



BUILDING N(5)GRAM LANGUAGE MODEL - SAMPLE PROBABILITIES

Building a sentence with the context-[<start>, <start>, <start>, <start>, industrial] with a probability of 0.00081

Word	probability	
production	0.5278	1
growth	0.0833	
restructuring	0.0556	
has	0.2105	2
in	0.1579	
also	0.1053	
continued	0.5000	3
been	0.2500	
shown	0.2500	
to	1.0000	4

Word	probability	
expand	0.6667	5
rise	0.3333	
at	0.5000	6
,	0.5000	
a	1.0000	7
most	1.0000	8
satisfactory	1.0000	9
rate	1.0000	10
.	1.0000	11
<end>	1.0000	12

Sentence : Industrial production has continued to expand at a most satisfactory rate. Probability of the sentence = 1.5033637764498063e-05

PERPLEXITY

- A measure to evaluate a language model
- Quantifies how well a language model predicts the next word in a sequence
- Lower the perplexity implies a better language model

The perplexity of the language model on the test set is a measure of how well the language model generalizes to unseen text.

PERPLEXITY OF A BIGRAM MODEL

The perplexity of a bigram model is calculated using the following equation:

$$\begin{aligned} \mathrm{PP} &= \exp\left(-\frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\log \mathrm{P}(w_{\mathrm{i}}|w_{\mathrm{i}-1})\right) \\ &= \exp\left(-\frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\log\left(\frac{\mathrm{count}(w_{\mathrm{i}},w_{\mathrm{i}-1})}{\mathrm{count}(w_{\mathrm{i}-1})}\right)\right) \end{aligned} \tag{28}$$

where

- PP is the measure of the language model
- N is the number of words in the test set
- w_{i} is the ith word in the test set
- $w_{\mathrm{i}-1}$ is the $(\mathrm{i}-1)$th word in the test set
- $\mathrm{count}(w_{\mathrm{i}},w_{\mathrm{i}-1})$ is the number of times the bigram $(w_{\mathrm{i}},w_{\mathrm{i}-1})$ appears in the training corpus
- $\mathrm{count}(w_{\mathrm{i}-1})$ is the number of times the word $w_{\mathrm{i}-1}$ appears in the training corpus

PERPLEXITY OF A TRIGRAM MODEL

The perplexity of a trigram model is calculated using the following equation:

$$\text{PP} = \exp\left(-\frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}}\log\left(\frac{\text{count}(w_{\text{i}}, w_{\text{i}-1}, w_{\text{i}-2})}{\text{count}(w_{\text{i}-1}, w_{\text{i}-2})}\right)\right) \tag{29}$$

where

PP is the perplexity of the language model

N is the number of words in the test set

The only difference is that the trigram language model uses the probability of the trigram $(w_{\text{i}}, w_{\text{i}-1}, w_{\text{i}-2})$ to predict the next word, while the bigram language model uses the probability of the bigram $(w_{\text{i}}, w_{\text{i}-1})$ to predict the next word.

THE PERPLEXITY OF N-GRAM MODELS

For an N-gram model, perplexity is calculated using the following generalized equation:

$$\mathrm{PP} = \exp\left(-\frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\log\left(\frac{\mathrm{count}(w_{\mathrm{i}}, w_{\mathrm{i}-1}, w_{\mathrm{i}-2}, \ldots, w_{\mathrm{i}-(\mathrm{n}-1)})}{\mathrm{count}(w_{\mathrm{i}-1}, w_{\mathrm{i}-2}, \ldots, w_{\mathrm{i}-(\mathrm{n}-1)})}\right)\right) \tag{30}$$

where PP is the perplexity of the language model

PERPLEXITY - EXAMPLE I

Assuming the estimated probabilities for the seen sentence "I like cats" are as follows:

$$P(\text{I}|\triangleleft\text{start}\triangleright) = 0.25;\ P(\text{like}|\text{I}) = 0.5$$
$$P(\text{cats}|\text{like}) = 0.5;\ P(\triangleleft\text{end}\triangleright|\text{cats}) = 0.05$$
$$P(\triangleleft\text{end}\triangleright|\text{dogs}) = 0.05$$

PERPLEXITY - EXAMPLE II

We can calculate the probability of the sentences as follows:

$$\begin{aligned} P(\text{I like cats}) &= P(\text{I}|\triangleleft\text{start}\triangleright) \cdot P(\text{like}|\text{I}) \cdot P(\text{cats}|\text{like}) \cdot P(\triangleleft\text{end}\triangleright|\text{cats}) \\ &= 0.25 \cdot 0.5 \cdot 0.5 \cdot 0.05 = 0.003125 \end{aligned}$$

$$\begin{aligned} PP &= \sqrt[3]{\frac{1}{P(\text{I like cats})}} \\ &= \sqrt[3]{\frac{1}{0.003125}} = 6.84 \end{aligned}$$

UNSEEN SENTENCE/WORD EXAMPLE I

Now, let's calculate the perplexity for the unseen sentence "I like rabbits." Smoothing can be applied to "rabbits" as it is unseen in the training corpus. We can either use smoothing for all bigrams or use smoothing only for unseen word. In this case, we are using the smoothing only for the unseen tokens/words. Assuming we use additive smoothing with $\alpha = 0.5$ and a vocabulary size of $V = 1000$, we can estimate:

$$
\begin{aligned}
P_{\text{smooth}}(\text{rabbits}|\text{like}) &= \frac{\text{count}(\text{like}, \text{rabbits}) + \alpha}{\text{count}(\text{like}) + \alpha \cdot V} \\
&= \frac{0 + 0.5}{2 + 0.5 \cdot 1000} \\
&= 0.001
\end{aligned}
$$

UNSEEN SENTENCE/WORD EXAMPLE II

The probability of the unseen sentence "I like rabbits" using the smoothing function can be calculated as:

$$P_{smooth}(\triangleleft end \triangleright | rabbits) = \frac{0 + 0.5}{2 + 0.5 \cdot 1000} = 0.001$$

$$\begin{aligned} P(\text{I like rabbits}) &= P(\text{I}|\triangleleft start \triangleright) \cdot P(\text{like}|\text{I}) \cdot P_{smooth}(\text{rabbits}|\text{like}) \cdot P(\triangleleft end \triangleright | \text{rabbits}) \\ &= 0.25 \cdot 0.5 \cdot 0.001 \cdot 0.001 \\ &= 1.25e-07 \end{aligned}$$

UNSEEN SENTENCE/WORD EXAMPLE III

The perplexity of this unseen sentence is

$$\begin{aligned}\text{Perplexity}_{\text{unseen}} &= \sqrt[3]{\frac{1}{\text{P(I like rabbits)}}} \\ &= \sqrt[3]{\frac{1}{1.25e-07}} \\ &= 200\end{aligned}$$

In conclusion, the perplexity for the seen sentence "I like cats" is approximately 17.89. However, the perplexity for the unseen sentence "I like rabbits" is very high (2828.43) as it contains an unseen word, *rabbit*. Perplexity serves as a metric to evaluate language models, where lower values indicate better model performance in predicting given sentences.

CURSE OF DIMENSIONALITY

Vocabulary size = V. The parameters for

- Unigram model - V parameters
- Bigram model - V^2 parameters
- Trigram model - V^3 parameters
- n-gram model - V^n parameters

The curse of dimensionality refers to the exponential increase in the size of the parameter space as the dimensionality of the data increases. In the context of an n-gram language model, the curse of dimensionality becomes apparent due to the rapidly growing number of parameters as the value of n increases.

As the free parameter space becomes larger, several challenges arise:

- Data Sparsity - Many n-grams will have zero or very low counts, making it difficult to reliably estimate their probabilities.
- Over fitting - n-gram models may become overly sensitive to specific patterns in the training data and may not generalize well
- Computational Complexity - Training and inference become computationally expensive as the number of parameters grows

TECHNIQUES

- Smoothing - Improves the estimation of probabilities for unseen n-grams
- Pruning - Low count n-grams can be removed to improve the estimation of probabilities and computational efficiency
- Back-off and interpolation - combines n-gram models of different orders to balance the trade-off between data sparsity and the complexity of the model

CURSE OF DIMENSIONALITY

In probalistic Language Model, we took advantage of the idea - ***temporally closer words in the word sequence are statistically more dependent***

- A fundamental problem that makes language modelling and other learning problems difficult is the curse of dimensionality
- It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variables
- If one wants to estimate the joint probability distribution of n-grams words in a language with a $|V|$ words as vocabulary, then we need to estimate a maximum of $|V|^n$ free parameters

DISADVANTAGES OF PLM

- Scalability wrt increase in context window size
- Markov assumption. It is difficult to predict a word in a longer sequence

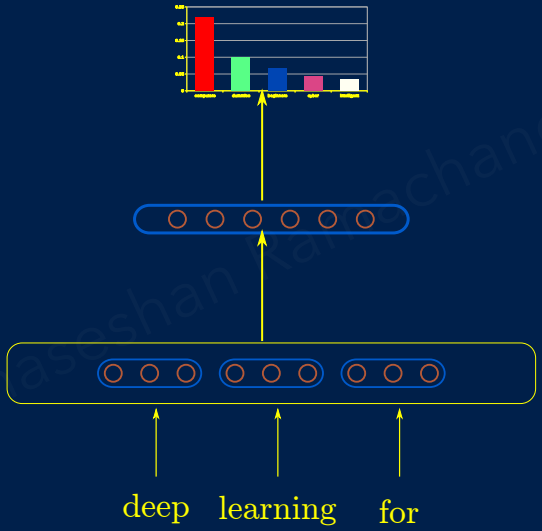
  As he is from Tamil Nadu, it is most probable that his mother tongue is ....
- Free parameter size increases exponentially with the context window size
- The context words closer to the word to be estimated are given more importance
- It does not use any semantic or syntactic similarity for the estimation

IDEAL LEARNING OF A SEQUENCE

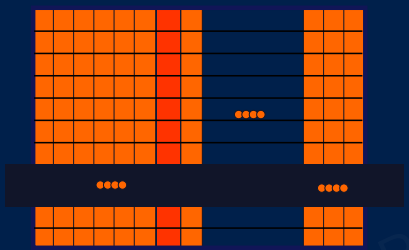
- Associate every word in the vocabulary as a distributed feature vector
- Let the feature vectors represent the joint probability of the n-gram sequences
- Learn the features as parameters to represent the sequence

Neural Language Models

ANN FOR LM



EMBEDDING LAYER CREATION



$d \times |V|$

Word Vector Size

$|V| \times 1$

One-hot vector

$d \times 1$

NEURAL LM MODEL

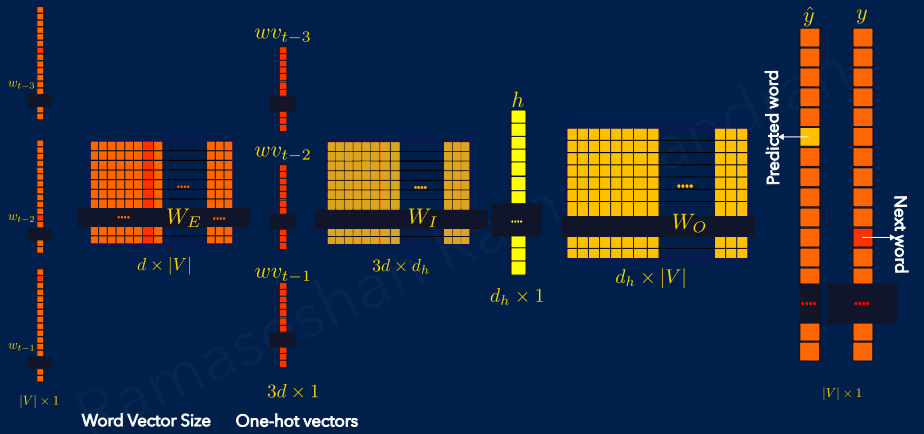


Figure: Adopted from https://web.stanford.edu/~jurafsky/slp3/7.pdf - Chapter 7

FORWARD PASS

- Forward pass is used to compute the probability distribution over the possible outputs.
- Word embedding matrix is used
- One-hot vector is used to get the index of the input word to get the corresponding word vector from the embedding matrix
- Hidden layer values are computed using word vectors of the context words and the input weights W_I
- The output $\hat{y}$ is computed using the hidden layer and the output weights W_O
- Embedding weights W_E is shared among all the context words
- h depends on W_E and W_I
- The parameters of this model are W_E, W_I, h, W_O

LOSS FUNCTION

The loss function measures how much the output $\hat{y}$ differs from the true observation, y. The loss function $\mathscr{L}$ is with respect to the correct output y is a maximum likelihood estimate. The parameters of the models are chosen to maximize the probability given the input-output relationships.

$$\hat{y} = p(y|\text{context words}) \tag{31}$$

We can use cross-entropy as loss function if the estimated result and the target are probability distributions. Since we know in this case that there is only one correct outcome, given the context words, the above equation can take the form[1]

$$p(x) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases} \tag{32}$$

Now, $p(y|\text{context words})$ in equation(31) can be written as

$$p(y|\text{context words}) = \hat{y}^{y} - (1-\hat{y})^{1-y} \tag{33}$$

[1] Since there are only two discrete outcomes (1 or 0), this is a Bernoulli distribution

LOSS FUNCTION

Taking log on both sides of equation(33)

$$\begin{aligned}\log(p(y|\text{context words})) &= \log(\hat{y}^y - (1-\hat{y})^{1-y}) \\ &= y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \end{aligned} \tag{34}$$

By incorporating equation(31), the loss function will be a cross entropy loss function which we need to minimize during the learning process

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \tag{35}$$

Replacing $\hat{y} = f(\mathbf{w_O}.\mathbf{h} + b)$ in equation we get

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(f(\mathbf{w_O}.\mathbf{h} + b)) - (1-y)\log(1-f(\mathbf{w_O}.\mathbf{h} + b)) \tag{36}$$

SIGNIFICANCE OF THE CROSS-ENTROPY LOSS FUNCTION

As we are dealing with one-hot vector as the true output, the equation(35) becomes

$$\begin{aligned}\mathscr{E}(W_E, W_I, h, W_O) &= -y \log(\hat{y}) \\ &= -y \log(f(\mathbf{w_O}.\mathbf{h} + b)) \text{ or} \\ \mathbf{E}(W_E, W_I, h, W_O) &= -\sum_{d=1}^{|V|} \mathbf{y}_d \log(\hat{\mathbf{y}}_d) \end{aligned} \tag{37}$$

LEARNING USING CROSS-ENTROPY

During the learning process, if

$$\mathscr{E} = \begin{cases} \text{very small, then the classifier is good - parameters are learned} \\ \text{large, then the classifier is bad or the entropy is large - Lack of predictability - Needs more training} \end{cases}$$

Since $\log(\hat{\mathbf{y}}) \in (0, \inf)$, the loss function in (35) which is a cross-entropy between $\mathbf{y}$ and the estimated $\hat{\mathbf{y}}$, ensures that the context word to be predicted gets higher probability and the incorrect one gets a minimum probability.
The idea is to

1. Learn the weights for all context and target words
2. Minimize the loss function $\mathscr{E}$
3. Generalize over all training samples (or maximize the input-output relationship)

DRAWBACKS OF TRADITIONAL ANN

- Memory-less and does not bother where the words and context come from
- Traditional Neural networks do not accept arbitrary input length
- Architecture is fixed with respect to the context window or input
- Every context is considered in isolation
- Some important tasks depend on the entire sequence of data
  $y(t+1) = f(x(t), x(t-1), x(t-2)...x(t-n))$

DRAWBACKS OF TRADITIONAL ANN

- Traditional Neural networks are not designed as a state machine
- Anything outside the context window has no impact on the decision being made
- Some NLP tasks require semantic modeling over the whole sentence
- Temporal dependencies are important and are not captured
- Do not or struggle to capture correlations between the temporal elements in sequences

SEQUENCE LEARNING

Sequence learning is the study of machine learning algorithms designed for applications that require sequential data or temporal data

- Helps in modeling complex temporal patterns in the data
- Captures the dependencies and correlations between the different elements in the sequence
- Maintains internal state that captures the context of the previous occurrences of words

APPLICATIONS

- Named Entity Recognition
- Paraphrase detection - identifying semantically equivalent questions
- Language Generation
- Machine Translation
- Speech recognition
  - Wreck a nice beach or recognize speech
- Automatically generating subtitles for a video
- Spell Checking
- Predictive typing
- Chat-bots/Dialog understanding
- Generate missing text
- Correct Hand-written text

RECURRENT NEURAL NETWORK

- Sequential data prediction is considered as a key problem in machine learning and artificial intelligence
- Unlike images where we look at the entire image, we read text documents sequentially to understand the content.
- The likelihood of any sentence can be determined from everyday use of language.
- The earlier sequence of words (in terms of time) is important to predict the next word, sentence, paragraph or chapter
- If a word occurs twice in a sentence, but could not be accommodated in the sliding window, then the word is learned twice
- An architecture that does not impose a fixed-length limit on the prior context

RNN

- States are important in the reading exercise- the next state depends on the previous states
- In order to use the previous state, we need to store it or remember it
- Inherent ability to model sequential input
- Handle variable length inputs without the use of arbitrary fixed-sized windows
- Use its own output as input
- RNNs encode not only attributional similarities between words, but also similarities between pairs of words
  - Analogy - $\mathrm{Chennai} : \mathrm{Tamil} :: \mathrm{London} : \mathrm{English}$ or $\mathrm{go} : \mathit{went} :: \mathrm{run} : \mathrm{Ran}$ or $\mathit{queen} \approx \mathrm{king} - \mathrm{man} + \mathit{woman}$

Language modeling using Recurrent Neural Networks

A SIMPLE RECURRENT NEURAL NETWORK

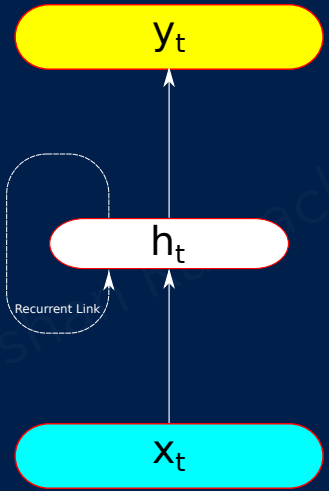
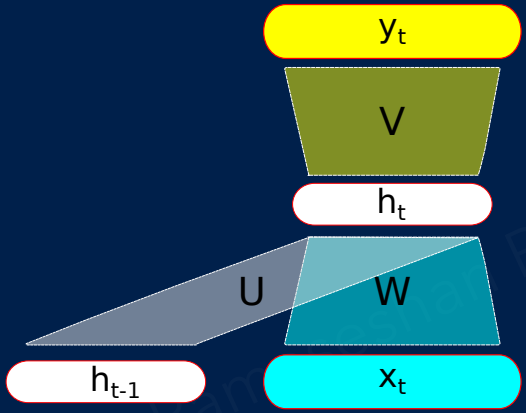


Figure: A simple Recurrent Neural Network

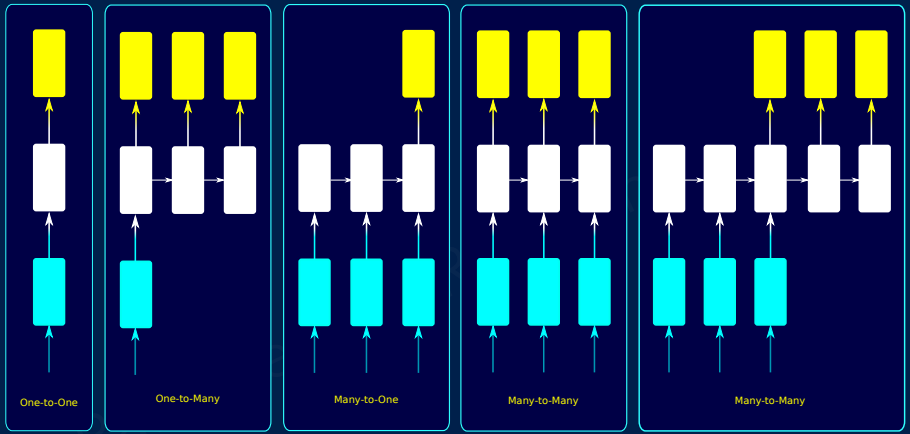
RNN - AN EXTENSION OF A FEED-FORWARD NETWORK



- the memory includes the information from the start of the sentence with no imposition of window size
- The hidden weights U from the time-stamp h_{t-1} is the significant addition to RNN
- The past weights from the previous time-stamp determines memory of the network

h_t	$= f(h_{t-1}, x_t; \theta)$
y_t	$= Vh_t$
x_t	: Input at time t
h_{t-1}	: State of hidden weights at time $t-1$

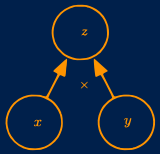
MULTIPLE ARCHITECTURES OF RNN



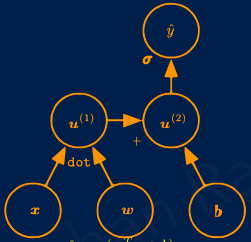
One-to-One: Classification
One-to-Many: Image captioning and image description
Many-to-One: Sentiment Analysis
Many-to-Many:Machine translation
Many-to-Many: Synced sequence input and output - frame by frame labelling

Source:http://karpathy.github.io/2015/05/21/rnn--effec:veness/

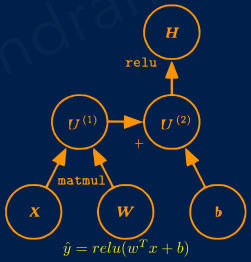
COMPUTATIONAL GRAPH[1]



(a)

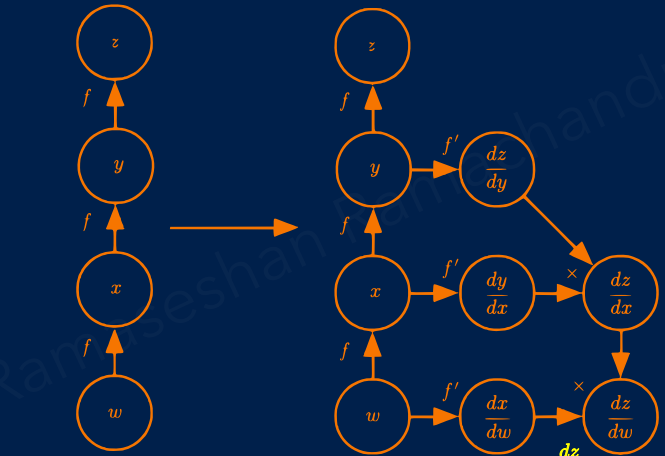


(b)

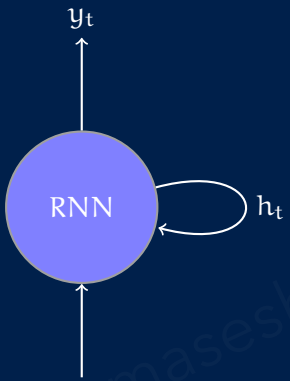


Examples of computational graphs

COMPUTATION OF DERIVATIVES[1]



RECURRENT NEURON



We can process a sequence of vectors x by applying a recurrence formula at every time step:

$h_t = f_w(h_{t_1}, x_t)$, where

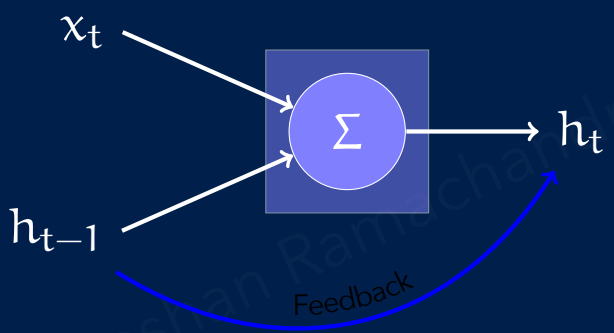
h_t is the new state,

h_{t-1} is the old state and

x_t is the input at state t or at time t

h_t	$= f(Uh_{t-1} + Wx_t + b)$
y_t	$= Vh_t$
x_t	: Input at time t
h_{t-1}	: State of hidden weights at time $t-1$

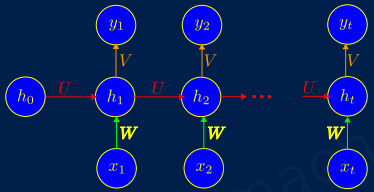
RECURRENT NEURON



$$h_t = f(U * h_{t-1} + W * x_t + b)$$

x_t : Input at time
h_{t-1} : State of neuron at time $t-1$

UNROLLED RNN



Parameters for RNN

- W - input to hidden weights
- U - hidden to hidden weights
- V - the hidden to output.

All W, U and V are shared.

$$h_0 = \tanh(Wx_0 + b) \quad (38)$$

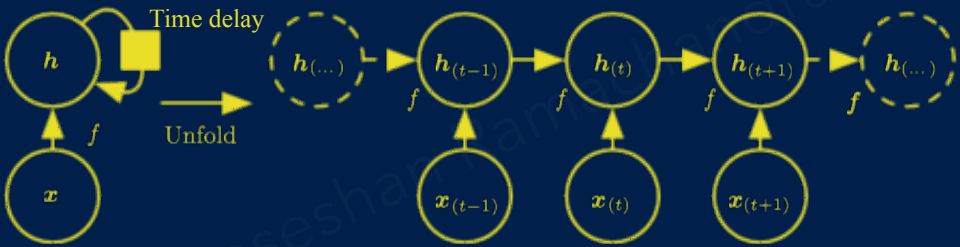
$$h_1 = \tanh(Uh_0 + Wx_1 + b) \quad (39)$$

$$\ldots \quad (40)$$

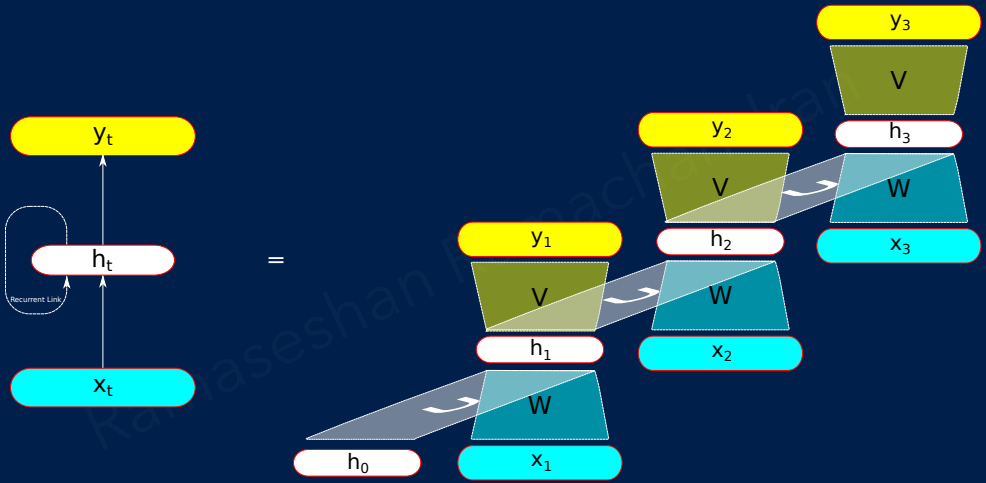
$$h_n = \tanh(Uh_{n-1} + Wx_n + b), \forall n \quad (41)$$

$$y_n = V * h_n, \text{ where } n = 1, 2, \ldots N \quad (42)$$

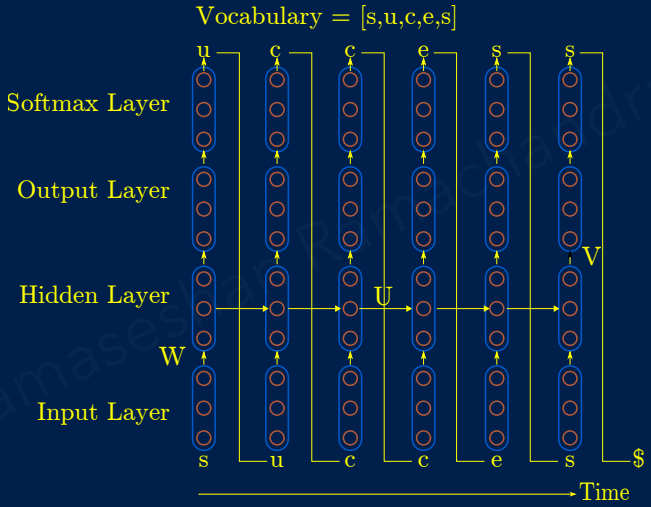
RNN WITHOUT OUTPUT[1]



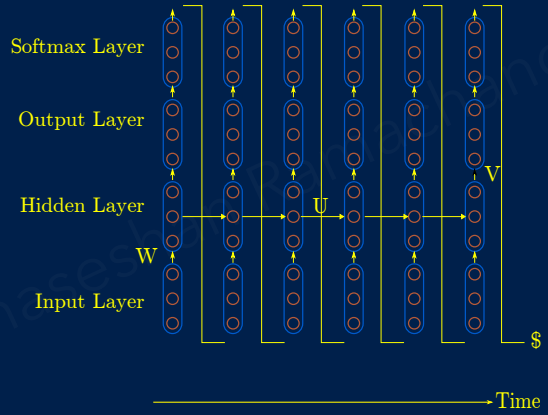
RNN UNROLLED IN TIME



CHARACTER BASED LM - RNN



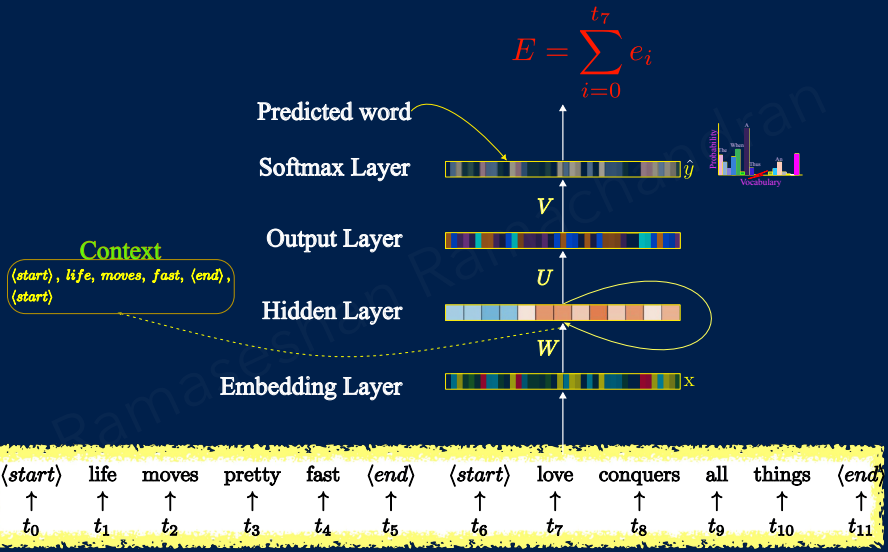
LANGUAGE MODEL - RNN



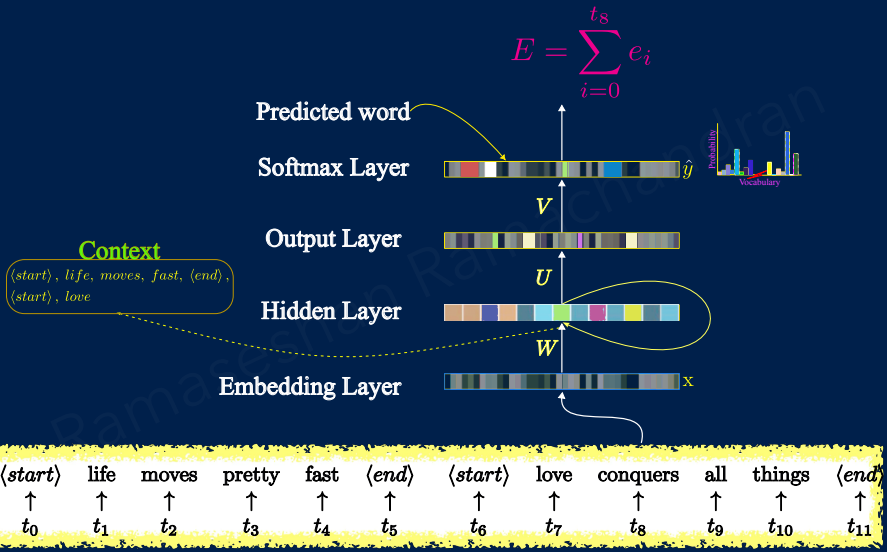
TRAINING

- FF network is static - does not worry about the sequence of the or order of the patterns, it does not matter where they occur
- The sequence must be preserved
- Two kinds of Training
  - back propagation through time
  - real time recurrent learning

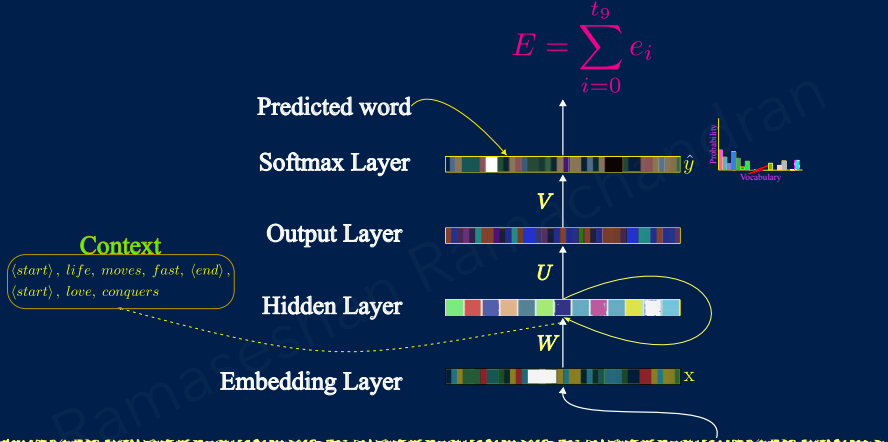
RNN TRAINING - 1



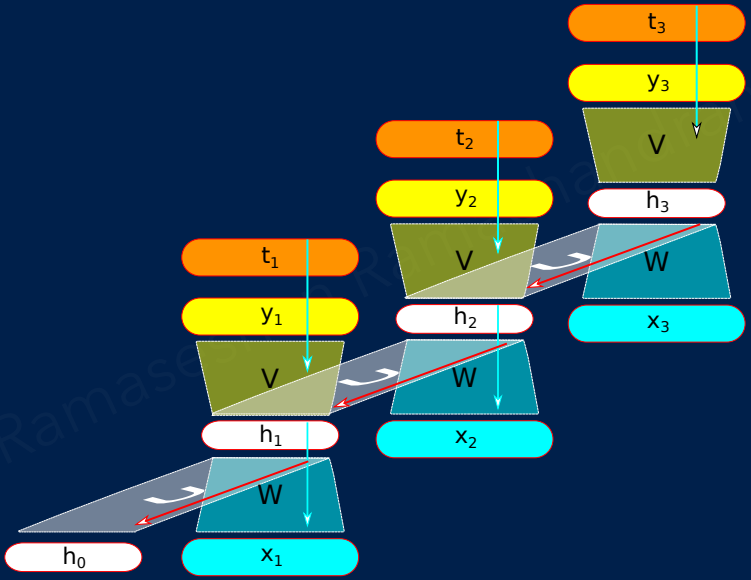
RNN TRAINING - 2



RNN TRAINING - 3



RNN TRAINING- BACK PROPAGATION THROUGH TIME



SHORT-TERM MEMORY OF RNN

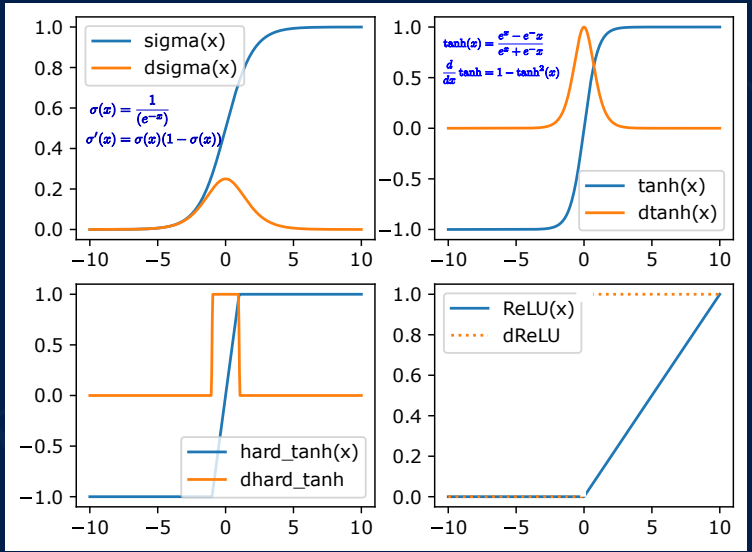
Learning to store information over extended time intervals via recurrent takes a very long time, mostly due to insufficient, decaying error back flow [2]

- Vanilla RNNs are good at learning from sequential recency rather than from long term dependency
- The temporal evolution of the back-propagated error exponentially depends on the size of the weights
- The gradients tend to either (1) explode or (2) vanish:
  - If it explodes up, then the learning may lead to oscillating weights
  - If it vanishes, then either takes a lot of time to learn or fails

DERIVATIVES OF ACTIVATION FUNCTIONS

Activation Function	Derivative
$y = \left(\frac{1}{1+e^{-x}}\right)$	$\frac{dy}{dx} = \left(\frac{1}{1+e^{-x}}\right)\left(1-\frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$
$y = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$	$\frac{dy}{dx} = \frac{\cosh(x)\cosh(x)-\sinh(x)\sinh(x)}{\cosh^2(x)} = (1-\tanh^2(x))$
$y = \max(c,x)$	$\frac{dy}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ c, & \text{if } x \leq 0 \end{cases}$ For RELU, $c = 0$ and $\frac{dy}{dx}$ does not exist at $x = 0$

DERIVATIVES OF SIGMOID, TANH, HARD TANH, RELU



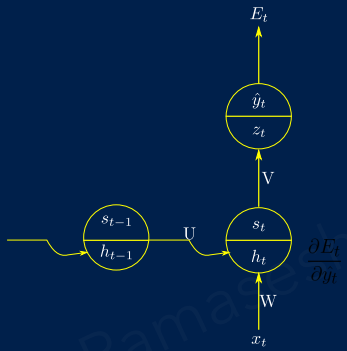
WHY $\tanh$?

$\tanh$ is symmetrical around the origin.

- Zero -centring of $\tanh$ allows faster convergence during training process
- $\partial(\tanh)$ is steep at the origin or gradients are larger when compared to sigmoid function
- Allows efficient backpropagation of errors during the training
- Speeds up the learning process or updates weights faster than sigmoid
- $\partial(\sigma)$ has a range $[0, 0.25]$. Changes slowly at the origin - may result in very slow learning at around the origin
- $\partial(\tanh)$ should be able to capture small changes
- $\partial(\tanh)$ has a range $[0, 1]$. The rapid change may minimize the problem of vanishing gradient

Is Sigmoid function symmetrical around origin?

FORWARD PASS I



Forward Pass

$$h_t = Wx_t + Uh_{t-1} \quad (43)$$

$$s_t = \tanh(h_t) \quad (44)$$

$$z_t = Vs_t \quad (45)$$

$$\hat{y_t} = softmax(z_t) \quad (46)$$

$$E = -\sum_t y_t \log(\hat{y_t}) \quad (47)$$

FORWARD PASS II

If the corpus contains T words, then $(x_1, x_2, x_3, \ldots, x_T)$ are the corresponding word vectors

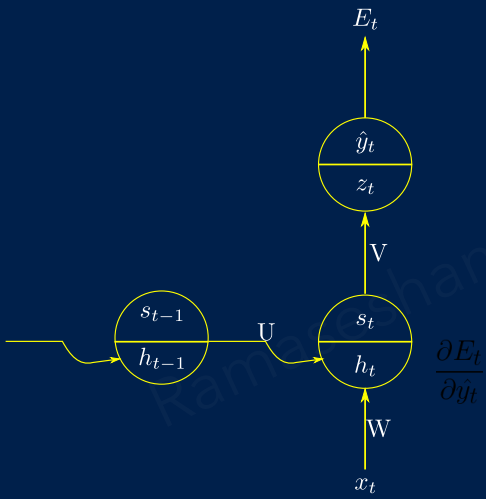
- $x_t \in R^{D_{|w|}}$ represents the input word at time t and D_w is the dimension of the word vector. If one-hot vector, it will be $x^{D_{|V|}}$
- $W \in R^{D_w \times D_h}$ is the weight matrix that conditions the input vector
- $U \in R^{D_h \times D_h}$ matrix that keeps the dependency of the word sequence
- $V \in R^{|V| \times R^{D_h}}$
- s_{t-1} is the output of the non-linear function ($\tanh$) of the time step $t-1$
- $\hat{y_t}^t \in R^{|V|}$ is the probability distribution of the predicted word at time step t for the given context of $x_1, x_2, x_3, \ldots x_t$, where $|V|$ is the size of the vocabulary

SIZE OF THE RNN NETWORK

If we assume the size of the word vector as 100 and the number of the hidden neurons as 500, and $|V| = 10000$, then

Parameter	Size
Word Vector	100
W	500x100
h_t, s_t	500
U	500x500
V	500x10000
$\hat{y_t}$	10000

BPTT - DERIVATIVE FOR V



$$h_t = Wx_t + Uh_{t-1}$$

$$s_t = \tanh(h_t)$$

$$z_t = Vs_t$$

$$\hat{y}_t = softmax(z_t)$$

$$E_t = -y_t \log(\hat{y}_t)$$

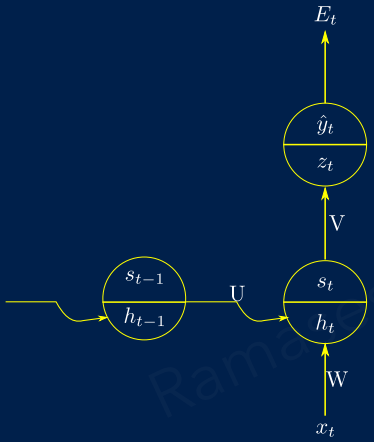
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial V} \tag{48}$$

$$\text{Let } \delta^t_{out} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}$$

$$\frac{\partial E_t}{\partial V} = \delta^t_{out} s_t \tag{49}$$

Here δ^t_{out} is the loss for each of the units in the output layer

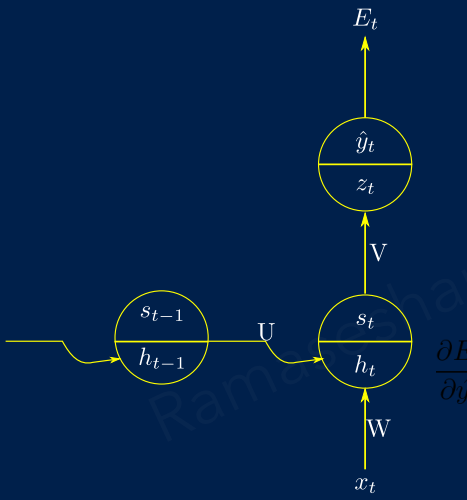
BPTT - DERIVATIVE FOR W



$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y_t}} \frac{\partial \hat{y}}{\partial z_t}} \frac{\partial z_t}{\partial s_t} \frac{\partial s_t}{\partial h_t} \frac{\partial h_t}{\partial W} = \delta^t_{out} V \sigma'(h_t) x_t \quad (50)$$

Since the hidden layer activation depends on the previous time state, we have another similar term δ_{t-1} that get added to (70)

BPTT - DERIVATIVE FOR U

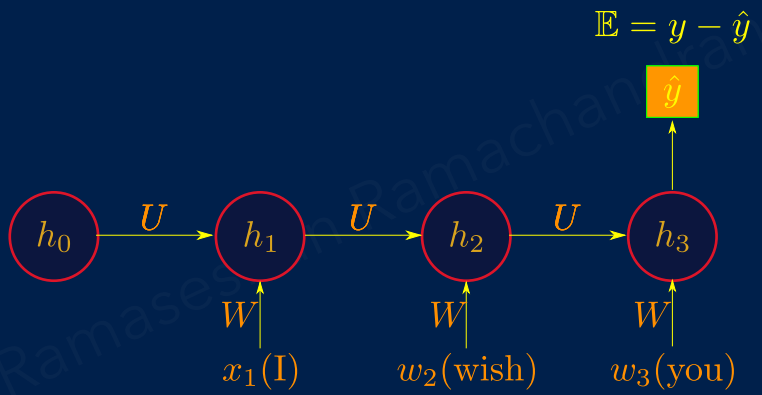


$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}}\frac{\partial h_t}{\partial U} \tag{51}$$

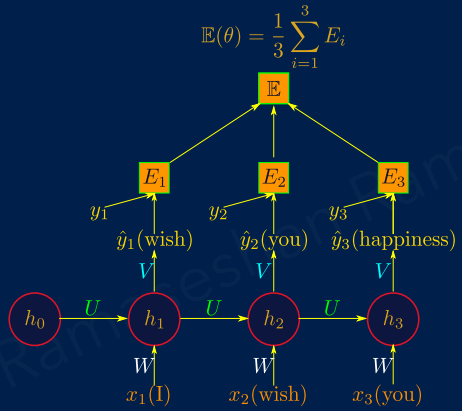
$$= \delta^t_{out} V \sigma'(h_t) h_{t-1} \tag{52}$$

Since we are back propagating the error from the current state to the previous state, $\delta_{next} = \sigma(h_t) U \delta^t_{out} V \sigma'(h_t)$ needs to be added

ERROR COMPUTATION - 1



ERROR COMPUTATION - 2



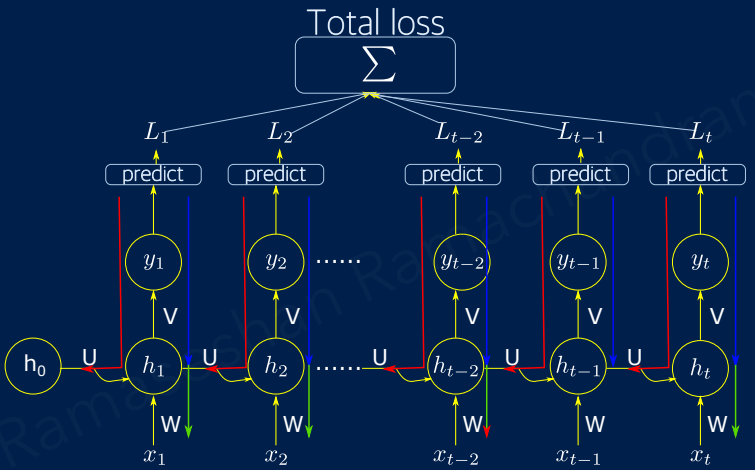
$$\mathbb{E}(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y}_{t,j})$$

We need to compute the following derivatives

$$\frac{\partial \mathbb{E}}{\partial \hat{y}_t}, \frac{\partial \mathbb{E}}{\partial V}, \frac{\partial \mathbb{E}}{\partial W}, \frac{\partial \mathbb{E}}{\partial h_t}, \text{ and } \frac{\partial \mathbb{E}}{\partial U}$$

$$h_t \bowtie h_1^{t-1}$$

BPTT - UNROLLED RNN



The error for the entire duration of T for all the vocabulary is the sum of all the error across the layers

$$\sum_{t}^{T} \sum_{v}^{|V|}$$

PERPLEXITY

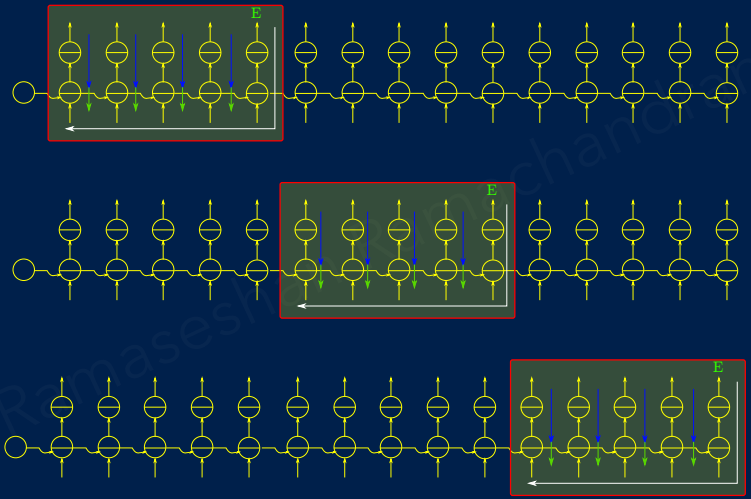
Perplexity is a measurement of how well a model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } \mathrm{PP}(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{54}$$

$$\text{For trigram model } \mathrm{PP}(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \tag{55}$$

A good model gives maximum probability to a sentence or minimum perplexity to a sentence

TBPTT



TRUNCATED BPTT

For applications with long sequences, the input is truncated into manageable fixed-sized segments. This approach is called Truncated Backpropagation Through Time (TBPTT).

Example

Consider a sequence of 5000 samples. We could split this in to 50 sequences of 100 samples each, and the BPTT is computed for each sequence. This works most of the time, but it is blind to temporal dependencies that may span across two sequences. One way to solve this is to have a sentence separator as the conditional BPTT.

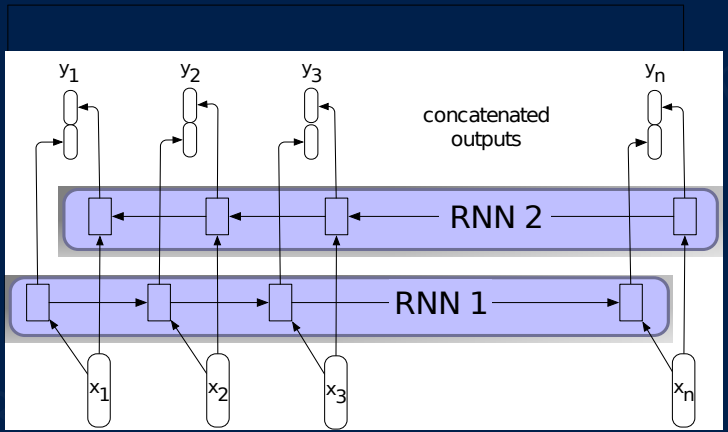
Hyper-parameter

- *batch* - the number of samples or subset of the training dataset
- *Epoch* - complete pass through the entire training dataset

BI-DIRECTIONAL RNN

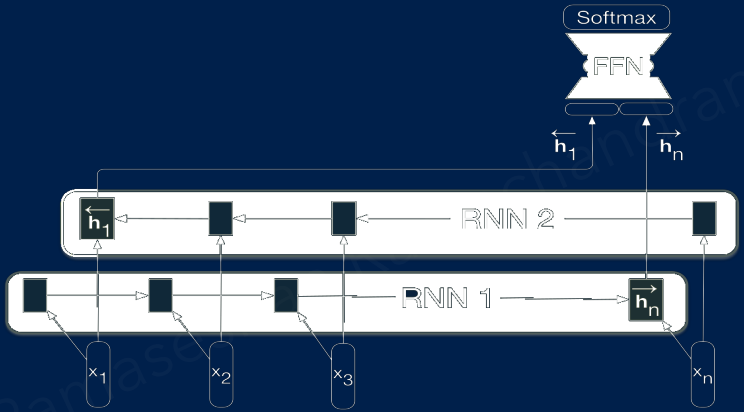
- Capture the word meaning as well as its contextual information to enable different word embeddings for the same word -contextualized word embedding
- The language model does not only have a predict the next word, but also the previous word.

BIDIRECTIONAL RNN WITH MULTIPLE OUTPUT



A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

BIDIRECTIONAL RNN WITH SINGLE OUTPUT

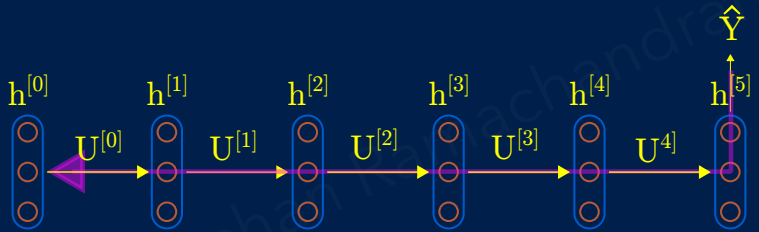


A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This com-bined representation serves as input to the subsequent classifier.

WHAT IS THE PROBLEM WITH RNN?

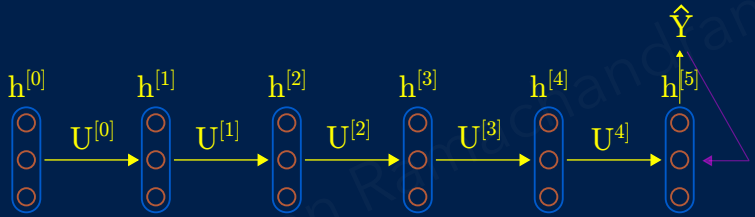
- Long term behavior depends on the eignen values of the hidden matrix U
- $Uv_i = \lambda_i v_i$
- Theoretically, it is possible to store all historical information in the RNN
- If the eigen values $\lambda > 1$, then the system will explode with very large gradients
- if the eigen values $\lambda < 1$, then the system will vanish quickly
- Vanishing gradient problem - The diminishing value of δ makes it difficult to capture the long term memory as we move down the memory lane or layers of hidden nodes
- What is the solution?

EXPLODING/VANISHING GRADIENT



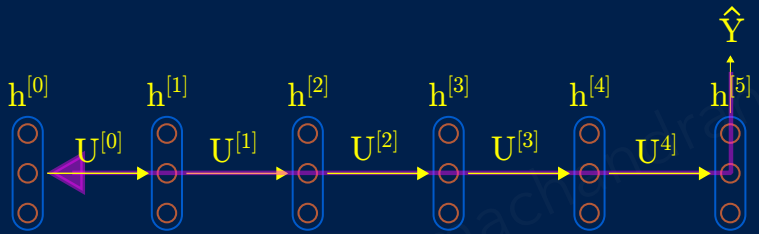
Let us assume that $\hat{y}_t = f(U, h)$. We want to propagate the error through back propagation

EXPLODING/VANISHING GRADIENT



The error E at 5th time state depends on $h^{[5]}$. Hence we need to estimate $\frac{\partial E}{\partial h^{[5]}}$. $h^{[5]}$ depends on $h^{[4]}$, $h^{[4]}$ depends on $h^{[3]}$, $h^{[3]}$ depends on $h^{[2]}$, $h^{[2]}$ depends on $h^{[1]}$, and $h^{[1]}$ depends on $h^{[0]}$.

EXPLODING/VANISHING GRADIENT



$$\frac{\partial E^{[5]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \times \frac{\partial h^{[5]}}{\partial h^{[4]}} \times \frac{\partial h^{[4]}}{\partial h^{[3]}} \times \frac{\partial h^{[3]}}{\partial h^{[2]}} \times \frac{\partial h^{[2]}}{\partial h^{[1]}} \times \frac{\partial h^{[1]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \prod_{t=1}^{t=5} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \tag{56}$$

Generalizing

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \tag{57}$$

where τ represents depth of the layers or the size of the time series

EXPLODING/VANISHING GRADIENT

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t<\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}}$$

$$h^{[t]} = \sigma\left(WX^{[t]} + Uh^{[t-1]}\right) \tag{58}$$

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} = diag(\sigma'(WX^{[t-1]} + Uh^{[t-1]}))U \tag{59}$$

where σ' computes element-wise the derivative of σ

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} \text{ is a Jacobian} \tag{60}$$

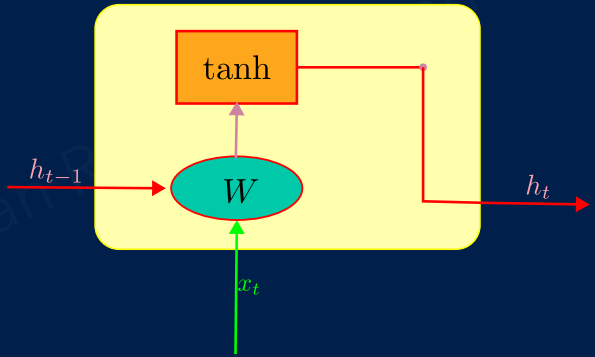
$$\therefore \frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} U^{\tau} \prod_{t=1}^{t=\tau} diag(\sigma'(WX^{[t-1]} + Uh^{[t-1]})) \tag{61}$$

Multiplication of the Jacobian is always a shrinking operation. Values of U become very small when the depth increases[2]

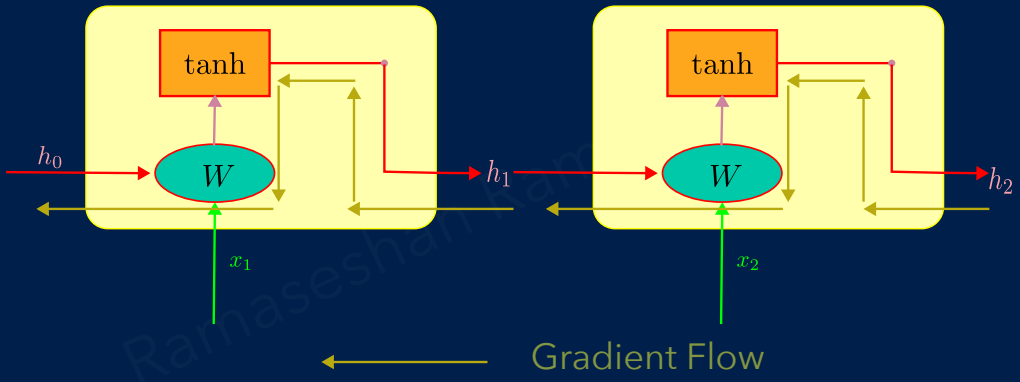
[2] Source: "On the difficulty of training recurrent neural networks", Pascanuet al, 2013 -

ANOTHER REPRESENTATION

$$
\begin{aligned}
h_t &= \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
\tag{62}
$$



GRADIENT FLOW IN RNN



EXPLODING/VANISHING GRADIENT

Consider the following sentence:
Raj entered CoffeeDay to meet his partner Dru. Raj said "Hi Dru. In the next few hours they discussed their start-up and devised a plan to develop a product on knowledge management. After a the long discussion and fruitful discussion, Raj said goodbye to his _____(47^{th} word).

The target word is ***partner***. If the long distance gradient (the gap between $U^{[7]}$ and $U^{[\$]}$ is large), then the target word is lost in the gradient as it would be to small to contribute

The decay in the gradient value is proportional to the depth of the network. The deeper the net network, the the chance of getting a smaller value of the gradient towards the end of the back propagation. If the some of the values are in the range of $[(0.01, 0.5), (0.03, 0.01)]$, then the the derivative would vanish to zero - $0.01^{47} = 1.0e-94$ and $0.5^{47} = 7.1054274e-15$

GRADIENT CLIPPING

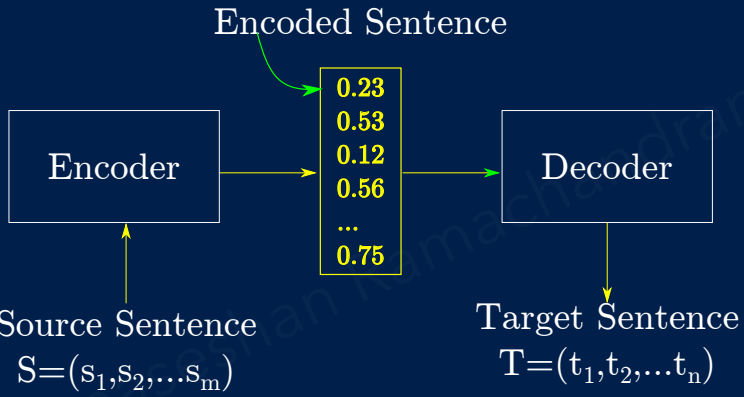
- The gradient is either very large or very small. This can cause the optimizer to converge slowly.
- To speed up training, clip the gradient at certain values
  - If $g < 1$, or if , $g > 1$, then $g = 1$
  - Or
  - If $||g|| > threshold$, then $g \leftarrow \frac{threshold}{||g||} g$
- Clip the gradient if it exceeds a threshold

Neural Language Translation

NEURAL MACHINE TRANSLATION

Neural Machine Translation (NMT) is the mechanism of modeling the Machine translation process using artificial neural network Let F and E be the source and the target sentences in a parallel corpora, respectively.

ENCODER-DECODER MODEL



- All sentences (of varying length) are encoded into fixed sized vector
- Uses fraction of the memory needed by traditional SMT models[3]
- Performance of this model decreases as the length of a source sentence increase

[3] Cho et al, On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, 2014

RNN-BASED TRANSLATION MODEL

- Uses RNN for both encoding and decoding
- Encoder maps the variable length sentence into a fixed-length vector
- Decoder translates the vector representation back to a variable-length target sequence
- Two networks are trained jointly to maximize the conditional probability of the target sentence, given the source sentence - $P(f|e)$
- This model learns a continuous space representation of a phrase that preserves both the semantic and syntactic structure of the phrase[3].

RECURRENT NEURAL NETWORK

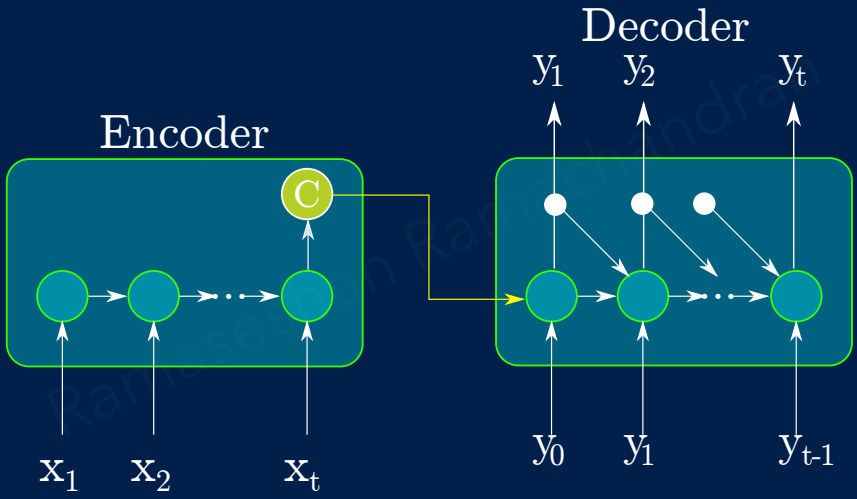
- Input units - variable length source sequence $x = (x_1, x_2, \ldots, x_T)$
- Output units - variable length target sequence $y = (y_1, y_2, \ldots, y'_T)$
- Hidden units for each input state,

$$h_t = f(h_{(t-1)}, x) \tag{63}$$

where f is a simple non-linear activation function (sigmoid or $\tanh$) or a complex LSTM/GRU cell

- RNN is trained to predict the next word in the sequence or RNN learns a probability distribution over a sequence
- The output at each time step $t = p(x_t|x_{t-1}, \ldots x_1)$
- The output distribution (Softmax layer) size is equal to the size of the vocabulary V at every unit
- Then, p(**x**) = $\prod_{t=1}^{T} p(x_t|x_{t-1}, \ldots x_1)$

RNN-BASED ENCODER-DECODER



RNN-BASED ENCODER

- RNN learns to map an input sentence of variable length into a fixed-dimensional vector representation.
- It learns to decode a fixed length vector representation back into a variable length sequence
- This model learns to predict a sequence given a sequence $p(y_1, y_2, \ldots, y'_T | x_1, x_2, \ldots, x_T)$. T and T′ may differ
- Encoder reads every symbol in $\mathbf{x}$, sequentially
- Hidden state changes according to Eq.(63)
- C is the summary of the hidden states at time T and has encoded all the symbols in the sequence

RNN-BASED DECODER

- This is trained to predict the next symbol y_t and generate the output sequence, given the previous state h_t
- y_t and $\mathbf{h}_t$ are conditioned on the summary from the encoder, C and its previous hidden state
- Decoder's hidden state is given by
- Conditional distribution for the next symbol is

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{C}) \tag{64}$$

$$P(y_t | y_{t-1}, y_{t-2} \ldots, y_1, \mathbf{C}) = g(h_{t-1}, y_{t-1}, \mathbf{C}) \tag{65}$$

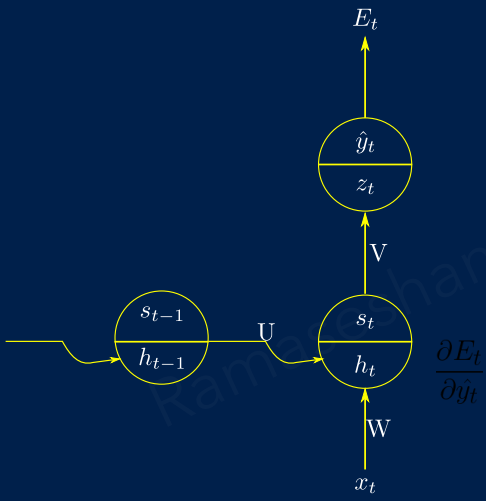
ESTIMATING MODEL PARAMETERS

Both encoder and decoder are jointly trained to maximize the conditional likelihood

$$J(\theta) = \max_{\theta} \frac{1}{N} \log p_\theta(\mathbf{y}_n|\mathbf{x}_m) \tag{66}$$

where θ is the set of model parameters that will be learned during the BPTT and $(\mathbf{x}_m, \mathbf{y}_n)$ is the source sentence sequence and target sequence pair

BPTT - DERIVATIVE FOR V



$$h_t = (Wx_t + Uh_{t-1})$$

$$s_t = \tanh(h_t)$$

$$z_t = Vs_t$$

$$\hat{y}_t = softmax(z_t)$$

$$E_t = -y_t \log(\hat{y}_t)$$

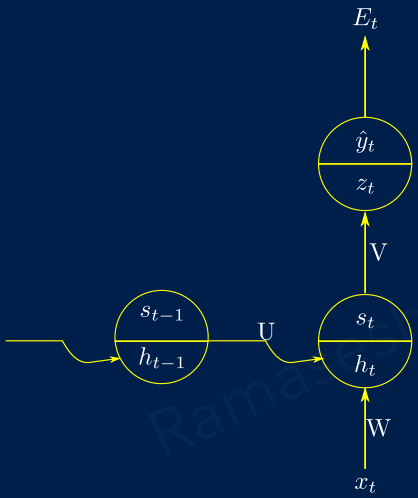
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial V} \quad (67)$$

$$\text{Let } \delta^t_{out} = \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}$$

$$\frac{\partial E_t}{\partial V} = \delta^t_{out} s_t \quad (68)$$

Here δ^t_{out} is the loss for each of the units in the output layer

BPTT - DERIVATIVE FOR W

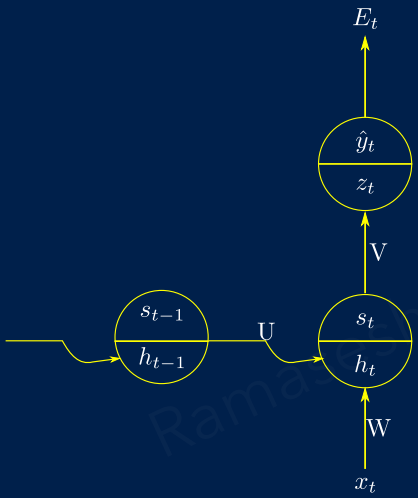


$$\frac{\partial E_t}{\partial W} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}\frac{\partial h_t}{\partial W} \tag{69}$$

$$= \delta^t_{out} V \sigma'(h_t) x_t \tag{70}$$

$\frac{\partial E_t}{\partial \hat{y}_t}$ Since the hidden layer activation depends on the previous time state, we have another similar term δ_{t-1} that get added to (70)

BPTT - DERIVATIVE FOR U



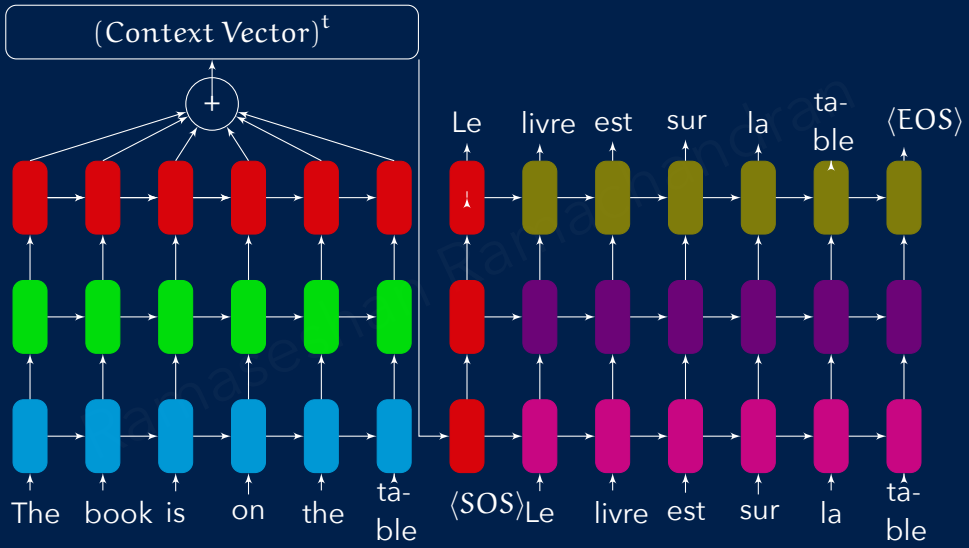
$$\frac{\partial E_t}{\partial U} = \underbrace{\frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}}{\partial z_t}\frac{\partial z_t}{\partial s_t}\frac{\partial s_t}{\partial h_t}}\frac{\partial h_t}{\partial U} \tag{71}$$

$$= \delta^t_{out} V \sigma'(h_t) h_{t-1} \tag{72}$$

$\frac{\partial E_t}{\partial \hat{y}_t}$

Since we are back propagating the error from the current state to the previous state, $\delta_{next} = \sigma(h_t) U \delta^t_{out} V \sigma'(h_t)$ needs to be added

SEQ2SEQ TRANSLATOR

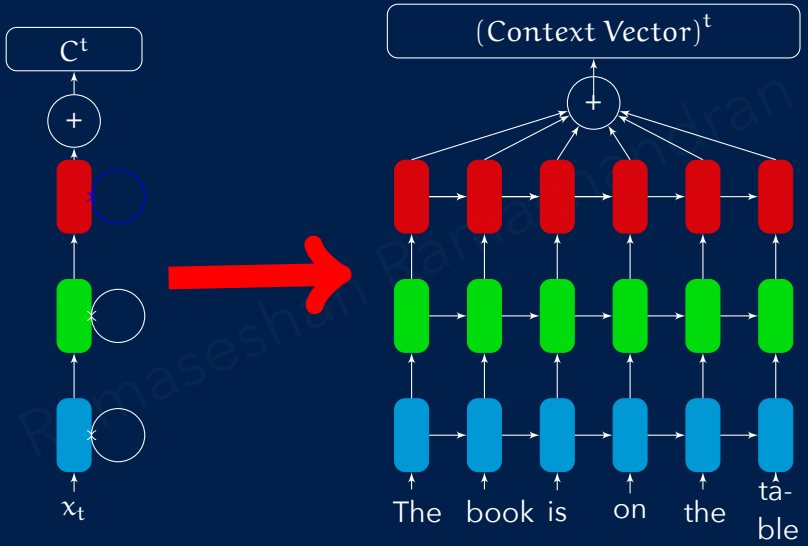


VARIATIONS IN RNN MODELS

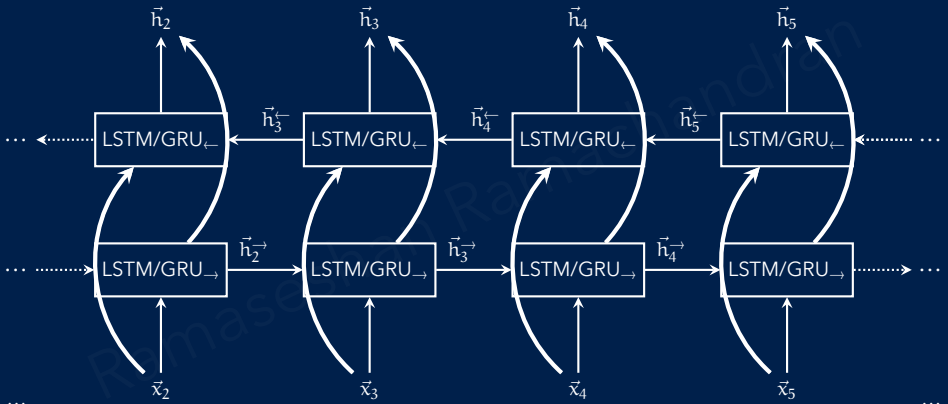
Choices vary in picking the Translation Architecture

- Directionality - Unidirectional or bidirectional
- number of hidden layers and units
- Plain vanilla RNN
- Long Short-term Memory units
- Gated Recurrent Unit
- Choice of Learning Algorithm

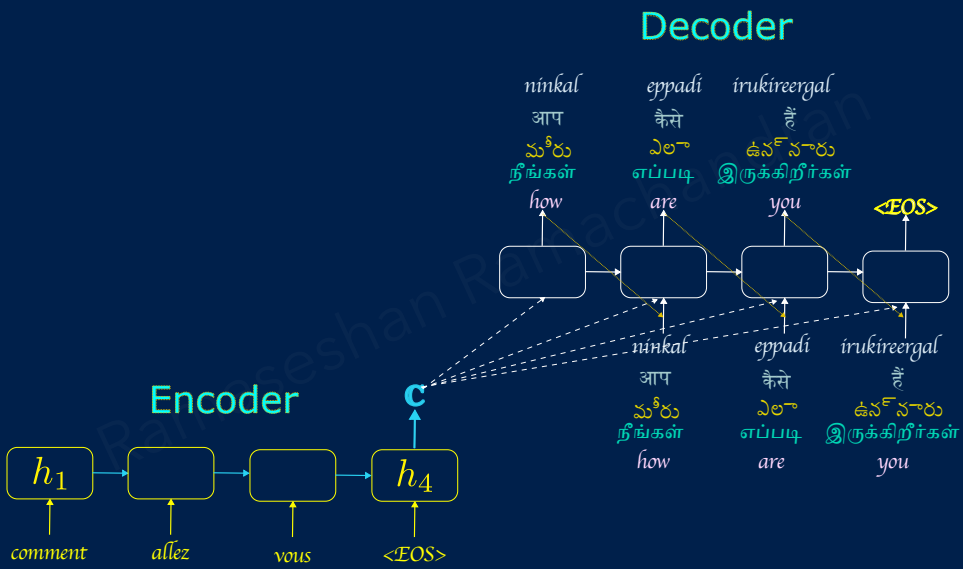
SEQ2SEQ ENCODER



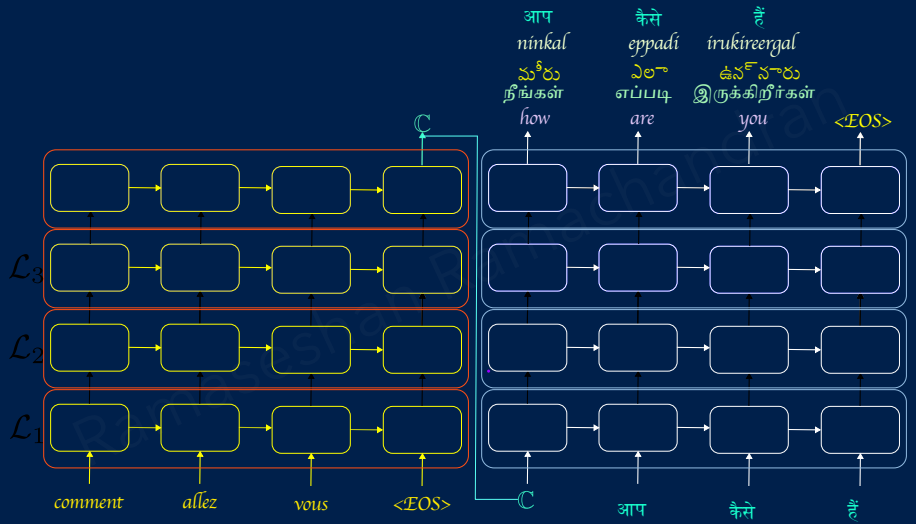
BIDIRECTIONAL RNN



SEQUENCE TO SEQUENCE TRANSLATION - NMT



SEQUENCE TO SEQUENCE TRANSLATION - DEEP RNN



APPLICATIONS

- Translation
- Dialog
- Code generation!

RNN WITH ALIGNMENT

- The objective of attention is to capture the information from the passage tokens that is relevant to the contents of the translation
- Different parts of an input have different levels of significance
- Different parts of the output may even consider different parts of the input as ”important”
- The purpose of the attention mechanism is to let the decoder *peek* at the relevant information encapsulating the source sentence as it generates the answer
- Attention mechanisms provide the decoder network with the entire input sequence at every decoding step; the decoder can then decide what input words are important at any point in time

- The attention-based model learns to assign significance to different parts of the input for each step of the output.
- In the context of translation, attention can be thought of as "alignment."
- Bahdanau et al [4] argue that the attention scores α_{ij}, at decoding step i, signify the words in the source sentence that align with word j in the target.
- We can use attention scores to build an alignment table. It is a table mapping of words in the source to corresponding words in the target sentence - based on the learned encoder and decoder from our Seq2Seq NMT system.

ENCODER

Let $x(x_1, x_2, \ldots, x_n)$ and $y(y_1, y_2, \ldots, y_m)$ be the source and target sentences
The encoder reads the input sentence x and converts into a context vector c

$$h_t = f(x_t, h_{t-1}) - \text{hidden values calculated at time } t \quad (73)$$

$$c = g(h_1, h_2, \ldots, h_n) - \text{context vectors computed using all } h_t \text{ values} \quad (74)$$

where functions f and g are non-linear functions.

How does c differ from the *context* of an n-gram language model?

DECODER

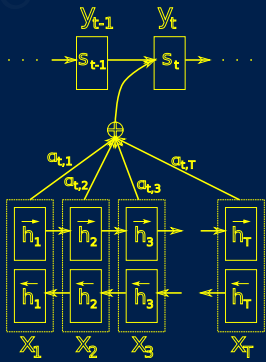
Decoder is trained to predict the next word using the c computed by the encoder

$$p(y) = p(y_t|c, \{y_1, y_2, \ldots, y_{t-1}\}) \quad (75)$$

For the RNN, the probability of the next word $p(y_t)$ is computed using

$$p(y_t) = g(y_{t-1}, s_t, c) \quad (76)$$

The hidden states s of the decoder are computed using a recursive formula of the form $s_i = f(s_{i-1}, y_{i-1}, c_i)$, where s_{i-1} is the previous hidden vector, y_{i-1} is the generated word at the previous step, and c_i is a context vector that capture the context from the original sentence that is relevant to the time step i of the decoder.



NMT WITH ATTENTION

Conditional probability for each output neuron

$$p(y_i|y_1, y_2, \ldots, x) = g(y_{i-1}, s_i, c_i) \tag{77}$$

where s_i if the RNN hidden neuron at time i and

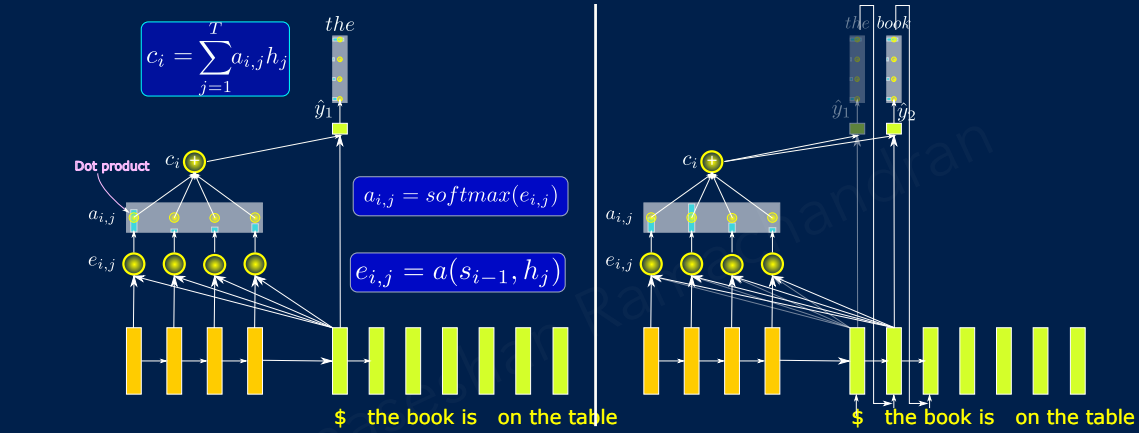
$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{78}$$

The context vector c_i depends on the sequence of annotations $(h_1, h_2, \ldots . h_{Tx})$[5]. Each h_i contains information about every word with a strong focus on context words surrounding the i^{th} word of the input sequence.
The context vector c_i is computed as the weighted sum of these annotations h_i

$$c_i = \sum_{j=1}^{T_x} \alpha_{i,j} h_j \qquad \alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{t_x} \exp(e_{i,k})} \qquad e_{i,j} = a(s_{i-1,h_j})$$

α_{ij} of each annotation h_j is computed by is the alignment model. This learns how well the inputs surrounding position j and the output at position i match

- The alignment is explicitly computed and not latent
- This alignment model is also trained along with the translation model
- α_{ij} is the probability that the target word y_i is aligned to the source x_j
- c_i is the expected annotation over all possible annotations α_{ij}
- α_{ij} or e_{ij} reflects the importance of the annotation h_j wrt to the previous hidden state s_{i-1} of the target. This enables the next state s_i to generate y_i
- The decoder decides which part of the input is important to generate a respective translation rather than depending on the encoded vector of the entire sentence
- Decoder has control over the input sequence and selectively learns to align words/phrases automatically

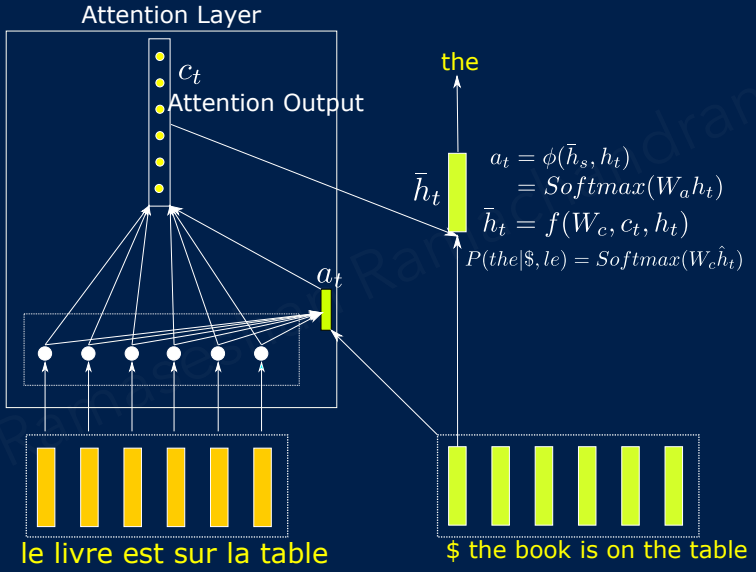


$e_{i,j}$ – attention score

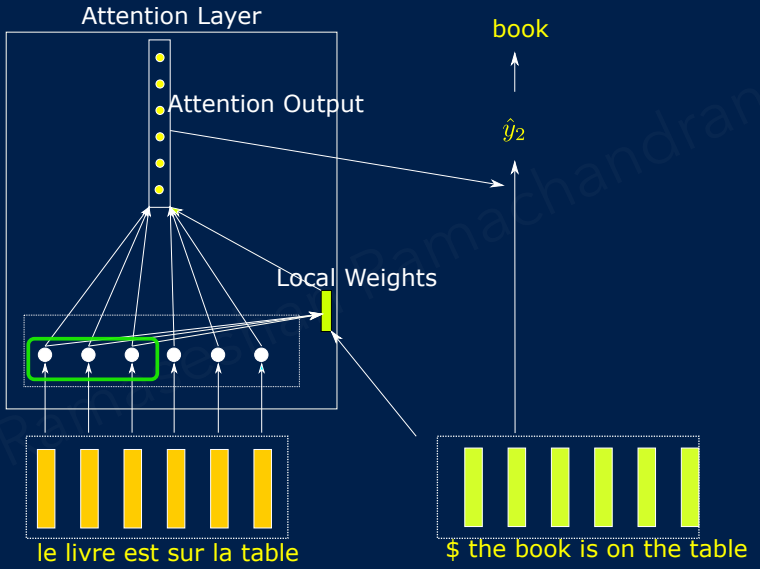
$a_{i,j}$ – attention distribution

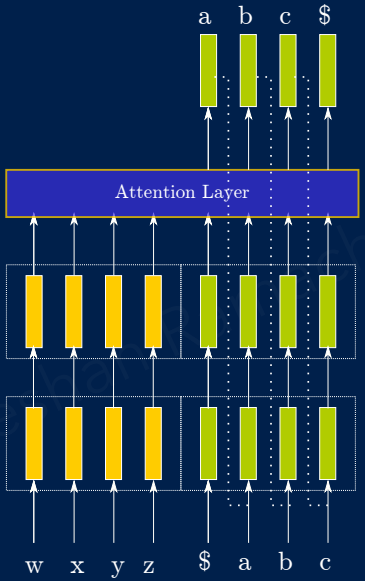
c_i – attention output

TRANSLATION WITH GLOBAL ATTENTION



TRANSLATION WITH LOCAL ATTENTION





Source: Minh-Thang Luong et al, Effective Approaches to AIttention-based Neural Machine Translation

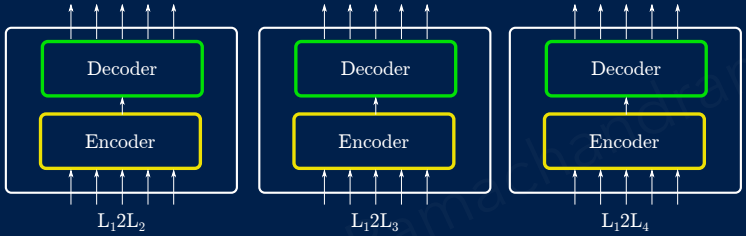
A TYPICAL SETUP

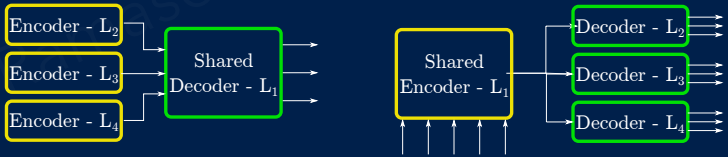
Sentence pairs	3-5M
English words	110M
French words	116M
Vocabulary	≈50K (Source and Target)
Word Embedding size	1000
Hidden layer	1000 LSTM cells
Stacked Hidden Layer	4-8
Learning Rate	Initially as high as 1 and exponential reduction
Training	
Mini batch Gradient Descend size	128
Training Time	1 GPU - about 7-10 days
Evaluation	Bleu - scores ranging from 27-32

ADVANTAGES OF ATTENTION

- Ability to focus on significant part of the sentence
- Ability to peek into source sentence
- Reduces the problem of vanishing gradient
- Alignments are found automatically during the training process
- Improves NMT performance for alignment

DIFFERENT TYPE OF NMT



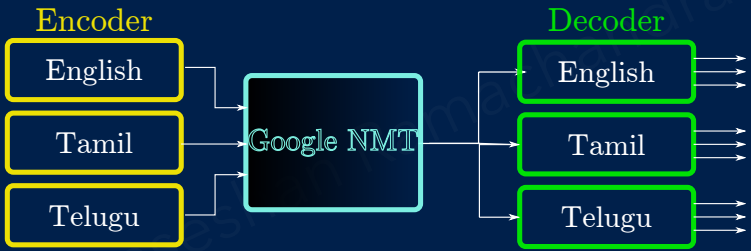


NMT FROM GOOGLE - ZERO-SHOT TRANSLATION

- Moved away from maintaining Seq2Seq model for every pair of languagess
- A single system that translates between any two languages even in the absence of the training corpus for these two languages
  - Assume that only examples of Japanese-English and Korean-English translations are available, Google found hat the multilingual NMT system trained on this data could actually generate reasonable Japanese-Korean translations.
  - Is it trained create the Interlingua?
  - Is the system learning a common representation or a translational knowledge?

Ref: Johnson et el. 2016, "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation"

ZERO-SHOT TRANSLATION



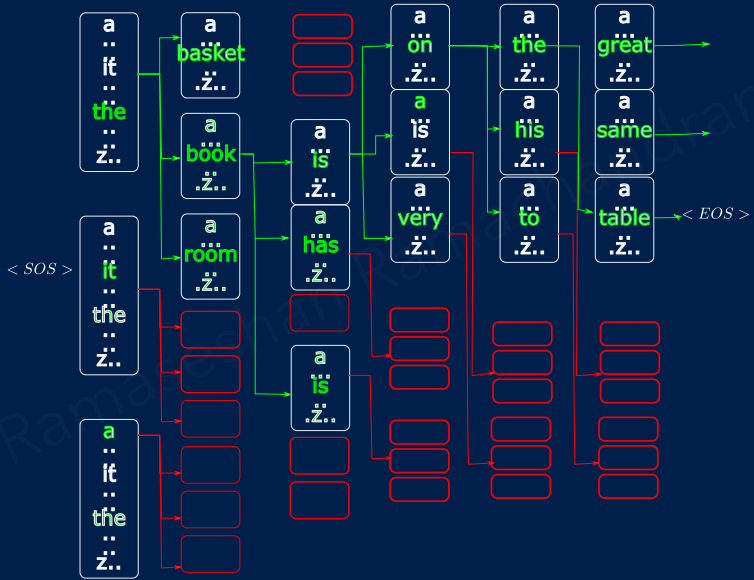
BEAM SEARCH

Beam search is a heuristic search algorithm that selects a few candidate hypothesis from $|V|$. It reduces memory requirement by using only a $M < |V|$ candidates using a score.

- Maintain M candidates/hypothesis at each time step - $C_t = (x_1^1, ..x_t^1)...(x_1^M ...x_t^M)$
- Compute C_{t+1} by expanding C_t and keeping the best M candidates
- $\tilde{C} = \bigcup_{i=1}^{M} C_{t-1}^i$

Typical Beam width of size 5-10 used in NMT. The bilingual evaluation understudy (BLEU) scores computed using Beam search using B=5-10 are comparable

BEAM SEARCH - BEAM WIDTH = 3



BEAM SEARCH EXAMPLE

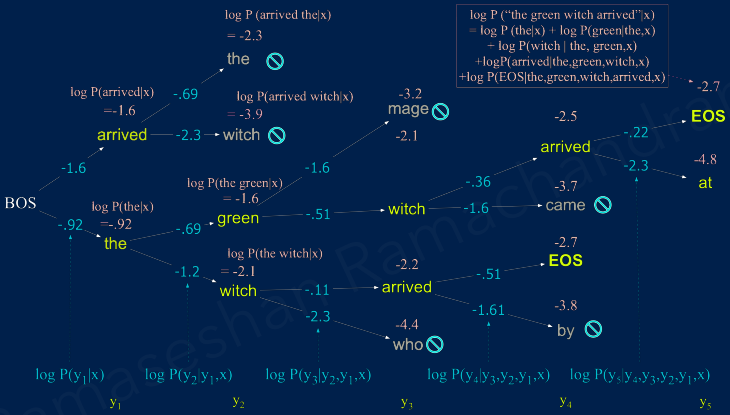


Figure: Scoring for beam search decoding with a beam width of k= 2. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step

BEAM SEARCH SUMMARY

1. Use all possible partial translations - exhaustive search
2. Beam size, $b = 1$ - greedy search - Words are predicted until the $< EOS >$ is found
3. $b > 1$ - several hypotheses
4. Each hypothesis will be produced until the $< EOS >$ is found
5. Each hypothesis will have a translation
6. The length of all hypothesis may not be the same
7. We could use different ***terminate*** conditions
   - Fixed time steps
   - Compute until $< EOS >$ is reached for each hypothesis
8. Use either log probability or product of conditional probability to find the scores for each hypothesis that maximizes
   - $P(y_1, y_2, ...y_m|\mathbf{X}) = \prod_{t=1}^{T} P(y_t| < SOS >, \ldots, y_{t-1}, \mathbf{X})$
   - $P(y_1, y_2, ...y_m|\mathbf{X}) = \sum_{t=1}^{T} \log P(y_t| < SOS >, \ldots, y_{t-1}, \mathbf{X})$

DIFFICULTIES WITH HUMAN EVALUATION OF MT

- Human evaluations are extensive but expensive
- A need for quick, reusable, inexpensive method that correlates highly with human evaluation
- Many aspects of translation,including adequacy and fluency should be considered during the automatic evaluation
- Automatic evaluation is a boon to developers of MT
- Two important aspects required for automatic evaluation
  1. A good metric
  2. A good/gold standards as references

THE IDEA

- Many translations possible for a given sentence
- A good translator identifies a good candidate using adequacy and fluency

The main idea is to use a weighted average of variable length phrase matches against the reference translations[6]

Candidate 1: **It is a guide to action which ensures that the military always obeys the commands of the party**

Candidate 2: **It is to insure the troops forever hearing the activity guidebook that party direct**

Reference: **It is a guide to action that ensures that the military will for ever heed Party commands**

If many words and phrases are shared between the candidate and the reference translations, then it a good choice

Can n-grams help in matching the words and phrases?

UNIGRAM PRECISION

C1: It is a guide to action which ensures that the military always **obeys** the commands of the party

R1: It | is | a | guide | to | action | that | ensures | that | the | military | will forever heed | Party | commands

R2: It is the guiding principle which guarantees the military forces always being under the command | of | the | Party.

R3: It is the practical guide for the army to heed the directions of the party.

Unigram precision $= \frac{17}{18}$

C2: It is to **insure** the **troops** forever **hearing** the **activity guidebook** that party **direct**

R1: It | is | a guide to action | that | ensures that | the | military will | forever | heed Party commands

R2: It is the guiding principle which guarantees the military forces always being under the command of the | Party |.

R3: It is the practical guide for the army always to heed the directions of the party.

Unigram precision $= \frac{8}{14}$

MODIFIED- N-GRAM PRECISION

Compare the number of n-grams in the candidate and in the reference translation
Penalize models that produces many words of the same type

- Count the number of times a word occurs in any single reference translation
- $\text{Count}_{clip} = \min(\text{Candidate Count}, \text{Maximum Reference Count})$

Refer the previous slide for the examples

Modified unigram precision for C1 $= \frac{17}{18}$ • Modified unigram precision for C2 $= \frac{8}{14}$

C3: **the the the the the the the**
R4: **the cat is on the mat**

Unigram precision $= \frac{7}{7}$

Modified unigram precision $= \frac{2}{7}$

Modified bigram precision $= 0$

Modified Unigram precision defines the adequacy of the translation, while modified bigram precision matches the fluency of the translation

MODIFIED BIGRAM PRECISION

(It,is),(is,a),(a,guide),
(guide,to),(to,action),
(action,which),(which,ensures),
(ensures,that),(that,the),
(the,military),(military,always),
(always,obeys),(obeys,the),
(the,commands),(commands,of),
(of,the),(the,party)

Modified bigram precision for C1 $= \frac{10}{17}$

```
(It,is),(is,a),(a,guide),(guide,to),
(to,action),(action,that),(that,ensures),
(ensures,that),(that,the),(the,military),
(military,will),(will,forever),(forever,heed),
(heed,Party),(Party,commands)

(It,is),(is,the),(the,guiding),
(guiding,principle),(principle,which),
(which,guarantees),(guarantees,the),
(the,military),(military,forces),(forces,always),
(always,being),(being,under),(under,the),
(the,command),(command,of),
(of,the),(the,Party)

(It,is),(is,the),(the,practical),(practical,guide),
(guide,for),(for,the),(the,army),
(army,always),(always,to),(to,heed),
(heed,the),(the,directions),
(directions,of),(of,the),(the,party)
```

MODIFIED BIGRAM PRECISION - CANDIDATE 2

(It,is),(is,to),(to,insure),
(insure,the),(the,troops),
(troops,forever),(forever,hearing),
(hearing,the),(the,activity),
(activity,guidebook),
(guidebook,that),(that,party),
(party,direct)

Modified bigram precision for C2 $= \frac{1}{13}$

```
(It,is),(is,a),(a,guide),(guide,to),
(to,action),(action,that),(that,ensures),
(ensures,that),(that,the),(the,military),
(military,will),(will,forever),(forever,heed),
(heed,Party),(Party,commands)

(It,is),(is,the),(the,guiding),
(guiding,principle),(principle,which),
(which,guarantees),(guarantees,the),
(the,military),(military,forces),(forces,always),
(always,being),(being,under),(under,the),
(the,command),(command,of),
(of,the),(the,Party)

(It,is),(is,the),(the,practical),(practical,guide),
(guide,for),(for,the),(the,army),
(army,always),(always,to),(to,heed),
(heed,the),(the,directions),
(directions,of),(of,the),(the,party)
```

COMBINING N-GRAM PRECISIONS

- Modified n-gram precisions decay exponentially as n increases[6]
- BLEU uses a average log with a uniform weights to tackle the decay problem to get a score equivalent to the geometric mean of modified n-gram precisions
- $c < r$ inflates the precision
- A brevity penalty (BP) is introduced when $c \leq r$

$$BP = \begin{cases} 1, & \text{if } c > r \\ \exp(1 - \frac{r}{c}), & \text{if } c \leq r \end{cases}$$

where r is the effective length of the reference corpus and c is the length of the candidate sentence

BLEU score is obtained by

$$\text{BLEU} = \text{BP}.\exp \sum_{n=1}^{N} w_n \log p_n \tag{79}$$

where N is the n-gram size (BLEU uses 4-gram by default),w_n is the weights associated with unigram,bigram,trigram and 4-grams,and p_n is the modified precision score of the test corpus. Here,$\sum_{n=1}^{N} w_n = 1$. One option for $w_n = \frac{1}{N}$

$$p_n = \frac{\sum_{c \in C} \sum_{ngrams \in C} \text{Count}_{clip}(ngrams)}{\sum_{c \in C} \sum_{ngrams \in C} \text{Count}(ngrams)} \tag{80}$$

BLEU - DEMO

BLEU Demo

APPLICATIONS OF BLEU

BLEU is designed as a corpus measure

- Machine translation
- Image labeling
- Text summarization
- Speech recognition

OTHER METRICS

- NIST - National Institute of Standards and Technology - based on BLEU
- METEOR - Metric for Evaluation of Translation with Explicit ORdering
  - Uses stemming and synonymy matching
- WER - Word Error Rate
  - Uses edit distance (Levenshtein distance)
  - Finds minimum number of edit operations such as insertion,deletions or substitutions,needed to change the candidate sentence into the reference sentence
- GLEU - Google BLEU
  - Correlates well with BLEU,and works with sentence level translation

Large Language Model

LARGE LANGUAGE MODELS

- LLMs are typically trained on massive datasets of text and code
- Large language models transform natural language processing
- Capable of generating human-like text
- Learn patterns, context, and semantic relationships in data
- GPT/BARD are state-of-the-art models using LLMs
- Revolutionizing various applications across industries

TYPICAL APPLICATIONS OF LLMS

- Natural language understanding
- Natural language generation
- Machine translation
- Question answering
- Text summarization
- code generation

LANGUAGE UNDERSTANDING

- Comprehends and interprets text with high accuracy
- Captures meaning, context, and nuances in language
- Understands complex queries and generates relevant responses
- Enables chatbots and virtual assistants to communicate effectively
- Enhances machine translation and language comprehension tasks

CREATIVE CONTENT GENERATION

- Generates coherent and contextually appropriate text.
- Creates engaging and original stories, articles, and scripts.
- Assists with content creation for marketing and advertising.
- Provides innovative solutions for writing, gaming, and entertainment.
- Expands creative possibilities for artists and content creators.

KNOWLEDGE EXPANSION

- Accesses vast amounts of information from the Internet.
- Processes and summarizes complex texts and articles.
- Helps in research, data analysis, and decision-making.
- Provides insights and answers to specific questions.
- Enables continuous learning and knowledge sharing

.

REAL WORLD APPLICATIONS

- Improves customer service through chatbots and support systems.
- Personalizes user experiences in e-commerce and recommendation systems.
- Enhances medical diagnostics and healthcare decision-making.
- Powers virtual assistants and voice-activated technologies.
- Transforms education and online learning platforms.

HOW DO LARGE LANGUAGE MODELS WORK?

- Uses transformer architecture
- Learns long-range dependencies between words
- Generalize the statistical regularities of human language

WHO ARE MY NEIGHBOURS?

Traditional domain in an windowed (ramp or otherwise) approach[7]

- Frequency count - cooccurence
- Correlation

SVD Domain

- The transformation of the incidence matrix into SVD domain leads to the define the relationship between two words based on the magnitude of the diagonal matrix Σ

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^T \tag{81}$$

The low-rank approximation captures as much information/energy as possible with respect to the word relationships

GloVe Domain

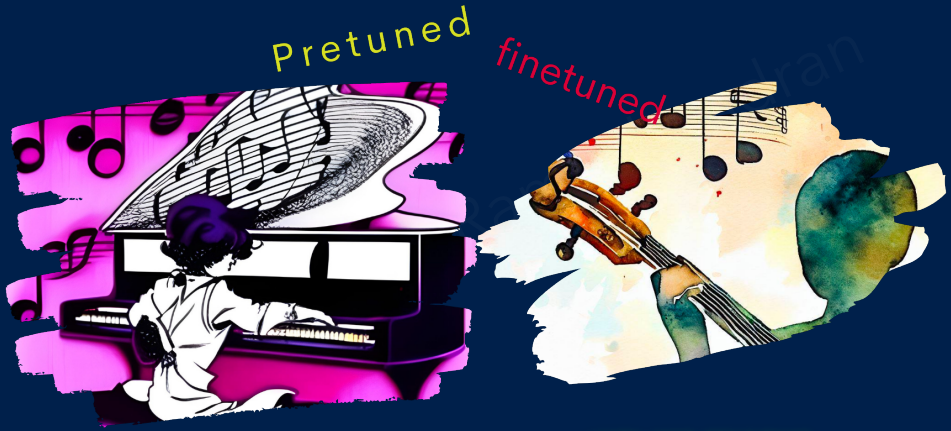
- Ratio of the co-occuring word probabilities are transformed into word embeddings[8]

LANGUAGE MODELS

Useful in

- Auto-generation of sentences
- Auto-completion of sentences
- Creating contextualized word-vectors
- Machine Translation

PRE-TUNED AND FINE-TUNED



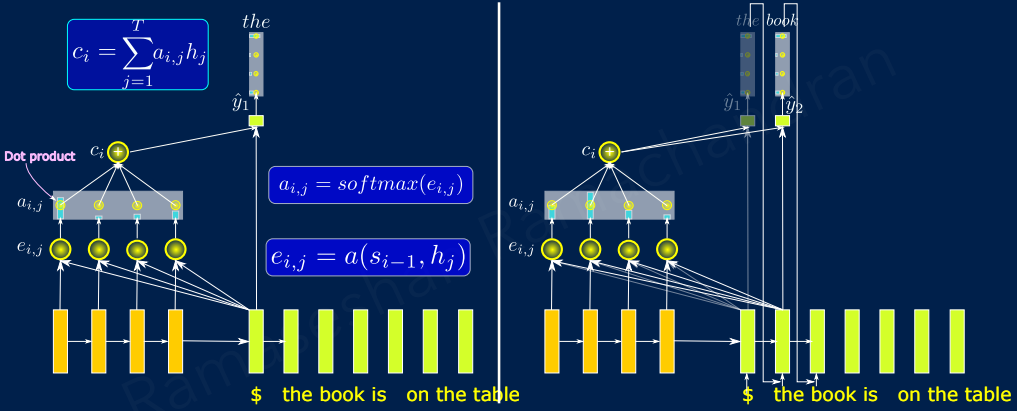
TRANSFER LEARNING

- Models are trained on a specific task - Word-embedding $\rightarrow$ build contextual embedding
- Use the learned weights for another task - attention in machine learning - transfer the hidden layer (contextual information) and use it for another task - mostly done using bidirectional LSTM - Embeddings from Language Model (ELMO)[9]
  - Train using a sentence - forward and reverse Sequence
  - Concatenate hidden layers of (1)
  - Depending on the task on hand, multiply each (2) vector by a scale factor
  - Sum (2) to get the contextualized embedding

    $\rightarrow \mathrm{ELMO}_k = \gamma \sum_{j=0}^{L} \mathrm{SoftMax}_j h_{j,k}$

    where $h_{j,k} = \mathrm{BiLSTM}(w_{1:T}, k)$, k is the index of the word and $1 \leq j \leq L$

ATTENTION[4] I

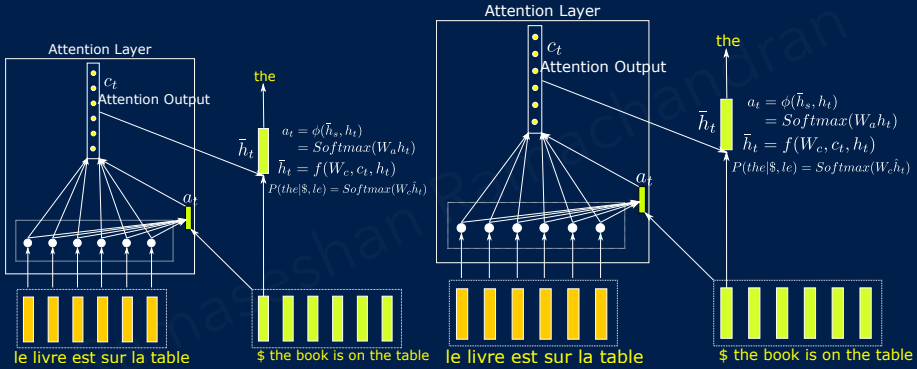


$e_{i,j}$—attention score

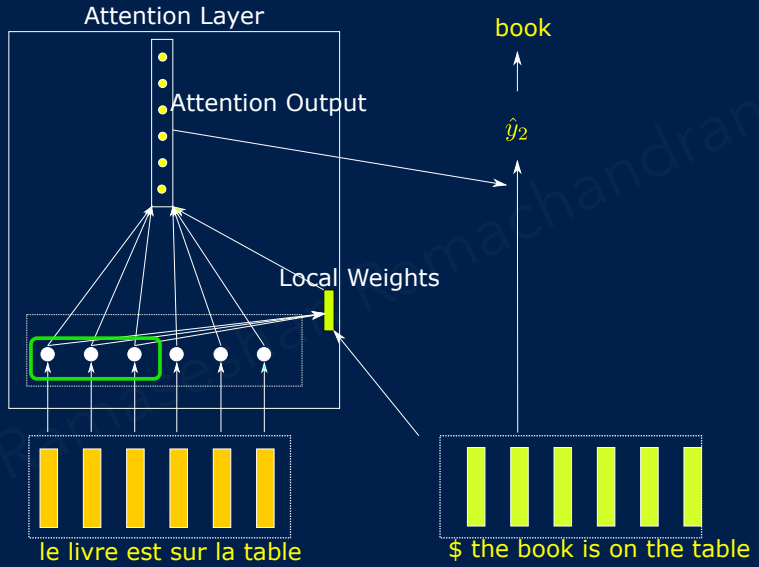
$a_{i,j}$—attention distribution

c_i—attention output

ATTENTION[4] II



ATTENTION[4] III



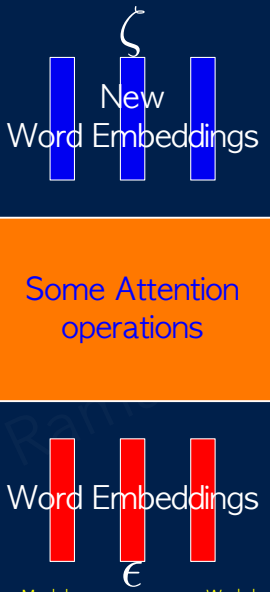
DO I KNOW MORE ABOUT MYSELF AND MY NEIGHBORS?

- How much should I be influenced by my neighbors?
- How do I capture information about my neighbors?
- Ideally a model creates a word vector that has enough information about its neighbors irrespective of how well it is connected with them statistically

ANY CHANGE REQUIRED?

- Is it possible to encode the time-series differently?
- How can the complexities of the recurrent models?
- Is it possible to look at each embedding differently?

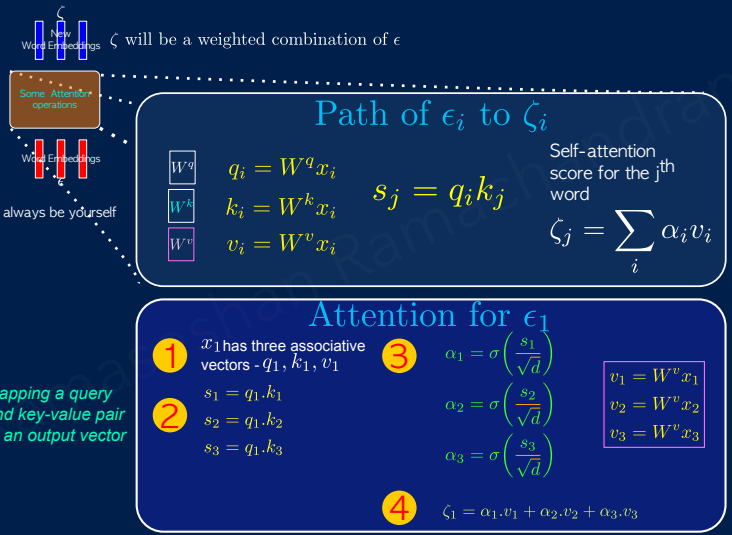
IDEAL ENCODING OF EMBEDDING



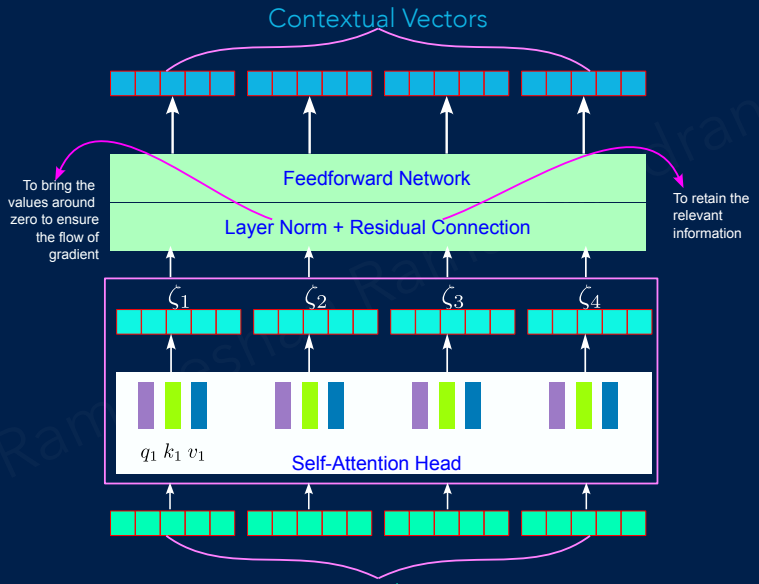
ζ will be the weighted combinations of ϵ

Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

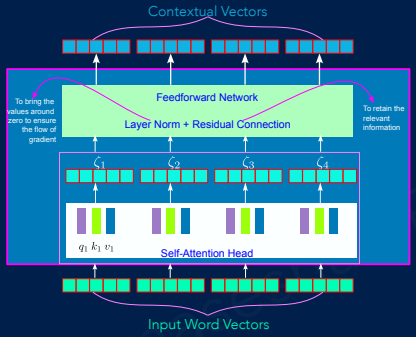
SELF-ATTENTION LAYER

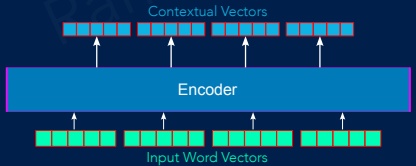


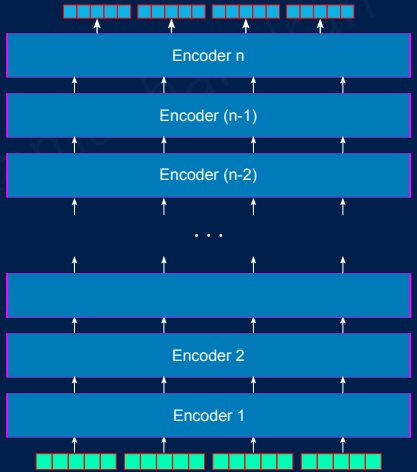
SINGLE TRANSFORMER LAYER



TRANSFORMER ENCODER







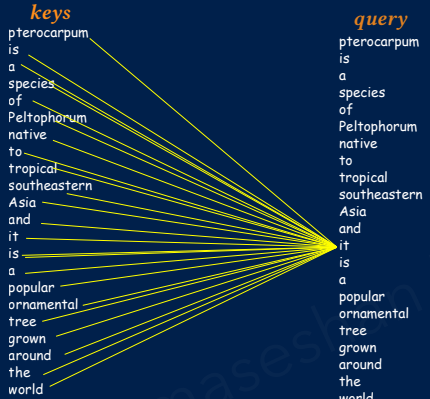
SELF-ATTENTION

Peltophorum pterocarpum is a species of **Peltophorum**, native to tropical southeastern Asia and **it** is a popular ornamental tree grown around the world

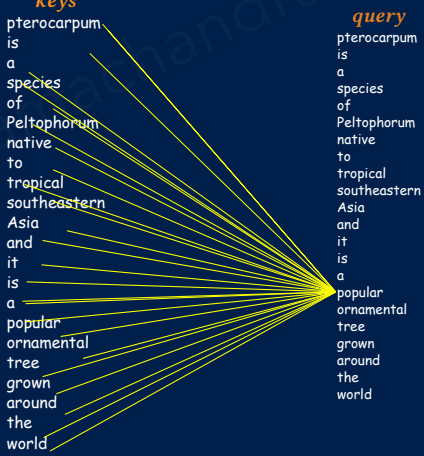
What does it refer to?

Self-attention allows us to associate *it* with Peltophorum pterocarpum

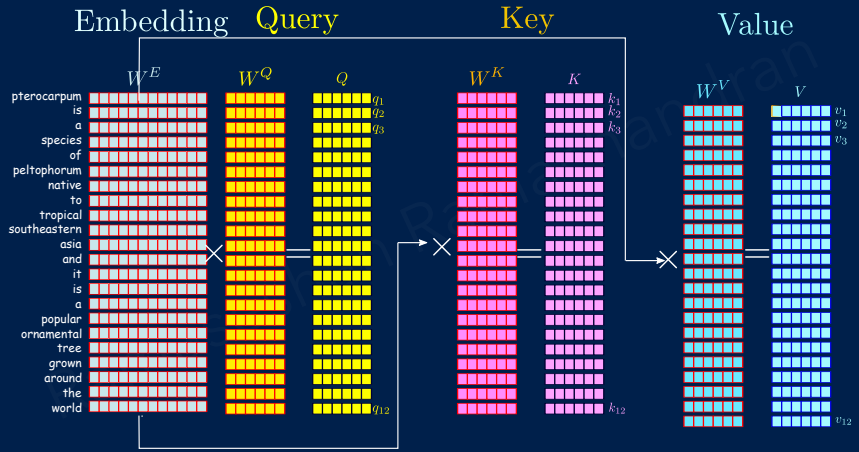
SELF-ATTENTION - WORD LEVEL



Self-attention allows each word to align itself to other words using their positions and looks for clues for a better contextul encoding

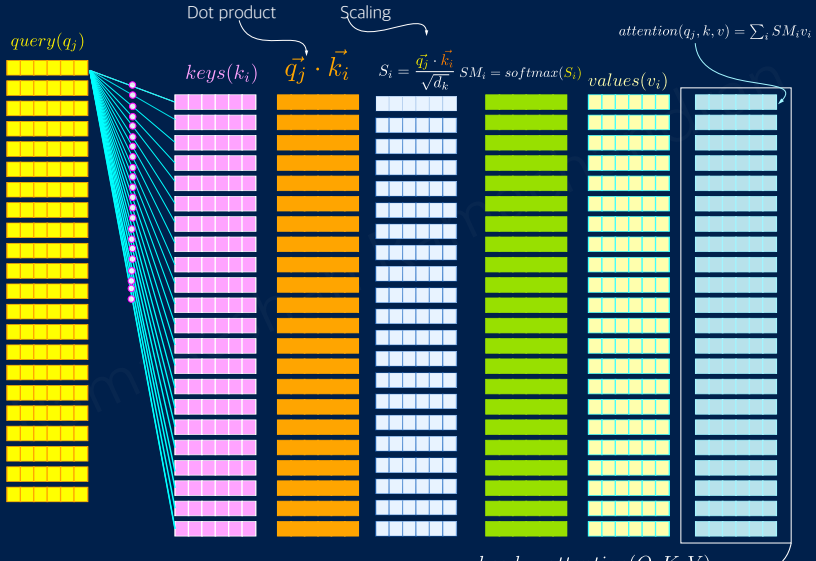


CREATING QUERY, KEY AND VALUES



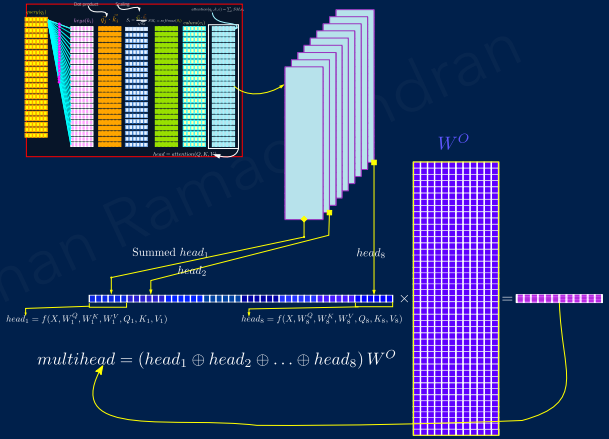
W^Q, W^K and W^V are learned during the training process

COMPUTING ATTENTION(Q,K,V)

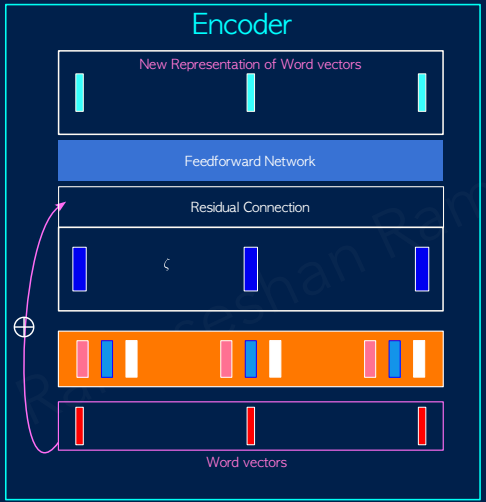


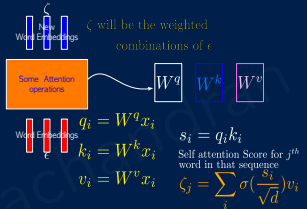
MULTI-HEAD ATTENTION

The summed attention score, $\text{attention}(q, k_i, v_i) = \sum_i \frac{e^{\frac{q \cdot k_i}{\sqrt{d_k}}}}{\sum_j e^{\frac{q \cdot k_j}{\sqrt{d_k}}}} v_i$, has information about all the word, but may have more information about words that have similarity scores higher than others in that context.



SELF-ATTENTION





ATTENTION COMPUTATION: A SIMPLIFIED EXAMPLE I

Let's compute attention using a simplified example with word embeddings.
We'll use a small set of embeddings for demonstration purposes.
Assume we have the following word embeddings for simplicity:

- **Query (q)**: Embedding for "who"
- **Keys (k)**: Embeddings for "is", "the", "best"
- **Values (v)**: Same as Keys for simplicity, but in practice, they could be different.

ATTENTION COMPUTATION: A SIMPLIFIED EXAMPLE II

Let's say each embedding is a 3-dimensional vector:

$$q : [0.1, 0.2, 0.3]$$
$$k_1 \text{ (is)} : [0.4, 0.5, 0.6]$$
$$k_2 \text{ (the)} : [0.2, 0.1, 0.3]$$
$$k_3 \text{ (best)} : [0.6, 0.7, 0.8]$$
$$v_1 \text{ (is)} : [0.4, 0.5, 0.6]$$
$$v_2 \text{ (the)} : [0.2, 0.1, 0.3]$$
$$v_3 \text{ (best)} : [0.6, 0.7, 0.8]$$

ATTENTION COMPUTATION: A SIMPLIFIED EXAMPLE III

Dot Product:

$$q \cdot k_1 = 0.1 \cdot 0.4 + 0.2 \cdot 0.5 + 0.3 \cdot 0.6 = 0.04 + 0.1 + 0.18 = 0.32$$
$$q \cdot k_2 = 0.1 \cdot 0.2 + 0.2 \cdot 0.1 + 0.3 \cdot 0.3 = 0.02 + 0.02 + 0.09 = 0.13$$
$$q \cdot k_3 = 0.1 \cdot 0.6 + 0.2 \cdot 0.7 + 0.3 \cdot 0.8 = 0.06 + 0.14 + 0.24 = 0.44$$

Exponentiation:

$$e^{0.32} \approx 1.38$$
$$e^{0.13} \approx 1.14$$
$$e^{0.44} \approx 1.55$$

Softmax (Normalization):

Sum of exponents: $1.38 + 1.14 + 1.55 = 4.07$

ATTENTION COMPUTATION: A SIMPLIFIED EXAMPLE IV

Weights:

$$\alpha_1 = \frac{1.38}{4.07} \approx 0.34$$
$$\alpha_2 = \frac{1.14}{4.07} \approx 0.28$$
$$\alpha_3 = \frac{1.55}{4.07} \approx 0.38$$

Weighted Sum:

$$\begin{aligned}
\text{Attention}(q,k,v) &= \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 \\
&= 0.34 \cdot [0.4, 0.5, 0.6] + 0.28 \cdot [0.2, 0.1, 0.3] + 0.38 \cdot [0.6, 0.7, 0.8] \\
&= [0.136 + 0.056 + 0.228, 0.17 + 0.028 + 0.266, 0.204 + 0.084 + 0.304] \\
&= [0.42, 0.464, 0.592]
\end{aligned}$$

ATTENTION COMPUTATION: A SIMPLIFIED EXAMPLE V

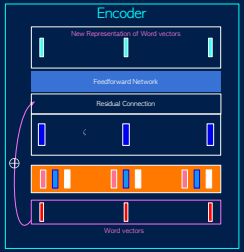
The attention output for the query "who" given the keys and values "is", "the", "best" is approximately:

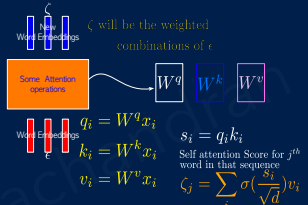
$$[0.42, 0.464, 0.592]$$

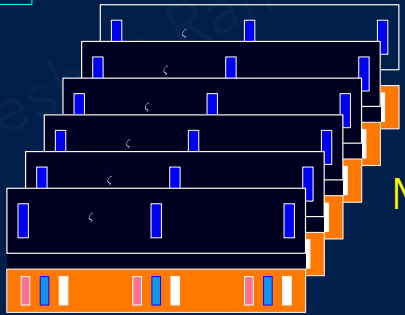
This vector represents a weighted combination of the input values, where the weights are determined by how relevant each key is to the query, as computed through the attention mechanism.

Note: This is a highly simplified example. In real-world scenarios, embeddings are much higher dimensional, and the attention mechanism is applied across all positions in the sequence, often in parallel for efficiency.

MULTIHEAD ATTENTION







Multihead

POSITIONAL ENCODING

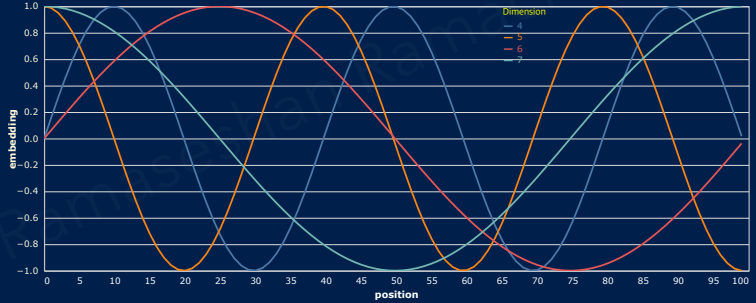
- RNN models encode the time signal in a sequential manner
- Positional encoding is important for contextual learning
- Transformers a separate positional vector is added to the embedding
- Positional encoding should be unique, not just a scalar
- positional values should be bounded
- Positional values between any two positions should be consistent
- Sentence length should not be a constraint
- Deterministic

POSITIONAL ENCODING

$$\vec{pe}_t^{\,i} = \begin{cases} \sin(\omega_k \cdot t), \text{ if } i = 2k \\ \cos(\omega_k \cdot t), \text{ if } i = 2k+1 \end{cases}$$

where $\omega_k = \frac{1}{10000^{(2k/d)}}$.

This positional vector will be a pair of $\sin$ and $\cos$ values and i is the index of the dimension

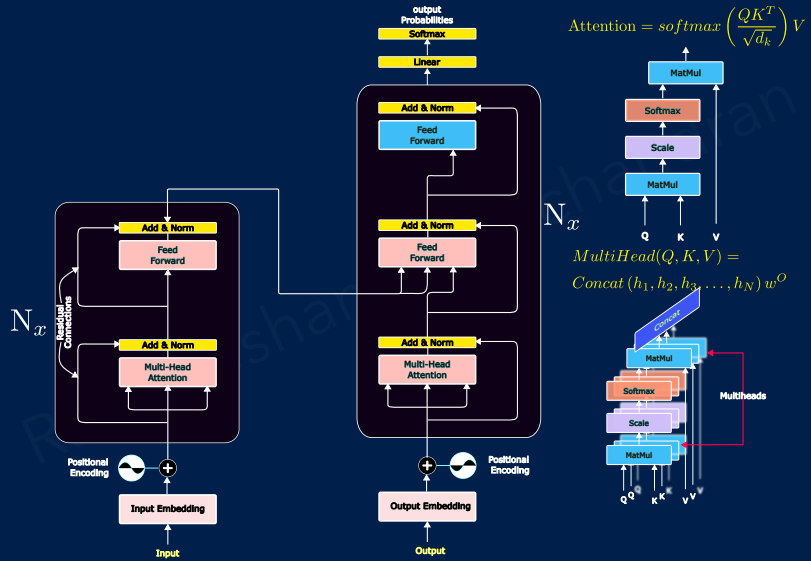


1. Good reference for Transformer - The Illustrated Transformer, Jay Alammar
2. The Annotated Transformer, Austin Huang et al.

POSITIONAL ENCODING - PYTHON CODE

```
def positional_encoding(position, d_model):
  angle_rates = 1 / np.power(10000, (2 * (np.arange(d_model) // 2
  position = position.reshape(-1, 1)
  angle_rates = angle_rates.reshape(1, -1)
  angle_rates = position * angle_rates
  angle_rates[:, 0::2] = np.sin(angle_rates[:, 0::2])
  angle_rates[:, 1::2] = np.cos(angle_rates[:, 1::2])
  return angle_rates
```

TRANSFORMER ARCHITECTURE



NUMBER OF PARAMETERS IN A TRANSFORMER

Embedding Layer - $N_{emb} = V \times D$
where N_{emb} is the number of parameters in the embedding layer, V is the vocabulary size and D is the embedding dimension

Multi-Head Attention -
$N_{attn} = 3 \times D \times d_k \times h + h \times d_v \times d_k + 2 \times h \times d_k(biasses)$
where N_{attn} is the number of parameters in multi-head attention, D is the embedding dimension, d_k is the dimension of the key vector, h is the number of heads and d_v is the dimension of the value vector

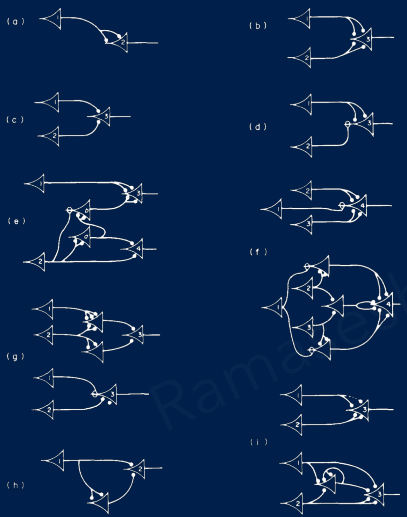
Feed-Forward Network -
$N_{ffn} = 2 \times D \times H + H + H \times D$
where N_{ffn} is the number of parameters in the feed-forward network, D is the embedding dimension and H is the hidden dimension

Layer Norm - $N_{ln} = 2 \times D$
where N_{ln} is the number of parameters in the layer normalization, D is the embedding dimension

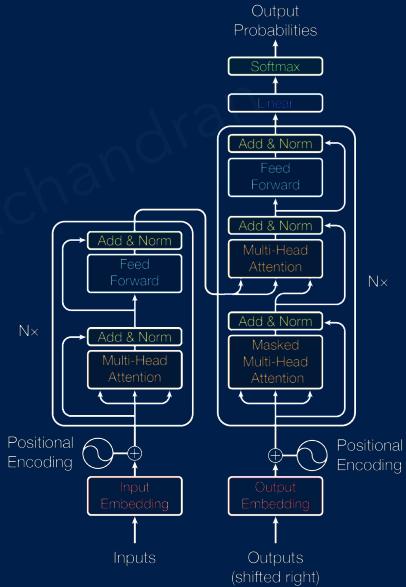
Total Number of Parameters per Encoder/Decoder Layer -
$N_{layer} = N_{attn} + N_{ffn} + N_{ln}$

Total Number of Parameters in the Transformer - $N_{total} = N_{emb} + N_{layer} \times L$
where L is the number of layers

MCCULLOCH-PITTS VS TRANSFORMER



1943 -> 2017



WHY MULTIHEAD?

QUESTION: WHAT DO TRANSFORMER HEADS SPECIALIZE AT?

- All attention heads receive the same input embeddings.
- Heads start with random weights.
- How do they specialize in attending to different aspects of input?

RANDOM INITIALIZATION OF WEIGHTS

- Each head has unique weight matrices (W_Q, W_K, W_V, W_O).
- Random initialization leads to diverse transformations of input.
- Ensures heads start with distinct attention patterns.

SPECIALIZATION THROUGH TRAINING

- Heads specialize via backpropagation during training.
- Loss function guides heads to focus on:
  - Syntactic relationships (e.g., dependencies between words).
  - Semantic relationships (e.g., context or meaning).
  - Positional or structural patterns.

MULTI-HEAD ATTENTION MECHANISM

- Each head computes its own attention scores and representation.
- Representations are concatenated and linearly transformed.
- Enables model to aggregate diverse perspectives, such as:
  - Word-level dependencies.
  - Sentence-level context.
  - Positional relevance.

LAYER STACKING

- Outputs from one layer (all heads) serve as inputs to the next.
- Subsequent layers refine or build on earlier attention patterns.
- Hierarchical structure deepens specialization.

IMPLICIT COMPETITION

- Heads compete to minimize loss:
  - Redundant attention patterns provide diminishing returns.
  - Specialization maximizes each head's contribution.
- Encourages efficient use of capacity.

EXPERIMENTAL FINDINGS

- Certain heads consistently specialize across tasks and datasets:
  - Positional encoding.
  - Syntax parsing.
  - Semantic roles.
- Some heads act as:
  - **Generalists:** Broad attention across sequences.
  - **Specialists:** Focused attention on specific patterns.

SUMMARY

- Specialization arises from:
  - Random initialization of weights.
  - Differentiated learning through backpropagation.
  - Layer interactions and multi-head architecture.
- Leads to a division of labor among attention heads.
- Key to the power and flexibility of transformer models.

PURPOSE OF RESIDUAL CONNECTIONS

- **Mitigate Vanishing Gradients:**
  - Ensure gradients flow through deeper networks.
- **Enable Deeper Architectures:**
  - Support the use of multiple layers without degradation.
- **Easier Optimization:**
  - Simplify the training of complex models.

MECHANISM OF RESIDUAL CONNECTIONS

- **Post-Attention:**

$$\text{Output} = \text{LayerNorm}(\text{Input} + \text{SelfAttention}(\text{Input}))$$

- **Post-Feed-Forward:**

$$\text{Output} = \text{LayerNorm}(\text{PreviousOutput} + \text{FeedForward}(\text{PreviousOutput}))$$

BENEFITS OF RESIDUAL CONNECTIONS

- **Stability in Training:**
  - Prevents exploding or vanishing gradients in deep models.
- **Feature Preservation and Enhancement:**
  - Maintains input features while enhancing them through transformation.
- **Increased Model Expressiveness:**
  - Allows the model to learn complex representations.

WORD EMBEDDING

1. Word embedding is fundamental to Deep Learning of NLP
2. Contextual word embedding is now used in most of the downstream applications

CLOZE TEST

According to a report in yesterday's newspaper (1)––– police dog was taken to Raj Bhavan (2)––– Monday. This was to trace the (3)––– of the "very important horse" which (4)–––– reported missing on Sunday. The dog picked (5)–––– the scent on some traces of (6)––– and ran a few yards before losing the (7)––––. The police have launched a vigorous (8)–––– into the whole affair. They have (9)–––– the services of a forensic expert, (10)–––– fingerprint expert and a photographer. (11)–––– are now fourteen horses at Raj Bhavan (12)–––– are kept in a large shed near the gate.

1	once	a	new
2	at	next	on
3	police	killers	dogs
...	...	...	...
11	There	We	So
12	who	were	which

The purpose of the Cloze test is to measure the reading comprehension of a student with respect to grammar, usage, vocabulary, and contextual understanding.

MASKED LM

Objectives

- Predict the masked word using the context
- Combine left and right contexts to predict the masked word
- Use sentence pairs to predict the next sentence
- Use the trained model for token-level and sentence-level tasks

- Mask some of the tokens from the input
- Predict the original token using its context
- Use Bidirectional deep transformer model fuse the left and right context and develop a contextual representation

HOW DO WE MASK?

- 15% of the words are randomly masked
- Perform the following operation on the the i^{th} token
  1. Replace with a $<MASK>$ 80% of the time
  2. Replace with a random token 10% of the time
  3. Retain the original 10% of the time

BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS (BERT) - ARCHITECTURE

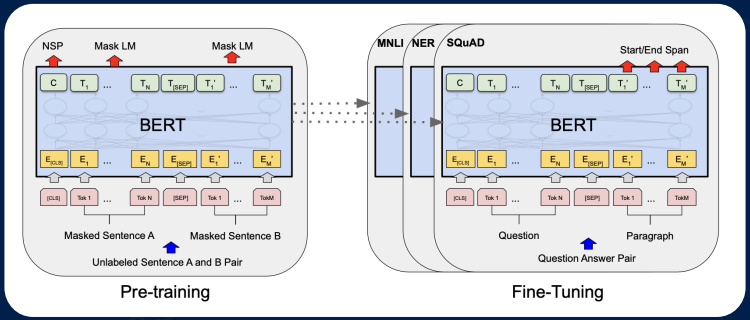


Figure: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architec- tures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).Next Sentence Prediction (NSP) And Perturbation is a method used to assess or enhance the performance of BERT-like models by altering or analyzing their training dynamics, specifically in tasks involving next sentence prediction.

DOES BERT PREDICT THE "NEXT WORD" FOR [MASK]?

- **Unlike GPT-style models, BERT does not predict the next word in the sequence**
- Instead, BERT predicts the **original word at the masked position**, leveraging:
  - **Bidirectional context** from both preceding and following tokens.
- **Example:**
  - Input: `The [MASK] sat on the mat.`
  - BERT considers both sides of `[MASK]`:
    - Context: "The" and "sat on the mat."
    - Prediction: "cat."
- **Why is this effective?**
  - Bidirectional prediction enables **contextual understanding**.
  - Excels in tasks like **question answering** and **natural language inference**.

GENERAL PURPOSE TRANSFORMER - GPT

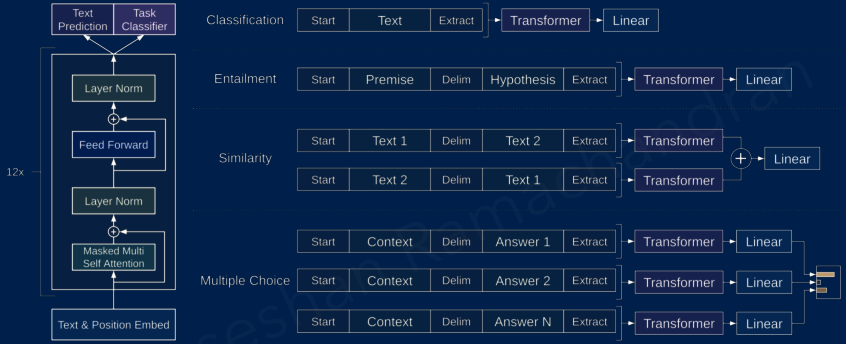
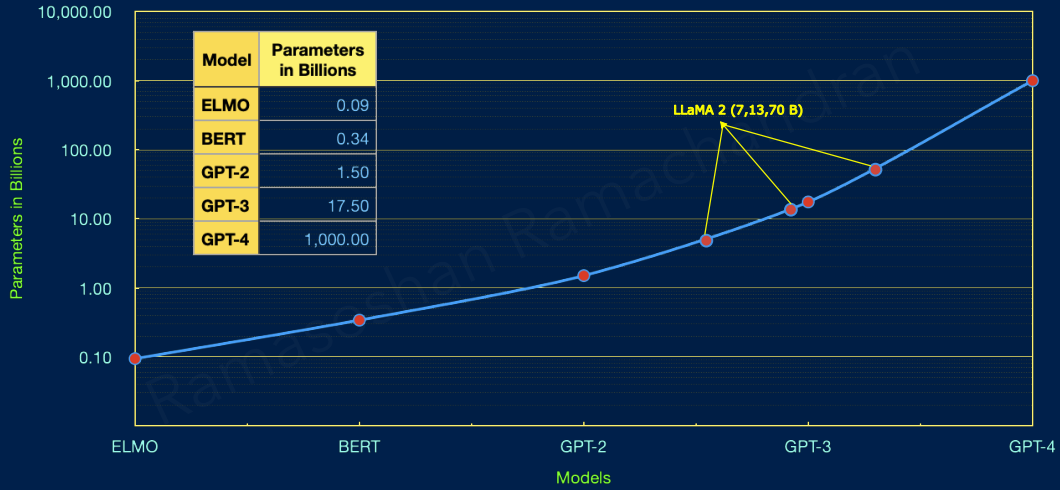


Figure: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer

PARAMETERS OF LARGE LANGUAGE MODEL



Model	Parameters in Billions
ELMO	0.09
BERT	0.34
GPT-2	1.50
GPT-3	17.50
GPT-4	1,000.00

LLAMA 2 OVERVIEW

- Open-source foundation model: Llama 2
- Fine-tuned chat models for conversational AI
- Improves upon Llama 1 with new architectures and training methods
- Released by Meta AI
- Goals: Advance conversational AI, promote open research

KEY CONTRIBUTIONS

- New architecture: Modified Transformer with improved attention mechanisms
- Enhanced training methods: Supervised and reinforcement learning
- Large-scale pre-training: 15 trillion parameters, 32K tokens
- Fine-tuning strategies: Task-specific adaptation
- Open-source release: Model weights, code, and datasets

TRAINING OF LLAMA 2-CHAT

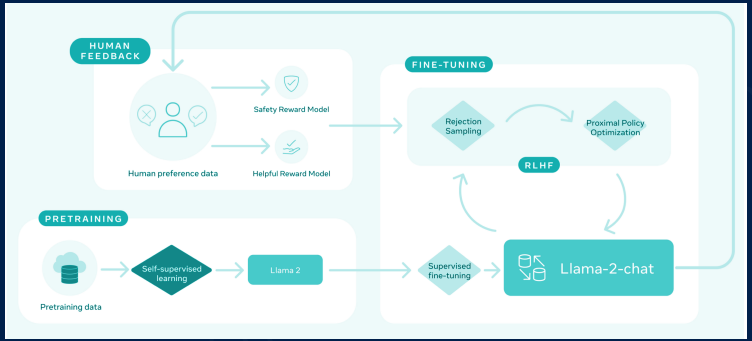


Figure: Thisprocessbeginswiththepretraining of Llama 2 usingpublicly availableonlinesources. Followingthis,wecreateaninitialversionof Llama 2-Chat throughtheapplication of supervised fine-tuning. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (RLHF) methodologies, specifically through rejection sampling and Proximal Policy Optimization(PPO).ThroughouttheRLHFstage,theaccumulationof iterativerewardmodelingdata in parallel with model enhancements is crucial toensure

IMPACT AND RESULTS

- State-of-the-art performance on conversational benchmarks
- Improved conversational flow and coherence
- Enhanced ability to understand context and nuance
- Demonstrates potential for real-world applications
- Opens doors for future research in conversational AI

SAMPLE TEXT GENERATION - MINIGPT

```
const LEARNING_RATE: f64 = 0.0003;
const BLOCK_SIZE: i64 = 128;
const BATCH_SIZE: i64 = 64;
const EPOCHS: i64 = 100;
const SAMPLING_LEN: i64 = 4096;
```

```
Config {
    vocab_size: 130,
    n_embd: 128,
    n_head: 8,
    n_layer: 8,
    block_size: 128,
    attn_pdrop: 0.1,
    resid_pdrop: 0.1,
    embd_pdrop: 0.1,
};
```


for the gbr genome determines a comprehensive comparison from existing quality was commonly important to a lung w fusion if they can be a compared x-rayed, with the blood scenario of the lesion of t-cell support probability.in this later, the next for the first room of responses to materials and infection has classified a retrospective response of younger genes [138] .

LLAMA

Llama-2 was pretrained on publicly available online data sources. The fine-tuned model, Llama-2-chat, leverages publicly available instruction datasets and over 1 million human annotations.

- LLaMA stands for Large Language Model Meta AI.
- Llama 2 pretrained models are trained on 2 trillion tokens

Here is the link to the paper

MODEL DETAILS

	Training Data	Parameters	Con-tent Length	GQA[4]	Tokens	Learn-ing Rate
Llama 2	Publicly available online data	7B	4k	NA	2.0T	3.0×10^4
Llama 2	″	13B	4k	NA	2.0T	3.0×10^4
Llama 2	″	70B	4k	YES	2.0T	1.5×10^4

GQA

[4]Grouped-Query Attention

Decoding Techiques

PRETRAINING

- Learns linguistic patterns & world knowledge from massive text datasets
- Trains models to predict text sequences, building foundational language understanding

FUNDAMENTALS OF LLM DECODING

- **Core Components:**
  - Context window (prior generated text)
  - Vocabulary probability scores
  - Decoding strategy algorithm
- Autoregressive generation: $P(w_t|w_{<t})$
- Key challenge: Balance between:
  - Accuracy vs. Creativity
  - Coherence vs. Diversity

GREEDY DECODING

$$\hat{w}_i = \arg\max_{w \in V} P(w_i | w_{<i})$$

- **Advantages:**
  - Maximizes local probability
  - High coherence
- **Limitations:**
  - Repetitive outputs
  - Lacks creativity
  - No error recovery

TEMPERATURE SCALING

$$P(w_t) = \frac{\exp(z_i/T)}{\sum \exp(z_j/T)}$$

- **Effects:**
  - $T = 1$: Normal sampling
  - $T > 1$: Increased randomness
- Applications:
  - $T < 1$: Technical writing
  - $T > 1$: Creative writing

TOP-K & TOP-P SAMPLING

Top-k:

- Fixed candidate count
- $k = 1 \Rightarrow$ greedy
- Risk of including implausible tokens

Comparison:

- Top-p generally more flexible
- ChatGPT uses top-p

Top-p (Nucleus):

- Dynamic candidate selection
- Cumulative probability threshold
- $\sum p_i \geq p$

MIN-P SAMPLING

Adaptive threshold calculation:

$$\text{Threshold}_t = \max(P_t) \times \text{min\_p}$$

- Retain tokens where $p_i \geq$ Threshold
- Advantages:
  - Dynamic vocabulary selection
  - Maintains model confidence
  - Reduces low-quality outputs
- Typical min_p values: 0.05-0.2

DECODING STRATEGY PROPERTIES

Method	Temp.	Stochastic
Greedy	No	No
Beam Search	No	No
Temp. Sampling	Yes	Yes
Top-k	Yes	Yes
Top-p	Yes	Yes
Min-p	Yes	Yes

- Key considerations:
  - Application requirements
  - Output quality needs
  - Computational resources

Sentiment Analysis

The process of computing and classifying opinions from an unstructured text

CLASSIFICATION OF EMOTION

- Emotion: Most of living react in the same way to something happening outside or inside them - anger, sadness, joy, fear, shame, pride, elation, desperation
- Mood Definition: A diffuse affect state characterized by a change in subjective feeling, often without apparent cause
  - *Intensity*: Low intensity but relatively long duration
  - *Examples*: Cheerful, gloomy, irritable, listless, depressed, buoyant
- Interpersonal Stance: Affective stance taken toward another person in a specific interaction
  - *Examples* : Distant, cold, warm, supportive, contemptuous
- Attitudes: Relatively enduring, affectively colored beliefs, preferences, and predispositions
  - *Components*: Beliefs, preferences, and predispositions
  - *Objects*: Objects or persons

OTHER NAMES

- Opinion mining
- Evaluation Analysis
- Appraisal Assessment
- Attitude mining
- Emotion extraction
- Subjectivity analysis
- Aspect extraction
- Affect extraction
- Review mining
- …

CORE CONCEPTS

- Sentiment reflects emotional tone of text
- Polarity: positive, negative, or neutral
- Emotion detection for joy, anger, sadness
- Aspect-based sentiment: specific components focus
- Analyzed at document, sentence, phrase level

FORMAL DEFINITION

- An opinion consists of a target and sentiment.
- Target (g): Entity or aspect opinion is about.
- Sentiment (s): Positive, negative, neutral, or numeric score.
- Example: 1-5 stars or sentiment polarity.
- Polarity types: Positive, negative, or neutral sentiment.
- Targets can include entire entities or specific aspects.
- Example: Target in "Canon G12" is camera.
- Example: Target in "Picture quality" is image aspect.

TARGET DOMAIN EXAMPLES

Sentiments on the attributes of products

- Example: Mobile phone case - design, fitment, price, durability, protection, shell type (poly carbonate, plastic), color, weight, etc.

Services

- Banks, restaurants, sports centers, fitness centers, repair shops, etc.

Individuals

- Example: Fitness instructors/trainers, teachers,

Public Issues

- Political, non-political, governance, etc.

Social media

- Monitoring social media for issues, products, trends, etc

Events

- Music events, workshops, Topics

...

SAMPLES

Product details

Smartphone · Single SIM · iOS · 5G · Wireless Charging · Dual Lens · With OLED Display · Facial Recognition · 2778 x 1284 · Water Resistant

The camera system receives its most significant boost yet, with next-generation technology that captures far more information, while the super-slick Pro Motion display ensures a smoother experience when surfing, gaming, or simply checking social media.

Reviews

Long battery life (478) | Easy to use (249) | Heavy (215) | Performs well (190) | Attractive (150) | Easy to set up (110) | Large display (110) | Quality display (86)

★★★★★ 7 December, 2021

THE BOTTOM LINE Apple's iPhone 13 Pro Max is the ultimate mobile content creation machine, with the best camera and longest battery life of any iPhone. The ultimate phone for photo and video creators! The iPhone 13 Pro Max (starting at $1,099) is the ultimate professional content creator's phone. It combines Apple's excellent camera algorithms and software support with true two-day battery life for a massive phone that's always ready to realize your dreams. While the standard iPhone 13 (starting at $799) seems to be the best choice for most people, with a terrific balance of size, power, battery life, and price, the iPhone 13 Pro Max is a terrific alternative for heavy users and artists, with its killer cameras and beautiful buttress of a battery. The iPhone 13 Pro ... More

chantal.v · Review provided by influenster.com

All reviews

SENTIMENTS ON ATTRIBUTES

Customer reviews

4.2 out of 5

2,071 global ratings

5 star	57%
4 star	23%
3 star	10%
2 star	4%
1 star	6%

How are ratings calculated?

By feature

Value for money	4.2
Sturdiness	4.1
Durability	4.1
Sheerness	4.0

See less

Reviews with images

See all customer images

Read reviews that mention

value for money | yellow within | good quality | within a week | fits perfectly | within few days | started turning | perfect fit | fit for iphone | back cover | overall good | sturdy back | attracts a lot

Top reviews

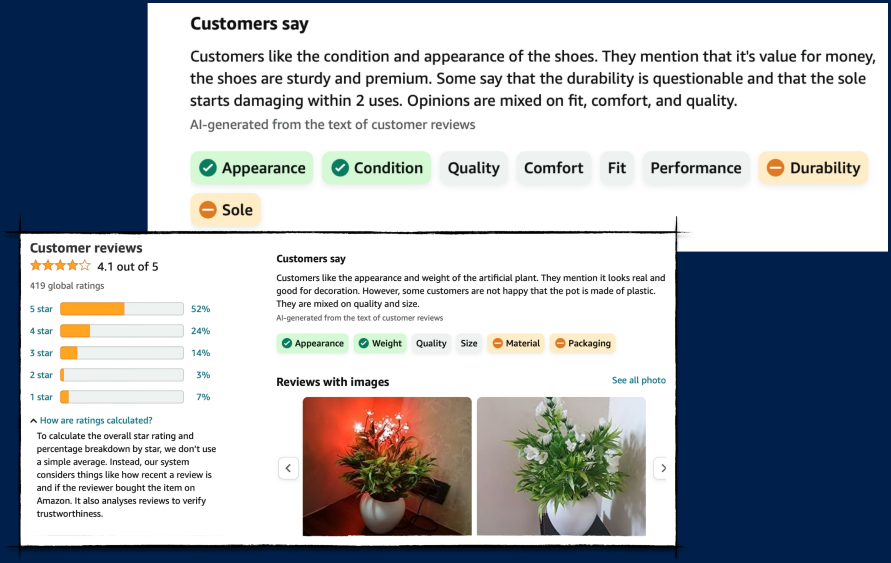
Top reviews from India

Amazon Customer

Not sure about the Durability, But so far, so good!

Reviewed in India on 20 October 2022

SENTIMENTS ON ATTRIBUTES 2/2



SAMPLE SENTENCES

- When you don't want to spend a whole lot of cash but want a great deal….
- This is the shop to buy from
- It is a very cute case
- Cannot argue with the price or appearance
- The jewels do fall off
- It is a beautiful phone case but it is also hard to remove
- Arrived broken and very flimsy
- Fits perfectly but needs a little attention at the installation
- The design and color combination makes the case simple yet elegant, and not too bold and flashy Don't believe that these screen protectors have glue in them

SENTIMENT ANALYSIS AS A CLASSIFICATION PROBLEM

- Sentiment analysis can be treated as a classification task.
- Assigns input text into predefined sentiment categories.

  Examples of sentiment categories
  - Positive
  - Negative
  - Neutral
- Classification output can also be represented on a numerical scale.

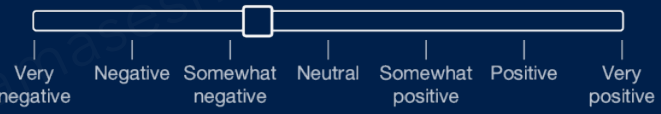


Figure: The labeling interface

WHY USE A SCALE?

- Provides finer granularity than simple positive, negative, or neutral labels.
- Enables better differentiation between sentiments of similar types.
- Allows for more nuanced analysis, e.g., distinguishing between slightly positive (+5) and highly positive (+20).
- Facilitates advanced tasks like regression-based sentiment prediction.

APPROACHES TO SENTIMENT ANALYSIS

- Rule-based: lexicons and manual rules
- Machine learning: Naive Bayes, SVM, others
- Deep learning: CNNs, RNNs, Transformers
- Trade-offs: simplicity vs. computational complexity

CHALLENGES

- Context impacts interpretation of sentiment
- Sarcasm and irony are difficult to detect
- Sentiment varies across different domains
- Handling ambiguous and mixed sentiments is tricky
- Multilingual and multimodal sentiment needs innovation

APPLICATIONS

- Analyze customer reviews for feedback insights
- Monitor political opinions and public reactions
- Assess sentiment in healthcare and treatments
- Predict audience reactions to entertainment content
- Monitor brand sentiment on social media

TOOLS AND LIBRARIES

- TextBlob: easy sentiment classification library
- VADER: social media sentiment lexicon-based tool
- NLTK: preprocessing and sentiment classification
- Commercial APIs: IBM Watson, AWS Comprehend

TAXONOMY

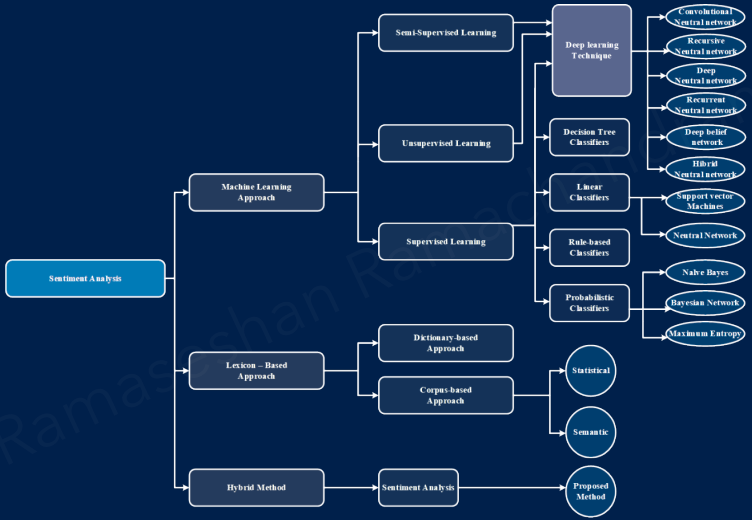


Figure: Taxonomy of sentiment analysis techniques

TRADITIONAL APPROACHES

- Lexicon based Approach
- Naive Bayes Classifier
- Support Vector Machines (SVM)
- Effective in high-dimensional spaces for binary or multi-class classification
- Logistic Regression
- Decision Trees and Random Forests
- K-Nearest Neighbors (KNN)

FUTURE DIRECTIONS

- Combining text, voice, and video sentiment
- Real-time sentiment analysis for live events
- Summarizing sentiment across vast datasets
- Enhancing model explainability and transparency

Modern AI Engines

OVERVIEW OF CONVERSATIONAL AI

What is Conversational AI?

- AI-driven technology that enables machines to **simulate human-like conversations**.
- Utilizes **Natural Language Processing (NLP)**, **Machine Learning (ML)**, and **contextual awareness**.
- Applications in customer service, virtual assistants, healthcare, and more.

Key Components:

- **Speech Recognition** – Converts spoken language into text.
- **Natural Language Understanding (NLU)** – Interprets user intent.
- **Dialogue Management** – Maintains conversation context.
- **Natural Language Generation (NLG)** – Produces human-like responses.

CHATBOT SYSTEMS

What are Chatbots?

- Programs that simulate conversations with users.
- Used for customer support, automation, and personal assistance.

Types of Chatbots:

- **Rule-Based Chatbots** – Follow predefined scripts and rules.
- **AI-Powered Chatbots** – Use ML and NLP to learn and adapt responses.

RULE-BASED VS. AI-POWERED CHATBOTS

Rule-Based Chatbots:

- Follow fixed decision trees.
- Limited in handling variations in user input.
- Best for structured, repetitive queries.

AI-Powered Chatbots:

- Use NLP and ML to improve over time.
- Can handle unstructured and dynamic conversations.
- Provide more natural, human-like interactions.

GEMINI: AN ADVANCED AI MODEL

What is Gemini?

- A state-of-the-art AI model developed by Google DeepMind.
- Designed for multimodal tasks, integrating text, images, and other data.
- Capable of advanced reasoning, complex task execution, and real-time adaptability.

Key Features:

- Multimodal understanding and generation.
- Enhanced contextual reasoning.
- Scalability across different applications.

CHATGPT: A CONVERSATIONAL AI MODEL

What is ChatGPT?

- A language model developed by OpenAI, based on the GPT (Generative Pre-trained Transformer) architecture.
- Trained on vast amounts of text data to generate human-like responses.
- Used in chatbots, content creation, coding assistance, and more.

Key Features:

- Context-aware conversational capabilities.
- Fine-tuned for various applications.
- Continuous learning through reinforcement learning and fine-tuning.

TECHNOLOGY BEHIND GEMINI AND CHATGPT

Underlying Technologies:

- **Transformer Architecture** – Both models rely on transformers for language understanding and generation.
- **Large-Scale Pretraining** – Trained on extensive datasets to enhance contextual awareness.
- **Multimodal Capabilities** – Gemini integrates text, images, and other modalities, whereas ChatGPT focuses primarily on text-based interactions.
- **Reinforcement Learning** – ChatGPT leverages reinforcement learning from human feedback (RLHF) to improve response quality.
- **Scalability** – Both models are designed to run on high-performance computing clusters to handle complex queries efficiently.

BIAS IN LARGE LANGUAGE MODELS

What is Bias in LLMs?

- LLMs learn from vast datasets that may contain **societal, historical, or systemic biases**.
- These biases can be **reflected, amplified, or perpetuated** in generated outputs.

Types of Bias:

- **Social Bias** – Gender, race, and cultural stereotypes.
- **Political Bias** – Favoring particular ideologies or perspectives.
- **Confirmation Bias** – Reinforcing existing views rather than diversifying perspectives.
- **Representation Bias** – Underrepresentation of minority groups in training data.

CHALLENGES IN LARGE LANGUAGE MODELS

Key Challenges:

- **Bias and Fairness** – Ensuring neutral and unbiased responses.
- **Hallucinations** – Generation of **false or misleading** information.
- **Data Privacy** – Risk of memorizing and exposing **sensitive** information.
- **Computational Costs** – High energy consumption and **expensive infrastructure**.
- **Explainability** – Lack of clear interpretability behind model outputs.
- **Ethical Concerns** – Spread of misinformation, deepfakes, and harmful misuse.

Mitigation Strategies:

- **Diverse training data** to minimize biases.
- **Fact-checking mechanisms** to detect hallucinations.
- **Fine-tuning with ethical guidelines** to enhance fairness.

Bias and Challenges

REFERENCES I

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. 2016. URL: https://www.deeplearningbook.org.

[2] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. Cambridge, MA, USA, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735. URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[3] KyungHyun Cho et al. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* abs/1409.1259 (2014). arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". English (US). In: *arXiv* (2014).

REFERENCES II

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015.

[6] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040.

[7] Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. "An improved model of semantic similarity based on lexical co-occurrence". In: *Communications of the ACM* 8.627-633 (2006), p. 116.

REFERENCES III

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[9] Matthew E. Peters et al. "Deep contextualized word representations". In: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

Thank you

LinkedIN
ramaseshanr@gmail.com