Workshop on Language Models (Day 1 - Session 2)

Ramaseshan Ramachandran

IDEAL LANGUAGE MODEL

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

THE LANGUAGE MODEL

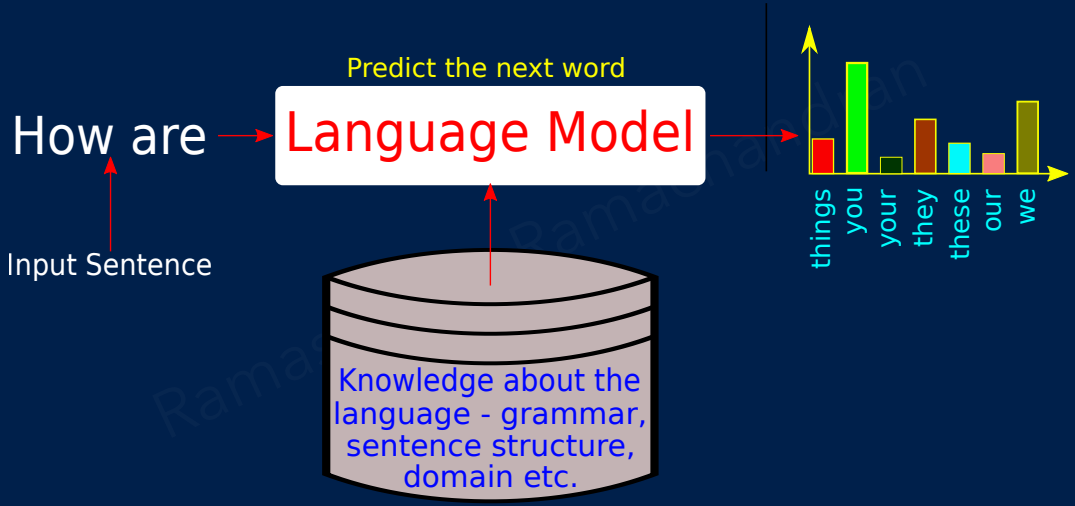
- Natural language sentences can be described by parse trees which use the morphology of words, syntax and semantics
- Probabilistic thinking - finding how likely a sentence occurs or formed, given the word sequence.
- In probabilistic world, the Language model is used to assign a probability $P(W)$ 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 deterministic 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 determining 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 - $\mathrm{p}(word|next) = \frac{\mathrm{p}(next, word)}{\mathrm{p}(next)}$

LANGUAGE MODELS

We want the language models to answer

- How likely is the following word?
- How probable is it that a sequence of English words constitutes good English?
- How probable is the second sentence, given that the preceding sentence is contextually relevant?
- How likely is it that the word selected is the most appropriate choice?
- How probable is it that the sentence is considered fluent in terms of syntax and semantics?

  $\mathbf{p}(\text{the cat sat on the mat}) > \mathbf{p}(\text{the mat sat on the cat})$
- Probabilistic assertions are about possible worlds

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

PROBABILISTIC MODEL

Representation of probability score of every occurrence of ***n-gram***

DEFINITION

- A probability model describes the likelihood of outcomes in an event.
- Let V be a finite vocabulary, $\triangleleft$ and $\triangleright$ be start and stop symbols, and $|V|$ denote V's size.
- Let W be infinite sequences of words from V, starting with $\triangleleft$ and ending with $\triangleright$. A language model is a probability distribution of a random variable X taking 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

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

Task: Given the context, find the next word using

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

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.
Ideally, the probability of P(The cat roars) computed 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.
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 (words).
- 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

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

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. Only last k words are included in history (older words are irrelevant) - k^{th} order Markov model

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

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 in the given context and normalizing by the total count of that context.

count$(w_{1:i-1}, w_i)$ represents the number of times the context $w_{1:i-1}$ is followed by w_i, while count$(w_{1:i-1})$ is the count of the context itself.

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:

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

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

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

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:

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

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

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

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

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
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
```

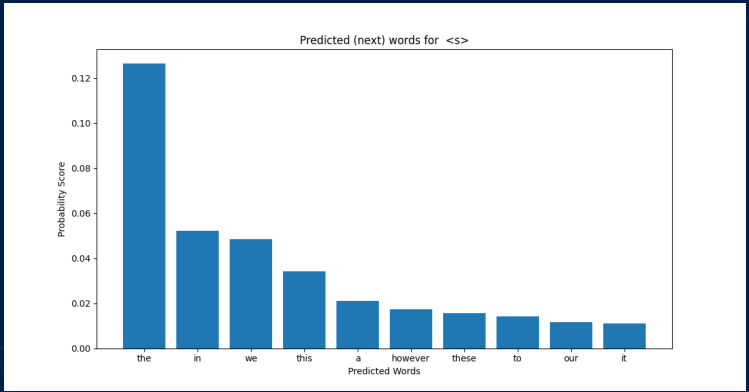
```
<defaultdict, len() = 226126>
  > special variables
  > function variables
  > '<s>': <defaultdict, len() = 27215>
  > 'bovine': <defaultdict, len() = 209>
  > 'viral': <defaultdict, len() = 1541>
  > 'diarrhoea': <defaultdict, len() = 121>
  > 'bvd': defaultdict(<function build_bigram_model.<locals>.
  v 'has': <defaultdict, len() = 1423>
      > special variables
      > function variables
        'been': 0.31419770372444694
        'made': 0.004620554466535984
        'shown': 0.013114907122187996
        'declared': 0.0014001680201624195
        'achieved': 0.0009334453467749463
        'proved': 0.0018668906935498926
        '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')
model = read_model()
```

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 reliable model assigns a high probability to a text written in genuine English. This can also be measured using cross entropy.

$$H(W) = \frac{1}{n} \log p(w_i | w_{i-1} \tag{28}$$

$$\text{perplexity}(W) = 2^{H(W)} \tag{29}$$

- 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:

$$\begin{aligned} \text{PP} &= \exp\left(-\frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}} \log \text{p}(w_\text{i}|w_{\text{i}-1})\right) \\ &= \exp\left(-\frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}} \log\left(\frac{\text{count}(w_\text{i}, w_{\text{i}-1})}{\text{count}(w_{\text{i}-1})}\right)\right) \end{aligned} \tag{30}$$

where

- PP is the perplexity 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
- $w_{\text{i}-1}$ is the $(\text{i}-1)$th word in the test set
- $\text{count}(w_\text{i}, w_{\text{i}-1})$ is the number of times the bigram $(w_\text{i}, w_{\text{i}-1})$ appears in the training corpus
- $\text{count}(w_{\text{i}-1})$ is the number of times the word $w_{\text{i}-1}$ appears in the training corpus

PERPLEXITY OF A TRIGRAM MODEL

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

$$\text{PP} = \exp\left(-\frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}}\log\left(\frac{\text{count}(w_{\text{i}}, w_{\text{i}-1}, w_{\text{i}-2})}{\text{count}(w_{\text{i}-1}, w_{\text{i}-2})}\right)\right) \tag{31}$$

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:

$$PP = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{count(w_i, w_{i-1}, w_{i-2}, \ldots, w_{i-(n-1)})}{count(w_{i-1}, w_{i-2}, \ldots, w_{i-(n-1)})}\right)\right) \tag{32}$$

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

Let's calculate the perplexity for the unseen sentence "I like rabbits." Smoothing can be applied to "rabbits" since it's unseen in the training corpus. We can either smooth all bigrams or only for unseen words. In this case, we're smoothing only for 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:

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

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

UNSEEN SENTENCE/WORD EXAMPLE III

The perplexity of this unseen sentence is

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

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.
- It becomes apparent in the context of an n-gram language model due to the rapidly growing number of parameters as the value of n increases.

CHALLENGES

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

BACK-OFF AND INTERPOLATION

- Higher and lower order n-gram models exhibit distinct strengths and weaknesses.
- Higher order n-grams are sensitive to a broader context, but their counts are sparse.
- Lower order n-grams consider only a limited context, but their counts are robust.

$$p(w_3|w_1,w_2) = \lambda_1 p_1(w_3) \times \lambda_2 p_2(w_3|w_2) \times \lambda_3 p_3(w_3|w_1,w_2) \tag{33}$$

The lambdas $(\lambda_1, \lambda_2, \lambda_3)$ are weights that determine the contribution of each n-gram order to the final probability estimate. They control how much we trust each level of context.
The key constraint on the lambdas:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{34}$$

EXERCISE

- Extend the program[1,2] to compute the probability and perplexity of a test sentence
  1. Build the language model using corpus
     - Big corpus → Long time to build a model
     - Development → Choose a smaller corpus
     - Testing/Production → use a bigger corpus to build your model
  2. Check the model parameters using the debugger
  3. Once satisfied with the learned model, test your sentences using the model
- **Input**:
  1. A sentence consisting of words in the corpus that you have used for creating the language model
  2. A sentence with one more words that are OOV
- **Output**:Probability and perplexity of the input sentence
- Try this exercise for trigram and 4-gram language models

[1] https://github.com/Ramaseshanr/anlp/blob/master/BigramLM.ipynb
[2] https://github.com/Ramaseshanr/anlp/blob/master/TrigramLM.ipynb

GENERALIZATION

Most living beings have the ability to generalize or transfer information from previous experience so that they can behave appropriately in novel situation

- Global shape
- Local features
- Physical charateristics

  shape, Size, doors, windowa, wheels, etc.
- Recognition from Different Angles
- Prior Knowledge and Experience
  - Pattern completion or fill-in capabilities

Combination of cues, both visual and contextual, allows humans to identify a bus from all directions and with either partial or full views in the 3-D space or in 2-D drawings

ADVANCED PERSPECTIVES ON GENERALIZATION

- Generalization is crucial in both machine learning and scientific research
- It enables computed models or theories to apply knowledge to unseen contexts
- Thi is vital for predictive accuracy beyond training data.

A BROADER VIEW

- Involves formulating universal principles based on specific observations
- Aims to predict phenomena in new scenarios without needing additional evidence for every case
- Predict real-world complexities, ensuring predictions are applicable/meaningful across a wide array of situations

GENERALIZATION IN MACHINE LEARNING

- Refers to a model's ability to predict outcomes on unseen data - not just from the training set
  - Captures the underlying patterns of the data rather than just memorizing the training examples
- Refers to true effectiveness to predict the outcome beyond training performance

TRAINING VS TESTING

- Models are trained on labeled data to find patterns
- Weights and biases model the relationship between inputs and outputs
- The linear combination of features followed by a non-linear activation function creates a generalized model

$$h_i = \sum_{j=1}^{n} w_{ij} x_i$$

- Non-linearity enables the model to learn complex patterns and relationships that cannot be captured by just linear combinations
- This non-linearity is crucial for generalization, as it allows the model to fit a wider range of data distributions

FEATURE ABSTRACTION

- Each neuron learns to recognize specific patterns or combinations of features in the data. The neurons in the hidden layerlearn and abstract complex features from textual data
- As the number of hidden layers increase, the intution is that they capure more nuances of the text, including, grammar, meaning, context, or intricate patterns.
  - The meaning of the pattern such as spill the beans is associated with an idiomatic expression than with a straight-forwrd literal meaning
  - The sarcasm in this sentence Wow, they improved by a whopping 4 runs this time from their lowest! cannot be captured using a single layer of hidden units
  - The word **wow** is usually present in a positive context, but here it does not refer to the positive sentiment
  - A model that relies only on linear operations wouldn't capture these kinds of nuances, as it would attempt to separate words and meanings using linear decision boundaries

TRANSFORMATION OF INPUT DATA

- Hidden layers serve as transformation layers that convert input textual data into abstract representations
  - GloVe and Word2VEc captures semantic similarities and context-aware representations
  - Hidden layers facilitate hierarchical learning where each layer captures increasingly abstract features
  - Each hidden layer builds upon the previous layer's learned features - brings in the deep leaning capabilities

REPRESENTATION LEARNING I

- The concept of representation learning highlights that hidden layers learn to represent input data through transformations that reflect its underlying structure.
- **Contextual Understanding** - Modern NLP models like Transformers learn representations that incorporate context, allowing a word to have different embeddings based on its surrounding words, improving handling of polysemy and ambiguity.
- **Transfer Learning** - Pre-trained language models (e.g., BERT, GPT) are used as a base, and their learned representations are fine-tuned on specific downstream tasks, improving generalization to new NLP problems with minimal task-specific data.

REPRESENTATION LEARNING II

- **Hierarchical Feature Learning**- Neural networks with multiple layers (especially deep models) build hierarchical representations, where initial layers capture lower-level features (like word patterns), and deeper layers capture more abstract semantics, improving model performance on complex tasks.
- **Unsupervised Learning**: Representation learning often leverages unsupervised or self-supervised techniques (e.g., masked language models), enabling models to learn representations from raw text data without requiring labeled examples.

OVERFITTING AND UNDERFITTING

- Occurs when a model memorizes training data, including noise and outliers, leading to poor performance on unseen data.
- Happens when a model fails to learn significant patterns from the training data, causing it to perform poorly on both training and test sets.

GENERALIZATION TECHNIQUES

- **Regularization**: Methods like L1 or L2 regularization introduce penalties for model complexity, reducing overfitting.
- **Cross-Validation**: Splits the dataset into subsets to ensure consistent performance across different samples.
- **Data Augmentation**: Expands the dataset through transformations, helping models learn more robust features by introducing variation in the data.

STANDARD ALGORITHMS

An algorithm is a sequence of instructions to solve a problem

- The steps to solve problems are well defined
- Steps are coded in some ordered sequence to transform the input from one form to another
- Rules are unambiguous
- Sufficient Knowledge is available to fully solve the problem

HUMAN/MACHINE LEARNING

- There are problems whose solutions cannot be formulated using standard rule-based algorithms
- Problems that require subtle inputs cannot be solved using standard algorithmic approach - face recognition, speech recognition, hand-written character recognition, etc.
- Finding Examples and using experience gained in similar situations are useful
- Examples provide certain underlying patterns
- Patterns give the ability to predict some outcome or help in constructing an approximate model
- **Learning** is the key to the ambiguous world

MODEL

- Quantitative expression of the real world observations
- Mathematical Models translate the real world observations into mathematical expressions
- A machine learning model is a mathematical construct trained to recognize patterns in data
- It makes predictions or classifications based on those patterns
- Helps in understanding and interpreting information for decision making/classification
- Data driven decisions under uncertainty
- Model Parameters store vast amount of information captured from the data
  - Lexical, linguistic and semantic knowledge in the case of NLP
- Access the *knowledge* (learned from the data) of the Model by using the context

AN EXAMPLE OF A MODEL

Modeling is about describing the object using a set of mathematical relationships

1.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix}$$

$$\implies x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \cdots \\ a_{n1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ \cdots \\ a_{n2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ a_{3n} \\ \cdots \\ a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix}$$

$$Ax = \sum_{j=1}^{n} a_{ij} x_j$$

2. Another example $\langle w_i, \tilde{w}_k \rangle \approx \log(X_{ik})$

DISCRIMINATIVE MODEL

- A class of models that focuses on learning the decision boundary between different classes or outcomes
- It models the conditional probability distribution of the target variable given the input variables, $\mathbf{p}(\mathbf{y}|\mathbf{x})$.
- In the case of classification, this is $\mathbf{p}(\mathbf{C_k}|\mathbf{x})$
- OR, Learn a function, *discriminant function*, that maps inputs $\mathbf{x}$ directly into decisions

GENERATIVE MODEL

- If a function models input and output into probability distributions, then it is in general known as a generative model
- Given a training data, generate new samples similar to the training samples from the same distribution
  - Language model - generating the next word given the context
  - It is possible to generate synthetic data points (or new sentences) using these distributions

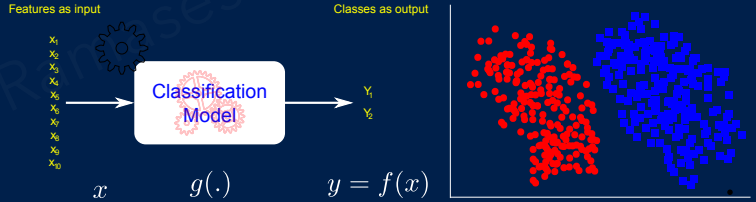
CLASSIFICATION

Classification is the task of assigning predefined dis-joint categories to objects

- Detect Spam emails
- Find the set of mobile phones $<$ Rs.10000 and received 5* reviews
- Identify the category of the incoming document as sports, politics, entertainment or business
- Determine whether a movie review is a positive or negative review

DEFINITION OF CLASSIFICATION

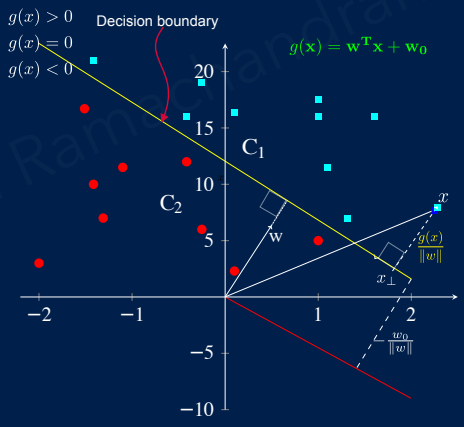
- The input is a collection of records
- Each record is represented by a tuple $(\mathbf{x}, \mathbf{y})$
- $\mathbf{x} = x_1, x_2, \ldots, x_n$ and $\mathbf{y} = y_1, y_2, \ldots, y_n$ are the input features and the classes respectively
- $\mathbf{x} \in \mathbf{R}^2$ is a vector - the set of observed variables
- (x, y) are related by an unknown function. The goal is to estimate the unknown function $g(.)$, also known as a classifier function, such that $g(x) = f(x), \forall x$



WHAT DOES THE CLASSIFIER FUNCTION DO?

Assuming that we have a linearly separable X, the linear classifier function $g(.)$ implements decision rule

- Fitting a straight line to a given data set requires two parameters ($w_0 \, and \, w$)
- The decision rule divides the data space into two sub-spaces - separating two classes using a boundary
- The distance of the boundary from the origin = $\frac{w_0}{||w||}$
- Distance of any point from the boundary = $\mathrm{d} = \frac{g(\mathrm{x})}{||w||}$

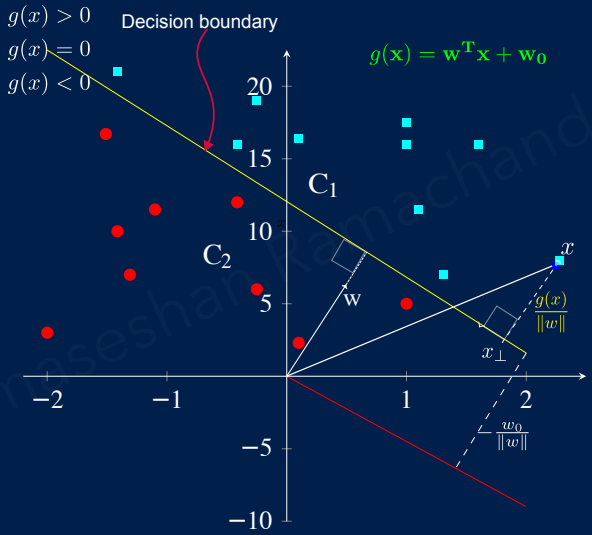


LINEAR MODELS FOR CLASSIFICATION[10.5555/1162264]

The goal of classification is to take a vector $\mathbf{X}$ and assign it to one of the $\mathbf{N}$ discrete classes $\mathbb{C}_n$, where $n = 1, 2, 3, \ldots, N$.

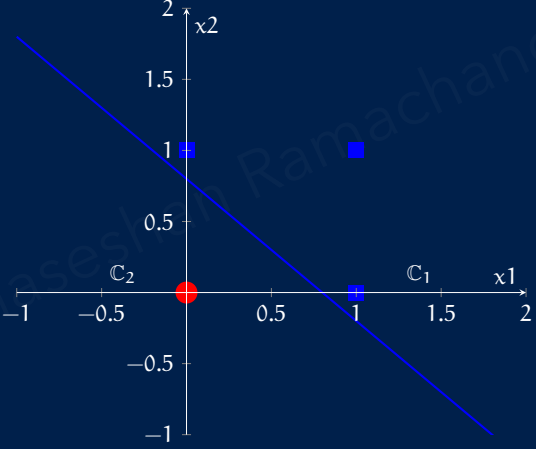
- The classes are disjoint and an input is assigned to only one class
- The input space is divided into *decision regions*
- The boundaries are called as *decision boundaries or decision surfaces*
- In general, if the input space is $\mathbf{N}$ dimensional, then $g(x)$ would define an $N-1$ hyperplane

GEOMETRY OF THE LINEAR DISCRIMINANT FUNCTION



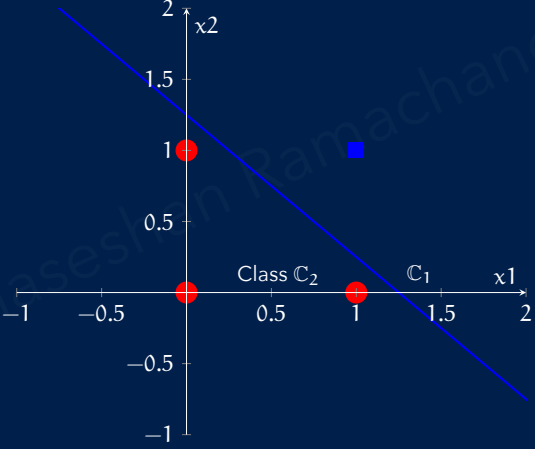
DECISION BOUNDARY FOR OR GATE

The decision regions are separated by a hyperplane and it is defined by $g(x) = 0$. This separates linearly separable classes $\mathbb{C}_1$ and $\mathbb{C}_2$



DECISION BOUNDARY FOR AND GATE

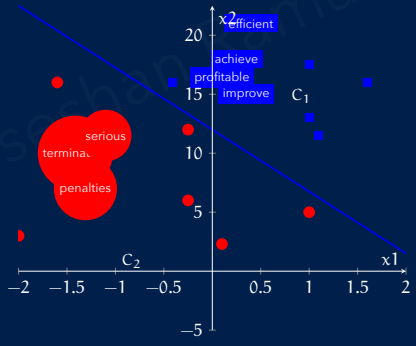
The decision regions are separated by a hyperplane and it is defined by $g(x) = 0$. This separates linearly separable classes $\mathbb{C}_1$ and $\mathbb{C}_2$



DECISION BOUNDARY FOR SENTIMENTS

Let us consider some positive and negative sentiment terms which are contained in two classes $\mathbb{C}_P$ and $\mathbb{C}_N$

$$\mathbb{C}_P = [achieve\ efficient\ improve\ profitable] = +1$$
$$\mathbb{C}_N = [termination\ penalties\ misconduct\ serious] = -1$$



LINEAR REGRESSION

How do we build/develop models?

- Let $x \in R^n$ denote the real valued random data points/features as input
- Let $y \in R$ be the real valued random output with a joint distribution of $p(x, y)$
- Let us find a $f(x)$ to predict y given x
- Let us define a loss function $L(y, f(x))$ for penalizing errors during prediction
- Most common loss function is the squared error loss. The expected error prediction is $EPE(f) = E(y - f(x))^2$
- A point-wise expected error prediction is a conditional expectation and is written as $f(x) = E(y \mid x)$ - this is the regression function
- Linear regression, a common application of least squares, assumes that $f(x)$ can be approximated by a global linear function

LOGISTIC REGRESSION

Supervised Learning

Known relationship (x, y): x is the data and y is the label

Goal: Learn a the relationship and identify a function to map $x \rightarrow y$ **Model**: An optimized learning algorithm to learn the relationship

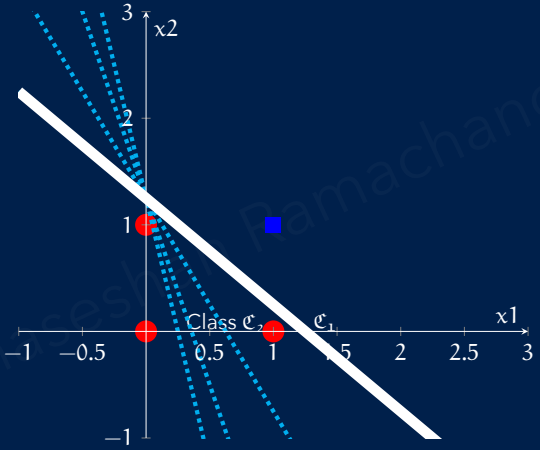
Examples: Classification, regression, sentence generation, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

Known relationship: Only data x. Input-output relationship is not known

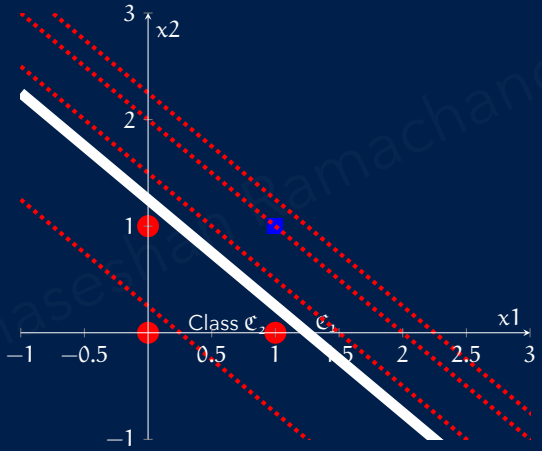
Examples:Clustering, PCA, dimensionality reduction

DECISION BOUNDARY- VARIATION OF w_j

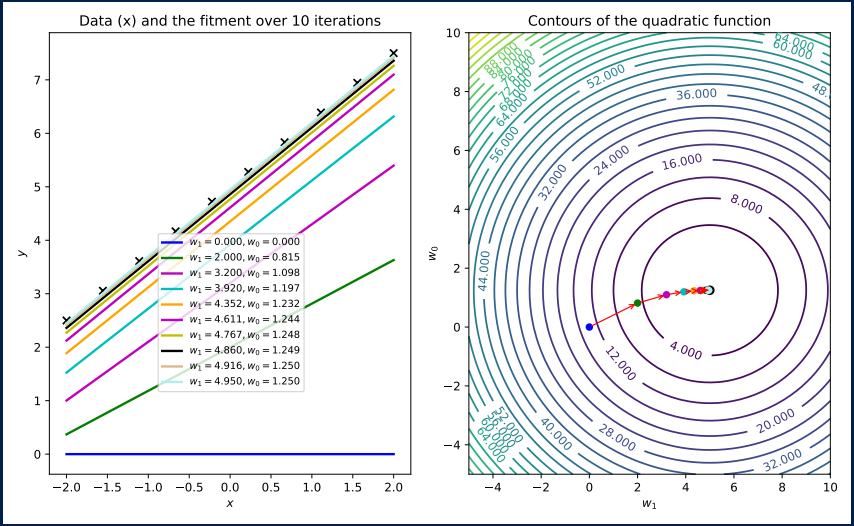


DECISION BOUNDARY - VARIATION OF BIAS

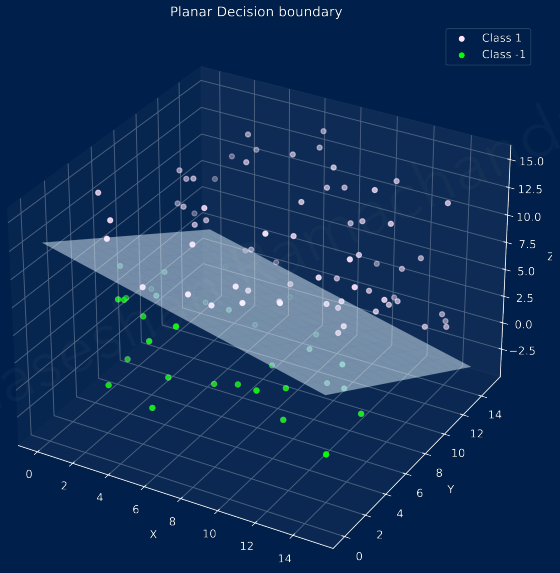
The contribution of bias to the the creation of the decision boundary



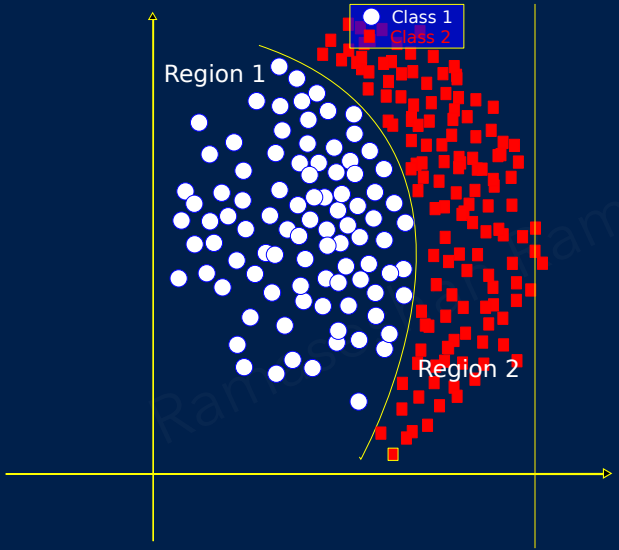
DECISION BOUNDARY AND GRADIENT DESCENT



DECISION SURFACE FOR A 3-D VECTOR



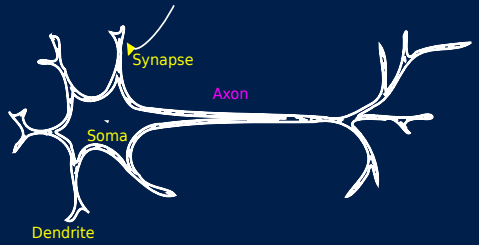
LINEARLY SEPARABLE?



Is this separable?

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

NEURAL NETWORK



- Each individual neuron can form thousands of links with other neurons.
- A typical brain has well over 100 trillion synapses
- Functionally related neurons connect to each other to form neural networks

- The electro-chemical connections between neurons are not static
- The more signals sent between two neurons, the stronger the connection grows and with each new experience and each remembered event, the brain slightly re-wires its physical structure.
- Our brains form a million new connections for every second of our lives

LAWS OF ASSOCIATION

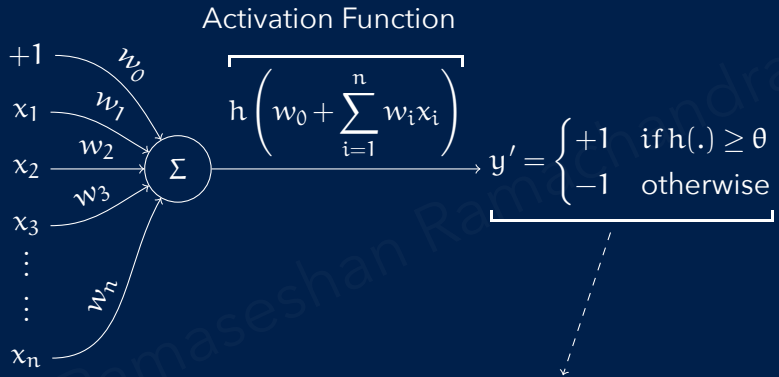
Aristotle's attempts on fundamental laws of learning and memory

The law of similarity	If two things are similar, the thought of one will tend to trigger the thought of the other - word2vec If you recollect one birthday, you may find yourself thinking about others as well
The law of contrast	Seeing or recalling something may also trigger the recollection of something completely opposite
The law of contiguity	Things or events that occur close to each other in space or time tend to get linked together in the mind If you found a snake in the corner of the street, every time you cross the corner, you tend to look for one. Events are conditioned based on the time and space
The law of frequency	The more often two things or events are linked, the more powerful will be that association - think of next word prediction - strength of the association decides

PERCEPTRON

Neuron	Perceptron
Biological	A mathematical model of a biological neuron
Dendrites receive electrical signals	Perceptron receives mathematical values as input
Electro-chemical signals between Dendrites and axons	The weighted sum represents the total strength of the signal
The electro-chemical signals are not static	Weights change during the training process

PERCEPTRON



PERCEPTRON LEARNING

- Perceptron learns the weights
- They are adjusted until the output is consistent with the target output in the training examples
- $w^{(k+1)} \propto (y - \hat{y})$
- The weights are updated as below

$$w_j^{(k+1)} = w_j^{(k)} - \eta(y_i - \hat{y}^{(k)})x_{ij}$$

where $w^{(k)}$ is the weight parameter associated with the i^{th} input at k^{th} iteration

η is the learning parameter and x_{ij} is the j^{th} attribute of the i^{th} training sample

- If $(y - \hat{y}) \approxeq 0$, no prediction error
- During the training the weights contributing most to the error require adjustments

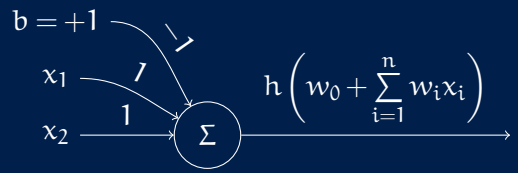
ALGORITHM FOR PERCEPTRON LEARNING

1: Total number of input vectors = k
2: Total number of features = n
3: Learning parameter $\eta = 0.01$, where $0 < \eta < 1$
4: epoch[3] count $t = 1, j = 1$
5: Initialize weights w_i with random numbers
6: Initialize the input layer with $\vec{x_j}$
7: Calculate the output using $\Sigma w_i x_i + w_0$
8: Calculate the error $(y - \hat{y})$.
9: Update the weights $w_j(t+1) = w_j - \eta(y - \hat{y})x_j$
10: Repeat steps 7 and 9 until: the error is less than θ or a predetermined number of epochs have been completed.

To provide a stable weight update for this step, $w_j(t+1) = w_j - \eta(y - \hat{y})x_j$, we require a small η. This results in slow learning. Bigger η would be good for fast learning. What are the problems? . What is the compromise?

[3] An epoch is one complete presentation of the data set to be learned to a learning machine.

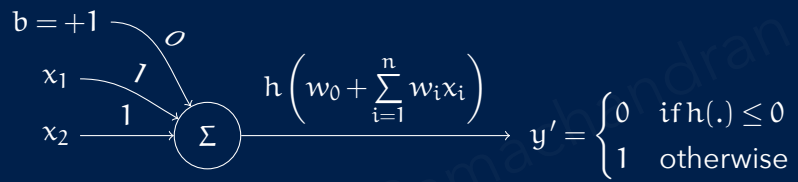
LOGICAL AND



Input x_1	Input x_2	$x_1.w_1 + x_2.w_2 + b.w_b$	output-y
0	0	0.1+0.1-1	0
0	1	0.1+1.1-1	0
1	0	1.1+0.1-1	0
1	1	1.1+1.1-1	1

Here, the perceptron is already trained and the learned weights are shown in the diagram

LOGICAL OR



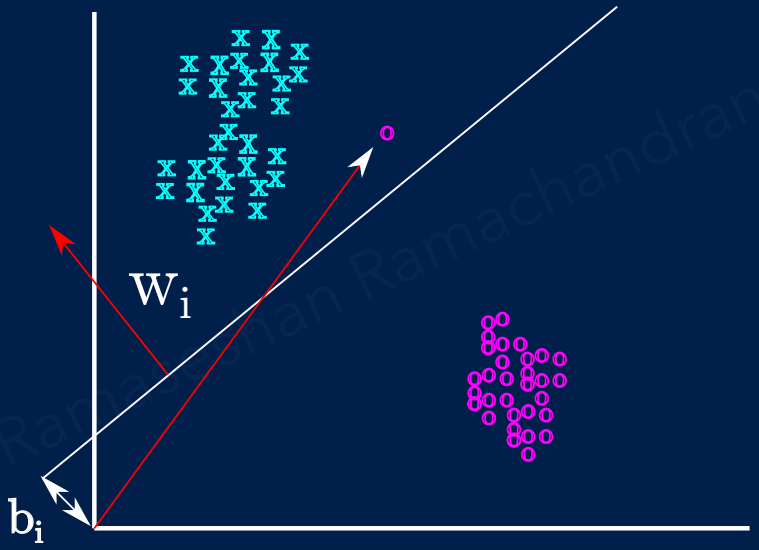
Input x_1	Input x_2	$x_1.w_1 + x_2.w_2 + b.w_b$	output-y

SENTIMENT ANALYSIS - USING PERCEPTRON

- Ability to classify reviews as positive or negative
- Positive and negative words for training
- Glove word embedding as features - input
  - 50 element word embedding[4]
  - Training Data generated using the intersection of the sentiment word list and word embedding from Glove

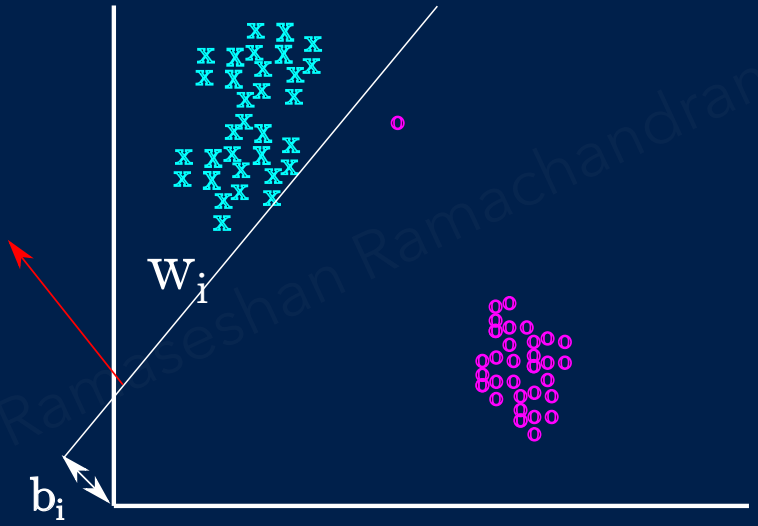
[4] data from https://nlp.stanford.edu/projects/glove/

PERCEPTRON LEARNING

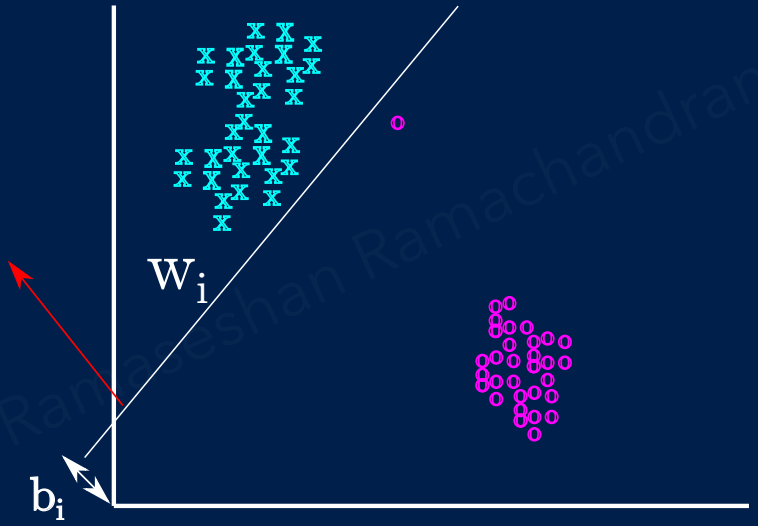


Figure

PERCEPTRON LEARNING



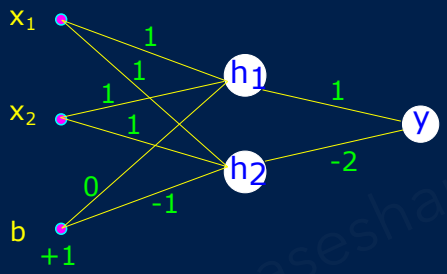
PERCEPTRON LEARNING



PERCEPTRON LIMITATIONS

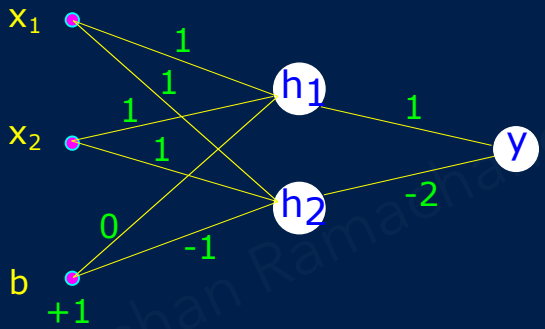
- It is based on the linear combination of fixed basis functions
- Updates the model only based on misclassification
- Documents that are linearly separable are classified

LOGICAL XOR



Input x_1	Input x_2	output-y
0	0	0
0	1	1
1	0	1
1	1	0

LOGICAL XOR



Input x_1	Input x_2	b	h_1	h_2	output-y
0	0	1	$f(0) = 0$	$f(-1) = 0$	0
0	1	1	$f(1) = 1$	$f(0) = 0$	1
1	0	1	$f(1) = 1$	$(f0) = 0$	1
1	1	1	$f(2) = 2$	$(f1) = 1$	0

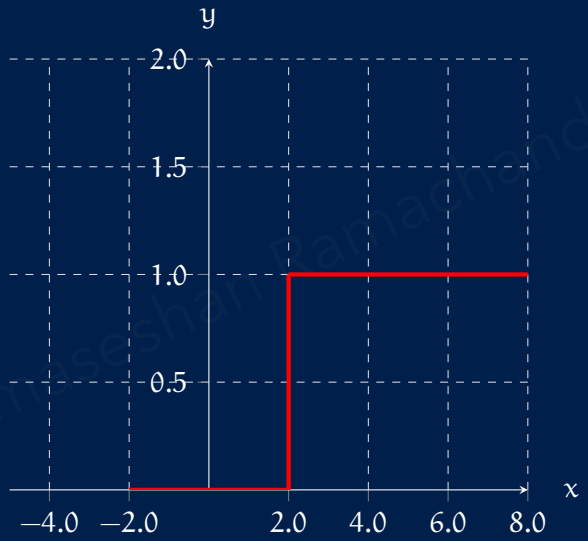
INTUITION

- Input space is transformed into hidden space
- Hidden layer represents the input layer
- Learns automatically the input representation and patterns
- $(0,1)$ and $(1,0)$ are merged into one in the h-space
- Patterns yielding similar results are merged into one
- Dimensionality reduction
- Learns to extract meaningful/latent features or representations from the input data
- Transforms the input information and create a representation in the new space
- Uses activation functions (e.g., sigmoid, tanh, or ReLU) to introduce non-linearity into the network, allowing it to capture complex relationships between input and output
- Are hidden layer neurons joining piecewise linear representations to create non-linear boundaries?

ACTIVATION FUNCTIONS

- Hard threshold
- Sigmoid
- Tanh
- ReLu - Rectified Linear Unit
- Leaky ReLu
- Softmax

HARD THRESHOLD



SIGMOID 1/3

Let us consider two classes c_1 and c_2. The posterior probability for c_1 using Bayes theorem is

$$p(c_1|w) = \frac{p(w|c_1)p(c_1)}{p(w|c_1)p(c_1) + p(w|c_2)p(c_2)}$$
$$= \frac{1}{1 + \frac{p(w|c_2)p(c_2)}{p(w|c_1)p(c_1)}}$$
$$\text{where } x = \log\left(\frac{p(w|c_1)p(c_1)}{p(w|c_2)p(c_2)}\right)$$

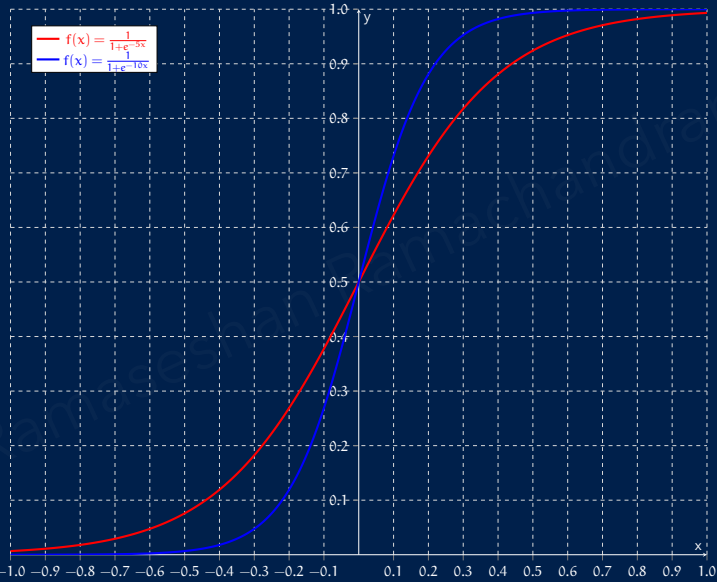
or

$$\sigma(x) = \frac{1}{1+\exp(-x)} \Rightarrow \text{Sigmoid}$$
$$\sigma(-x) = 1 - \sigma(x)$$

SIGMOID 2/3

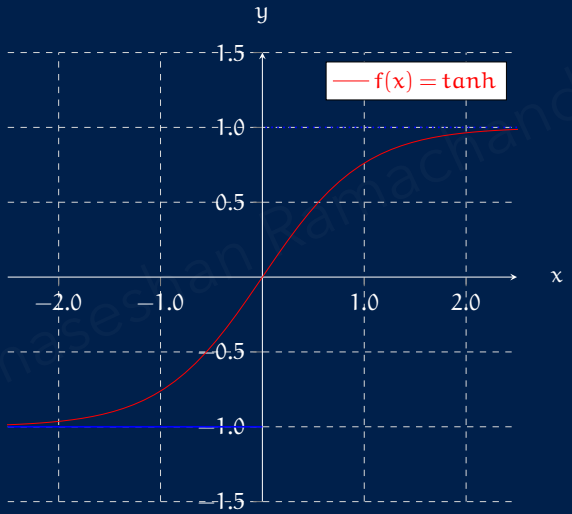
- The sigmoid is a non-linear function
- Better than hard threshold function as it squashes the net output into the range $[0, 1]$
- The values closer to the tails become 0 or 1
- In some cases, the values quickly saturate at 0 or 1
- At the bottom tail, most values become zero during the training and hence the most important aspect of learning of neural network is inhibited
- Sigmoid outputs are not zero-centered
- It is undesirable to have all the values squashed near the tails, where the gradient is ≈ 0

SIGMOID 3/3


$f(x) = \frac{1}{1+e^{-5x}}$
$f(x) = \frac{1}{1+e^{-10x}}$
y
x
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
−1.0 −0.9 −0.8 −0.7 −0.6 −0.5 −0.4 −0.3 −0.2 −0.1
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

TANH

This is a zero based non-linear function.



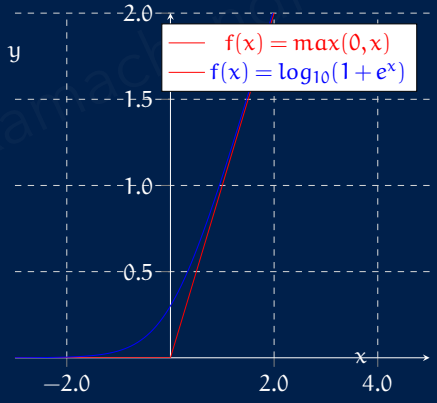
RELU - RECTIFIED LINEAR UNIT

- There is a continuous gradient for the neurons to be in active state
- Produces a non-zero gradient for values closer to zero
- Leaky ReLu

$$f(x) = \begin{cases} x, & \text{if} x > 0 \\ 0.01x, & otherwise \end{cases}$$

- Produces efficient propagation of the gradient
- Computationally efficient
- Scale invariant
- Unbounded and not zero centered

Learning rate (usually, very small) has to be fine tuned to minimize the death of neurons



MULTICLASS DECISION FUNCTION

- All linear classifier are used for binary classifications
- In NLP problems, we need to identify more than two classes
  - Document classification
  - Sentiment Analysis - positive, negative, neutral and non-sentiment word
- We need a decision function that predicts more than two classes by providing appropriate values
- An extension of the case function would be hard to manage

SOFTMAX

- Need a function that takes as input a vector of of size with N real numbers, and normalizes it into a K classes.
- Need a function that normalizes the net output and classes well separated (ideal condition)
- Need a function that fits the classes using probability and distributes the probability density

$$p(c_j|x) = \frac{p(x|c_j)p(c_j)}{\sum_k (p(x_k|c_k)p(c_k)} = \frac{\exp(x_k)}{\sum_k \exp(x_k)} \Rightarrow \mathbf{Softmax}$$

ACTIVATION FUNCTION - PYTHON CODE

```
import numpy as np
def sigmoid(X,W,b):
    return 1.0/(1.0+ np.exp(-(np.dot(W.T,X)+b)))
def tanh(X,W,b):
    z = np.exp(-(np.dot(W.T,X)+b))
    return (np.exp(z) - np.exp(-z))/(np.exp(z) + np.exp(-z))
def relu(X,W,b):
    return np.maximum(np.dot(W.T,X) + b,0)
def leaky_relu(X,W,b):
    return np.maximum(0.01*(np.dot(W.T,X)+b),np.dot(W.T,X)+b)
def softmax(X,W,b):
    z_exp = np.exp(np.dot(W,X)+b)
    return z_exp/np.sum(z_exp)
```

```python
if __name__ == '__main__':
    W = np.array([[1,2,3],[2,3,8],[1,5,7]])
    X=np.array([0.2, 0.1, 0.3])
    b=1.5

    print(sigmoid(X,W,b))    # [0.90024951 0.97587298 0.99330715]
    print(tanh(X,W,b))       # [0.11035192 0.02471849 0.00673785]
    print(relu(X,W,b))       # [2.2 3.7 5. ]
    print(leaky_relu(X,W,b))# [2.2 3.7 5. ]
    print(softmax(X,W,b))    # [0.08672022 0.52462674 0.38865305]
```

OBJECTIVE FUNCTION

- Provides information related to how well the model is in sync with the original data or gold standard
- Refers to a function used for optimization in a machine learning model, whether through minimization or maximization.
- In the context of machine learning, it usually refers to minimizing errors
- nI reinforcement learning, it could refer to maximizing the reward

LOSS FUNCTION I

Consider the prediction of y, given x

$$\hat{y} = w^T x + w_0 \tag{35}$$

where $\hat{y}$ is the predicted value
w_0 is the bias
$\mathbf{x}$ is the input vector $\mathbf{w}$ is the weight vector
If $y$is the target (0 or 1) and $\hat{y}$ is the predicted probability of y being 1 (a value between 0 and 1), then the loss function is:

LOSS FUNCTION II

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})] \tag{36}$$

Minimize cross-entropy loss, which measures the difference between predicted and true class distributions. When L is negligible, we confidently predict the correct class x.

$L \rightarrow 0$

- The model's prediction is very close to perfect.
- If the true sentiment is positive (y = 1), and L is close to zero, then $\hat{y}$ is very close to 1.
- If the true sentiment is negative (y = 0), and L is close to zero, then $\hat{y}$ is very close to 0.

L and Confidence

- A low L value also indicates high confidence in its prediction.
- A high L value indicates its low confidence in predicting the label

COST FUNCTION

Let's assume we have vectors for training. In sentiment analysis, these are vectors from word embedding methods, representing sentiment words. We could also use one-hot vectors.

$$\mathbf{X} = [x_1, x_2, x_3, \ldots, x_N] \tag{37}$$

Combining the model parameters $w_0, w_1, w_2, w_3, \ldots, w_n$ with the loss function, we get a **Cost function**, averaged over all the input training samples

$$J(\theta) = \frac{1}{2} \sum_i L(y_i, \hat{y}_i)^2 \tag{38}$$

- The loss is a function of prediction and target values of a single training example
- The cost is a function of model parameters and bias - average of the loss over all training samples

COST FUNCTION I

Let's assume we have a set of training vectors, denoted as:

$$X = [x_1, x_2, x_3, ..., x_N] \tag{39}$$

- where each x_i represents a data point. In sentiment analysis, these vectors could be obtained from word embedding methods (e.g., Word2Vec, GloVe) to represent sentiment words, capturing semantic relationships. Alternatively, one-hot vectors could be used, although they are less informative.
- The Cost Function is a function of the model parameters, denoted as θ, and is obtained by averaging the loss function over all training samples. It provides an overall measure of how well the model is performing on the entire training set.

COST FUNCTION II

A common form of the cost function, using the squared error loss, is:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}i) = \frac{1}{2N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{40}$$

- where N is the total number of training samples.
- y_i is the true sentiment label for the i^{th} sample.

  $\hat{y}_i$ is the predicted sentiment label for the i^{th} sample.
- The choice of loss function depends on the specific nature of the problem and the desired behavior of the model.
- Additionally, the cost function may include regularization terms to prevent over fitting and encourage simpler models.

GRADIENT DESCENT

- GD iteratively used to adjust the weights and as a result to minimize the cost function
- Initialize the weights to random values
- Iteratively adjust the weights in the direction of the steepest descent or in the direction that most decreases the cost function. To update the weights in the steepest descent, a learning parameter η is used

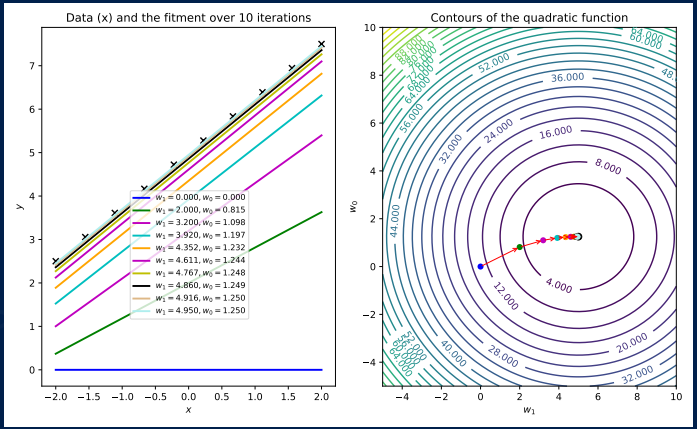
$$w_j \leftarrow w_j - \eta \frac{\partial J(\theta)}{\partial w_j} \tag{41}$$

$$= w_j - \eta \sum_{i=1}^{N} x_j{}^i (y - \hat{y}) \tag{42}$$

where η is the learning parameter and usually takes the value between 0.01 and 0.001

GD ADVANTAGES

- Iterative
- Computationally efficient
- Generic and could be used to solve even non-linear equations
- Suitable for large models
- It works!
- It is very slow when it reaches close to the local minima



SEQUENTIAL NATURE OF DATA

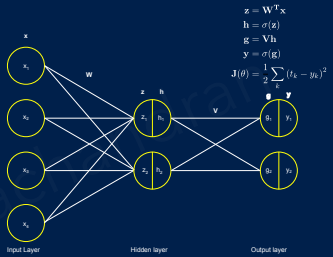
- Speech
- Documents
- Videos
- Weather forecast
- Financial - Stock market

PROPERTIES OF ANN

- Massively parallel distributed structure
- Ability to learn
- Ability to learn from training samples
- Ability to find latent patterns in the data
- Generalize and associate data

BACKPROPAGATION MODEL

The goal of backpropagation is to change the weights so that the *estimated target* $\approx$ target, thereby minimizing the error for each neuron and the network as a whole.

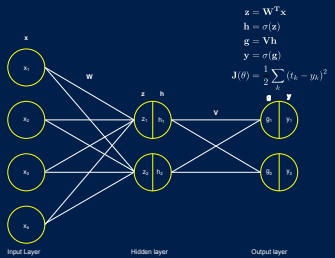


The goal is to minimize

$$J(\theta) = 1/2\Big((t_1 - y_1)^2 + (t_2 - y_2)^2\Big) \tag{43}$$

* We want to the adjust weights coming in and going out of hidden layer so that $\mathbf{t} - \mathbf{y}$ is minimized
* $\Delta\mathbf{W} \propto -\dfrac{\partial J(\theta)}{\partial \mathbf{W}}$

BACK PROPAGATION MODEL - FORWARD PASS



$$z_1 = \mathbf{w}^T\mathbf{x} + b_1 \quad (44)$$

$$z_2 = \mathbf{w}^T\mathbf{x} + b_1 \quad (45)$$

$$z_1 = x_1.w_{11} + x_2.w_{21} + \cdots + b_1 \quad (46)$$

$$z_2 = x_2.w_{12} + x_2.w_{22} + \cdots + b_1 \quad (47)$$

=

$$h_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}} \quad (48)$$

$$h_2 = \sigma(z_2) = \frac{1}{1+e^{-z_2}} \quad (49)$$

$$g_1 = h_1 * v_{11} + h_2.v_{21} \quad (50)$$

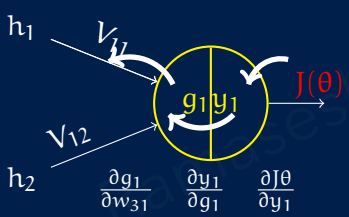
$$g_2 = h_2 * v_{12} + h_2.v_{22} \quad (51)$$

$$y_1 = \sigma(g_1) = \frac{1}{1+e^{-g_1}} \quad (52)$$

$$y_2 = \sigma(g_2) = \frac{1}{1+e^{-g_2}} \quad (53)$$

BACKWARD PASS–ADJUST HIDDEN-OUTPUT LAYER WEIGHTS

$$J(\theta) = 1/2\Big((t_1 - y_1)^2 + (t_2 - y_2)^2\Big)$$



$$\frac{\partial J(\theta)}{\partial V_{11}} = -\alpha\left(\frac{\partial J(\theta)}{\partial y_1}\frac{\partial y_1}{\partial g_1}\frac{\partial g_1}{\partial v_{11}}\right) \tag{54}$$

$$\frac{\partial J(\theta)}{\partial y_1} = -(t_1 - y_1) \tag{55}$$

$$\frac{\partial y_1}{\partial g_1} = y_1(1 - y_1) \tag{56}$$

$$\frac{\partial g_1}{\partial V_{11}} = h_1 \tag{57}$$

BACKWARD PASS-ADJUST HIDDEN-OUTPUT LAYER WEIGHTS

$$\frac{\partial J(\theta)}{\partial v_{11}} = -\alpha \left(\frac{\partial J(\theta)}{\partial y_1} \frac{\partial y_1}{\partial g_1} \frac{\partial g_1}{\partial v_{11}} \right) \tag{58}$$

$$= \alpha(t_1 - y_1) y_1 (1 - y_1) h_1 \tag{59}$$

Now, the weights can be updated by $v_{11}^{t+1} = v_{11}^{t} - \eta * \frac{\partial J(\theta)}{\partial v_{11}^{t}}$. In the same fashion, compute the error to be propagated back to the other weights.

BACKWARD PASS – ADJUST INPUT-HIDDEN LAYER WEIGHTS

$$\frac{\partial J(\theta_1)}{\partial w_{11}} = -\alpha \frac{\partial J(\theta_1)}{\partial y_1} \frac{\partial y_1}{\partial g_1} \frac{\partial g_1}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_{11}} \tag{60}$$

$$\frac{\partial J(\theta_2)}{\partial w_{11}} = -\alpha \frac{\partial J(\theta_2)}{\partial y_2} \frac{\partial y_2}{\partial g_2} \frac{\partial y_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_{11}} \tag{61}$$

$$\frac{\partial J(\theta)}{\partial w_{11}} = \frac{\partial J(\theta_1)}{\partial w_{11}} + \frac{\partial J(\theta_2)}{\partial w_{11}} \tag{62}$$

$$\frac{\partial J(\theta_1)}{\partial y_1} = -(t_1 - y_1) \tag{63}$$

$$\frac{\partial y_1}{\partial g_1} = y_1(1 - y_1) \tag{64}$$

$$\frac{\partial g_1}{\partial h_1} = v_{11} \quad \frac{\partial z_1}{\partial w_{11}} = x_1 \tag{65}$$

$$\frac{\partial h_1}{\partial z_1} = \sigma(z_1)(1 - \sigma(z_1)) \tag{66}$$

$$\frac{\partial J(\theta_2)}{\partial y_2} = \cdots \tag{67}$$

Now, input-hidden layer weights can be updated using $\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta * \frac{\partial \mathbf{J}(\theta)}{\partial \mathbf{w}}$

FEED FORWARD NETWORK

Once trained,

- The hidden layer of a trained model is a lookup table
- Hidden weights act as an associative memory and capture the relational similarities

INTRODUCTION TO LOGISTIC REGRESSION

- **Logistic regression** is a classification algorithm used to predict the probability of a binary outcome.
- We want to model the conditional probability $p(y = 1|x)$ as a function of x for the binary output variable y and making $p(.)$ be a linear function of x won't work
- Consider using $\log p(x)$ as a linear function - unbounded in one direction Consider $\frac{p}{1-p}$ is even better. This is known as logistic transformation of ***logit***
- The prediction is based on the logistic function (sigmoid):

$$P(y = 1|x) = \frac{1}{1 + e^{-w^T x}}$$

where x is the input vector, w is the parameter vector.

- Logistic regression outputs a probability and classifies based on a threshold (usually 0.5).

WORD EMBEDDINGS

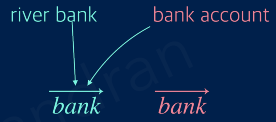
- **Word2Vec** is a neural network model that learns word embeddings.
- Words are mapped to high-dimensional vectors where semantically similar words have similar vector representations.
- The word embeddings are pre-trained and can be used as features in downstream tasks.
- Embeddings transform sparse word representations (one-hot vectors) into dense, lower-dimensional continuous vectors.

$$\text{Embedding: } \mathbf{w} \in \mathbb{R}^d$$

- Handles contextual similarity

VECTOR REPRESENTATION OF WORDS

- Word embeddings are the most fundamental to NLP
- Capturing meaning: Represents meaning of a word numerically
- Dimensionality reduction: Projects words onto a much lower-dimensional space
- Semantic relationships: Captures semantic relationships like synonymy, antonymy and semantically similar words
- Context free:Captures the meaning based on co-occurrence statistics



IDENTIFIES SIMILAR WORDS

Thank you for your help

Thank you for your assistance
Thank you for your aid
Thank you for your support
Thank you for your guidance
Thank you for backing

Identifies Microsoft and IBM as similar words - they are similar in semantic sense

WORD VECTOR EXAMPLES

```
Similar words for apple
('apple',    0)
('iphone',    0.266)
('ipad',    0.287)
('apples',    0.356)
('blackberry',    0.361)
('ipod',    0.365)
('macbook',    0.383)
('mac',    0.391)
('android',    0.391)
('google',    0.395)
('microsoft',    0.418)
('ios',    0.433)
('iphones',    0.445)
('touch',    0.446)
('sony',    0.447)
```

Similar words for - american

```
'american', 0
'america', 0.255
'americans', 0.312
'u.s.', 0.320
'british', 0.323
'canadian', 0.329
'history', 0.356
'national', 0.364
'african', 0.374
'society', 0.375
'states', 0.386
'european', 0.387
'world', 0.394
'nation', 0.399
'us', 0.399
```

VECTOR DIFFERENCE BETWEEN TWO WORDS

$$\overrightarrow{\text{apple}} - \overrightarrow{\text{iphone}}$$

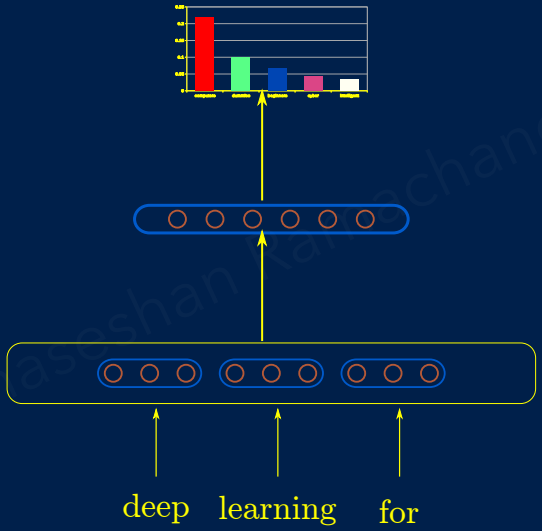
```
('raisin', 0.5744591153088133)
('pecan', 0.5760617374141159)
('cranberry', 0.5840016172254104)
('butternut', 0.5882322018694753)
('cider', 0.5910795032086132)
('apricot', 0.6036644437522422)
('tomato', 0.6073715970323961)
('rosemary', 0.6150986936477657)
('rhubarb', 0.6157884153793192)
('feta', 0.6183016129045151)
('apples', 0.6226003361980218)
('avocado', 0.6235366677962004)
('fennel', 0.6306016018912576)
('chutney', 0.6312524337590703)
('spiced', 0.6327632200841328)
```

VECTOR ARITHMETIC ON WORD VECTORS...

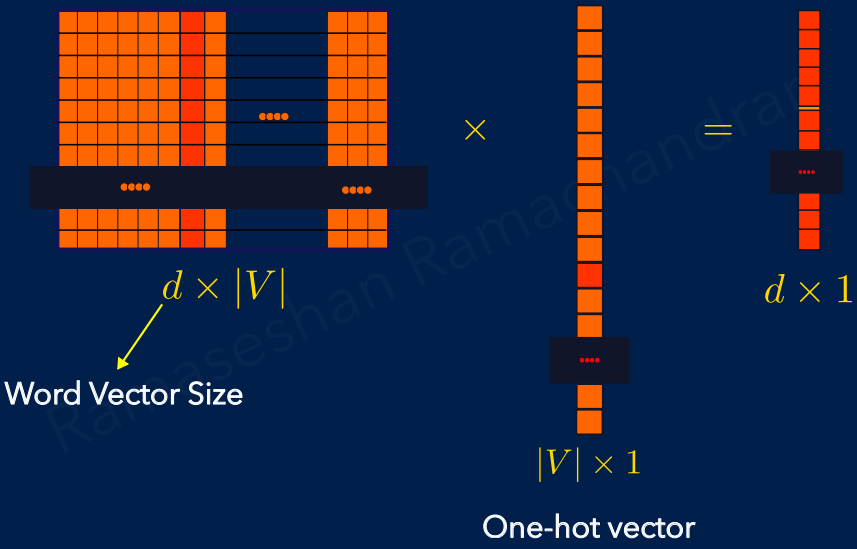
840B words and 300 elements word vectors used for this computation

$\overrightarrow{\text{apple}}$	$\overrightarrow{\text{apple}} - \overrightarrow{\text{iphone}}$	$\overrightarrow{\text{apple}} - \overrightarrow{\text{fruit}}$
('apple', 0)	('apples', 0.39)	('ipad', 0.412)
('apples', 0.25)	('fruit', 0.43)	('iphone', 0.433)
('blackberry', 0.31)	('grape', 0.44)	('macbook', 0.435)
('Apple', 0.35)	('tomato', 0.44)	('ipod', 0.445))
('iphone', 0.37)	('pecan', 0.45)	('imac', 0.465
('fruit', 0.37)	('rhubarb', 0.45)	('3gs', 0.473)
('blueberry', 0.38)	('pears', 0.45)	('Ipad', 0.490)
('strawberry', 0.38)	('cranberry', 0.452)	('itouch', 0.512)
('ipad', 0.39)	('raisin', 0.453)	('ipad2', 0.514)
('pineapple', 0.39)	('apricot', 0.459)	('Iphone', 0.514)
('pear', 0.39)	('carrot', 0.461)	('ios', 0.520)
('cider', 0.39)	('candied', 0.462)	('Macbook', 0.524)
('mango', 0.40)	('blueberry', 0.463)	('ibook', 0.534)
('ipod', 0.40)	('apricots', 0.466)	('IPhone', 0.541)
('raspberry', 0.40)	('tomatoes', 0.466)	('32gb', 0.545)

ANN FOR LM



EMBEDDING LAYER CREATION



NEURAL LM MODEL

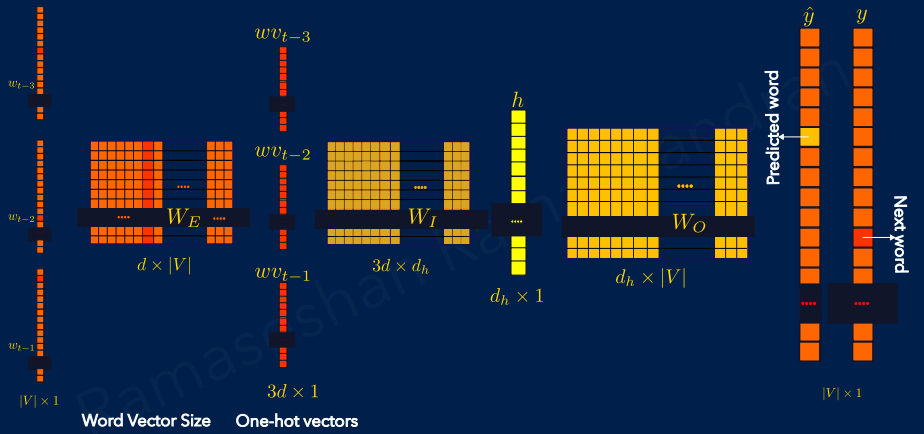


Figure: Adopted from https://web.stanford.edu/~jurafsky/slp3/7.pdf - Chapter 7 -[1]

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.

$$\hat{y} = p(y|\text{context words}) \tag{68}$$

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[5]

$$p(x) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases} \tag{69}$$

Now, $p(y|\text{context words})$ in equation(68) can be written as

$$p(y|\text{context words}) = \hat{y}^{y} - (1-\hat{y})^{1-y} \tag{70}$$

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

LOSS FUNCTION

Taking log on both sides of equation(70)

$$\begin{aligned}\log(p(y|\text{context words})) &= \log(\hat{y}^y - (1-\hat{y})^{1-y}) \\ &= y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \end{aligned} \tag{71}$$

By incorporating equation(68), the loss function will be a cross entropy loss function which we need to minimize during the learning process

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \tag{72}$$

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

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(f(\mathbf{w_O}.\mathbf{h} + b)) - (1-y)\log(1-f(\mathbf{w_O}.\mathbf{h} + b)) \tag{73}$$

As we are dealing with one-hot vector as the true output, the equation(72) becomes

$$\begin{aligned}\mathscr{E}(W_E, W_I, h, W_O) &= -y \log(\hat{y}) \\ &= -y \log(f(\mathbf{w_O}.\mathbf{h} + b)) \text{ or} \\ \mathbf{E}(W_E, W_I, h, W_O) &= -\sum_{d=1}^{|V|} \mathbf{y}_d \log(\hat{\mathbf{y}}_d) \end{aligned} \tag{74}$$

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 (72) 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 $\mathrm{go} : \mathrm{went} :: \mathrm{run} : \mathrm{Ran}$ or $\mathrm{queen} \approx \mathrm{king} - \mathrm{man} + \mathrm{woman}$

REFERENCES I

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

Break