

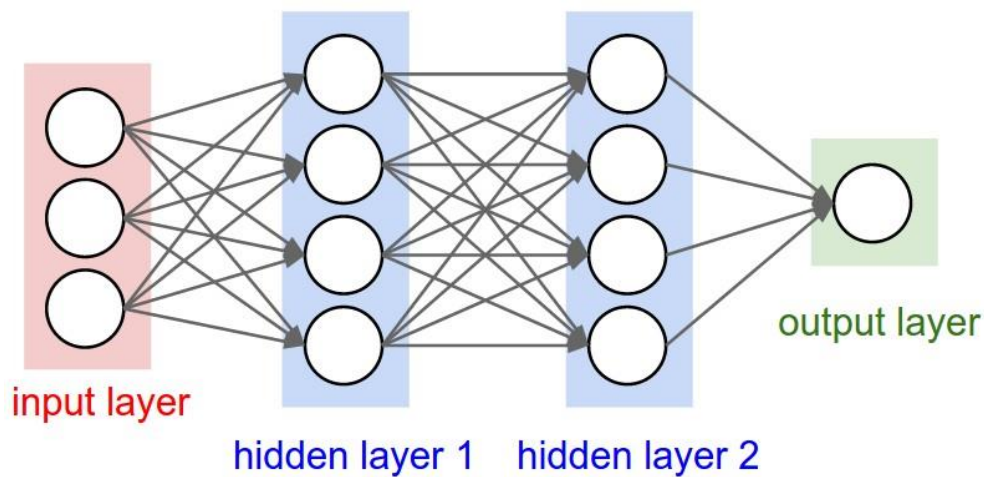
SPEECH EMOTION RECOGNITION

FINAL REPORT SOC 2024

NIRANJAN VERMA
210020085

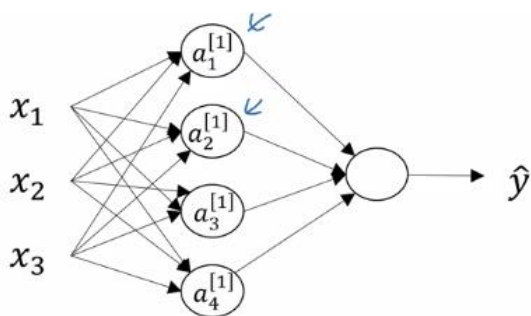
NEURAL NETWORKS

Neural networks are computational models inspired by the human brain's structure and function. They consist of interconnected nodes or "neurons" that process information in layers. These networks are widely used in machine learning for tasks like classification, regression, and pattern recognition, leveraging their ability to learn from data through a process called training.



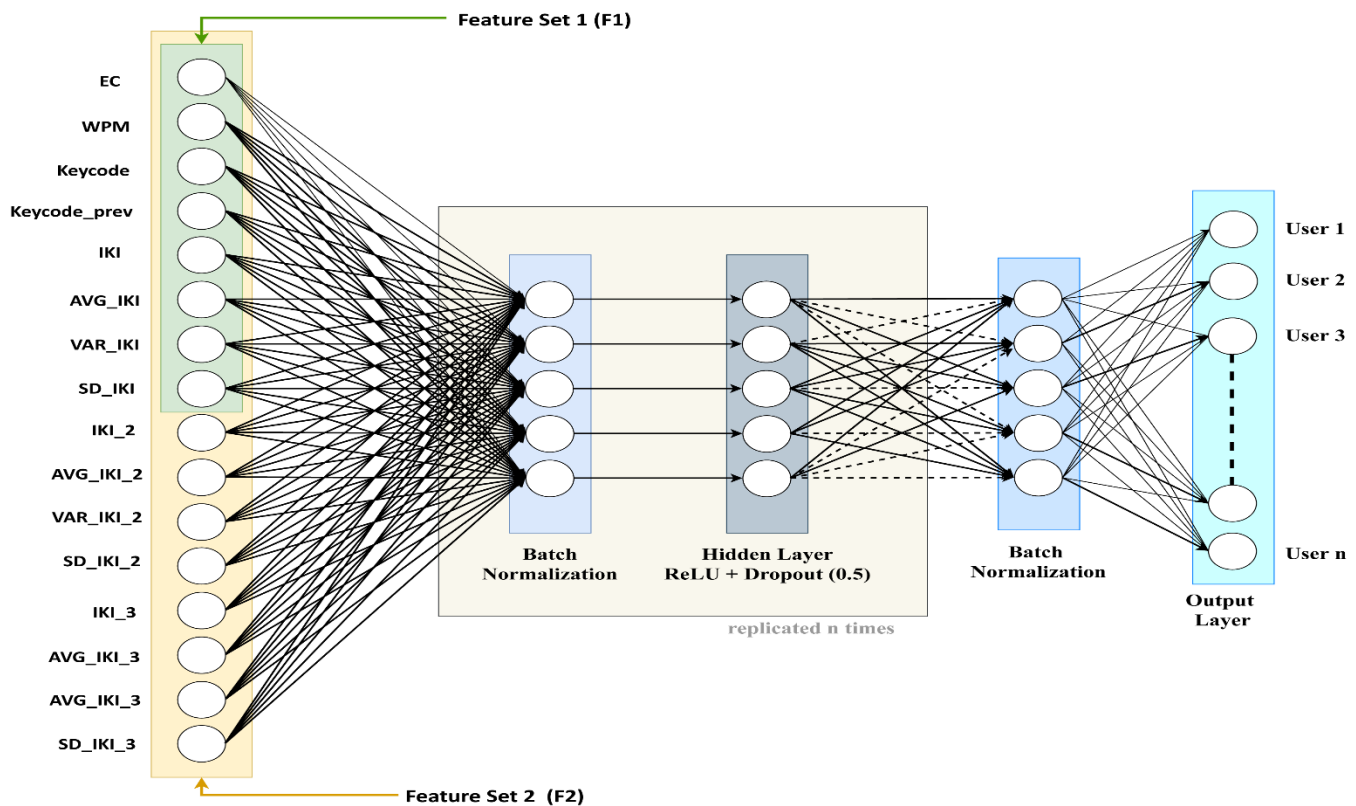
SHALLOW NEURAL NETWORKS

Shallow neural networks typically consist of one or two hidden layers between the input and output layers.

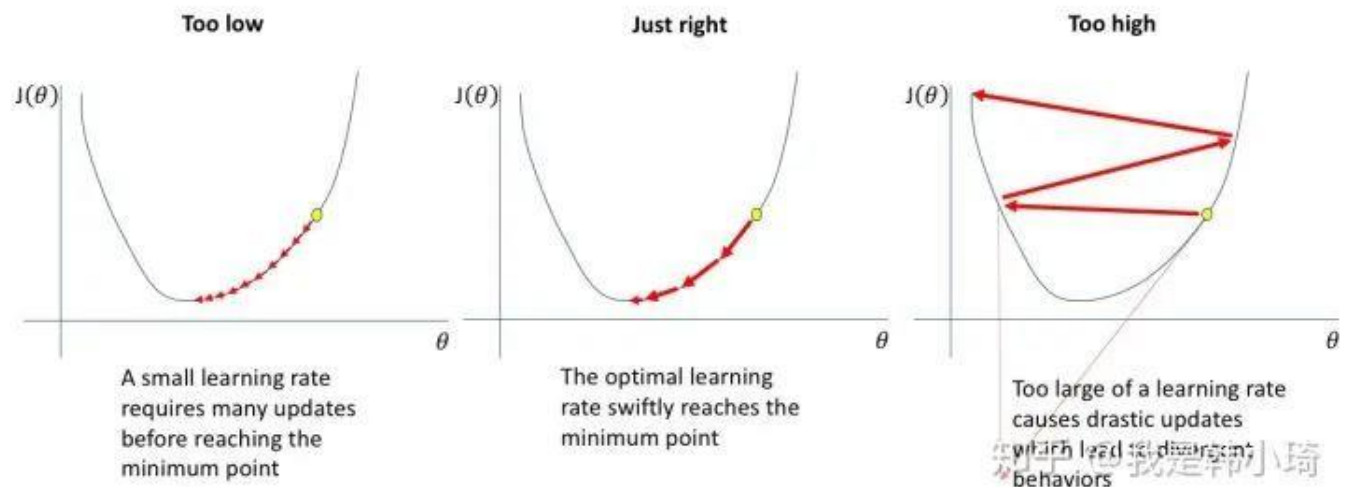


DEEP NEURAL NETWORKS

Deep neural networks (DNNs) feature multiple hidden layers, enabling them to model highly complex and abstract patterns in data. This depth allows DNNs to excel in a wide range of applications, including image and speech recognition, natural language processing, and autonomous systems. Despite their powerful capabilities, DNNs require significant computational resources and large datasets for effective training.



OPTIMIZATION



There are many types of optimizers:

1. Gradient Descent
 - a. Stochastic Gradient Descent (SGD)
 - b. Mini-batch Gradient Descent
 - c. Batch Gradient Descent
2. Adam (Adaptive Moment Estimation)
3. AdaGrad (Adaptive Gradient Algorithm)
4. RMSProp (Root Mean Square Propagation)
5. SGD with Momentum

NORMALIZATION

Normalization is a data preprocessing technique used to scale input features so they have a consistent range or distribution.

1. Batch Normalization
2. Layer Normalization
3. Group Normalization

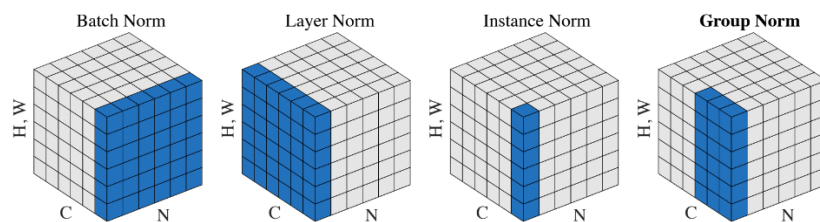
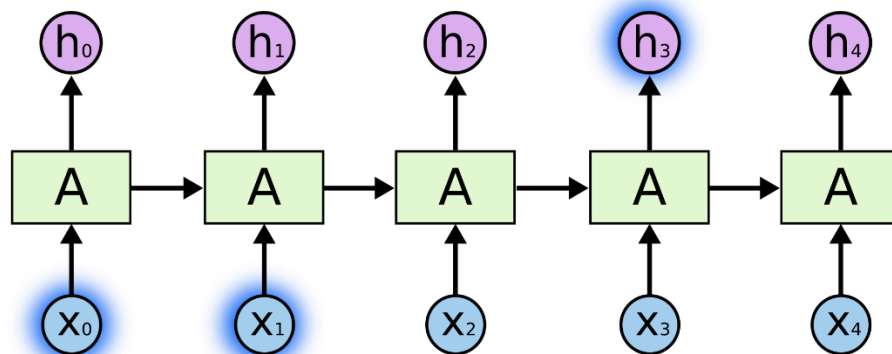


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

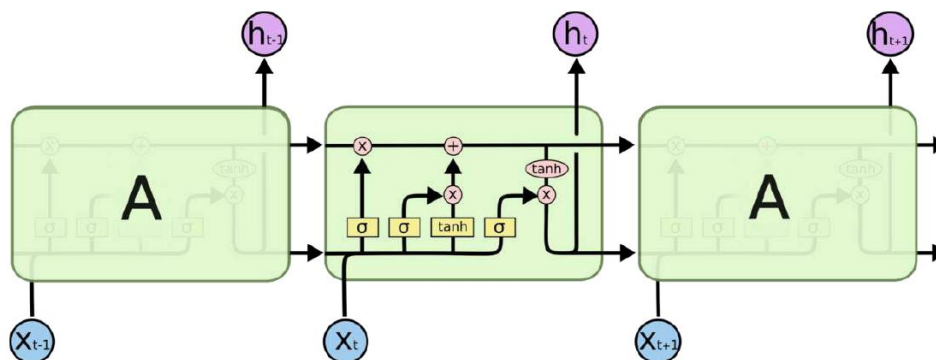
RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs) are a class of neural networks designed for sequence data. Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a memory of previous inputs. This makes them particularly effective for tasks involving sequential data, such as time series prediction, natural language processing, and speech recognition.



LSTM (LONG SHORT-TERM MEMORY)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) specifically designed to capture long-term dependencies in sequence data. They address the vanishing gradient problem faced by standard RNNs through a sophisticated architecture that includes cell states and three types of gates: input, forget, and output gates.

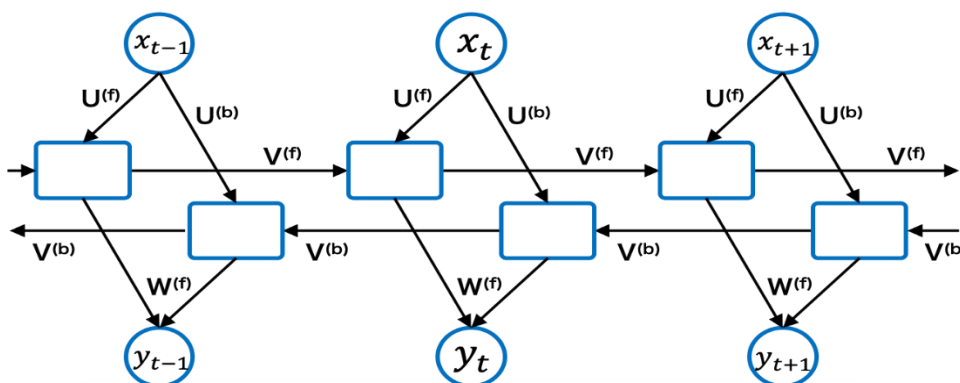


GRU (GATED RECURRENT UNITS)

Gated Recurrent Units (GRUs) are a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and improve the learning of long-term dependencies. GRUs simplify the LSTM architecture by combining the forget and input gates into a single update gate and merging the cell state and hidden state.

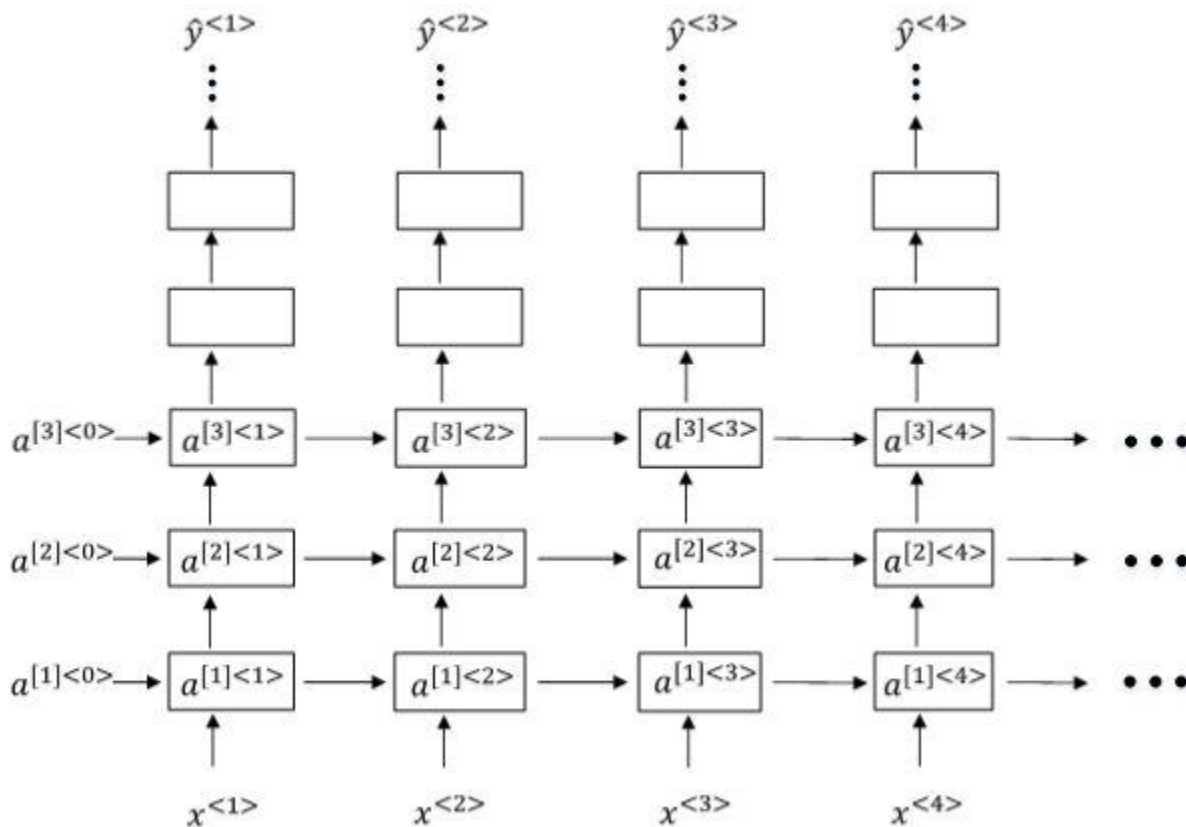
BI DIRECTIONAL RNN'S

Bi-Directional Recurrent Neural Networks (Bi-RNNs) are an extension of traditional RNNs designed to capture context from both past and future states. This is particularly useful for tasks where understanding the full context of a sequence is crucial.



DEEP RNN'S

Deep Recurrent Neural Networks (Deep RNNs) extend the concept of traditional RNNs by stacking multiple RNN layers on top of each other, creating a deep architecture. This depth allows Deep RNNs to capture more complex patterns and hierarchical features in sequential data.



WORD EMBEDDINGS

Word embeddings are dense vector representations of words in a continuous vector space, where words with similar meanings have similar representations. These representations capture semantic relationships and syntactic structures among words, making them crucial for natural language processing tasks.

Numerical Indexing for each word

Each word in the vocabulary can be given a numerical index to represent them.

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

But it doesn't make any semantic sense.

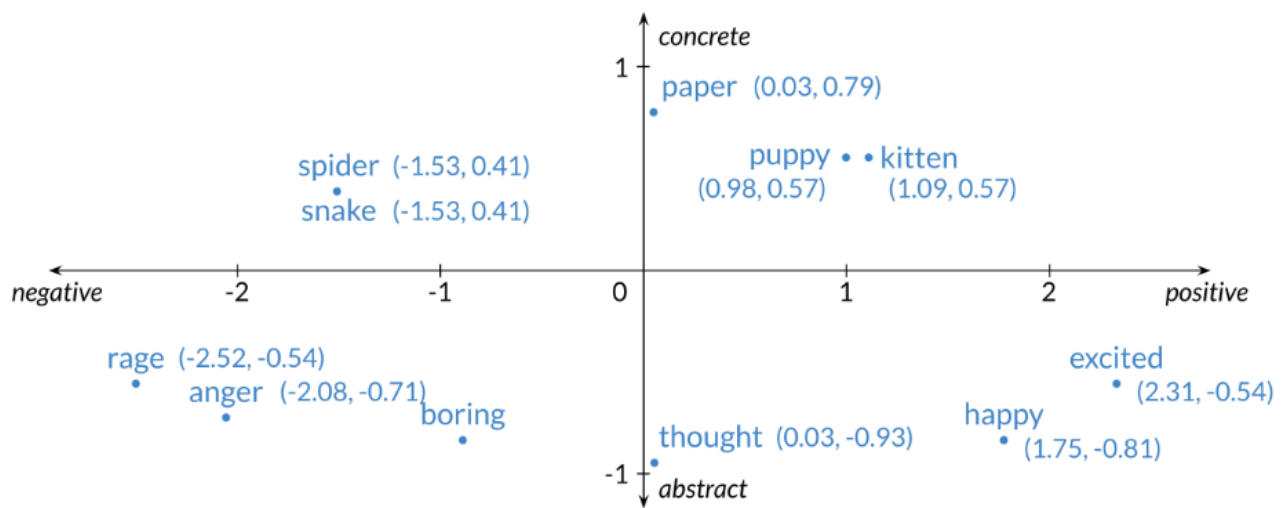
One hot vector

Word	Number		
a	1	1	0
able	2	2	0
about	3	3	0
...
hand	615	615	0
...
happy	621	621	1
...
zebra	1000	1000	0

Huge Vectors

No embedded meaning

Word Embedding



From the plot above, you can see that when encoding a word in 2D, similar words tend to be found next to each other. Perhaps the first coordinate represents whether a word is positive or negative. The second coordinate tell you whether the word is abstract or concrete. This is just an example; in the real world you will find embeddings with hundreds of dimensions.

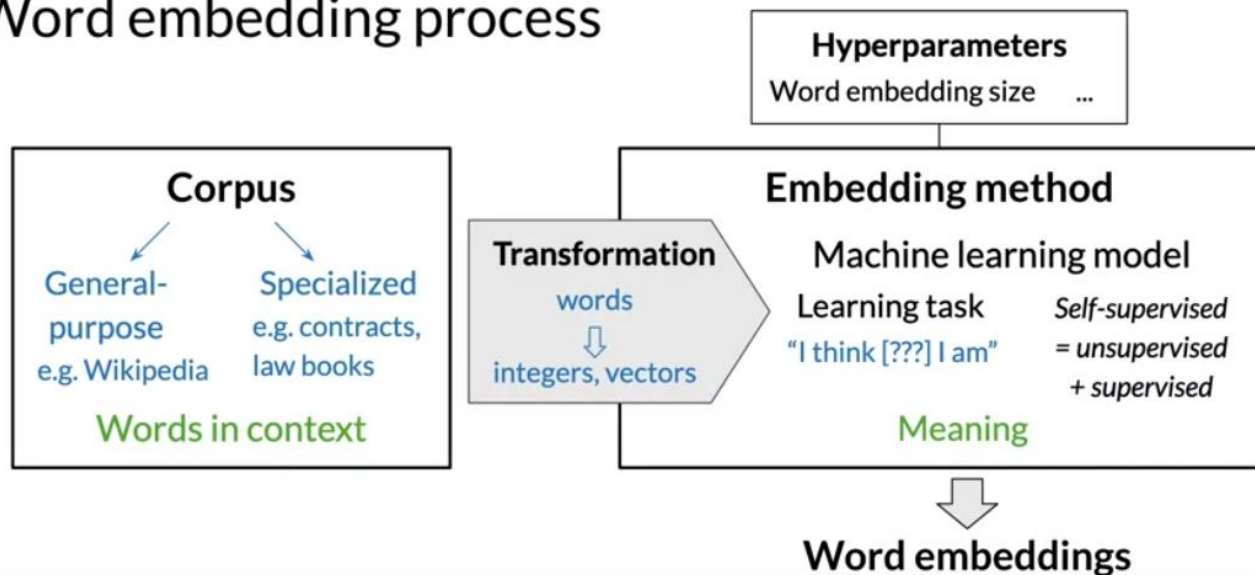
Pros:

Low dimension vectors

Allows you to encode meaning

How to create Word Embedding?

Word embedding process



The task is self-supervised: it is both unsupervised in the sense that the input data — the corpus — is unlabeled, and supervised in the sense that the data itself provides the necessary context which would ordinarily make up the labels.

Word Embedding Methods

Basic word embedding methods

- word2vec (Google, 2013)
 - Continuous bag-of-words (CBOW)
 - Continuous skip-gram / Skip-gram with negative sampling (SGNS)
- Global Vectors (GloVe) (Stanford, 2014)
- fastText (Facebook, 2016)
 - Supports out-of-vocabulary (OOV) words

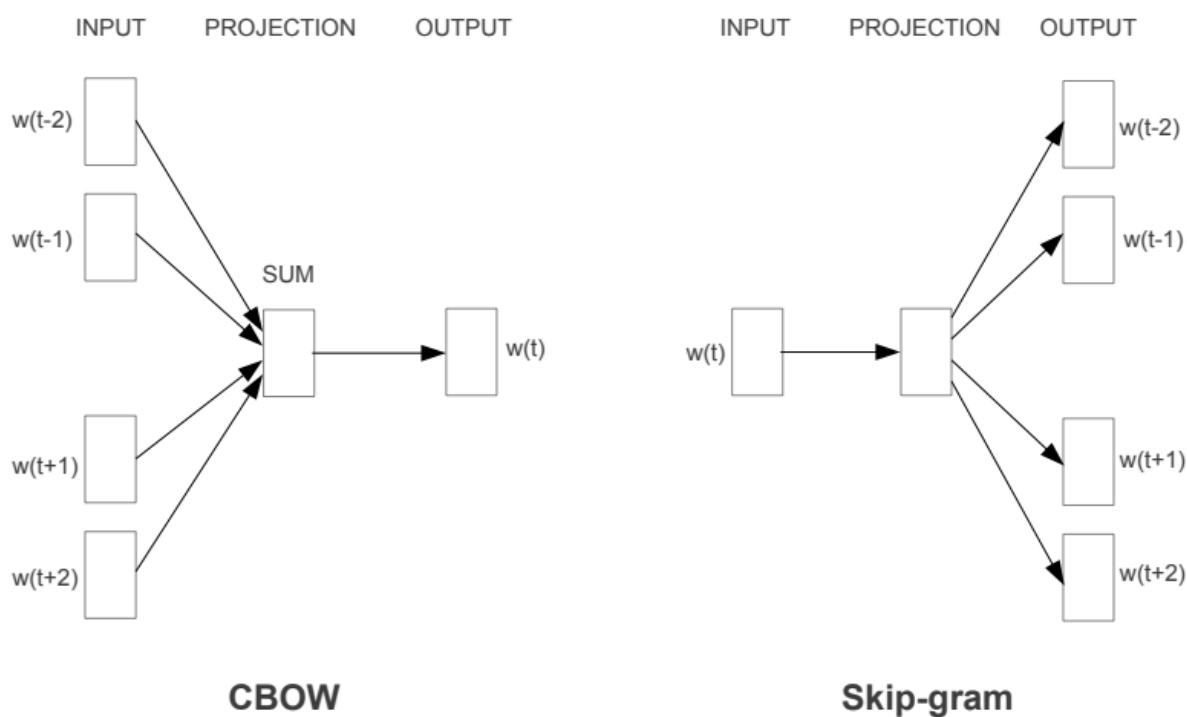
Advanced word embedding methods

Deep learning, contextual embeddings

- BERT (Google, 2018)
 - ELMo (Allen Institute for AI, 2018)
 - GPT-2 (OpenAI, 2018)
- } Tunable pre-trained models available

WORD2VEC

Word2Vec is a popular technique for learning word embeddings from large text corpora. It uses shallow neural networks to predict a target word based on its context (Skip-gram model) or predict context words given a target word (Continuous Bag of Words, CBOW model).



GLOVE

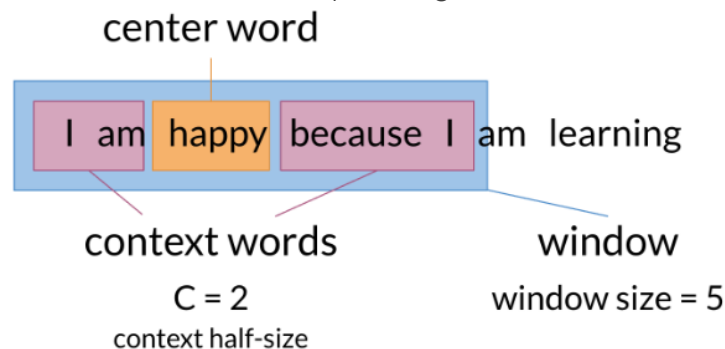
GloVe is another widely used method for learning word embeddings, which combines the advantages of global matrix factorization techniques and local context window-based methods.

CONTINUOUS BAG OF WORDS

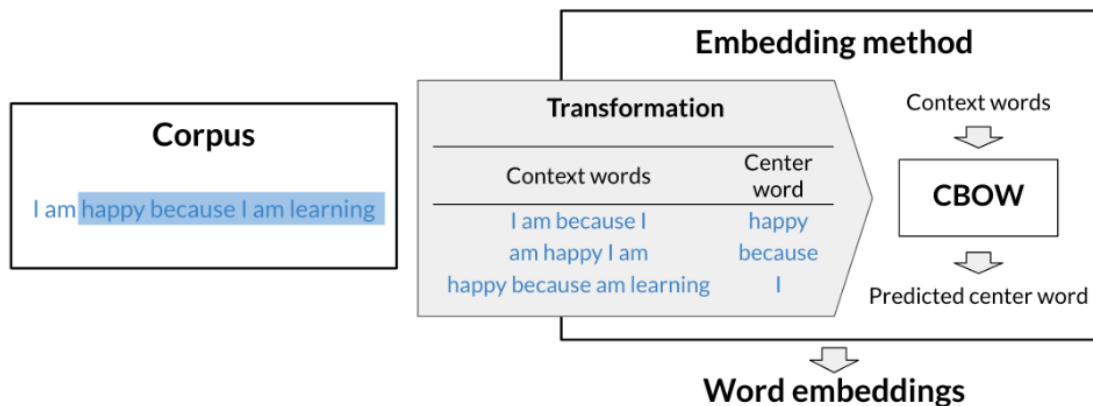
The Continuous Bag of Words (CBOW) model is a neural network-based approach for learning word embeddings in natural language processing (NLP). It is one of the two main models introduced by Mikolov et al. in the Word2Vec framework, the other being the Skip-gram model. CBOW predicts the target word from its context words, effectively learning to understand word meanings and their relationships.

Key concepts:

- **Word Embeddings:** Dense vector representations of words in a continuous vector space, capturing semantic similarities.
- **Context Words:** The surrounding words in a fixed-size window around a target word.
- **Target Word:** The word that the model tries to predict given its context words.



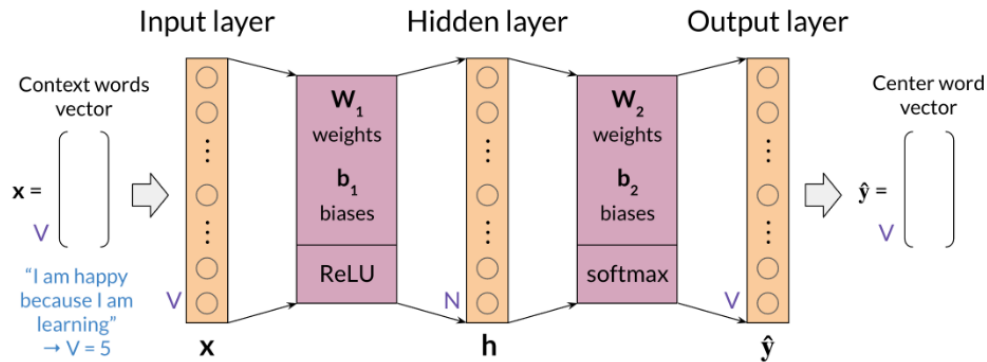
Model:



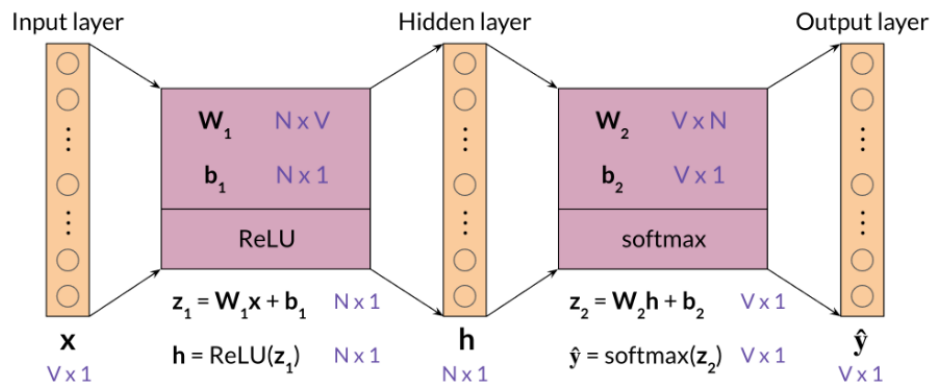
Cleaning and Tokenization

Letter case	"The" == "the" == "THE" → lowercase / upper case
Punctuation	, ! . ? → . " ' « » ' ' → ∅ ... !! ??? → .
Numbers	1 2 3 5 8 → ∅ 3.14159 90210 → as is/<NUMBER>
Special characters	▽ \$ € § ¶ ** → ∅
Special words	😊 #nlp → :happy: #nlp

Architecture:



Dimensions:



Extracting Word Embedding Vector

Option 1: Use W_1 matrix

$$W_1 = \begin{bmatrix} w^{(1)} & \dots & w^{(V)} \end{bmatrix} \quad \text{size } N \times V$$

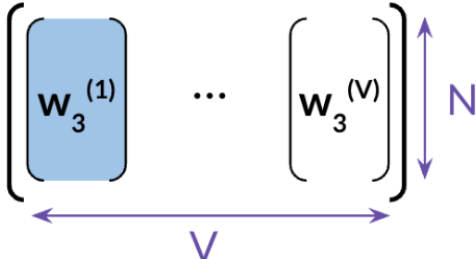
$$x = \begin{bmatrix} \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{bmatrix} \quad \text{size } V \times 1$$

Option 2: Use W_2 matrix

$$W_2 = \begin{bmatrix} w^{(1)} \\ \dots \\ w^{(V)} \end{bmatrix} \quad \text{size } V \times N$$

$$x = \begin{bmatrix} \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{bmatrix} \quad \text{size } V \times 1$$

Option 3: Take average of both matrix

$$\mathbf{W}_3 = 0.5 (\mathbf{W}_1 + \mathbf{W}_2^T) = \begin{bmatrix} \boxed{\mathbf{w}_3^{(1)}} & \dots & \mathbf{w}_3^{(V)} \end{bmatrix}$$


The diagram shows a matrix \mathbf{W}_3 with dimensions V (width) and N (height). The matrix is represented as a row of vectors $\mathbf{w}_3^{(1)}, \dots, \mathbf{w}_3^{(V)}$. The first vector $\mathbf{w}_3^{(1)}$ is highlighted with a blue box. A horizontal double-headed arrow below the matrix indicates the width V , and a vertical double-headed arrow to the right indicates the height N .

Evaluating Word Embeddings:

1. Intrinsic Evaluation: allows you to test relationships between words. It allows you to capture semantic analogies as, "France" is to "Paris" as "Italy" is to $\langle ? \rangle$ and also syntactic analogies as "seen" is to "saw" as "been" is to $\langle ? \rangle$.

Few ways that allow to use intrinsic evaluation:

- a. Analogies
 - b. Clustering
 - c. Visualization
2. Extrinsic Evaluation: tests word embeddings on external tasks like named entity recognition, parts-of-speech tagging, etc.
Evaluates actual usefulness of embeddings
Time Consuming
More difficult to trouble shoot

SEQUENCE TO SEQUENCE MODEL

Sequence-to-Sequence (Seq2Seq) models are a type of neural network architecture designed to transform one sequence into another, making them particularly useful for tasks where the input and output are sequences of different lengths and structures.

Seq2Seq models typically consist of two main components:

1. Encoder: Processes the input sequence and encodes it into a fixed-length context vector (also called a thought vector or latent vector). The encoder is usually an RNN, LSTM, or GRU that reads the input sequence one token at a time and produces hidden states.
2. Decoder: Generates the output sequence from the context vector. The decoder is another RNN, LSTM, or GRU that predicts the next token in the sequence based on the context vector and its own previous hidden states and outputs.

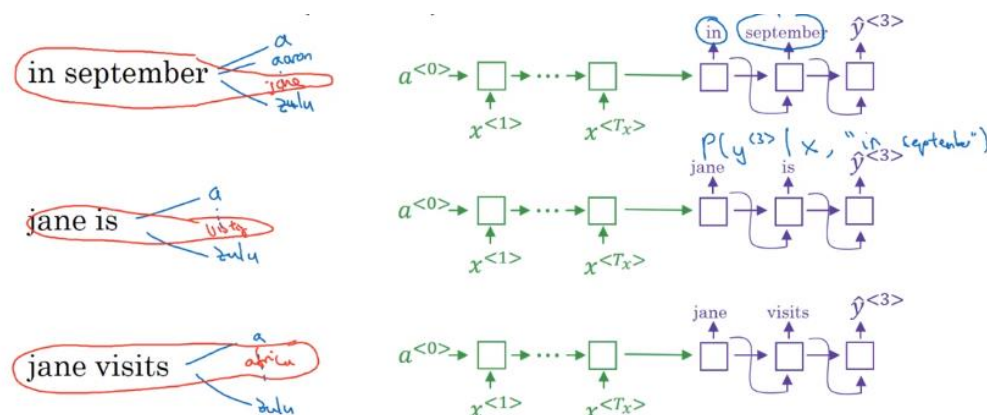
BEAM SEARCH

Beam Search is a heuristic search algorithm used in sequence generation tasks to efficiently explore possible sequences while maintaining a balance between completeness and computational feasibility. It is particularly useful in sequence-to-sequence models for tasks like machine translation, text summarization, and image captioning.

Beam Width (k): Beam Search maintains a fixed number of top candidate sequences (k) at each step.

Pruning: The algorithm retains only the top k sequences with the highest scores, discarding the rest.

Termination: The search process continues until a predefined stopping criterion is met, such as reaching a maximum sequence length or generating an end token for a sufficient number of sequences.



BLEU SCORE

The BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate the quality of text generated by machine translation systems by comparing it to one or more reference translations. It measures how closely the machine-generated text matches the reference texts.

N-gram Precision: BLEU calculates precision for different n-grams (contiguous sequences of n items) by comparing the n-grams in the generated text to those in the reference texts.

Brevity Penalty: To discourage generating overly short translations, BLEU includes a brevity penalty.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}.$$

Final BLEU Score: The BLEU score is the geometric mean of the precision scores for the n-grams, multiplied by the brevity penalty:

$$BLEU = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right).$$

ATTENTION MODEL

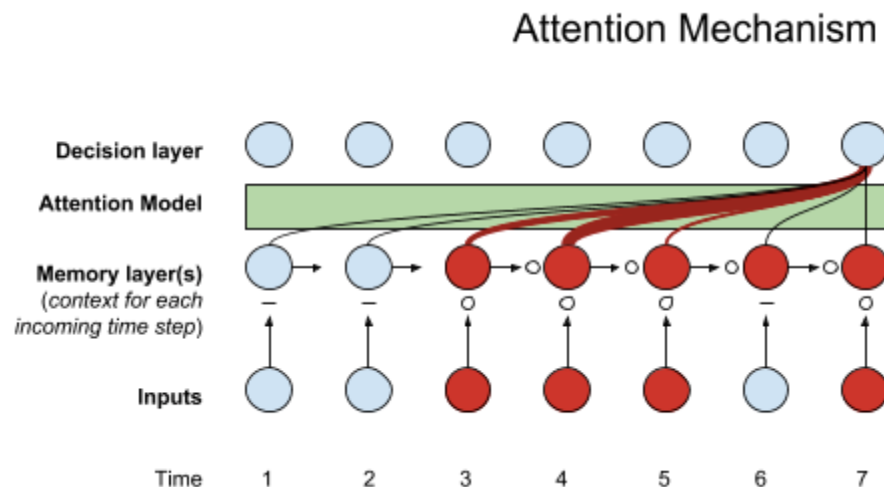
Attention models are a critical component in many advanced neural network architectures, particularly in natural language processing (NLP) and machine translation tasks. The attention mechanism allows the model to focus on different parts of the input sequence when generating each part of the output sequence.

Types of Attention Mechanisms

1. **Additive Attention:** Introduced by Bahdanau et al., this method computes alignment scores using a feedforward neural network that combines the decoder's previous hidden state and the encoder's hidden states.
2. **Multiplicative Attention:** Introduced by Luong et al., this method computes alignment scores using the dot product between the encoder's hidden states and the decoder's current hidden state.

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \begin{cases} \mathbf{h}_t^\top \mathbf{W} \bar{\mathbf{h}}_s & \text{[Luong's multiplicative style]} \\ \mathbf{v}_a^\top \tanh(\mathbf{W}_1 \mathbf{h}_t + \mathbf{W}_2 \bar{\mathbf{h}}_s) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

3. **Scaled Dot-Product Attention:** Commonly used in Transformer models, this method scales the dot product of the queries and keys by the square root of the dimensionality of the keys.



CTC COST FOR SPEECH RECOGNITION

Connectionist Temporal Classification (CTC) is a popular loss function used in training neural networks for sequence-to-sequence tasks where the alignment between input and output sequences is unknown, such as speech recognition.

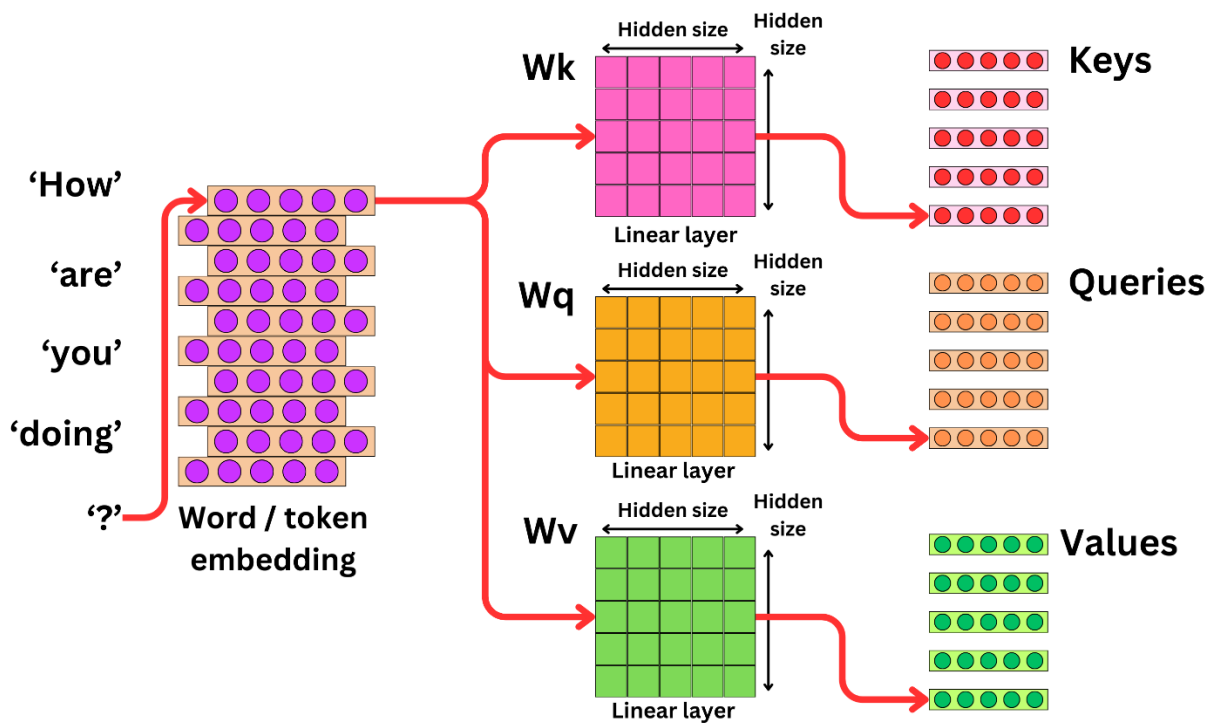
TRANSFORMER

The Transformer is a deep learning model introduced in the paper "Attention is All You Need".

Key components include:

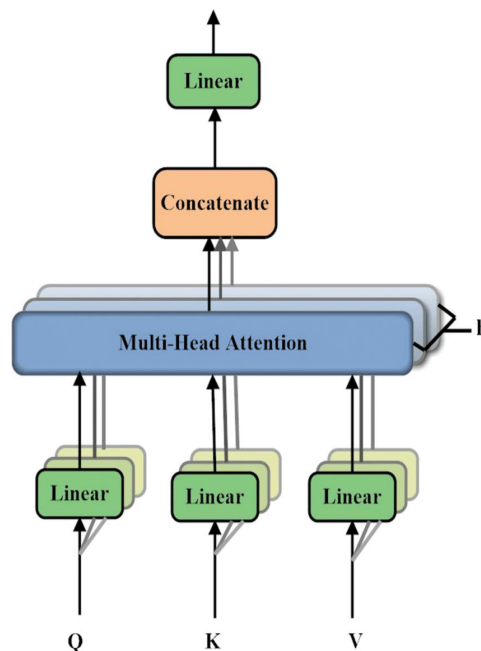
Self-attention mechanism

The self-attention mechanism is a powerful component in many modern neural network architectures, particularly in the Transformer model. It allows a model to weigh the importance of different tokens in the input sequence when generating a representation for each token.



Multi head attention

Multi-head attention is an extension of the self-attention mechanism, designed to improve the model's ability to focus on different parts of the input sequence simultaneously. It is a key component of the Transformer architecture, allowing the model to capture various aspects of the data and enhance the overall performance.



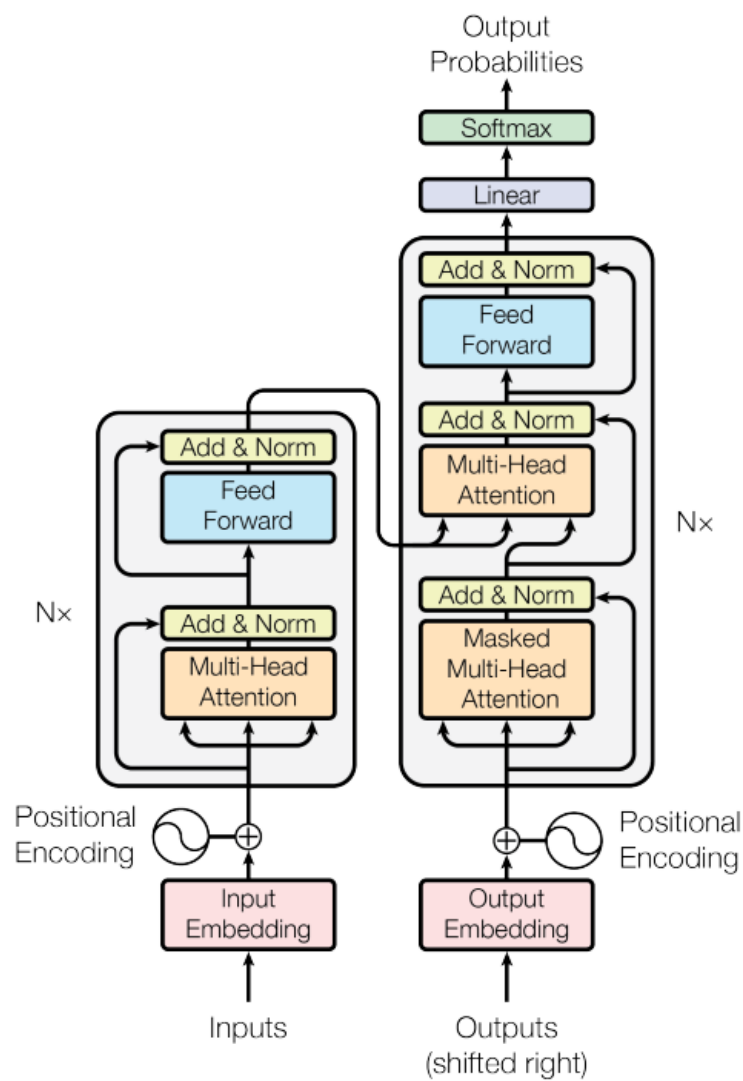
Positional Encoding

Positional encoding is a technique used in the Transformer model to inject information about the relative or absolute position of tokens in a sequence. Since Transformers do not inherently capture the

order of tokens (unlike RNNs or CNNs), positional encodings provide a way to incorporate sequence order information into the model.

$$\text{PE}_{(\text{pos}, 2i)} = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right)$$
$$\text{PE}_{(\text{pos}, 2i+1)} = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right)$$

Transformer Architecture



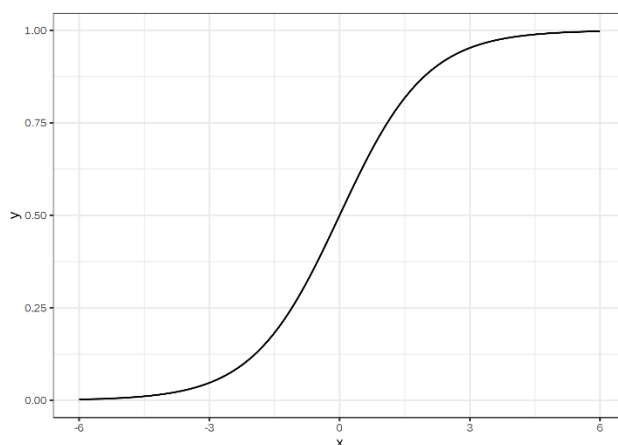
E.g.: BERT, GPT

SENTIMENT ANALYSIS

Sentiment analysis, also known as opinion mining, is the process of determining the emotional tone behind a body of text. It is a common task in natural language processing (NLP) that involves analyzing text to classify its polarity as positive, negative, or neutral.

Sentiment Analysis with Logistic Regression

1. **Feature Extraction:** Convert the text into numerical features using methods like BoW, TF-IDF, or word embeddings.
2. **Model Training:** Train a logistic regression model on the features and their corresponding sentiment labels.
3. **Prediction:** Use the trained model to predict the sentiment of new text data.



Sentiment Analysis with Naive Bayes

1. **Feature Extraction:** Similar to logistic regression, convert the text into numerical features.
2. **Model Training:** Train a Naive Bayes classifier, which is based on the assumption that the features are conditionally independent given the class label.
3. **Prediction:** Use the trained model to classify the sentiment of new text data.

Naive Bayes

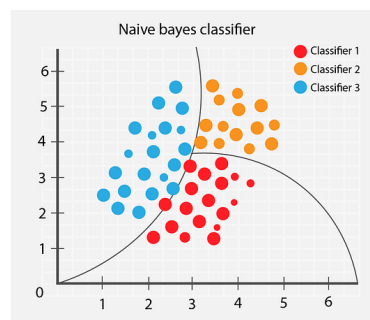


In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$



VECTOR SPACE MODEL

The Vector Space Model (VSM) is a fundamental concept in information retrieval and natural language processing. It represents text documents as vectors in a multi-dimensional space, allowing for the comparison and ranking of documents based on their similarity.

1. Word by Word design
2. Word by Document Design

Similarity is measured by

1. Euclidean Distance

$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^n (v_i - w_i)^2} \longrightarrow \text{Norm of } (\vec{v} - \vec{w})$$

2. Cosine Similarity

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

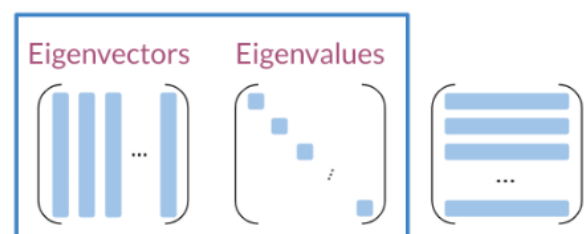
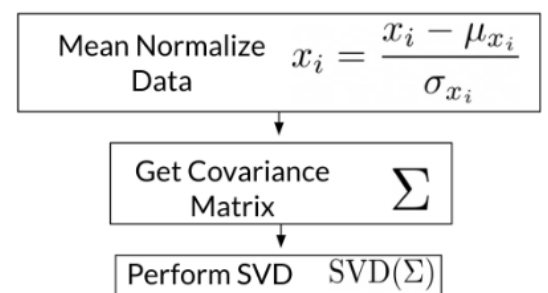
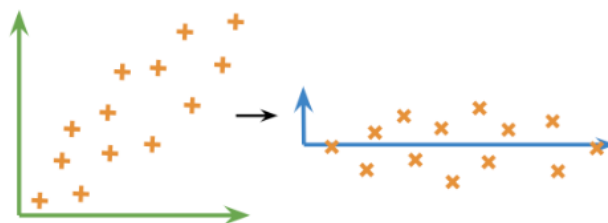
PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a powerful statistical technique used for dimensionality reduction and feature extraction in data analysis. It transforms high-dimensional data into a lower-dimensional space while preserving as much variance as possible.

Steps to Compute PCA:

- Mean normalize your data
- Compute the covariance matrix
- Compute SVD on your covariance matrix. This returns $[USV] = \text{svd}(\Sigma)$. The three matrices U , S , V are drawn above. U is labelled with eigenvectors, and S is labelled with eigenvalues.
- You can then use the first n columns of vector U , to get your new data by multiplying $XU[:,0:n]$.

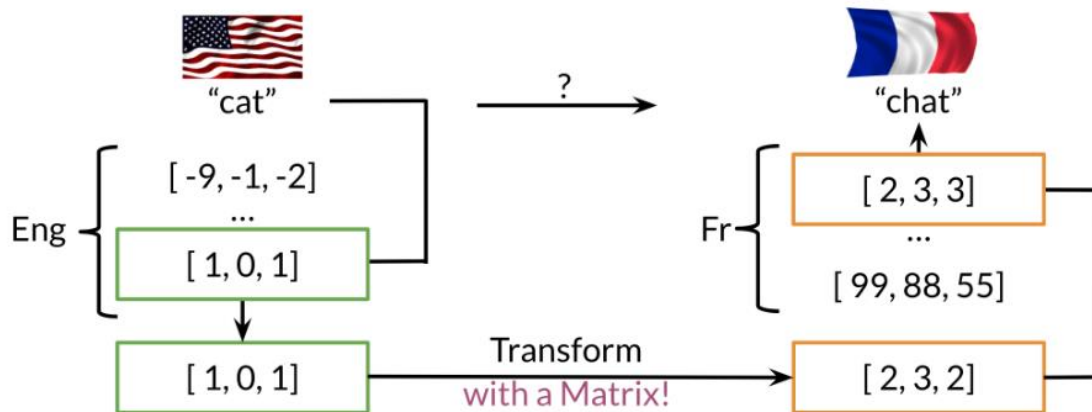
PCA algorithm



MACHINE TRANSLATION

Transforming Word Vectors

Here we need to learn how to take a word vector and learn a mapping that will allow you to translate the words by learning a transformation matrix.



Here is a visualization of that showing you the aligned vectors:

$$\begin{pmatrix} \text{["cat" vector]} \\ \text{[... vector]} \\ \text{["zebra" vector]} \end{pmatrix} \mathbf{X} \quad \mathbf{XR} \approx \mathbf{Y} \quad \begin{pmatrix} \text{["chat" vecteur]} \\ \text{[... vecteur]} \\ \text{["z bresse" vecteur]} \end{pmatrix} \mathbf{Y}$$

subsets of the full vocabulary

Steps required to learn R

- Initialize R
- For loop

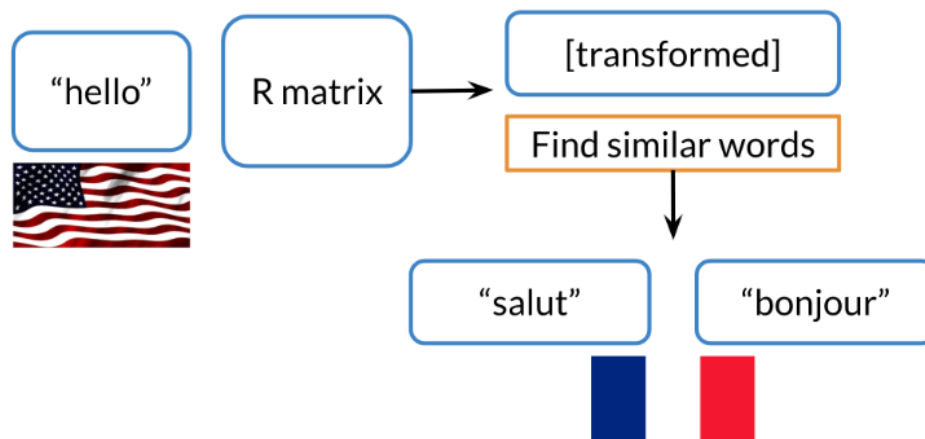
$$Loss = \|XR - Y\|_F$$

$$g = \frac{d}{dR} Loss$$

$$R = R - \alpha * g$$

K nearest neighbors

After computing the output of XR, we get a vector. We need to find the most similar vectors of our output.

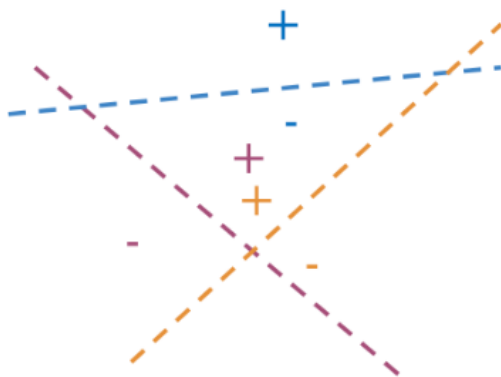


Hash Table

A hash table is a data structure that provides fast and efficient data retrieval. It stores data in an array-like structure, where each data item is mapped to a unique index using a hash function. This allows for constant time complexity, $O(1)$, for both insertion and lookup operations in average cases.

Locality Sensitive Hashing

Locality sensitive hashing is a technique that allows you to hash similar inputs into the same buckets with high probability.



$$\mathbf{P}_1 \mathbf{v}^T = 3, \text{sign}_1 = +1, h_1 = 1$$

$$\mathbf{P}_2 \mathbf{v}^T = 5, \text{sign}_2 = +1, h_2 = 1$$

$$\mathbf{P}_3 \mathbf{v}^T = -2, \text{sign}_3 = -1, h_3 = 0$$

$$\begin{aligned} \text{hash} &= 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3 \\ &= 1 \times 1 + 2 \times 1 + 4 \times 0 \end{aligned}$$

$$= 3$$

AUTO CORRECT

Building the model

1. Identify the misspelled word

When identifying the misspelled word, you can check whether it is in the vocabulary. If you don't find it, then it is probably a typo.

2. Find strings n edit distance away

- Edit: an operation performed on a string to change it
- Insert (add a letter) 'to': 'top', 'two' ...
- Delete (remove a letter) 'hat': 'ha', 'at', 'ht'
- Switch (swap 2 adjacent letters) 'eta': 'eat', 'tea'
- Replace (change 1 letter to another) 'jaw': 'jar', 'paw', ...

3. Filter candidates

In this step, you want to take all the words generated above and then only keep the actual words that make sense and that you can find in your vocabulary.

<u>deah</u>	<u>deah</u>
_eah	yeah
d_ar	→ dear
de_r	dean
... etc	... etc

4. Calculate word probabilities

$$P(w) = \frac{C(w)}{V}$$

$P(w)$ Probability of a word

$C(w)$ Number of times the word appears

V Total size of the corpus

MINIMUM EDIT DISTANCE

Minimum edit distance allows you to:

- Evaluate similarity between two strings
- Find the minimum number of edits between two strings
- Implement spelling correction, document similarity, machine translation, DNA sequencing, and more

Remember that the edits include:

- Insert (add a letter) 'to': 'top', 'two' ...
- Delete (remove a letter) 'hat': 'ha', 'at', 'ht'
- Replace (change 1 letter to another) 'jaw': 'jar', 'paw', ...

Edit cost:

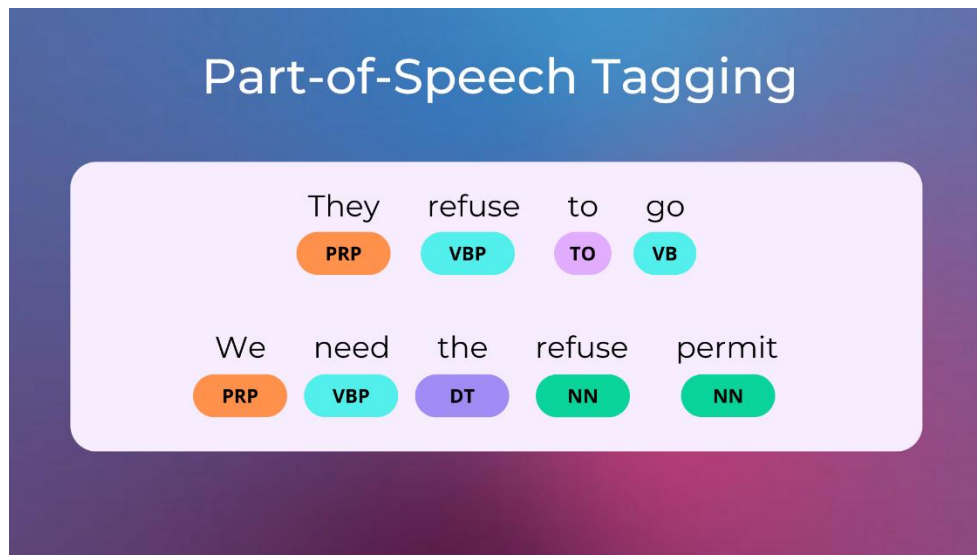
Insert 1

Delete 1

Replace 2

PARTS OF SPEECH TAGGING

Parts of speech (POS) tagging is the process of assigning grammatical labels (tags) to words in a text corpus based on their syntactic roles within sentences.



POS tagging is essential in natural language processing tasks such as syntactic parsing, information extraction, machine translation, and sentiment analysis. It helps in understanding sentence structure, disambiguating word meanings, and improving the accuracy of downstream NLP applications.

MARKOV CHAINS

A Markov chain is a mathematical model that describes a system which transitions between states according to certain probabilistic rules.

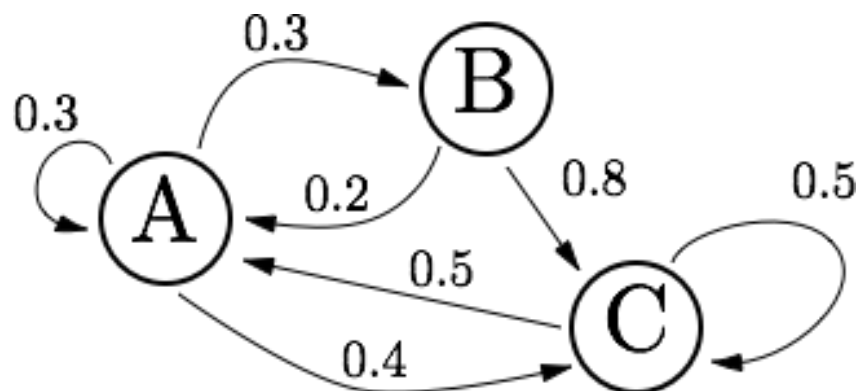
You can use Markov chains to identify the probability of the next word. For example, below, you can see that the most likely word after a verb is a noun.

Why not learn swimming?

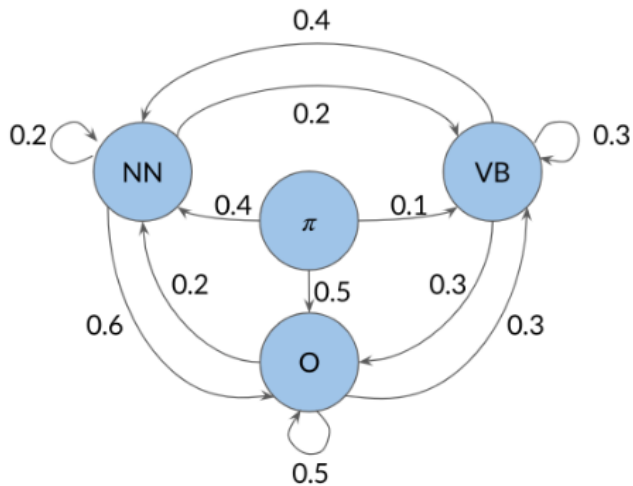
verb noun

Why not learn swim?

verb verb



Transition Matrix



$A =$

	NN	VB	O
π (initial)	0.4	0.1	0.5
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

In more general notation, you can write the transition matrix A , given some states Q , as follows:

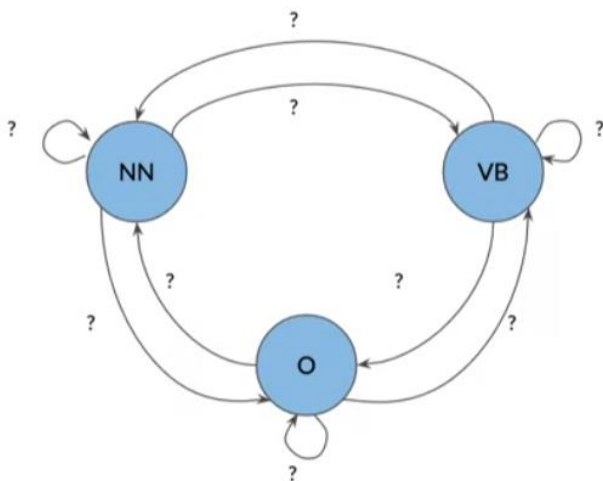
States

$$Q = \{q_1, \dots, q_N\}$$

Transition matrix

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \dots & a_{N+1,N} \end{pmatrix}$$

Calculating Transition probabilities



1. Count occurrences of tag pairs

$$C(t_{i-1}, t_i)$$

2. Calculate probabilities using the counts

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^N C(t_{i-1}, t_j)}$$

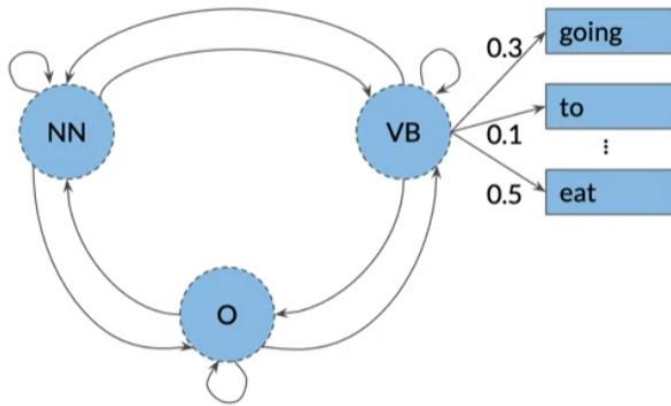
Unfortunately, sometimes you might not see two POS tags in front each other. This will give you a probability of 0. To solve this issue, you will "smooth" it as follows:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \epsilon}{\sum_{j=1}^N C(t_{i-1}, t_j) + N * \epsilon}$$

HIDDEN MARKOV MODEL

A Hidden Markov Model (HMM) is a probabilistic model used to model sequential data where the underlying system is assumed to be a Markov process with unobservable ("hidden") states.

Emission probabilities



$B =$

	going	to	eat	...
NN (noun)	0.5	0.1	0.02	
VB (verb)	0.3	0.1	0.5	
O (other)	0.3	0.5	0.68	

States

Transition matrix

Emission matrix

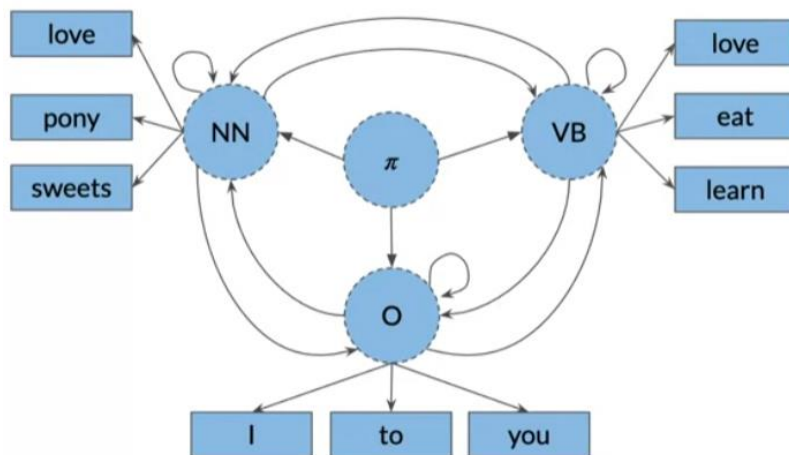
$$Q = \{q_1, \dots, q_N\} \quad A = \begin{pmatrix} a_{1,1} & \dots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{N+1,1} & \dots & a_{N+1,N} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & \dots & b_{1V} \\ \vdots & \ddots & \vdots \\ b_{N1} & \dots & b_{NV} \end{pmatrix}$$

$$\sum_{j=1}^V b_{ij} = 1$$

Calculating emission matrix probabilities

$$\begin{aligned} P(w_i | t_i) &= \frac{C(t_i, w_i) + \epsilon}{\sum_{j=1}^V C(t_i, w_j) + N * \epsilon} \\ &= \frac{C(t_i, w_i) + \epsilon}{C(t_i) + N * \epsilon} \end{aligned}$$

THE VITERBI ALGORITHM



<s> I love to learn

It has three main steps:

1. Initialization step
2. Forward Pass
3. Backward Pass

Initialization Step

C → intermediate optimal probability

$$C = \begin{array}{c|cccc} & w_1 & w_2 & \dots & w_K \\ \hline t_1 & c_{1,1} & & & \\ \hline \dots & & & & \\ \hline t_N & c_{N,1} & & & \end{array}$$

$$\begin{aligned} c_{i,1} &= \pi_i * b_{i, \text{cindex}(w_1)} \\ &= a_{1,i} * b_{i, \text{cindex}(w_1)} \end{aligned}$$

D → indices of visited state

$$D = \begin{array}{c|cccc} & w_1 & w_2 & \dots & w_K \\ \hline t_1 & d_{1,1} & & & \\ \hline \dots & & & & \\ \hline t_N & d_{N,1} & & & \end{array}$$

$$d_{i,1} = 0$$

Forward Pass

To populate a cell (i.e. 1,2) in the image above, you have to take the max of [kth cells in the previous column, times the corresponding transition probability of the kth POS to the first POS times the emission probability of the first POS and the current word you are looking at].

$$c_{i,j} = \max_k c_{k,j-1} * a_{k,i} * b_{i, \text{index}(w_j)}$$

$$d_{i,j} = \operatorname{argmax}_k c_{k,j-1} * a_{k,i} * b_{i, \text{index}(w_j)}$$

Note that the only difference between c_{ij} and d_{ij} , is that in the former you compute the probability and, in the latter, you keep track of the index of the row where that probability came from. So, you keep track of which k was used to get that max probability.

Backward Pass

Backtrack through the states to determine the most likely sequence of hidden states.

$C =$

	w_1	w_2	w_3	w_4	w_5
t_1	0.25	0.125	0.025	0.0125	0.01
t_2	0.1	0.025	0.05	0.01	0.003
t_3	0.3	0.05	0.025	0.02	0.0000
t_4	0.2	0.1	0.000	0.0025	0.0003

$s = \operatorname{argmax}_i c_{i,K} = 1$

$D =$

	w_1	w_2	w_3	w_4	w_5
t_1	0	1	3	2	3
t_2	0	2	4	1	3
t_3	0	2	4	1	4
t_4	0	4	4	3	1

$\langle s \rangle$	w_1	w_2	w_3	w_4	w_5
π	$\leftarrow t_2$	$\leftarrow t_3$	$\leftarrow t_1$	$\leftarrow t_3$	$\leftarrow t_1$

AUTOCOMPLETE

Autocomplete is a feature commonly used in search engines, text editors, and messaging apps to predict and suggest possible completions for a partially typed word or phrase. It enhances user experience by speeding up typing, reducing errors, and providing relevant suggestions.

N-GRAMS AND PROBABILITIES

An n-gram is a sequence of n words. Punctuation is also treated as a word.

Sequence Notation

- $w_1^m = w_1 w_2 w_3 \dots w_m$
- $w_1^3 = w_1 w_2 w_3$
- $w_{m-2}^m = w_{m-2} w_{m-1} w_m$

Unigram Probability

Given the following corpus: I am happy because I am learning.

Size of corpus $m = 7$.

$$P(I) = \frac{2}{7}$$

$$P(happy) = \frac{1}{7}$$

The probability of a unigram is $P(w) = C(w)/m$

Bigram Probability

$$P(y|x) = \frac{C(x \ y)}{\sum_w C(x \ w)} = \frac{C(x \ y)}{C(x)}$$

Trigram Probability

$$P(w_3 | w_1^2) = \frac{C(w_1^2 w_3)}{C(w_1^2)}$$

$$C(w_1^2 w_3) = C(w_1 w_2 w_3) = C(w_1^3)$$

N-gram Probability

$$P(w_N | w_1^{N-1}) = \frac{C(w_1^{N-1} w_N)}{C(w_1^{N-1})}$$

$$C(w_1^{N-1} w_N) = C(w_1^N)$$

Sequence Probabilities

$$P(\text{the teacher drinks tea}) = P(\text{the})P(\text{teacher the})P(\text{drinks the teacher})P(\text{tea} | \text{the teacher drinks})$$

One of the main issues with computing the probabilities above is the corpus rarely contains the exact same phrases as the ones you computed your probabilities on. Hence, you can easily end up getting a probability of 0. The Markov assumption indicates that only the last word matters. Hence:

$$\text{Bigram} \quad P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-1})$$

$$\text{N-gram} \quad P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-N+1}^{n-1})$$

You can model the entire sentence as follows:

$$P(w_1^n) \approx \prod_{i=1}^n P(w_i | w_{i-1})$$

$$P(w_1^n) \approx P(w_1) P(w_2 | w_1) \dots P(w_n | w_{n-1})$$

THE N-GRAM LANGUAGE MODEL

Count Matrix

In the count matrix:

Rows correspond to the unique corpus N-1 grams.

Columns correspond to the unique corpus words.

- Bigram count matrix

“study I” bigram

Corpus: <s>I study I learn</s>

	<s>	</s>	I	study	learn
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	1	1
study	0	0	1	0	0
learn	0	1	0	0	0

Probability Matrix

To convert it into a probability matrix, you can use the following formula:

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})}$$

$$\text{sum}(\text{row}) = \sum_{w \in V} C(w_{n-N+1}^{n-1}, w) = C(w_{n-N+1}^{n-1})$$

Probability matrix

	<s>	</s>	I	study	learn
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	0.5	0.5
study	0	0	1	0	0
learn	0	1	0	0	0

Problems with N Gram Model

- Large N-grams needed to capture dependencies between distant words
- Need a lot of space and Ram

PERPLEXITY

Perplexity is defined as the exponentiation of the entropy of the model. For a language model, it is a measure of how surprised the model is by the test set, given the probabilities it assigns to the sequences of words.

$$PP(W) = P(s_1, s_2, \dots, s_m)^{-\frac{1}{m}}$$

$$PP(W) = \sqrt[m]{\prod_{i=1}^m \prod_{j=1}^{|s_i|} \frac{1}{P(w_j^{(i)} | w_{j-1}^{(i)})}}$$

Low Perplexity: Indicates that the model predicts the test data well. The model's probability distribution is closer to the true distribution of the data.

High Perplexity: Indicates that the model is less effective at predicting the test data. The model's probability distribution is further from the true distribution of the data.

Log Perplexity

$$\log PP(W) = -\frac{1}{m} \sum_{i=1}^m \log_2 (P(w_i | w_{i-1}))$$

SMOOTHING

The three main concepts covered here are dealing with missing n-grams, smoothing, and Backoff and interpolation.

$$P(w_n | w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})} \text{ can be 0}$$

Hence, we can add-1 smoothing as follows to fix that problem:

$$P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{\sum_{w \in V} (C(w_{n-1}, w) + 1)} = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$

Add-k smoothing is very similar:

$$P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{\sum_{w \in V} (C(w_{n-1}, w) + k)} = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + k * V}$$

When using back-off:

If N-gram missing => use (N-1)-gram, ...: Using the lower-level N-grams (i.e. (N-1)-gram, (N-2)-gram, down to unigram) distorts the probability distribution. Especially for smaller corpora, some probability needs to be discounted from higher level N-grams to use it for lower-level N-grams.

Probability discounting e.g. Katz backoff: makes use of discounting.

"Stupid" backoff: If the higher order N-gram probability is missing, the lower order N-gram probability is used, just multiplied by a constant. A constant of about 0.4 was experimentally shown to work well.

You can also use interpolation when computing probabilities as follows:

$$\hat{P}(w_n | w_{n-2}w_{n-1}) = \lambda_1 \times P(w_n | w_{n-2}w_{n-1}) + \lambda_2 \times P(w_n | w_{n-1}) + \lambda_3 \times P(w_n)$$

$$\sum_i \lambda_i = 1$$

NAMED ENTITY RECOGNITION (NER)

Named Entity Recognition (NER) is a crucial task in Natural Language Processing (NLP) that involves identifying and classifying named entities in text into predefined categories such as names of persons, organizations, locations, dates, and other proper nouns. NER is widely used in information retrieval, question answering, machine translation, and many other applications.

SIAMESE NETWORK

A Siamese network is a type of neural network architecture that is designed to learn similarity metrics between pairs of inputs. It consists of two or more identical subnetworks that share the same parameters and weights. The network is particularly useful for tasks that involve comparing or finding relationships between two inputs, such as signature verification, face recognition, and sentence similarity.

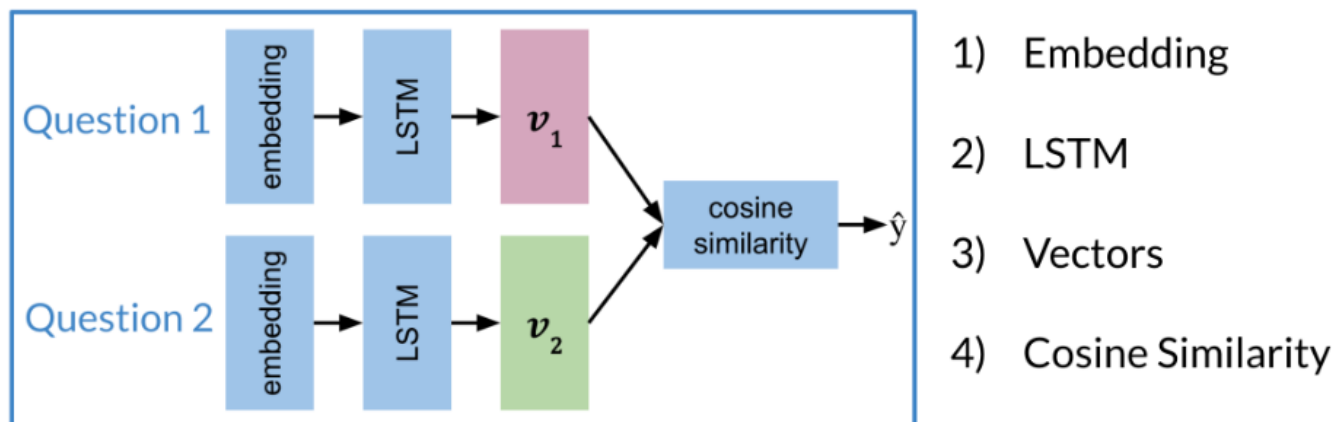
Key Concepts

Twin Networks: The architecture consists of two or more subnetworks (often called "twin networks") with the same architecture and shared weights.

Shared Weights: The subnetworks share the same weights, ensuring that the features learned are consistent and comparable across the inputs.

Distance Metric: The output of the subnetworks is typically compared using a distance metric (e.g., Euclidean distance, cosine similarity) to determine the similarity between the inputs.

Architecture:



Loss Function:

How old are you?

Anchor

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1|| ||v_2||}$$

What is your age?

Positive

$$\cos(A, P) \approx 1$$

Where are you from?

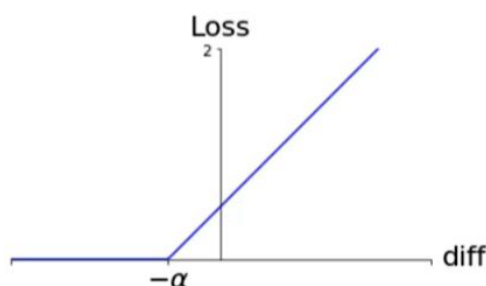
Negative

$$\cos(A, N) \approx -1$$

$$-\cos(A, P) + \cos(A, N) \leq 0$$

Triplets:

Triplet Loss



Simple loss:

$$\text{diff} = s(A, N) - s(A, P)$$

With non-linearity

$$\mathcal{L} = \begin{cases} 0; & \text{if } \text{diff} \leq 0 \\ \text{diff}; & \text{if } \text{diff} > 0 \end{cases}$$

With alpha margin

$$\mathcal{L} = \begin{cases} 0; & \text{if } \text{diff} + \alpha \leq 0 \\ \text{diff} + \alpha; & \text{if } \text{diff} + \alpha > 0 \end{cases}$$

Triplet Selection

Triplet A, P, N

$\left\{ \begin{array}{ll} \text{duplicate set:} & A, P \\ \text{non-duplicate set:} & A, N \end{array} \right.$

$$\mathcal{L} = \max(\text{diff} + \alpha, 0)$$

Random

$$\text{diff} = s(A, N) - s(A, P)$$

Easy to satisfy. Little to learn

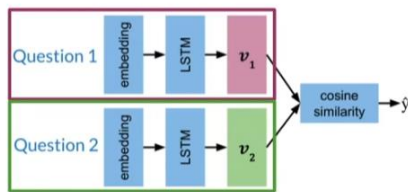
Hard

$$s(A, N) \approx s(A, P)$$

Harder to train. More to learn

Cost and Loss computation:

Computing The Cost



Batch 1
 What is your age?
 Can you see me?
 Where are thou?
 When is the game?

Batch 2
 How old are you?
 Are you seeing me?
 Where are you?
 What time is the game?

$v_1 = (1, d_model)$

$v_{1,1}$				
$v_{1,2}$				
$v_{1,3}$				
$v_{1,4}$				

v_2

$v_{2,1}$				
$v_{2,2}$				
$v_{2,3}$				
$v_{2,4}$				

$s(v_1, v_2)$

		v_1			
		_1	_2	_3	_4
v_2	_1	0.9	-0.8	0.3	-0.5
	_2	-0.8	0.5	0.1	-0.2
	_3	0.3	0.1	0.7	-0.8
	_4	-0.5	-0.2	-0.8	1.0

$$\mathcal{L}(A, P, N) = \max(\text{diff} + \alpha, 0)$$

$$\text{diff} = s(A, N) - s(A, P)$$

$$\mathcal{J} = \sum_{i=1}^m \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

Mean Negative: which is the mean negative of all the other off diagonals in the row

Closest Negative: which corresponds to the highest number in the off diagonals

$$\text{Cost} = \max(-\cos(A, P) + \cos(A, N) + \alpha, 0)$$

$$\text{Cost1} = \max(-\cos(A, P) + \text{mean_neg} + \alpha, 0)$$

$$\text{Cost2} = \max(-\cos(A, P) + \text{closest_neg} + \alpha, 0)$$

The full cost is defined as: Cost 1 + Cost 2.

PROJECT INITIATION: ROUGH APPROACH

Our aim is to make a model which will analyze the input speech data, and then express the emotion present in the speech audio file.

We will be given with a raw audio file, on which we have to implement our model.

DATA PRE-PROCESSING STEPS

Some data pre-processing steps include:

1. DATA COLLECTION

- a. We need a diverse set of audio recordings, representing various emotions.

2. DATA CLEANING

- a. Remove Silence: We need to remove the silence or non-speech segments from the file.
- b. Noise Reduction: To minimize the background noise and improve audio quality.

3. EMOTION REPRESENTATION

- a. Need to label all the emotions in numbers or in strings, uniformly. E.g.: '01' for 'angry' etc.

4. NORMALIZATION

- a. Amplitude Normalization: Need to normalize the amplitude to ensure consistency across all raw files.
- b. Length Normalization: Pad or Trim to a fixed length

5. RESAMPLING

- a. Need to Resample the raw data set for consistent sample rate. E.g. 10kHz for all data

FEATURE EXTRACTIONS

There are a lot of feature extraction steps which can be done on a speech data file. Some of them are:

1. PROSODIC FEATURES

- a. PITCH: to measure the highness and lowness of the voice
- b. INTENSITY: reflects the loudness of the speech
- c. JITTER: variations in the pitch

2. SPECTRAL FEATURES

- a. MFCCs: Mel Frequency Cepstral Coefficients
- b. SPECTRAL CONTRAST: difference in amplitude peaks and valleys
- c. SPECTRAL BANDWIDTH: to measure the width of spectrum

3. TEMPORAL FEATURES

- a. ZCR: Zero Crossing Rate
- b. Short Time Energy

More features can be extracted depending on the dataset and the output accuracy desired.

TRAIN TEST AND VALIDATION SETS

Need to split the data into training, testing and validation.

MODEL SELECTION AND TRAINING

We need to make the model RNN/LSTM/GRU's whichever suits the best for our time-series dataset.

The model parameters such as layers, initial weightings etc. also need to be decided.

Hyperparameters needs to be taken into considerations and a valid optimizer should be selected.

We also need to select the batch size, depending on the model and GPU performance.

MODEL EVALUATION

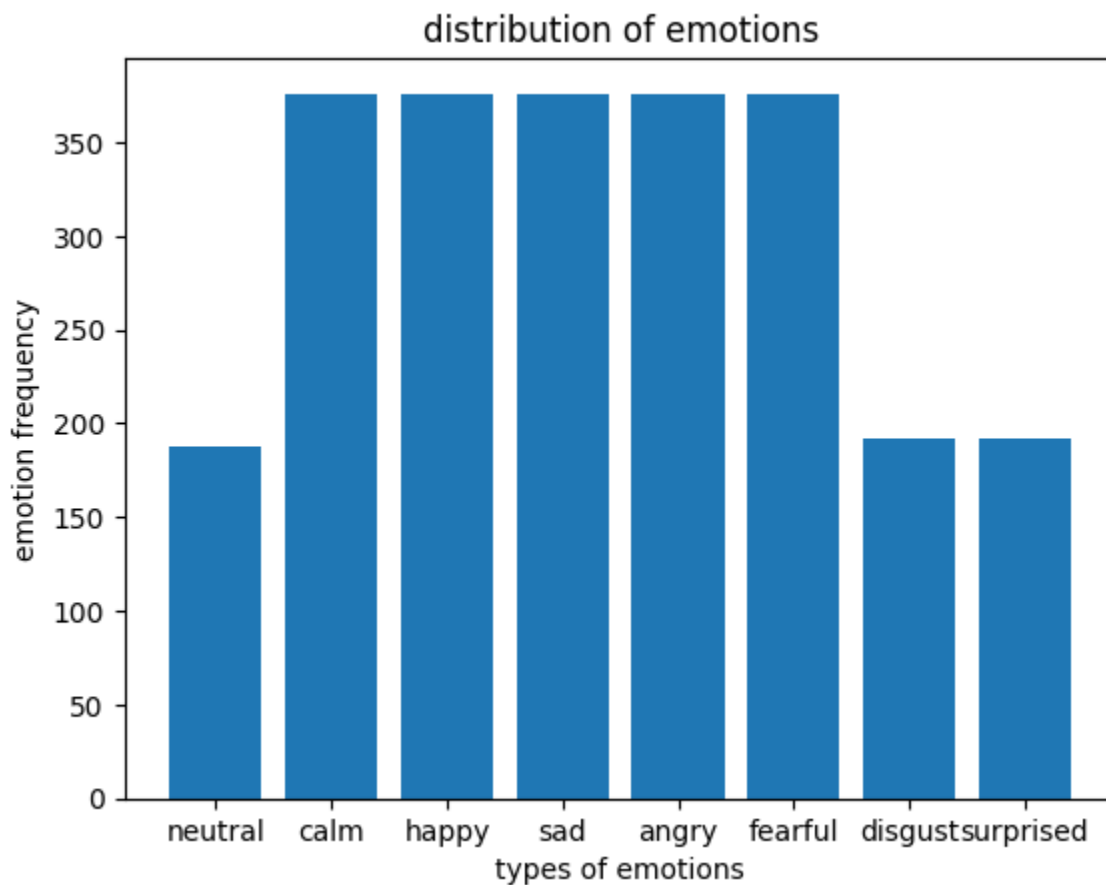
After training we need to evaluate our model on some basis like loss, accuracy score, confusion matrix for the validation and test sets

FINAL MODEL

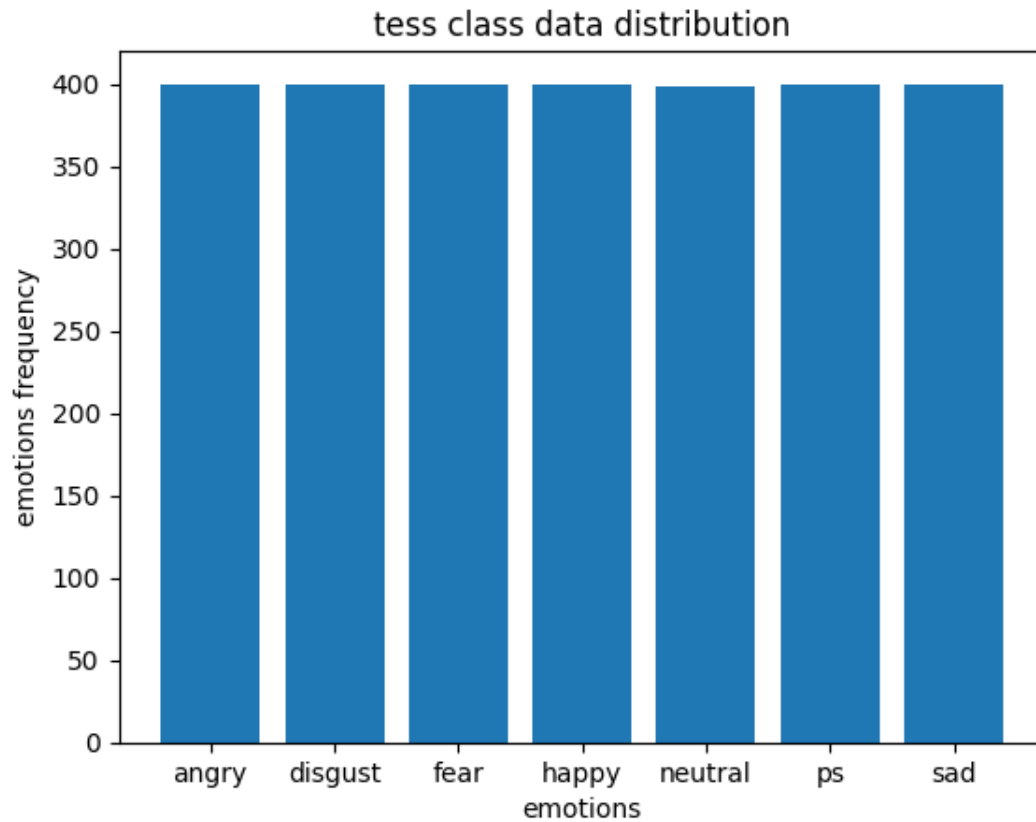
Data visualization

Distribution of emotions from RAVDESS dataset

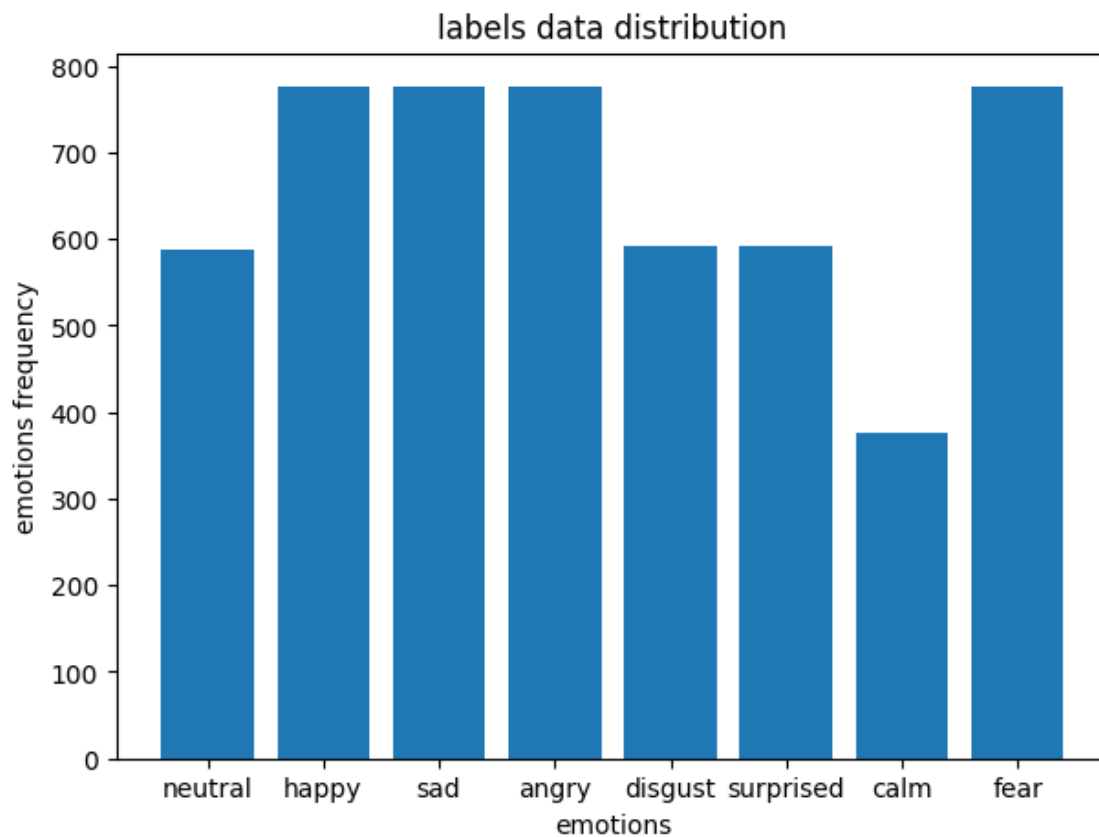
These are distribution from two dataset, RAVDESS SPEECH and RAVDESS SONG. 8 different types of emotions and their frequency can be seen in the following bar chart.



Distribution of emotions from TESS Dataset

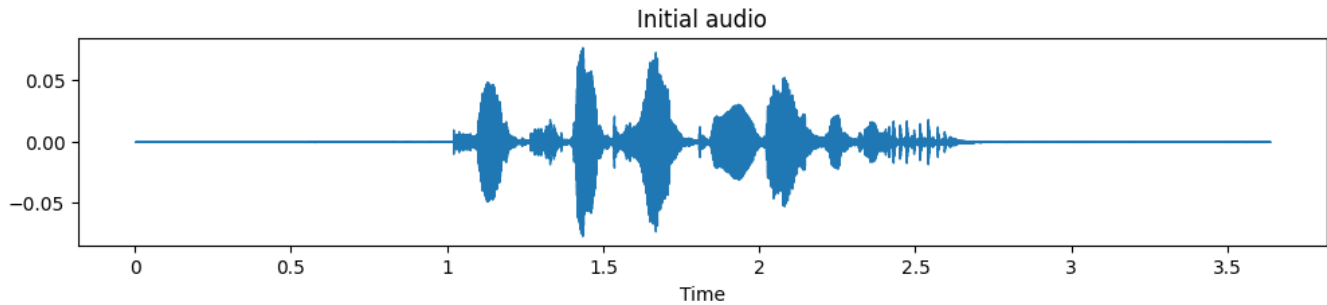


Total emotions distributions



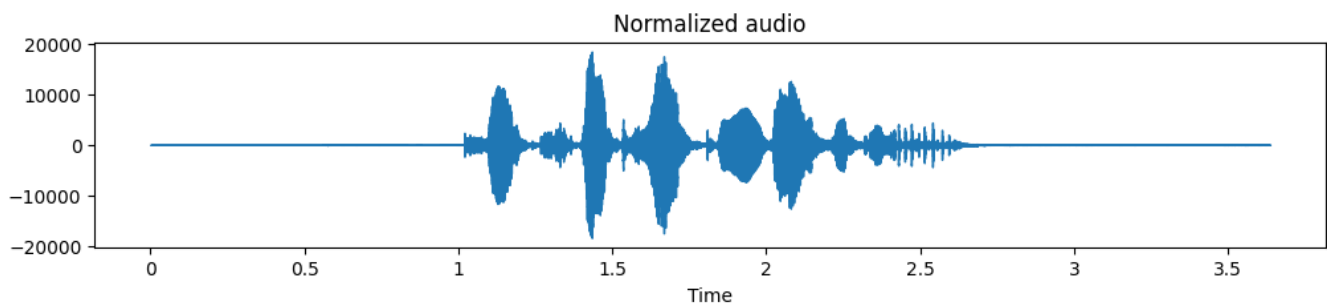
Data Pre-processing

Audio visualization



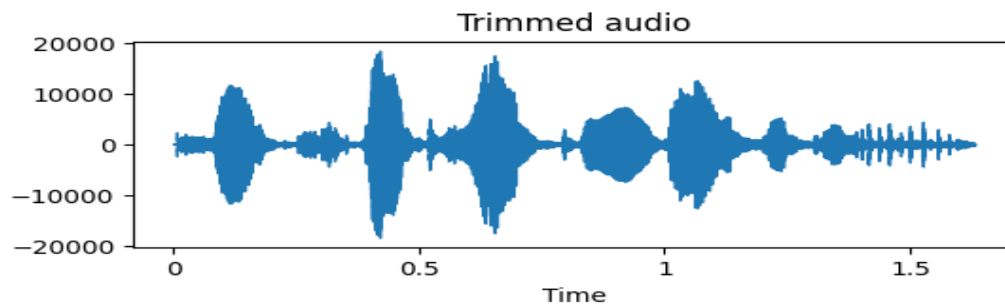
Normalization

Each 'AudioSegment' object is normalized to + 5.0 dBFS.



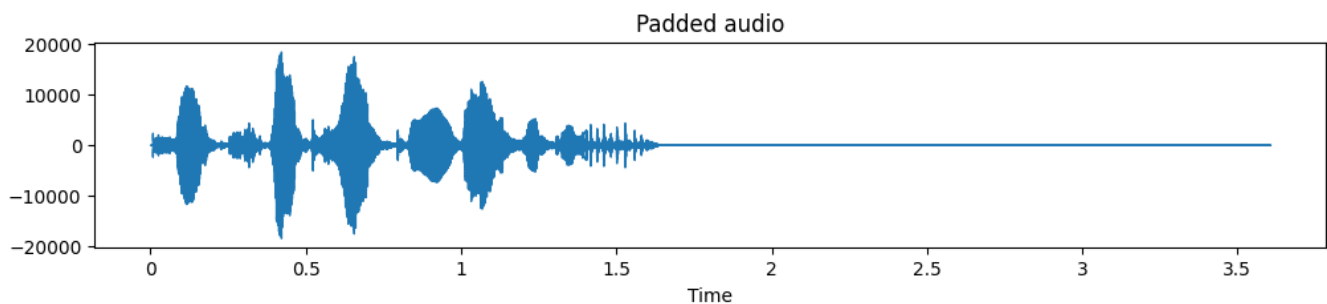
Trimmed Audio

Each audio is trimmed to remove the silence in the beginning and end.

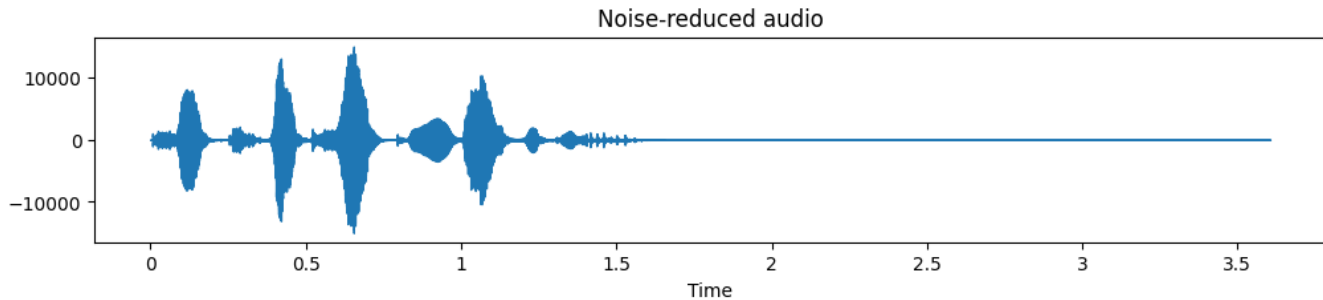


Padding

We pad every audio file to make it of same length.



Noise Reduction



All these steps are performed for every audio file.

Feature Extraction

RMS (root mean square): RMS is a measure of the energy or loudness of an audio signal.

ZCR (zero crossing rate): ZCR is the rate at which the audio signal changes sign from positive to negative or vice versa.

MFCC (Mel-Frequency Cepstral Coefficients): MFCCs are a representation of the short-term power spectrum of an audio signal, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

1. LSTM MODEL

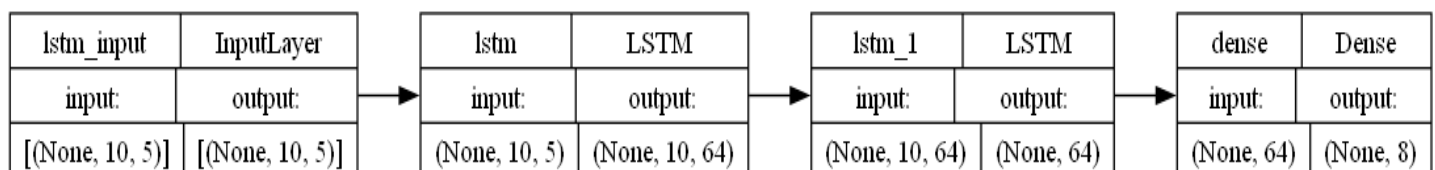
TRAINING ACCURACY → 0.9798

VALIDATION ACCURACY → 0.9522

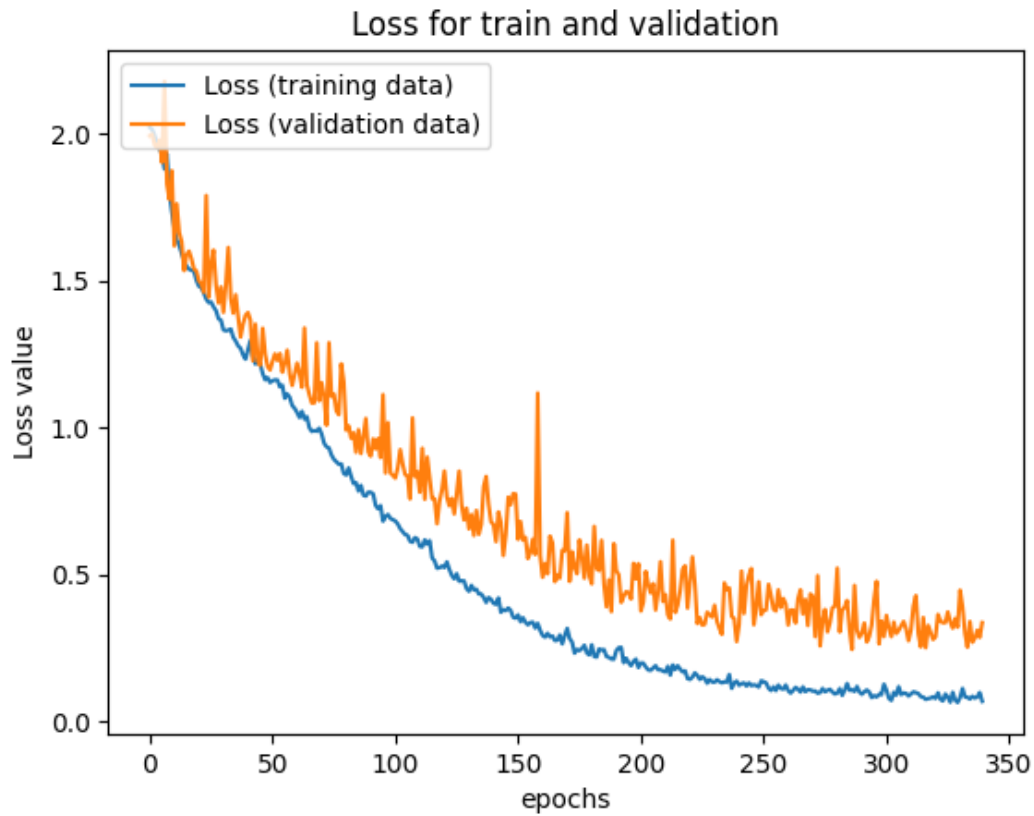
TESTING ACCURACY → 0.9078

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 476, 64)	20480
lstm_3 (LSTM)	(None, 64)	33024
dense_1 (Dense)	(None, 8)	520

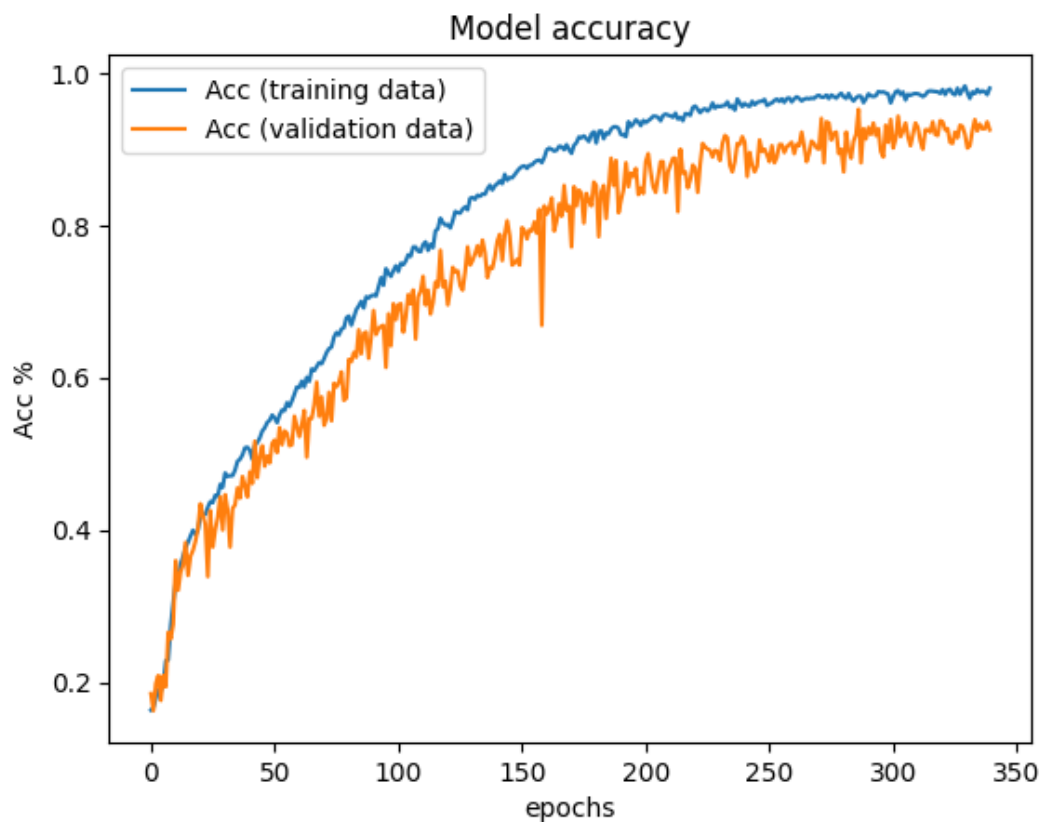
=====
Total params: 54024 (211.03 KB)
Trainable params: 54024 (211.03 KB)
Non-trainable params: 0 (0.00 Byte)



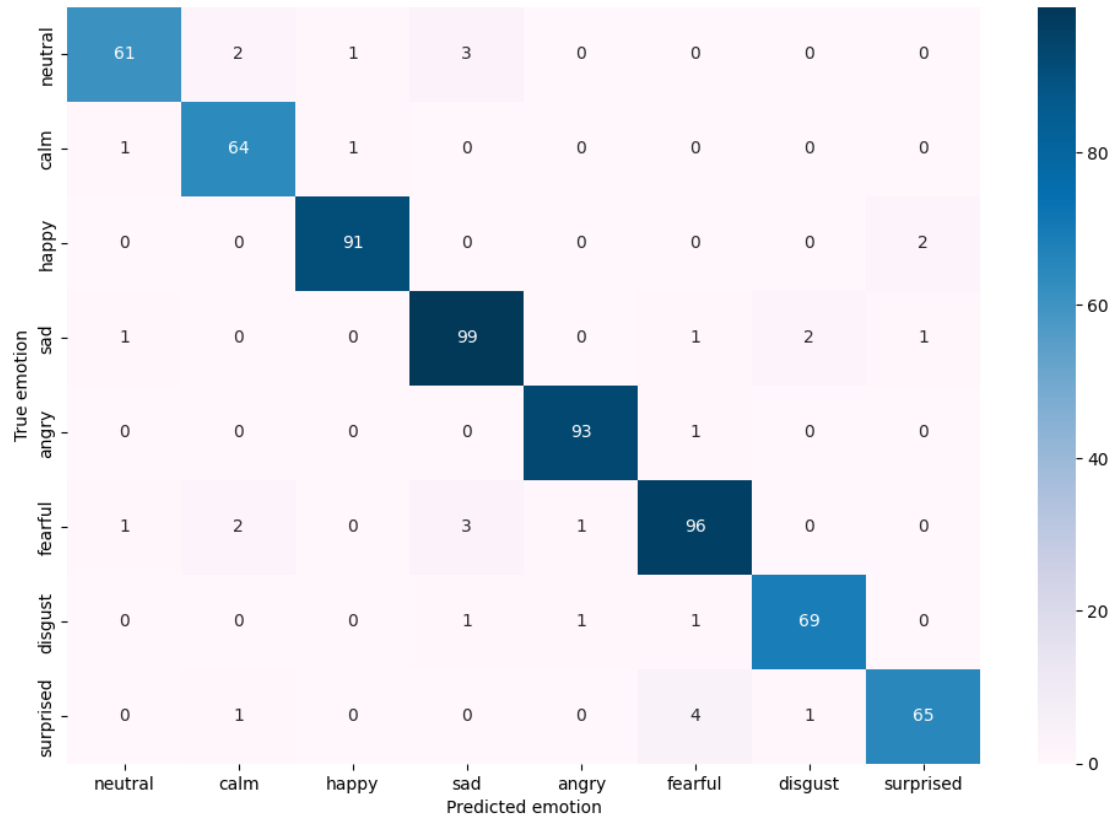
Loss graph for training and validation



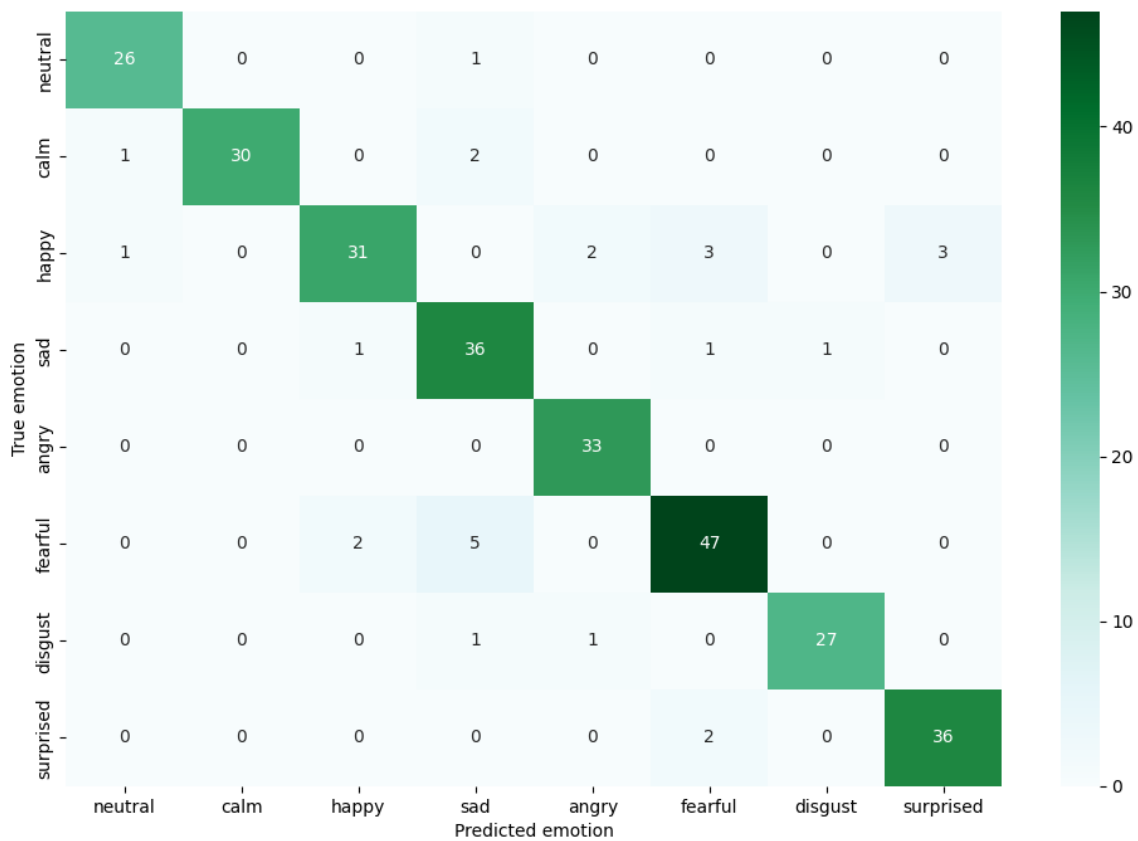
Model accuracy graph



Validation Confusion Matrix



Testing Confusion matrix



2. BIDIRECTIONAL LSTM MODEL

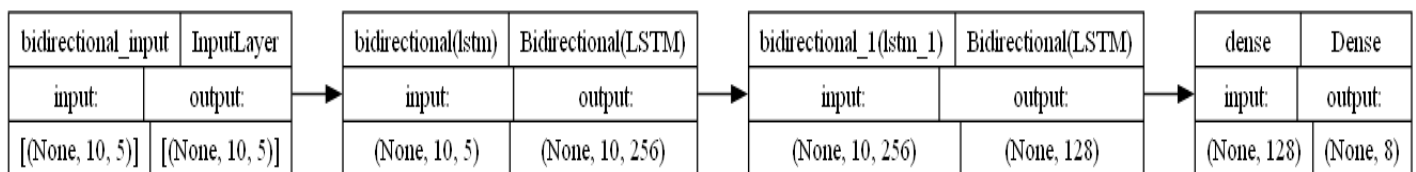
TRAINING ACCURACY → 0.9961

VALIDATION ACCURACY → 0.9627

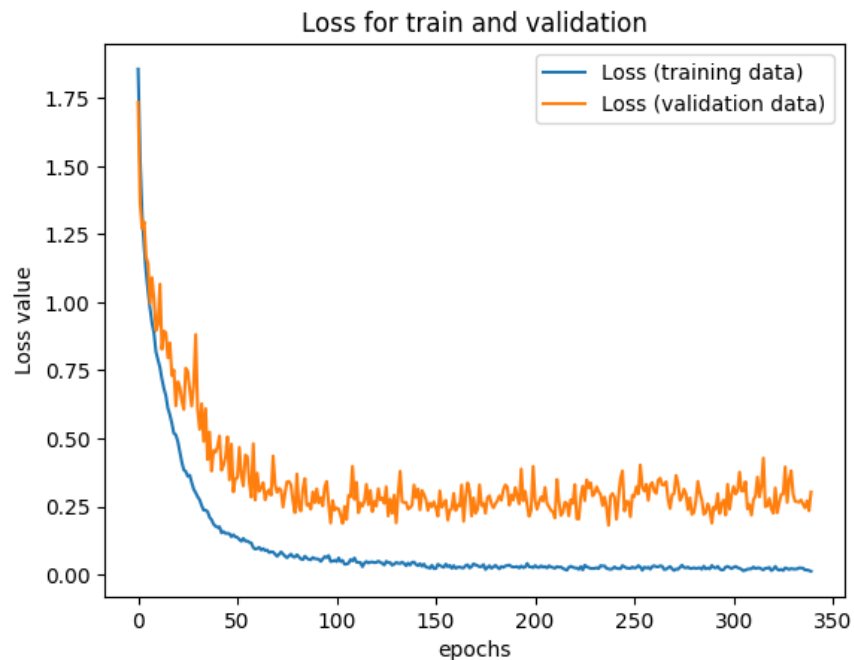
TESTING ACCURACY → 0.9352

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)	(None, 476, 256)	147456
bidirectional_3 (Bidirectional)	(None, 128)	164352
dense_1 (Dense)	(None, 8)	1032

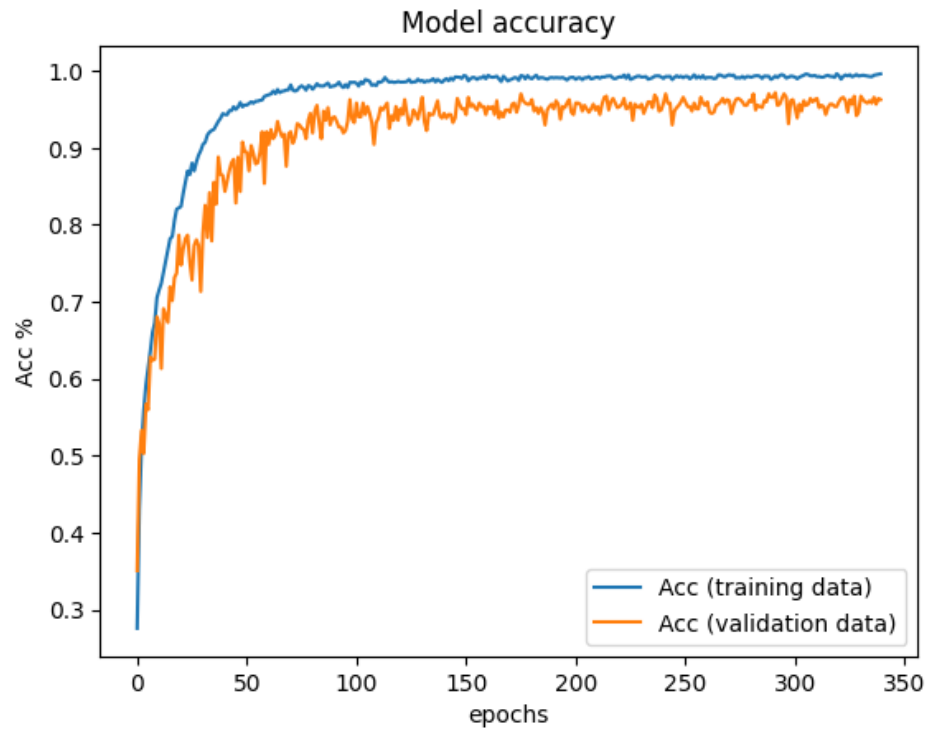
=====
Total params: 312840 (1.19 MB)
Trainable params: 312840 (1.19 MB)
Non-trainable params: 0 (0.00 Byte)



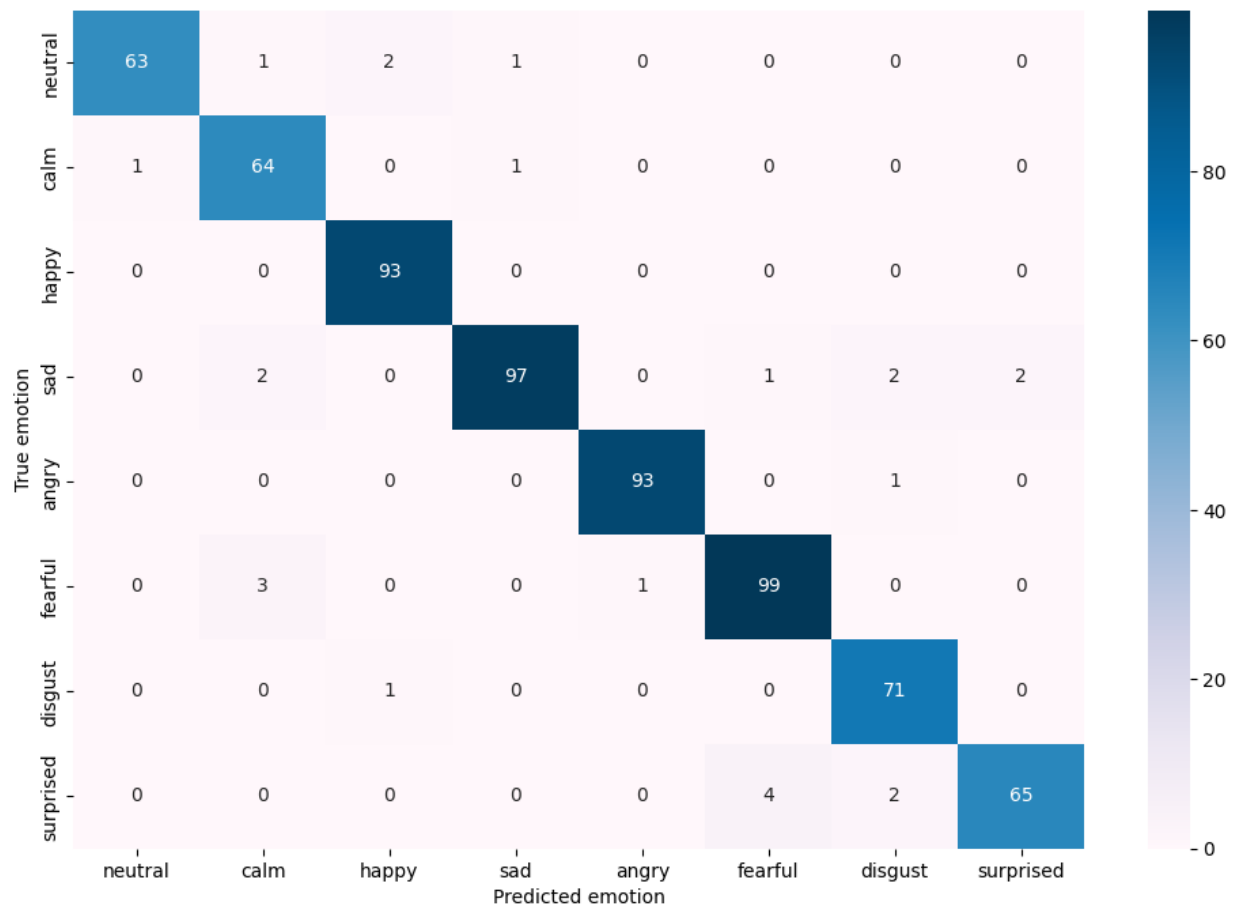
Loss graph for training and validation



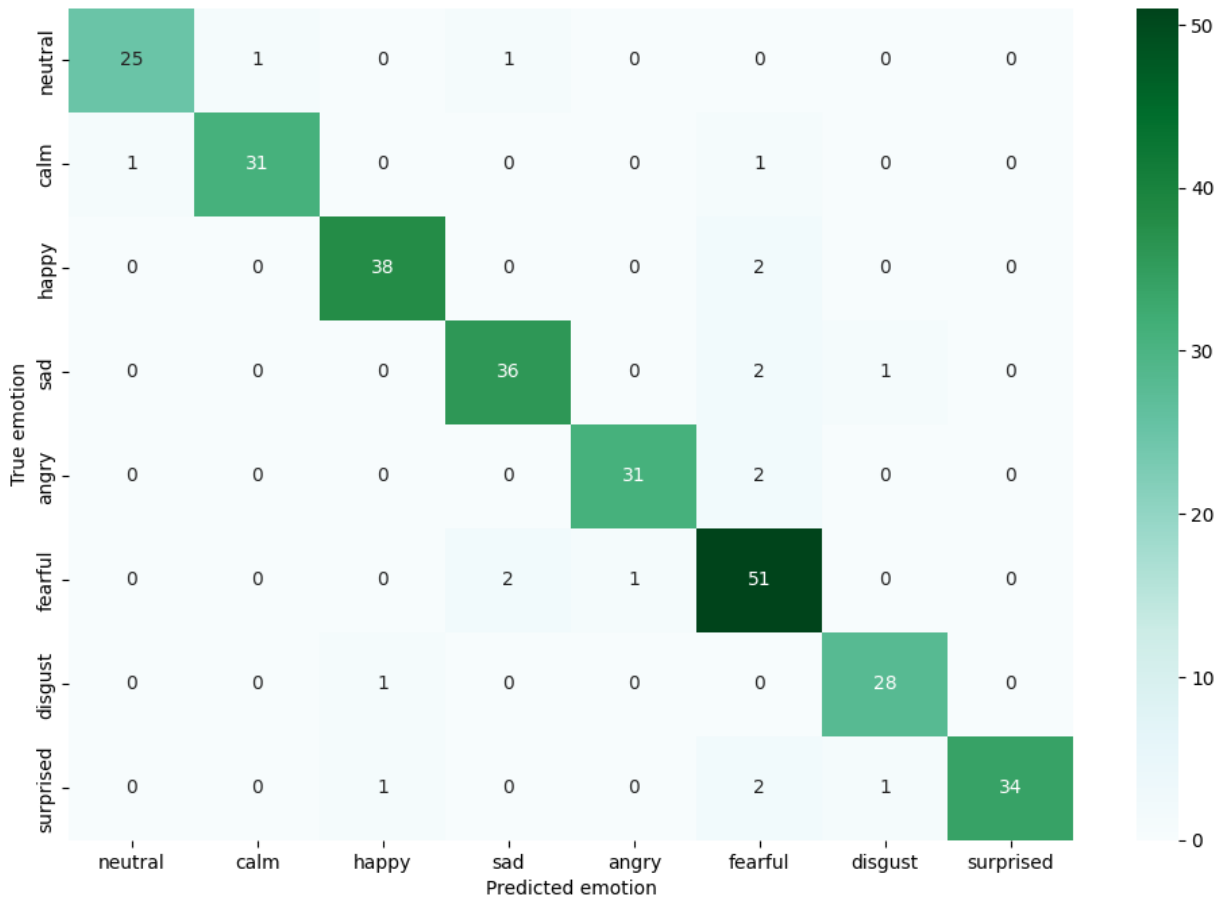
Model accuracy graph



Validation Confusion Matrix



Testing Confusion matrix



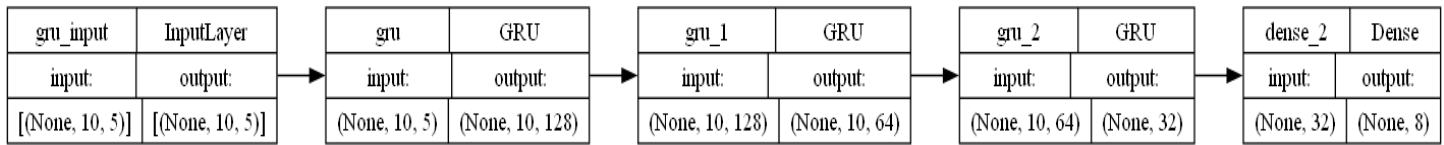
3. GRU MODEL

TRAINING ACCURACY → 0.9936

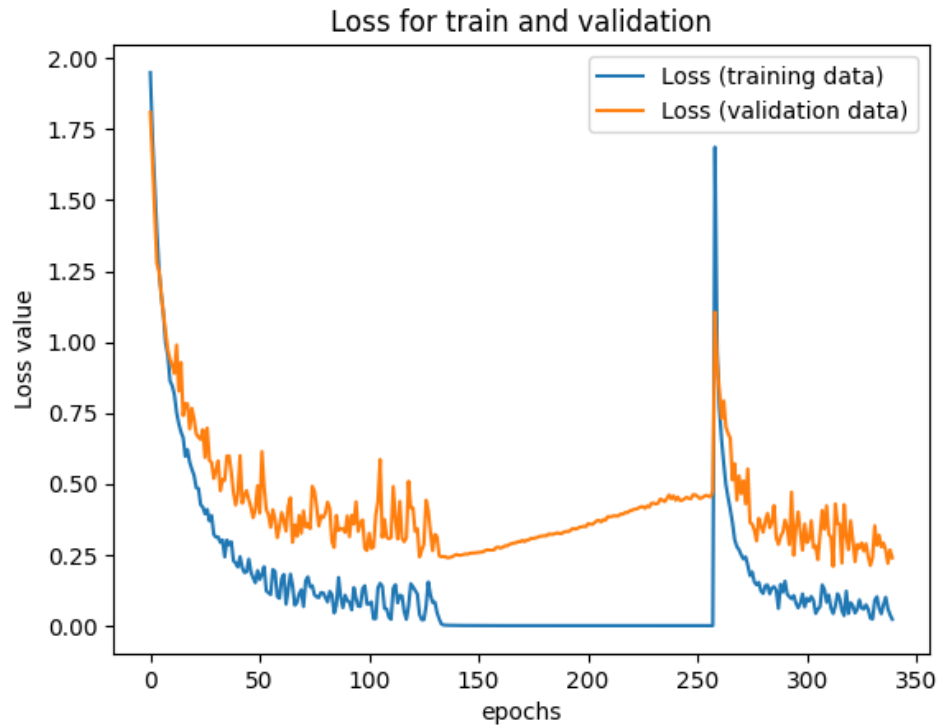
VALIDATION ACCURACY → 0.9478

TESTING ACCURACY → 0.9249

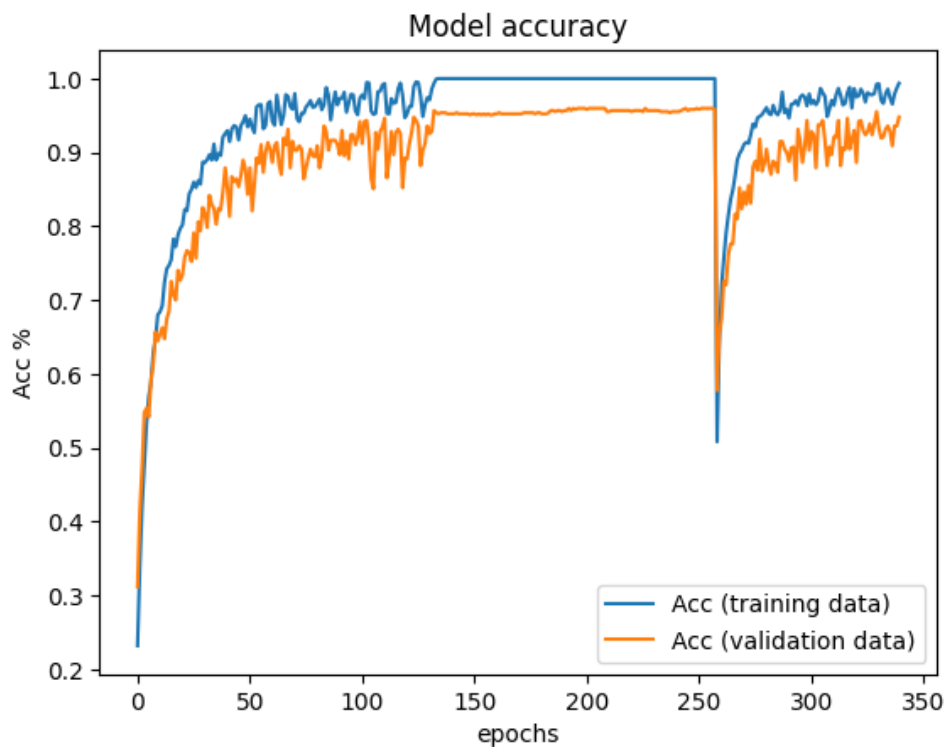
Layer (type)	Output Shape	Param #
=====		
gru_3 (GRU)	(None, 476, 128)	55680
gru_4 (GRU)	(None, 476, 64)	37248
gru_5 (GRU)	(None, 32)	9408
dense_1 (Dense)	(None, 8)	264
=====		
Total params: 102600 (400.78 KB)		
Trainable params: 102600 (400.78 KB)		
Non-trainable params: 0 (0.00 Byte)		



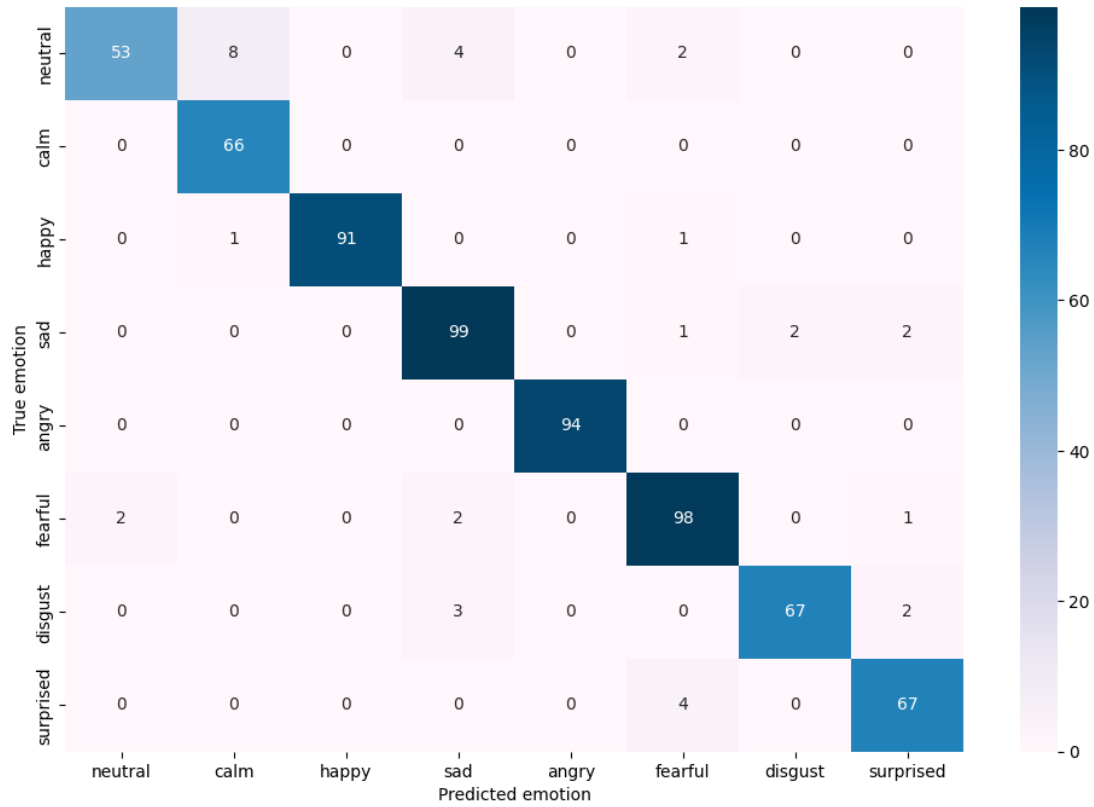
Loss graph for training and validation



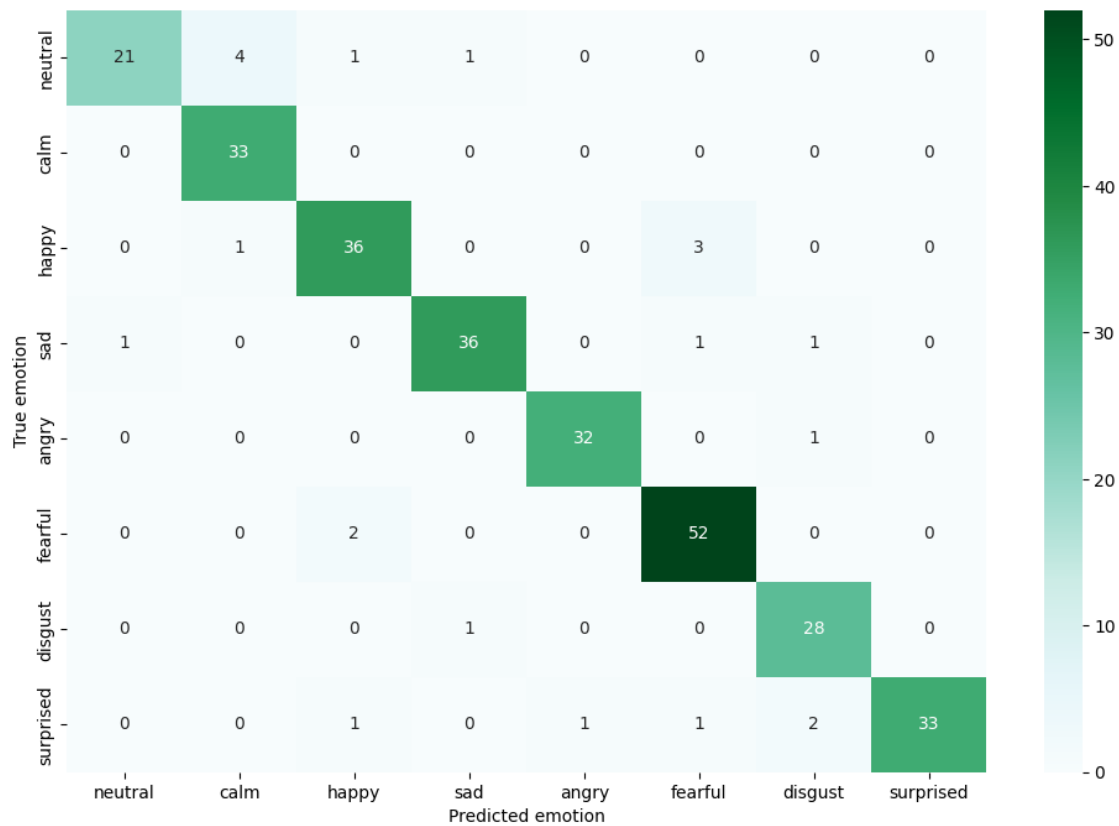
Model accuracy graph



Validation Confusion Matrix



Testing Confusion matrix



CONCLUSION

ACCURACY	LSTM MODEL	BIDIRECTIONAL LSTM MODEL	GRU MODEL
TRAINING	0.9798	0.9961	0.9936
VALIDATION	0.9522	0.9627	0.9478
TESTING	0.9078	0.9352	0.9249

PERFORMANCE COMPARISON

BIDIRECTIONAL LSTM > GRU > LSTM

RESOURCES AND RESEACH PAPERS:

1. NEURAL NETWORKS AND DEEP LEARNING - COURSERA
<https://www.coursera.org/learn/neural-networks-deep-learning?action=enroll>
2. IMPROVING DEEP NEURAL NETWORKS – COURSERA
<https://www.coursera.org/learn/deep-neural-network?specialization=deep-learning>
3. SEQUENCE MODELS – COURSERA
<https://www.coursera.org/learn/nlp-sequence-models?action=enroll>
4. NATURAL LANGUAGE PROCESSING – COURSERA
<https://www.coursera.org/specializations/natural-language-processing?specialization=deep-learning>
5. Efficient Estimation of Word Representations in Vector Space
<https://arxiv.org/pdf/1301.3781>
6. GloVe: Global Vectors for Word Representation
<https://nlp.stanford.edu/pubs/glove.pdf>
7. Attention is All You need
<https://arxiv.org/pdf/1706.03762>
8. Transformers: State-of-the-Art Natural Language Processing
<https://arxiv.org/pdf/1910.03771>
9. Language Models are Unsupervised Multitask Learners
https://d4mucfpksyww.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
10. BLEU: a Method for Automatic Evaluation of Machine Translation
<https://dl.acm.org/doi/pdf/10.3115/1073083.1073135>
11. NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE
<https://arxiv.org/pdf/1409.0473>
12. Connectionist Temporal Classification
https://www.cs.toronto.edu/~graves/icml_2006.pdf

THANK YOU