15/12/24

Page No.:

Date: / /

Roadmap for NLP

→ Spam Classification → Email → Spam / Not Spam

Email Subject	Email Body	Spam/Ham
"Billionaire"	"You won a lottery of Billions $$"	1 → NLP

↓

Meaningful vectors

eg :- ① Alexa
② Google Home

Roadmap :

Cleaning the I/p

~~I/p Text → Vectors~~ Step 1 :- Text Preprocessing - Tokenization, lemmatization,

Input Text → vectors Step 2 :- Text Preprocessing 2 - BOW, TF IDF, Unigram, Bigrams

I/p Text → vectors Step 3 :- Text Preprocessing 3 :- Word2vec, Avg word2vec

Neural Network Step 4 :- RNN, LSTM - RNN, GRU, RNN → Deep learning

(Keras) + (PyTorch)

Tensorflow (Google)

Step 5 :- Text Preprocessing → Word Embedding

~~TRANSFORMER~~

TRANSFORMER

Step 6

Step 7

BERT

Tokenization In NLP

Page No.:
Date: / /

Topics

① Corpus - Paragraph

② Documents - Sentences

③ Vocabulary - Unique words

④ Words -

Tokenization

corpus = " My name is Maniya Manzar and I know have interest in Deeplearning. I am also a Youtuber. "

⇓

Tokens (sentences)

Tokenization ① My name is ~~Krish~~ Maniya and I have interest in ML, NLP & DL

Words ← Tokens

② I am also a Youtuber

[I like to drink Apple Juice. My friend likes Mango Juice]

⇓ Tokenization

Tokens (sentences)

→ 11 words

① I like to drink Apple Juice
② My friend like Mango Juice

⇒ Unique words ⇒ 10 words

⇓ 9 words

Vocabulary

Tokenization :

→ Tokenization is the process of converting either a paragraph or a sentences into tokens.

if paragraph —tokenize→ sentences → each sentence is a token

if sentences —tokenize→ words → each word is a token

Tokenization Practicals

Page No.:
Date: / /

① → Installing nltk = There are two libraries → spacy & nltk

② → corpus = """ Hello Welcome, to Krish Naik's NLP tutorials. Please do watch the entire course! to become expert in NLP. """

③ from nltk.tokenize import sent_tokenize

④ sent_tokenize (corpus)
↳ it will make three tokens & print it
↳ type → list

⑤ from nltk.tokenize import word_tokenize

⑥ word_tokenize (corpus)
↳ it will form tokens based on words
it will also take '.' & '!' & '.' & ',' as tokens

⑦ from nltk.tokenize import wordpunct_tokenize

⑧ ~~word_punct~~ wordpunct_tokenize (corpus)
↳ it will also take inverted commas as token → ''

⑨ from nltk.tokenize import TreeBankWordTokenizer

⑩ tokenizer = TreeBankWordTokenizer()

⑪ tokenizer.tokenize (corpus)
↳ it will only not include first step last step full stop as a token, & rest similar to word_tokenize

Text Preprocessing Stemming using NLTK

Page No.:
Date: / /

Stemming :-

Stemming is the process of reducing a words to its words prefixes or affixes. ie the roots of words i/e lemma.

"""

If is imp in NLP & NLU (w -> understand)

Eg :- # Classification problem
Comments of products is a +ve review or -ve review.
Reviews ----> eating, eat, eaten [going, gone, goes] --> go.

PorterStemmer

① -> from nltk.stem import PorterStemmer

② -> stemming = PorterStemmer()

-> for words in words: (words → an list contain words defined previously; word)
print (word + "--->" + stemming.stem(word))

O/p -> eating -> eat
eats -> eat
eaten -> eaten
programming -> program
history -> histori
↳ error

Conclusion :- PorterStemmer work good for a good number of words, But sometimes it produces error

RegexpStemmer class

~~It basically takes a single expression~~

① uses Regular expression

-> from nltk.stem import RegexpStemmer

↳ reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)

reg_stemmer.stem('eating')
o/p -> eat

reg_stemmer.stem('ingeating')
o/p -> ingeat

Snowball Stemmer → performs better than PorterStemmer

-> from nltk.stem import SnowballStemmer
snowballstemmer = SnowballStemmer('english')
for word in words:
print(word + "--->" + snowballstemmer.stem(word))
O/p -> eating --> eat
eats --> eat
history --> histori

Lemmatization

Similar to Stemming. The output we get after lemmatization is called 'lemma', which is a root word rather than root stem, the output of stemming.

After lemmatization we will be getting a valid word that means the same thing. → It uses the class uses Morphy() func

Q/A & Chatbots use this

→ from nltk.stem import WordNetLemmatizer

→ lemmatizer = WordNetLemmatizer()

→ lemmatizer.lemmatize("going", pos="v")
o/p → go

'''

POS - Noun - n
Verb - v
adjective - a
adverb - r

'''

Stopwords

→ w.r.t different language different texts you can apply different stopwords

→ Stopwords are common words that are often removed from text before its analyzed because they don't usually carry much meaning on their own.

eg:- a, The, is, and

→ paragraph = ''' APJ Abdul Kalam Speech '''

→ from nltk.corpus import stopwords

→ import nltk
nltk.download("stopwords")

→ from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer()
sentences = nltk.sent_tokenize(paragraph)

Apply stopwords and filter and then Apply Stemming

for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words) # converting all the words into sentence

→ Now this new sentences list will have the consequential words. The

We could also use Lemmatization

→ from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [lemmatizer.lemmatize(word, pos='v') for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)

Parts of Speech Tagging

→ It is the process of assigning a part of speech (such as noun, verb, adjective etc)

→ POS Tagging helps to understand the grammatical structure of a sentence and the function of each word within it.

```
import nltk
from nltk.stem import
nltk.download('averaged_perceptron_tagger')

for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [word for word in words if word not
            in set(stopwords.words('english'))]
    pos_tag = nltk.pos_tag(words)
    print(pos_tag)
```

CC -

Tags	Description
CC	Coordinating conjunction
NN	Noun, singular or mass
NNS	Noun, plural
PRP	Personal pronoun

Named Entity Recognition

Person eg: Krish C Naik

Place or location : eg :- India

Date eg : September, 24-09-1989

Time eg: 4:30pm

Money eg : 1 million dollars

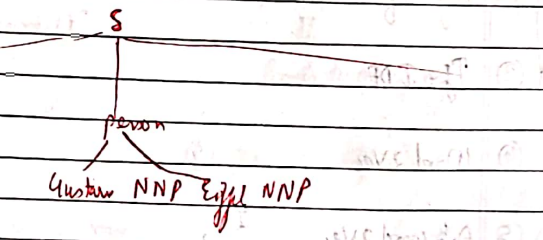
Organization eg :- iNeuron Private Limited

Code

sentence → is a story

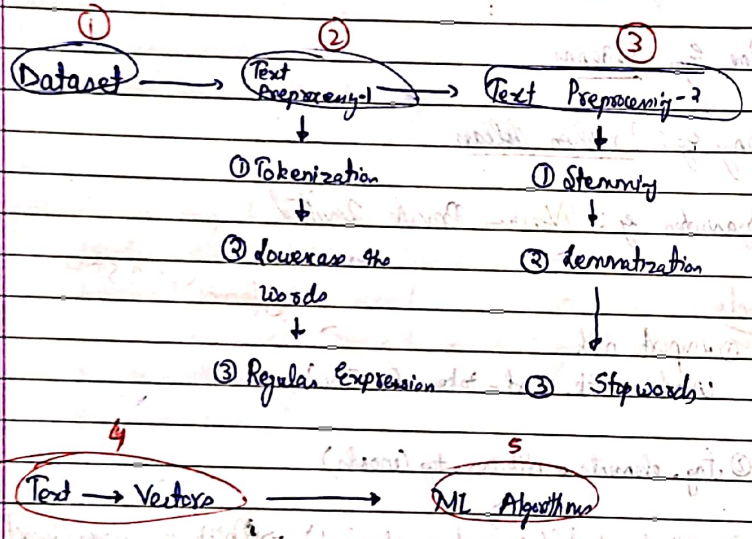
```
① import nltk
   words = nltk.word_tokenize(sentence)
② tag_elements = nltk.pos_tag(words)
③ nltk.download('maxent_ne_chunker') → ④ nltk.download('words')
⑤ nltk.ne_chunk(tag_elements).draw
```

o/p →



What we have learnt ?

Text	o/p
① The food is good	1
The food is bad	0



① One Hot Encoded
② Bag of Words (BOW)
③ TF-IDF
④ Word 2 Vec
⑤ Avg word 2 Vec

One Hot Encoding

V=7 Vocabulary (unique words)

	Text	o/p
	1 2 3 4	
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is Amazing	1

The food is good bad Pizza Amazing

D2: 1 0 0 0 0 0 0
D3: 0 1 0 0 0 0 0

D1 [[1 0 0 0 0 0 0], D2 → [[1 0 0 0 0 0 0], D3 → [[0 0 0 0 0 0 1],
[0 1 0 0 0 0 0], [0 1 0 0 0 0 0], [0 0 1 0 0 0 0],
[0 0 1 0 0 0 0], 4x7 [0 0 1 0 0 0 0], 3x7 [0 0 0 0 0 0 1]
[0 0 0 1 0 0 0]) [0 0 0 0 1 0 0]))

4x7 ↓

No fixed toy size

each time when it runs it will check all the unique words, i.e 7 so in this way it runs 4 times & everytime it will check all the words

Advantages

① Easy to implement with python [Sklearn, OHE, pd.get_dummies()]

Disadvantages

① Sparse matrix (lots of zeros) ↓ leads to overfitting

② for ml Algorithms → We need fix sized Input

③ No semantic (similar) meanings getting captured

I am not able to understand what the most important word, how this word is relative to another.

Out of Vocab (OOV) → # So this will not work if the word is not in Vocabulary
Out of Vocabulary (OOV)

② Bags of Words

Text	O/p
He is a good boy	1
She is a good girl	1
Boy & girl are good	1

Lower all the words
Cap ⇒
& Stopword

S1 → good boy

S2 → good girl

S3 → Boy girl good

Vocabulary	frequency
good	3
boy	2
girl	2

	good	boy	girl	O/p
S1	1	1	0	→ 1
S2	1	0	1	→ 1
S3	1	1	1	→ 1

Now, there are two kinds of BOW

Binary BOW
{1 and 0}

BOW
{Count will get updated based on frequency}

Advantages

1. Simple & intuitive
2. Fixed size input → Helps in ML algorithm training

Disadvantages

① Sparse matrix or array leads to Overfitting

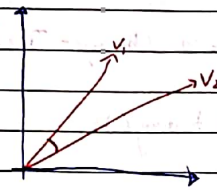
② Order of word is getting changed

③ Out of Vocabulary is still a issue

④ Semantic meaning is still not getting captured
(which is the most imp word)

→ e.g.: ① The food is good → [1 1 1 0 1] → v_1

② The food is not good → [1 1 1 1 1] → v_2



Spam Classifier Practical

```
import pandas as pd
messages = pd.read_csv('smsclean.csv', sep='\t', names=["label", "message"])
```

Data Cleaning & Preprocessing

```
import re
import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

corpus = []
for i in range(0, len(messages)):
    review = re.sub('[^a-zA-Z]', ' ', messages['message'][i])
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not
              word in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)
```

Create the Bag of Words model

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 100, binary = True)

X = cv.fit_transform(corpus).toarray()
```

N-grams (BOW)

eg: bigrams, trigrams

	food	not	good
S1 → The food is not good.	1	0	1
S2 → The food is not good.	1	1	1

Bigram

	[food not	not good]	food good	food not	not good]
S1 →	1	0	1	0	0
S2 →	1	1	0	1	1

sklearn → n-grams = (1,1) → Unigrams → only one word per vocabulary
= (1,2) → unigram, bigram
⇒ (1,3) → unigram, bigram, trigram
⇒ (2,3) → Bigram, trigram

Create the Bag of words model with ngram

```
from sklearn.feature_extraction.text import CountVectorizer

# for Binary BOW enable binary=True

cv = CountVectorizer(max_features = 100, binary = True, ngram_range = (2,3))

X = cv.fit_transform(corpus).toarray()

cv
```

Words ⟶ Vectors

4) TF - IDF [Term Frequency - Inverse Document Frequency]

S1 → good boy
S2 → good girl
S3 → boy, girl, good

Term Freq (TF) = $\frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$

$$IDF = \log_e\left(\frac{\text{No. of sentences}}{\text{No. of sentences containing}}\right)$$

Term Frequency

	S1	S2	S3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

IDF

Words	IDF
good	$\log_e(3/3) = 0$
boy	$\log_e(3/2)$
girl	$\log_e(3/2)$

Final ÷ TF - IDF

	good	boy	girl
sent.1	0	$1/2 \times \log_e(3/2)$	0
sent.2	0	0	$\frac{1}{2}\log_e(3/2)$
sent.3	0	$\frac{1}{2}\log_e(\frac{3}{2})$	$\frac{1}{3}\log_e(3/2)$

Advantages

1) Intuitive
2) Fixed Size → Vocab size
3) Word importance is getting captured

Disadvantages

1) Sparsity still exists
2) OOV

TF-IDF Practicals

```
from nltk.stem import WordNetLemmatizer
lemmatize = WordNetLemmatizer()

corpus = []
for i in range(0, len(messages)):
    review = re.sub('[^a-zA-Z]', ' ', messages['message'][i])
    review = review.lower()
    review = review.split()
    review = [word for word in review if not
              word in stopwords.words('english')]
    review = ' '.join(review)
    corpus.append(review)
```

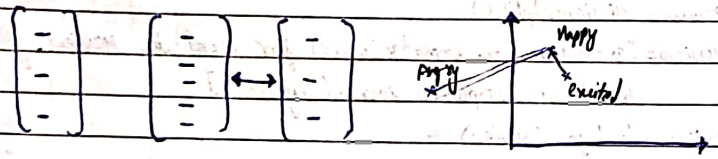
Create TF-IDF and NGrams

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=100)  → tfidf = TfidfVectorizer(max_features=100,
                                                ngram_range=(2,3))
X = tfidf.fit_transform(corpus).toarray()

import numpy as np
np.set_printoptions(edgeitems=30, linewidth=100000, formatter=dict(float=lambda
                    x: "%.3g" % x))

tfidf.vocabulary_
```

Word Embedding

- In nlp word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that words that are closer in the vector space are expected to be similar in many.

→ Angry Happy excited



Word Embedding

- Count or frequency
  1. OHE
  2. BOW
  3. TF-IDF
- Deep learning Trained model
  - Word2 Vec
    - CBOW [Continuous Bag of WORDS]
    - Skipgram

Word2Vec → Feature Representation

It uses a neural network model to learn word associations from a large corpus of text.

Once trained, such a model can ~~predict~~ detect ~~sim~~ synonymous words or suggest additional words for a partial sentence.

WORD2VEC

each distinct word → (convert) list of numbers called vectors

Vocabulary → Unique Words in → Corpus (paragraph)

	Boy	Girl	King	Queen	Apple	Mango
Gender	-1	1	-0.92	+0.93	0.01	0.05
Royal	0.01	0.02	0.95	0.96	-0.02	0.07
Age	0.03	0.02	0.75	0.68	0.95	0.96
Food	-	-	-	-	0.91	0.93
⋮	-	-	-	-	-	-
	-	-	-	-	-	-
nth	-	-	-	-	-	-

Feature Representation

300 dimension

With this we can get conclusion like King - Boy + Queen = Girl

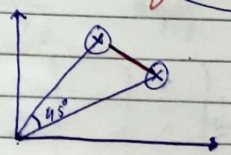
Page No.:
Date: / /

g →: King [0.95, 0.96] Man [0.95, 0.98]

Queen [-0.96, 0.95] Women [-0.94, -0.96]

King - Man + Queen = Women

Cosine Similarity



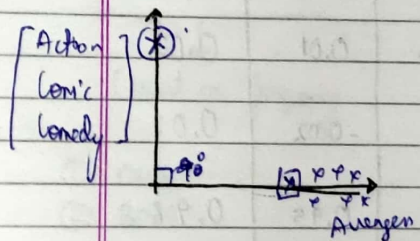
Distance = 1 - Cosine Similarity

Cosine similarity = $\cos\theta$

for g. $\theta = 45°$, $\cos 45° = 0.707$

Distance = 1 - 0.707

⇒ 0.29



Distance = 1 - 0 = 1

if Distance is closer to zero, this means words are similar

if Distance is equal to one → There is no similarity

Page No.:
Date: / /

① Word 2Vec → CBOW [Continuous Bag of Words]

Word 2Vec → Skipgram

Pretrained Model

Train A model for Scratch from

① CBOW →

Corpus is our Dataset

[iNeuron Company Is Related To DATA SCIENCE] → Vocabulary

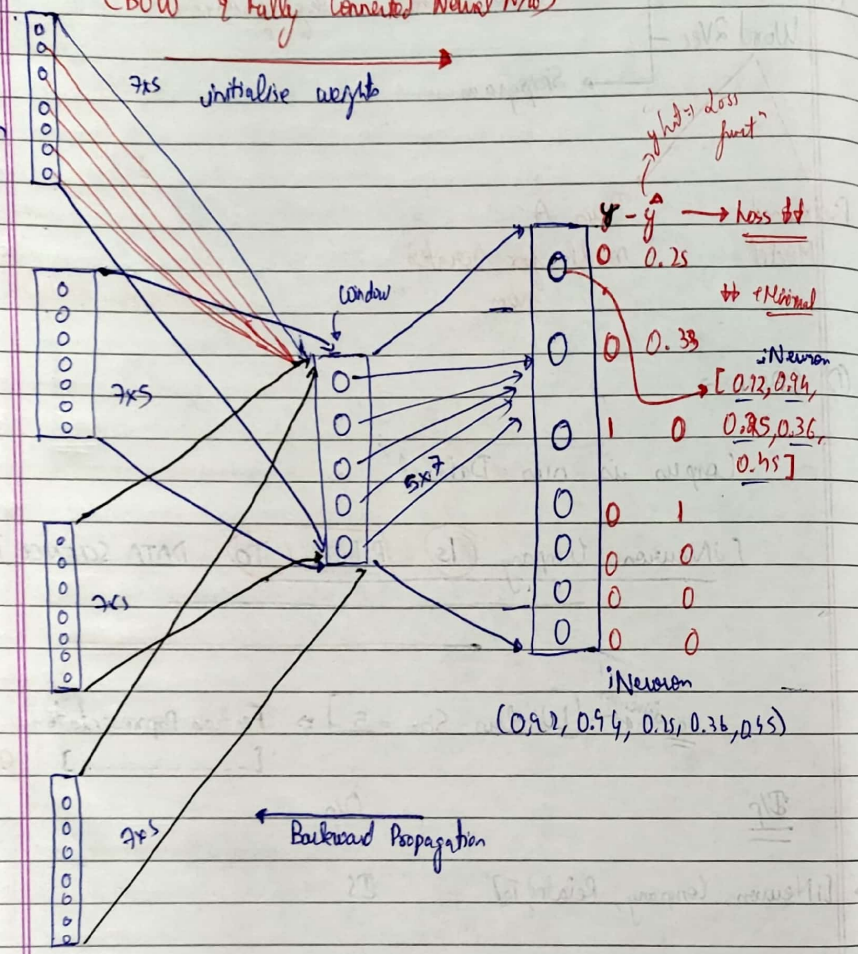
300 dimension ← Window Size = 5 ⇒ Feature Representation

[. . . . .] OHE

I/P	O/P
→ [iNeuron, Company, Related, To]	Is
→ Company, Is, To, DATA	Related
→ Is, Related, DATA, SCIENCE	To

OHE

iNeuron [1 0 0 0 0 0 0]
Company [0 1 0 0 0 0 0]
Related [0 0 1 0 0 0 0]
To [0 0 0 1 0 0 0]



Skip Gram → opp of CBOW

Dataset: iNeuron Company is related to Data Science

Window size = 5

O/p	I/p
→ [iNeuron, Company, Related, To]	Is
→ [Company, Is, To, Data]	Related
→ [Is, Related, Data Science]	To

↓↓ → here Forward propagation happens

Q) When should we apply CBOW or Skipgram

small Dataset = CBOW
Huge Dataset → Skipgram

Q) How to improve CBOW or Skipgram

1) Increase the Training Data
2) Increase the Window size & vector Dimension as also increase as a consequence.

eg: Google Word2Vec

3 Billion words → Google News

300 columns hogi ek bee word ke liye.

[features representation of 300 dimension vectors]
cricket → [. . . . .]

Advantages of Word2Vec

*) Sparse Matrix ⟶ Dense Matrix

*) Semantic Info is getting captured [Honest, good]

*) Vocabulary size ⟶ Fixed set of dimension
↳ Google Word2vec [300 dimension]

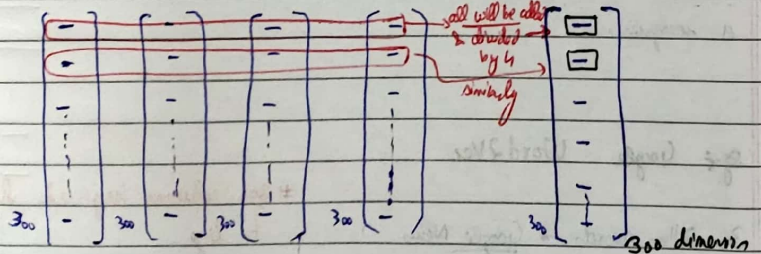
*) OOV is also solved.

305 video → Avg Word2Vec

	Text	O/p	300 Avg Word2vec	O/p
D1 ⟶	The food is good.	1	[_ _ _ _]	1
D2 ⟶	The food is bad.	0	[_ _ _ _]	
D3 ⟶	Pizza is Amazing.	1	[_ _ _ _]	

Google pretrained Word2Vec Model

[The food is good] → D1 → Avg Word2Vec



* So, first of all Now after this for every sentence this Avg Word2Vec will be stored & based on it we will get the output.

Word2Vec Practical Implementation Gensim

1) !pip install gensim
import gensim

2) from gensim.models import Word2Vec, KeyedVectors

3) import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
vec_king = wv['King']
↳ 1.6gB file downloaded

4) ~~wv['cricket']~~ wv.most_similar('cricket') → 53% similarity
↳ op: cricketer, 0.83
cricketers, 0.81
Test_cricket, 0.80
Cricket, 0.75

5) wv.most_similar('happy')
↳ op → 'glad', 0.740
'pleased', 0.66
'ecstatic', 0.662

6) wv.similarity('hockey', "sports')
↳ 0.535

7) vec = wv['King'] - wv['man'] + wv['woman']

8) wv.most_similar([vec]) → 84% similarity
↳ king, 0.84
queen, 0.73

Imp

Best Practise for ML Algorithm

① Preprocessing and Cleaning
② BOW And TFI-IDF (Sentences --> Vectors)
③ Train Test Split
④ Trained Our Models

✗ Wrong method

(1) Preprocessing and Cleaning
(2) Train test split
(3) BOW & TF-IDF (Sentences ---> vectors) { Prevently Data leakage }
(4) Trained Our Models ✓

Spam Ham Classification Project Using BOW

1) import pandas as pd
messages = pd.read_csv('smss pamcollection', sep='\t', names=['label', 'message'])

2) ## Data Cleaning & Preprocessing

2 -> import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

3) corpus = []
for i in range(0, len(messages)):
review = re.sub('[^a-zA-z]', ' ', messages['message'][i])
review = re.lower()
review = re.split()
review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
review = ' '.join(review)
corpus.append(review)

3) ## Create Bag of Words

# Output Feature
y = pd.get_dummies(messages['label'])
y = y.iloc[:, 0].values

# Train Test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(corpus, y, test_size=0.20)
(corpus = X)

# Create the Bag of Words model
from sklearn.feature_extraction.text import CountVectorizer

for Binary BOW enable Binary = true
cv = CountVectorizer(max_features=2500, ngram_range=(1,2))

Independent features
X_train = cv.fit_transform(X_train).toarray()
X_test = cv.transform(X_test).toarray()

⑤ ML model

→ from sklearn.naive_bayes import MultinomialNB
spam_detect_model = MultinomialNB().fit(X_train, y_train)
y_pred = spam_detect_model.predict(X_test)

⑤ Performance Metrics

from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_test, y_pred)
print('classification_report(y_test, y_pred))

⑥ Similarly the TF-IDF Model will be implemented

→ ~~import gensim~~
~~gensim.models.Word2Vec()~~ = ψ

→ AvgWord2Vec → It ~~does~~ gives you the entire sentence in vector form

Spam Ham Projects Using Word2Vec, AvgWord2Vec

Simple_preprocess() → Convert a document into a list of lowercase tokens, ignoring tokens that are too short or too long.

from gensim.utils import simple_preprocess

words = []
for sent in corpus:
    sent_token = sent_tokenize(sent)
    for sent in sent_token:
        words.append(simple_preprocess(sent))

words ~~allist~~ → ['go', 'until', 'jurong', 'point'], ['ok', 'bye',]]

lets train Word2Vec from scratch

import gensim
model = gensim.models.Word2Vec(words)
# To get all the Vocabulary
model.wv.index_to_key

model.corpus_count → 5569

model.epochs → 5 epochs ∝ better model value

Avg - word 2 Vec
Let's make

```
def avg_word2vec(doc):
    # remove out-of-vocabulary words
    # sent = [word for word in doc if word in
    #         model.wv.index_to_key]
    # print(sent)
    return np.mean([model.wv[word] for word in doc if
                    word in model.wv.index_to_key])
```

→ so, our previous code only made a single sentence into vectors, to convert entire sentences into vector (corpus)

we will, from tqdm import tqdm

```
# apply for the entire sentences
import numpy as np
X = []
for i in tqdm(range(len(words))):
    X.append(avg_word2vec(words[i]))
```

```
# making independent features
X_new = np.array(X)
```

X_new.shape → 5569 , X_new[0].shape → (100,)

```
# Dependent Features, # Output Feature
y = messages['label'].map(lambda x: len(x)>0, corpus))
y = pd.get_dummies(y['label'])
y = y.iloc[:, 0].values
```

→ y.shape → (5569,)

→ X[0].reshape(1,-1).shape → (1, 100)

```
# This is the final independent features
df = pd.DataFrame()
for i in range(0, len(X)):
    df = df.append(pd.DataFrame(X[i].reshape(1,-1)), ignore_index=True)
```

df.head() — Their 3 dimensions

	0	1	2
(Apple Banana) 0	0.35	0.25	0.65
(Apple Chalu)	0.45	0.75	0.75

let's say apple → (0.3, 0.2, 0.7)
Banana → (0.4, 0.3, 0.6)

Applying Avg Word2Vec
(0.3+0.4)/2 → 0.35
0.2+0.3/2 → 0.25
0.7+0.6/2 → 0.65

Now for sentence "apple Banana"
vec → [0.35, 0.25, 0.65]

Continue Code

```
→ df.dropna(inplace=True)
→ X.isnull().sum()
→ y = df['Output']
# Independent feature
X = df
# Train Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
# Model Training
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

Page No.:
Date: / /

```
from sklearn.metrics import accuracy_score, classification_report
print(accuracy_score(y_test, y_pred))
```

NLP → Amazon Kindle Reviews Sentiment Analysis

① Train Test Data preprocessing	② Train Test split	③ Bert → Vec	④ ML Algorithm train
① Checking Null values, ② Checking unique values ③ encoding the output to 0 & 1 (if its a classification problem) ④ Lower all the text ⑤ Removing special characters, stopwords ⑥ Applying lemmatization	simple train test split pass your X & y accordingly to your independent & dependent feature	Apply BOW, TFIDF, Word2Vec, ↓ X_train_wv = fit_transform(X_train) X_test_wv = transform(X_test) ↓	

Page No.:
Date / /

Deep Learning

→ part of ML used to mimic human brain
→ called "deep" because it uses multiple layers of neural networks to help the system understand & interpret data.

Deep Learning

① ANN → Artificial Neural N/w → Classification, Regression

② CNN → Convolutional Neural N/w → I/p : Images Video frames → RCNN, Masked RCNN, Detection, YOLO V5 V6 V7

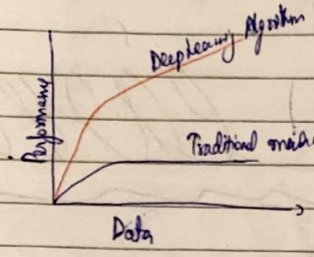
③ RNN → Recurrent Neural N/w → NLP → Time series, NLP

I/p → Text, Time series

{ Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT }

Framework
↳ Tensorflow, Pytorch

Why Deep Learning is becoming Popular?

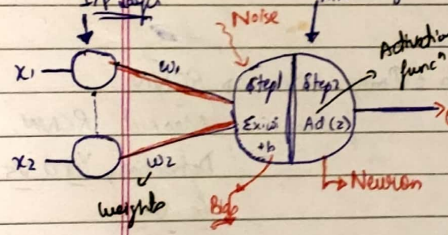


Perceptron (Artificial Neuron or Neural Network Unit)

used in solving Binary Classification Dataset

	IQ	No. of study hours	o/p Pass/Fail
① Input layer ✓			
② Hidden layer ✓			
③ Weights ✓	→ 95	3	0
④ Activation function ✓	→ 110	4	1

~~Dataset~~



Loss function → predicted o/p → actual o/p

↳ 1 - 0 = 1 → error

↳ This error func^n gets back to our weight & new weights are initialized. This process is done until your error is zero.

$Z = w_1x_1 + w_2x_2 + b$ (bias)

$$Z = \sum_{i=1}^{n} x_i w_i + b$$

Bias → It makes your data input data flexible

Activation func^n

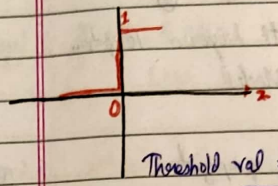
Transform the output b/w

① 0 to 1

② -1 to 1

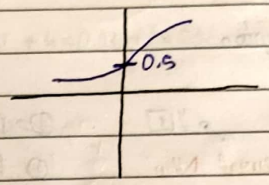
Two types of Activation func^n

Step function

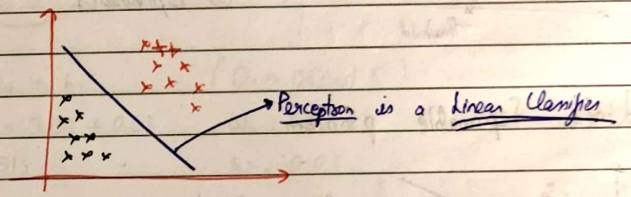


$$\begin{cases} 0 & z \le 0 \\ 1 & z > 0 \end{cases}$$

Sigmoid function



$$\begin{cases} 1 & z > 0.5 \\ 0 & z \le 0.5 \end{cases}$$



Step1

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$z = b + w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_n x_n$$

Linear problem statement

$$y = mx + c$$

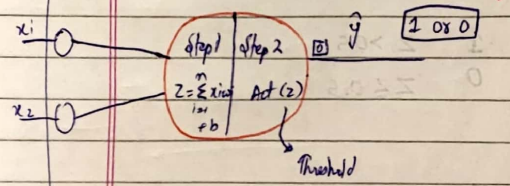
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1 + \beta_3 x_1 \dots \beta_n x_n$$

Perceptron Models [ANN]

Single Layer Perceptron Model

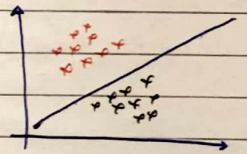
Multi Layered Perceptron Model

Feed Forward Neural N/w



① Forward propagation
② Backward propagation
③ Loss function
④ Activation functⁿ
⑤ Optimizers

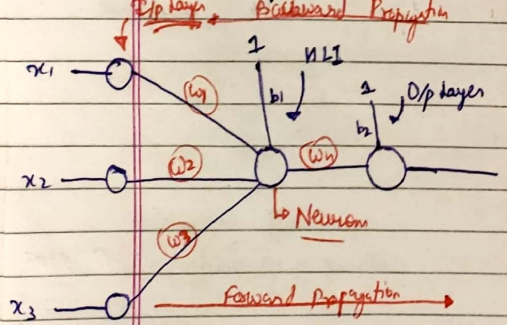
Linear Separable problem



Binary classifier
⇓
Linearly Separable

→ The single layer perceptron is only good for linearly separable dataset

MultiLayered Perceptron Model [ANN]



x_1 IQ	x_2 study hrs	x_3 Play hrs	Pass/fail o/p
95	4	4	1
100	5	2	1
95	2	7	0

Loss funct^n ⇒ (1 - 0.51129)
≈ 0.49 ↓↓ error

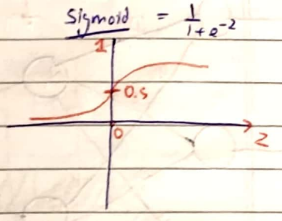
Hidden layer 1

Step1 ÷ $Z = 95 \times 0.01 + 4 \times 0.02 + 4 \times 0.03 + 1 \times 0.01$
$= 1.151$

Step2 ÷ Activation(z)

$$f(z) = \frac{1}{1 + e^{-1.151}} \Rightarrow 0.759$$

Sigmoid $= \frac{1}{1+e^{-z}}$



Hidden layer 2

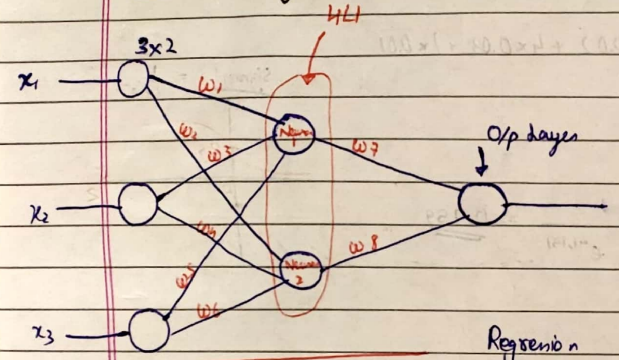
Step1 ÷ $Z = O_1 + w_4 + b_2$ (O_1 ⇒ Output 1)
$= 0.759 \times 0.02 + 0.03$ $w_4 = 0.02$
$= 0.04518$ $b_2 = 0.03$

Step2 ÷ Activation (z) $\frac{1}{1 + e^{-(0.04518)}} \Rightarrow 0.51129$

$O_2 = 0.51129 \Rightarrow \hat{y}$

Note ÷ After this we will take this value & calculate our loss funcⁿ & then our main priority will be to minimize the loss error. first we will update the value of w_4 ~~and then we will recalculate~~ if loss funct^n not minimised then we will go further back & update values of w_1, w_2, w_3

Back Propagation And Weight Updation Formula



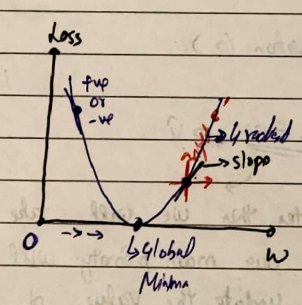
Regression
1. MSE
2. MAE
3. Huber loss

Classification
1. Binary Cross Entropy
2. Categorical Cross Entropy

Weight Updation Formula

$$W_{7new} = W_{7old} - \eta \left[\frac{dL}{dW_{7old}}\right]$$ (slope → loss)

$$W_{8new} = W_{8old} - \eta \frac{dL}{dW_{8old}}$$ (learning rate)



$$W_{new} = W_{old} - \eta \frac{dh}{dW_{old}} \Rightarrow \text{Weight Updation formula}$$

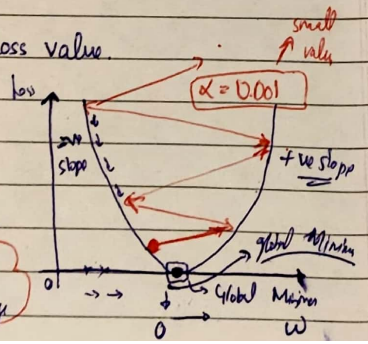
Optimizers: To reduce the loss value.

$$W_{New} = W_{old} - \eta(-ve) = W_{old} + \eta(+ve)$$

learning rate

$W_{new} >> W_{old}$

η = large value ⇓ It may never converge



$\frac{d(x)^2}{dt}$: y ; Chain rule $\Rightarrow \frac{d(x)}{dt} = \frac{dx}{dy} \times \frac{dy}{dt}$

$W_{new} = W_{old} - \eta(+ve)$

$= W_{old} - \eta(+ve)$

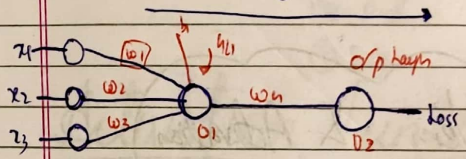
$W_{new} << W_{old}$

When We reaches global Minima

$W_{new} = W_{old}$

Chain Rule of Derivative

- In Backward propagation, when you go backwards.



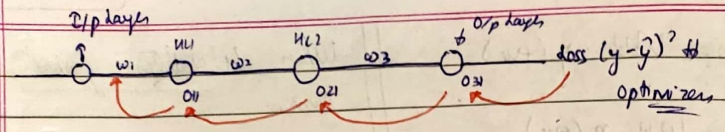
$$W_{new} = W_{old} - \eta \frac{dh}{dW_{old}}$$

$$W_{4new} = W_{4old} - \eta \frac{dh}{dW_{4old}}$$

$$\frac{dh}{dW_{4old}} = \frac{dh}{dO_2} * \frac{dO_2}{dW_{4old}}$$ ⇒ Chain Rule of Derivation

$$W_{1new} = \frac{dh}{dW_{1old}} = \frac{dL}{dO_2} * \frac{dO_2}{dO_1} \times \frac{dO_1}{dW_{1old}}$$

Front propagation



$\sigma(z) \Rightarrow 0$ to 1
↳ sigmoid

The weight updat^n is dependent on $\frac{dL}{dw_{old}}$ & this is depended on sigmoid activation func^n

$\rightarrow W_{inew} = W_{i\,old} - \eta \frac{dL}{dw_{i\,old}}$ (0.01) ⇒ small value

Vanishing Gradient Problem & Activation funct^n

0-0.25 (0-0.25) (0-0.25) ⇒ smaller value

$$\frac{dL}{dw_{i\,old}} = \frac{dL}{dO_{31}} * \frac{dO_{31}}{dO_{21}} * \frac{dO_{21}}{dO_{11}} * \frac{dO_{11}}{dw_{i1}}$$

$$O_{31} = \sigma(w_3 * O_{21} + b_3) \qquad z = w_3 * O_{21} + b_3$$

$$O_{31} = \sigma(z)$$

$$\frac{dO_{31}}{dO_{21}} = \frac{d(\sigma(z))}{d(z)} * \frac{dz}{dO_{21}} \quad \{\text{Chain Rule}\}$$

$0 \le \sigma(z) \le 0.25$ ⇒ Derivative of Sigmoid

$$* \frac{\sigma((w_3 * O_{21}) + b_3)}{\sigma(O_{21})}$$

$0 - \sigma\ 0.25 * w_3\,old$

⇒ Now, $W_{inew} \approx W_{i\,old}$

((small val^n $-\eta \frac{dL}{dw_{i\,old}}$ → small value (see prev page)

⇒ It will not matter

∴ There is no change in gradient descent, & the value of loss function is stagnant

∴ To fix this problem, Researchers started exploring other Activation funct^n

① Tanh
② ReLu
③ Prelu
④ Swish

Activation Functions

① Sigmoid Activation function : [0 to 1]

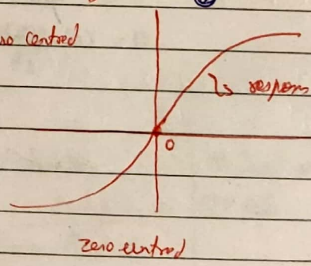
$\sigma(z) = \frac{1}{1+e^{-z}}$

Advantages

① Suitable for Binary Classification
② Clear prediction i.e very close to 1 or 0

Disadvantages

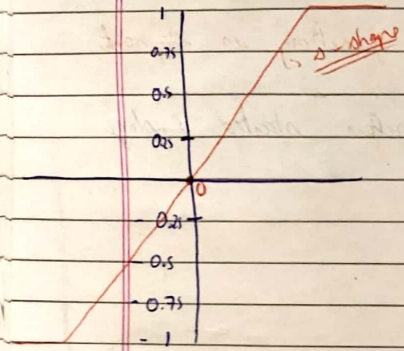
③ mathematical operat^n is relatively Time consuming
① Prone to vanishing Gradient Problem.
② Function output is not Zero Centered (mean ≠ zero)



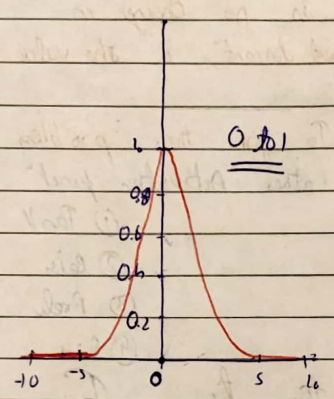
Activation

② Tanh / function

Tanh(x)



d Tanh(x) /dx



0 to 1

Range from -1 to 1
derivative range from 0 to 1

Advantages

① Zero Centric ✓
↳ weight updation is effied

Disadvantages

① Prone to Vanishing Gradient Problem
② Time complexity (mathematical func^n is relatively time consuming)

③ Relu

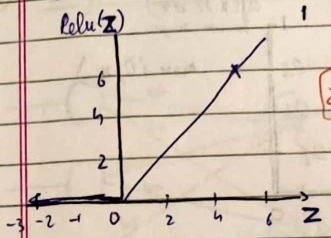
Sigmoid : 0 to 0

Page No.:
Date: / /

→ Rectified Linear Unit

③ Relu Activation Funct^n

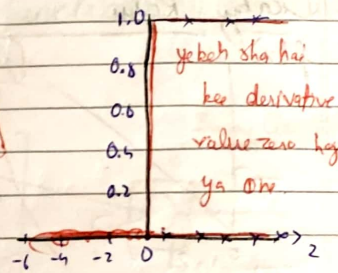
Relu(z)



$z = (O_2 * w_3) + b_3$

ReLU(z)

d (Relu(z)) /dz



yeh sha hai
ke derivative ke
value zero hogi
ya One.

Range from 0 to $+\infty$ (x) → if x is negative, O/p = 0, if x is +ve, whatever is the value of x ya usi ka O/p ⇒ as value of x

Derivatives. either 0 or 1.

If Derivative of Relu output is (1) ⇒ weight updation will happen

If → $w_{2\,new} = w_{2\,old} - \eta \left[\frac{dh}{d w_{2\,old}}\right] \Rightarrow 0$

⇓

If derivative of Relu(z) is 0 → $W_{2\,new} \approx W_{2\,old}$ ⇒ Dead Neuron (Aneuron which do not function as

If z = +ve ⟶ $\frac{d\,Relu(z)}{dz} = 1$

If z = -ve ⟶ $\frac{d\,Relu(z)}{dz} = 0$

Advantages

① Solving Vanishing Gradient Problem
② Max(0, x) → Calculation is superfast. The Relu function has a linear relationship
③ It is much faster than sigmoid or Tanh.

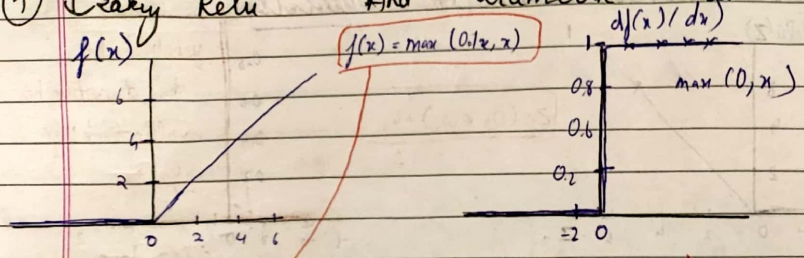
Tanh, Sigmoid are nonlinear

Disadvantages

① Dead Neuron
② Relu funct^n O/p
(0, x) ⇒ 0 or +ve number
⇓
It is not zero centric.
(as mean is not zero)

Page No.:
Date: / /

④ Leaky Relu And Parametric Relu



(In leaky Relu, $\alpha = 0.01$)

(In parametric Relu ⇒ we hyperparameter, the α value)

max (αx, x)

↳ hyperparameter, α = alpha = 0.01, 0.02, 0.03 }

Relu → Dead Neuron → Dead Relu Problem

Advantages

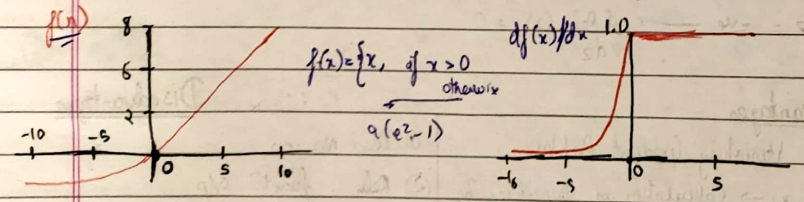
① Leaky ~~Relus~~ Relu has all the advantage of Relu

② It removes the Dead Relu Problem (zero value ~~ni~~ aayegi, ~~yhn~~ very small value aayegi zero multiply)

Disadvantage

① It is not zero centric

⑤ ELU (Exponential Linear Units)



Advantages

① No Dead Relu Issues

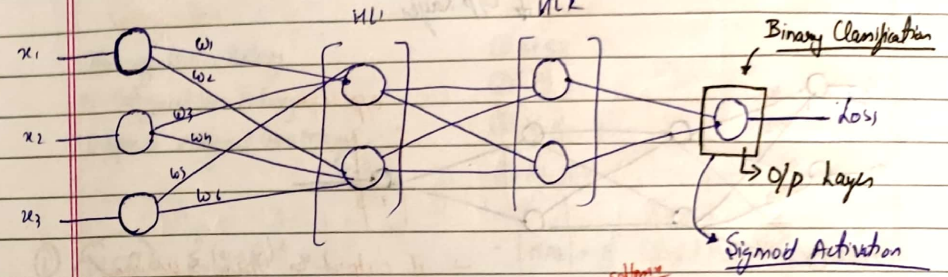
② Zero Centered

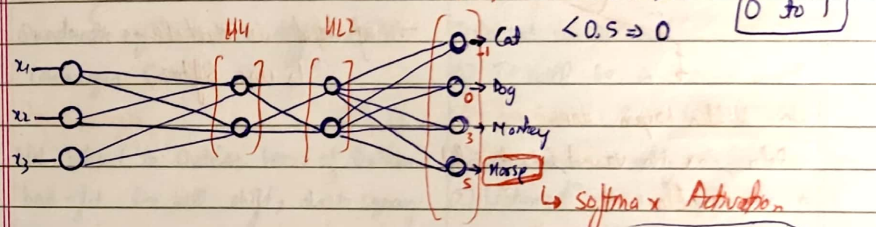
Disadvantages

① Slightly more computationally Intensive.

Page No.:
Date: / /

⑥ Softmax Activation function [Multiclass Classification Problem]





$$\text{Softmax} = \frac{e^{y_i}}{\sum_{K=0}^{n} e^{y_K}}$$

$y_i = O * \omega + b$ (predicted output)

Softmax

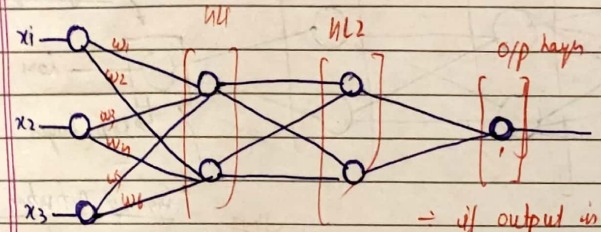
- Cat = $\frac{e^{1}}{e^{-1}+0+3.5}$ = 0.0003 3 ↳ predicted output / Transformed output
- Dog = $\frac{e^{0}}{e^{-1+0+3+5}}$ = 0.0024
- Monkey = $\frac{e^{3}}{e^{-1+0+3+5}}$ = 0.0183
- Horse = $\frac{e^{5}}{e^{-1+0+3+5}}$ = 0.1353

Pr (Horse) = 0.1353

0.00033 + 0.0024 + 0.0183 + 0.1353 ≈ 86%

⑦ Which Activation Function To Use When?

O/p layer

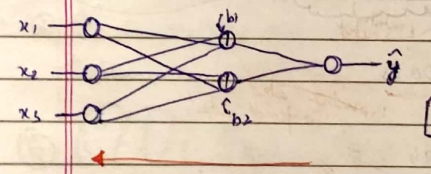


→ if output is Binary => use Sigmoid

→ if output is Multiclass => use Softmax

in Hidden layers use
Relu, or its variant (Leaky Relu)
Elu

Difference b/w Loss function And Cost function



Cost fn = $\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)$ ← Error

Loss = $(y-\hat{y})^2$ (This is MSE)

optimizers
Gradient Descent

	x_1	x_2	x_3	O/p
→	–	–	–	0
→	–	–	–	1
→	–	–	–	0

Loss function
→ MSE = $(y-\hat{y})^2$

Cost function
$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$

* It calculate for one data point at a time

It calculate the error for all data points in a single iteration

ANN

Classification	Regression
① Binary Cross Entropy	① MSE
② Categorical Cross Entropy	② MAE
③ Sparse Categorical Cross Entropy	③ Huber Loss
	④ RMSE

① MSE: $\frac{\sum(y-\hat{y})^2}{n} \times \frac{1}{n}$

→ Advantages
① Quadratic eq^n → so it is differentiable
② Converges Faster

Disadvantages
① Not robust to Outliers, bcoz of Outliers best-fit line will shift, due to square

② MAE: $MAE = \sum_{i=1}^{n}\frac{(y-\hat{y})}{n} \times \frac{1}{n}$

Advantages
① Robust to Outliers
② It will be in same unit

Disadvantages
① Convergence usually takes more time
② Optimization is a complex task

③ Huber loss :

① combines the advantage of MSE & MAE provides balance b/w the two by being quadratic for smaller errors & linear for larger errors

Cost fn → MSE: $\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2$ if $|y-\hat{y}| \le \delta$ ⇒ Hyperparameter (type of Threshold)

MAE: $\delta|y-\hat{y}| - \frac{1}{2}\delta^2$, otherwise

(If b ⇒ $|y-\hat{y}| <= \delta$, means no outliers is present.)

④ RMSE

Cost fn = $\sqrt{\frac{1}{N}\times\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}$

Advantages
① Differentiable

Disadvantages
① Not robust to Outliers

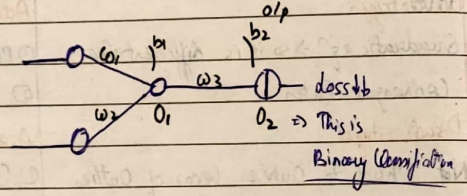
Classification → Cross Entropy
- → Binary [2 Binary]
- → Categorical [Multiclass]
- → Sparse Categorical [Multiclass]

① Binary Cross Entropy (log loss)

$Loss = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$

y = Actual value
$\hat{y}$ = Predicted value

$$Loss = \begin{cases} -\log(1-\hat{y}) & \text{if } y=0 \\ -\log(\hat{y}) & \text{if } y=1 \end{cases}$$



we will → Sigmoid activation

$\hat{y} = O_2 \quad [O_2 = \sigma(O_1 * \omega_3 + b_2)]$

Reuse again !! (input)

$\hat{y} = \sigma(z)$

$\sigma = \frac{1}{1+e^{-z}}$

② Categorical Cross Entropy (Multiclass Classificat^n)

f_1	f_2	f_3	o/p	(good $j=1$)	(Bad $j=2$)	(Neutral $j=3$)
2	3	4	Good	[1 (y_{11})	0 (y_{12})	0 (y_{13})]
5	6	7	Bad	[0 (y_{21})	1 (y_{22})	0 (y_{23})]
8	9	10	Neutral	[0 (y_{31})	0 (y_{32})	1 (y_{33})]

OVE ⇒ One Hot Encd

C = No of Categories

$$Loss(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y}_{ij}) \qquad i = 1 \text{ to } n$$

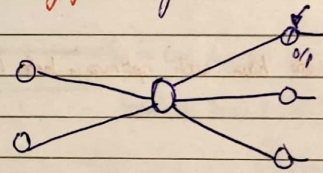
Actual value ⇒ $y_{ij} = [y_{11}, y_{12}, y_{13} \dots y_{1C}]$ C = No of categories
$[y_{21}, y_{22}, y_{23} \dots y_{2C}]$

Actual value → $y_{ij} = [y_{11}\ y_{12}\ y_{13} \dots y_{1C}]$ (C = No. of categories)

$[y_{21}, y_{22}, y_{23} \dots y_{2C}]$

$$y_{ij} = \begin{cases} 1 & \text{if the element is in the class} \\ 0 & \text{otherwise} \end{cases}$$

Prediction ⇐ $\hat{y}_{ij}$ ⇒ Softmax Activation = $Sof(z) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$



o/p ⇒ The output will be in the form of Probabilities

↳ o/p = $\hat{y}_{ij}$ = Probabilities [0.1, 0.2, 0.3, 0.2, 0.2] ∈ 1

Categorical ⇒ [0.2, 0.3, 0.5]

Cross Entropy

[This also gives the probability of other categories]

③ Sparse Categorical Cross Entropy

What will be output if there are equal Probability? (what will happen)

0 1st 2nd → Categories

[0.2, 0.3, 0.5]

⇓

2nd Index ⇒ o/p here it was neutral

Instead of giving a list of probability it will just return the index with the max^m probability

Disadvantage

① Loosing info about the probabi of other categories.

Which loss funct° to use When?

Combinations

	Hidden Layers	o/p layers (Activation funct° applied here)	Problem Statement	loss funct°
①	Rnn or its Variants	Sigmoid	Binary Classification	Binary Cross Entropy
②	Rnn or its Variants	Softmax	Multi Class	Categorical or Sparse CE
③	Rnn or its Variants	Linear	Regression	MSE, MAE, Huber loss, RMSE

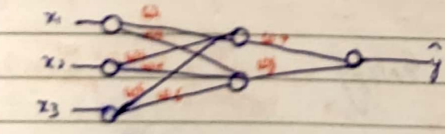
Optimizers (To know the how both these with optimizers both from)

① Gradient Descent
② Stochastic Gradient Descent (SGD)
③ Mini Batch SGD
④ SGD With Momentum
⑤ Adagrad and RMSPROP
⑥ Adam Optimizers

(1) Gradient Descent Optimizer :- [weight updation formula]

Forward & Backward propagation

[1]



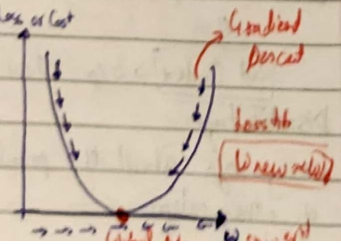
loss = [ ] ↓↓↓

Optimizers

Back ward Propagation

$$W_{new} = W_{old} - \eta \frac{db}{dW_{old}}$$

(learning rate)



MSE

loss $f_n = (y - \hat{y})^2$

Cost $f_n = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

Dataset = 1000 Data points

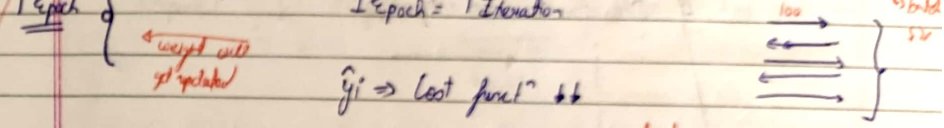
Epochs, Iteration

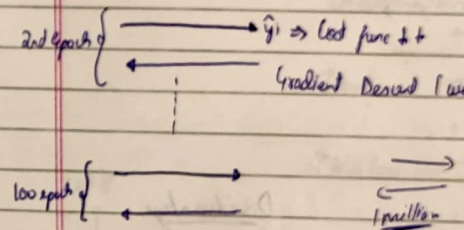
1 Epoch { 1000 datapoint → weight will get updated

$\hat{y}_i$ ⇒ Cost funct° ↓↓

1 Epoch = 1 Iteration

Gradient Descent optimizer [weight updation]





Advantages

① Convergence will happen.

Disadvantages

② Huge Resource RAM, GPU

↓

Resource Intensive

Epoch & Iteration are used to describe training process of a model.

Epoch refers to one complete pass through the entire training dataset. During one epoch the model sees every example in training dataset once.

Iteration refers to one update of the model's parameters (weights & biases) using a single batch of data. no. of iterations depends on the batch size.

③ Stochastic Gradient Descent (SGD)

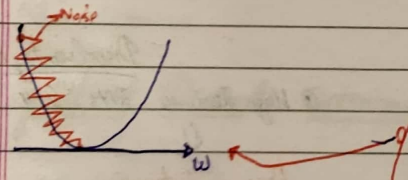
→ here, w.r.t to every iteration will be taking one data point and updating the weights.

Epoch 1
- Iteration 1 — 1 datapoint → $\hat{y}$ → loss func^n & optimizers; weight update
- Iteration 2 — 2nd datapoint → $\hat{y}$
- ⋮
- Iteration 1000 — 1000 datapoint

Epoch 2 → { Iteration 1 ⟹ ; Iteration 1000 ⟹ }

Advantage

① Solve Resource Issue



Disadvantage

① Time Complexity ↑ (Since in one iteration one datapoint is considered)

② Convergence will also take more time

③ Noise gets Introduced

③ Mini Batch SGD

Epoch, Iteration, Batch-size

No. of Iterations = 100000 ⇒ 100 iterations
1000

No. of iterations = Total No. of datapoints / Batch size = 100000 / 1000 = 100

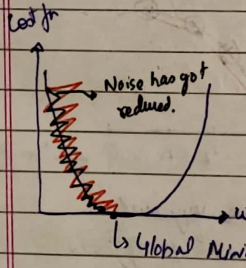
Epoch 1 — Iteration 1 — 1000 → cost fn = $\sum_{i=1}^{1000} (y_i - \hat{y}_i)^2$ & & ; change the weight
- loss will not be used
- optimizer ⇒ Mini Batch SGD

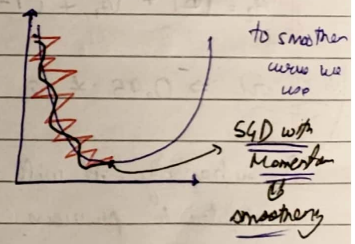
Iteration 2 — 1000 ⟹

Iteration 3 ⟹

⋮

Iteration 100 ⟹





Advantages

① Convergence speed will increase

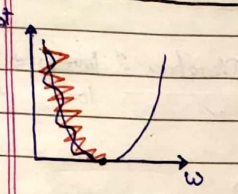
② Noise will be less when compared to SGD.

③ Efficient Resource usage (RAM)

Disadvantage

① Noise still exist.

4) SGD with Momentum Cur



$$W_{new} = W_{old} - \eta \frac{d\,loss}{d\,W_{old}}$$ (learning Rate)

$$b_{new} = b_{old} - \eta \frac{d\,loss}{d\,b_{old}}$$

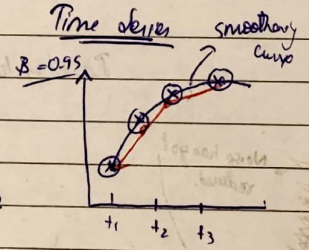
Weight updation formula!

$$W_t = W_{t-1} - \eta \frac{d\,loss}{d\,w_{t-1}}$$ → Current w.r.t time series

Exponential Weight Average (smoothening) ⇒ ARMA, SARIMAX

Time = t_1 t_2 t_3 t_4 ---- t_n

Values = a_1 a_2 a_3 a_4 ---- a_n

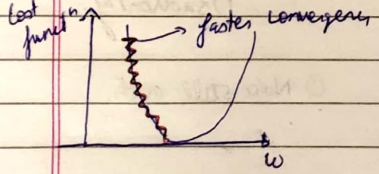


$V_{t_1} = a_1$ (smoothing function)

$V_{t_2} = \beta * V_{t_1} + (1-\beta) * a_2$

$\beta = 0.95 \Rightarrow 0.95 * a_1 + (0.05)\, a_2$

→ so basically it will make the the next value, very much related to previous value, resulting in decrease of difference b/w them & as a result smooth curve & faster convergence



Advantages

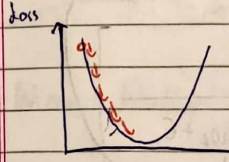
① Reduces the noise

② Quick convergence

5) Adagrad ÷ Adaptive Gradient Descent

η = ~~fixed~~ ⇒ Dynamic

$$w_t = w_{t-1} - \eta' \frac{d\,loss}{d\,w_{t-1}}$$ (learning Rate = 0.001; η = learning Rate)



As the convergence happen → when I am not exactly the learning rate should global minima → η should be high change → when I have reached near (global minima (η should be reduced)

$$w_t = w_{t-1} - \eta' \frac{d\,loss}{d\,w_{t-1}}$$

$w_t \approx w_{t-1}$

α_t ↑↑↑

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$ (Epsilon = small value)

η' ↓↓↓ α_t ↑↑

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{d\,loss}{d\,w_t}\right)^2$$

$t=1$ $t=2$ $t=3$

$\eta' = 0.01$ $\eta' = 0.005$ $\eta' = 0.003$

Disadvantage

① $\eta' \rightarrow$ Possibility to become a very small value ≈ 0

⑥ Adadelta And RMS PROP

Exponential Weighted Avg.

$$\eta' = \frac{\eta}{\sqrt{Sdw_t + \epsilon}}$$

$\beta = 0.95$

$Sdw_t = 0$ (initialized with zero)

$$Sdw_t = \beta * Sdw_{t-1} + (1-\beta)\left(\frac{d\,loss}{d\,w_{t-1}}\right)^2$$

$$w_t = w_{t-1} - \eta' \frac{d\,loss}{d\,w_{t-1}}$$

[Dynamic LR + Smoothening EWA]

ϵ

S = Smoothening EWA

Best Optimizers

⑦ Adam Optimizer

SGD with Momentum + RMSPROP [Dynamic Learning rate + Smoothening]

$w_t = w_{t-1} - \eta' V dw$ → weight updation

$b_t = b_{t-1} - \eta' V db$ → Bias updation

$\eta' = \frac{\eta}{\sqrt{S dw_t + \epsilon}}$

EWA

$S dw_0 = 0$

$S dw_t = \beta * S dw_{t-1} + (1-\beta)\left(\frac{d loss}{d w_{t-1}}\right)^2$

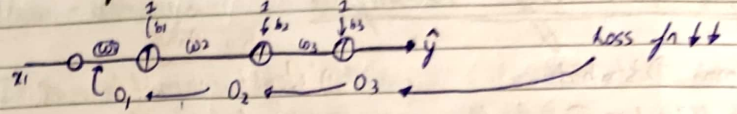
EWA (Exponential Weighted Avg)

$V dw_i = \beta * V dw_{t-1} + (1-\beta)\frac{d loss}{d w_{t-1}}$

$V db_t = \beta * V db_{t-1} + (1-\beta)\frac{d loss}{d b_{t-1}}$

⇒ Momentum smoothening

Exploding Gradient Problem



$\rightarrow z = (O_2 * w_3 + b_3) \rightarrow \sigma(z)$

$W_{inew} = W_{iold} - \eta\left[\frac{d loss}{d W_{iold}}\right]$ ⟹ two possibilities could come

if $\eta \frac{d loss}{d W_{iold}}$ is a very high −ve value

then $W_{inew} >>>> W_{iold}$ — ①

if $\eta \frac{d loss}{W_{iold}}$ is a very high +ve value

then $W_{inew} <<<< W_{iold}$

$\frac{d loss}{d W_{iold}} = \frac{d loss}{d O_3} \times \frac{d O_3}{d O_2} \times \frac{d O_2}{d O_1} \times \frac{d O_1}{d W_{iold}}$

$b_{ij} \times b_{ij} \times b_{ij} \times b_{ij} \Rightarrow$ b_{ij} value

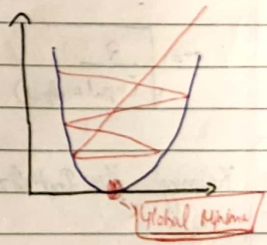
weight initialization ⇒ very big value

$\frac{d O_3}{d O_2} = \left|\frac{d \sigma(z)}{dz}\right| \times \frac{dz}{d O_2}$

$= [0 - 0.25] * \frac{d (O_2 * w_3 + b_3)}{d O_2}$

range: zero to 0.25

$\Rightarrow [0 - 0.25] * w_3$



To solve Exploding Gradient Problem these techniques, were ~~founded~~ invented

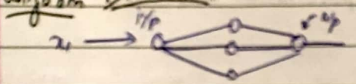
Weight Initialization Techniques

Key Points
- ① Weights should be small
- ② Weights should not be same
- ③ Weights should have good variance

1) Uniform Distribution
2) Xavier / Glorot Initialization
3) Kaiming He Initialization

(almost same)

① Uniform Distribution



input = 1
output = 1

weights = $\left[\frac{-1}{\sqrt{input}}, \frac{1}{\sqrt{input}}\right]$

$\left[\frac{-1}{\sqrt{1}}, \frac{1}{\sqrt{1}}\right]$

② Xavier / Glorot Initialization

① Xavier Normal Init

$W_{ij} \approx N(0, \sigma)$ (mean, std deviation)

$\sigma = \sqrt{\frac{2}{(input + output)}}$

② Xavier Uniform

$W_{ij} \approx$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{input + output}}, \frac{\sqrt{6}}{\sqrt{I/p + o/p}}\right]$

③ Kaiming He Initialization

① He Normal

$W_{ij} \approx N(0, \sigma)$

$\sigma = \sqrt{\frac{2}{Input}}$

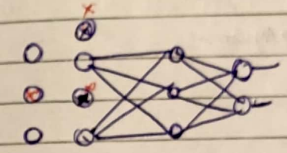
② He Uniform

$W_{ij} \approx$ Uniform Distribution

$\left[\frac{-\sqrt{6}}{\sqrt{input}}, \frac{\sqrt{6}}{\sqrt{input}}\right]$

Drop-out layer

It is a regularization technique use to prevent overfitting in neural network, dropout randomly drops a fraction of neurons during training, forcing the network to learn more robust features.



The features are only dropped in training data not in test data.

$0 \leq p \leq 1$ → selected by Hyperparameter

Convolutional Neural Network (CNN)

~~1) ANN → Supervised learning →~~

CNN is a type of deep learning algorithm primarily designed for image recognition and processing.

① ANN → Supervised learning → Classification, Regression

Data: I/p feature O/p

② CNN → I/p ⇒ Images eg: Image classification, Object Detection, Segmentation

Brain: two imp part of the brain that CNN
- Visual Area: sight, img perception, img recognition
- Motor function Area: Eye Movement and orientation (Hippocampal formation) (cerebral cortex)

V1, V2, ... Visual cortex

Visual Cortex (V_1 - V_5)

⇒ The region of brain that receives, integrates and processes visual information relayed from the retina.

→ V_1 → Primary Visual Cortex [Orientation of edges and lines]

V_2 [Differences in color, complex patterns, object orientation]

V_3, V_4, V_5 [Object Recognition]

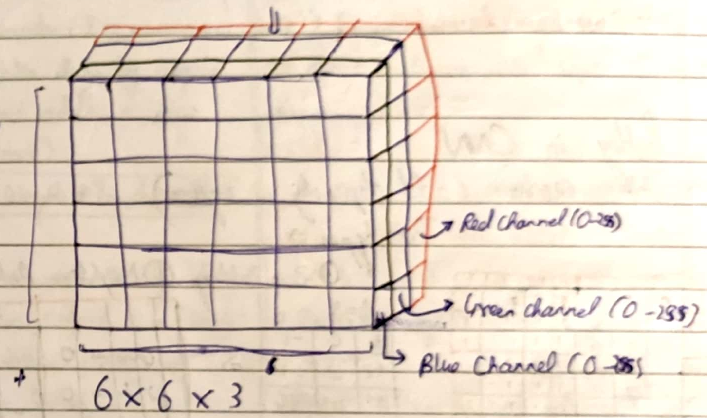
after all visualize the image

All about Images

③ RGB Image and Grayscale Image

An Image

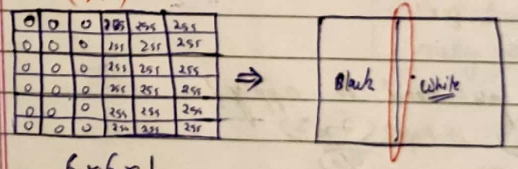
⇒ RGB → Red, Green, Blue ⇒ let's Visualize



④ Convolution Operation In CNN

[0-255] → This is the intensity of colour.

(0, 1) → black color, white color

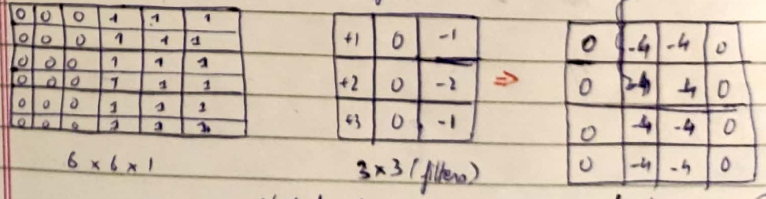


6 × 6 × 1

Steps:
① Normalize
Divide by 255

n=6, f=3

Convolution operation



n - f + 1 = 6 - 3

[Op size = (Ip size - filter size) + 1]

Step 2 → De-

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

⇒

White	Black	White

↳ here we have denormalise

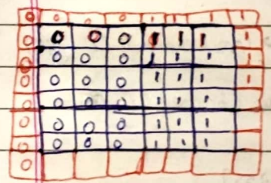
the

Padding in CNN

adding cushion (add generally on ~~options~~)

Two types ⇒

① Zero Padding ② Neighbour Padding

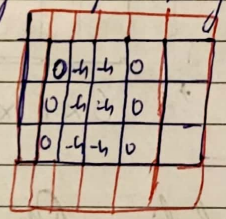


6×6 n=6 8×8

f=3

+1	0	-1
+2	0	-2
+1	0	-1

⇒



6×6

$n - f + 2p + 1 = o/p$

$6 - 3 + 2p + 1 = 6$

$3 + 2p + 1 = 6$

$p = 1$

Q ⇒ How much padding you need to apply?

(7×7) (3×3) (7×7)

i/p o/p

⇒ 1 layer of padding

Page No.:
Date: / /

6) Operation of CNN Vs ANN

ANN → (basic brain model for computer)

Step 1 : It takes inputs (like numbers or data), multiplies them by weights (importance values), and adds a bias (small adjustment)

Formula = Output = (Weights * Inputs) + Bias

Step 2 : It applies an activation function (like Relu) to this Output. Relu simply turns negative values to zero and keeps +ve values as they are

Step 3 : Calculate the error & update the weights

CNN

Step 1 : It takes an image and applies filters to detect patterns like edges, colors or shapes.

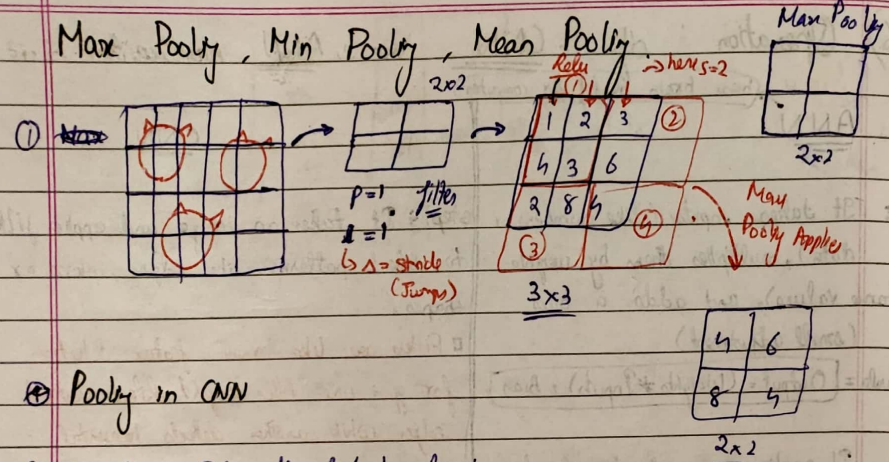
- Filters are like small feature detectors. for eg :- one filter might detect vertical edges, while another detects horizontal edges
- Initially, filters are random,

Step 2 :- after applying filters, it performs convolution operation (multiplying & adding values) to create feature maps.

Step 3 : It applies the Relu activation to feature maps. This helps in making network more efficient

Step 4 :- After Relu, it performs Max Pooling

Max Pooling, Min Pooling, Mean Pooling



Pooling in CNN

① Max Pooling: Takes the highest value in a region. Keeps strong features.

② Min Pooling: Takes the lowest value in a region. Rarely used

③ Mean Pooling: Takes the average of all values in a region. Smoothes features

Purpose of Pooling

① Reduces image size: Makes the img smaller, network runs faster

② Focuses on important features: ?

③ Prevents overfitting: Helps the network generalize to new data

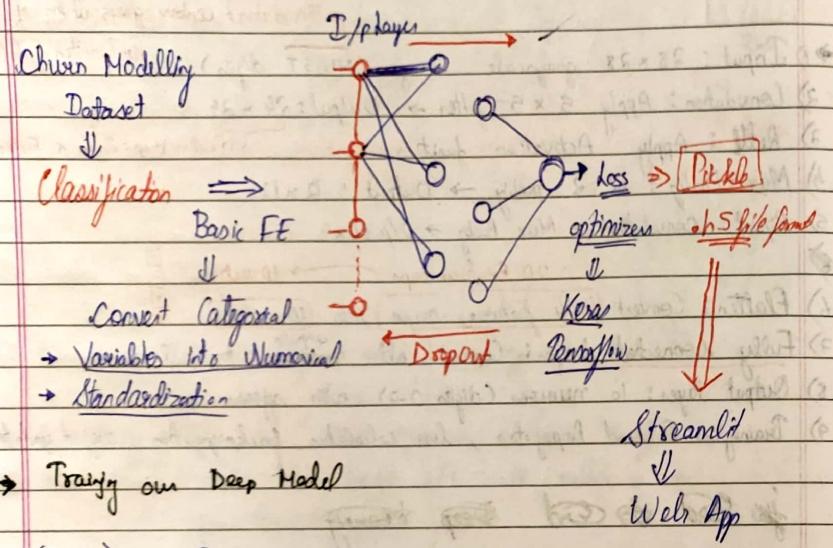
④ Location Invariance: A CNN can recognise features (like objects or Patterns) no matter where they are located in an image. achieved. this is achieved through pooling

How CNN works in the context of handwritten digit dataset.

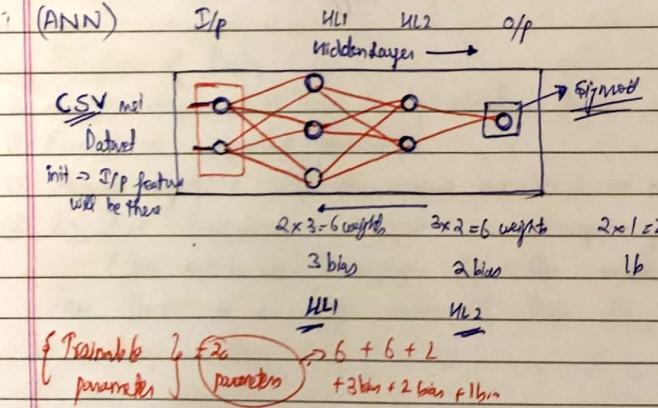
1) Input: 28×28 grayscale image (MNIST digits). ⇒ This dataset contains grayscale img of handwritten digits (0-9)
2) Convolution: Apply 5×5 filter → Output: 24×24. ↳ Feature Map = Img I/p × Filter
3) ReLU: Apply Activation function.
4) Max Pooling: 2×2 pooling → Output: 12×12
5) Repeat: Convolution + Max Pooling → O/p: 4×4
6) Flatten: Convert 4×4 feature maps into 1D vector (2D Feature maps → 1D vector)
7) Fully Connected Layer: Connect all inputs to all neurons.
8) Output layer: 10 neurons (digits 0-9) with softmax.
9) Training: Forward Propagation, loss calculation, backpropagation, weight updates

~~Deep~~

End to End Deep Learning Project :- ANN



→ Training our Deep Model



① Sequential N/w → This goes in sequence
② Dense → bit (To set no. of neurons in hidden layer)
③ Activation func" → sigmoid, tanh, Relu, Leaky Relu
④ Optimizers → Back Propagation → Updating the weights
⑤ Loss func" ++
⑥ Metrics → [accuracy], [mse, mae] ⑦ Training → logs → folder → Tensorflow → Visualize

ANN Implementation

```
① import tensorflow as tf
import tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard
import Datetime

(X_train.shape[1],)
        → (12,)

# Build our ANN model

② model = Sequential([                    HL1 connected with input layer
        Dense(64, activation="relu", input_shape=(X_train.shape[1],)),
        Dense(32, activation="relu"),      ## HL2
        Dense(1, activation="sigmoid")     # O/p layer
    ])

③ → model.summary()

④ import tensorflow
opt = tensorflow.keras.optimizers.Adam(learning_rate=0.01)
loss = tensorflow.keras.losses.BinaryCrossEntropy()

# Compile the Model
model.compile(optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])

# Set up the Tensorboard
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard

log_dir = "logs/fit" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorflow_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)
```

Page No.:
Date: / /

5. # Set up Early Stopping
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

6. # Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, callbacks=[tensorflow_callback, early_stopping_callback])

7. model.save('model.h5')

8. # load Tensorflow extension
%load_ext tensorboard

9. for visualization :- %tensorboard --logdir logs/fit

Page No.:
Date: / /

Experiments.ipynb

1. import pickle, scalar, encoder & others & train_test_split
2. Read the data (read_csv)
3. Preprocess the data
   a) Drop unnecessary columns b) Encode categorical & OHE (fit_transform)
   b) data[['Geo']].toarray()
   c) convert to df (OHE) d) combine df to original data (OHE)
4. Save label encoder & OHE (Using pickle dump)
5. Do scaling & save it
6. # ANN -> import tensorflow, sequential, Dense, EarlyStopping, tensorboard, datetime
7. Using (X_train.shape[1]) we get no. of i/p of first hidden layer
8. # Build our ANN model
   model = Sequential([Dense(64, activation='relu', input_shape=(X_train.shape[1],)), HL1 connected with I/p layer
) HL2 (
   HL3)])
   # Optimizer
9. import tensorflow.. -> select optimizer & loss
   opt = tensorflow.keras.optimizers.Adam(learning_rate=0.01)
   loss = ?
10. # Compile the model -> model.compile(optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])
    # Set up Early Stopping & Train Model -> EarlyStopping(monitor="val_loss", patience=10, restore best weights=True)
11. history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=[100], callbacks=[tensorflow_callback, early_stopping_callback])
12. model.save(model.h5)

Page No.:
Date: / /

Prediction.ipynb

1) Import ⇒ tensorflow.keras.models, load_model, pickle, numpy, pandas

2) load the trained model
model = load_model("model.h5")

3) load the encoders & scales
using pickle.load(file)

Example i/p data

4) input_data = {
Convert input into DF

5) OHE 'Geography'

6) Encode categorical variable

7) Concatenate & scale.

8) Predict churn