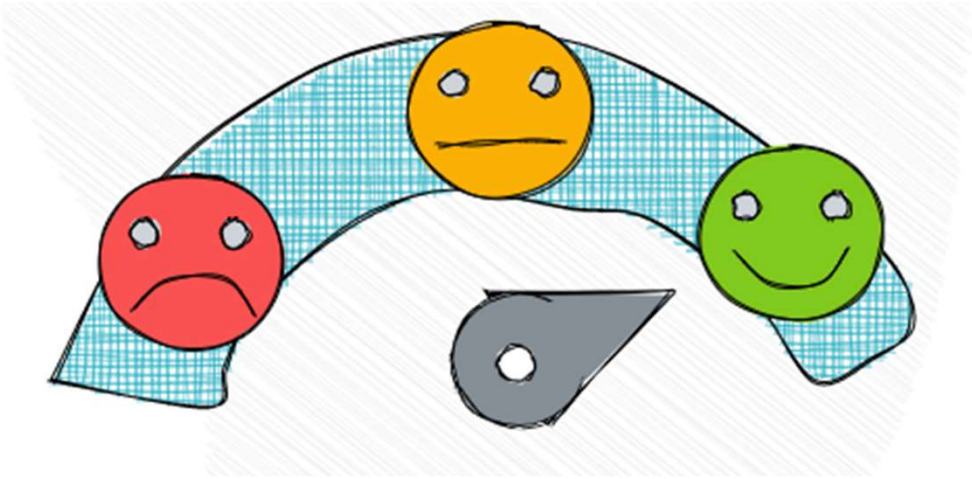
# Sentimental AI

BRAIN AND COGNITIVE SOCIETY

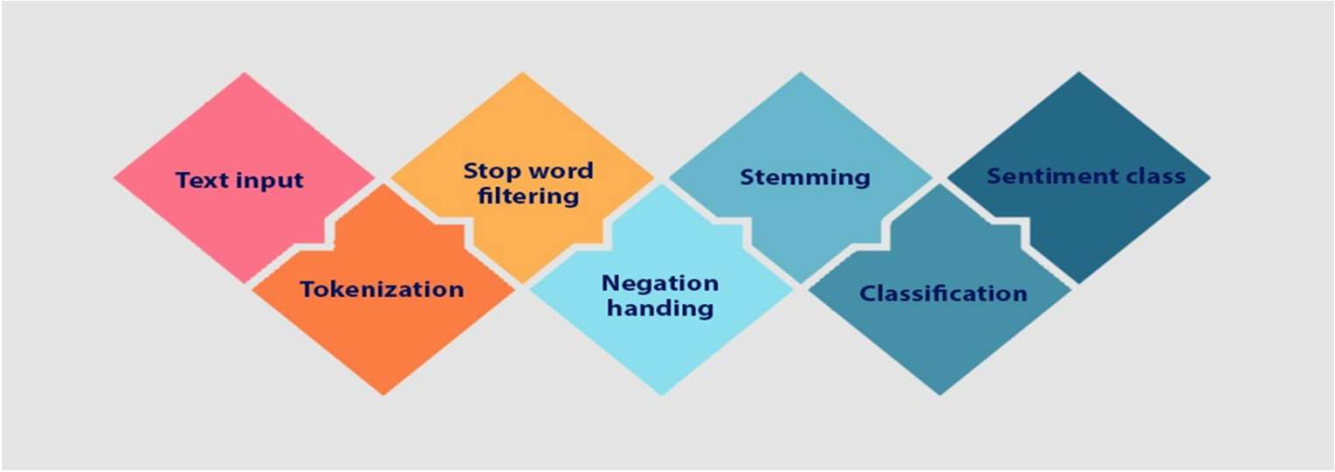
IIT KANPUR



# Overview

Sentimental AI uses NLP and ML to analyse the emotion behind a piece of text.

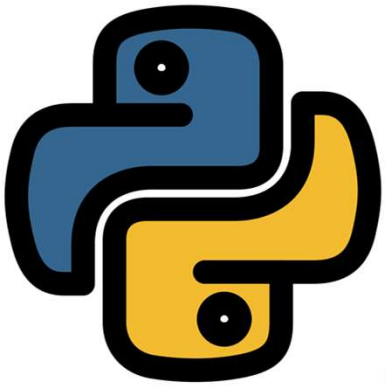
Basics of ML and DL→CNNs,RNNs,LSTMs →Train RNNs and LSTMs Transformers→Implementing emotion classifier



# Progress Till Now

- Learned basics of Python, Jupyter Notebook, Numpy, Pandas, Matplotlib, Tensorflow, GitHub and Git
- Learned basics of Machine Learning, Deep Learning, Neural Networks, Activation functions, Overfitting and Underfitting
- Solved 1st Assignment which included implementation of Neural Network
- Learned about RNN, LSTMs and NLP in tensorflow
- Solved 2nd Assignment which included implementation of RNNs, LSTMs and NLP in tensorflow
- Worked on Transformers and imple

Week 1 - Getting familiar with the tools





NumPy


TensorFlow

git

GitHub

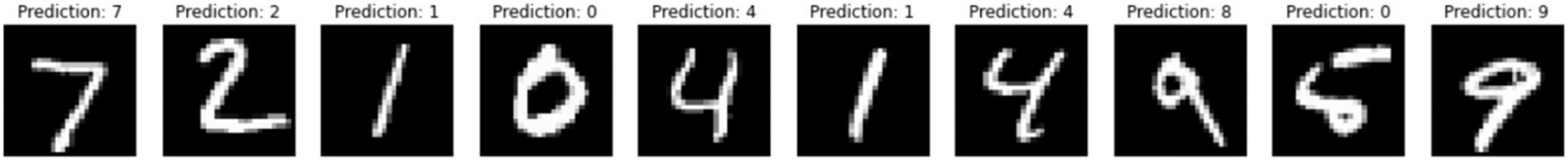
matplotlib

# Week2

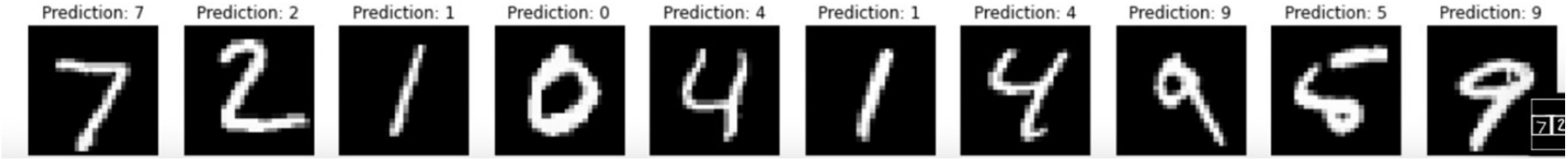
- Got introduced to the techniques in supervised learning; including linear regression and logistic regression.
- Gradient Descent Algorithm: We calculate the error in each iteration and update the model parameters accordingly.
- Under-Fitting: When the model does not fit the training data well. Has low variance but high bias.
- Over-Fitting: When the model fits the training data well but performs badly on testing data. Has high variance but low bias.
- Techniques to prevent overfitting: Reducing number of features, Early Stopping, Regularization
- Cross Validation: Dividing available data to text and validate model.

# Assignment 1

- Implemented a Neural Network from scratch using Numpy
- It was applied on recognizing handwritten digits from the MNIST Dataset
- Our neural network had 4 layers and we used Sigmoid and Softmax activation functions.




- Later we also implemented the same using Tensorflow 2.0



# Week 3

- We explore the concepts of Recurrent Neural Networks (RNN), including the tool of Long short-term memory (LSTM), starting from first principles.
- We delve into Natural Language Processing (NLP) with all its intricacies, and its implementation in Tensorflow.
- We get familiar with the process of tokenization, as well as the practice of padding sentences in order to sequence them efficiently.
- We learn about training a neural network to detect emotions in a sample of text.
- We build towards the generation of meaningful text by our model, like the prediction of words in a sentence.

# Assignment 2 – Generating poetic texts using recurrent neural networks:-

```
tokenizer = Tokenizer()
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sonnets.txt \
    -O /tmp/sonnets.txt
data = open('/tmp/sonnets.txt').read()

corpus = data.lower().split("\n")


tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

Tokenization is performed on the corpus to obtain tokens. The following tokens are then used to prepare a vocabulary. Vocabulary refers to the set of unique tokens in the corpus.

The list of tokens can then be used as sequences as an input for implementation of the model.

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

Padding is added to the input sequences to make the length equal to the length of the largest sentence.

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))  #(# Your Em
model.add(Bidirectional(LSTM(150, return_sequences=True)))  #(# An LSTM Layer)
model.add(Dropout(0.2))  #(# A dropout layer)
model.add(LSTM(100))  #(# Another LSTM Layer)
model.add(Dense(total_words/2, activation='relu'))  #(# A Dense Layer including regu
model.add(Dense(total_words, activation='softmax'))  #(# A Dense Layer)
# Pick an optimizer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics='accuracy')
print(model.summary())
```

Embedding layer - direction of each word is learned epoch by epoch.
Bidirectional Layer - connect two hidden layers of opposite directions to the same output.
Dropout layer - prevents a model from overfitting.
Then the data is fed into atypical dense layer using 'relu' and 'softmax' activations.

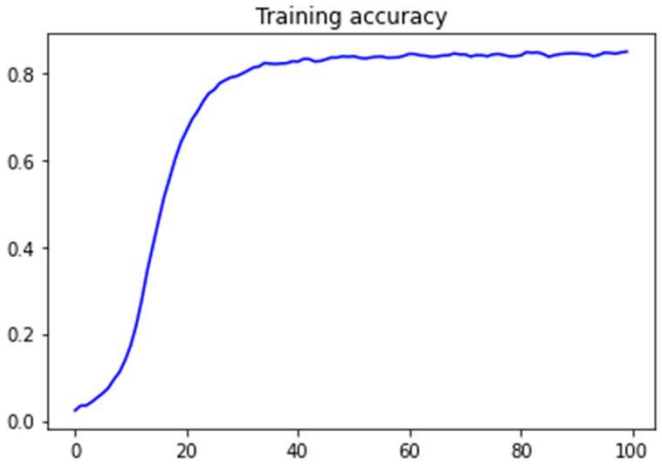
*n - gram sequencing is built by counting how often word sequences occur in corpus text and then estimating the probabilities.

# Implementation

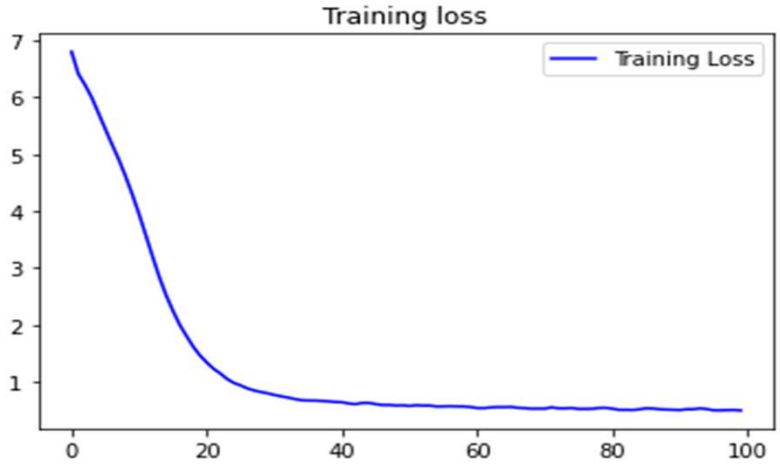
- Created a corpus of lines to fit the tokenizer on corpus
- Created input and pad sequences
- Used the Keras Sequential Model added with different layers and compiled it using Adam as the optimizer
- Fitted the training data and trained the model on 100 Epochs

| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding (Embedding) | (None, 10, 100) | 321100 |
| bidirectional (Bidirectional) | (None, 10, 300) | 301200 |
| dropout (Dropout) | (None, 10, 300) | 0 |
| lstm_1 (LSTM) | (None, 100) | 160400 |
| dense (Dense) | (None, 1605) | 162105 |
| dense_1 (Dense) | (None, 3211) | 5156866 |

*No. of epochs vs accuracy*




*No. of epochs vs loss*



# Week 5 & Week 6


- Got to learn about **Transformers**, a current state-of-art type of model for dealing with sequences.
- It is used primarily in the fields of NLP and CV.
- It was first proposed in the paper "**Attention is All You Need**".
- It has, perhaps the most prominent application in text processing tasks, and the most prominent of these is machine translation.
- It is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data.
- Proceeded towards understanding **BERT**, a SOTA model which will be used for sentimental analysis.

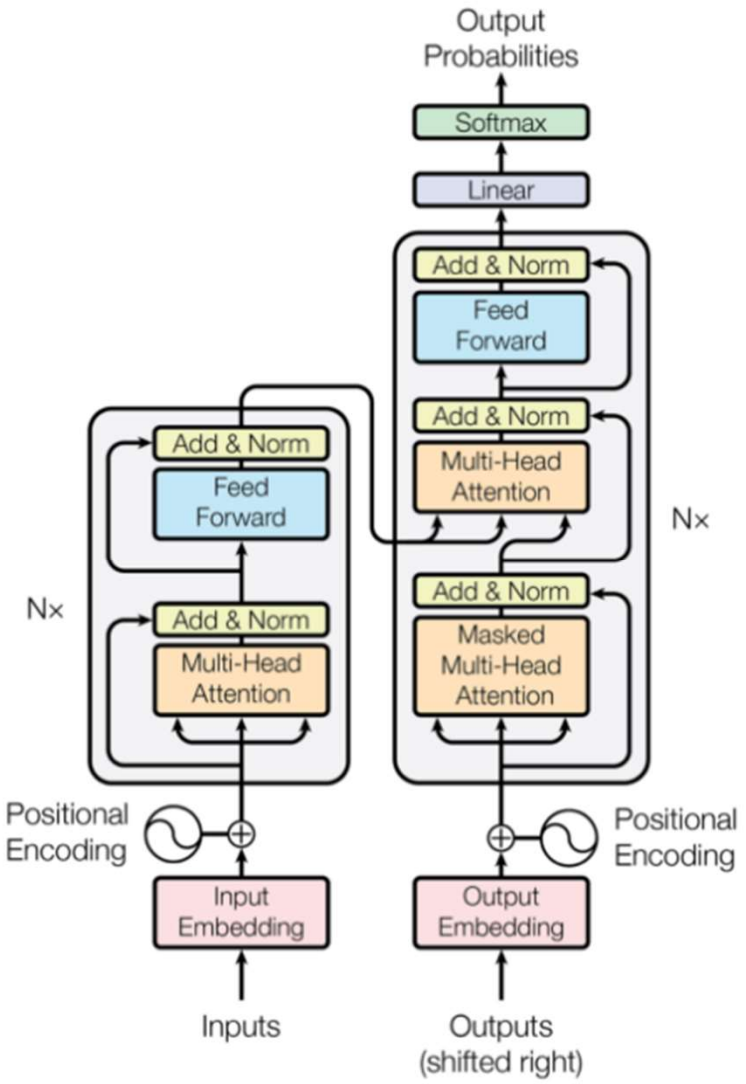



Figure 1: The Transformer - model architecture.

# Transformer

1. The Encoder is on the left and the Decoder is on the right. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times, which is described by *Nx* in the figure.
2. the modules consist mainly of Multi-Head Attention and Feed Forward layers.
3. The inputs and outputs (target sentences) are first embedded into an n-dimensional space since we cannot use strings directly.
4. Since we have no recurrent networks that can remember how sequences are fed into a model, we need to somehow give every word/part in our sequence a relative position since a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word
5. Multi -headed attention gives how each of the words have an influence on one another.
6. the Transformer applies a mask to the input in the first multi-head attention module to avoid seeing potential ‘future’ sequence elements.
7. This is specific to the Transformer architecture because we do not have RNNs where we can input our sequence sequentially.

# BERT - *Bidirectional Encoder Representations from Transformers*

1. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it's non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word)

### Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with softmax.

### Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence.

To predict if the second sentence is indeed connected to the first, the following steps are performed:

1. The entire input sequence goes through the Transformer model.
2. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
3. Calculating the probability of ISNextSequence with softmax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

# Week 7

- After learning about pre-trained transformers, we used them by fine-tuning them for real world problems like Sentimental Analysis.
- We built a Emotion Classifier using BERT (language model).
- We also used fast tokenizers to efficiently tokenize and pad input text as well as prepare attention masks.
- We learnt to prepare reproducible training code with the help of PyTorch Lightning library.
- We then compared the Emotion Classification with vanilla LSTM and Nested LSTM.

# Overall outcome

- We were introduced to the basics of deep learning. Then, we learned about and implemented several deep learning architectures: **Neural Networks**, **Long Short Term Memory (LSTM) models** and **transformers**
- We were introduced to the scientific computing/machine learning libraries **Numpy**, **Tensorflow** and **Keras**.
- We finally implemented emotion classifiers using LSTMs, based on a 2019 research paper.

# Mentors

Shashwat Gupta

Hitesh Anand

Sahil Bansal

# Thank You