

Neural Networks

Team Members:
Vedantee Pathak (MT2023123)
Sreya Goswami (MT2023167)



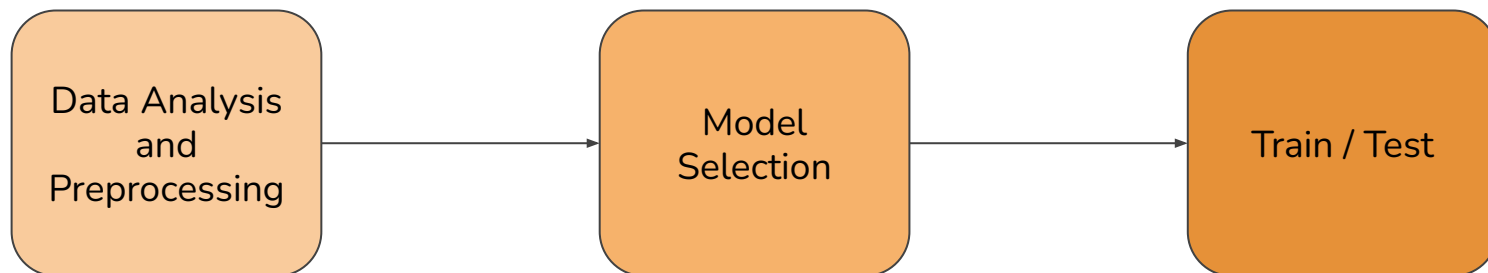


Agenda - Book Review

- Process Overview
- Data Analysis
- Data Preprocessing
- Model Selection
- Results



Process Overview



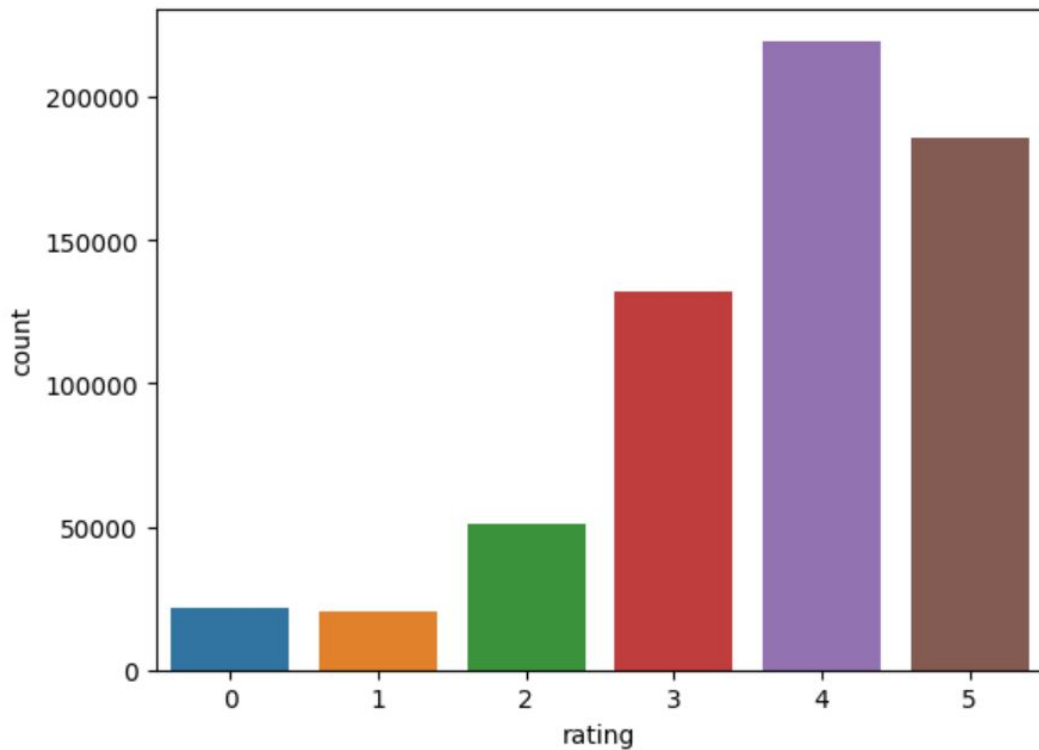


Exploratory Data Analysis

- Visualize the data using histograms, box plots, scatter plots, etc., to understand relationships between features and the target variable.
- Check for correlations between features using correlation matrices or heatmaps to identify highly correlated features.
- Explore distributions, trends, and patterns in the data that might inform feature engineering or model selection.



Rating vs Count Histogram





Number of Comments vs Rating Count Plot





Data Cleaning

Prepare the data for model training by performing various preprocessing steps:

- Handle missing values: Decide on strategies such as imputation (filling missing values with a measure of central tendency) or removal of rows/columns with missing data.
- Handle duplicates: Check for and remove any duplicate rows in the dataset if they exist.
- Remove irrelevant or redundant features that might not contribute meaningfully to the task at hand.
- Handle outliers: Decide how to deal with outliers, whether it's through removal, transformation, or capping/extending values.



Data Transformation

- Convert categorical variables into numerical representations (One-Hot Encoding, Label Encoding) if your machine learning algorithm requires numerical input.
- Normalize or scale numerical features to a similar scale (StandardScaler, MinMaxScaler) to prevent certain features from dominating due to their scale.



```
stop = {'a', 'the', 'www', 'http', 'https', 'com'}
```

```
def remove_stopwords(text):  
    return ' '.join([word for word in text.split() if word not in stop])
```

```
import string  
remove = string.punctuation  
  
period = '.'  
remove = remove.replace(period, '')  
  
def remove_punctuation(text):  
    pattern = re.compile(r"[{}]".format(re.escape(remove)))  
  
    res = []  
    for word in text.split():  
        # remove all punctuations except periods  
        new_word = pattern.sub(r' ', word)  
        new_word = new_word.strip(period)  
  
        try:  
            float(new_word)  
        except:  
            new_word = new_word.replace(period, ' ')  
  
        res.append(new_word)  
  
    return ' '.join(res)
```



Model Selection

- Base Model
 - Basic Neural Network
 - Pre-trained model: `xlm-roberta-base`
- Selecting layers
 - The input layer size should match the dimensionality of your input data.
 - The number of hidden layers and their neurons depends on the complexity of the problem.
 - As it is a multi-class classification problem, we use neurons equal to the number of classes with softmax activation.
- Loss Function
 - Categorical Cross-Entropy: Suitable for multi-class classification tasks.
- Activation Function
 - ReLU (Rectified Linear Unit): Commonly used due to its simplicity and effectiveness in overcoming vanishing gradient problems.
 - Softmax: Typically used in the output layer for multi-class classification problems to get probability distributions over classes.



Hardware Selection

Selecting hardware for Kaggle competitions depends on various factors including the specific tasks, computational requirements, budget constraints, and personal preferences. Generally, both GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) are commonly used for machine learning tasks due to their ability to accelerate computations in neural networks and other complex models. Here's a breakdown:

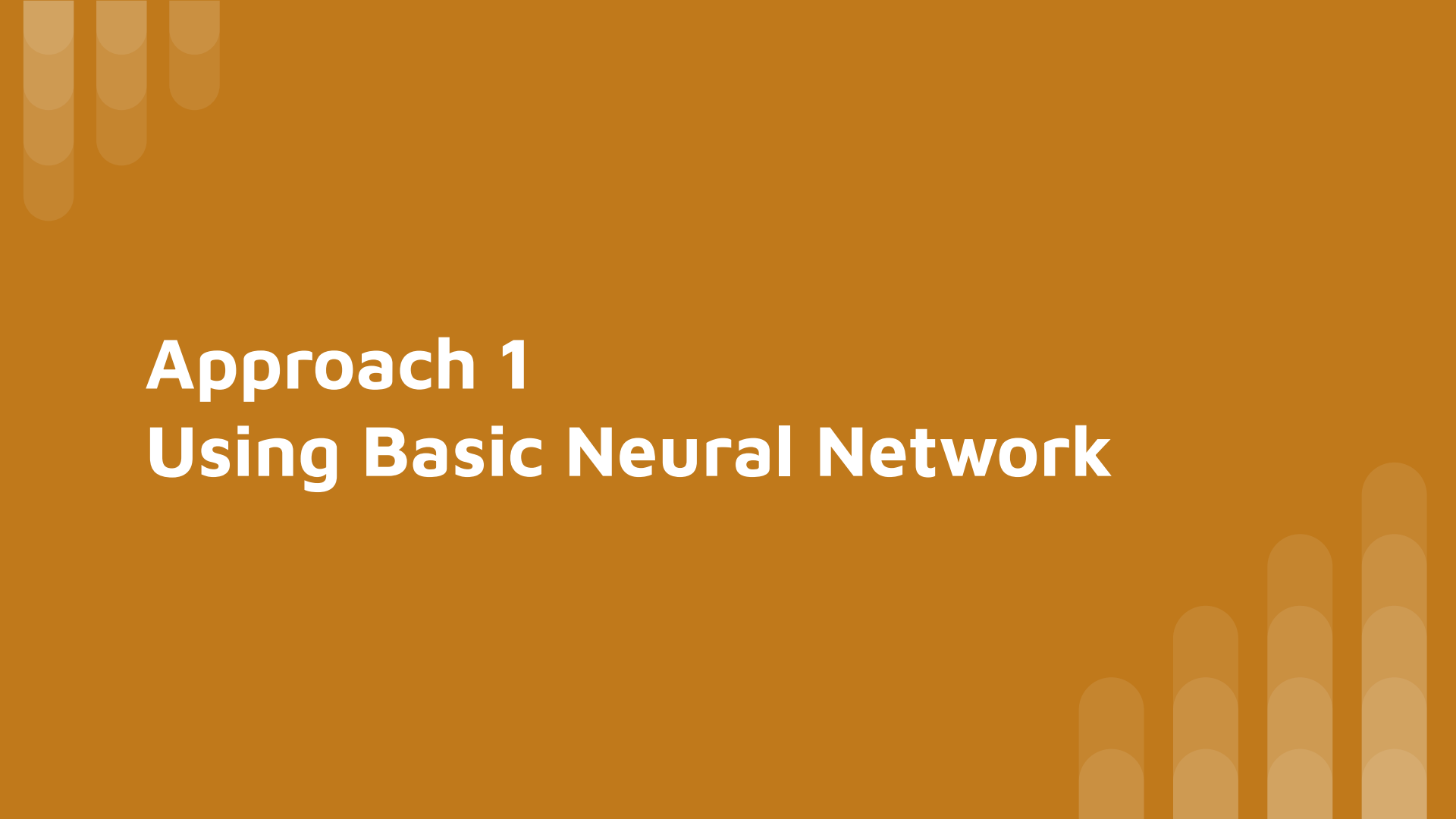
1. GPU (Graphics Processing Unit):
 - NVIDIA GPUs: Popular choices for deep learning tasks due to their CUDA support and extensive libraries like TensorFlow and PyTorch optimized for NVIDIA architectures.
 - AMD GPUs: Increasingly gaining attention for machine learning tasks with improvements in their software ecosystem and performance.
2. When selecting a GPU, consider factors like the amount of VRAM (Video RAM), CUDA cores (for NVIDIA), and the specific model's performance in the tasks you intend to perform.
3. TPU (Tensor Processing Unit):
 - Google Cloud TPU: Google's custom hardware accelerators designed specifically for machine learning workloads, particularly efficient for large-scale neural network training and inference.
4. TPUs are highly efficient for certain types of tasks, particularly when using Google's TensorFlow framework and working within Google Cloud.



Hardware Selection

```
tpu = None
gpus = tf.config.experimental.list_logical_devices("GPU")

if tpu:
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('Running on TPU')
elif len(gpus) > 1:
    strategy = tf.distribute.MirroredStrategy([gpu.name for gpu in gpus])
    print('Running on multiple GPUs ', [gpu.name for gpu in gpus])
elif len(gpus) == 1:
    strategy = tf.distribute.get_strategy()
    print('Running on single GPU ', gpus[0].name)
else:
    strategy = tf.distribute.get_strategy()
    print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)
```



Approach 1

Using Basic Neural Network



Layers

Conv1D (1D Convolutional Layer)

- Purpose: Performs 1D convolution on sequential data.
- Usage: Typically used in tasks such as natural language processing (NLP) or time-series analysis.
- Functionality: Learns local patterns in the data by applying filters (kernels) across the sequence, capturing dependencies between neighboring elements.



Layers

MaxPooling1D (1D Max Pooling Layer)

- Purpose: Performs downsampling by retaining the maximum value within each window.
- Usage: Often applied after convolutional layers to reduce dimensionality and computational complexity.
- Functionality: Helps in extracting dominant features while reducing the input size, improving computational efficiency, and preventing overfitting.



Layers

Bidirectional LSTM (Bidirectional Long Short-Term Memory)

- Purpose: A type of recurrent neural network (RNN) that processes sequences bidirectionally.
- Usage: Suitable for sequential data tasks where past and future contexts are both important.
- Functionality: Contains two LSTM layers, one processing the input sequence forward and the other backward, capturing information from past and future timesteps simultaneously.



Layers

Dropout

- Purpose: A regularization technique used to prevent overfitting in neural networks.
- Usage: Applied during training to improve generalization and reduce the likelihood of model memorization.
- Functionality: Randomly sets a fraction of input units to zero at each update during training, which helps in preventing the co-adaptation of neurons and encourages robustness.



Layers

Dense (Fully Connected Layer)

- Purpose: Classic fully connected layer connecting every neuron from the previous layer to the next layer.
- Usage: Often used as the output layer or intermediate layers in neural networks.
- Functionality: Each neuron in the layer is connected to every neuron in the previous and following layers, applying a linear transformation followed by an activation function.



Result - Score

0.57598



Approach 2

Using XLM-RoBERTa





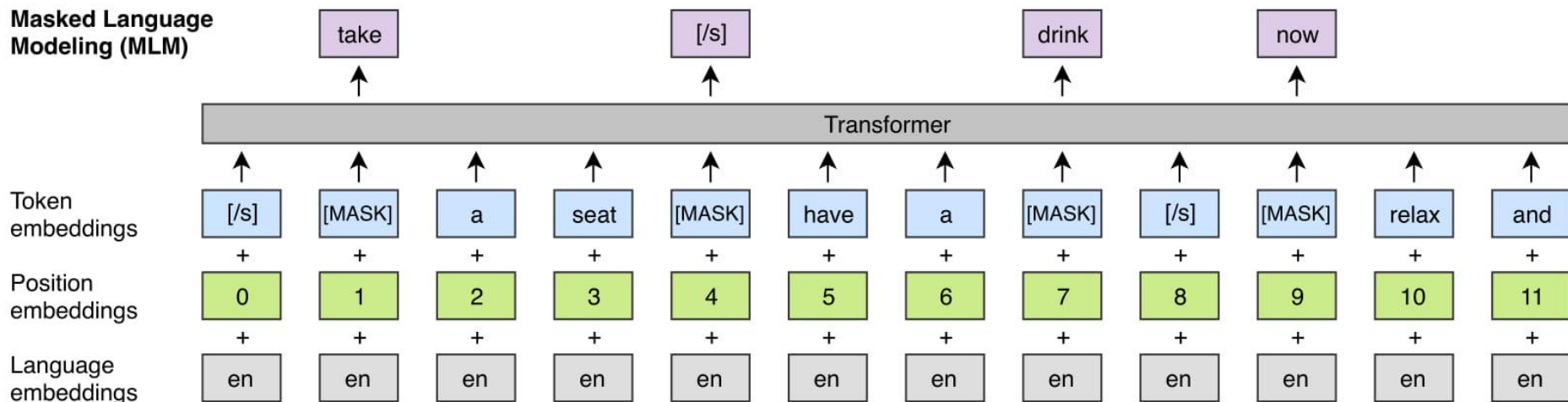
XLM-RoBERTa (Cross-lingual Language Model with RoBERTa architecture)

A pre-trained language representation model developed by Facebook AI. It's an extension and adaptation of RoBERTa (Robustly optimized BERT pre training approach) that is designed to handle multilingual data and tasks.

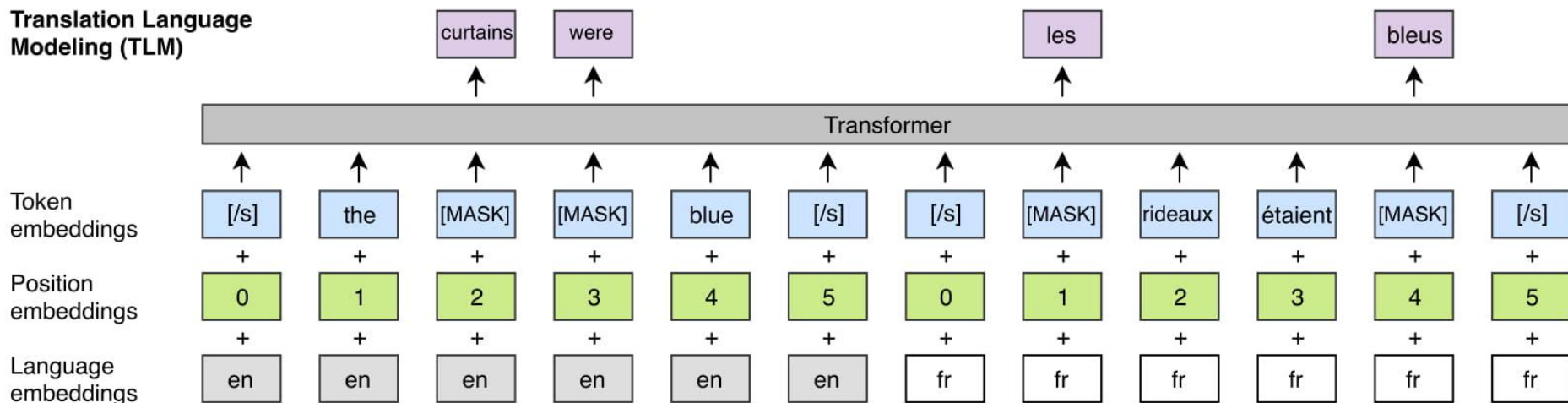
RoBERTa Architecture: XLM-RoBERTa is built upon the RoBERTa architecture, which is an extension of the original BERT (Bidirectional Encoder Representations from Transformers) model. RoBERTa makes improvements in the pretraining process by leveraging larger datasets and training for longer durations, employing dynamic masking, and removing the Next Sentence Prediction (NSP) task, among other modifications. These changes help enhance the model's performance across various natural language understanding tasks.

Transformer Architecture: Like BERT and RoBERTa, XLM-RoBERTa utilizes the Transformer architecture, which consists of stacked self-attention layers. Transformers enable the model to capture contextual relationships within text sequences effectively, allowing it to learn rich representations of language.

Masked Language Modeling (MLM)



Translation Language Modeling (TLM)



```

with strategy.scope():
    encoder = TFAutoModel.from_pretrained(model_name)

    input_word_ids = Input(shape = (MAX_LEN, ), dtype = tf.int32, name = "input_ids")
    input_mask = Input(shape = (MAX_LEN, ), dtype = tf.int32, name = "attention_mask")

    embedding = encoder([input_word_ids, input_mask])[1] # pooled_output
    x = Dropout(0.3)(embedding)
    x = Dense(128, activation = 'relu', kernel_regularizer = regularizers.L2(0.1))(x)
    x = Dropout(0.3)(x)
    x = Dense(32, activation = 'relu', kernel_regularizer = regularizers.L2(0.1))(x)
    x = Dropout(0.3)(x)
    x = Dense(y_cat.shape[1], activation = 'softmax')(x)

    model = Model(inputs = [input_word_ids, input_mask],
                  outputs = x)

    model.compile(Adam(learning_rate = 1e-5),
                  loss = 'categorical_crossentropy',
                  metrics=['accuracy'], steps_per_execution = 200)

model.summary()

```



Results - Score

0.7658