Machine Learning & Predictive Analytics

Final Project Report

May 25, 2023 | Chicago, Illinois

Kshitij Mittal

Problem Statement

Problem Statement

- The objective is to evaluate the English language proficiency of 8th-12th grade students by analyzing their essays
- To enhance the support for ELLs, a model will be developed using Deep Learning techniques
- This model can help reduce the grading burden for teachers and provide better feedback to students on their essays

Who is an ELL?

- ELLs refer to students who face difficulty communicating effectively and learning in English
- ELLs belong to households and backgrounds where English is not the primary language.
- Due to this, they need special or adjusted instruction to improve their language skills and academic performance.

Data Assumptions

- The dataset presented comprises argumentative essays written by 8th-12th grade English Language Learners (ELLs).
- The essays have been scored according to six analytic measures: cohesion, syntax, vocabulary, phraseology, grammar, and conventions.
- Each measure represents a component of proficiency in essay writing, with greater scores corresponding to greater proficiency in that measure. The scores range from 1.0 to 5.0 in increments of 0.5.

Our task is to predict the score of each of the six measures for the essays given in the test set.

3911 text essays

6 continuous target variables

PII Masked to ensure personnel privacy

Text: Feature Engineering

1
Text Cleaning

a. Apply contractions (don't → do not)

b. Cleaning:
- Lower text, Strip extra spaces, Tab spaces
- Remove Stop words and Punctuation (and, the, for)

c. Word Tokenization

2
Lemmatize & POS

a. Part of Speech Identification:
- Adverb
- Noun
- Verb
- Adjective

b. Lemmatization (Studying → Study)

3
Spelling Mistakes

a. Identifying Mistakes

b. Correcting Mistakes to reduce token redundancy while modeling

4
Count Aggregation

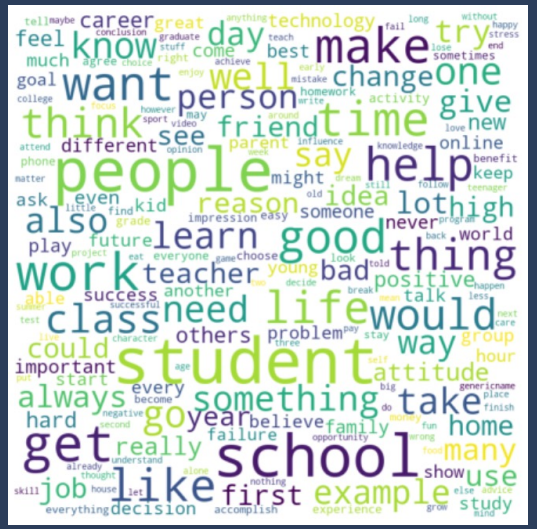
a. Word Count

b. Count Spelling Mistakes

c. Count Sentence Metrics
- Length
- Count

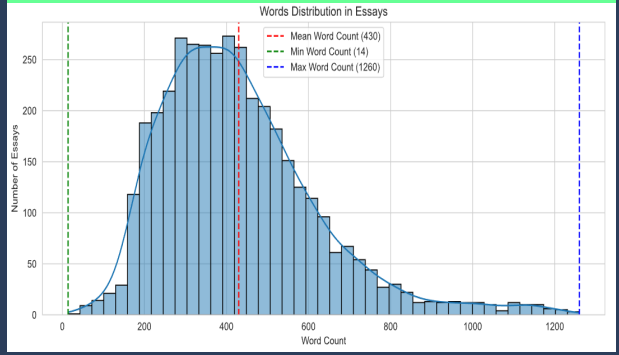
d. POS counts – Noun, Adjective, Adverb & Verb

Exploratory Data Analysis



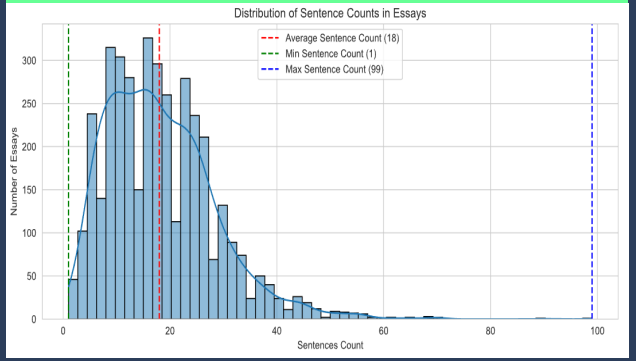
Most words are related to student life, schooling and education

‘Cleaned’ Essay Lengths



Some students used more words to write essays and made the normal distribution rightly skewed.

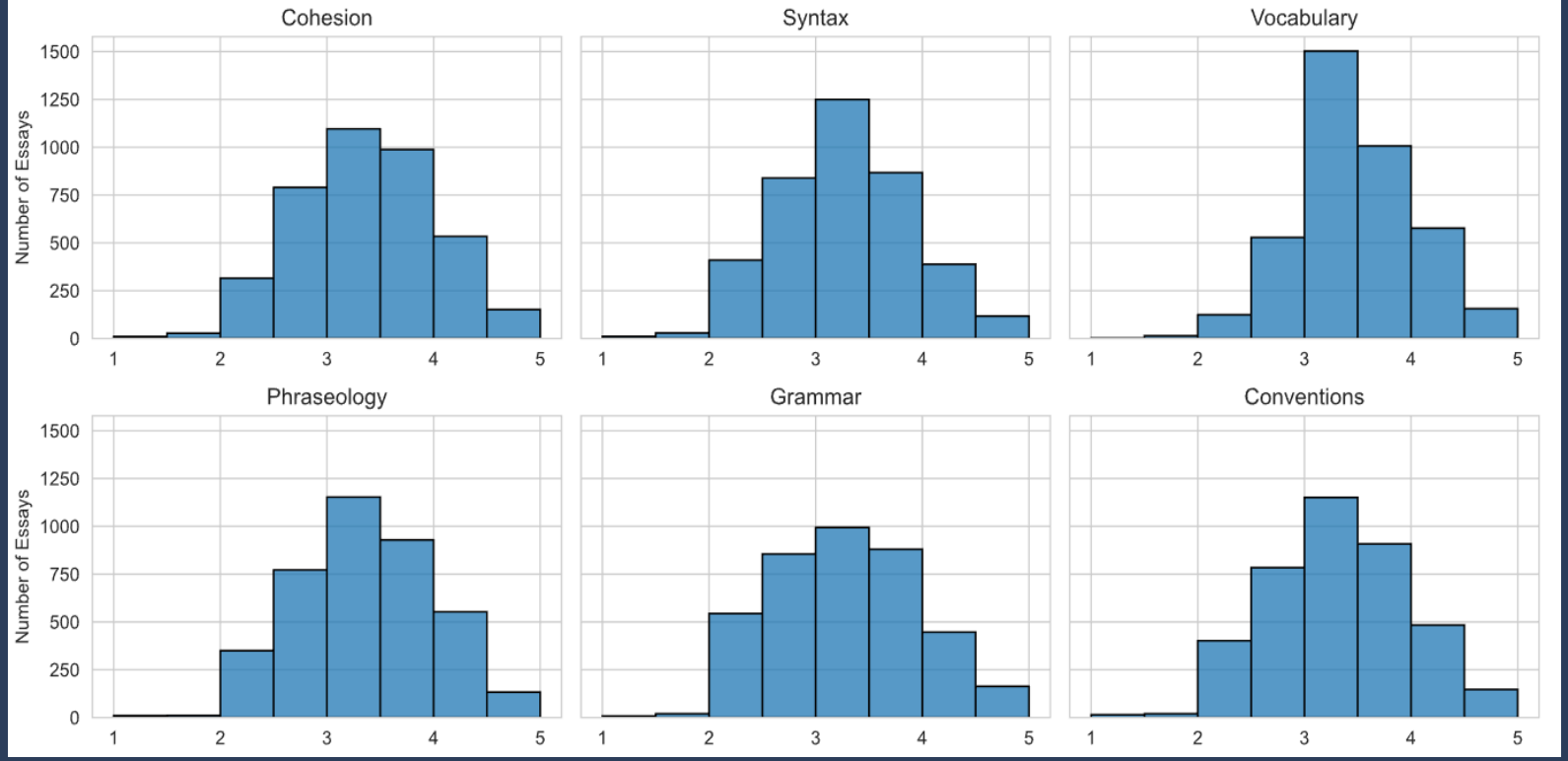
Distribution of Sentence Count



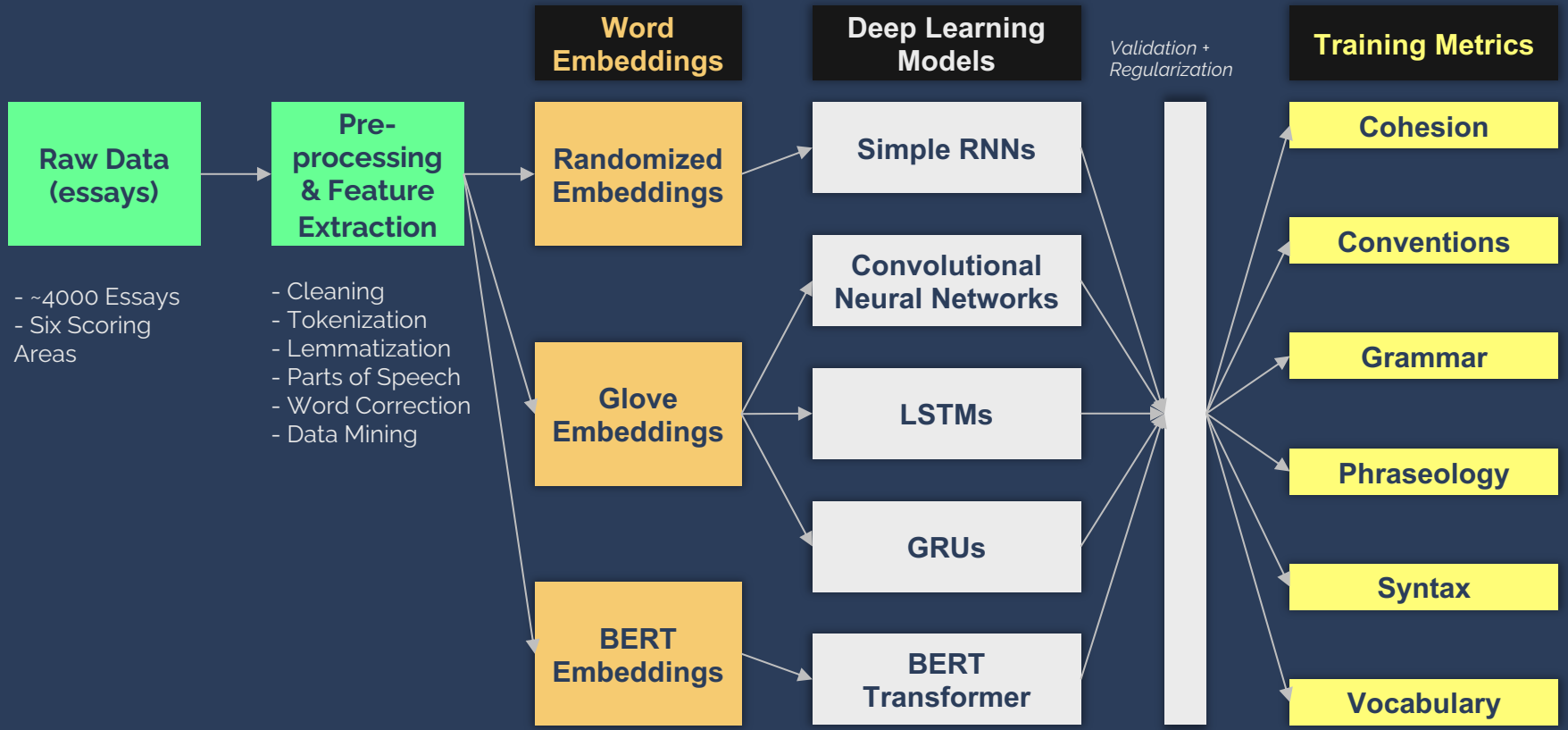
Each student exhibits a distinct writing style, with number of sentences varying in each essay

Target Variable

6 Target variables are used to measure different components of a student's writing proficiency



Proposed Modeling Approaches



Proposed Approaches

Owing to the sequential nature of the data (text), multiple deep learning approaches were employed to predict scores for each essay.

Word Embeddings used to pre-process text

Each token in the text was embedded into a deeply nested list of numerical representations.

- I first tried Keras native embedding, which is trainable and initialized randomly
- Then, I used GloVe embeddings. GloVe (Global Vectors for Word Representation) capture both semantic and syntactic relationships between words. GloVe Gigaword provided 100 dimensional embeddings from 400,000 tokens vocabulary
- Post GloVe, I experimented with BERT embeddings. These are contextual word representations generated using a transformer-based model that considers the entire sentence context. BERT has a vocabulary size of 30,522 tokens, and a maximum input limit of 512 tokens

Deep Learning Models explored in this project

- **RNNs:** Processed sequential data but suffered from the vanishing gradient problem,
- **CNNs:** Using kernel's sliding capabilities for capturing sequential hierarchies and n-grams
- **LSTMs**: RNNs with memory cells and gating mechanisms, capable of capturing long-term sequential dependencies
- **GRUs**: Captured long-term dependencies and requiring fewer parameters than LSTMs
  (Combinations of CNNs, LSTMs and GRUs were also explored)
- **BERT Transformer**: Bidirectional Encoder Representations from Transformers, a transformer-based model pre-trained on large-scale text data, capable of capturing contextual information

Proposed Approaches

Loss Function

A custom loss function called **MCRMSE (Mean Column-wise Root Mean Squared Error)** was devised. This loss function is used to calculate the mean column-wise root mean squared error between the true values (y_true) and predicted values (y_pred) for 6 target metrics.

```
def mcrmse(y_true, y_pred):
    colwise_mse = tf.reduce_mean(tf.square(y_true - y_pred), axis=0)
    return tf.reduce_mean(tf.sqrt(colwise_mse), axis=0)
```

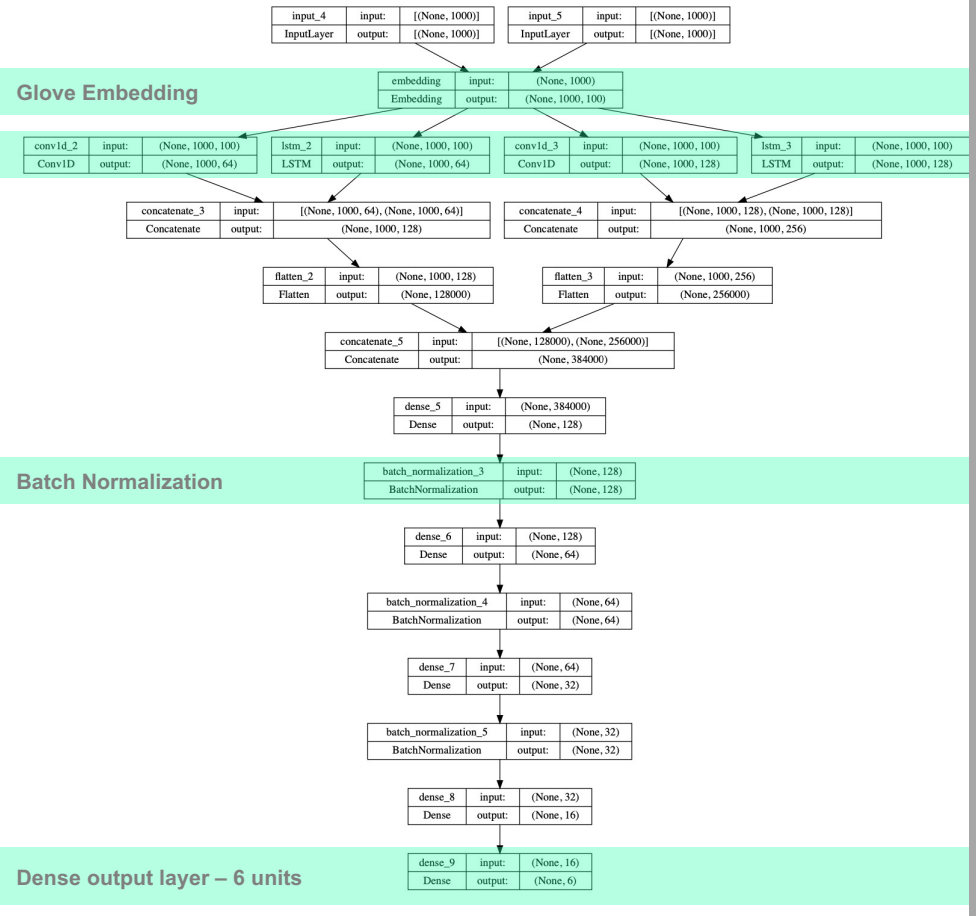
Regularization

- **Dropout:** The regular Dropout layer applies dropout to the individual elements in the input tensor. It randomly sets a fraction of input units to 0 during training
- **Spatial Dropout 1D:** Specifically designed for 1D input data, such as texts. It applies dropout not only to individual elements but also across the temporal dimension (along the sequence length). Instead of setting individual elements to 0, it sets entire timesteps (along the sequence length) to 0.

Model Performance

- **Batch Normalization:** Normalized the activations of each layer by subtracting the batch mean and dividing by the batch standard deviation, thereby reducing internal covariate shift and accelerating training and convergence
- **Reduce LR On Plateau:** Used this callback to reduce the learning rate when a metric monitored during training stopped moving (wait for a certain number of 'patience' iterations)
- **Early Stopping:** Used this callback to stop the training process early if a monitored metric has stopped moving (wait for a certain number of 'patience' iterations)
- **Compiler:** ADAM was primarily used

Model Selection: CNN+LSTM Architecture




```python
def LSTM_CNN1D_3():
    drop_lstm = 0.25
    drop_dense = 0.25
    num_lstm=150

    input_1 = Input(shape=(sequence_length,))
    embedding_layer_1 = embedding_layer(input_1)
    conv_1_1 = Convolution1D(64,3,strides=1, padding='same',activation='relu')(embedding_layer_1)
    lstm_1_1 = LSTM(64,dropout=drop_lstm,return_sequences=True,dtype=tf.float32)(embedding_layer_1)
    concate_1 = keras.layers.Concatenate(axis=-1)([conv_1_1, lstm_1_1])
    flatten_1 = Flatten()(concate_1)

    input_2 = Input(shape=(sequence_length,))
    embedding_layer_2 = embedding_layer(input_2)
    conv_1_1 = Convolution1D(128,3,strides=1, padding='same',activation='relu')(embedding_layer_2)
    lstm_1_2 = LSTM(128,dropout=drop_lstm,return_sequences=True,dtype=tf.float32)(embedding_layer_2)
    concate_2 = keras.layers.Concatenate(axis=-1)([conv_1_1, lstm_1_2])
    flatten_2 = Flatten()(concate_2)

    concatenated_layer = keras.layers.concatenate([flatten_1,flatten_2],axis=-1)

    # creating further layers
    x = Dense(128, activation = 'relu')(concatenated_layer)
    x = BatchNormalization()(x)
    x = Dense(64, activation = 'relu')(x)
    x = BatchNormalization()(x)
    x = Dense(32, activation = 'relu')(x)
    x = BatchNormalization()(x)
    x = Dense(16, activation = 'relu')(x)
    output = Dense(6,activation='linear')(x)

    model = Model(inputs = [input_1,input_2], outputs = [output])
    optimizer=Adam()
    model.compile(optimizer=optimizer, loss = mcrmse, metrics = mcrmse)
    return model
```

- Number of Epochs: 30
- Batch Size: 64 essays
- Processing time: ~3000 seconds
- Validation Split: 20%
- Text was fed into the model through two inputs, both going through 2 different LSTM_+ CNN combinations (different number of units and kernels)
- The outputs were then flattened and concatenated before passing through a series of 5 dense and 3 batch normalization layers

Model Selection: BiGRU

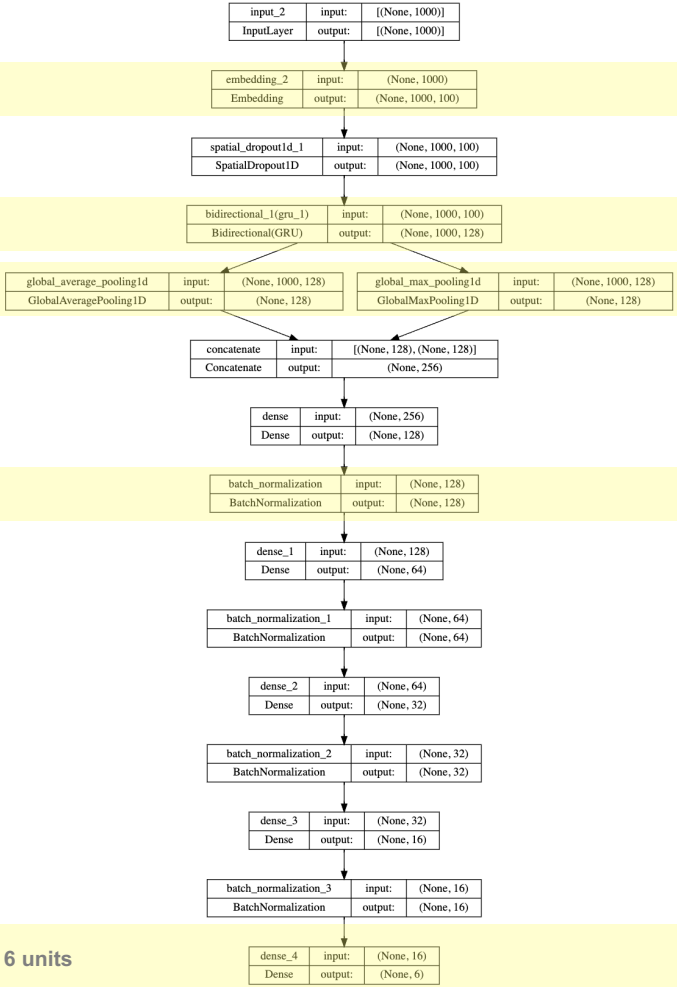
Glove Embedding

BiGRU Layer

Avg + Max Pooling

Batch Normalization

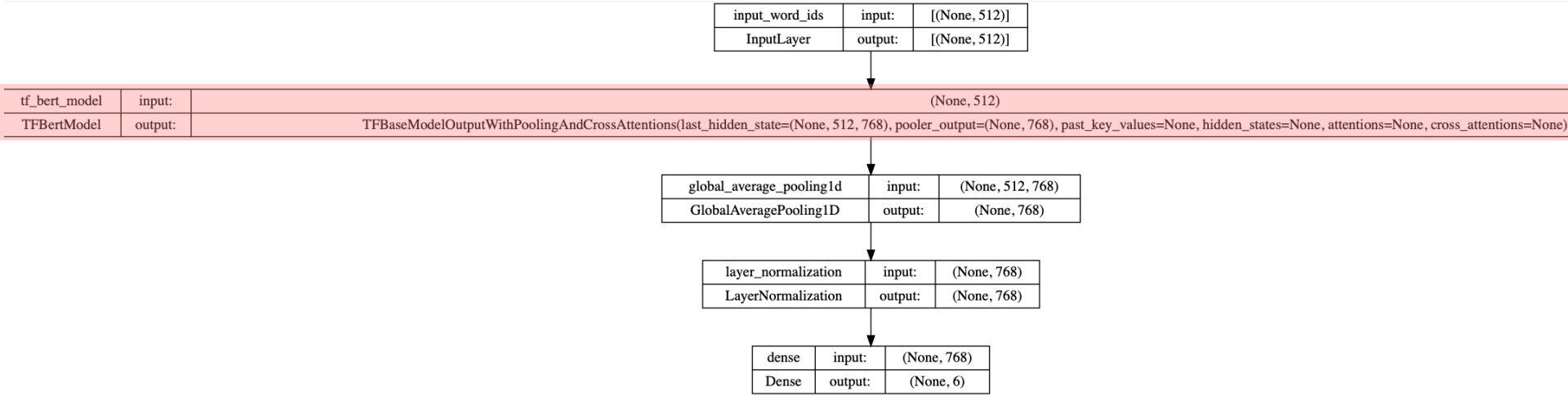
Dense output layer – 6 units




```
inputs = Input(shape=(sequence_length, ))

x = Embedding(num_tokens, embedding_dim, weights=[embedding_matrix])(inputs)
x = SpatialDropout1D(0.2)(x)
x = Bidirectional(GRU(gru_dims, return_sequences=True))(x)
flatten_1 = Flatten()(x)

# Create a concatenation of GlobalAveragePooling1D and GlobalMaxPooling1D layers
avg_pool = GlobalAveragePooling1D()(x)
max_pool = GlobalMaxPooling1D()(x)
concatenated = concatenate([avg_pool, max_pool])
x = Dense(128, activation = 'relu')(concatenated)
x = BatchNormalization()(x)
x = Dense(64, activation = 'relu')(x)
x = BatchNormalization()(x)
x = Dense(32, activation = 'relu')(x)
x = BatchNormalization()(x)
x = Dense(16, activation = 'relu')(x)
x = BatchNormalization()(x)
outputs = Dense(6, activation = 'linear')(x)

model_gru_1 = Model(inputs=inputs, outputs=outputs)
model_gru_1.compile(loss = mcrmse, optimizer='adam', metrics = mcrmse)
```

- Number of Epochs: 40
- Batch Size: 64 essays
- Processing time: ~4000 seconds
- Validation Split: 20%
- Outputs from the BiGRU layer were both average and max pooled, and later concatenated
- The outputs were then processed through a series of 4 dense and batch normalization layers, ultimately ending with a dense layer with 6 units (for 6 outputs)

Model Selection:
BERT Encoder



- Number of Epochs: 4
- Batch Size: 10 essays
- Processing time: 17170 seconds
- Validation Data: 20% removed from training data
- Essays were limited to 512 tokens to comply with BERT's token limit
- Last layer of BERT encoder yields 768 dimensions. This was further pooled and normalized before feeding to a dense function

```
def create_model_BERT1():
    bert_encoder = TFBertModel.from_pretrained(bert_path)
    input_word_ids = tf.keras.Input(shape=(MAX_LEN,), dtype=tf.int32, name="input_word_ids")

    embedding = bert_encoder(input_word_ids)[0]

    x = tf.keras.layers.GlobalAveragePooling1D()(embedding)
    x = tf.keras.layers.LayerNormalization()(x)

    #Output layer without activation function because regression task
    output = tf.keras.layers.Dense(6,)(x)

    model = tf.keras.models.Model(inputs=input_word_ids, outputs=output)
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), loss=MCRMSE
                  , metrics=MCRMSE)

    return model
```

Results

Approximately 15 models were experimented upon with 3 architecture schemas. The final models from them yielded the following results:

Embedding Model	Model	Train MCRMSE	Validation MCRMSE	Test MCRMSE
Glove Embedding	CNN + LSTM	0.27	0.62	~0.62
Glove Embedding	BiGRU	0.51	0.59	~0.59
BERT Embedding	BERT	0.51	0.53	0.53

- **Even though Train MCRMSE is the least for a CNN+LSTM model, it is not generalizing well with Validation and Test datasets. This means that our hypothesis that kernel filters can be used for gauging n-grams and sequential dependencies is rejected**
- **Both BERT and BiGRU gave better generalizing models, with a simple pre-trained BERT architecture showing the highest Validation MCRMSE**
- **However, BERT had significantly high training times on a 16CPU, 60GB system without GPUs**

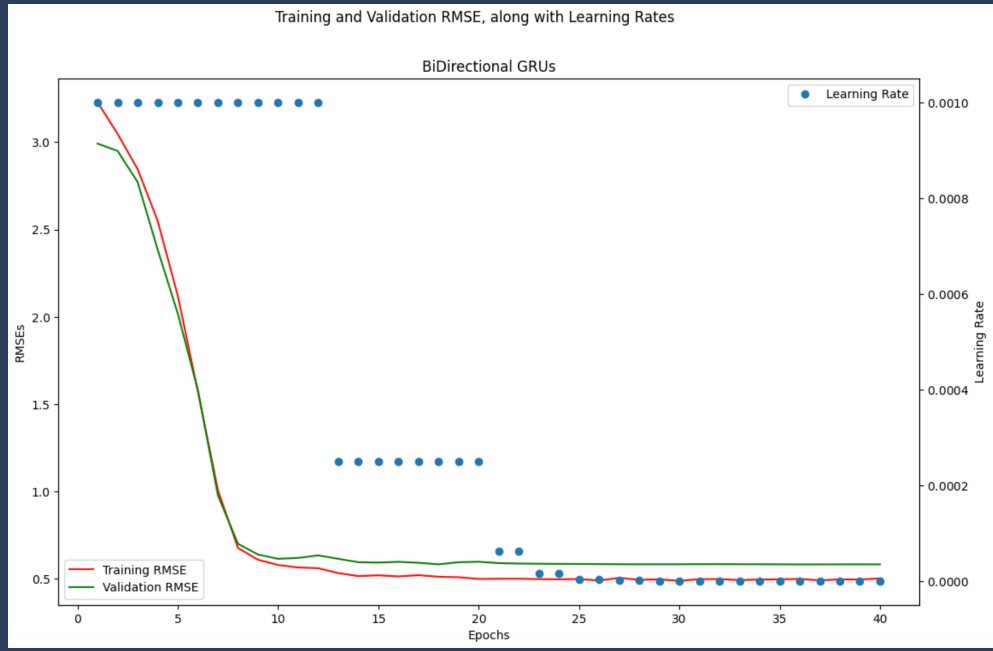
Bi-GRUs: A way forward (and backwards)

Even though BERT transformer yielded the best results, Bi-GRUs were impressive in providing low test RMSE at a fraction of computational cost.

While transformers have gained significant popularity, Bi-GRUs offer a compelling alternative in specific scenarios where **efficiency, sequential processing, contextual understanding, training time, or interpretability are important considerations**

- **Bi-GRUs have fewer parameters compared to transformers, making them more memory and computationally efficient**, especially for smaller datasets or constrained environments.

- **Bi-GRUs are inherently more interpretable than transformers since they operate on the sequential input and explicitly model the dependencies between the past and future states**. This can aid in understanding the inner workings of the model and diagnosing potential issues.



Next Steps

Modeling

- Exploring more transformer models that can yield similar results, but with lesser computational times (example: DistilBert, GPT2 etc)

- Due to the multi-layer structure of Transformers, different layers capture different levels of representations. They learn a rich hierarchy of linguistic information i.e. with surface features in lower layers, syntactic features in middle layers, and semantic features in higher layers. Experimenting with intermediate representations from various layers can help in understanding more model nuances

Deployment

- Building an interactive tool powered by a Bi-GRU/Transformer model we obtained from our training data. This tool will be built for ELLs, and can provide many learning features that can help them improve their writing

Thank You!

Follow this project on GitHub

https://github.com/kshitij-mittal/automatic-essay-grading-DL