Recurrent Neural Network

Tan Wen Tao Bryan
2214449
DAAA/FT/2B/01

Background Research

Background Info

Next-Word Prediction

- A language modelling task in Machine Learning that aims to predict the most probable word or sequence of words that follows a given input context.
- Utilizes statistical patterns and linguistic structures to generate accurate predictions based on the context provided.
- Requires models like LSTM,GRU or even Bi-Directional models (More Info Below)

Recurrent Neural Networks

- Neural networks with a hidden state that evolves over time, making them powerful for tasks involving sequences like natural language processing, speech recognition and time series prediction.
- Perform same task for every element of a sequence, with the output being depended on the previous computations
- RNN is like having a "memory" which captures information about what has been calculated so far
- Make use of information in arbitrarily long sequences, but they are limited to looking back only a few steps
- Image below shows that the RNN being unrolled into a full network, unrolling means the network can be unrolled into a 5-layer neural network, one layer for each word.
- Key Components: (More Info Below)
  - Hidden State
  - Recurrence
  - Shared Weights

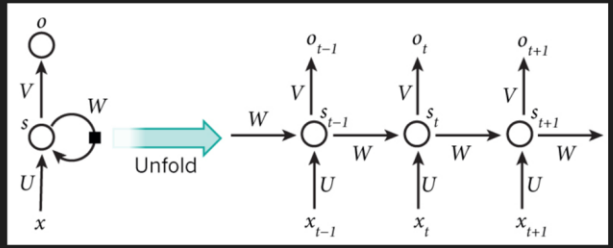


Image Source: Prashant Banerjee, 2019

Step 1: EDA

Descriptive Statistics

```
# Description of the dataset
display(data.describe(include='all'))

print(f'Shape of the Dataset: {data.shape}')
print(f"Number of Missing Values: {data['Quotes'].isnull().sum()}")

unique_quotes = data['Quotes'].unique()
print(f"Number of Unique Quotes: {len(unique_quotes)}")
```

	Quotes
count	1000
unique	890
top	Radiate acceptance, and find peace in embracing what is.
freq	5

```
Shape of the Dataset: (1000, 1)
Number of Missing Values: 0
Number of Unique Quotes: 890
```

Observations

- The dataset contains 1000 quotes. (1000, 890)
- There are 890 unique quotes, indicating some repetition.
- There are no missing values in the dataset since it is a text file with one column.

Removing Duplicated Phrases

- Removing duplicated phrases will help reduce the biasness when training.

```
# Remove duplicate quotes
data.drop_duplicates(subset='Quotes', keep='first', inplace=True)

print(f"Shape of dataset after removing duplicates: {data.shape}")
```

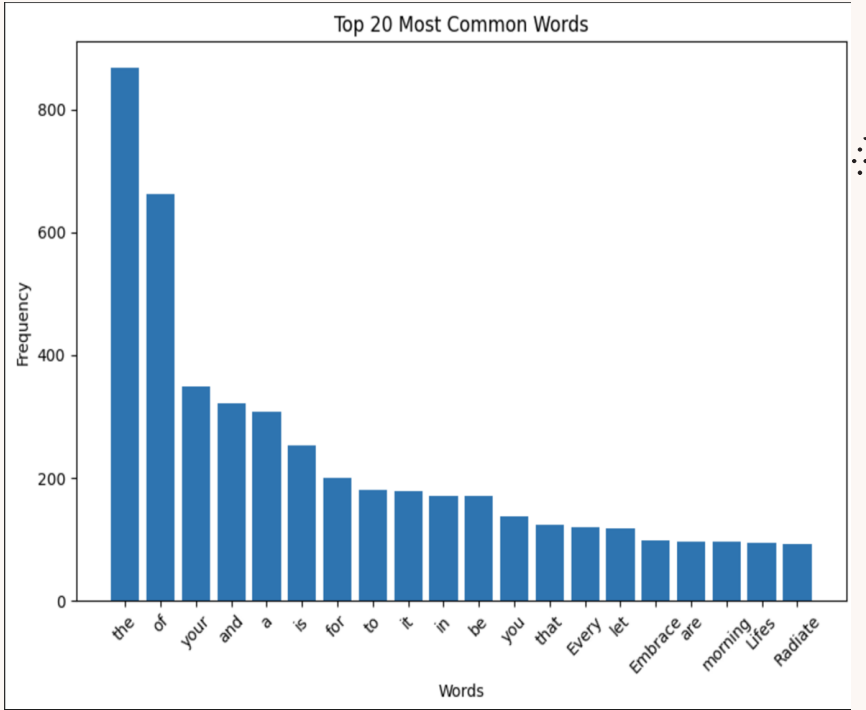
```
Shape of dataset after removing duplicates: (890, 1)
```

Step 1: EDA

Word Frequency Analysis

- Finding out how frequent the words occur in the dataset

```
# Concatenate all the quotes into a single string
all_words = " ".join(data['Quotes'])
# Remove any punctuations
all_words = all_words.translate(str.maketrans("", "", string.punctuation))

# Split the string into words
all_words = all_words.split()

# Count the frequency of each word
word_count = Counter(all_words)

# Get the 20 most common words
most_common_words = word_count.most_common(20)

# Prepare data for plotting
words, counts = zip(*most_common_words)

plt.figure(figsize=(8, 6))
plt.bar(words, counts)
plt.title('Top 20 Most Common Words')
plt.xticks(rotation=45)
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Observations:

- We can see that the first few most frequent words are just common English words, including some stop words like 'the', 'of', 'and', 'a'.
- Since the dataset look like life quotes, words like 'let' and 'Every' are quite common in quotes like that.

Step 1: EDA

Unique Words

- Find number of unique words and find rate of uniqueness out of the total words

```python
# Number of unique words
print(f"Number of Unique Words: {len(word_count)}")

# Find total amount of words
total_words = len(all_words)
print(f"Total Number of Words: {total_words}")

# Find the percentage of unique words
unique_words = len(word_count) / total_words * 100
print(f"Percentage of Unique Words: {unique_words:.2f}%")
```

Py

```
Number of Unique Words: 1220
Total Number of Words: 10600
Percentage of Unique Words: 11.51%
```

Observations

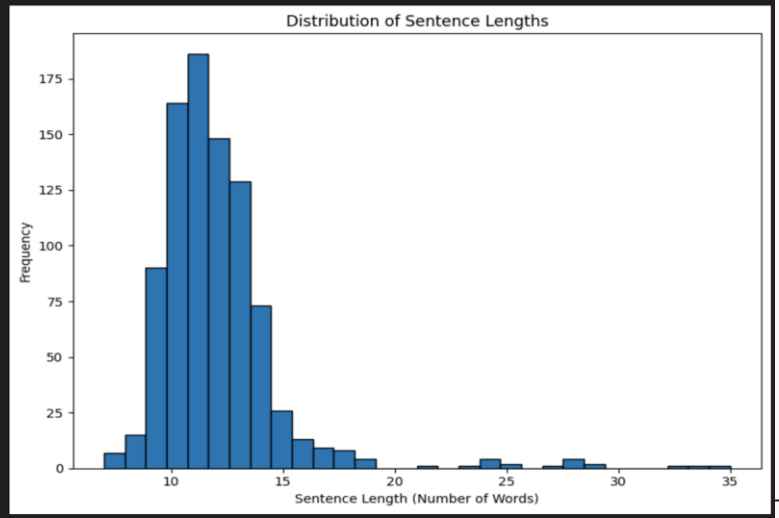
- The unique words make up to approximately 11.5% of the total words which means approximately 88.5% of the words in the dataset are repeated.
- For text generation tasks, a larger vocabulary is actually needed to handle the diversity of words that the model can predict.

Step 1: EDA

Sentence Length Analysis

- Find out how long each quote is roughly.

```python
# Analyse the distribution of sentence lengths
sentence_lengths = data["Quotes"].str.split().str.len()

# Plot the distribution of sentence lengths
plt.figure(figsize=(8,6))
plt.hist(sentence_lengths, bins=30, edgecolor='black')
plt.title('Distribution of Sentence Lengths')
plt.xlabel('Sentence Length (Number of Words)')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Observations

- Histogram is skewed to the left, clustering around 10-15 words each which shows that shorter sentence are more frequent.
- We will need to ensure that the sentence lengths are equal before fitting to the models.

Word Cloud

- A graphical representation of text data, where words from a text document are displayed in various sizes, with the most frequently occuring words appearing larger.
- Often used for visualizing and gaining insights from text data, such as identifying key terms in a document, website or social media content.

```python
# Generate a word cloud
wordcloud = WordCloud(
    width =800, height=400, background_color='white'
).generate(' '.join(all_words))

plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Python

Observations

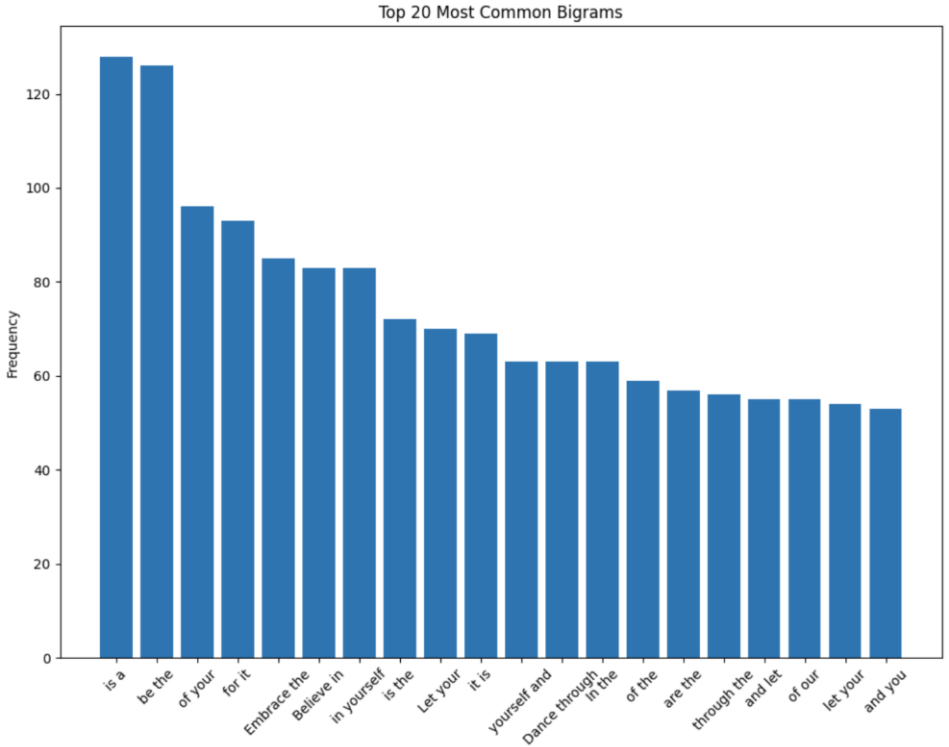
- According to the word cloud, excluding the stop words, words liked let and life are more common than the other words.

Step 1: EDA

Bigram Analysis

- Phrases of two words were checked to see which is the most frequent.

```python
def generate_bigrams(words):
    return zip(words, words[1:])

# Generate bigrams from list of words
bigram_list = list(generate_bigrams(all_words))

# Count the frequency of each bigram
bigram_freq = Counter(bigram_list)

# Get the 20 most common bigrams
most_common_bigrams = bigram_freq.most_common(20)

# Prepare data for plotting
bigram_words, bigram_counts = zip(*most_common_bigrams)
bigram_words = [" ".join(bigram) for bigram in bigram_words]

plt.figure(figsize=(10, 8))
plt.bar(bigram_words, bigram_counts)
plt.title('Top 20 Most Common Bigrams')
plt.xticks(rotation=45)
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



Observations

- Most of the common bigrams like 'be the', 'through the', 'In the' includes the word 'the' which are commonly used expressions.

Step 2: Feature Engineering

2) Feature Engineering

```python
# Convert the quotes into the list
data_list = list(data["Quotes"].values)
print(data_list)
```

Python

```
["Embrace the beauty of every sunrise; it's a fresh chance to paint your world with joy.", 'Embrace challenges; they are the step
```

Input-Output Pairing

- Associate the input data with the corresponding output data during training
- Increase the dataset volume
- Minimise the loss of predicted and actual outputs during training

Ensure that **every word** in the sequence get to be the **starting character** of the sequence

EXAMPLE:

Expanding Window Tokenization

- Based off time series expanding window
- Create overlapping sequences of tokens from the original text data, allowing the model to predict the next word based on a context window of preceding words
- Approach is different from typical input-output pairing, where each treated as an independent sequence
- Texts to sequences to converts word into a list of integers

```python
# Demonstrate how to use the tokenizer
text_data = ["hello world my name is"]

tokenizer =Tokenizer()
tokenizer.fit_on_texts(text_data)

#Texts to sequences helps
sequences = tokenizer.texts_to_sequences(text_data)

# Create rolling window sequences
for window_size in range(1, len(text_data[0].split())-1):
    rolling_sequences = [
        sequences[0][i:i+window_size+1] for sequence in sequences
        for i in range(len(sequence)-window_size)
    ]

    print(f"Window Size of {window_size}: {rolling_sequences}")
```

```
Window Size of 1: [[1, 2], [2, 3], [3, 4], [4, 5]]
Window Size of 2: [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
Window Size of 3: [[1, 2, 3, 4], [2, 3, 4, 5]]
```

Step 2: Feature Engineering

```python
# Perform it on the actual dataset
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data_list)

all_rolling_sequences = []

for text in data_list:
    sequences = tokenizer.texts_to_sequences([text])[0]

    # Create rolling window sequences for different window size
    rolling_sequence_list = []

    for window_size in range(1, len(sequences)-1):
        rolling_sequences = [
            sequences[i:i+window_size+1] for i in range(len(sequences)-window_size)
        ]
        rolling_sequence_list.append(rolling_sequences)

        if text == data_list[0]:
            print(f"Window Size of {window_size}: {rolling_sequences}")

    # Combine rolling window sequences of different window size
    all_rolling_sequences.extend([item for sublist in rolling_sequence_list for item in sublist])
```

Python

Tokenization

```
Window Size of 1: [[17, 1], [1, 49], [49, 2], [2, 13], [13, 77], [77, 372], [372, 5], [5, 163], [163, 486], [486, 10], [10, 101],
Window Size of 2: [[17, 1, 49], [1, 49, 2], [49, 2, 13], [2, 13, 77], [13, 77, 372], [77, 372, 5], [372, 5, 163], [5, 163, 486],
Window Size of 3: [[17, 1, 49, 2], [1, 49, 2, 13], [49, 2, 13, 77], [2, 13, 77, 372], [13, 77, 372, 5], [77, 372, 5, 163], [372,
Window Size of 4: [[17, 1, 49, 2, 13], [1, 49, 2, 13, 77], [49, 2, 13, 77, 372], [2, 13, 77, 372, 5], [13, 77, 372, 5, 163], [77,
Window Size of 5: [[17, 1, 49, 2, 13, 77], [1, 49, 2, 13, 77, 372], [49, 2, 13, 77, 372, 5], [2, 13, 77, 372, 5, 163], [13, 77, 3
Window Size of 6: [[17, 1, 49, 2, 13, 77, 372], [1, 49, 2, 13, 77, 372, 5], [49, 2, 13, 77, 372, 5, 163], [2, 13, 77, 372, 5, 163
Window Size of 7: [[17, 1, 49, 2, 13, 77, 372, 5], [1, 49, 2, 13, 77, 372, 5, 163], [49, 2, 13, 77, 372, 5, 163, 486], [2, 13, 77
Window Size of 8: [[17, 1, 49, 2, 13, 77, 372, 5, 163], [1, 49, 2, 13, 77, 372, 5, 163, 486], [49, 2, 13, 77, 372, 5, 163, 486, 1
Window Size of 9: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10], [49, 2, 13, 77, 372, 5, 16
Window Size of 10: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486, 10], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101], [49, 2, 13, 77,
Window Size of 11: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3], [49, 2,
Window Size of 12: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3, 46], [
Window Size of 13: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3, 46], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3, 46
Window Size of 14: [[17, 1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3, 46, 22], [1, 49, 2, 13, 77, 372, 5, 163, 486, 10, 101, 3
```

Padding

- Adding special tokens of 0 to make input sequences of consistent length, up to the length of the longest sequence in the dataset
- Necessary for training and processing of neural networks

```python
# Pad the combined sequences
max_sequence_rolling_len = max([len(x) for x in all_rolling_sequences])
X_padded = pad_sequences(
    [window[:-1] for window in all_rolling_sequences],
    maxlen = max_sequence_rolling_len,
    padding='pre'
    )
y = [window[-1] for window in all_rolling_sequences]

y_categorical = to_categorical(y, num_classes=len(tokenizer.word_index)+1)

total_words_rolling = len(tokenizer.word_index) + 1 # index 0 is reserved for padding
print(f"Total number of words: {total_words_rolling}")
```

Python

Padding

```
Total number of words: 1199
```

Step 2: Feature Engineering

Splitting Train, Validation & Test

- Dataset is split into train, validation and testing: 60%, 20% and 20%

```
# Split the rolling window dataset into training, validation and test sets
X_train_roll, X_test_roll, y_train_roll, y_test_roll = train_test_split(
    X_padded, y_categorical , test_size =0.2, random_state=42, shuffle=True
    )
X_train_roll, X_val_roll, y_train_roll, y_val_roll = train_test_split(
    X_train_roll, y_train_roll, test_size=0.25, random_state=42, shuffle=True
    )

print(f"X_train: {X_train_roll.shape}")
print(f"y_train: {y_train_roll.shape}")
print(f"X_val: {X_val_roll.shape}")
print(f"y_val: {y_val_roll.shape}")
print(f"X_test: {X_test_roll.shape}")
print(f"y_test: {y_test_roll.shape}")
```

```
X_train: (36707, 34)
y_train: (36707, 1199)
X_val: (12236, 34)
y_val: (12236, 1199)
X_test: (12236, 34)
y_test: (12236, 1199)
```

Step 3: Model Selection

3) Model Selection/Model Evaluation

Models List:

- RNN with SimpleRNN layer
- RNN with LSTM layer
- RNN with GRU layer
- Bi-Directional LSTM
- Bi-Directional GRU

```python
# Plot accuracy_curve
def plot_learning_curve(history):
    history_df = pd.DataFrame(history)
    epochs = list(range(1,len(history_df)+1))

    fig, ax = plt.subplots(1,2, figsize=(16,6))

    # Training loss and validation loss
    ax1=ax[0]
    ax1.plot(epochs, history_df["loss"], label="Training Loss")
    ax1.plot(epochs, history_df["val_loss"], label="Validation Loss")
    ax1.legend()
    ax1.set_ylabel("Loss")
    ax1.set_xlabel("Number of Epochs")
    ax1.set_title("Training and Validation Loss")

    # Training accuracy and validation accuracy
    ax2=ax[1]
    ax2.plot(epochs, history_df["accuracy"], label="Training Accuracy")
    ax2.plot(epochs, history_df["val_accuracy"], label="Validation Accuracy")
    ax2.legend()
    ax2.set_ylabel("Accuracy")
    ax2.set_xlabel("Number of Epochs")
    ax2.set_title("Training and Validation Accuracy")
    plt.show()
```

Python

Step 3: SimpleRNN

Simple RNN (Recurrent Neural Network)

- Starts with the Embedding layer which turn the integers in the sequences into fixed-length vectors
- Simple RNN has these key components:
  - Hidden State - At each time step, an RNN maintains a hidden state that is updated based on the current input and the previous hidden state. Hidden state serves as a memory to capture information about the sequence so far.
  - Recurrence - Derived from the connections that allow information to be passed from one step of the sequence to the next. Enables the network to consider the context of previous inputs when processing the current input.
  - Shared Weights - RNNs typically share the same set of weights across all time steps. Weights allow the model to learn patterns and relationships that are consistent across the entire sequence.
- Tend to suffer from the vanishing gradient problem, making it challenging to train on long sequence, unlike LSTM or GRU which are more effective in capturing long-range dependencies in sequence
- Usually uses tanh as it has to do with its second derivative decaying very slowly to zero.

Important Parameters:

- return_sequences = True: Output shape is (batch_size, timesteps, units)
- return_sequences = False: Output shape is (batch_size, units)

Simple RNN Version 1

Baseline Model

- Baseline Model
- Start off with 1 SimpleRNN layer with 64 units

```
tf.keras.backend.clear_session()

# Create the model
simpleRNN = Sequential(
    name='simpleRNN_v1',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        SimpleRNN(64, activation='tanh'),

        Dropout(0.4),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
simpleRNN.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

simpleRNN_history = simpleRNN.fit(
    X_train_roll, y_train_roll,
    epochs=100,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

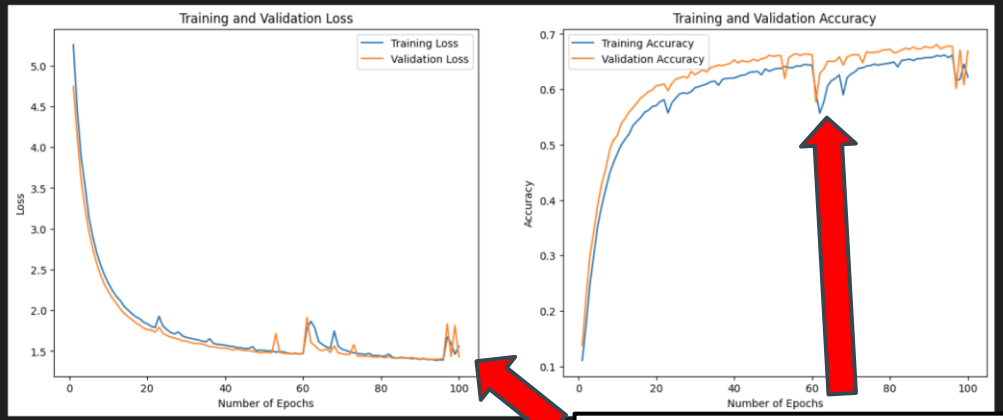
```
simpleRNN.summary()
```

```
Model: "simpleRNN_v1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 simple_rnn (SimpleRNN)      (None, 64)                4800

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 1199)              77935

=================================================================
Total params: 94,725
Trainable params: 94,725
Non-trainable params: 0
_________________________________________________________________
```

Step 3: SimpleRNN

```
plot_learning_curve(simpleRNN_history.history)
```

Python



Constant dip in accuracy and **sharp rise in loss** means the model is unable to **converge smoothly**, might be due to **learning rate** being **too high**

Observations

- The training and validation loss curve shows constant fluctuations in the curve, same for its accuracy which suggest this model is not a good one to go about.

```
simpleRNN.evaluate(X_test_roll, y_test_roll)
```

Python

```
383/383 [==============================] - 2s 4ms/step - loss: 1.4270 - accuracy: 0.6719

[1.4270331859588623, 0.671869740005493]
```

Observations

- Accuracy of 0.6719 is considered decent for prediction but we need to explore other models.

Step 3: LSTM

LSTM (Long-Short Term Memory)

- Designed to avoid the long-term dependency problem.
- e.g. The man who ate my pizza has purple hair. Purple hair is for the man and not the pizza.
- Overcome the constraints of regular RNNs which frequently struggle with vanishing gradient problem.
- Has a hidden state and a memory cell with three gates thate are forgotten, input and output gate.

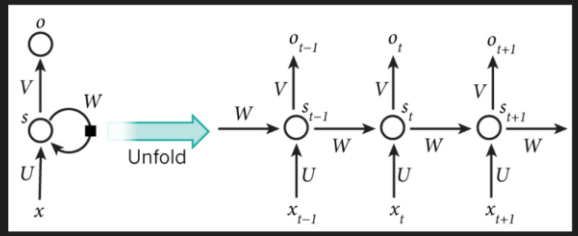


Image Source: Prashant Banerjee, 2019

- The forget gate is mainly used to get good control of what information needs to be removed which isn't necessary.
- Input gate makes sure that newer information is added to the cell and output makes sure what parts of the cell are output to the next hidden state.
- Sigmoid function used in each gate equation helps being down the value to either 0 or 1.

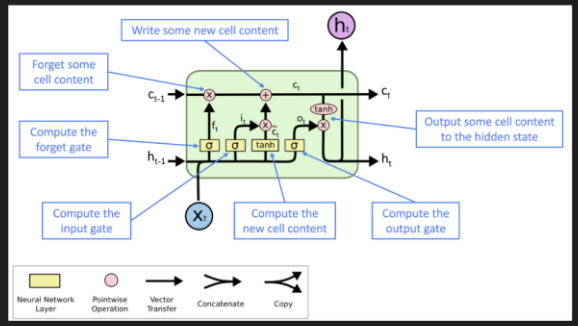


Image Source: Siddharth, M., 2021

LSTM Version 2

- LSTM with 64 units
- Dropout of 0.3

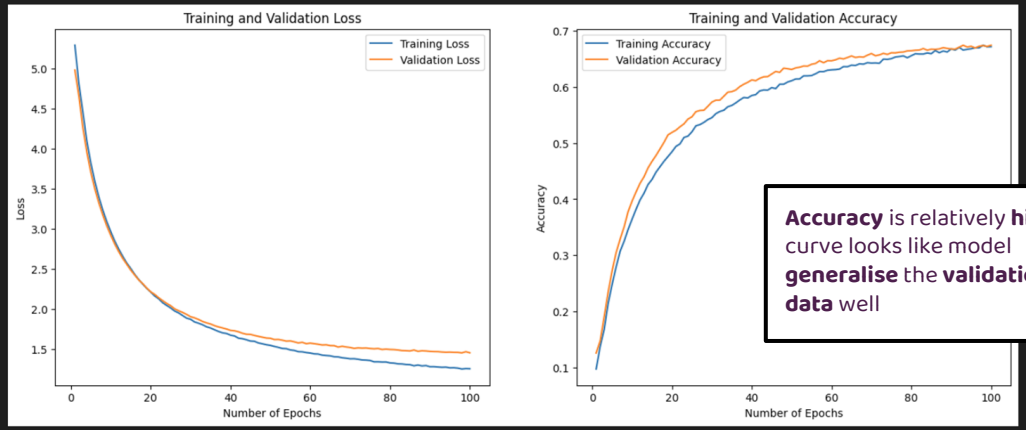
```
tf.keras.backend.clear_session()

# Create the model
LSTM_V2 = Sequential(
    name='lstm_v2',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        LSTM(64, activation='tanh'),

        Dropout(0.3),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
LSTM_V2.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

LSTM_V2_history = LSTM_V2.fit(
    X_train_roll, y_train_roll,
    epochs=100,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)

LSTM_V2.summary()
```

```
Model: "lstm_v2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 lstm (LSTM)                 (None, 64)                19200

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 1199)              77935

=================================================================
Total params: 109,125
Trainable params: 109,125
Non-trainable params: 0
_________________________________________________________________
```

Step 3: LSTM

```
plot_learning_curve(LSTM_V2_history.history)
```

Python



Accuracy is relatively **high**, curve looks like model **generalise** the **validation data** well

Observations

- The curve actually looks quite good, the training and validation curve are converging into a single point and are able to reach quite a high accuracy.

```
LSTM_V2.evaluate(X_test_roll, y_test_roll)
```

Python

```
383/383 [==============================] - 1s 2ms/step - loss: 1.4599 - accuracy: 0.6816

[1.4599246978759766, 0.6815952658653259]
```

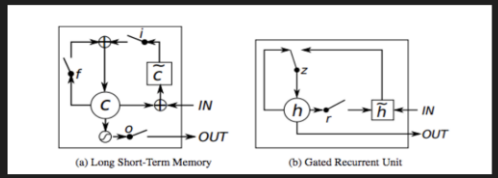
Observations

- As mentioend, the test accuracy of 0.68 is quite high, which shows that this model is good

Step 3: GRU

GRU (Gated Recurrent Unit)

- Retains the LSTM vanishing gradient problem
- Faster than LSTM since fewer computations are needed to make updates to its hidden state
- Fewer gates than LSTM and no separate cell state as it relies solely on a hidden state for memory transfer between recurrent units.
- Has only two gates:
  - Reset gate - Determines how much of past information should be forgotten. Takes the concatenation of current input and previous hidden state as input and outputs a value between 0 and 1 for each element in the hidden state.
  - Update gate - Decides how much of the new information should be included in the new hidden state. Also takes the concatenation of current input and previous hidden state as input and outputs a value between 0 and 1 for each element in the hidden state.



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Image Source: Dishashree, G., 2020

GRU Version 2

- GRU with 256 units
- Dropout of 0.3

```
tf.keras.backend.clear_session()

# Create the model
GRU_V2 = Sequential(
    name='gru_v2',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        GRU(256, activation='tanh'),

        Dropout(0.3),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
GRU_V2.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

GRU_V2_history = GRU_V2.fit(
    X_train_roll, y_train_roll,
    epochs=100,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

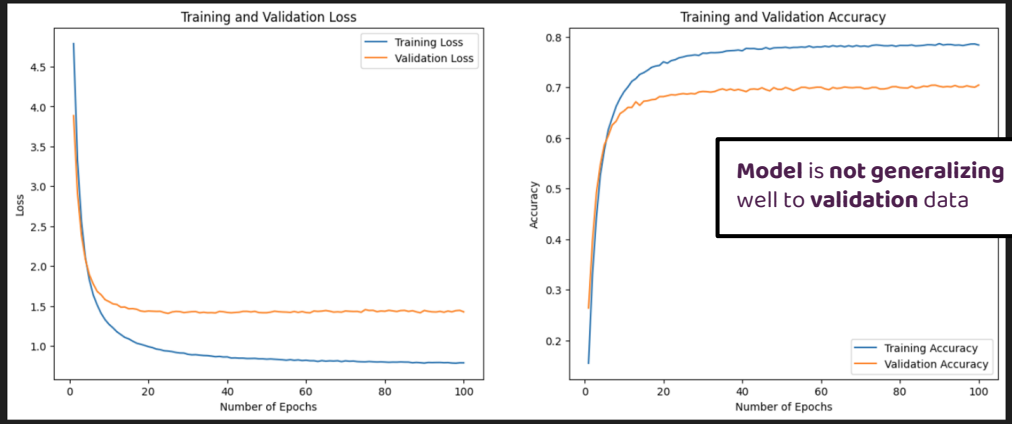
```
GRU_V2.summary()
```

```
Model: "gru_v2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 gru (GRU)                   (None, 256)               205824

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 1199)              308143

=================================================================
Total params: 525,957
Trainable params: 525,957
Non-trainable params: 0
_________________________________________________________________
```

Step 3: GRU

```
plot_learning_curve(GRU_V2_history.history)
```

Python



Model is **not generalizing** well to **validation** data

Observations

- The validation loss is much higher than the training loss while the validation accuracy is much lower than the training accuracy.

```
GRU_V2.evaluate(X_test_roll, y_test_roll)
```

Python

```
383/383 [==============================] - 1s 2ms/step - loss: 1.4258 - accuracy: 0.7071

[1.4258129596710205, 0.7070938348770142]
```

Observations

- This model shows the highest test accuracy so far for GRU.

Step 3: Bi-Directional LSTM/GRU

Bi-Directional LSTM

- In traditional LSTM, the input sequence is processed only in forward direction, starting from beginning to end.
- In Bidirectional LSTM, the input sequence is processed both in forward direction (from the beginning to end) and in the backward direction (from end to beginning)
- Hidden states of Bidirectional LSTM are computed by concatenating the hidden states from forward and backward passes at each time step, hence containing information from both directions
- Example:
  - First Statement: Server can you bring the dish.
  - Second Statement: He crashed the server.
  - The word server has different meanings and this relationship depends on the following and preceding words in the statement.
  - This relationship is understood by bidirectional LSTM better than unidirectional LSTM.

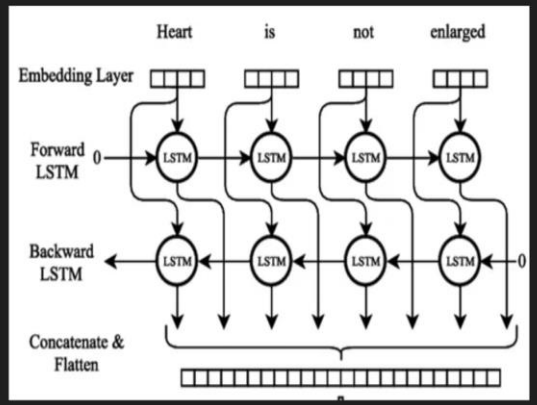


Image Source: Kumari, K., 2023

Also tried **Bi-Directional GRU**, and **many other architectures**, can refer to **Jupyter Notebook**

Bi Directional LSTM Version 4

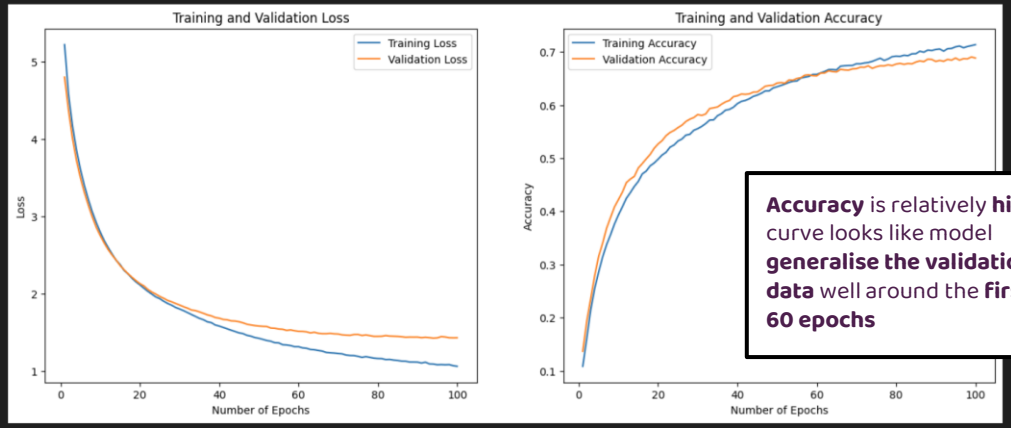
- Bidirectional LSTM of 64 units
- Dropout of 0.3

```
tf.keras.backend.clear_session()

# Create the model
Bi_LSTM_V4 = Sequential(
    name='bi_directional_lstm_v4',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        Bidirectional(LSTM(64, activation='tanh')),

        Dropout(0.3),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
Bi_LSTM_V4.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

Bi_LSTM_V4_history = Bi_LSTM_V4.fit(
    X_train_roll, y_train_roll,
    epochs=100,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

```
Model: "bi_directional_lstm_v4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 bidirectional (Bidirectiona  (None, 128)              38400
 l)

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 1199)              154671

=================================================================
Total params: 205,061
Trainable params: 205,061
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Bi-Directional LSTM/GRU

```
plot_learning_curve(Bi_LSTM_V4_history.history)
```



Accuracy is relatively **high**, curve looks like model **generalise the validation data** well around the **first 60 epochs**

Observations

- The curve seems to converge quicker than usual, looks like this is quite a good model.

```
Bi_LSTM_V4.evaluate(X_test_roll, y_test_roll)
```

```
383/383 [==============================] - 1s 3ms/step - loss: 1.4263 - accuracy: 0.6929

[1.4263392686843872, 0.692873477935791]
```

Observations

- This model has a decently high test accuracy.

Step 3: Model Evaluation

Evaluation Metrics

Corpus BLEU (Bilingual Evaluation Understudy) Score:

- Commonly used for evaluating the quality of machine-generated text
- Measures how well a generated text aligns with one or more referenced texts, to serve as a quantitative measure of the similarity between the generated and reference texts.
- Requires one or more reference texts to compare against the generated text
- BLEU calculates precision for each n-gram and then combines them into a single score
- Includes a brevity penalty to discourage overly short generated texts (i.e. generated texts that are significantly shorter than the reference texts)
- Ranges from 0 to 1, with 1 indicating a perfect match between generated and referenced texts
- Uses differents weight for each n-gram precision:
  - Decaying Weights - Use weights that decay as the n-gram length increase
  - e.g. (0.1, 0.2, 0.3, 0.4)

1. **Calculate Precision for each n-gram:**

$$\text{Precision} = \frac{\text{Count of n-grams in model and reference}}{\text{Count of n-grams in model}}$$

2. **Calculate Brevity Penalty (BP):**

$$\text{BP} = \begin{cases} 1 & \text{if model length} \geq \text{reference length} \\ e^{\left(1 - \frac{\text{reference length}}{\text{model length}}\right)} & \text{otherwise} \end{cases}$$

3. **Calculate BLEU Score:**

$$\text{BLEU} = \text{BP} \cdot \exp\left(\frac{1}{4}\sum_{n=1}^{4} \log(\text{Precision}_n)\right)$$

Metrics used: BLEU score, BERTScore, Perplexity

BERTScore (Bidirectional Encoder Representations from Transformers)

- Uses contextual embeddings to capture meaning of words in context to evaluate quality of generated texts
- BERTScore combines traditional precision/recall based metrics with cosine similarity of word embeddings. Measures how well the candidate sentence capture meaning and context of reference sentence
- BERTScore compares the entire sentences rather than focusing solely on word-level matches.

1. **Calculate Precision, Recall, and F1 for each token:**
   - Precision:

$$P = \frac{\text{model} \cap \text{reference}}{\text{model}}$$

   - Recall:

$$R = \frac{\text{model} \cap \text{reference}}{\text{reference}}$$

   - F1:

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

2. **Calculate Weighted F1 for each token:**

$$F1_w = \frac{\sum_i \text{F1}_i \cdot \text{weight}_i}{\sum_i \text{weight}_i}$$

3. **Calculate Overall BERTScore:**

$$\text{BERTScore} = \frac{\sum_i F1_w \cdot \text{weight}_i}{\sum_i \text{weight}_i}$$

Perplexity

- A measure of how well a probability distribution or probability model predicts a sample with the actual distribution of words in the test set.
- Lower perplexity indicates better peformance.

$$\text{Perplexity} = 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 P(w_i|w_1,w_2,...,w_{i-1})}$$

- N: total number of words in the test set
- $P(w_i|w_1, w_2, ..., w_{i-1})$: conditional probability of the i-th word given the previous words in the sequence

Step 3: Model Evaluation

Reference:
Based on every word

BLEU Score

```
# Add in reference text
reference = []

# Headers include phrases that the seed texts have existing in the dataset
referenceHeader = [
    "Embrace", "Radiate", "Believe in yourself", "Life's", "Dance through", "Let your",
    "Every", "Singapore" ,"Our planet", "This morning, let"
    ]

for quote in data["Quotes"].values:
    # Check if the quote in the dataset contains any of the mentioned headers
    if any(text in quote for text in referenceHeader):

        # Convert quote to lowercase except for Singapore
        if "Singapore" not in quote:
            quote = quote.lower()
        # Remove any punctuations from the quotes
        reference.append(re.sub('[,;.]','', quote).split(' '))

references =[reference]
```

Decaying weights initialized

```
# Evaluate the BLEU score
def evaluate_bleu_score(text):
    candidate = ["".join([text]).split(" ")]
    score = corpus_bleu(references, candidate, weights=(0.1, 0.2, 0.3, 0.4), smoothing_function=SmoothingFunction().method1)
    print(f"BLEU Score: {score:.4f}")
    return score
```

Python

Perplexity

Based on its own probability distribution

```
# Calculate perplexity
def calculate_perplexity(loss, num_words):

    perplexity = 2**(-loss/num_words)
    print(f"Perplexity: {perplexity:.4f}")
    return perplexity
```

The lower the better

Prediction Time

- Use BLEU scores to determine which models are the better ones to use for model improvement.

```
# Temperature is used to control the randomness of the prediction
# The higher the temperature, the more random the prediction
def predict_next_words(input_text, model, temperature, num_words=1):
    log_likelihood = 0.0
    for _ in range(num_words):
        tokens = tokenizer.texts_to_sequences([input_text])[0]
        tokens = pad_sequences([tokens], maxlen=max_sequence_rolling_len, padding='pre')
        predicted_prob = model.predict(tokens, verbose=0)[0]
        prediction = np.log(predicted_prob) / temperature
        exp_preds = np.exp(prediction)
        predicted_probs = exp_preds / np.sum(exp_preds)
        chosen_word_index = np.random.choice(range(len(predicted_probs)), p=predicted_probs)
        predicted_word = tokenizer.index_word[chosen_word_index]
        input_text += " " + predicted_word
        log_likelihood += np.log2(predicted_prob[chosen_word_index])
    return input_text, log_likelihood
```

(variable) predi

Temperature helps to **control randomness of word generation**

Reference:
Based on every sentence

BERTScore

```
# Add in reference text
reference2 = []

# Headers include phrases that the seed texts have existing in the dataset
referenceHeader = [
    "Embrace", "Radiate", "Believe in yourself", "Life's", "Dance through", "Let your",
    "Every", "Singapore" ,"Our planet", "This morning, let"
    ]

for quote in data["Quotes"].values:
    # Check if the quote in the dataset contains any of the mentioned headers
    if any(text in quote for text in referenceHeader):

        # Convert quote to lowercase except for Singapore
        if "Singapore" not in quote:
            quote = quote.lower()
        # Remove any punctuations from the quotes
        reference2.append(re.sub('[,;.]','', quote))

references2 =[reference2]
```

```
# Evaluate BERTScore
def evaluate_bert_score(text):
    candidate = ["".join([text])]

    # Calculate BERT score
    precision, recall, f1 = score(candidate, references2, lang="en", verbose=False)

    print(f"Precision: {precision.mean().item():.4f}")
    print(f"Recall: {recall.mean().item():.4f}")
    print(f"F1 Score: {f1.mean().item():.4f}")
    # Return the mean of the scores
    return precision.mean().item(), recall.mean().item(), f1.mean().item()
```

Step 3: Model Evaluation

Perplexity and BLEU Score

```python
#Function to plot a graph to consolidate BLEU scores and perplexity
def calc_bleu_and_perplexity(model_name ,model, seed_texts, temperature, num_word=10):

    bleu_scores = []
    perplexity_scores = []
    for seed_text in seed_texts:
        prediction, loss = predict_next_words(seed_text, model, temperature, num_word)
        print(''.join(prediction))
        bleu_score = evaluate_bleu_score(prediction)
        perplexity = calculate_perplexity(loss, len(prediction.split(' ')))
        print()
        bleu_scores.append(bleu_score)
        perplexity_scores.append(perplexity)

    # Plot a bar graph to show the BLEU scores
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,8), sharex=True)

    # Plot BLEU scores
    ax1.bar(range(1, len(bleu_scores)+1), bleu_scores, color='blue', alpha=0.7)
    ax1.set_title(f'BLEU Scores: {model_name}')
    ax1.set_ylabel("BLEU Score", color="blue")
    ax1.axhline(y=0.5, color='r', linestyle='-')

    # Plot perplexity scores
    ax2.bar(range(1, len(perplexity_scores)+1), perplexity_scores, color='green', alpha=0.7)
    ax2.set_title(f'Perplexity Scores: {model_name}')
    ax2.set_ylabel("Perplexity", color="green")
    ax2.axhline(y=1.5, color='grey', linestyle='--')

    plt.tight_layout()
    plt.show()
```

[497] Python

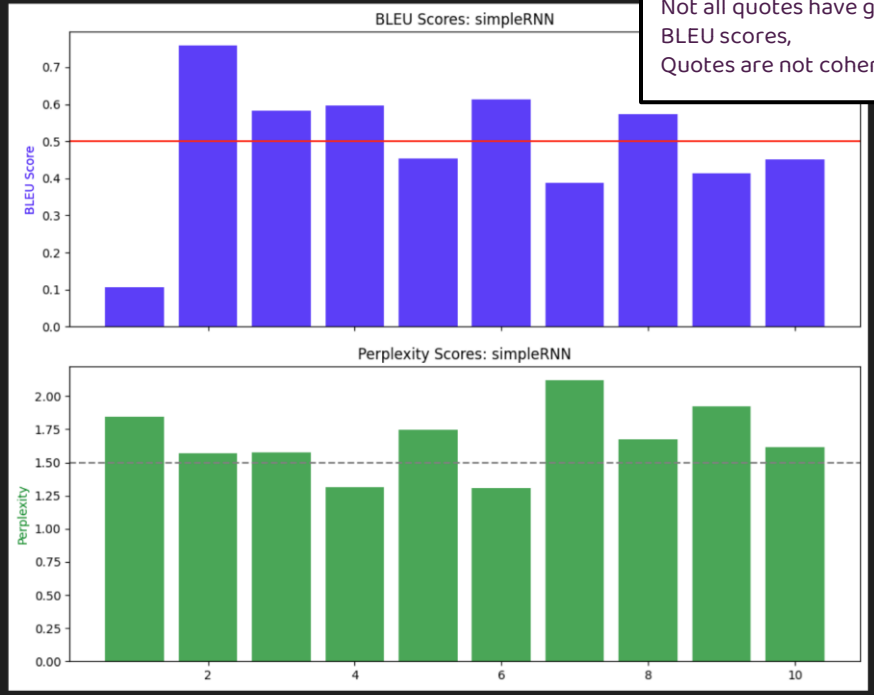
Function to display graph of **perplexity and BLEU score**, as well as show predictions

Step 3: Model Evaluation

Baseline Model:
Perplexity is not so good,
Not all quotes have good BLEU scores,
Quotes are not coherent

```
calc_bleu_and_perplexity("simpleRNN", simpleRNN, seed_texts, 0.4)
```

```
embrace each day is a precious gift to true beauty of our planet
BLEU Score: 0.1055
Perplexity: 1.8448

radiate some strength and let it be the compass that guides you
BLEU Score: 0.7573
Perplexity: 1.5710

believe that powers your journey a tale worth telling with warmth and
BLEU Score: 0.5820
Perplexity: 1.5738

life's actual purpose is a testament to the goodness in your soul filling it
BLEU Score: 0.5965
Perplexity: 1.3119

dance through each and every will be a force that drives you forward is a
BLEU Score: 0.4545
Perplexity: 1.7440

let your time and energy through the canvas of your journey a tale worth telling
BLEU Score: 0.6125
Perplexity: 1.3063

every person is a celebration of our planet offers the promise of blossoming
BLEU Score: 0.3883
Perplexity: 2.1174

our country Singapore is a testament to the beauty of our planet's breath is
BLEU Score: 0.5723
Perplexity: 1.6722

planet earth is a wonder of the beauty of our uniqueness and the
BLEU Score: 0.4135
Perplexity: 1.9249

morning and evening would make it be the compass that guides you home moments that define
BLEU Score: 0.4516
Perplexity: 1.6176
```



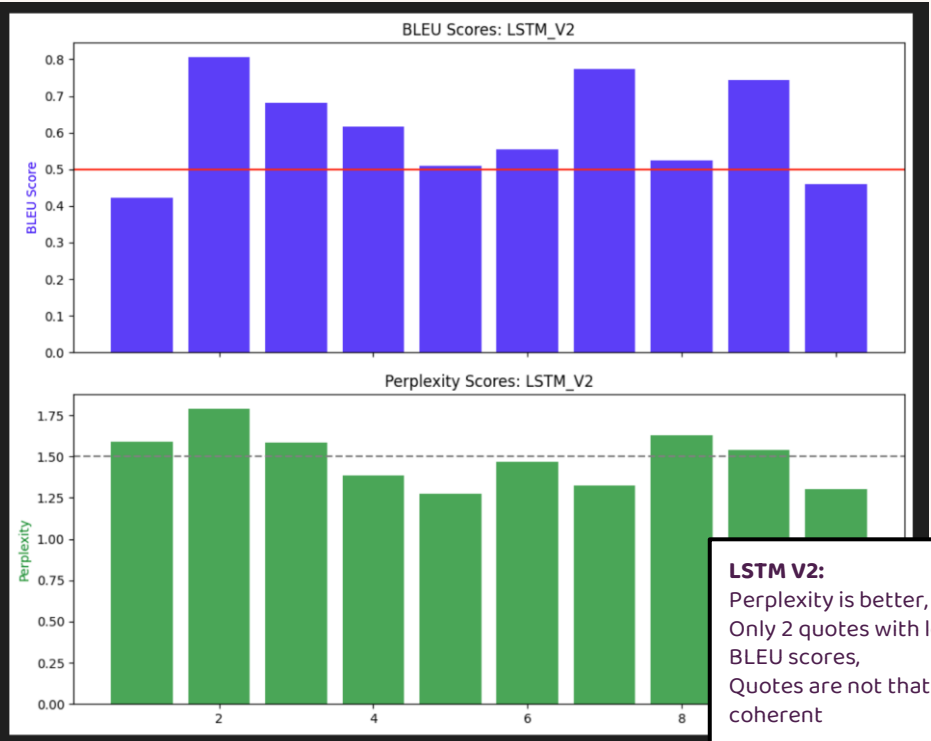
Observations

- The simpleRNN V1 is not performing that well as many of the phrases BLEU score go below 0.5.
- BLEU Score of Quotes 2 and 9 is quite high, implying that those are quite coherent phrases.
- Quotes 4 and 6 has lower perplexity which shows that the model is more confident in its predictions and better at assigning the high probabilities to the actual words in the sequence.

Step 3: Model Evaluation

```
calc_bleu_and_perplexity("LSTM_V2", LSTM_V2, seed_texts, 0.2)
```

```
embrace each day is a precious gift a gift a reminder that you
BLEU Score: 0.4223
Perplexity: 1.5902

radiate some enthusiasm and let it be the foundation of your strength
BLEU Score: 0.8055
Perplexity: 1.7878

believe that heals and unites and resentment a path towards healing and
BLEU Score: 0.6823
Perplexity: 1.5829

life's actual purpose is a testament to the beauty of our uniqueness and growth
BLEU Score: 0.6156
Perplexity: 1.3850

dance through each and every opportunities to shine for they hold the keys to your
BLEU Score: 0.5100
Perplexity: 1.2736

let your time and energy it is the music of a joyful heart for it
BLEU Score: 0.5538
Perplexity: 1.4680

every person is a testament to the beauty of the human spirit a
BLEU Score: 0.7744
Perplexity: 1.3263

our country Singapore is the heartbeat of our planet and let it be the
BLEU Score: 0.5235
Perplexity: 1.6284

planet earth is a classroom where we learn to love and forgive and
BLEU Score: 0.7440
Perplexity: 1.5427

morning and evening would make it can light up even the darkest days together wonder of
BLEU Score: 0.4605
Perplexity: 1.3002
```



LSTM V2:
Perplexity is better,
Only 2 quotes with low BLEU scores,
Quotes are not that coherent

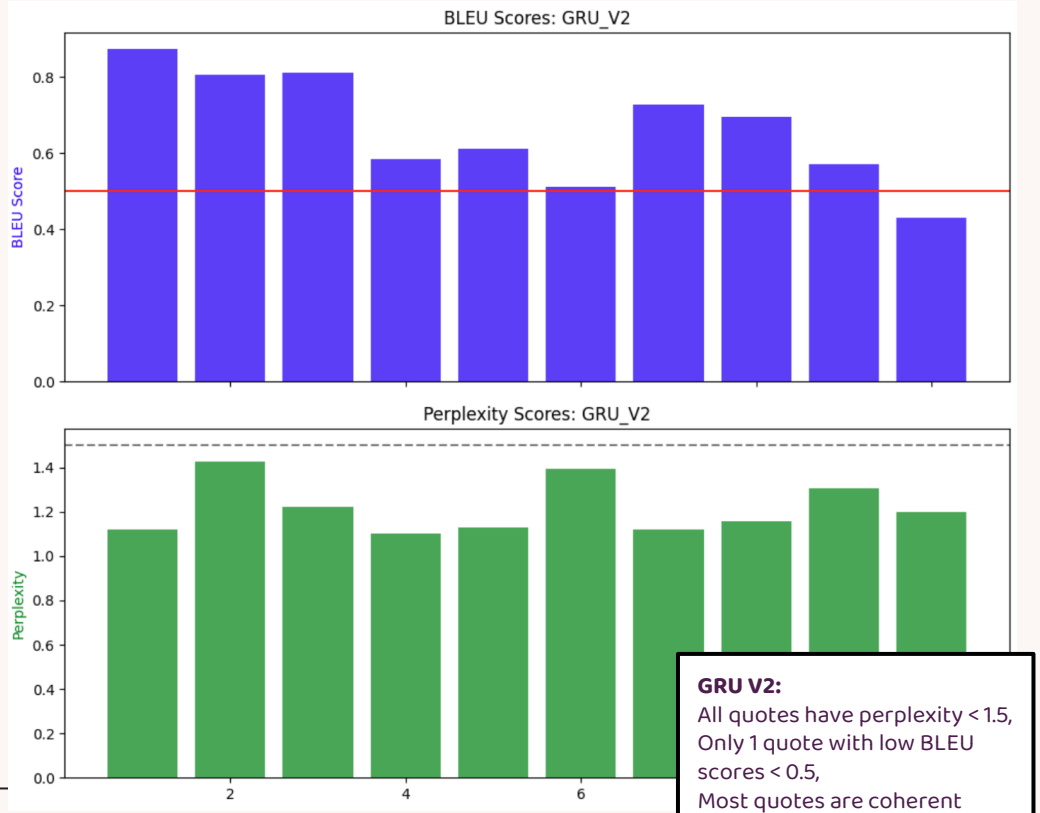
Observations

- With the exception of Quote 1 & 10, the BLEU score of the phrases generally perform quite well.
- Perplexity is also showing the values are close to 1.5, indicating that the model is relatively confident in its predictions.

Step 3: Model Evaluation

```
calc_bleu_and_perplexity("GRU_V2", GRU_V2, seed_texts, 0.1)
```

```
embrace each day with a heart full of gratitude for it is the
BLEU Score: 0.8735
Perplexity: 1.1224

radiate some gratitude for it is the heartbeat of a joyful heart
BLEU Score: 0.8055
Perplexity: 1.4275

believe that yourself and you will be a source of light for
BLEU Score: 0.8125
Perplexity: 1.2205

life's actual purpose is the pursuit of our passions and dreams and aspirations and
BLEU Score: 0.5845
Perplexity: 1.0996

dance through each and every inspire is a testament to the beauty of our uniqueness
BLEU Score: 0.6125
Perplexity: 1.1297

let your time and energy will follow and become a beacon of light in the
BLEU Score: 0.5114
Perplexity: 1.3926

every person is the jewels set in the crown of the sea of
BLEU Score: 0.7281
Perplexity: 1.1199

our country Singapore is a testament to the nation's resilience and unwavering determination and
BLEU Score: 0.6961
Perplexity: 1.1576

planet earth is a testament to your inner strength and resilience that resonates
BLEU Score: 0.5714
Perplexity: 1.3073

morning and evening would make it is the compass of endless discovery soul and inner peace
BLEU Score: 0.4312
Perplexity: 1.1988
```

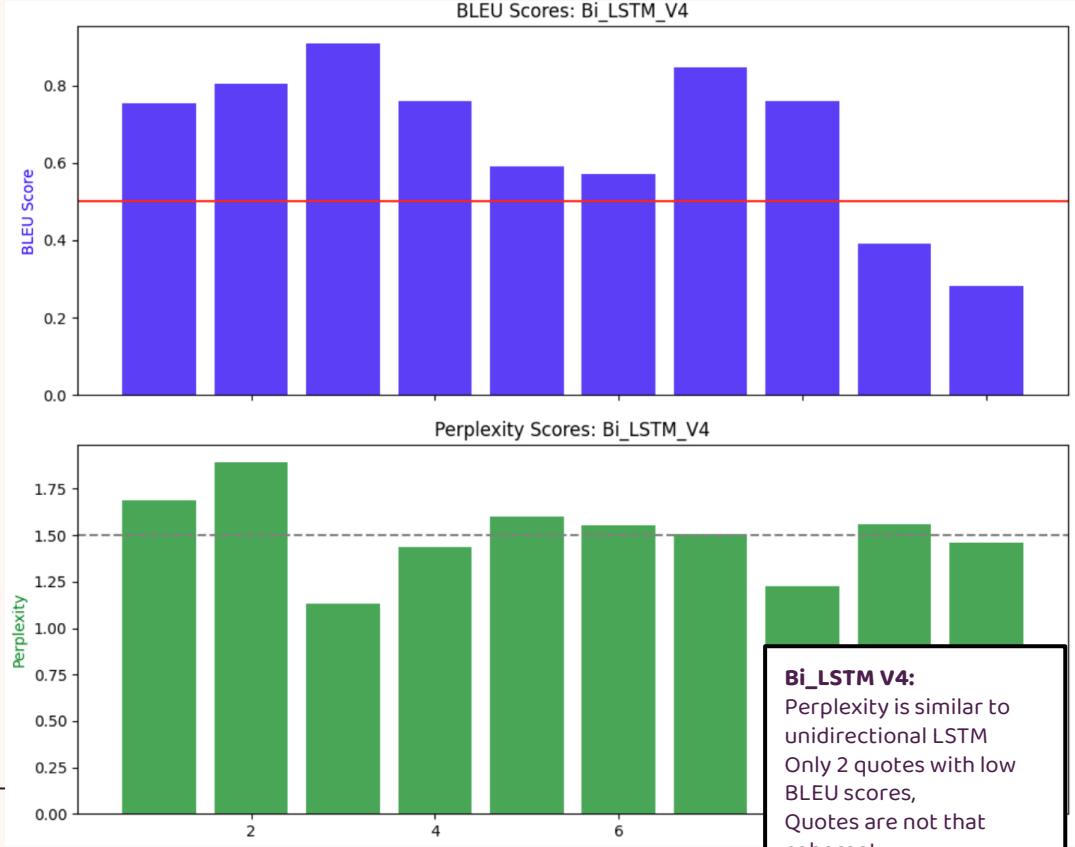


GRU V2:
All quotes have perplexity < 1.5,
Only 1 quote with low BLEU scores < 0.5,
Most quotes are coherent

Step 3: Model Evaluation

```
calc_bleu_and_perplexity("Bi_LSTM_V4", Bi_LSTM_V4, seed_texts, 0.3)
```

```
embrace each day with a heart full of gratitude and determination shape destinies
BLEU Score: 0.7543
Perplexity: 1.6876

radiate some grace and let it be the fortress of your soul
BLEU Score: 0.8055
Perplexity: 1.8912

believe that reverberates in the heart spreading warmth and happiness far and
BLEU Score: 0.9068
Perplexity: 1.1312

life's actual purpose is the symphony of life and let your heart be the
BLEU Score: 0.7607
Perplexity: 1.4338

dance through each and every step we take towards our dreams and aspirations and the
BLEU Score: 0.5906
Perplexity: 1.5995

let your time and energy and let it be the foundation of your greatness of
BLEU Score: 0.5722
Perplexity: 1.5539

every person is a gesture of hope for the future of our planet
BLEU Score: 0.8463
Perplexity: 1.5053

our country Singapore is an investment in a brighter future in the realization of
BLEU Score: 0.7607
Perplexity: 1.2228

planet earth is the light of experience they hold the promise of a
BLEU Score: 0.3915
Perplexity: 1.5598

morning and evening would make it with intention and love that believe in our souls and
BLEU Score: 0.2826
Perplexity: 1.4591
```



Bi_LSTM V4:
Perplexity is similar to unidirectional LSTM
Only 2 quotes with low BLEU scores,
Quotes are not that coherent

Step 4: Model Improvement

4) Model Improvement

Attention Mechanism

- Allow models to focus on specific parts of the input sequence when making predictions, rather than relying on the entire input uniformly.
- Helps improve model's performance, especially when dealing with long sequences.
- Self-attention layers enable parallelization and capturing long range dependencies more effectively.

```
class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    # Build method
    def build(self,input_shape):
        # Create a trainable weight variable for this layer
        self.W=self.add_weight(name="att_weight",shape=(input_shape[-1],1),initializer="he_normal")
        # Create a trainable bias variable for this layer
        self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1),initializer="zeros")
        super(attention, self).build(input_shape)

    # Call method
    def call(self,x):
        # Computes the attention scores
        et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
        # Computes the attention weights using softmax activation function
        at=K.softmax(et)
        at=K.expand_dims(at,axis=-1)
        # Compute the weighted sum of the input vectors
        output=x*at
        return K.sum(output,axis=1)

    # Compute output shape
    def compute_output_shape(self,input_shape):
        return (input_shape[0],input_shape[-1])

    # Returns a dictionary containing the configuration used to initialize this layer
    def get_config(self):
        return super(attention,self).get_config()
```

GRU Attention Mechanism

- GRU layer of 256 units with custom attention layer

```
tf.keras.backend.clear_session()

# Create the model to add in the attention for the GRU model  with 256 units
GRU_with_attention = Sequential(
    name='GRU_with_attention',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        GRU(256, activation='tanh', return_sequences=True),
        attention(),

        Dropout(0.3),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
GRU_with_attention.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

GRU_with_attention_history = GRU_with_attention.fit(
    X_train_roll, y_train_roll,
    epochs=100,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

Step 4: Attention Mechanism

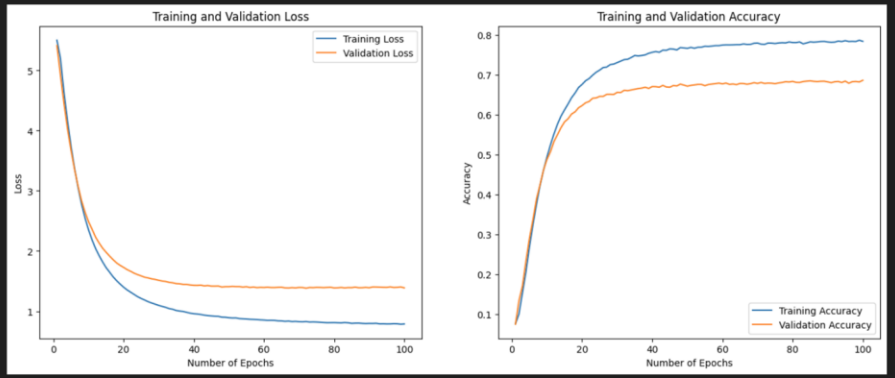
Attempted **many weight and bias initializers**, every one of them gave roughly the same results, which **worsens** compared to **before attention mechanism**

```
GRU_with_attention.summary()
```

```
Model: "GRU_with_attention"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 gru (GRU)                   (None, 34, 256)           205824

 attention (attention)       (None, 256)               290

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 1199)              308143

=================================================================
Total params: 526,247
Trainable params: 526,247
Non-trainable params: 0
_________________________________________________________________
```

```
plot_learning_curve(GRU_with_attention_history.history)
```

Python



Observations

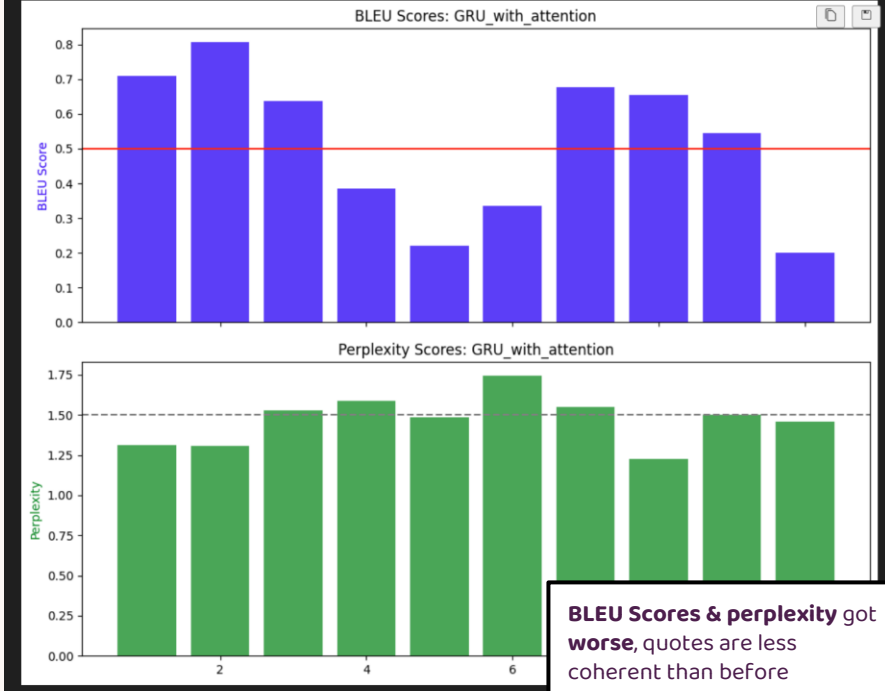
- It looks like attention mechanism really did not converge as well as the training data.

```
GRU_with_attention.evaluate(X_test_roll, y_test_roll)
```

Python

```
ETA: 6s                                    - 1s 2ms/step - loss: 1.3926 - accuracy: 0.6821
[1.3926458358764648, 0.6820856332778931]
```

Step 4: Attention Mechanism



BLEU Scores & perplexity got **worse**, quotes are less coherent than before

Observations

- As observed, the attention layer actually reduced the BLEU score of the predictions compared to before.
- Possible reason is that the quotes are not long enough for the attention mechanism to be very effective, weights are not being allocated as efficiently.
- Perplexity values have also increased compared to before, showing that the model is less confident in its predictions.

After Inserting Attention Layer

```
# After
calc_bleu_and_perplexity("GRU_with_attention", GRU_with_attention, seed_texts, 0.1)
```

```
embrace each day with a heart full of gratitude gratitude is a step
BLEU Score: 0.7087
Perplexity: 1.3148

radiate some gratitude for it turns even the smallest gifts into treasures
BLEU Score: 0.8055
Perplexity: 1.3092

believe that spreads are in the treasury of cherished memories a testament
BLEU Score: 0.6363
Perplexity: 1.5293

life's actual purpose is the legacy of love and compassion our planet and aspirations
BLEU Score: 0.3853
Perplexity: 1.5855

dance through each and every moment for a moment of a fresh start for new
BLEU Score: 0.2213
Perplexity: 1.4826

let your time and energy with joy and kindness will light up the world creating
BLEU Score: 0.3356
Perplexity: 1.7417

every person is a testament a gift to the boundless power of the
BLEU Score: 0.6769
Perplexity: 1.5491

our country Singapore is a canvas of your destiny each decision shaping the masterpiece
BLEU Score: 0.6544
Perplexity: 1.2263

planet earth is a treasure trove in the beauty in transitions in the
BLEU Score: 0.5456
Perplexity: 1.4991

morning and evening would make it brings forth new beginnings to inspire change a new day
BLEU Score: 0.1997
Perplexity: 1.4581
```

Step 4: Optimizers

Optimizers

Adam

- An adaptive learning rate optimization algorithm
- Maintains a moving average of both the gradients and the second moments of the gradients
- Adapts the learning rate of each parameter individually

SGD (Stochastic Gradient Descent)

- Basic optimization algorithm which update the model parameters based on the negative gradient of the loss function with respect to each parameter.
- Each update is based on a random subset of the training data (mini batch)

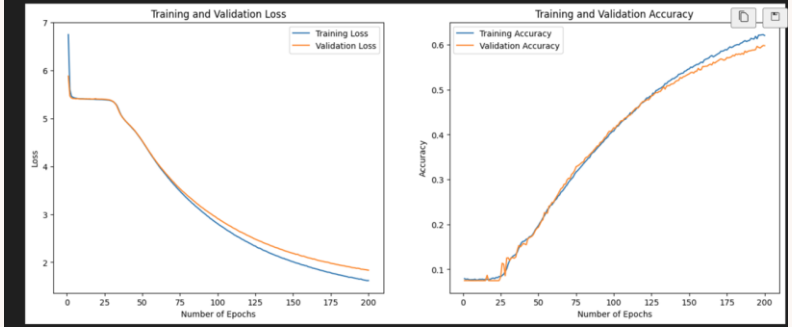
SGD is **underperforming** compared to Adam, hence **Adam** will still be used

GRU with SGD optimizer

```
# SGD optimizer
tf.keras.backend.clear_session()

# Create the model
GRU_SGD = Sequential(
    name='GRU_SGD',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        GRU(256, activation='tanh'),

        Dropout(0.3),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = SGD(learning_rate=0.001,  momentum = 0.9, nesterov= True)
GRU_SGD.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

GRU_SGD_history = GRU_SGD.fit(
    X_train_roll, y_train_roll,
    epochs=200,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

```
Model: "GRU_SGD"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 gru (GRU)                   (None, 256)               205824

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 1199)              308143

=================================================================
Total params: 525,957
Trainable params: 525,957
Non-trainable params: 0
_________________________________________________________________
```

```
plot_learning_curve(GRU_SGD_history.history)
```



Observations

- The loss curve just show a sharp and quick drop of the training and validation loss in the initial epochs.
- Accuracy curve just had show a slight increase in the first 25 epochs then increase gradually in the next few epochs.
- Maybe we will have to change the learning rate and intialise weights differently for SGD

```
GRU_SGD.evaluate(X_test_roll, y_test_roll)
```

```
383/383 [==============================] - 1s 2ms/step - loss: 1.8300 - accuracy: 0.6006

[1.830045461654663, 0.600604772567749]
```

Step 4: Regularisation

L2 Regularized GRU + Increased Dropout

- GRU layer with 256 units
- Added L2 Regularization of 0.01
- Increased Dropout to 0.5
- **L2 Regularization**:
  - Known as weight decay and usually better over L1. Forces the weights to decay towards zero.

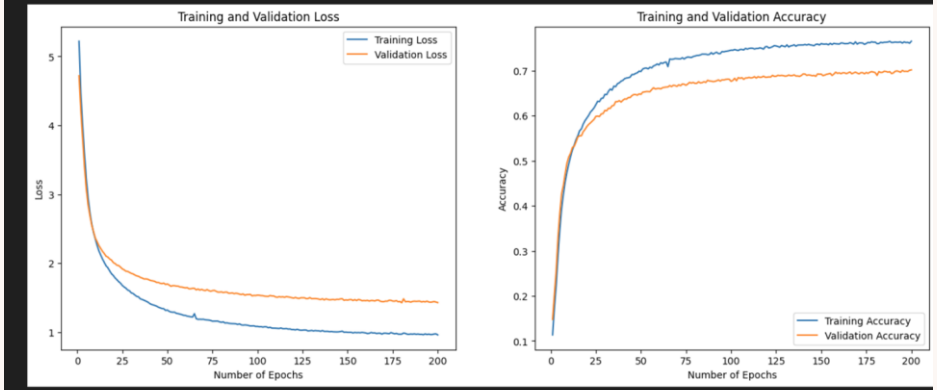
$$L2(\theta) = \lambda \sum_{i=1}^{n} w_i^2$$

- **Dropout**
  - Have been using this form of regularisation so far.
  - At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.

```
tf.keras.backend.clear_session()

# Create the model
GRU_l2 = Sequential(
    name='GRU_L2_Regularizer',
    layers=[
        Embedding(total_words_rolling, 10, input_length = max_sequence_rolling_len),
        GRU(256, activation='tanh', kernel_regularizer=l2(0.01)),

        Dropout(0.5),
        Dense(total_words_rolling, activation='softmax')
    ]
)

opt = Adam(learning_rate=0.001)
GRU_l2.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

GRU_l2_history = GRU_l2.fit(
    X_train_roll, y_train_roll,
    epochs=200,
    validation_data = (X_val_roll, y_val_roll),
    batch_size=32,
    verbose=1
)
```

```
Model: "GRU_L2_Regularizer"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 34, 10)            11990

 gru (GRU)                   (None, 256)               205824

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 1199)              308143

=================================================================
Total params: 525,957
Trainable params: 525,957
Non-trainable params: 0
_________________________________________________________________
```

```
plot_learning_curve(GRU_l2_history.history)
```



Observations

- Learning curves of both loss and accuracy seems to converge well to a certain point.
- Training loss is still lower than validation loss and the training accuracy is still higher than the validation accuracy.

```
GRU_l2.evaluate(X_test_roll, y_test_roll)
```

```
383/383 [==============================] - 1s 3ms/step - loss: 1.4379 - accuracy: 0.6980

[1.437853455543518, 0.6980222463607788]
```

Observations

- The test accuracy evaluated was a bit lower than before regularizing, previously 0.70.

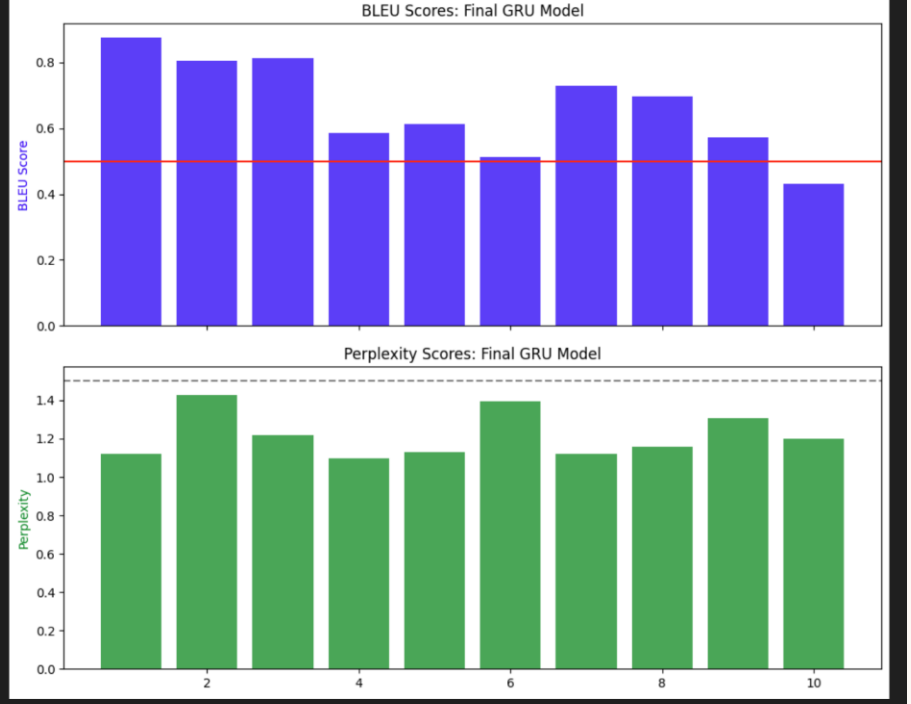
Step 4: Model Evaluation

```
embrace each day with a heart full of gratitude for it is the
BLEU Score: 0.8735
Perplexity: 1.1224
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9212
Recall: 0.9651
F1 Score: 0.9426

radiate some gratitude for it is the heartbeat of a joyful heart
BLEU Score: 0.8055
Perplexity: 1.4275
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9435
Recall: 0.9491
F1 Score: 0.9455

believe that yourself and you will be a source of light for
BLEU Score: 0.8125
Perplexity: 1.2205
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9359
Recall: 0.9287
F1 Score: 0.9323

life's actual purpose is the pursuit of our passions and dreams and aspirations and
BLEU Score: 0.5845
Perplexity: 1.0996
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9127
Recall: 0.9426
F1 Score: 0.9274

dance through each and every inspire is a testament to the beauty of our uniqueness
BLEU Score: 0.6125
Perplexity: 1.1297
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9018
Recall: 0.9402
F1 Score: 0.9206

let your time and energy will follow and become a beacon of light in the
BLEU Score: 0.5114
Perplexity: 1.3926
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.8814
Recall: 0.9007
F1 Score: 0.8902
```

```
every person is the jewels set in the crown of the sea of
BLEU Score: 0.7281
Perplexity: 1.1199
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9185
Recall: 0.8905
F1 Score: 0.9042

our country Singapore is a testament to the nation's resilience and unwavering determination and
BLEU Score: 0.6961
Perplexity: 1.1576
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9069
Recall: 0.9117
F1 Score: 0.9093

planet earth is a testament to your inner strength and resilience that resonates
BLEU Score: 0.5714
Perplexity: 1.3073
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.9130
Recall: 0.9364
F1 Score: 0.9245

morning and evening would make it is the compass of endless discovery soul and inner peace
BLEU Score: 0.4312
Perplexity: 1.1988
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Precision: 0.8675
Recall: 0.8877
F1 Score: 0.8775
```

BERTScore are relatively high except the last quote

Step 4: Final Model



Observations

- As seen, the BERTScore shows that the precision, recall and f1 score for all the quotes were relatively high, except the last quote, only obtaining mean values of 0.8 for the 3 aforementioned metrics.
- High BLEU Score indicates a good level of agreement between the predicted text and the referenced text.
- Perplexity values were all below the 1.5 threshold, indicating that this model is more confident in its predictions and is better at assigning high probabilities to the actual words in the sequence, implying how meaningful the quotes are.

Final Model: GRU with 256 units, Dropout of 0.3, Regularisation L2

Save Weights of Final Model

- Saving the weights of the final model.

```
# Save weights of the final model
GRU_V2.save_weights("./RNN_FinalModels/Final_GRU_256.h5")
```

Summary

In summary, for next word prediction, there is **insufficient data** to make the **accuracy as high as models** used for other tasks, **predictions are also not very coherent** because of that. The model I think performed best for this task is the **GRU with 256 units and a Dropout of 0.3** based on **how coherent** the text was, **how high the BLEU score was** and **how low the perplexity values** were.

Thank You!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**