# Dylan Govender – 221040222
# COMP703 – Assignment III Report – 24/05/2024
# Character-Level Machine Translation

---

## [1] Defining Sequence-to-Sequence Models for Character-Level Machine Translation

### [1.1] Common Tasks Accomplished when Defining Each Model

#### [1.1.1] Importing the Data - *Section 1 in each Code File*

The data was downloaded from the website, *http://www.manythings.org/anki/fra-eng.zip*, as a zip file unpacked to retrieve the '*fra-eng.txt*' text file using the '*!wget*' and '*!unzip*' commands.

#### [1.1.2] Data Preprocessing - *Section 2 in each Code File*

The data preprocessing code for character-level machine translation starts by setting a sample limit ('*MAX_SAMPLES*') to 10,000 sentences from the imported bilingual text file. It initialises two lists to store input (input language) and target sentences (output language) ('*input_texts'*, '*target_texts*') and two sets to collect unique characters from both languages ('*input_characters*', '*target_characters*'). The data preprocessing code reads and extracts sentences from the text file, adds start (\t) and end (\n) tokens to the target sentences and populates the lists and sets. It then sorts the character sets and calculates the number of unique characters ('*num_encoder_tokens*', '*num_decoder_tokens*') and the maximum sentence lengths ('*max_encoder_seq_length*', '*max_decoder_seq_length*'). Index mappings ('*input_token_index*', '*target_token_index*') are created for each character with shapes based on the number of samples, maximum sequence length, and character vocabulary length. Zero matrices ('*encoder_input_data*', '*decoder_input_data*', '*decoder_target_data*') are initialised to hold one-hot encoded representations of the sentences. The preprocessing populates these matrices, where each character in the input and target sentences is represented as a one-hot vector, with the decoder target data shifted by one timestep to align with the next character in the sequence. Below is an output after preprocessing the data:

```
Number of samples: 10000
Number of unique input tokens: 70
Number of unique output tokens: 91
Max sequence length for inputs: 14
Max sequence length for outputs: 59
```

#### [1.1.3] Training Each Model - *Section 4 in each Code File*

This section trains a neural network model for sequence-to-sequence tasks, using specific configurations and then saves the trained model. There are two hyperparameters: '*BATCH_SIZE*', sets the number of samples per batch of computation to 64. '*EPOCHS*', specifies the number of complete passes through the training dataset. Each model is compiled using the '*model.compile*' configuring the model for training and includes the *optimizer="rmsprop"* that uses the RMSprop optimiser, the *loss="categorical_crossentropy"* that sets the loss function to categorical cross-entropy, suitable for multi-class classification, and the *metrics=["accuracy"]* that tracks accuracy during training and evaluation. Each model was then trained using the '*model.fit*' function which trains the model based on the hyperparameters, the '*encoder_input_data*', and '*decoder_input_data*' as arguments. The '*encoder_input_data*', and '*decoder_input_data*' are related to the input data for the encoder and decoder parts of the sequence model. Finally, the model was saved using the '*model.save*' function provided by the Keras API for later

reconstruction and evaluation. No callbacks were used, to disallow any model from achieving an advantage. Every hyperparameter, input size, output size, and parameter were kept constant throughout each model definition to allow for a fairer comparison analysis.

### [1.1.4] Evaluating Each Model - *Section 5 in each Code File*

Each model was evaluated using the final epoch accuracy and the BLEU score. The accuracy determines the proportion of correct predictions made by the model compared to the total number of predictions during training. It is a metric used to evaluate the performance of the model, indicating how well the model is correctly classifying the input data into the correct categories.

BLEU score is the Bilingual Evaluation Understudy score, a metric used to evaluate the quality of machine-translated text against one or more reference translations. It measures how closely the machine-generated output matches human-produced reference translations, based on the precision of n-grams (contiguous sequences of n characters or words). In each character-level machine translation model, the BLEU score determines the accuracy and fluency of the translated output at the character level. Specifically, it assesses how well the translated sequence captures the correct characters in order, comparing the machine translation to reference translations by calculating the overlap of character n-grams. A higher BLEU score indicates a closer match to the reference translations, implying better translation quality.

Each model was reconstructed layer by layer to be reevaluated and to decode some sequence of characters. Using the encoder and decoder models, each model uses the 'decode_sequence' function to generate a decoded sequence from a given input sequence. For each input sequence from the first 20 sequences in the training set, it encodes the input into state vectors, initialises the target sequence, and iteratively predicts and samples the next character until the stop condition is met, printing the input and decoded sentences.

The NLTK library evaluated the BLEU score for each character-level machine translation model. The '*bleu_score*' function takes the '*TEST_SIZE*' as an argument which determines the number of sequences to evaluate. It uses *SmoothingFunction().method4* to handle cases where n-grams might not appear in the reference. For each sequence (target and predicted), unwanted characters were removed (such as tabs, newlines, spaces, and non-breaking spaces) from both the reference and candidate translations. The BLEU score for each sequence is calculated using the '*sentence_bleu*' function with unigram precision *weights=(1, 0, 0, 0)*, and the average BLEU score across all evaluated sequences is returned. Finally, the code prints the average BLEU scores for 100 and 1000 samples from the dataset.

## [1.2] [a] Bi-LSTM Encoder & LSTM Decoder Model - '*BiLSTM_Encoder_LSTM_Decoder.ipynb*'

### [1.2.1] Building the Model - *Section 3 in Code File*

This section is used to construct a sequence-to-sequence model for character-level machine translation using a Bidirectional LSTM (BiLSTM) encoder and an LSTM decoder with a latent dimension of 256. The encoder processes the input sequence '*encoder_inputs*', with a shape of (None, num_encoder_tokens), and consists of the BiLSTM layer. The BiLSTM outputs are split into forwards and backwards hidden, and cell states (forward_h, forward_c, backward_h, backward_c), concatenated to form the combined states '*state_h*' and '*state_c*'. These concatenated states are used as the initial states for the decoder. The decoder takes '*decoder_inputs*' of shape (None, '*num_decoder_tokens*') and uses an LSTM layer configured to return full output sequences and internal states (return_sequences=True, return_state=True).

The LSTM outputs are passed through a dense layer with a '*softmax*' activation (decoder_dense) to generate the final output probabilities for each character. The model is defined with inputs '*encoder_inputs*' and '*decoder_inputs*' producing the output '*decoder_outputs*', and the model summary is printed to display its architecture.

**[1.2.2] Model Evaluation**
**Training Accuracy:** 0.9725 (97.25%)
**Average BLEU score on 100 samples:** 0.5514 (55.14%)
**Average BLEU score on 1000 samples:** 0.5836 (58.37%)

The results suggest that the BiLSTM-LSTM model achieves a high training accuracy of 97.25%, indicating proficient learning on the training data. However, the moderate BLEU scores of 55.14% for 100 samples and 58.37% for 1000 samples suggest that the model's character-level translation quality is only moderate on unseen data, indicating potential limitations in generalisation.

**[1.3] [b] Bi-GRU Encoder & GRU Decoder Model - *'BiGRU_Encoder_GRU_Decoder.ipynb'***

**[1.3.1] Building the Model - *Section 3 in Code File***
This section constructs a character-level machine translation model with a bidirectional GRU encoder and a single-layered GRU decoder. The encoder takes input sequences (encoder_inputs) of shape (None, num_encoder_tokens) and incorporates a Bidirectional GRU layer with a latent dimension, 256, to process the input. The final hidden states from the forward and backward GRU layers are concatenated to form the encoder states. The decoder receives input sequences (decoder_inputs) of shape (None, num_decoder_tokens) and includes a single-layered GRU with a latent dimension of 512, initialised with the concatenated encoder states. The model is defined to map encoder and decoder input data to decoder target data, producing output probabilities for each character. The final dense layer uses a '*softmax*' activation function to generate the final output probabilities for each character.

**[1.3.2] Model Evaluation**
**Training Accuracy:** 0.9840 (98.40%)
**Average BLEU score on 100 samples:** 0.5422 (54.22%)
**Average BLEU score on 1000 samples:** 0.5871 (58.71%)

The training accuracy of 98.40% indicates that the model performs very well on the training data, achieving high accuracy in predicting the correct output. However, the average BLEU scores of 54.22% for 100 samples and 58.71% for 1000 samples suggest that the translation quality of the model is moderate on unseen data. The Bi-GRU-GRU model has performed better than the Bi-LSTM-LSTM model in terms of training and predicting on a larger sample of data.

**[1.4] [c] 2-Layer GRU Encoder & GRU Decoder Model - *'2GRU_Encoder_GRU_Decoder.ipynb'***

**[1.4.1] Building the Model - *Section 3 in Code File***
This section constructs a character-level machine translation model with a 2-layered GRU encoder and a single-layered GRU decoder. The encoder takes input sequences (encoder_inputs) of shape (None, num_encoder_tokens) and consists of two GRU layers, each with a latent dimension of 256. The decoder receives input sequences (decoder_inputs) of shape (None, num_decoder_tokens) and incorporates a GRU layer with a latent dimension of 512, concatenated with the final states from the encoder. The model is defined to map encoder and

decoder input data to decoder target data, producing output probabilities for each character. The final dense layer uses a '*softmax*' activation function to generate the final output probabilities for each character.

**[1.4.2] Model Evaluation**
**Training Accuracy:** 0.9725 (97.25%)
**Average BLEU score on 100 samples:** 0.5256 (52.56%)
**Average BLEU score on 1000 samples:** 0.5066 (50.66%)

The training accuracy of 97.25% indicates that the model performs well on the training data, achieving high accuracy in predicting the correct output. However, the average BLEU scores of 52.56% for 100 samples and 50.66% for 1000 samples suggest that the translation quality of the model is relatively lower on unseen data compared to the Bi-GRU-GRU model.

## [2] Model Comparison and Evaluation

a. **Which Gated-RNN, GRU or LSTM, performs better than the other on the dataset provided?**
   The Gated-RNN GRU performs better than the LSTM as depicted by the BLEU and accuracy evaluation. GRU has a simpler architecture compared to LSTM, with fewer parameters. This simplicity can make the GRU easier to train and less prone to overfitting, especially when computational resources are limited. GRU has a more streamlined approach to managing memory compared to LSTM. It combines the forget and input gates of LSTM into a single update gate, reducing redundancy and making it more efficient in capturing long-range dependencies. GRU's design helps alleviate the vanishing gradient problem, which can occur during the training. This problem arises when gradients become extremely small, hindering the learning process. GRU's gating mechanism allows it to maintain better gradient flow through time, leading to more stable training.

b. **Does a layered architecture perform better than the bidirectional architecture on the provided dataset?**
   No. The bidirectional architecture outperforms the 2-layered architecture. Bidirectional architectures capture information from past and future contexts simultaneously, providing a more comprehensive understanding of the input sequence. This enhanced contextual understanding and generalisation ability of the model where computational resources are limited, making bidirectional architectures better at character-level machine translation applications. Bidirectional architectures may be more effective at attending to relevant parts of the input sequence due to their ability to capture bidirectional dependencies. This enhanced attention mechanism can help the model focus on relevant information during both the encoding and decoding stages, leading to better translation quality. The 2-layered architecture would have performed better if the computational resources were sufficient (sequence-level) which would have allowed for more depth in the model, enabling it to learn more complex patterns and representations from the data.

## [3] Conclusions

The GRU's overall performance is better than the LSTM as depicted by the BLEU score and training accuracy. The Bidirectional GRU encoder and GRU decoder architecture outperformed the 2-layered GRU encoder and GRU decoder architecture.