**Project: Machine Translation**

**Dataset selection**

For the project 2, Machine translation, we have decided to translate French to English. So, we have downloaded the fra.txt as our dataset for running the different experiments and gauging the performance of different modeling and fine-tuning processes.

**Dataset preprocessing**

The fra.txt file contains the equivalent English translation for the French text. This is a dataset with ~30,000 parallel English and French sentences, each with ~12 words per sentence.

1. We coded up the models in PyTorch and using TorchText to help us do all of the pre-processing required. We have also used spaCy to assist in the tokenization of the data.
2. We created the tokenizers. A tokenizer is used to turn a string containing a sentence into a list of individual tokens that make up that string e.g. "good morning!" becomes ["good", "morning", "!"]. spaCy has model for each language ("fr" for French and "en" for English) which need to be loaded so we can access the tokenizer of each model.
3. we created the tokenizer functions. These can be passed to TorchText and will take in the sentence as a string and return the sentence as a list of tokens.
4. We have set the tokenize argument to the correct tokenization function for each, with French being the SRC (source) field and English being the TRG (target) field. The field also appends the "start of sequence" and "end of sequence" tokens via the init\_token and eos\_token arguments, and converts all words to lowercase.
5. The dataset is created by reading in the .txt file as a pandas data frame
6. The dataset is split to training and testing datasets and further training to train and validation datasets with a ratio of 0.8 for both
7. We got 8578 unique tokens for French and 14007 for English

**Experiment 1**

We have built our model in three parts. The encoder, the decoder and a seq2seq model that encapsulates the encoder and decoder and will provide a way to interface with each.

**Encoder & Decoder**

The encoder, a 2 layer LSTM. The decoder, which will also be a 2-layer LSTM

### Seq2Seq: For the final part of the implementation, we have implemented the seq2seq model. This will handle:

* receiving the input/source sentence
* using the encoder to produce the context vectors
* using the decoder to produce the predicted output/target sentence

The Seq2Seq model takes in an Encoder, Decoder, and a device (used to place tensors on the GPU, if it exists).

We then trained the seq2seq model with Adam optimizer and CrossEntropyLoss function

**Experiment 2 - Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation**

This model achieves improved perplexity whilst only using a single layer RNN in both the encoder and the decoder. In LSTM, the decoder crams information into the hidden states. Whilst decoding, the hidden state will need to contain information about the whole of the source sequence, as well as all the tokens have been decoded so far. By alleviating some of this information compression, and using GRU, this becomes a better model.

**Encoder:** The encoder is a single-layer GRU. We don't pass the dropout as an argument to the GRU as that dropout is used between each layer of a multi-layered RNN. GRU only requires and returns a hidden state, there is no cell state. Inside GRU, gating mechanisms control the information flow in to and out of the hidden state. The encoder takes in a sequence, passes it through the embedding layer, recurrently calculates hidden states, and returns a context vector (the final hidden state) z.

**Decoder:** GRU takes the embedded target token, the previous hidden state and context vector 𝑧 as inputs. The same context vector is reused by the encoder for every time-step in the decoder. The next token is predicted and only the top-layer decoder hidden state is returned. The embedding of current token and the context vector is passed to the linear layer.

**Seq2Seq Model:** The outputs tensor holds predictions, and the source sequence is fed into the encoder to receive a context vector. Initial decoder hidden state is set to context vector and a batch of <sos> tokens are used as first input. In decoder loop, receive a prediction and new hidden state and decide if going to teacher force or not.

**Experiment 3 - Neural Machine Translation by Jointly Learning to Align and Translate**

This model eliminates the need for the context vector to contain all information by allowing the decoder to look at the entire source sentence at each decoding step by using attention. First, attention vector is calculated that is the length of the source sentence. Each element is between 0 and 1, and the entire vector sums to 1. We then calculate a weighted sum of source sentence hidden states to get a weighted source vector.

**Encoder:** The encoder is a single-layer bidirectional GRU. With a bidirectional RNN, we have two RNNs in each layer. A forward RNN going over the embedded sentence from left to right and a backward RNN going over the embedded sentence from right to left.

First, initialize both the forward and backward initial hidden states to 0s. We'll also get two context vectors. The RNN returns outputs and hidden. As the decoder is not bidirectional, it only needs a single context vector to use as its initial hidden state. We concatenate the two context vectors together, passing them through a linear layer, and applying the tanh activation function.

**Attention:** the attention layer takes in the previous hidden state of the decoder and all the stacked forward and backward hidden states from the encoder. The attention vector represents which words in the source sentence we should pay the most attention to correctly predict the next word. We calculate the *energy* between the previous decoder hidden state and the encoder hidden states.

The weighted sum of the energy across all encoder hidden states. These weights tell how much we should attend to each token in the source sequence. The parameters are initialized randomly but learned with the rest of the model via backpropagation.

**Decoder:** The decoder contains the attention layer, the encoder hidden states, and returns attention vector. this attention vector is used to create a weighted source vector, which is a weighted sum of the encoder hidden states. The embedded input word, the weighted source vector, and the previous decoder hidden state are then passed into the decoder RNN. We then pass them through the linear layer to make a prediction of the next word in the target sentence.

**Seq2Seq Model:** The encoder RNN and decoder RNN don’t have the same hidden dimensions, however the encoder has to be bidirectional. The encoder returns both the final hidden state. The outputs tensor holds predictions, and the source sequence is fed into the encoder to receive a context vector. Initial decoder hidden state is set to context vector and a batch of <sos> tokens are used as first input. In decoder loop, receive a prediction and new hidden state and decide if going to teacher force or not.

# **Experiment 4: Packed Padded Sequences, Masking, Inference and BLEU**

In this Experiment 4, packed padded sequences and masking - to the model from the previous notebook. Packed padded sequences are used to tell our RNN to skip over padding tokens in our encoder. Masking explicitly forces the model to ignore certain values, such as attention over padded elements. Both techniques are commonly used in NLP.

Finally, we'll use the BLEU metric to measure the quality of our translations.

**Encoder**:The changes here all within the forward method. It now accepts the lengths of the source sentences as well as the sentences themselves. After the source sentence (padded automatically within the iterator) has been embedded, we can then use pack\_padded\_sequence on it with the lengths of the sentences. packed\_embedded will then be our packed padded sequence. This can be then fed to our RNN as normal which will return packed\_outputs We then unpack our packed\_outputs using pad\_packed\_sequence which returns the outputs and the lengths of each, which we don't need.

**Attention:** The attention module is where we calculate the attention values over the source sentence. Previously, we allowed this module to "pay attention" to padding tokens within the source sentence. However, using masking, we can force the attention to only be over non-padding elements. The forward method now takes a mask input. This is a [batch size, source sentence length] tensor that is 1 when the source sentence token is not a padding token, and 0 when it is a padding token

**Decoder:**The decoder only needs a few small changes. It needs to accept a mask over the source sentence and pass this to the attention module. As we want to view the values of attention during inference, we also return the attention tensor.

**Seq2Seq:** We use the pad token index to create the masks, by creating a mask tensor that is 1 wherever the source sentence is not equal to the pad token. This is all done within the create\_mask function.The sequence lengths as needed to pass to the encoder to use packed padded sequences.

**Experiment 5 - Convolutional Sequence to Sequence Learning**

This model is drastically different to the previous models. There are no recurrent components used at all. Instead, it makes use of convolutional layers, typically used for image processing. The model is made of an encoder and decoder. The encoder *encodes* the input sentence, in the source language, into a *context vector*. The decoder *decodes* the context vector to produce the output sentence in the target language.

A small filter (kernel size of 3) and a high number of layers (5+) is beneficial to be used. So we have used 10 conv. blocks in both encoder and decoder

**Experiment 6 - Attention is All You Need**

The Transformer does not use any recurrence and model is entirely made up of linear layers, attention mechanisms and normalization. We use a learned positional encoding, Adam optimizer with a static learning rate and do not use label smoothing.

**Encoder and Encoder Layer** It produces a sequence of context vectors. Tokens pass through embedding layer and a positional embedding layer. Input is position of the token within the sequence. The token and positional embeddings are elementwise summed together to get a vector. The combined embeddings are then passed through encoder layers to get context vector.

The source sentence and its mask are passed into multi-head attention layer and dropout layer, then a residual connection is applied and passed through a [Layer Normalization](https://arxiv.org/abs/1607.06450) layer. The multi head attention layer applies attention over itself instead of another sequence, hence we call it self-attention.

### Multi Head Attention Layer: The Transformer uses scaled dot-product attention, where the query and key are combined by taking the dot product between them, then applying the softmax operation and scaling. We then re-combine the heads, so that each hidden\_dim pays attention to ℎ different concepts.

### Decoder and Decoder Layer: Decoder takes encoded representation of the source sentence and convert it into predicted tokens in the target sentence. has two multi-head attention layers. It uses positional embeddings and combines them with the scaled embedded target tokens, followed by dropout. The number of layers in the encoder does not have to be equal to the number of layers in the decoder. Self-attention is done using by using the decoder representation so far as the query, key and value. The encoder attention is how we feed the encoded source sentence into decoder. In this multi-head attention layer, the queries are the decoder representations, and the keys and values are the encoder representations.

### Seq2Seq and training model: Encapsulates the encoder and decoder. The source mask is created by checking where the source sequence is not equal to a pad token. It is 1 where the token is not a pad token and 0 when it is. It is then unsqueezed so it can be correctly broadcast when applying the mask to the energy. The target mask is a diagonal matrix where the elements above the diagonal will be zero and the elements below the diagonal will be set to whatever the input tensor is. This shows what each target token (row) looks at (column). The masks are used with the encoder and decoder to get our predicted target sentence along with the decoder's attention.

### Results

### The validation loss and perplexity of each model is evaluated and compared below using the test data. A low value of perplexity indicates the performance of the model

|  |  |  |
| --- | --- | --- |
| Experiment | Validation Loss | Perplexity |
| Experiment 1 | 2.714 | 15.085 |
| Experiment 2 | 2.682 | 14.611 |
| Experiment 3 | 2.389 | 10.899 |
| Experiment 4 | 2.351 | 10.498 |
| Experiment 5 | 1901936.373 | Inf |
| Experiment 6 | 1.044 | 2.841 |

### Findings:

### The experiment 5 where CNN model is used to analyze sequential text data is giving a high value of perplexity indicating that its performance is not comparable to that of other models that we have used for machine translation

### From the perplexity values of each model, experiment 6 gave the lowest value which is 2.841. Since the evaluation of the models is done based on their respective perplexity values, model built in experiment 6 is the best model for machine translation

### BLEU Scores

|  |  |
| --- | --- |
| Experiment | BLEU SCORE |
| Experiment 4 | 33.32 |
| Experiment 6 | 42.16 |

### Comparing the best model with SOTA language models to translate French to English

### Referring to the link below which explains the performance of different SOTA models like

### <https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french>

|  |  |
| --- | --- |
| Model | BLEU |
| [Transformer+BT (ADMIN init)](https://paperswithcode.com/paper/very-deep-transformers-for-neural-machine) | 46.4 |
| [Noisy back-translation](https://paperswithcode.com/paper/understanding-back-translation-at-scale) | 45.6 |
| [mRASP+Fine-Tune](https://paperswithcode.com/paper/pre-training-multilingual-neural-machine) | 44.3 |
| Transformer (Admin Init) | 43.8 |
| MUSE (Parallel multi-scale attention) | 43.5 |

From the experiments, multi head attention model seems to give the best performance (BLEU 42.16)