Comparison of Attention Functions In NMT

Authored by: Dang Yen Trang

Contents

Introduction

In this report is the progress, explanation for a project focused on the comparison of different attention functions employed in the various models of MT. Since attention has become a mechanism crucial in any state-of-the-art translation model and even been applied to many other tasks, I believe it is best to spend a few weeks familiarizing with its theory and many variations.
The code and data for this project had been uploaded to Github:
https://github.com/Jirachiii/AttentionNMT.git

1, Project Overview

1.1, Project Objective

In machine translation, attention mechanisms are used to enable the model to focus on relevant parts of the source text while generating the translation. However, not all attention mechanisms are created equal, and different types of attention functions can offer varying benefits depending on the nature of the task and the data being used. For instance, some attention functions may be better suited for capturing long-range dependencies, while others may be more efficient at handling rare or out-of-vocabulary words. Therefore, it is important to explore and develop different attention functions to improve the accuracy and efficiency of machine translation systems.

With such different methods in calculating attention for MT model, one can't help but wonder if there exists any discrepancy in their practical performance. Which is where the objective question for this project came from:

- ➔ Which attention function can return the best translation model given the same amount of training?

1.2, General Approach

Four of the most well-known attention functions were investigated during the course of this project:

- **Bahdanau attention function**: Introduced in *Neural Machine Translation by Jointly Learning to Align and Translate[1].* The first attention function, used alongside traditional RNN architectures.
- **Luong global attention function**: Introduced in *Effective Approaches to Attention-based Neural Machine Translation[2].* The first dot-product attention function, also used alongside traditional RNN architectures.
- **Multi-head attention function**: Introduced in *Attention Is All You Need[3].* A version of dot-product attention function scaled on key/query size. Used in transformer architecture, a network dispensing with recurrence and convolutions entirely.
- **Average attention function**: Introduced in *Accelerating Neural Transformer via an Average Attention Network[4].* A slight variation of multi-head attention transformer that aims to speed up the inference process.

Seeing how the four functions are deployed on two different architectures, they will be paired up on the criterion that:

➔ Two functions in a pair run on the same architecture, so they can be compared fairly on the same network configuration.

➔ The two pairs are: Bahdanau attention function and Luong attention function; Multi-head attention function and Average attention function.

➔ Each pair of attention functions will be given the same architecture, hardware, dataset, and optimizer, we will observe their translation performance after being trained for the same number of iterations/epoches.

1.3, Evaluation Criteria

- **BLEU:** BLEU score is the default method of assessing translation quality. The BLEU score compares a sentence against one or more reference sentences and tells how well does the candidate sentence matched the list of reference sentences. It gives an output score between 0 and 100. A BLEU score of 100 means that the candidate sentence perfectly matches one of the reference sentences.
- **Perplexity**: Perplexity is a metric commonly used to evaluate the performance of language models. It measures how well the model is able to predict the next

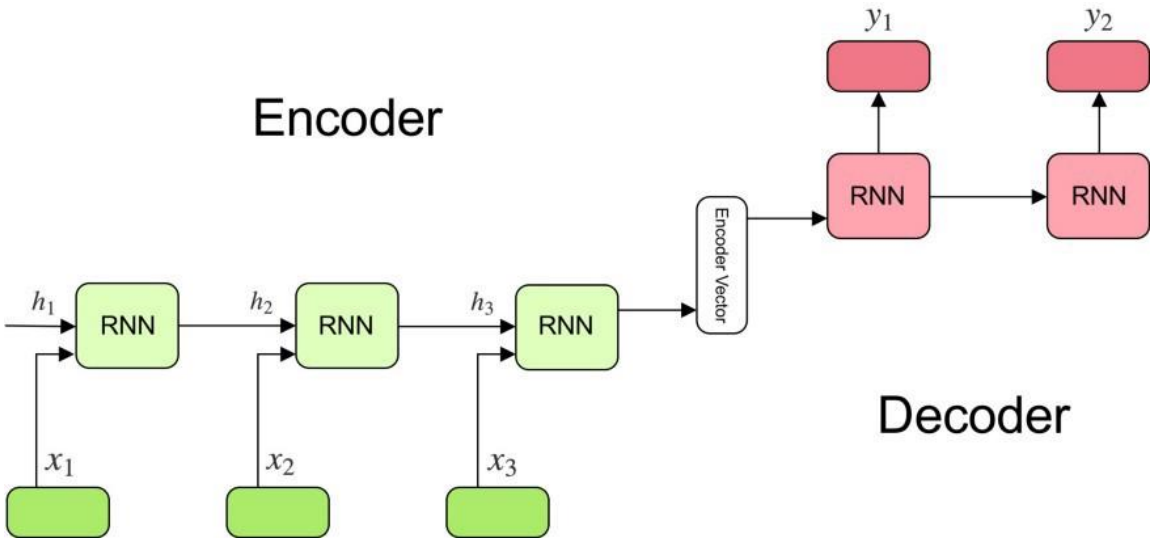
word in a given sequence of words. In the context of machine translation, perplexity is calculated by comparing the probability distribution of the target language generated by the model with the actual target language. A lower perplexity score indicates that the model is able to predict the correct word with higher accuracy, and hence is considered to be performing better.

- **Accuracy**: Exactly as it says. Accuracy measures how many percent of the words generated by the translation model matches with the target sentence.
- **Cross Entropy Loss**: Cross-entropy loss is a commonly used loss function in machine learning for classification tasks. It measures the difference between the predicted probability distribution and the actual probability distribution. In other words, it measures how well the model is able to predict the correct class. The cross-entropy loss is calculated by taking the negative log of the predicted probability of the correct class. The lower the value of the cross-entropy loss, the better the model's performance.
- **Training Time**: Total time needed for the model to train for the number of iterations or epoches specified. A shorter training time is more desirable, as this means the model requires less computation for each iteration.
- **Inference Time:** Total time needed for the model to translate the entirety of the test set. A shorter inference time is better than a long inference time for the same reason why we would want a short training time.

2, Background Theory

2.1, The Encoder-Decoder Architecture

Encoder-decoder architecture is a popular approach for machine translation. It consists of two main components: an encoder and a decoder.



A typical encoder-decoder network

The encoder processes the input source sequence and produces a fixed-length representation, often called a context vector, that contains all the relevant information needed to generate the target sequence. The decoder takes the context vector as input and generates the target sequence one token at a time, predicting the next token based on the previously generated tokens and the context vector.

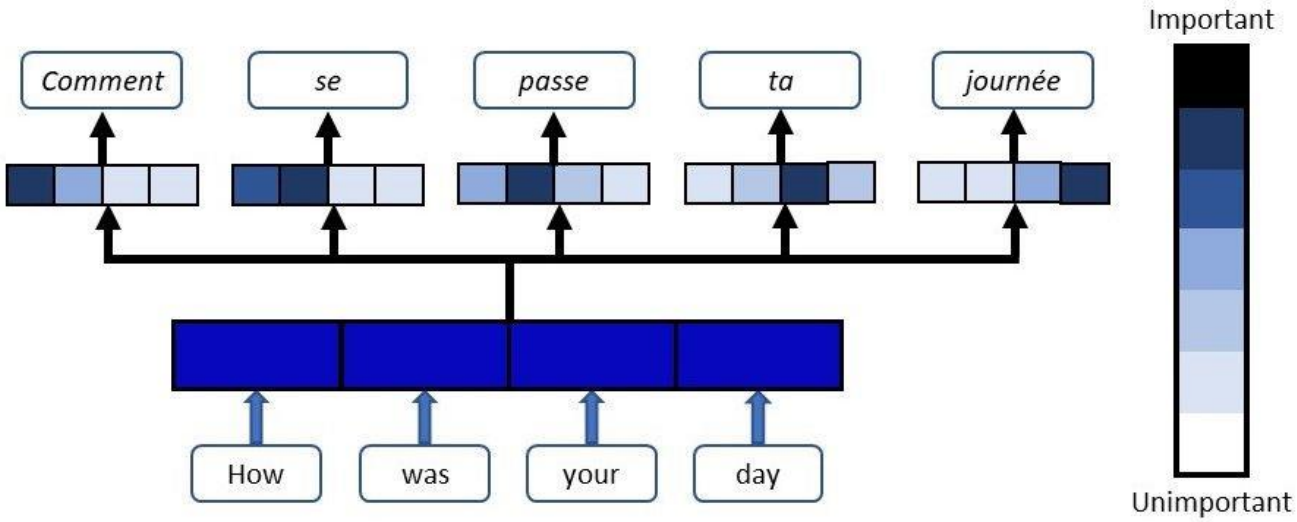
The encoder and decoder are typically implemented using neural networks, with the encoder usually being a bidirectional recurrent neural network and the decoder being a unidirectional recurrent neural network.

- ➔ This is also the RNN network architecture that we will use to run the first two attention functions: Luong and Bahdanau

Earlier works have an encoder neural network that reads and encodes a source sentence into a fixed-length context vector connected to a decoder that outputs a translation conditioned on the encoded context vector. However, a potential issue with this fixed context vector approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector, which will prove to be more diffcult the longer the source sentence is. That's where attention mechanism comes in.

2.2, General Attention Function

In machine translation, attention is a mechanism that enables the model to selectively focus on different parts of the source sentence during the decoding process. Rather than encoding the entire input sequence into a fixed-length context vector, attention allows the model to compute a weighted sum of the encoder hidden states, where the weights are learned dynamically based on the current decoder state. This allows the model to pay more attention to the most relevant parts of the input sequence for generating the current target word, leading to better translation quality.



An attention map will be different based on the current state of the decoder. Attention is more focused on the parts of the input that is relevant to the word being translated.

In *Attention Is All You Need[3]*, Vaswani et al. explains the general form of an attention function: Given a set of vectors Value(V), Query(Q), and Key(K), attention is a technique to retrieve a **selective summary** of the **information contained in the values**, where the **compatibility between key and query** determines the amount to take. Attention functions always involve three steps:

1, Compute attention score e, which denotes K's compatibility with Q.

2, Take softmax to get attention distribution:

$$\alpha = softmax(e)$$

3, Use weighted sum to take selective summary from V:

$$a = \sum_{i=1}^{N} \alpha_i . V_i$$

2.3, Bahdanau Attention Function

Steps to calculate Bahdanau attention are as follow:

1, We have **concatenation of the forward and backward hidden states in the bi-directional encoder** $h_1, h_2, \dots, h_N$.

2, At time step i, we have the **previous decoder hidden state** s_{i-1}

3, Attention score e_i of step i consists of N components:

$$e_{ij} = a(s_{i-1}, h_j)$$

Where a is a **feedforward neural network** trained to score how well the inputs around position j and the output at position i match.

4, Take softmax to get attention distribution for this step:

$$\alpha^i = softmax(e_i)$$

5, Context vector c_i for step i is then computed as the weighted sum of the encoder hidden states:

$$c_i = \sum_{j=1}^{N} \alpha_j^i . h_j$$

2.4, Luong General Attention Function

Steps to calculate Luong general attention are as follow:

1, We have **encoder hidden states at the top layers** $h_1, h_2, \dots, h_N$

2, At time step I, we have the **current decoder hidden state** s_i

3, Attention score e_i of step i consists of N components:

$$e_{ij} = a(s_i, h_j) = s_i^T . h_j$$

Where alignment function a is a **dot product** of encoder hidden state and current decoder hidden state.

4, Take softmax to get attention distribution for this step:

$$\alpha^i = softmax(e_i)$$

5, Context vector c_i for step i is then computed as the weighted sum of the encoder hidden states:

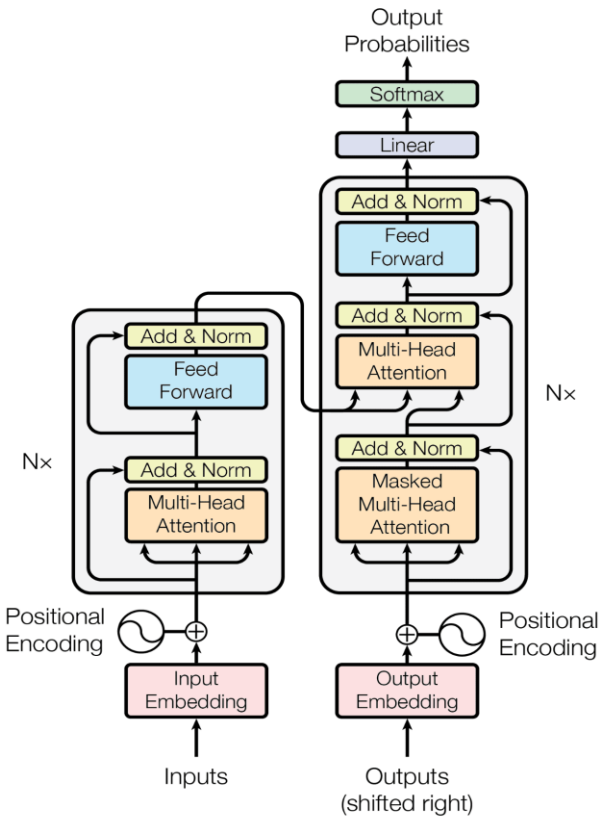
$$c_i = \sum_{j=1}^{N} \alpha_j^i . h_j$$

2.5, Comparison Between Bahdanau Attention and Luong Attention

A quick comparison shows us the main differences between these two attention functions:

Bahdanau Function	Luong Function
Use decoder previous hidden state	Use decoder current hidden state
Use a feedforward neural network to calculate alignment score	Use dot product to calculate alignment score
Use bi-directional encoder	Use stacking LSTM layers

2.6, Transformer Architecture



Transformer architecture

The transformer architecture is a type of neural network commonly used in natural language processing tasks such as machine translation and language modeling. Unlike traditional sequence-to-sequence models that use recurrent neural networks, the

transformer is based on a self-attention mechanism that allows it to process input sequences in parallel, making it more efficient and scalable. Transformer also has a encoder-decoder architecture, except all the recurrent connections have been replaced by the attention modules.

Transformer employs both context attention and self-attention:

- Self-attention is a mechanism that allows the a model to attend to different parts of the input sequence when computing the representation of a given (input) token. In self-attention, the keys,queries, and values all come from the same place that is the input sequence. Self-attention is used in the encoder layers of the transformer.
- Context attention, on the other hand, is a mechanism that allows a model to attend to different parts of the source sentence when generating each target word in the translation. Context attention is used in the decoder layers of the transformer.

2.7, Multi-head Attention Function

Multi-head attention is built upon a variation of the Luong Attention Function. This variation is called by its author 'Scaled Dot-Product Attention'. Scaled dot-product attention is calculated following the steps below:

1, Attention score is calculated as dot product of Q and K

$$e = QK^T$$

2, Attention score is the scaled on a factor of $\frac{1}{\sqrt{d_k}}$, where d_k is the dimension of K.

3, Take softmax to get attention distribution:

$$\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

4, Attention output is calculated as weighted sum of V values:

$$\text{Attention}(Q,\ K,\ V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

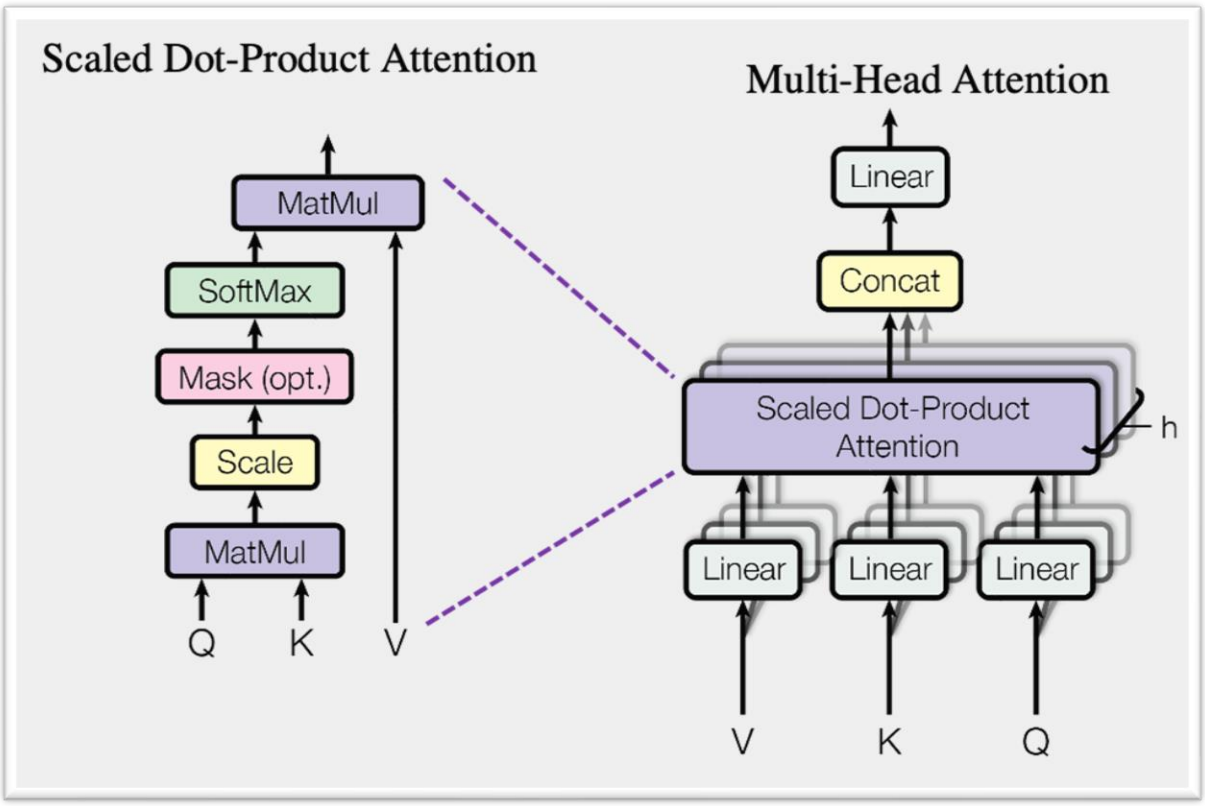
From there, in place of a single attention function with keys, values and queries, multi-head attention function linearly projects the queries, keys and values h times with different linear projections.

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K. VW_i^V)$$

The attention function on each of these attention head is done in parallel, shortening the computation time compared to previous variations.

These attention heads are concatenated and once again projected, resulting in the final attention values.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O$$



The relationship between scaled dot-product attention and multi-head attention

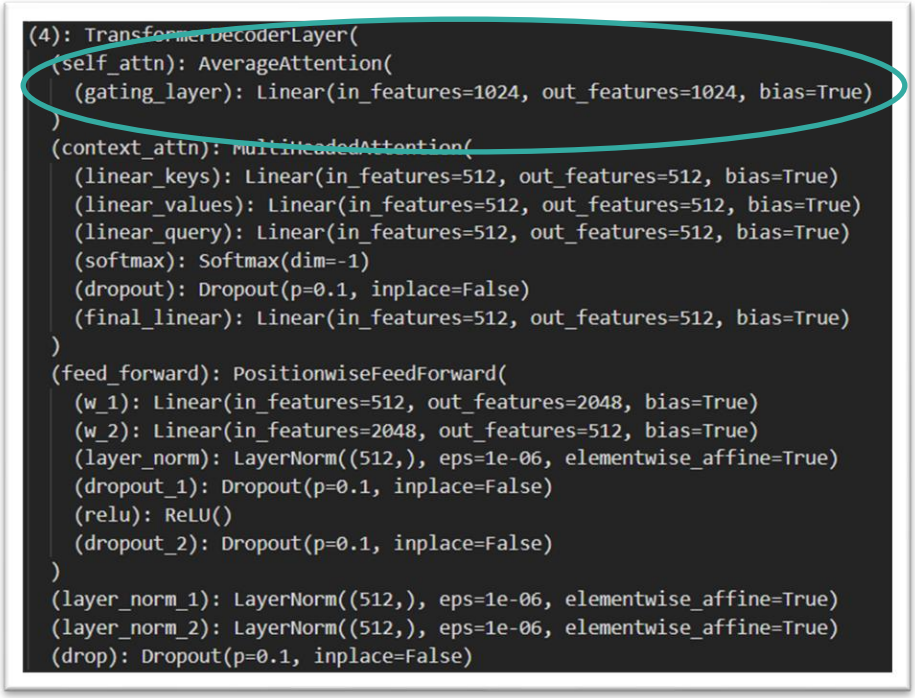
2.8, Average Attention

The average attention network introduced in *Accelerating Neural Transformer via an Average Attention Network[4]* is pretty much identical to the Transformer introduced above. The only difference is that it replaces the original dynamically computed attention weights by the self-attention network in the decoder of the neural Transformer with simple and fixed average weights.

Given input layer $y = \{y_1, y_2, \dots, y_m\}$, average attention generate context-sensitive representation for each input embedding as follows:

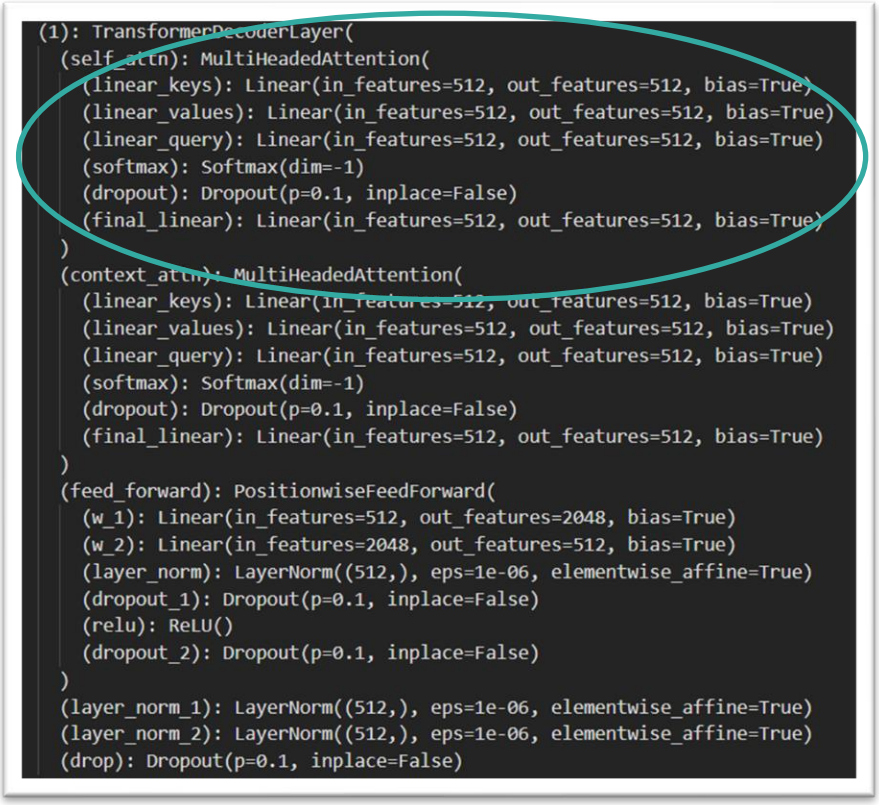
$$g_j = FFN(\frac{1}{j}\sum_{k=1}^{j} y_k)$$

2.9, Comparison between Multi-head Attention Network and Average Attention Network



Average attention decoder in OpenNMT

The implimentation of an average attention module only consists of a linear feedforward layer that will ensure simple and quick computation during encoding



Multi-head attention decoder in OpenNMT

The implementation of a multi-head attention module is much more complicated, consisting of three linear projections for key/query/value, a softmax layer, a dropout layer, and a final linear layer.

3, Expereimental Setup

3.1, Dataset

The dataset used for this project was collected by The Tatoeba Project, and can be found on OPUS[5].

The language pair chosen for experimentation was English-German, for 2 reasons:

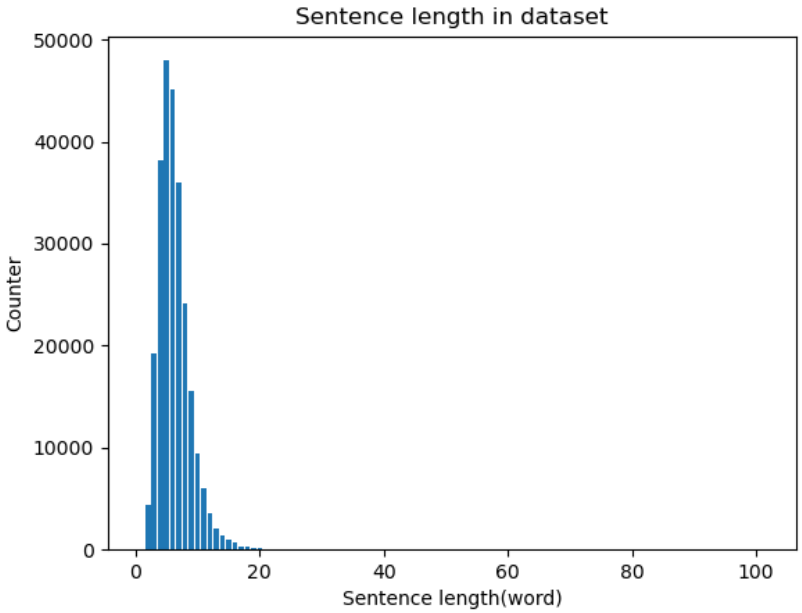
- They belong to the same language family, have similar grammar and sentence structure. This means that the task of translating between these two languages won't be too difficult.

- Rich in resources. Easy to find large amount of parallel text of high quality, ensuring that the final translation models won't be affected negatively by bad training data.

The dataset has 255817 parallel sentences, which was divided into training, validation and test set:

- 80% into train set
- 10% into valid
- 10% into test set

To gain a better understanding of the data we're working with, below is a histogram showing the distribitution of sentence lengths in this dataset:



The histogram shows that the longest sentence in the dataset has up to ~100 words, but most sentences only have between 1 and 20 words.

3.2, Model Implementation

The model implementations for this project was by OpenNMT-py[6].

3.3, Hyperparameters

General hyperparameters and optimizer hyperparameters are shared between all four models.

General Options	
Training Steps	**40000, run validation every 2000 steps**
Batch Size	**256 for training, 8 for validating**
Optimizer Options	
Optimizer Function	**Adam**
Learning Rate	**0.001**
Learning Rate Decay (for Luong attention)	**Start_decay_at: 16000; Decay_rate: 0.5; Decay_every: 4000**
Initialization Method	**Glorot init is required for transformer, none for RNN**

Most of the hyperparameters for model architecture was taken directly from BaseTransformer[3].

Model Architecture	
Word Vector Size	**512**
Hidden State Size	**512**
Dropout	**0.1 for all layers, even attention layers**
Transformer Layers	**6**
Attention Heads	**8**
Position-wise Feed Forward Network Size	**2048**
RNN Types	**Bi-layer GRU with the encoder being bi-directional**

3.4, Evaluation metrics and How to find them

To better understand the value of the evaluation metrics, here are the formula for their computation, as implemented by OpenNMT-py:

- **BLEU:**

$$BLEU = \text{BP}(\prod_{i=1}^{4} precision_i)^{(\frac{1}{4})}$$

With the Brevity Penalty and precision defined as:

$$BP = \min\left(1, \exp\left(1 - \frac{ref}{can}\right)\right)$$

$$precision_i = \frac{\sum_{sentence \in cand} \sum_{i \in sentence} \min\left(m^i_{cand}, m^i_{ref}\right)}{w^i_t}$$

Where:

- m^i_{cand} is the count of i-gram in candidate matching the reference translation.
- m^i_{ref} is the count of i-gram in the reference translation.
- w^i_t is the total number of i-grams in candidate translation.

- **Cross Entropy Loss:**

$$\text{CrossEntropyLoss} = \frac{\text{Total CrossEntropyLoss}}{\text{Number of words}}$$

- **Perplexity:**

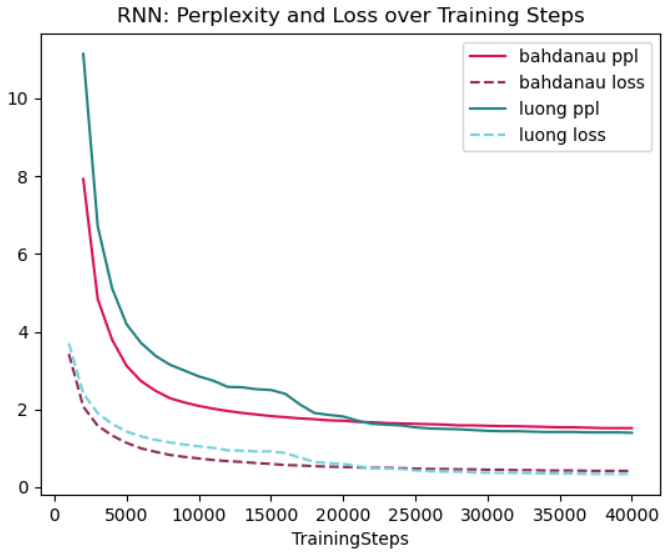
$$\text{Perplexity} = \exp\left(\min\left(\frac{\text{CrossEntropyLoss}}{\text{Number of words}}, 100\right)\right)$$

- **Accuracy:**

$$\text{Accuracy} = \frac{\text{Number of correctly translated words}}{\text{Number of translated words}}$$

4, Result

4.1, Luong vs. Bahdanau Attention

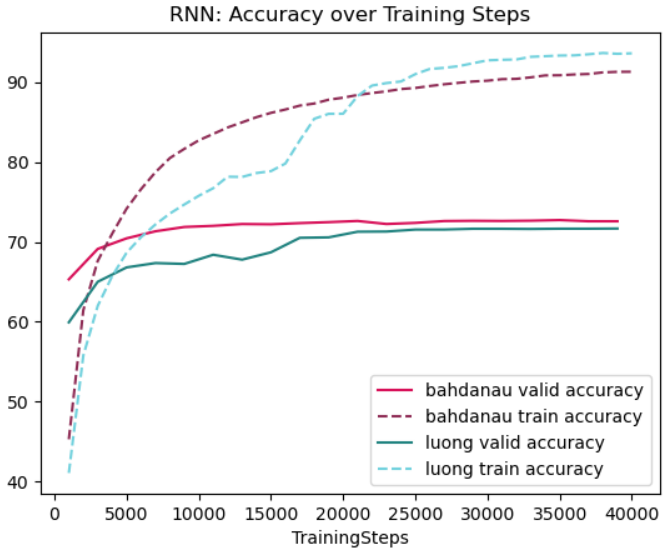


Luong attention statistics:

- Lowest perplexity: **1.4**
- Lowest cross entropy loss: **0.34**

Bahdanau attention statistics:

- Lowest perplexity: 1.52
- Lowest cross entropy loss: 0.42

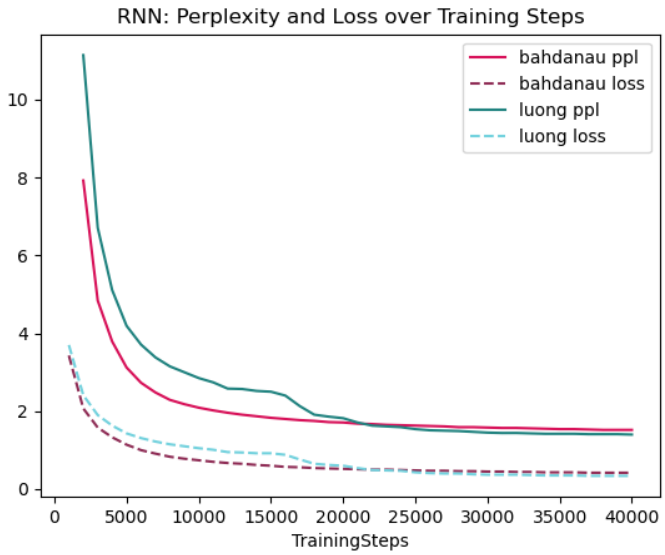


Luong attention statistics:

- Highest accuracy: **93.67** on train set, 71.6772 on valid set

Bahdanau attention statistics:

- Highest accuracy: 91.32 on train set, **72.7389** on valid set



Luong attention statistics:

- Lowest perplexity in train set: **1.4**
- Lowest perplexity in valid set: 5.23807

Bahdanau attention statistics:

- Lowest perplexity in train set: 1.52
- Lowest perplexity in valid set: **4.72036**

Training and Inference time comparison:

- **Bahdanau:**
  - Training time: 5221
  - Inference time: 619
- **Luong:**
  - Training time: **4675**
  - Inference time: **584**

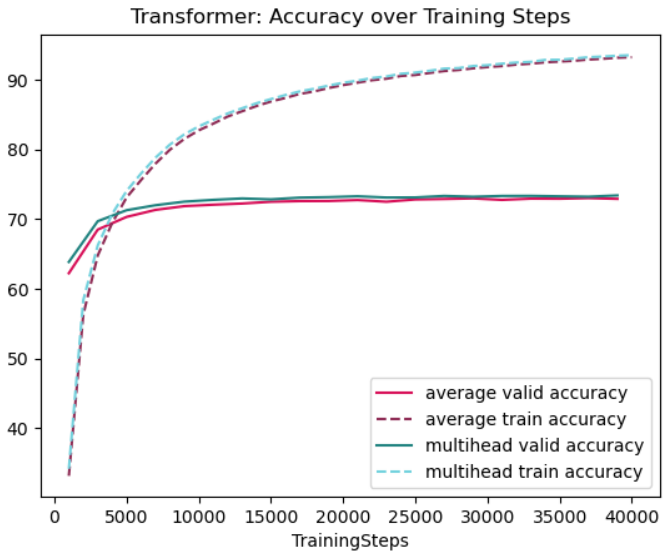
4.2, Multi-head vs Average Attention

Multi-head attention statistics:

- Lowest perplexity: **1.31**
- Lowest cross entropy loss: **0.27**

Average attention statistics:

- Lowest perplexity: 1.33
- Lowest cross entropy loss: 0.28

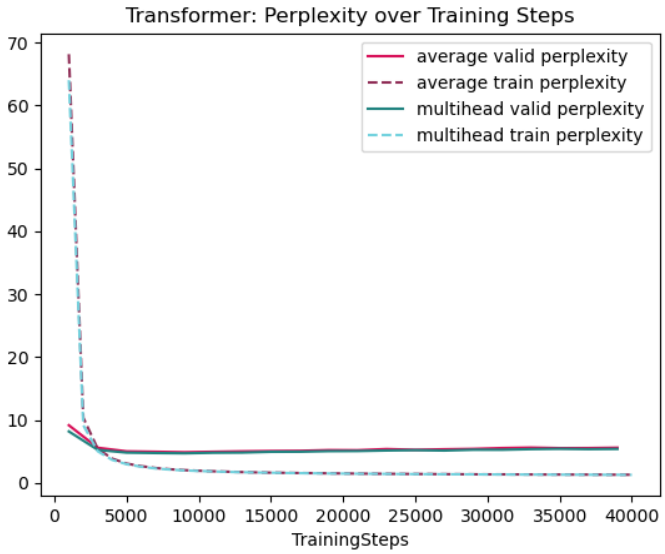


Multi-head attention statistics:

- Highest accuracy: **93.54** on train set, **73.4132** on valid set

Average attention statistics:

- Highest accuracy: 93.24 on train set, 73.0303 on valid set



Multi-head attention statistics:

- Lowest perplexity in train set: **1.31**
- Lowest perplexity in valid set: 4.90823

Average attention statistics:

- Lowest perplexity in train set: 1.33
- Lowest perplexity in valid set: **4.7336**

Training and inference time comparison:

- **Multi-head:**
  - Training time: 7154
  - Inference time: 858
- **Average:**
  - Training time: **6824**
  - Inference time: **804**

4.3, BLEU Score

Training Steps	10000	20000	30000	40000
Bahdanau	**40.9**	**42.1**	**42.2**	**42.4**
Luong	**35.5**	**40.6**	**42.1**	**42.3**
Multi-Head	**40.7**	**42.1**	**42.3**	**42.6**
Average	**40.5**	**42.1**	**42.7**	**42.6**

5, Observations

- Luong attention model has better results on test set than Bahdanau attention model, but this is no longer true on validation set and final translation model.

➔ Luong attention model is more prone to overfitting than Bahdanau.

- Multi-head attention model and Average attention model returns almost identical statistics during both training and validating process.

➔ However, taking into consideration the resultant BLEU scores, we can say that average attention yields better translators than multi-head attention.

- Luong attention model runs faster than Bahdanau attention model both during training and translating.

➔ This is to be expected, as multiplicative attention models such as Luong can be implemented using highly optimized matrix multiplication code.

- Average attention model runs faster than Multi-head attention model both during training and translating.

➔ This is according to theoretical calculation, as average attention model can be fully parallelized thanks to its simple computation

REFERENCES

[1] https://arxiv.org/abs/1409.0473
[2] https://arxiv.org/abs/1508.04025
[3] https://arxiv.org/abs/1706.03762
[4] https://arxiv.org/abs/1805.00631
[5] https://opus.nlpl.eu/Tatoeba.php
[6] https://opennmt.net/OpenNMT-py/