# Translator

Neural Machine Translation With four different models’

# Introduction

Businesses all around the world rely on translation services as an essential component of their daily operations. Translation helps organizations function easily and efficiently across international borders results in increased sales.

An increase in humans' dependence on computer-aided systems has pushed me to work on more effective communication technologies that simulate interactions as well as natural languages translation. Over the last few years, translation has become a hotly debated academic topic. So, I'll primarily concentrate on four use case algorithms in this research namely: Transformer, Seq2Seq with Attention, BERT, and Statistical Machine Translation techniques.

The translation is required for the transmission of information, knowledge, and ideas. Effective and empathetic communication across various cultures is critical. Let us begin with a single model. In machine translation software, it would take a sentence in one language and translate it into another. Because the Seq2Seq Attention Model uses both Recurrent Attention and Transformer, it may quickly learn the most likely decoding sequence in the target language. The capabilities for fine-tuning data to a given linguistic environment and resolving problems encountered on a daily basis. This deep learning model is extremely intriguing in terms of delivering a high-performing model.

Machine translations are not good for lengthy periods. The accuracy percentage of automated translations ranges from 70% to 90%, but seldom exceeds 100%. It's unusual to come across a digital gadget that can properly translate each word. Human labor is recommended for the highest accuracy. Although human translation appears to outperform artificial intelligence, it is far from perfect. The primary issue is the delayed speed. When compared to translation technologies, humans take far longer to produce translations. Human translation appears to be considerably more sophisticated than AI as the best option for translating context, although AI is improving every day. To achieve the ideal balance of accuracy and quality in translation, any good consumer of translation will consider both machine and human translating technologies.

#### **System Requirements:**

To build Translator, your system must meet the following minimum requirements:

|  |  |
| --- | --- |
| **Processor:** | Minimum Intel i3 8TH Gen cocked at 2.5Ghz or AMD Ryzen 3(3200u), recommended Intel(R) Core(TM) i5 8TH Gen or AMD Ryzen 5(3600G) and above clocked at 3.33 GHz or equivalent. |
| **Memory:** | With 1 language pair: minimum 1 GB RAM, recommended 2GB (and 500 MB per additional language pair).  With several language pairs: min 8 GB , recommended 16GB (and 500 MB per additional language pair.) |
| **Disk Space:** | 25 GB (and 2.5 GB per additional language pair) |
| **Operating System**: | Windows 8 or 10 (32 or 64-bit)(future Windows 11 is also compatible) |
| **GPU:** | Minimum Nvidia GTX 980M (VRAM-4GB) or AMD Radeon RX580 , recommended Nvidia GTX 1060 (VRAM-6GB) or AMD Radeon RX5500 or equivalent. |

# Dataset

**WMT: Dataset** (WMT 2012)

Link: <https://nlp.stanford.edu/projects/nmt/>

The above website includes information on the most recent neural machine translation (NMT) dataset at the Stanford NLP group. They release a codebase, which provides cutting-edge outcomes in a variety of translation jobs, including English-German and English-Spanish, etc. In addition, we share the preprocessed data that we used to train NLP models, as well as a trained dataset that is easily useable with the codebase, to encourage reproducibility and transparency. This dataset is usually used in Statistical Machine Translation.

**OPUS: Dataset** (OPUS 100)

Link: <https://opus.nlpl.eu/>

OPUS is a growing collection of web-translated literature. The OPUS project aims to convert and align freely available internet data, add linguistic annotation, and offer the community with a publicly available parallel corpus. OPUS is built on open source technologies, and the corpus is likewise available as an open content package. They assembled the present collection using a variety of tools. All of the pre-processing is handled automatically. There have been no manual corrections.

**OPUS: Dataset** (TED2019)

Link: <https://opus.nlpl.eu/TED2019.php>

This dataset contains a crawl of nearly 4000 TED and TED-X transcripts from July 2019. The transcripts have been translated by a global community of volunteers (https://www.ted.com/participate/translate) to more than 100 languages.

The transcripts for the different languages have been sentence aligned to generate a parallel corpus that can be used to train machine translation systems.

# OPUS: Dataset (wiki v20210402)

Link: <https://opus.nlpl.eu/wikimedia.php>

Wikipedia translations published by the wiki foundation and their article translation system. The parallel data sets are published at <https://dumps.wikimedia.org/other/contenttranslation>

NEW: additional sentence alignment to avoid long segments in translation units

306 languages, 2,575 bitexts  
total number of files: 306  
total number of tokens: 918.05M  
total number of sentence fragments: 31.62.

# Models

I have chosen four distinct models in this project, notably: -

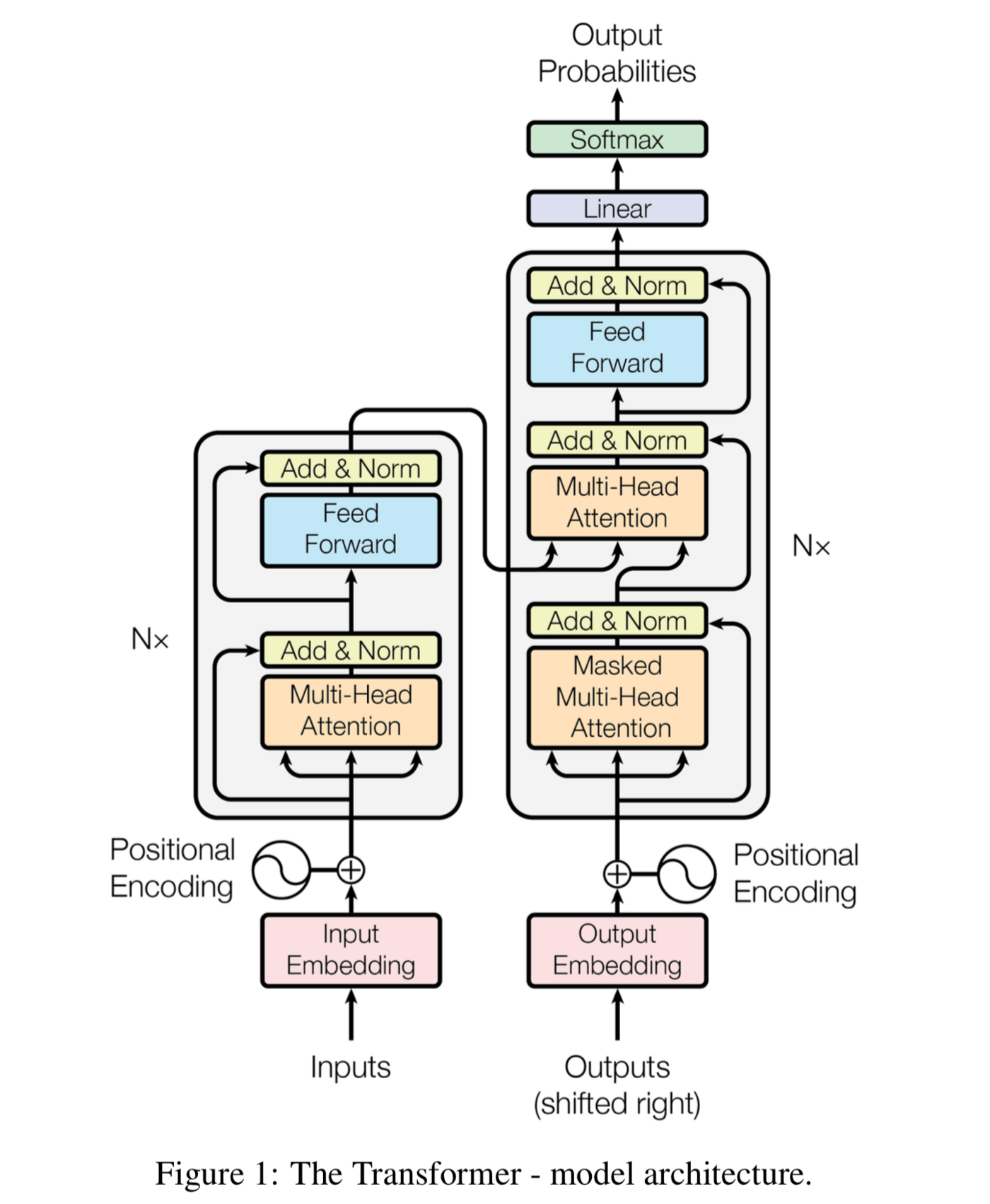
* Transformer (NLP)
* Sequence to Sequence with Attention
* Bidirectional Encoder Representations from Transformers
* Statistical Machine Translation

# Transformer:

The transformer uses a self-attention mechanism at every stage, which directly models the connections between every word and its place.

Like RNNs the Transformer is an architecture used to convert an encoder decoder from an existing Seq2Seq model but does not have a recurring network. It is a transformer (GRU, LSTM, etc.). But the transformer processes the whole input sequence at once and does not iterate word after word, unlike the RNNs.

Now let's dig deep into the transformer's design and see how it works. The Transformer architecture is seen in the diagram below. The encoder is on the left, while the decoder is on the right.



Stacks of Encoders and Decoders

Encoder: The encoder is made up of N = 6 identical layers. Each layer is divided into two sub-layers. The first is a multi-head self-attention mechanism, while the second is a basic, completely linked feed-forward network that is positionally coupled. We use a residual connection [11] to connect the two sub-layers, followed by layer normalization [1]. That is, each sub-output layer's is Layer Norm (x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. All sub-layers in the model, as well as the embedding layers, provide outputs with dimension model.

The decoder is also made up of a stack of N = 6 identical layers. The decoder inserts a third sub-layer, which conducts multi-head attention over the encoder stack's output, in addition to the two sub-layers in each encoder layer. We use residual connections surrounding each of the sub-layers, similar to the encoder, followed by layer normalization. We additionally alter the decoder stack's self-attention sub-layer to prevent positions from attending to succeeding positions. This masking, along with the fact that the output embeddings are offset by one place, ensures that predictions for location I can only be based on known outputs at positions less than.

Batching and Training Data

I trained using the standard OPUS English-German dataset, which has around 1.5 million sentence pairings. Sentences were encoded with byte-pair encoding, which has a shared source target vocabulary of about 37000 tokens. I utilized the much smaller OPUS English-Spanish dataset of 36M phrases and divided tokens into a 32000 word-piece vocabulary for English-French. Sentence pairs were grouped together based on their estimated sequence length. Each training batch included a collection of phrase pairings with roughly 25000 source and 25000 target tokens.

Hardware and Timetable

My models were trained on a single computer using NVIDIA GTX 1060 GPUs. Each training phase for our base models utilizing the hyperparameters given throughout the study took around 0.4 seconds. The basic models were trained for a total of 100 steps, or 1.5 hours. The step time for my models was 1.0 second. The models were trained for a total of 300 steps (3.5 hours).

I used the Adam optimizer with 1 = 0.9, 2 = 0.98, and = 109 parameters. It changed the learning rate throughout training using the formula: late = d 0.5 model min. This corresponds to raising the learning rate linearly for the first warmup steps training steps and then reducing it proportionately to the inverse square root of the step number after that.

Regularization

Residual Dropout by applying dropout to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks.

The Transformer achieves better BLEU scores in state-of-the-art models on the English-to-German, Spanish, Hindi, Chinese:

**Transformer table of Bleu Score:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Source Lang | Target Lang | Dataset | Blue Score |
| Transformers | En (English) | De (German) | Opus (Wiki) | 34.14 |
| Transformers | En (English) | Es (Spanish) | Opus (Wiki) | 31.0 |
| Transformers | En (English) | Zh (Chinese) | Opus (Wiki) | 31.26 |
| Transformers | En (English) | Hi (Hindi) | Opus (Wiki) | 26.1 |

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

# Seq2Seq (Sequence to Sequence):

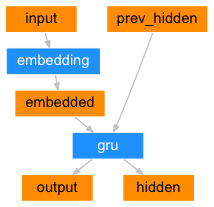
A Recurrent Neural Network, or RNN, is a network that operates on a sequence and uses its own output as input for subsequent steps.

A Sequence-to-Sequence network, also known as an Encoder Decoder network, is a model that consists of two RNNs, the encoder and decoder. The encoder reads an input sequence and produces a single vector, which the decoder reads to generate an output sequence.

Unlike sequence prediction with a single RNN, where every input correlate to an output, the seq2seq model is unconstrained by sequence length and order, making it suitable for language translation. Consider the sentence “I am not the black cat” → “No soy el gato negro”. Most of the words in the input sentence have a direct translation in the output sentence, but are in slightly different orders, e.g., “black cat” and “gato negro”. Because of the “Nació” construction there is also one more word in the input sentence. It would be difficult to produce a correct translation directly from the sequence of input words.

The Encoder's Function

A seq2seq network's encoder is an RNN that outputs a value for each word in the input phrase. The encoder generates a vector and a hidden state for each input word and utilizes the hidden state for the next input word.

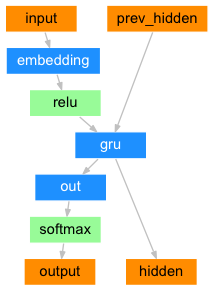


The Decoder's Function

The decoder is another RNN that takes the encoder's output vector(s) and generates a string of words to construct the translation.

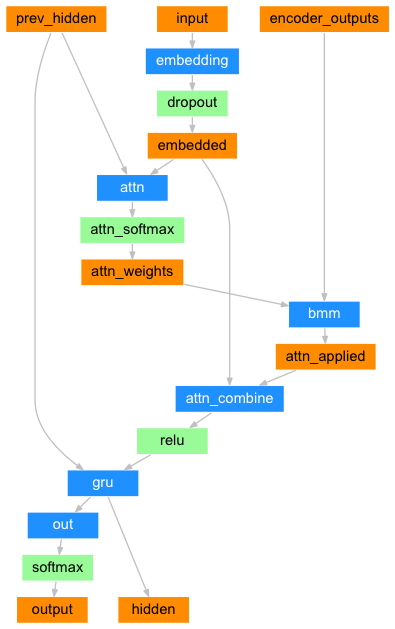
Decoder Simple

In the most basic seq2seq decoder, we just use the encoder's final output. Because it encodes context from the whole sequence, its final output is frequently referred to as the context vector. This context vector serves as the decoder's initial hidden state. The decoder is given an input token and a hidden state at each step of the decoding process. The context vector (the encoder's previous hidden state) is the initial input token, and the first hidden state is the start-of-string SOS> token.



Decoder of Attention

If just the context vector is transmitted between the encoder and decoder, that single vector must encode the whole phrase.For each step of the decoder's own outputs, the decoder network can "focus" on a different section of the encoder's outputs. We begin by calculating a set of attention weights. The encoder output vectors will be multiplied by these to form a weighted combination. The result (named attn applied in the code) should contain information about that particular section of the input sequence, assisting the decoder in selecting the appropriate output words.



Training Data Preparation

To train, model will require an input tensor (indexes of the words in the input phrase) and a target tensor for each pair (indexes of the words in the target sentence). This will append the EOS token to both sequences as we create these vectors.

Model Development

To train, passing the input text through the encoder, recording every output and the most recent hidden state. The decoder is then given the token as its first input and the encoder's final hidden state as its first hidden state.

“Teacher forcing” refers to the practice of using the actual target outputs as the next input rather than the decoder's guess as the next input. When using teacher forcing, it converges faster; however, when the trained network is exploited, it may exhibit instability.

The entire training procedure is as follows:

* Set a timer.
* Set up optimizers and criteria
* Make a set of training pairings.
* Begin with an empty losses array for graphing.

Evaluating and Training

With all of these assistance functions in place (it appears to be extra work, but it makes running numerous experiments easier), we can actually setup a network and begin training.

Please keep in mind that the input sentences were severely filtered. It can utilize relatively modest networks of 256 hidden nodes and a single GRU layer for this little dataset. We'll obtain some acceptable results after around 40 minutes on a Intel CPU and 21 minutes on a compatible GPU (GTX 1060).

Score Results

The Seq2Seq achieves better BLEU scores in state-of-the-art models on the English-to-German, Spanish, Hindi, Chinese:

**Seq2Seq table of Bleu Score:**

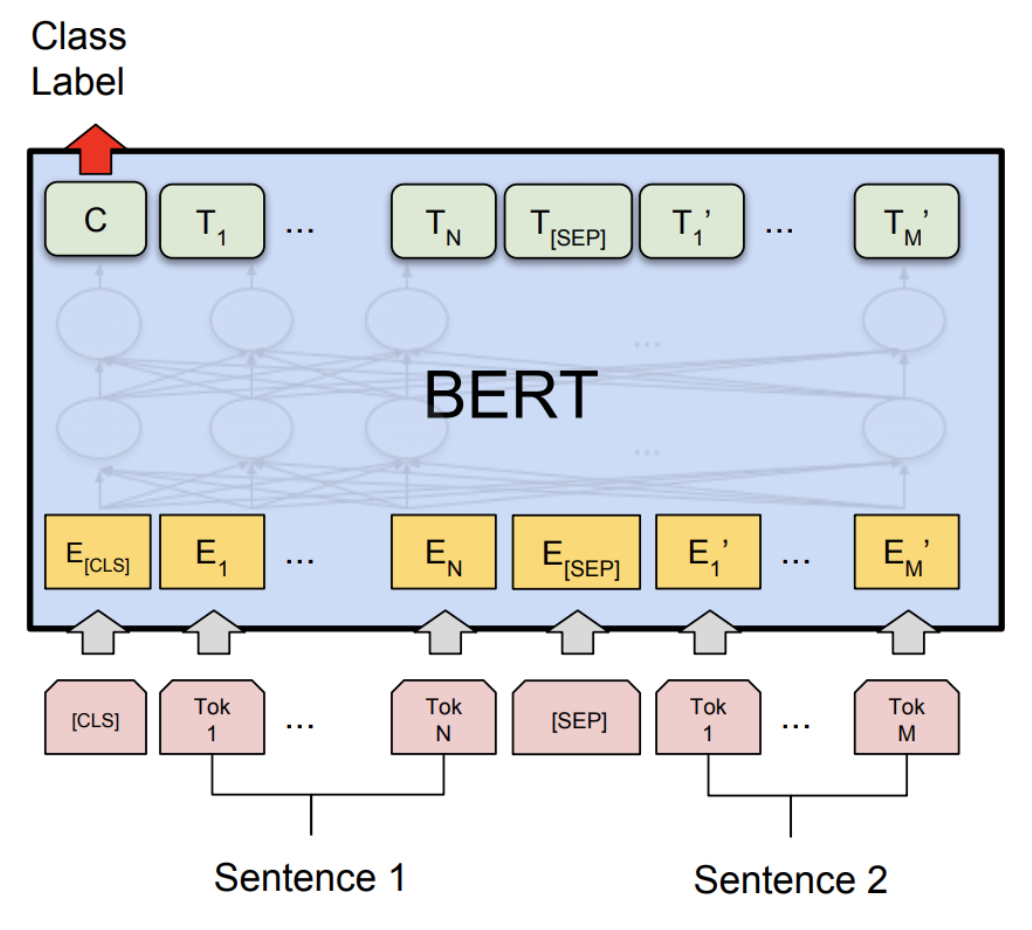
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Source Lang | Target Lang | Dataset | Blue Score |
| Seq2Seq | En (English) | De (German) | TED2019(OPUS) | 33.55 |
| Seq2Seq | En (English) | Es (Spanish) | TED2019(OPUS) | 36.9 |
| Seq2Seq | En (English) | Zh (Chinese) | TED2019(OPUS) | 31.12 |
| Seq2Seq | En (English) | Hi (Hindi) | TED2019(OPUS) | 29.36 |

A useful property of the attention mechanism is its highly interpretable outputs. Because it is used to weight specific encoder outputs of the input sequence, we can imagine looking where the network is focused most at each time step.

# BERT (Bidirectional Encoder Representations from Transformers):

BERT, or Bidirectional Embedding Representations from Transformers, is a new method of pre-training language representations that achieves state-of-the-art accuracy results on a wide range of popular Natural Language Processing (NLP) tasks, including question answering, text classification, and others. The original paper is available here(<https://arxiv.org/pdf/1810.04805.pdf>).

Dynamic PyTorch Quantization Support transforms a float model for weight and dynamic quantization for activations into a quantified version with statical int8 or float16 data types. When the masses are measured to int8, activations are dynamically (per batch). We have torch. quantization. Quantize dynamic API on PyTorch that substitutes for the specific modules with dynamic versions solely quantified by dynamic weight.



# 

Steps

* Load the pretrained model and tokenizer
* Load dataset
* Transform dataset into input (entails a minor model change)
* Train/finetune the model on our dataset
* Test the model

Setup:

Let's install PyTorch and Transformers here to start this. Furthermore, we will also install scikit-learn as we employ its included scarebleu calculation assist function.

Because we will be using the beta parts of the PyTorch, it is recommended to install the latest version of torch and torch vision.

The dataset setup:

We enter the OPUS100 data and disable it into a directory database before performing NPL operations.

Ex: dataset = load\_dataset('opus100en-de')

train\_dataset = dataset['train']

test\_dataset = dataset['test']

Fine-tune the BERT model

The spirit of BERT is to pre-train the language representations and then to fine-tune the deep bi-directional representations on a wide range of tasks with minimal task-dependent parameters, and achieves state-of-the-art results. In this tutorial, we will focus on fine-tuning with the pre-trained BERT model to classify semantically equivalent sentence pairs on NPL task.

Set global configurations

Here we set the global configurations for evaluating the fine-tuned BERT model before and after the dynamic quantization. We load the tokenizer and fine-tuned BERT sequence classifier model (FP32) from the configs. output dir.

Apply the dynamic quantization

We call torch.quantization.quantize\_dynamic on the model to apply the dynamic quantization on the Transformer BERT model. Specifically,

We specify that we want the torch.nn. Linear modules in our model to be quantized;

We specify that we want weights to be converted to quantized int8 values.

**quantized\_model** **=** **torch.quantization.quantize\_dynamic(**

**model,** **{torch.nn.Linear},** **dtype=torch.qint8**

**)**

print**(quantized\_model)**

The BERT model used in this is(bert-base-uncased) has a vocabulary size V of 30522. With the embedding size of 768, the total size of the word embedding table is ~ 4 (Bytes/FP32) \* 30522 \* 768 = 90 MB. So with the help of quantization, the model size of the non-embedding table part is reduced from 350 MB (FP32 model) to 90 MB (INT8 model).

Running this locally on a intel without quantization, inference (for all 408 examples in OPUS100 dataset) takes about 160 seconds, and with quantization and Nvidia GPU it takes just about 90 seconds. It summarize the results for running the BERT model .

Results:

We have 3% improvement in BLEU score accuracy after applying the post-training dynamic quantization on the fine-tuned BERT model on the NLP task. As compared to other model it better in its performance. The main difference is that we support the asymmetric quantization in PyTorch while that supports the symmetric BERT Model only.

**BERT table of Bleu Score:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Source Lang | Target Lang | Dataset | Blue Score |
| BERT | En (English) | De (German) | Opus100 | 35.95 |
| BERT | En (English) | Es (Spanish) | Opus100 | 36.37 |
| BERT | En (English) | Zh (Chinese) | Opus100 | 33.72 |
| BERT | En (English) | Hi (Hindi) | Opus100 | 32.16 |

Concluding how to convert a well-known state-of-the-art NLP model like BERT into dynamic quantized model. Dynamic quantization can reduce the size of the model while only having a limited implication on accuracy.

# Statistical Machine Translation

A basic English- four different languages using SMT system was developed utilizing parallel WMT train phrases. Use the development set to further refine assessed on a part of the test set provided

Probability of Word Translation Task

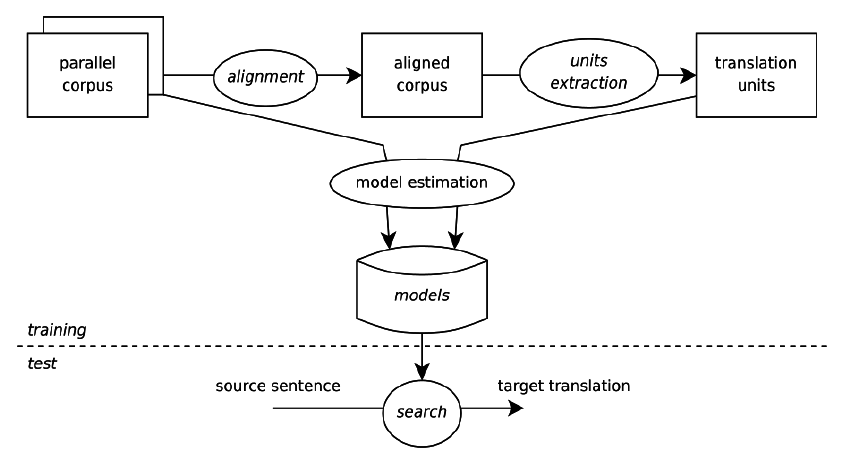
calculated from the scratch using conventional IBM Model 1 algorithm Maximization of standard expectation models 128 iterations trained algorithm. EM till convergence - under 0.005 probability changes. Taking into account 30,000 (German, Spanish, Hindi, Chinese) and English terms most commonly used till confluence.

Alignments for the first words of particular English sentences evaluated using the trained translation probabilities of IBM Model 1 following ‘HMM based Word Alignment in Statistical Translation’,

First order transition probabilities initialized using the technique outlined in ‘Word Alignment for Statistical Machine Translation Using Hidden Markov Models’ by Anahita Mansouri Bigvand.

Instead of capturing the absolute positions for word alignments, only the relative positions i.e., the jump widths are taken into consideration.

Language model for four languages with the intuition of generating coherent with the four languages texts: Bigram model with Laplace smoothing and backoff

 Evaluation

Since there were no Gold-Standard Annotations for alignment in parallel English-Other languages sentences, metrics such as accuracy alignment, alignment recall and error rate alignment cannot be calculated. Thus, scores from BLEU Using different smoothing approaches, were evaluated for automatic evaluation of machine translation by BLEU, Papineni et al., 2002.

Results

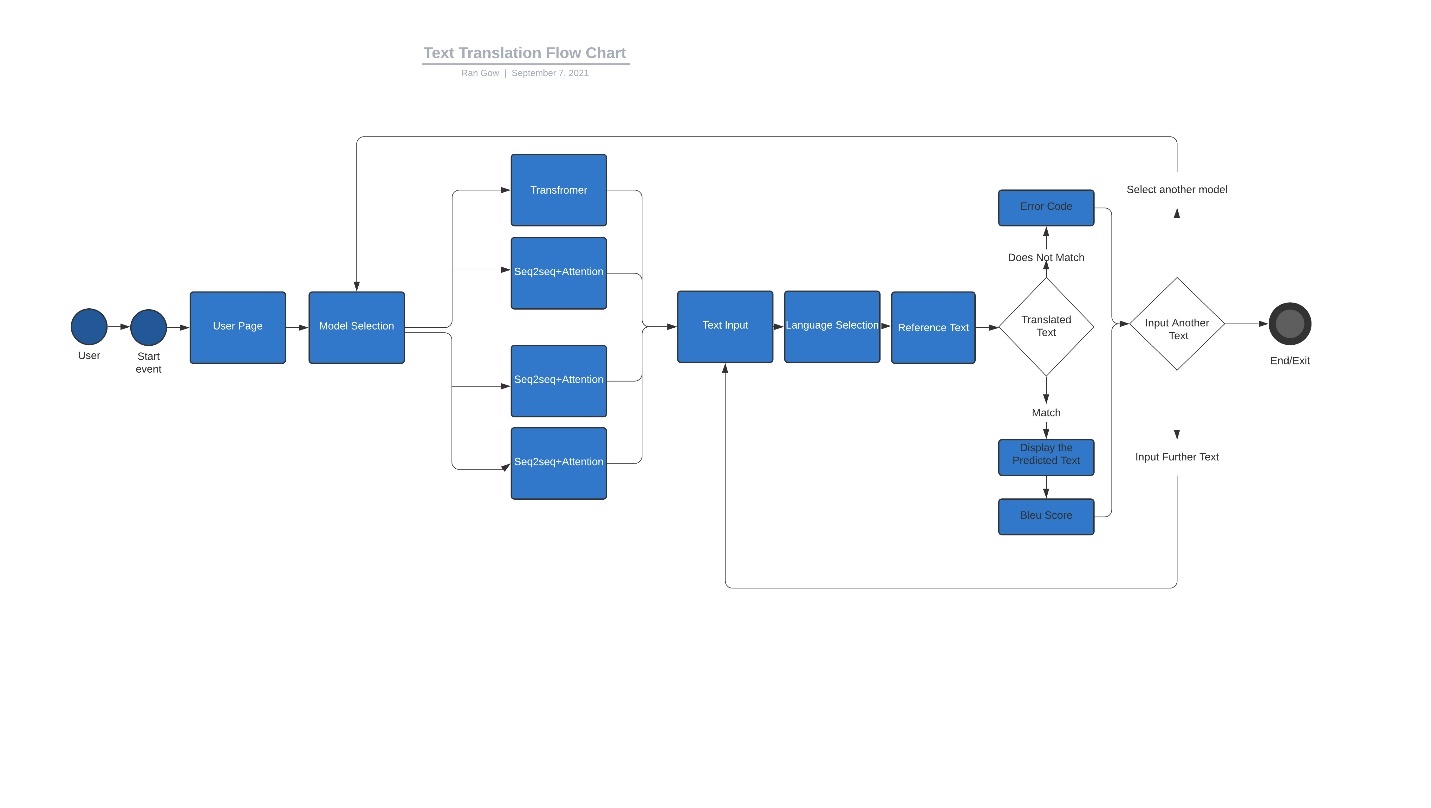
The fourth model performed better than the first three in terms of BLEU. It took only 24 iterations to reach convergence while the second model took 48. However, qualitatively looking at the translations, the first model did a somewhat better job. The average BLEU scores for the three models using six different smoothing methods.

**BERT table of Bleu Score:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models | Source Lang | Target Lang | Dataset | Blue Score |
| SMT(Phrase base) | En (English) | De (German) | WMT16en\_de | 26.47 |
| SMT(Phrase base) | En (English) | Es (Spanish) | WMT13en\_es | 22.12 |
| SMT(Phrase base) | En (English) | Zh (Chinese) | WMT14en\_zh | 19.74 |
| SMT(Phrase base) | En (English) | Hi (Hindi) | WMT16en\_hi | 20.55 |

# Application Flowchart:

The below diagram shows the workflow the Translator application:



Internationalization ensures that a software component may be modified to multiple languages and areas. This involves the use of methods to dynamically change content based on the specified location.

This contains texts, language and text formats, and perhaps pictures. This includes text fragments. This leads to a product generality so many languages, scripts may be handled. As far as translation is concerned, it is crucial not to include text into the code directly but instead to extract it from the enclosing format. The resource files are put to identify the correct material to display.

Starting with model selection, the new page content will include two text boxes, a language selection tab, and a reference text input towards its left, and two output boxes containing translation and blue score output. Once the translation is complete, examine it for translation score and overall similarity. Text quality is tough to assess. Systems that incorporate review capabilities, on the other hand, can enhance translation quality and accelerate localization in a timely way.

It is critical to emphasize the necessity of testing and quality assurance.

Is the resource text that was imported, correct?

Is the translation revealing typographic or layout issues in the user interface?

Are the texts grammatically, correct?

Is the localized outcome appropriate for the targeted culture in terms of color, design, and so on?

I released the localized programme once the translations were incorporated to the package and tested.

# References:

* This blog shows, how to train a translation model from scratch using Transformer. We will be using [Multi30k](http://www.statmt.org/wmt16/multimodal-task.html#task1) dataset to train a German to English translation model.(learn different technique of Transformer)

Link: <https://pytorch.org/tutorials/beginner/translation_transformer.html>

* In this project we will be learning a neural network to translate from French to English. **Author*:***[**Sean Robertson**](https://github.com/spro/practical-pytorch)

Link:<https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html>

* Specifically, we touch on transformers, a common NLP model for sequential data, and how they rely on a specific type of attention called self-attention. We load up our pretrained model, transformer, using the Hugging Face transformer library and model repository. **Author*:*Edan Meyer**

Link: <https://github.com/ejmejm/multilingual-nmt-mt5>

* A simple SMT system for English to Hindi trained using parallel sentences in the train-set, further fine-tuned using the development set, evaluated on a portion of the available test set . **Author*:*Sayer Ghosh Roy**

Link: <https://github.com/sayarghoshroy/Statistical-Machine-Translation>

* Attention is all you need paper.

Link:<https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>

* The annotated transformer.

Link:<https://nlp.seas.harvard.edu/2018/04/03/attention.html#positional-encoding>

# Sequence to Sequence Learning with Neural Network.

Link: <https://arxiv.org/abs/1409.3215> , **Author:** [**Ilya Sutskever**](https://arxiv.org/search/cs?searchtype=author&query=Sutskever%2C+I)**,** 