UNIVERSITY OF TARTU
Institute of Technology
Robotics and Computer Engineering

Md Rezwanul Islam

English-Estonian Machine Translation: Evaluation Across Different Models and Architectures.

Master's Thesis (30 EAP)

Supervisor(s):
Assoc. Prof. Gholamreza Anbarjafari (Shahab)
Hasan Sait Arslan

Tartu 2020

English-Estonian Machine Translation: Evaluation Across Different Models and Architectures.

Abstract:

This thesis is based on three main objectives: at first, the implementation of RNMT+ architecture with Relational-RNN model. This is an interaction between this architecture and the RNN model. Secondly, train three different translation models based on RNMT+, Transformer, and sequence to sequence architectures. Previously, we have witnessed the performance comparison among RNMT+ with LSTM, Transformer, seq2seq, etc. Finally, evaluate the translation model based on training data. When implementing RNMT+, the core idea was to use a newer type of Recurrent Neural Network (RNN) instead of a widely used LSTM or GRU. Besides this, we evaluate the RNMT+ model with other models based on state-of-the-art Transformer and Sequence to Sequence with attention architectures. This evaluation (BLEU) shows that neural machine translation is domain-dependent, and translation based on the Transformer model performs better than the other two in OpenSubtitle v2018 domain while RNMT+ model performs better compared to other two in a cross-domain evaluation. Additionally, we compare all the above-mentioned architectures based on their corresponding encoder-decoder layers, attention mechanism and other available neural machine translation and statistical machine translation architectures.

Keywords:

Neural Machine Translation, Natural Language Processing, LSTM, Relational-RNN, RNMT+, Transformer, Sequence-to-Sequence, Py-Torch, Encoder-Decoder, Attention, Evaluation.

CERCS: P176 Artificial Intelligence

Inglise-eesti masintõlge: hindamine erinevate mudelite ja arhitektuuri vahel.

Lühikokkuvõte:

See lõputöö põhineb kolmel põhieesmärgil: alguses RNMT + arhitektuuri rakendamine Relatsioon-RNN-mudeli abil. See on interaktsioon selle arhitektuuri ja RNN-mudeli vahel. Teiseks, koolitage kolme erinevat tõlkemudelit, mis põhinevad RNMT +, Trafo ja järjestusearhitektuuridel. Varem oleme olnud tunnistajaks RNMT + jõudluse võrdlusele LSTM, Transformeri, seq2seq jne abil. Lõpuks hinnake tõlkemudelit koolitusandmete põhjal. RNMT + rakendamisel oli peamine idee kasutada laialdaselt kasutatava LSTM või GRU asemel uuemat tüüpi korduvat närvivõrku (RNN). Lisaks hindame RNMT + mudelit koos teiste mudelitega, mis põhinevad tipptehnoloogial Transformer ja Sequence to Sequence koos tähelepanu arhitektuuridega. See hinnang (BLEU) näitab, et neuraalne masintõlge on domeenist sõltuv ja muunduril Transformer põhinev tõlge toimib paremini kui ülejäänud kaks OpenSubtitle v2018 domeenis, samal ajal kui RNMT + mudel toimib paremini kui ülejäänud kaks domeenidevahelist hindamist. Lisaks võrdleme kõiki ülalnimetatud arhitektuure nende vastavate kodeerija-dekoodri kihtide, tähelepanu mehhanismi ja muude saadaolevate närvi masintõlke ning statistiliste masintõlke arhitektuuride põhjal.

Võtmesõnad:

Neuraalne masintõlge, loomuliku keele töötlemine, LSTM, relatsiooniline-RNN, RNMT +, trafo, jada-järjestus, Py-taskulamp, kooder-dekooder, tähelepanu, hindamine.

CERCS: P176 Tehisintellekt

Table of Contents

Table of Figures

1 Introduction

Machine translation is a type of automated software that translates a source language into a target language. Machine translation has evolved with time from rule-based systems to statistical machine translation and then neural machine translation, which has further been advanced into better systems that combine features from different models and building new system architecture.

Based on the method of evaluation, machine translation can be classified into ruled based machine translation systems, hybrid machine translation systems, statistical machine translation systems, example-based machine translation, and neural machine translation systems.

Rule-based machine translation being the first type of translation system invented, which was based on grammar rules, a bilingual or multilingual lexicon to process the rules. This type of system is highly dependent on human effort to code all of the linguistic resources.

In this thesis, we evaluate neural machine translation systems and statistical machine translation systems.

Neural machine translation is a translation model that translates one language to another using artificial neural networks. The artificial neural network helps the system to predict the probability of a sequence of words. With Google Neural Machine Translation (GNMT) being one of the best performing neural machine translation advancement followed by Microsoft Neural machine translation.

Other advancements in neural machine translation include the Seq2seq translation model, Transformer model, and RNMT+ which is one of the most recent advancements in the neural machine translation system. This system was introduced by Google Ai in 2018.

In this thesis, we evaluate three neural machine translation systems and one statistical machine translation system and evaluate the results of the system based on the BLEU score.

2 Background

Natural language translation is one of the challenging processes in the field of artificial intelligence. Every year researchers apply their knowledge and experience to make the translation process more robust and performant. When working on machine translation, it's important to know the architecture and model development process. There are many architectures to follow, among those we choose four latest methods, develop translation models, and finally compare the architecture and performance.

The evaluation of MT is vital because it helps us to decide the performance and effectiveness of existing systems. Besides, it helps us develop a more robust system. In this thesis, we evaluate the translation model based on BLEU [1] score. BLEU score evaluation is not perfect still the de facto standard of evaluation. However, in the early stage of MT evaluation, it was human rating based on fluency, informativeness, and intelligibility [2].

To develop a translation model, at first, we need a corpus. There are two types of corpus, parallel and monolingual. A parallel corpus contains two monolingual corpora. To start, we define a corpus, which is a large set of text, mainly billions of words which are generated by real users of a given language and which are used to analyze how words, phrases, and language, in general, are used. The main users of the corpus include linguists and experts in natural language processing. Corpus can be classified into a monolingual corpus which contains text only in one language, a parallel corpus which is made up of two monolingual corpora where one corpus is the translation of the other corpora, multilingual corpora is more like parallel corpus, however, multilingual corpora contain texts in several languages which are all translations of the same text and are aligned in the same way as parallel corpora.

Related to parallel corpora are comparable corpora. This type of corpora is made up of texts which are similar in content, however, the content is not parallel. In my thesis, I used a parallel corpus collected from a different movie and TV subtitles, which is collected in

OpenSubtitles v2018[1]. For the untokenized raw English-Estonian corpus, there were 446612 documents and 3.2G tokens for English. On the other hand, 28837 documents and 168.2M token for Estonian. This collection is efficient over general communication since a linguistic perspective, subtitles cover a wide and interesting breadth of genres, from colloquial language or slang to narrative and expository discourse (as in e.g. documentaries) [3].

The encoder is a recurrent neural network model that takes a machine translation input sequence as input and encodes it into a fixed size that is referred to as a context vector. A context vector is a fixed-length vector representation. The decoder takes a context vector as input in a machine translation system and generates an output sequence.

Bilingual Evaluation Understudy (BLEU) this is a machine translation evaluation method that evaluates the precision score of a candidate machine translation against a reference human translation. The source codes[2] of this project are publicly available on GitHub.

[1] http://opus.nlpl.eu/OpenSubtitles-v2018.php

[2] https://github.com/rezwanshubh/machine-translation.git

3 Machine Translation

Machine translation includes all the process which translate a text or speech from one language to another. Usually, translation is done in sentence level. This task is managed by source sentences (eg: English) and target sentences (eg: Estonian). Machine translation assumes a word segmentation from its input. It is achieved by a process called ‘tokenization’. All the punctuation symbols, words, or numbers are considered separate tokens. In a few cases, an aggressive tokenization technique can separate compound words for the tokenization in sub-word and characters.

3.1 Neural and Statistical Machine Translation

Unlike the traditional phrase-based translation [4] which consists of many small sub-components that are tuned separately, neural machine translation models are a system that attempts to build and train a single, large neural network that read a sentence and output a correct translation, with most of the proposed neural machine translation models belonging to a family of encoder-decoder [5].

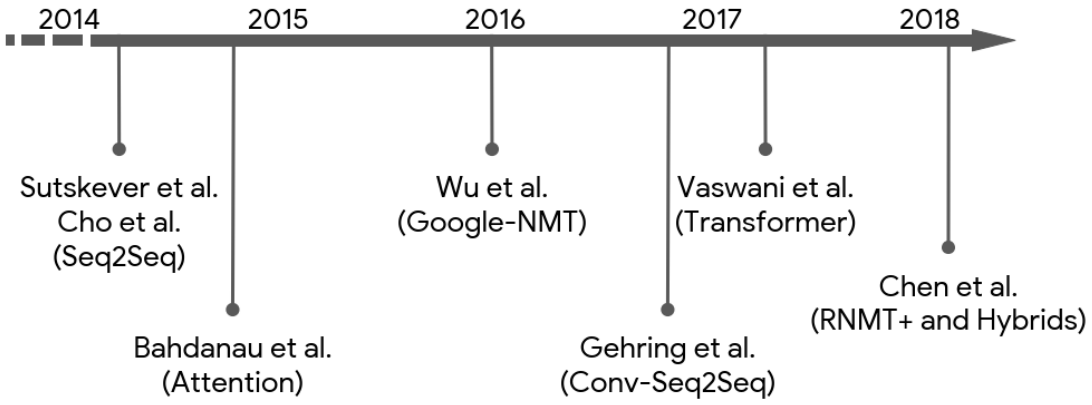

Figure 3.1: History of NMT [6]

Figure 3.1 shows the various advancement in Neural machine translation. With RNMT+ architecture being the most recent advancement in Neural machine translation, which was first introduced by Google AI in 2018.

In recent years neural machine translation has outperformed statistical machine translation (SMT). Previously, the field of machine translation was dominated by SMT models. An

SMT model is driven by analyzing human translation which is called bilingual text corpora. In general, an SMT system is not developed for a specific pair of languages. It just structured like a rule-based or example-based translation system. Unlike NMT, an SMT can be developed on both parallel and monolingual data.

On the other hand, in a neural machine translation system is based on an artificial neural network. Within a single integrated model, a sequence of words is predicted. This model is developed based on parallel corpora of source and target languages. There are a few advantages to develop an NMT system over SMT [7].

- An NMT system considers word similarities during training.
- NMT can handle word order, morphology, and agreement better.
- During training, NMT considers the whole sentence. Which ensures better results.
- It can learn and develop the model based on the complexity of sentences.

In NMT, the encoder-decoder layer and attention mechanism are important concepts. In this work, we evaluated three NMT models, among those only the Transformar model is not based on any encoder-decoder model, rather it fully depends on the attention mechanism. Moreover, an encoder-decoder model with an attention mechanism is the powerhouse of GNMT [8].

3.2 Encoder-Decoder

In a machine translation system, an encoder-decoder is text generation units. The encoder encodes the text sequences into a context vector. This context vector is used by the decoder to generate the output sequence.

Many translation architectures are solely based upon an RNN encoder-decoder model. Though attention mechanism along with the encoder-decoder layer improves the performance, it's also efficient without attention. For example, the sequence to sequence model first been implemented only RNN based encoder-decoder model. The encoder is responsible to encode the input sequence while the decoder decodes the output from the encoder.

The ability to train a single end-to-end model based on source and the target language is the basic advantage of using an encoder-decoder model in machine translation.

3.3 Attention

An attention mechanism refers to a type of translation model where the quality of translation is improved by focusing on sub-parts of the sentences. It produces an output sequence based on an input sequence. The attention mechanism is useful for not only language processing but also for image processing. There are different types of attention mechanisms are used to develop NMT architectures used in this work, here are few [9],

- **Local Attention:** Focuses on the subset of the source words. It's efficient when considering long sentences.
- **Global Attention:** Focuses on all sources words. It considers all encoder hidden states to find the context vector.
- **Self Attention:** Self-attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.
- **Multi-Head Attention:** Multi-Head attention consists of several attention layers in parallel. Multi-Head attention joint information from different positions [10].

All those attention mechanisms are explained in the respective architecture narration.

4 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step [11]. In traditional neural networks, all the inputs and outputs are independent of each other, but in a case like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN comes into existence, which solves this issue with the help of a hidden layer. The hidden state is the most important feature of RNN.

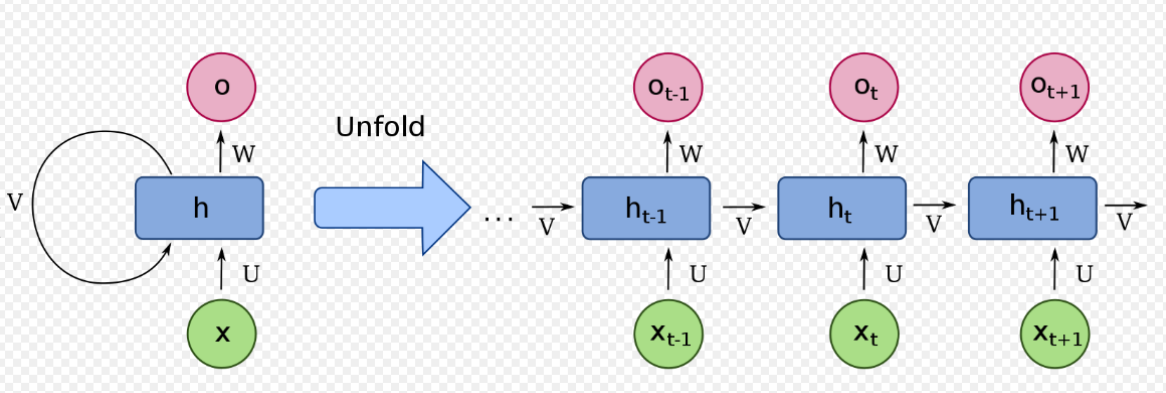

Figure 4.1: Recurrent Neural Network (RNN) [11].

Recurrent neural networks have a memory that remembers all the information about what has been calculated. This model uses the same parameters for each input as it performs the same task on all the input or hidden layers to produce the output. The use of a hidden state helps to reduce the complexity of parameters, unlike other neural networks.

The formula for calculating the current state of RNN

$$h_t = f(h_{t-1}, x_t)$$

Where:

$$h_t = \ Current\ State$$

$$h_{t-1} = Previous\ State$$

$$x_t = Input\ State$$

Tanh Activation

This is used to regulate the values flowing through the network. Tanh activation does this by squishing values to always be between $-1\ and\ 1$.

When vectors flow through neural networks, they undergo many transformations due to various mathematical operations. This leads to some values being extremely large, causing other values to seem insignificant. Therefore, the tanh function ensures that the values stay between $-1\ and\ 1$, thus regulating the output of the neural network.

The formula for applying activation function (tanh) of RNN

$$h_t = tanh(W_{hh}h_{t-1}\,,\ \ W_{xh}x_t)$$

Where:

$$W_{hh} = Weight\ at\ recurrent\ neuron$$

$$W_{xh} = Weight\ at\ input\ neuron$$

The formula for calculating output of RNN

$$Y_t = W_{hy}(h_t)$$

Where:

$$Y_t = Output$$

$$W_{hy} = Weight\ at\ Output\ Layer$$

Training Through RNN

- A single time step of the input is provided to the network.
- Then calculate its current state using set of current input and previous states.
- The current h_t becomes h_{t-1} for the next time step.
- One can go as many times steps according to the problem and join the information from all the previous states.
- Once all the time steps are completed the final current state is used to calculate the output.
- The output is then compared to the actual output i.e. the target output and the error is generated.

- The error is then backpropagated to the network to update the weights and hence the network (RNN) is trained.

Advantages of Recurrent Neural Networks

- An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short-term Memory.
- The recurrent neural network, are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of RNN

- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or rely upon as an activation function.

To solve the limitations of Recurrent Neural Networks more advanced RNN have been developed, this includes; LSTM, GRU, and Relational-RNN.

There are different types of RNN but here among those LSTM and GRU are very popular. However, here we would also explain another type which is Relational-RNN.

4.1 Long Short-Term Memory (LSTM)

LSTM is a special type of RNN, capable of learning long-term dependencies. Introduced by Hoch Reiter & Schmid Huber (1997) [12]. Explicitly designed to avoid long-term dependency. LSTM has similar control flow as a recurrent neural network with the main difference being its operations within the LSTM's cells. It processes data passing on information as it propagates forward.

The building blocks of LSTM's are the cell state and its various gates. The cell state act as a transport highway that transfers relative information down the sequence chain while the gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

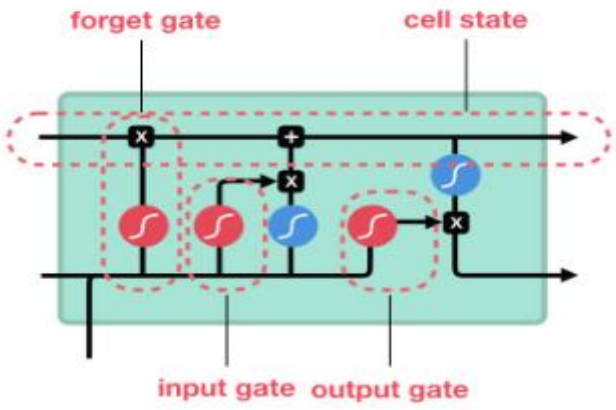

Figure 4.2: LSTM Gates & Cell State [13].

Sigmoid

Each gate of LSTM contains sigmoid activations, which is similar to the tanh activation in recurrent neural networks. Sigmoid activation squishes values between $0\ and\ 1$ instead of $-1\ and\ 1$ as in tanh activation in RNNs. This is helpful in the sense that any number getting multiplied by a 0 is 0, causing the value to disappear or be "forgotten" while any number multiplied by 1 is the same value, therefore, that value stays the same or is "kept." This enables the LSTM network to be able to learn which data is not important therefore, it can be forgotten or which data is important to keep.

LSTM Gates

LSTM's have three types of gates that regulate information flow in an LSTM cell. Figure 3.5 shows the various gates which include;

- A forget gate
- Input gate
- Output gate

Forget Gate

This gate decides which information should be kept per not kept. Information from previous hidden states and information from the current input is passed through the sigmoid function.

The output of sigmoid is valued between $0\ and\ 1$ with the closer to 1 meaning to keep and those close to 0 meaning to forget.

$$f_t = \sigma(W_f\ .[\ h_{t-1}, x_t]\ +\ b_f)$$

Input Gate

This gate is used to control any new information that is added to a cell state from the current input. To update the cell state, we first pass the previous hidden state and current input into a sigmoid function. This decides which value will be updated by transforming the values between $0\ and\ 1$. With 1 meaning important and 0 meaning not important. The hidden state and current input are then passed through tanh function to squish values between $-1\ and\ 1$, which helps to regulate the network. Finally, the sigmoid output is multiplied with tanh output with the sigmoid output deciding which information is important to keep from the tanh output.

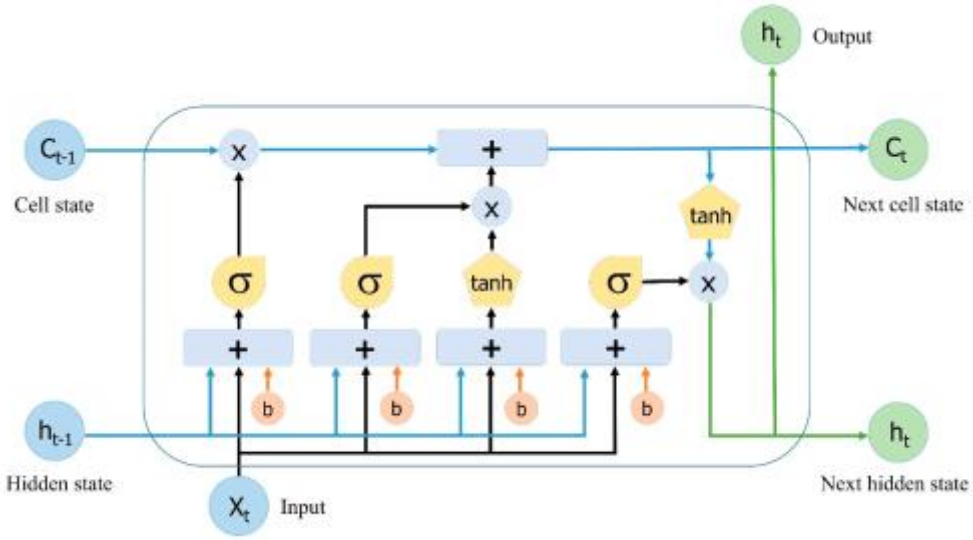

Figure 4.3: Structure of LSTM Neural Network [14].

$$i_t = \sigma(W_i\ .[\ h_{t-1}, x_t]\ +\ b_i)$$

$$\tilde{C}_t = tanh(W_c\ .[\ h_{t-1}, x_t]\ +\ b_c)$$

Memory Update or Cell State

The cell state aggregates the two components i.e. the old memory through forget gate and new memory through the input gate. This gives enough information to be able to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector, with a possibility of dropping values in the cell state if it gets multiplied by a value near 0. Then a pointwise addition of the output of the input gate is performed, which updates the cell state to a new, values that the neural network finds relevant. That gives us a new cell state.

$$C_t = f_t \times c_{t-1} + i_t \times \widetilde{c_t}$$

Output Gate

The output gate decides what to output from the memory. i.e. what the next hidden state should be. This is carried out by first passing the previous hidden state and the current input into a sigmoid function. Then the newly modified cell state is passed into the tanh function. The tanh output is then multiplied with the sigmoid output to decide what information the hidden state should carry. The resulting output is the hidden state. The new cell state and the new hidden state is then carried over to the next time step.

$$o_t = \sigma(W_o \,.[\, h_{t-1}, x_t] \,+\, b_o)$$

$$h_t = o_t \times tanh(C_t$$

4.2 Gated Recurrent Unit (GRU)

GRU is a newer generation of RNN and is pretty similar to an LSTM. GRU's were first introduced by Cho, et al [15]. in 2014. However, GRU's don't use cell states for transfer of information like LSTM, instead of their use of the hidden state to transfer information. GRU's have two gates a reset gate and an update gate.

GRU structure allows it to adaptively capture dependencies from large sequences of data without discarding information from earlier parts of the sequence. It achieves this through the gating units which are similar to the ones in LSTMs, which solve the vanishing/exploding gradient problem of traditional RNNs. The gates are responsible for regulating the information to be kept or discarded at each time step.

Reset Gate of GNU

This gate is derived and calculated using both the hidden state from the previous time step and the input data at the current time step.

This is done by multiplying the previous hidden state and current input with their respective weights and summing them before passing the sum through a sigmoid function [16]. The sigmoid function is used to transform the values to fall between $0\ and\ 1$, allowing the gates to filter between the most important and less important information in the subsequent steps. $-1\ and\ 1$ [17]

$$gate_{reset} = \sigma\left(W_{input_{reset}} x_t + W_{hidden_{reset}} h_{t-1}\right)$$

When backpropagation is used in training the entire network the weights in the equation are updated such that the vector learns to retain only the useful features. The previous hidden state first been multiplied by a trainable weight and then undergoes an element-wise multiplication (Hadamard Product) with the reset vector. This operation decides which information is to be kept from the previous time steps together with the inputs. At the same time, the current input will also be multiplied by a trainable weight before being summed with the product of the rest vector and previous hidden state. Lastly, a non-linear activation tanh function is applied to the final result to obtain r in the equation below [16].

$$r = tanh(gate_{reset} \odot (W_{h1}.h_{t-1}) + Wx_1.x_t)$$

Update Gate of RNN

This gate is computed using the previous hidden state and current input data. This gate determines how much of the previous hidden state is to be retained and what portion of the new proposed hidden state (derived from the reset gate) is to be added to the final hidden state.

Both the reset and update gate vectors are created using the same formula, but the weights multiplied with the input and hidden state are unique to each gate, which means that the final vectors for each gate are different.

$$gate_{update} = \sigma\left(W_{input_{update}} x_t + W_{hidden_{update}} h_{t-1}\right)$$

The updated vector then undergoes element-wise multiplication with the previous hidden state to obtain u which is used to compute the final output.

$$u = gate_{reset} \odot h_{t-1} t$$

Final Output Computations

In this stage, we take the element-wise inverse version of the same updated vector and do an element-wise multiplication with the output of the reset gate r. This ensures the update gate determines which portion of the new information should be stored in the hidden state. Lastly, the above result will be summed with the output of the update gate u, which will then give us our new and updated hidden state.

$$h_t = r \odot (1 - gate_{reset}) + u$$

Comparison Between GRUs and LSTMs

GRUs are faster to train as compared to LSTMs due to the fewer number of weights and parameters to update during training. This is attributed to the fewer number of gates in the GRU cell as compared to the 3 gates in LSTMs.

LSTMs have cell state while GRUs don't have cell state. With LSTMs using the cell state to store its longer-term dependencies in the cell state and short-term memory in the hidden state, the GRUs stores both in a single hidden state.

4.3 Relational-RNN

Relational-RNN combines the advantages of LSTMs in sequence modeling and the power of attention mechanism. It does this by extending the LSTM architecture and introducing interactive memory blocks using Multi-Head Dot Product Attention inside of the LSTM block. Relational-RNNs can simplify memory-based recurrent neural networks that can perform relational reasoning between input entries over time. They are based on iterative information selective storing into blocks and computing interactions between them. With most of the Relational-RNNs block contains several memory slots where the pertinent information is stored. Relational-RNN are characterized by:

- Per block information storing.
- Relational reasoning.
- No distance constrained analysis.
- Different focusing.

Generally, work on the principle of slicing the memory and the inputs into slots while the head, on the other hand, is provoking interactions between them. During this process, each of the memory slots is updated stepwise based on memory-memory attention and memory-input attention.

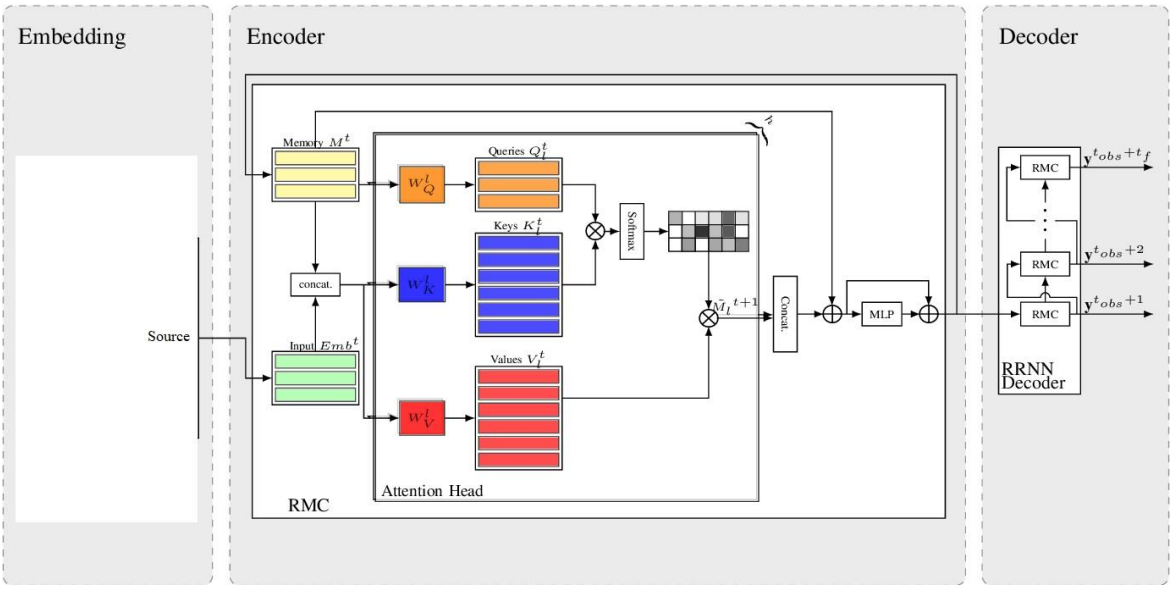

Figure 4.4: Basic of Relational Recurrent Neural Network [18].

Memory-Memory attention

Each memory slot attends over the other memory slots. This captures the interactions and dependencies in the stored information [18].

Memory-input attention

Each memory slot attends over the input embedding slots. Attention enables us to decide which information from the input would be stored in adequate memory slots based on its relation to what is already contained in the memory [18].

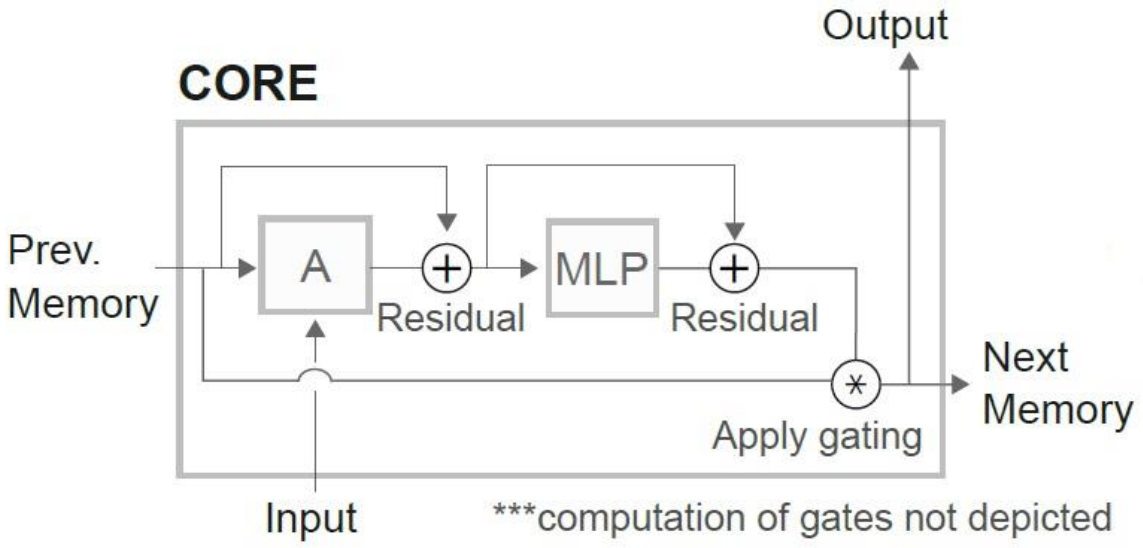

Figure 4.5: Architecture of Relational Memory Core [19].

Multi-Head Dot Product Attention (MHDPA)

Within each Relational-RNN block, linear projections of the previous memory M^t is used and the input embedding Emb^t at each time step t to generate the queries ${Q^t}_l = M^t {W^l}_Q$, keys ${K^t}_l = [\ M^t; Emb^t\ ]\ {W^l}_K$ and values ${V^t}_l = [\ M^t; Emb^t\ ]\ {W^l}_V$. [$M^t; Emb^t$] which are used to denote the row-wise concatenation of M^t and Emb^t. We then use multiple attention heads to enable the memory slots to share different information and represent different interactions. Which generates h sets of queries, keys, and values for $l = 1 \ldots . h$ using different projection matrices.

The memory is updated using multi-head dot product attention over the other memory slots and the current input embedding:

$$\widetilde{M}_l^{t+1} = A(Q_l^t \cdot K_l^t \cdot V_l^t) = softmax\left(\frac{Q_l^t transpose(K_l^t)}{\sqrt{d^k}}\right) V_l^t$$

Where:

$\widetilde{M}_l^{t+1}$= is an update of the memory where each slot is a weighted sum of the projections of the previous memory slots and the projections of the current embedding input.

d^k= is a scaling factor that corresponds to the dimensionality of the key vectors.

This attention operation is applied to each head. The resulting memory $\widetilde{M}_l^{t+1}$ is the column-wise concatenation of the memories $\widetilde{M}_l^{t+1}$ for $l = 1 \ldots . h$.

Residue connection is then employed around MHDPA followed by an MLP then a second residue connection. These operations are encapsulated into an LSTM cell. And therefore, the resultant memory block is gated and used as the next memory state M_l^{t+1}

Input Embedding

In Relational-RNN, the model of the source language is first embedded, then the encoder learns the source language and captures the dependencies in the input data using the Relational Memory Core (RMC) block. For each iteration, the RMC is fed with the previous memory matrix M^t and the current scene embedding Emb^t and to provoke the interaction between memory and input slots, the Multi-Head Dot Product Attention (MHDPA) is used. MHDPA operates by projecting each memory and input slot using row-wise shared weights i.e. ${W^l}_Q, {W^l}_K, {W^l}_V$ to generate the queries Q^t, keys K^t and values V^t respectively. The MHDPA module is then followed by a row-wise multilayer perceptron (MLP), then, the resultant memory is gated to form the next memory state and the output vector is then fed to the decoder at t_{obs}. The decoder which is composed of Relational-RNN's outputs the predicted sequence.

As previously mentioned that RNMT+ architecture was first introduced by Google AI in 2018. It proposed a hybrid model combining Sequence to Sequence and RNN architecture. RNMT+ architecture has 6 bidirectional encoder layers and 8 unidirectional decoder layers. With most RNMT+ implemented using LSTM. However, in my thesis, I implemented RNMT+ using Relational-RNN.

5 Sequence to Sequence with Attention and PyTorch

Sequence to Sequence is a model that was first introduced by Google in 2014. This model tries to map input text with fixed length to output text fixed-length where the length of input and output to the model may differ. It does so by using more advanced versions of recurrent neural networks (RNN) mainly LSTM or GRU. This is because Recurrent neural networks (RNN) suffer from the problem of vanishing gradient. LSTM is used in the version proposed by Google.

The approach involves two recurrent neural networks, one to encode the input sequence, called the encoder, and a second to decode the encoded input sequence into the target sequence called the decoder [20]. The system also incorporates the attention mechanism which helps to solve the translation of long sequences. The issue with long sequence translation is due to the challenge for the encoder to memorize the entire sequence into a fixed-size vector and to compress all the contextual information from the sequence.

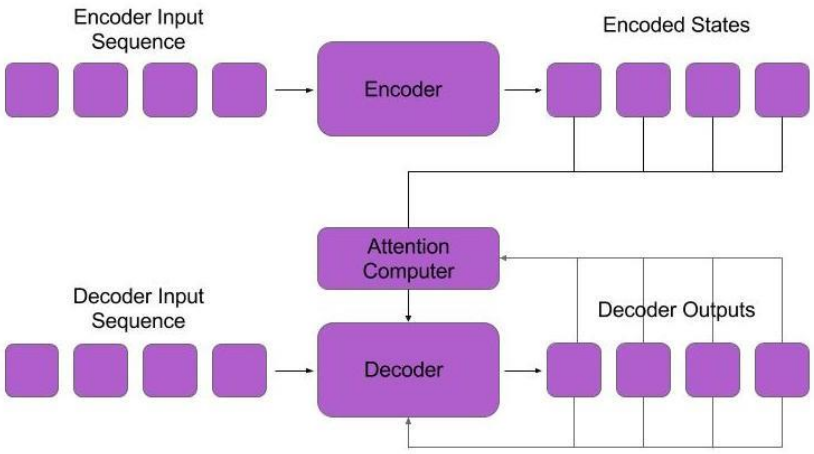

Figure 5.1: A sequence to sequence architecture with attention [21].

General Working of Seq2Seq Architecture

The encoder reads the input sequence and summarizes the information into internal state vectors. If the LSTM is used in the architecture, hidden state and cell state vectors will be generated. The output of the encoder is then discarded and only the internal states are preserved.x`

The LSTM decoder, initial states are then initialized to the final states of the LSTM encoder. With the available initial states, the decoder starts generating the output sequence. To overcome the problem of long sequence memorization of the encoder, the concept of attention mechanism is incorporated into the architecture. Attention mechanism a specific part of a sequence is given more importance instead of the entire sequence to predict that word. In this mode, the intermediate from the encoder state is utilized to generate context vector from all states so that the decoder gives output results. Instead of going through the entire sequence the attention mechanism only pays attention to specific words from the sequence and gives out the result based on that.

The attention layers are made up of;

1. Alignment layer
2. Attention weights
3. Context vector

Alignment Layer

This is made up of an alignment score that maps how well the inputs around position 'j' and the output at position 'I' match. With the score based on the previous decoder's hidden states, just before predicting the target word and the hidden state of the input sentence.

On the other hand, the decoder decides which part of the source sentence to pay attention to, instead of having the encoder encode all the information of the source sentence into a fixed-length vector. The alignment vector and source sequence have the same length and are computed at every time step of the decoder.

Attention Weights

To obtain the attention weights, a SoftMax activation function is performed on the alignment scores. The SoftMax function gives a probability whose sum will always be equal to 1. This helps to represent the weight of the influence for each of the input sequences. With the higher the attention weights of the input sequence, the higher will be its influence on predicting the target word.

Context Vector

This is used to compute the final output of the decoder. Where the context vector is the weighted sum of the attention weights and the encoder hidden states, which maps to the input sentence.

To predict the next target word, the decoder uses; the context vector, decoder's output from the previous time step, and the previous decoder's hidden state.

Attention-Based Models

Two types of attention mechanisms can be used in the sequence to sequence model. The attention mechanism used will depend on how context vector information you need to compress from the input sequence. Common to these two attention mechanisms is the fact that at each time step t in the decoding phase, both approaches first take hidden state h_t as input at the top layer of a stacking LSTM with the goal generating a context vector C_t that captures relevant source-side information to help predict the current target word y_t. The two mechanisms differ in the way they generate the context vector C_t however, they share the same subsequent steps [22].

Specifically, given the target hidden state h_t and the source-side context vector C_t, a simple concatenation layer is employed that combines the information from both vectors to produce an attentional hidden state [22].

$$\tilde{h}_t = tanh(W_c[C_t; h_t])$$

The attention vector $\tilde{h}_t$ which is then fed via the SoftMax layer to generate the predictive distribution formulated as;

$$p(y_t|y < t, x) = softmax(W_s\tilde{h}_t)$$

Global Attention

This attention mechanism model considers all the hidden states of the encoder when generating the context vector C_t.

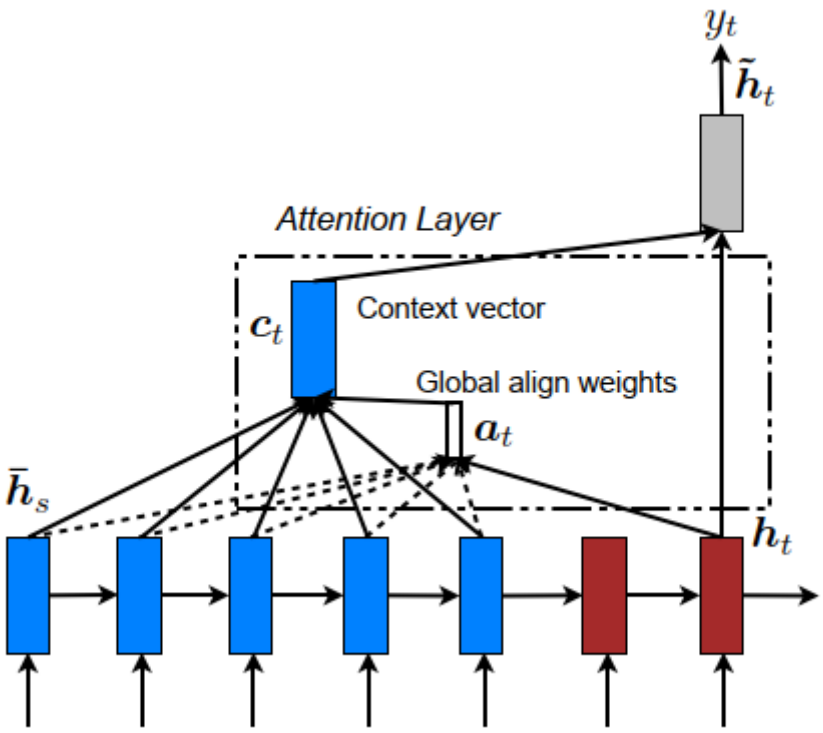

Figure 5.2: Global Attentional Model [22]

This model derives a variable-length alignment vector a_t, whose size equals the number of time steps on the source site. The variable-length alignment vector a_t is derived by comparing the current target hidden state h_t with each source hidden state $\check{h}_s$.

$$a_t(s) = align(h_t, \check{h}_s)$$

$$= \frac{\exp\left(score(h_t, \check{h}_s)\right)}{\sum_{s'} \exp\left(score(h_t, \check{h}_s)\right)}$$

In this model score is defined as the content-based function for which we consider three different alternatives;

Local Attention

This attention mechanism tries to solve the drawback of the global attention mechanism (i.e. global attention mechanism has to attend to all words on the source side for each target word, which is expensive and can potentially render it impractical to translate longer

sequences). To address this deficiency, a local attention mechanism is used that chooses to focus only on a small subset of the source positions per target word [22].

This approach selectively focuses on a small window of context and is differentiable. The local attentional mechanism comes to switch the advantage of avoiding the expensive computation incurred in soft attention and at the same time, is easier to train than the hard attention approach.

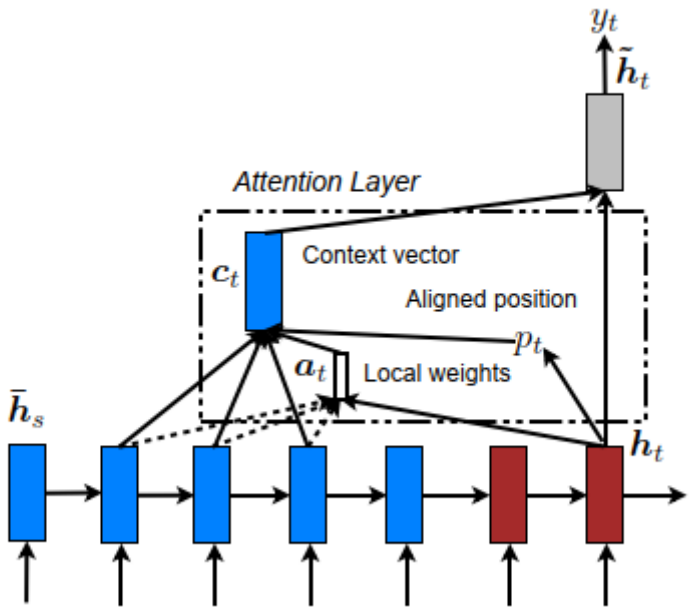

Figure 5.3: Local Attentional Model [22].

In concrete details, the local attentional mechanism first generates an alignment position p_t for each target word at time t. The context vector C_t is then derived as a weighted average over the set of source hidden states within the window $[p_t - D, p_t + D]$; with D empirically selected unlike the global approach, the local alignment vector a_t is now fixed-dimensional [23]. The same alignment equation is used in both mechanisms.

$$a_t(s) = align(h_t, \check{h}_s) = \frac{\exp\left(score(h_t, \check{h}_s)\right)}{\sum_{s'} \exp\left(score(h_t, \check{h}_s)\right)}$$

Prepare Train, Dev and Test Dataset

A machine learning algorithm learns from data. In a supervised translation system, data is the base of all resources which sometimes referred to as labels. After training a model we expect to translate any unknown sentences. In this case, we split the OpenSubtitle v2018 into 3 parts. The first one is for the training dataset, contains 29,194,732 sentence pairs. It's used to train the model. The second one is for the validation or development dataset. It contains 1,500 sentence pairs. This part provides an unbiased evaluation of the model fit on the training dataset while tuning model hyperparameters [24]. The final part is for model testing. It consists of 3,000 sentence pairs. We used another pair of a dataset for testing on the EU-law domain which consists of 1,314 sentence pairs. Test data is important for an unbiased evaluation of the model.

Train Model Using GPUs

A graphics processing unit (GPU) is a specialized electronic circuit that performs rapid mathematical calculations, primarily to render images. A GPU is more powerful than a CPU because of its high bandwidth, easily programmable register, and the ability of hiding latency in thread parallelism.

In this thesis, all three NMT architectures and their implementations are GPU enabled. Which means we can use GPU extensively. For all pre-processing and training, I used almost the same configuration. We can use GPU by slurm commands. 'sbatch' submits a batch script to Slurm. The batch script is given to sbatch through a file name on the command line, or maybe sbatch can read in a script from standard input. The batch script may contain options preceded with "#SBATCH" before any executable commands in the script. sbatch will stop processing further #SBATCH directives once the first non-comment non-whitespace line has been reached in the script[3].

[3] https://slurm.schedmd.com/sbatch.html

```
#!/bin/bash
#
#SBATCH --job-name=train_model
#SBATCH --partition=gpu
#SBATCH --gres=gpu:tesla:1
#SBATCH --time=7-
#SBATCH --cpus=1
#SBATCH --mem-per-cpu=120000
```

- SBATCH: submit a batch script to slurm.
- --job-name: the specified name will appear along with the job id number when querying running jobs on the system.
- --partition: request a specific partition for the resource allocation.
- --gres: Specifies a comma-delimited list of generic consumable resources. The format of each entry on the list is "name[[:type]:count]".
- --time: set a limit on the total run time of the job allocation.
- --cpus: advise Slurm that ensuing job steps will require ncpus processors per allocated GPU.
- --mem-per-CPU: minimum memory required per allocated CPU.

Data Pre-Processing and Training

Here is a sample script[4] to pre-process and train data. This source code is based on example [25] of PyTorch: Translation with Sequence to Sequence Network and Attention. After running this script, text in Estonian is saved in a file.

```
python ./seq2seq_translate.py
```

In this case, the hidden size is 512. Training run until 350,000 steps.

[4] https://github.com/rezwanshubh/machine-translation/tree/master/seq2seq

6 Transformer Model with OpenNMT-py

The Transformer model consists of 6 identical layers in Encoder and 6 identical layers in Decoder. Each layer of encoder has two sub-layers. The first sublayer being the multi-head self- attention mechanism while the second layer is the position-wise fully connected feed-forward network. There is a residual connection around each of the two sub-layers, which is followed by a normalization layer.

The decoder layer also has two sub-layers and incorporates a third sub-layer which conducts multi-head attention over the output of the encoder stack. Each of decoders sublayer is also surrounded by residual connections followed by normalization. To prevent positions from attending to subsequent positions, the self-attention layer in the decoder stack is modified.

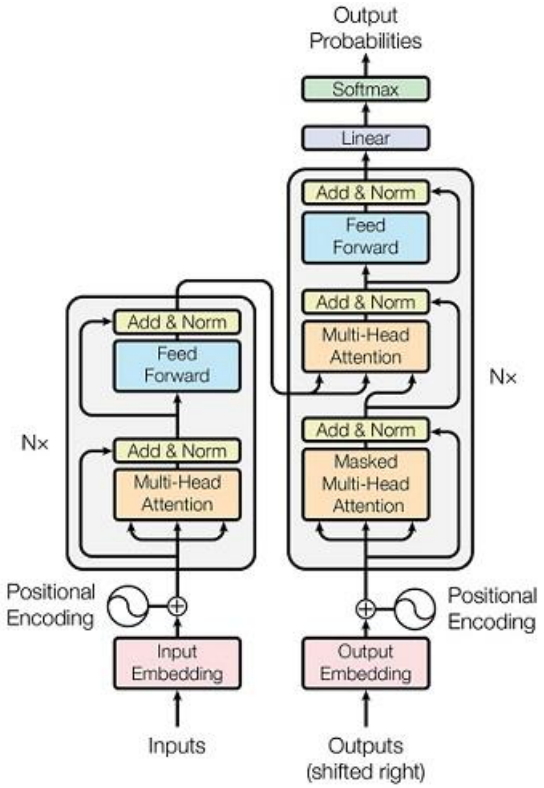

Figure 6.1: Transformer Architecture [10].

The Attention function involves mapping a query and a set of key-value pairs to an output [26]. The query, keys, values, and output are all vectors. The weighted sum of the values forms the output. The weight that's assigned to each value is calculated by a compatibility

function of the query with the corresponding key [26]. The Transformer architecture is shown in figure 6.1 above.

Attention is a mechanism in which the decoder can decide to attend to the hidden states of the input. i.e. the decoder can go back and look at the input. the way the decoder decides on what to look at is like an addressing scheme in any neural machine translation model.

Attention is mainly used to reduce path length, which reduces the number of computation steps thus attention is expected to work better.

Important Features of Transformer

The Transformer model is fully based on the attention mechanism. Like RNMT+ or sequence to sequence, no RNN (eg: LSTM, GRU) is involved with an encoder or decoder layer. It introduces the concept of the multi-head attention mechanism. It describes a new way of positioning words (to remember the order of the words) by using an explicit position encoding that I added to the input integrations. The Transformer solves the difficulty in parallelizing and in learning long-range dependencies within the input and output sequences which are found in RNN and seq2seq models.

The Transformer uses layer normalization and residual connections to facilitate optimization. It achieves impressive results (41.8 BLEU score) when using the WMT 2014 English-French and English Deutsch datasets.

Analysis of Transformer Architecture

In figure 6.1 the left side is the encoder layers and the right side is the decoder layers. The encoder block is made up of one layer of a Multi-Head Attention followed by another layer of Feed Forward Neural Network. The decoder, on the other hand, is made up of an extra Masked Multi-Head Attention. The encoder and decoder blocks are multiple identical encoders and decoders stacked on top of each other [10].

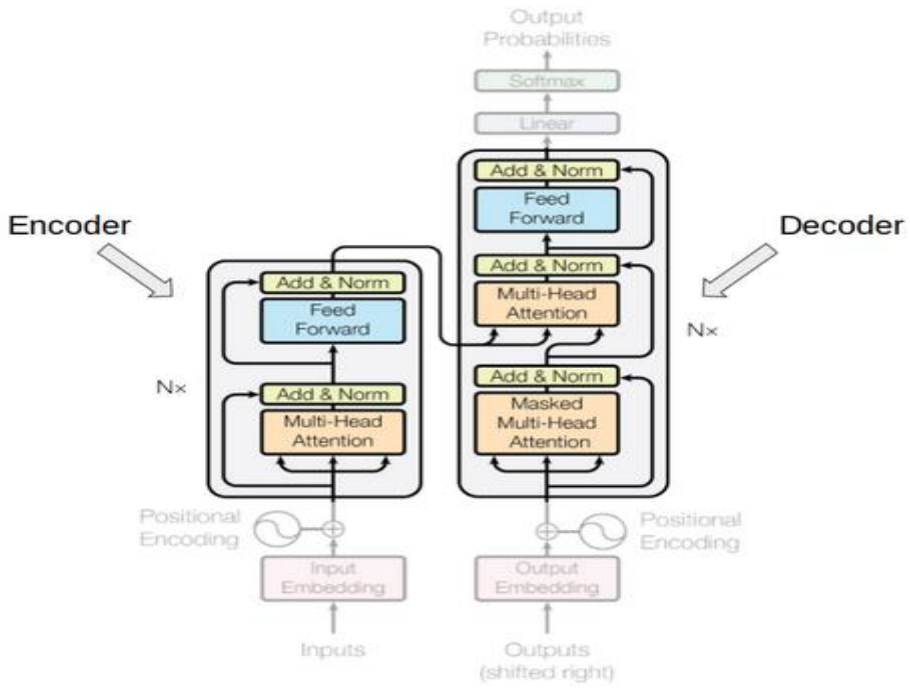

Figure 6.2: Encoder and Decoder of a Transformer model [10].

Working of Encoder-Decoder Stack

The word embeddings of the input sequence are passed to the first encoder, which is then transformed and propagated to the next encoder. The output from the last encoder in the stack is then passed to all the decoders in the decoder stack.

An additional layer called encoder-decoder attention is incorporated in the decoder which helps the decoder to focus on the appropriate parts of the input sequence.

Both the encoder has a self-attention layer. This self-attention layer is an attention mechanism relating to different positions of a single sequence to compute a representation of the sequence.

Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors [10]. The output is always computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding keys [10].

Multi-Head Attention

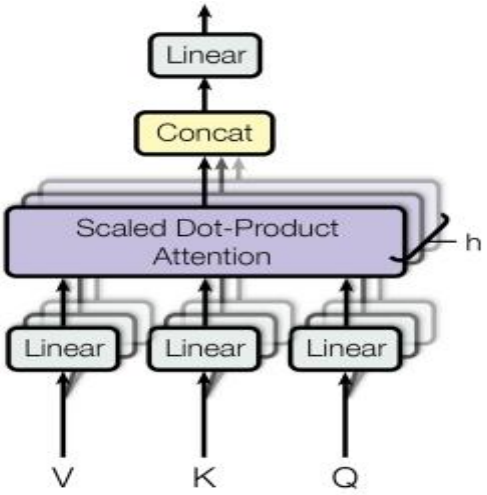

Figure 6.3: Multi-Head Attention [10]

Multi-Head Attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = concat(head_1, \cdots, head_h)W^O$$

$$Where\ head_1 = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend overall positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models [10].

The encoder contains self-attention layers. In a self-attention layer, all of the keys, values , and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder [10].

Similarly, self-attention layers in the decoder allow each position in the decoder to attend to

all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the SoftMax which correspond to illegal connections [10].

Position Encoding

Considering an RNN model, the words are feed sequentially to the model, each token being aware of how it was ordered. However, multi-head attention computes the output of each item in the sequence independently with no notion of word order. It is inefficient to model the sequence information without any special order or position. To account for the order of the words in the input sequence, the Transformer model adds a vector to each input embedding called Positional Encoding. Where positional Encoding from the Transformer model is computed by sine and cosine functions of different frequencies as:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{(10000)^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{(10000)^{\frac{2i}{d_{model}}}}\right)$$

$$where$$

$$i = represents\ the\ vector\ index\ we\ are\ looking\ at$$

$$pos = represents\ the\ position$$

$$d_{model} = represents\ the\ dimension\ of\ the\ input\ embeddings$$

With each dimension of the position encoding forming a sinusoid and which allows the model to be generalized to longer sequence lengths. Where PE_{pos+k} can be determined by a linear function of PE_{pos} with an offset K, which makes the relative position between different embeddings.

Embeddings and SoftMax

To convert the input tokens and output tokens to vectors of dimension d_{model}, learned embedding is used. Moreover, SoftMax function and linear transformation are also used to convert the decoder output to predicted next-token probabilities. In this model, we share the weight matrix between the two embeddings layers and the pre-SoftMax linear transformation. This embeddings layer is finally multiplied with $\sqrt{d_{model}}$

Position-Wise Feed-Forward Network

In addition to the attention sub-layer, each of the layers in the encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a REL activation in between [10].

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2$$

Limitations of the Transformer

- Attention can only deal with fixed-length text strings. The text has to be split into a certain number of segments or chunks before being fed into the system as input.
- This chunking of text causes context fragmentation. For example, if a sentence is split from the middle, then a significant amount of context is lost. In other words, the text is split without respecting the sentence or any other semantic boundary.

Open NMT

It is a community of projects supporting easy adoption neural machine translation [27]. Open MT has currently three main implementations;

- OpenNMT-lua

This is the original project developed in Touch 7. Full-featured, optimized, and stable code ready for quick experiments and production [27].

- OpenNMT-py

This is an OpenNMT-lua clone using PyTorch. Initially created by Adam Leer and the Facebook AI research team as an example, this implementation is easy to extend and particularly suited for research [27].

- OpenNMT-tf

This is an implementation following the style of TensorFlow. This is a newer project focusing on large scale experiments and high, performance model serving using the latest TensorFlow features [27].

OpenNMT-py

OpenNMT-py is a PyTorch port for NMT. A sharing mechanism is implemented for data loading which enables training on extremely large datasets that cannot fit into memory, and for back-propagation, which reduces memory footprints during training. Training a translation model using OpenNMT-py is easy and very efficient.

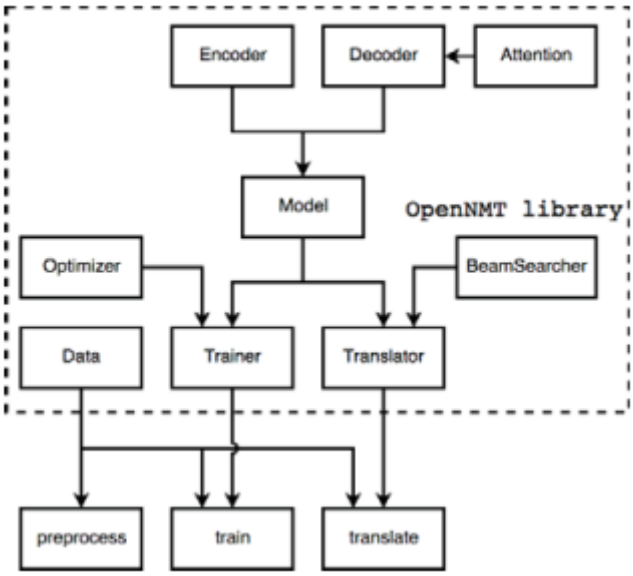

Figure 6.4: Schematic Overview of OpenNMT-py code [27]

Tokenizer and Detokenizer

It is evident that tokenization reduces vocabulary size and rise the number of examples of each word. Tokenization is important to understand the word size within a token. Because it changes the meaning of the sentences. For example, separately the English words 'there' and 'fore' has different meanings than together 'therefore'. That's why correct tokenization improves translation quality. Neural machine translation frequently relies on RNN and it’s

encoder-decoder unit. Mainly the source is projected into an encoded step then translated it using a particular translation model and finally projected into a decoder step [28].

In an encoding system, each word of a sentence is projected linearly to a fixed size vector along with an embedding matrix. After embedding, these words are fed into a bidirectional LSTM network. Thus, a sequence of annotations is developed by merging the hidden states from all the forward and backward layers [28].

SentencePiece

In this thesis, we used SentencePiece[5] as a tokenizer and detokenizer. It's a simple and easy to use unsupervised tokenizer and detokenizer mainly used for the neural network-based translation system.

SentencePiece includes four main components, which are Normalizer, Trainer, Encoder, and Decoder [29]. Normalizer normalizes semantically equivalent Unicode characters into the right forms. Trainer trains the subword segmentation model from the normalized corpus. Encoder internally normalizes the input text and tokenizes it into a subword sequence with the subword model trained by Trainer. The decoder converts the subword sequence into the normalized text [29].

SentencePiece is a language independent preprocessing technique. Any language-dependent pre-processor has several problems like the following one.

- Row text: Hello world.
- Tokenized: [Hello] [world] [.]

But in this case, the sequences are not reversibly convertible. We can see a 'space' does exist between [world] and [.] after tokenization.

On the other hand, SentencePiece implements a decoder system which is equivalent to inverse of the encoder. For example,

[5] https://github.com/google/sentencepiece

Decode(Encode(Normalize(text))) = Normalize(text)

This is lossless tokenization [29]. The main idea is to treat the input text as a sequence of Unicode characters. In SentencePiece, 'white space' is also considered a symbol. For example,

- Raw text: Hello_world.
- Tokenized: [Hello] [_wor] [ld] [.]

Here 'white space' is considered as segmented text that why when detokenized, we can replace '_' with space ' '.

Using SentencePiece gives us a performance gain in NMT over other sentence processors like Subword-NMT or Moses in different language pairs [29]. During all three NMT translation models in this thesis, I used SentencePiece as a standard tokenizer and detokenizer. To install SentencePiece using pip, we can run the following command.

```
pip install sentencepiece
```

To encode and decode the raw text into sentence pieces.

```
#encoder
% spm_encode --model=<model_file> --output_format=piece < input > output

#decoder
% spm_decode --model=<model_file> --input_format=piece < input > output
```

Data pre-processing

Here is a sample script to pre-process data using OpenNMT-py. This following script[6] has been run on the HPC rocket cluster, falcon1 node.

[6] https://github.com/rezwanshubh/machine-translation/tree/master/transformer

```
#!/bin/bash
#
#SBATCH --job-name=train_model
#SBATCH --partition=gpu
#SBATCH --gres=gpu:tesla:1
#SBATCH --time=7-
#SBATCH --cpus=1
#SBATCH --mem-per-cpu=120000

python ./preprocess.py \
-train_src ./en.train.enc \
-train_tgt ./et.train.enc \
-valid_src ./en.dev.enc \
-valid_tgt ./et.dev.enc \
-save_data ./data
```

Training a model

Here is a sample script to train a model using OpenNMT-py following Transformer architecture. This training run until 350,000 steps with hidden size 512 and batch size 4096.

```
#!/bin/bash
#
#SBATCH --job-name=train_model
#SBATCH --partition=gpu
#SBATCH --gres=gpu:tesla:1
#SBATCH --time=7-
#SBATCH --cpus=1
#SBATCH --mem-per-cpu=120000

python ./OpenNMT-py/train.py -data ./data -save_model ./models/ \
-layers 6 -rnn_size 512 -word_vec_size 512 -transformer_ff 2048 \
-heads 8 -encoder_type transformer -decoder_type transformer \
-position_encoding -train_steps 350000 -dropout 0.1 -batch_size 4096 \
-gpu_ranks 0
```

7 RNMT+ with Relational RNN

RNMT+ is a hybrid architecture, developed combining all the successes in seq2seq, convolution, and Transformer architecture [6]. To develop an RNMT+ model researcher proposed two ways. The first one is to identify different modeling and training techniques and apply them to basic RNN architecture. It results in a super performant RNMT+ architecture. The second procedure was combining the properties of all basic seq2seq architectures. A basic RNMT model consists of an RNN encoder unit and an RNN decoder unit coupled with an attention network. Source sentences directed into a set of vector in encoder then these encoded sequences are set through the attention mechanism of decoder.

Usually, an LSTM or GRU network is used for RNMT+ architecture. However, we use the Relational-RNN network to implement this model. In this proposed version of RNMT+, we use 6 bidirectional Relational-RNN layers instead of 6 bidirectional LSTM layers in the encoder. On the other hand, in the decoder layer, we used 8 unidirectional Relational-RNN layers instead of 8 unidirectional LSTM layers.

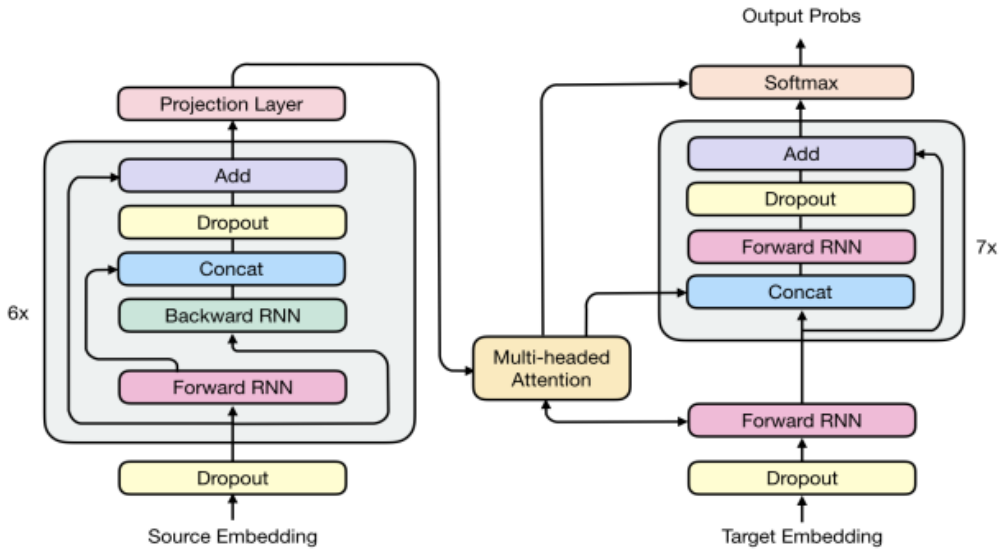

Figure 7.1: Model Architecture of RNMT+ [6]

We trained our models on En-Et, consists of 29,194,732 sentence pairs based on OpenSubtitles v2018. Each word is tokenized with SentencePiece [29]. We used a shared vocabulary of size 16k sub-words.

RNMT+ architecture is shown in Figure 5.1. Google Translator is based on GNMT [8]. There are similarities and dissimilarities between GNMT and RNMT+. In GNMT, there is 1 bidirectional LSTM layer followed by 7 unidirectional LSTM layers for the encoder. For the decoder, there are 8 unidirectional LSTM which is similar to the RNMT+. Single-head attention is used for GNMT while multi-head additive attention is used for RNMT+. Similar to GNMT, RNMT+ feeds the attention context to all decoder layers and then feed to the softmax. It's essential for the quality of the models and also for the stabilities. The following regularization technique has been applied in RNMT+. These methodologies and processes would be the same for both Relational-RNN and LSTM.

- **Dropout:** Dropout is used to prevent neural networks from overfitting. Thus randomly fitted neurons are ignored during training. Dropout is applied on both embedding layers and each Relational-RNN layer's output before it is inserted into the next layer. Dropout is the most successful technique to regularize the neural network. Efficient uses of dropout can improve performance in machine translation, speech recognition, language modeling, and image caption generation [30].
- **Level smoothing:** Smoothing is a way to smooth out data for presentation or make a forecast. Level smoothing has a positive impact on both RNMT+ and Transformer model [31].
- **Weight decay:** Weight decay is used in the corpus size of any language pair is smaller. We didn't use it in the case of En-Et.

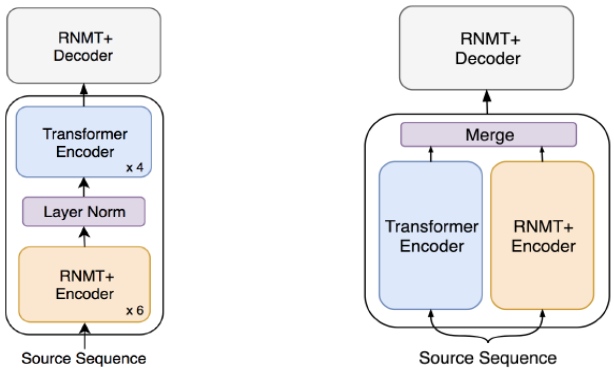

Figure 7.2: Vertical and Horizontal mixing of Transformer and RNMT+ components in an encoder [6].

An RNMT+ can be developed by applying new techniques of RNMT. Similarly, the approach of a hybrid model can be enhanced by merging with other available architectures like Transformer. In a hybrid model, the Transformer can be merged with RNMT+ either vertically or horizontally in encoder [Figure 5.2] and improve the performance.

PyTorch

In this proposed model Pytorch is used in training the model. PyTorch, which is a machine learning framework that was first introduced by Facebook in October 2016. it was designed to provide good flexibility and high speeds for deep neural network implementation. PyTorch uses a dynamic computation graph instead of static computational graphs, which are used in TensorFlow. In TensorFlow static computational graphs are defined before runtime while dynamic graphs in PyTorch are defined through the forward computation.

Advantages of Using PyTorch

In this thesis for all NMT models, I used PyTorch. OpenNMT also inherits the Torch machine learning library. There are many advantages to using PyTorch.

- **Pythonic nature of PyTorch**: Python being one of the fastest-growing programming languages and with most of the machine learning and artificial intelligence-related works being done using python. It makes PyTorch have a wide community.
- **Easy to learn**: just like the python language, PyTorch is considered relatively easier to learn due to its easy and intuitive syntax compared to other deep learning frameworks.
- **Strong community**: Regardless, of its being new, it has developed a dedicated community of developers very quickly with well-organized documentation.
- **Easy debugging**: Since PyTorch uses python, it can utilizing the debugging tools of python.

Data pre-processing

Data processing is a process to prepare raw data and make it suitable for a machine learning model. This is the first step to create a machine learning model. In the real world, raw data might have noise, unusable format, or missing data. Data pre-processing is important to get

rid of those abnormalities in data. Here is an example of a preprocessing script[7] for RNMT+ with Relational-RNN.

```
#!/bin/bash
#
#SBATCH --job-name=train_model
#SBATCH --partition=gpu
#SBATCH --gres=gpu:tesla:1
#SBATCH --time=7-
#SBATCH --cpus=1
#SBATCH --mem-per-cpu=120000

python ./preprocess.py \
-train_src ./en.train.enc \
-train_tgt ./et.train.enc \
-valid_src ./en.dev.enc \
-valid_tgt ./et.dev.enc \
-src_vocab ./en.vocab \
-tgt_vocab ./et.vocab \
-save_data ./data
```

Training a model

After data pre-processing, we can start to train the model. In this stage system train a model based on all available examples with minimum loss following the RNMT+ architecture. Here is an example of a training script. The final trained model is saved in a directory named 'models'. This training run until 350,000 steps with batch size 4096 and hidden size 512.

[7] https://github.com/rezwanshubh/machine-translation/tree/master/rnmt_plus

```
#!/bin/bash
#
#SBATCH --job-name=train_model
#SBATCH --partition=gpu
#SBATCH --gres=gpu:tesla:1
#SBATCH --time=7-
#SBATCH --cpus=1
#SBATCH --mem-per-cpu=120000

python ./train.py -data ./data -save_model ./models/
```

8 Result

Machine translation can be evaluated in two different paradigms. 1. Glass Box evaluation, where quality is measured based on internal system architecture, and 2. Black Box evaluation where quality is measured based on the result without considering the internal system.

Glass box evaluation focuses upon an examination of the system's linguistic coverage and the theories used to handle those linguistic phenomena [32] while on the other hand black-box evaluation focus only on objective behavior of the system upon a predetermined evaluation set. Black Box evaluation is based on two approaches intrinsic and extrinsic measures, which are used to evaluate the accuracy and usefulness of a machine-translation output. Where intrinsic measure evaluates quality based on the output of the MT and often involves quality comparisons between MT output and a set of reference translations that are predetermined to be of high quality [32]. On the other hand, extrinsic measures are focused on testing the effectiveness of an output of a machine translation (MT) model.

Intrinsic measures can further be classified into Human intrinsic measures and Automatic intrinsic measures. Where human intrinsic measures, determine the quality through human subjective judgments of certain characteristics of the output such as fluency and adequacy [32]. Whereas, Automatic intrinsic measures. Use an easily computed sentence similarity measure to produce rankings among machine translation models by comparing the corresponding machine-translation output against a fixed set of reference translations. Various types of Automatic intrinsic measures exist such as BLEU, NIST, METEOR, WER, PER, GTM, TER, and CDER.

In this project, we shall use Bilingual Evaluation Understudy (BLEU) scoring to evaluate the three machine translation models. BLEU scoring evaluates translations by comparing machine translations to human-produced translations. BLEU gives scores between 0 and 1, with 1 being a perfect match to the human translation. These scores are classified into

BLEU-1 for unigrams, BLEU-2 for bigrams, BLEU-3 for trigrams. And BLEU-4 for tetragrams.

Mathematically BLEU score is given by;

$$BLEU = min\left(1, exp\left(1 - \frac{reference - lenth}{output - length}\right)\right)\left(\prod_{i=1}^{4} precision_i\right)^{\frac{1}{4}}$$

With

$$precision_i = \left(\frac{\sum_{snt \in Cand-Compus} \sum_{i \in snt} min\left(m^i_{cand}, m^i_{ref}\right)}{w^i_t = \sum_{snt' \epsilon Cand-Corpus} \sum_{i' \epsilon snt'} m^{i'}_{cand}}\right)$$

Where

$$m^i_{cand} = \text{is the count of i} - \text{gram in candidate matching the reference translation}$$

$$m^i_{ref} = \text{ is the count of i} - \text{gram in the reference translation}$$

$$w^i_t = \text{ is the total number of i} - \text{grams in candidate translation}$$

The two main components of BLEU are n-gram precision and the length of the candidate sentences. In BLEU, precision measures the percentage of the correct n-gram in candidates. The trifling case is unigram (n = 1) accuracy which is just the proportion of the number of tokens shared between candidates and reference divided by the number of tokens in the candidates [1].

BLEU score evaluation using NLTK

NLTK[8] is a python toolkit to process natural language. Here we used the NLTK toolkit to evaluate the BLEU score. Mainly, the references and hypotheses would be compared. Reference is a human translated Estonian file which is a sequence of training data. On the other hand, the hypothesis is the machine-translated file. The source code[9] is there in the repository for better understanding.

In the following tables, we show the calculated BLEU score in two different domains. In the first table, we tested the translation model based on similar type data which was generally used to train the model. The training dataset is a collection of En-Et,

[8] https://www.nltk.org/api/nltk.translate.html

[9] https://github.com/rezwanshubh/machine-translation/tree/master/evaluation

OpenSubtitles[10] v2018. There were 446,612 documents and 3.2G tokens for English and 28,837 documents and 168.2M token for Estonian. In all three cases, the training steps were around 350,000. There were 29,194,732 sentence pairs for training each model. The training dataset in the movie subtitle domain and the EU-law domain contain 3,000 and 1,314 sentence pairs respectively. The training took place on the falcon-1 node of the HPC rocket cluster[11].

Model	BLEU
Sequence to Sequence with Attention	.51
Transformer	.76
RNMT+ with Relational-RNN	.59
Google Translator[12]	.83

Table 1: Model evaluation of the movie-subtitle (general) domain.

We observe the Transformer mode shows a tremendous performance and very close to the performance of Google translate. We have to acknowledge that the comparison would not be accurate since the training data for Google Translator is not the same and definitely, they use a large collection of data set including all possible domains.

Table 2 shows the BLEU score in the EU law domain for the same model.

Model	BLEU
Sequence to Sequence with Attention	.31
Transformer	.21
RNMT+ with Relational-RNN	.38
Google Translator	.64

Table 2: Model evaluation of the EU-law domain.

[10] http://opus.nlpl.eu/OpenSubtitles-v2018.php

[11] https://hpc.ut.ee/rocket-cluster/

[12] Training data for Google Translator is different.

It means, we evaluated a similar model but test data was from the EU law domain which was not included in the training exclusively.

In this case, we see dramatic changes in performance since RNMT+ and Sequence to Sequence model performed slightly better than the Transformer. It is also clear that to translate sentences in any particular domain, our trained model needs to be shaped accordingly. That's why we observe performance loss in all three models when used to translate the EU law. It's also observed that based on all three models, the quality of translation is better in shorter sentences consists of six words or less. The batch size, hidden size, training steps, and other pre-processing and evaluation methodologies remained the same in all training to maintain consistency.

9 Future Work

In this section, we present ideas for future works.

Train a Model Using Unsupervised SMT

As future works, we would consider developing an English-Estonian machine translation model based on Unsupervised Statistical Machine Translation [33]. Functionality-wise statistical machine translation can be compared with rule-based or example-based machine translation. This type of machine translation is quite hard for language pairs like English-Estonian. Hence, developing an unsupervised statistical machine translation model for the English-Estonian language pair will be considerable future work.

Improvement of RNMT+ Model

From the evaluation, we could understand the overall performance of RNMT+ is poor compared to the Transformer in a controlled environment. So, we have many scopes to improve this. In this case, we can organize our corpus with not only movie subtitles but also from different domains.

Train Model

Here I have trained a model using RNMT+ with Relational-RNN architecture. We also can focus on training models for other language pairs (eg: English-Finnish, English-Russian, etc). Since RNMT+ is one of the newest architectures, it would be interesting to train translation models based on different RNN (eg: Relational-RNN, LSTM, GRU) types.

10 Conclusions

The advancement of the RNN model along with machine translation architectures helps us to find the right translation process. It is no longer a dream to develop a translation model for languages with low resources. A neural machine translation system is easy to implement and achieve amazing performance. The overall implementation process of a neural machine translation is simpler than statistical machine translation or rule-based machine translation.

In this thesis, we observed how a new approach of the RNN model can be fit into an existing NMT architecture. Then we explored several architectures and processes to develop a translation model based on these. Finally, we demonstrated the result that a hybrid model like RNMT+ performs better in an unknown domain while the Transformer model outperforms others on OpenSubtitles v2018 En-Et training set.

Acknowledgment

Foremost, I would like to show my gratitude to my supervisors Assoc. Prof. Gholamreza Anbarjafari (Shahab) and Hasan Sait Arslan for their patience, motivation, and enthusiasm. Their knowledge and guidelines helped a lot complete this thesis.

Besides, I would like to thank the HPC center of the Univesity of Tartu for their continuous support and reply. Similarly, thanks go to all available public resources, managers, and developers.

References

[1] K. Papineni, S. Roukos, T. Ward and W.-J. Zhu, “Bleu: a Method for Automatic Evaluation of Machine,” *IBM Research Report,* 2001.

[2] B. Dorr, M. Przybocki, M. Snover, A. Le, G. Sanders, S. Bronsart, S. Strassel and M. Glenn, “Machine Translation Evaluation and Optimization,” in *Handbook of Natural Language Processing and Machine Translation*, 2011, p. 747.

[3] P. Lison and J. Tiedemann, “OpenSubtitles2016: Extracting Large Parallel Corpora”.

[4] P. Koehn, F. J. Och and D. Marcu, “Statistical phrase-based translation,” in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, 2003.

[5] D. Bahdanau, K. Cho and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[6] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen and others, “The best of both worlds: Combining recent advances in neural machine translation,” *arXiv preprint arXiv:1804.09849,* 2018.

[7] G. Diño, “3 Reasons Why Neural Machine Translation is a Breakthrough,” 2017. [Online]. Available: https://slator.com/technology/3-reasons-why-neural-machine-translation-is-a-breakthrough/.

[8] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes and J. Dean, “Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” *CoRR arXiv:1609.08144,* 2016.

[9] M.-T. Luong, H. Pham and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” *CoRR arXiv:1508.04025,* 2015.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017.

[11] V. Ghorakavi, “Neural Networks | A beginners guide,” [Online]. Available: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735-1780, 1997.

[13] M. Nguyen, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation,” 10 Jul 2019. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed 12 Mar 2020].

[14] S. Yan, “Understanding LSTM and its diagrams,” 2017. [Online]. Available: https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714.

[15] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[16] G. Loye, “Gated Recurrent Unit (GRU) With PyTorch,” 22 07 2019. [Online]. Available: https://blog.floydhub.com/gru-with-pytorch.

[17] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[18] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet and F. Nashashibi, “Relational recurrent neural networks for vehicle trajectory prediction,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[19] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu and T. Lillicrap, “Relational recurrent neural networks,” in *Advances in neural information processing systems*, 2018.

[20] I. Sutskever, O. Vinyals and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” *CoRR arXiv:1409.3215*, 2014.

[21] N. Lintz, “Sequence Modeling with Neural Networks (Part 2): Attention Models,” 2016. [Online]. Available: https://indico.io/blog/sequence-modeling-neural-networks-part2-attention-models/.

[22] M.-T. Luong, H. Pham and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv preprint arXiv:1508.04025,* 2015.

[23] R. Jozefowicz, W. Zaremba and I. Sutskever, “An empirical exploration of recurrent network architectures,” in *International conference on machine learning*, 2015.

[24] T. Shah, “About Train, Validation and Test Sets in Machine Learning,” 2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[25] S. Robertson, “NLP from scratch: Translation with a Sequence to Sequence Network and Attention,” 2019. [Online]. Available: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.

[26] D. Mwiti, “Research Guide for Transformers,” Oct 2019. [Online]. Available: https://www.kdnuggets.com/2019/10/research-guide-transformers.html. [Accessed 03 Mar 2020].

[27] G. Klein, Y. Kim, Y. Deng, V. Nguyen, J. Senellart and A. M. Rush, “Opennmt: Neural machine translation toolkit,” *CoRR arXiv preprint arXiv:1805.11462,* 2017.

[28] M. Domingo, M. Garc´ıa-Mart´ınez, A. Helle, F. Casacuberta and M. Herranz, “How Much Does Tokenization Affect Neural Machine Translation?,” *CoRR arXiv:1812.08621,* 2019.

[29] T. Kudo and J. Richardson, “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing.,” 2018.

[30] W. Zaremba, I. Sutskever and O. Vinyals, “Recurrent neural network regularization,” *CoRR arXiv preprint arXiv:1409.2329,* 2014.

[31] J. Chorowski and N. Jaitly, “Towards better decoding and language model integration in sequence to sequence models,” *CoRR abs/1612.02695,* 2016.

[32] B. Dorr, M. Snover and N. Madnani, “Part 5: Machine Translation Evaluation,” *Dostopljeno na: https://www. cs. cmu. edu/˜ alavie/papers/GALE-book-Ch5. pdf [8. 8. 2018],* 2006.

[33] M. Artetxe, G. Labaka and E. Agirre, “Unsupervised statistical machine translation,” *arXiv preprint arXiv:1809.01272,* 2018.

[34] M. R. Islam, “Source code: English-Estonian Machine Translation: Evaluation Across Different Models and Architecture.,” 2020. [Online]. Available: https://github.com/rezwanshubh/machine-translation.git.

License

Non-exclusive licence to reproduce thesis and make thesis public

I, Md Rezwanul Islam,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   English-Estonian Machine Translation: Evaluation Across Different Models and Architectures.,

   supervised by Assoc. Prof. Gholamreza Anbarjafari (Shahab) and Hasan Sait Arslan.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Md Rezwanul Islam

20/05/2020