UiA University of Agder

Master

Artificial Intelligence for Sign Language Recognition and Translation

A Multimodal Machine Learning Approach to Continuous Sign Language Recognition and Translation.

ANDREAS HAGLUND

SUPERVISOR
Christian W. Omlin

University of Agder, 2022
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Jeg erklærer herved at min besvarelse er mitt eget arbeid, og at jeg ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Jeg erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Jeg er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Jeg er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Je er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Jeg har satt meg inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Jeg gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

I would like to thank Sigurd Kristian Brinch for helping me with access to sufficient hardware through UiA, to Shikhar Bharadwaj for helping me with the Evolved Transformer implementation and to Rohitash Chandra for correspondence on sentiment analysis. I am also grateful to Arthur Tennøe, Andre Kåsen and Jon Arild Olsen from Nasjonalbiblioteket for following up on my project.

A special thanks to my supervisor Christian W. Omlin who has guided me through the project with a lot of positive energy.

Abstract

In a world where people are more connected, the barriers between deaf people and hearing people is more visible than ever. A neural sign language translation system would break many of these barriers. However, there are still many tasks to be solved before full automatic sign language translation is possible. Sign Language Translation is a difficult multimodal machine translation problem with no clear one-to-one mapping to any spoken language. In this paper I give a review of sign language and its challenges regarding neural machine translation. I evaluate the state-of-the-art Sign Language Translation approach, and apply a modified version of the Evolved Transformer to the existing Sign Language Transformer. I show that the Evolved Transformer encoder produces better results over the Transformer encoder with lower dimensions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sign language is the use of the body - hands, arms, fingers, head, posture, facial expression - in order to communicate in a nonverbal manner. According to the World Federation of the Deaf there are more than 200 sign languages, used by 70 million deaf people around the world [32]. Sign languages are complex visual languages which are very different from spoken language. Carrying their own grammatical rules and often different word orderings than spoken languages, there is no one-to-one mapping from signs to spoken language words [36].

Sign language translation is the idea of automatically recognising and translating sign languages into spoken text, or to generate signs from spoken text using neural machine translation. Neural machine translation has already approached human performance on translation of natural languages. Google Translate is capable of translating more than 100 languages, yet none of them are sign languages. A sign language translation system is needed in order to proceed towards universal translation and its ambition to communicate in all spoken, written and signed languages.

Deaf people often face challenges when interacting with hearing people. Most hearing people do not know a sign language. Furthermore, modern communication technologies are mostly being designed to support only spoken or written languages, which reinforces the existing communication barriers between deaf and hearing people. A neural sign language translation system would therefore greatly benefit the society, and break down social barriers between deaf and hearing people [5].

Many research efforts have been conducted in the field of sign language translation over the recent years. However, many of them focusing on only recognising hand gestures, or static signs, whereas the goal is to translate continuous sign language sentences. Due to the multimodal nature of the task, and due to the complexity of the sign languages and its distinctions from spoken languages, continuous sign language translation appears to be a difficult task even with modern machine translation architectures. [36].

In this thesis I will cover some of the challenges in continuous sign language translation, both technical and ethical. Furthermore, I will look into some work done in the field, going in depth into the state-of-the-art joint end-to-end sign language recognition and translation approach by Camgoz et al. [8]. Lastly, I will present my own contribution, where I apply a modified version of the Evolved Transformer [39] to the Sign Language Transformer model by Camgoz et al.

Chapter 2

Background

Sign Language Translation (SLT) is a challenging field for several reasons. In this chapter I will provide useful historical information about the Deaf culture, I will provide an overview of sign language and its complexity, and I will target some of the main challenges of creating sign language translation systems.

2.1 The Deaf Culture

Sign language users make up cultural minority groups with common language and life experience. In surroundings with only sign language speakers, deaf people have no disabilities. Deafness is therefore viewed as a cultural identity, rather than a disability. However, in interaction with non sign language speakers, both groups have disabilities [18].

We distinguish between capitalized "Deaf", referring to the Deaf cultures, and lowercase "deaf", referring to the audiological status of a person. Sign languages are sacred in the Deaf cultures. Development of sign language processing systems is therefore a highly sensitive task which should be developed with care and respect [5].

Historically, the Deaf community have experienced suppression of sign language communication. In late 1800s, an international congress decided that deaf students should only be educated using spoken language. Students were therefore trained to lip-read and speak, with varying level of success. Since then, it has been a passionate cause for the Deaf to use sign languages in school, work and public life. All this historical baggage can make the development of sign language software particular sensitive in the Deaf communities [5].

2.2 The Complexity of Sign Languages

Sign languages are naturally evolved languages, distinct from spoken languages, making the mapping from sign language to spoken language very hard. In order to understand the complexity of the language it is essential to take a brief look at some of the linguistics of the language.

Sign languages are composed of phonological features put together under certain rules, just like spoken languages [5]. A sign word consists of five parameters, which all have to be performed correctly. These are hand shape, palm orientation, movement, location, and facial expression. Signs use multiple channels to convey concurrent information. For instance, space and direction is used to convey relationship between objects [36].

Classes of nouns and verbs are represented by different classifiers. For instance, one handshape is used for vehicles and another one for flat objects. These handshapes can also be combined with movements to indicate how the objects move. These shapes and movements, however, are not reserved only for classifiers, but can appear in other signs. It is therefore important to catch the context in which a sign is performed [5].

In addition to signs, sign languages use fingerspelling for names and words they do not have signs for. While there is a more clear one-to-one mapping between fingerspelling letters and spoken letters, the fingerspelling is subject to coarticulation, meaning its handshape may vary, depending on the neighbouring letters. Sign language processing software must therefore be able to detect these variations, and also be able to separate fingerspelling from other signs [5].

Another important factor of sign language is the facial and body expressions. This involves movement of eyebrows, mouth, eyes, head and shoulders. For instance, eyebrows can be used to formulate a sentence as a question, while mouth position can indicate the size of an object. The body is used actively to depict different actions, such as filleting a fish [5]. Signs and fingerspelling can be subject to partial occlusion, meaning that parts of a sign might be hidden by the hand or other fingers. In such cases, facial and body expressions might add the extra layer of information needed in order to interpret the meaning of the sign. It is therefore essential to do translation on full body videos, not only hand gestures videos.

Furthermore, sign languages vary a lot based on ethnicity, geographic region, age, gender, education, hearing status etc. There are not only different sign languages, such as American Sign Language (ASL) and Norwegian Sign Language (NSL). There are different dialects within the different languages. For instance, Black ASL is different from ASL [5].

Lastly, sign languages differ from spoken languages in its variety in fluency. Deaf children are usually born to hearing parents, who do not know sign language. Deaf children, and their relatives (typically parents and siblings) are therefore not taught sign language until late childhood. This often results in lower fluency for all parts. Consequently, SLT systems must model and detect this vast variety, meaning we need datasets that reflect this variety [5].

2.3 Datasets

One of the biggest challenges in the field of SLT is the lack of good, public sign language datasets which reflect the variations in the language. Most machine learning techniques work best with large amount of data. However, most sign language corpora contain fewer than 100,000 signs. In comparison, speech recognition has had great success because it has been trained on corpora containing millions of words [5].

There are several challenges with existing sign language datasets. First of all, many of the datasets only contain individual signs. These might be good for sign language dictionaries, but they will not work well for continuous SLT. To be able to translate a real world sign language conversation we need datasets containing continuous sign language sentences, showing the coarticulation and that the meaning of certain signs might change in different contexts [5].

Secondly, there is a lack of native signers contributing to the datasets. Many of the signers are either novices, or professional sign language interpreters. Their language simply do not reflect the language used by native signers well enough. Their language vary in speed and fluency. In order to reflect the language of the native speakers, datasets contribution by native signers are needed [5].

Thirdly, datasets also lack variety. To be able to create generalizable models, the datasets must reflect the signing population. This means that datasets should include signers of different gender, skin tone, age, clothing, body size etc. Datasets must also capture different sign language dialects as well as different sign languages. There should also be variations in camera quality, camera angles and lightning conditions in order to make the software usable in real life situations [5].

Lastly, the datasets need annotations in order to train AI models with supervised learning. Annotating sign language datasets is time consuming and error prone, because no standard annotation system for sign languages exists. This is a major restriction in the field, and it prevents researchers from combining datasets, which for other machine learning tasks would greatly increase the power of the model. Existing datasets use different levels of annotations. Some are annotated with full spoken text translations, while others have gloss-level annotations (see Chapter 3.1 for an explanation of glosses.), capturing the sign identity and the same order as in spoken language sentences [5].

Chapter 3

Related Work

Some of the earliest research on hand and gesture recognition is dated back to 1987, proposed by Zimmerman et al. [44], measuring finger bending, positioning and orientation, using a glove containing flex sensors. Giant leaps have been made in technology since then, and with better cameras, faster computers and with the advent of deep learning, current research is now capable of recognising hand gestures through video or image-input, rather than sensor/glove-based systems. However, much of the research is only focusing on static signs, and just hand gestures. Sign language is a continuous language, utilizing the full body, not only hand gestures. For the purpose of this thesis it is therefore most relevant to look into work done on continuous sign language recognition/translation.

3.1 Sign Language Recognition and Sign Glosses

The ultimate goal of Sign Language Translation (SLT) is to generate fluent spoken language translations from sign language videos. However, because there is no easy one-to-one mapping between signs and spoken language words, many researchers have instead been focusing on Sign Language Recognition (SLR). The goal of SLR is to identify sign glosses of a sequence of continuous signs, neglecting the grammatical and linguistic structures that differ from spoken language [7]. It is therefore an easier task than SLT.

Sign glosses are used to associate a word (or words) with a sign, in order to give the sign a label. The gloss word do not necessarily convey the meaning of the sign, but it is often a good approximation. Glosses are not very tight bundled to their signs. There might be several good candidate glosses for the same sign. For instance, the words "important", "worth" and "value" are all good candidates for the same sign. It is essential to understand that there is no one-to-one mapping between signs and words. Signs convey meanings, not just words. It is therefore practical to identify signs by glosses [30].

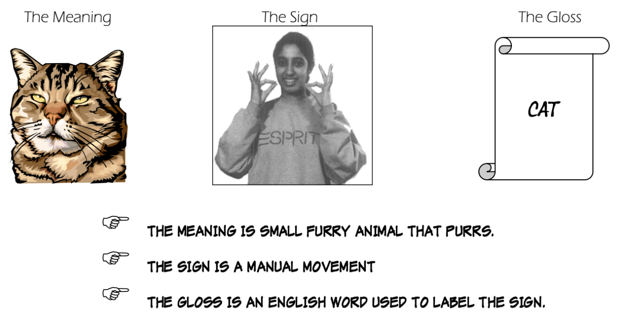
The Meaning
The Sign
The Gloss
CAT
THE MEANING IS SMALL FURRY ANIMAL THAT PURRS.
THE SIGN IS A MANUAL MOVEMENT
THE GLOSS IS AN ENGLISH WORD USED TO LABEL THE SIGN.

Figure 3.1: Sign glosses are simply just labels on signs, making it easier to talk, read and write about sign language [30].

3.2 Datasets

A few large datasets containing continuous dynamic sign language has been released in recent years, attracting more researchers into the field. The most interesting one is the RWTHPHOENIX-Weather-2014 (PHOENIX14) dataset, which is a continuous sign language recognition dataset conducted by the German public TV-station PHOENIX. The corpus is a subset of recordings from the daily weather forecast airings, recorded over a three year period from 2009 to 2011. It has been transcribed only with gloss annotation, making the dataset suitable for SLR [28].

Camgoz et al. [7] released PHOENIX14**T** in 2018, which is an extension of PHOENIX14, containing German spoken language translations as well as the gloss annotation. This is the first publicly available continuous SLT dataset.

PHOENIX14 and PHOENIX14**T** has been a major contribution to SLR and SLT, and are some of the most used datasets in the fields. The recordings are performed in a studio with a stationary color camera. The interpreters wear dark clothes, and are placed in front of a grey background with color transition [28].

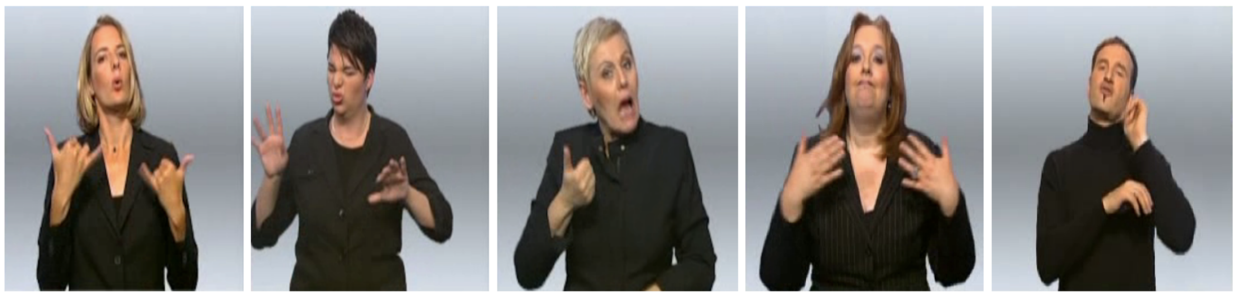

Figure 3.2: Example frames from PHOENIX14(**T**). Videos are recorded at 25 frames per second. The frame size is 210 by 260 pixels. Each frame only captures the interpreter box [28].

3.3 Continuous Sign Language Recognition

Several approaches have been proposed in the last years, using different architectures. With the advent of deep learning, researchers in SLR have quickly adopted CNNs for manual and non-manual feature representation, while Recurrent Neural Networks (RNN) have been adopted for temporal modelling [7].

The development of sequence to sequence (seq2seq) learning approaches was one of the most important breakthroughs in deep learning. Seq2seq problems introduced a new challenge regarding annotations. Strong annotations are hard to obtain for sequences. It is simply too time consuming annotating each frame of a sequence. To deal with this problem, Graves et al. proposed a new loss function, namely the Connectionist Temporal Classification (CTC) Loss [20]. The CTC considers all the possible alignments between the source and target sequences while calculating the error. Chapter 4.2.2 explains CTC in more details.

The CTC has become a dominant loss function for many seq2seq tasks, such as speech recognition and hand writing recognition. In recent years, CTC has also been used by computer vision researchers on weakly labeled visual problems like lip reading, hand shape recognition and CSLR [7].

A common seq2seq task is machine translation, aiming to learn the mapping between two languages. Because CTC assumes that the source and target sequences share the same order, it is not a suitable loss function for machine translation problems. CTC also assumes that there is no relationship between words in the target sequence. It is therefore not possible to learn an implicit language model using CTC. This resulted in the development of the Encoder-Decoder architecture, which led to the emergence of the Neural Machine Translation (NMT) field [7].

Encoder-Decoder networks use an intermediary latent space to map from one sequence to another, similar to the intermediate representation in auto-encoders [7]. A latent space is just a compressed representation of data in space, where similar data are closer together in space [40]. The first Encoder-Decoder architectures used RNNs to both the encoder and the decoder. These approaches improved machine translation performance, but introduced an information bottleneck in the encoding of the source sequences into a fixed sized intermediate vector. To deal with these issues, various attention based architectures have been proposed, calculating the alignment between source and target sequences [7].

Camgoz et al. (2018) [7] used an attention based architecture to realize the first end-to-end SLT models. They used CNNs in combination with attention-based NMT methods. In 2019, Ko et al. [26] proposed a similar approach, but using human keypoint extractions of face, hands and body parts as input to the network. They evaluated their method on their newly introduced dataset containing Korean sign language, namely the KETI sign language dataset [26].

In the mean time, Vaswani et al. introduced transformer networks, which took NMT to new heights [41]. Transformers improved the performance of translations over the RNN based encoder-decoder architectures. Additionally, transformers are faster and easy to parallelize compared to the old architectures. This has made transformers the go to architecture for many machine translation tasks [8].

Inspired by the success of transformers, Camgoz et al. (2020) [8] proposed a novel architecture using multiple co-dependent transformer networks, reporting state-of-the-art SLR and SLT results. The transformers are simultaneously trained to jointly solve related tasks. This architecture is used to simultaneously recognise and translate sign language. The encoder learns to recognize sign gloss representations, while the decoder learns spoken language translations from the sign gloss representations [8]. The work of Camgoz et al. is used as inspiration for my own work. Their approach is therefore described in more details in chapter 5.

Lastly, Coster et al. applied a pretrained BERT model to the work of Camgoz et al. and report an increase of 1 to 2 BLEU-4 score. To avoid overfitting, the majority of parameters were frozen during training, known as frozen pretrained transformer technique [12].

Chapter 4

Theory

In this chapter I will explain some of the theory behind the main components used in the models I have been working with.

4.1 Convolutional Neural Network

One of the most popular deep learning approaches is the Convolutional Neural Networks (CNN). It is known for being highly effective on diverse computer vision applications. A CNN is typically composed of three different layers, namely convolutional layers, pooling layers and fully-connected layers. The first layers learn simple features like colors and edges, while later layers put together the edges and colors to learn larger shapes and elements [15]. In this section I will explain the three different layers of a CNN.

4.1.1 Convolutional Layer

The main building block of a CNN is the convolutional layer. The convolutional layer involves three component: the input data, a filter and a feature map. A common input to a CNN is a colour image. The input will be a matrix with a height and width corresponding to the image height and width, and a depth of 3, one for each color channel (RGB) [15].

The filter, or kernel, is just another matrix containing weights. If we view the image matrix as a box, then we can view the filter matrix as a smaller box which fits inside the image box. Usually, the filter has the same depth as the image, meaning 3 in this case. This means that the filter-box can not change its direction in the z-direction (depth). The length and height of the filter is usually the same, commonly $3x3$, $5x5$ or even $1x1$ [15].

The process of convolving is simply just to move the filter around in the larger box (the image) calculating the dot product between the filter and the pixel values it encapsulates. Starting in the top left corner of the image, moving one pixel to the right for each calculation. When the end is reached, the filter moves back to start, and one pixel-row down, doing the same on the next rows. If the image is of size $10x10$, and the filter is of size $3x3$, it means that the filter can move $10 - 3 = 7$ times in x and y direction, giving us 8 possible alignments of the filter in each direction. This gives us a $8x8$ output matrix with the dot products from the convolving process. The output matrix is known as the feature map of the image [15].

Two additional parameters in a convolutional layer is stride and padding. The stride defines how many steps the filter should move for each calculation. The default is 1. With a higher stride, the output dimension decreases. Padding can be added around the image if the output dimension should be the same as the input dimension. Otherwise, the output dimension will be reduced with $input\_dimension - filter\_size + 1$ [15].

The depth dimension can be increased by adding multiple filters, appending the outputs of each filter at the end [15].

4.1.2 Pooling Layer

A pooling layer is a way to reduce the number of parameters in the input. It works similar to a convolutional layer, but it does not contain any weights. Instead, the kernel applies a function to the numbers it embraces. There are mainly two types of functions used by the pooling layer: max pooling and average pooling. Max pooling simply filters out the maximum number in the "pool", while average pooling takes the average of all numbers in the "pool" [15].

A lot of information disappears with the pooling layer. However, it reduces the complexity, improves the efficiency and reduces the chances of overfitting [15].

4.1.3 Fully-Connected Layer

The fully-connected layer is just a normal feed-forward fully-connected layer which is used as the last layers in order to do the classification [15].

4.2 Loss Functions

Loss functions are essential components in machine learning. They are used to compute the distance between the current output of the algorithm, and the expected output. This calculation can be used to measure how well an algorithm is doing its job. The measurements are used as feedback to the algorithm in order to adjust the algorithm. In this section I will briefly cover two loss functions, which will be referred to in this paper [34].

4.2.1 Cross-Entropy Loss

Cross-Entropy Loss is used to optimize classification models. Entropy refers to the level of uncertainty in a random variable. If a random variable has equal probability for all its outcomes, for instance a dice, the uncertainty is high, yielding an entropy of 1. On the other side, if a random variable only has one possible outcome, the uncertainty is low, yielding an entropy of 0. The cross-entropy loss is defined as

$$L_{CE} = -\sum_{i=1}^{n} t_i \log_2(p_i) \tag{4.1}$$

where t_i is the truth label and p_i is the probability for the i^{th} class. The goal is to lower the uncertainty, meaning a cross-entopy loss of 0 would be a perfect model [27].

4.2.2 Connectionist Temporal Classification

The connectionist temporal classification (CTC) loss function is commonly used in seq2seq models, when the alignment between input and output data is unknown. Formally, when an input sequence $X = [x_1, x_2, ..., x_T]$ is being mapped to an output sequence $Y = [y_1, y_2, ..., y_U]$, and the length T and U vary, CTC can be used to find an accurate mapping from X to Y. The CTC algorithm is therefore said to be alignment-free, because it does not require an alignment between the input and the output [21].

To understand CTC loss we first have to understand frame-level annotation. Figure 4.1 shows a frame-level annotation of a text input sequence. By removing all duplicate characters we get the word in the image, "to". To be able to differentiate on words like "to" and "too", a blank character, denoted by "_" can be used to separate each character. When removing duplicate characters, we only remove until the blank [37].

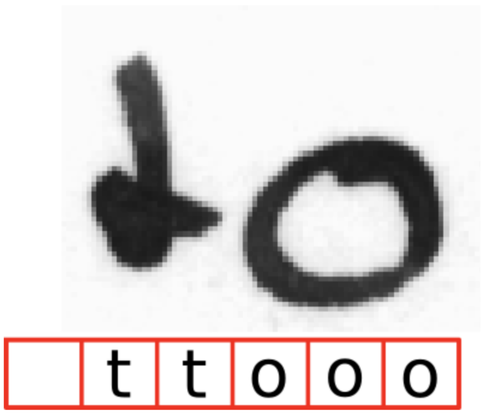

Figure 4.1: Frame-level annotation [37].

Another key to understanding CTC is that different aligmnet sequences can be decoded as the same word. For the example in Figure 4.1, "tt_ooo" and "ttt_oo" both ends up as "to" when duplicates and blank is removed [37].

The CTC loss function receives a matrix from the network, mapping each time-frame of a sequence to probabilities for each possible characters (glosses in the case of SLR). It is not essential for the CTC to find the correct alignment. Its only job is to train the network to output the correct word [37].

To find the word with the highest probability we have to find all the alignments which gives the same final word, and calculate the probability of each of these alignments by multiplying together the probabilities for each character in each time-frame. By summing up the probabilities for each sequence giving the same word, we get a probability for that word. However, for long sequences this is very time and computationally inefficient [37].

Another solution which is commonly used is called Best Path Decoding". It selects the character with the highest probability at each time step and constructs a sequence of these. While this often work as a good approximation, the probability might not always be higher than the joint probability calculated for each word in the last method [37].

Camgoz et al. use CTC loss with something called Beam Search. Beam Search is similar to Best Path Decoding, but instead of chosing the option with the best option at each time step, it chooses the n best option at the first time step, where n is the beam size. Then it calculates the joint probability for each of the three selected options with all the possible options at the next time step. The n best joint probabilities are then selected and continues to the next time step. This continues until the last time step is reached, where the best of the n paths that are left will be chosen [14].

Camgoz et al run beam search for all beam sizes from 1 to 10 [8].

4.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a neural network architecture used for detecting patterns in sequential data. The main difference between a normal Feedforward Neural Network is that the RNN pass information through the network in cycles, making information from previous inputs available for the next input [38].

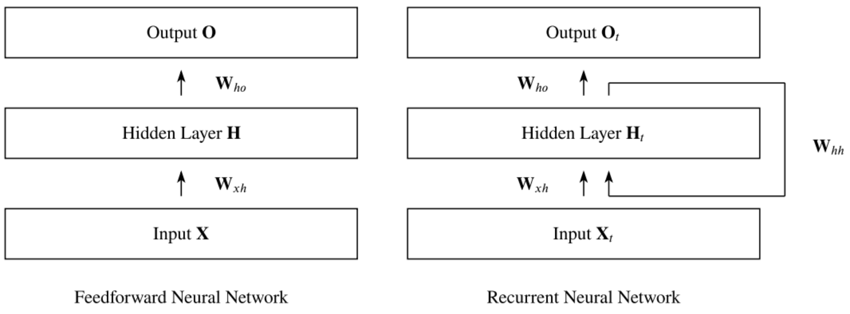

Figure 4.2: A visualisation of differences between a Feedforward NN and a RNN [38].

There are some challenges with RNNs, which make them less appropriate for certain tasks. First of all, RNNs take data sequentially, making the algorithm slow. Because all inputs rely on the preceding inputs, all calculations have to be performed sequentially. Another challenge is that for long sequences, RNNs have trouble capturing long term dependencies. Small values in the matrix multiplications cause the gradients to decrease with each layer, and finally vanishing. This is known as the vanishing gradient problem [38].

Long Short-Term Memory units (LSTMs) were introduced to tackle the latter problem. However, it still suffers from its sequential nature. LSTMs are not essential for the rest of this paper, and will therefore not be described in depth [38].

4.4 Transformer

Transformers are neural sequence transduction models which take input sequences parallel, rather than sequential, like RNNs. Similar to most competitive sequence transduction models, the transformer has an encoder-decoder structure. The encoder maps an input sequence $(x_1, ..., x_n)$ to an intermediate representation sequence $(z_1, ..., z_n)$. The decoder then takes z as input, and generates an output sequence $(y_1, ..., y_m)$, one element at the time. The decoder is auto-regressive, meaning that it takes the previously generated tokens as additional inputs when generating the next token [41].

4.4.1 Encoder

The left side of Figure 4.3 shows the encoder. The encoder consists of $N = 6$ identical layers. Each of them has one multi-head self-attention layer and one position-wise fully connected feed-forward network. Additionally, the encoder contains residual connections around each sublayers, followed by layer normalization. Both the embedding layer and the sub-layers produce outputs of dimension $d_{model} = 512$, in order to make the residual connections work [41].

4.4.2 Decoder

The right side of Figure 4.3 shows the decoder. The decoder is similar to the encoder layer, but it contains one extra layer, and a modified attention layer, called masked multi-head attention. The extra layer is the top most multi-head attention layer, which attends to the output of the encoder stack. The masking of the masked multi-head attention ensures that the predicted token at position i can depend only on the known outputs at positions less than i [41]. Chapter 4.4.7 explains the masking.

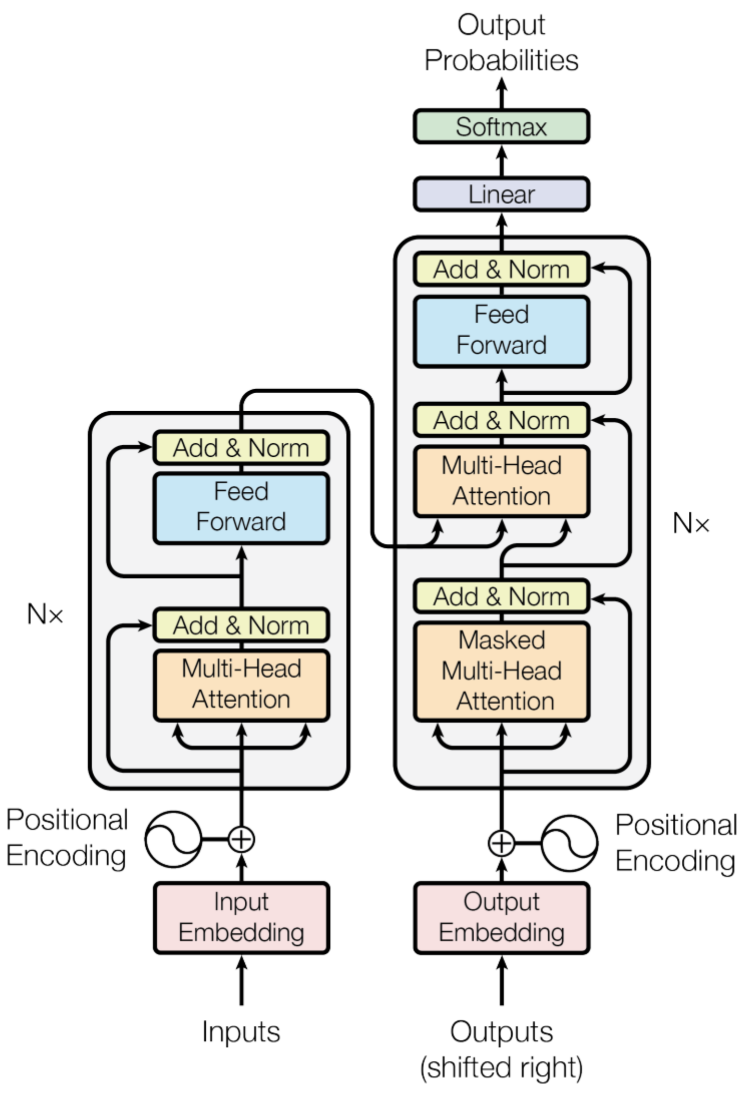

Figure 4.3: The Transformer architecture [41].

4.4.3 Self-Attention

The self-attention layer is a sequence-to-sequence operation mapping input vectors $x_1, x_2, ..., x_t$ to output vectors $y_1, y_2, ..., y_t$. The input vector is of same dimension as the output vector. For each x_i a corresponding y_i is calculated by taking a weighted average over all the input vectors as:

$$y_i = \sum_j w_{ij} x_j. \tag{4.2}$$

where j indexes over the input sequence x, and the weights sum to one. The weight w_{ij} can in its simplest form be the resulting matrix of the dot product between x_i^T and x_j [4]. However, the self-attention used in modern transformers are a little more complicated.

To explain how the weight w_{ij} is computed it is essential to introduce three new terms. In Figure 4.4, every x_i is used in three different ways. They are compared to all other vectors to establish the weight for its own output y_i, it is compared to every other vector to establish the weights for their outputs, and it is used together with the corresponding weights to compute each output vector. These three usages are called the *query*, *key* and *value*, respectively [4].

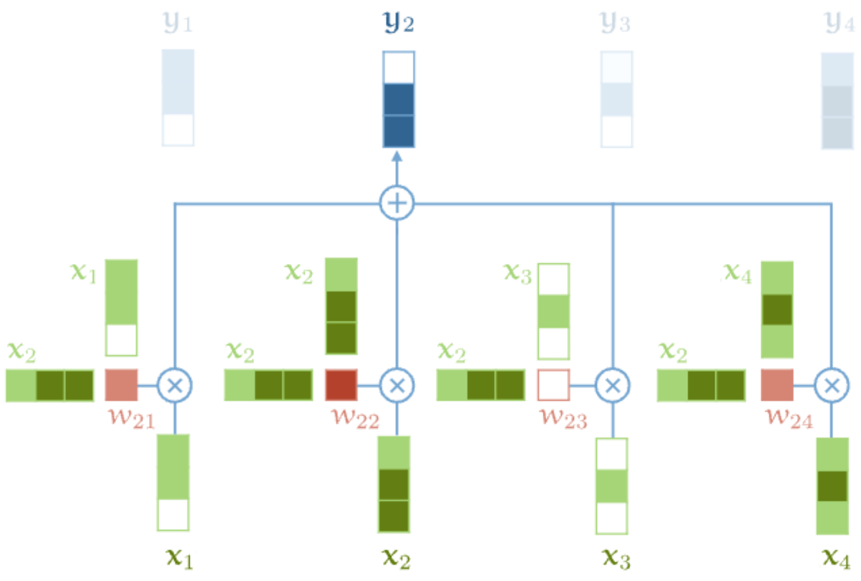

Figure 4.4: The simplest form of self-attention without any trainable weights [4].

In the simple attention, in Figure 4.4, each input vector is used as both query, key and value parameters. In order to get some learnable weights, the original attention mechanism introduce three weight matrices W_q, W_k and W_v and derive new query, key and value matrices as:

$$q_i = W_q x_i \qquad k_i = W_k x_i \qquad v_i = W_v x_i$$

The weight w_{ij} is then calculated as

$$w_{ij} = softmax(\frac{q_i^T k_j}{\sqrt{k}}).$$

The dot product produces values between $-\infty$ and ∞. We apply a softmax to yield values between 0 and 1, and to ensure that they sum to 1 over the whole sequence [4]. The softmax is sensitive to very large input values. We therefore divide the dot product on $\sqrt{k}$, because the average value of the dot product grows with the embedding dimension k [4]. The final vectors are then obtained by

$$y_i = \sum_j w_{ij} v_j.$$

To understand why self-attentions work, I will use an example provided by Peter Bloem [4], showing how movie recommendation systems work. Suppose we have a three-dimensional vector for every movie, containing values for how much romance, action and comedy the movie contains. Likewise, for each user in the system we have a corresponding three-dimensional vector showing how much the user likes these three properties. Values are from -1 to 1. By taking the dot-product of these two vectors we get a score showing how good of a match the movie is to a user [4].

The self-attention uses the same principle for word embeddings. Given learned word embeddings, the self-attention layer calculates the dot-product between each of them to find the relation between words in the sentence. For instance, in the sentence “the cat walks on the street”, the learned embeddings will be more similar for “cat” and “walks”, as these words relate to each other. The dot-product of two similar vectors will therefore be large, telling the system that these words are related, and should pay extra attention to each other.

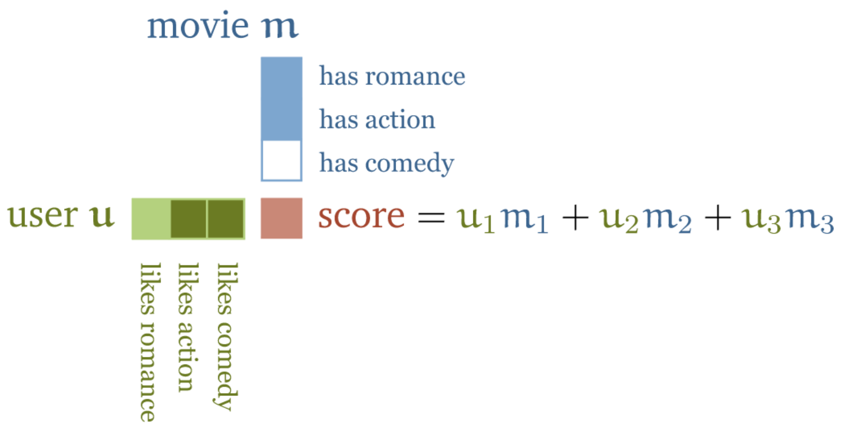

Figure 4.5: Dot-product between user vector and movie vector [4].

4.4.4 Multi-Head Attention

The transformer is using multiple self-attention layers. In the original paper they employ 8 parallel layers [41]. The reason they do this can be illustrated with another example by Peter Bloem [4]. Consider the sentence "Mary gave roses to Susan". The word "gave" has different relations to "Mary" (the giver), "roses" (the gift) and "Susan" (the receiver). In a single self-attention, all these relations are summed together. If the sentence changes, say Susan gave roses to Mary instead, the output vector would still be the same, even though the context has changed.

By combining several self attention mechanisms we can capture all these different relationships. All the different heads have their own weight matrices. In order to make a multi-head attention almost as fast as a single-head attention, the input vectors are split into chunks of equal size. This means that for a traditional 8 head attention, and a input vector of dimension 512, each chunk will be of size $512/8 = 64$. The weight of each self attention layer is adjusted to fit the sizes of the chunks. After all the self-attention layers have generated their y-values, the resulting vectors are appended together to form a 512 dimensional vector again [4].

4.4.5 Word Embedding

Machine learning models are mathematical functions which take vectors or scalars as input, and produce vector or scalar outputs. Therefore, when working with text, there is a need to convert words into vector representations, which in turn can be fed into the network. One simple solution to this could be to simply create one-hot encodings. One-hot encodings are just vectors of zeros, except for a one at the position of the word it represents in the vocabulary. Say we have a vocabulary of the words "have", "a", "good", "great", "day". The one-hot encoding for the word "have" will have a one in the first position, and zeros in the remaining four positions [23].

One-hot representation of words work fine if the words are just different classes in for instance an image classification problem. When working with NLP (natural language processing), however, there is a need for some information about the relations among the words. Using the same words as above, say we have a machine translation system outputting the sentence "have a good day", but the reference sentence is "have a great day". With no information about the relations among words, the words "good" and "great" will be treated as total different words.

The solution to this is word embedding. Word embedding are learned representations of words where similar words have a similar representation. A method called Word2Vec, introduced by Mikolov et al. in 2013 [31], is the de facto standard method for developing pre-trained word embedding [6]. In the same paper, it was found surprisingly mathematical relationships among the vectors. It turned out that by subtracting vector("Man") from vector("King"), and adding vector("Women") resulted in a vector closest to the vector representation of the word "Queen" [31].

4.4.6 Positional Encoding

Since the transformer model is parallel, and not recurrent, information about the words' position in the sentence is lost. In recurrent models, the tokens go into the model one at a time, and therefore the relevant position of the tokens are known to the model. In transformer models, however, all the tokens are fed in at the same time, eliminating the positional information. To make up for this loss, the transformer is using something called positional encodings. These are just simply vectors which are added to the input embedding. The positional encoding therefore must have the same dimension as the input embedding, d_{model} [24]. So what exactly do the positional encoding vectors contain?

There are many options for positional encodings, both learned and fixed. Positional encodings should, however, satisfy some criteria. First of all, the encodings should be unique for each position. Secondly, the distance between any two positions should be the same across sentences with different lengths. Thirdly, the model should be generalizable to longer sentences, meaning there values should be bounded. Lastly, the model must be deterministic [24].

Vaswani et al. proposed a new method which satisfies all of these criteria, using sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos is the position and i is the dimension [41].

Figure 4.6 demonstrates how the positional encoding with sinus and cosine curves work. When $d_{model} = 512$ there are 512 different curves to read the values from, where each curve corresponds to each index in the output vector. For demonstration purpose, only six curves are included with a small spread in the chosen indexes, to avoid the curves to overlap too much. The figure is created in Python using matplotlib, and inspiration is taken from the online resources of Chapter 10.6.3 in the book "Dive into Deep Learning" [43].

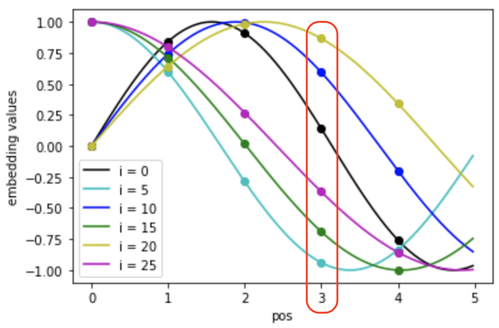

Figure 4.6: Positional encoding values for pos = 3.

4.4.7 Masking

Masking is used in the transformer in order to make the training go faster. The decoder is an auto-regressive model which gets the previously generated words when generating the next words of a sentence. During training, however, a technique called "teacher forcing" is used. Instead of giving the decoder the previously generated word, which most of the times is wrong during training, the decoder instead receives the previous ground truth word. This prevents the decoder from generating consequential errors [41].

Because the decoder now has all the words that normally is fed to the decoder sequentially, available at the beginning, it is not necessary to run this process sequentially. They can be run in parallel. However, it requires that for each position the decoder is generating a word for, it can only get access to the preceding words, and not the next words in the sentence. This is where masking comes in. Masking is applied to hide the next words of a sentence.

4.4.8 Residual Connections

Residual connections are used when training deep network in order to avoid gradients to shrink or increase exponentially, known as exploding or vanishing gradients. It is simply just a side-path for letting some data skip certain layers of the network. The data is then added to the data flowing throgh the normal path. Mathematically this can be expressed as $F(x) + x$ [42].

4.5 Measurements

One key part of machine translation is measuring translation performance. This is challenging due to the big variety in languages. The same message can be formulated in several different ways, without one being more correct than the other. Take for instance the Norwegian sentence "hunden spiser kjeks". Two possible English translations of this sentence are "the dog eats cookies" and "the dog is eating cookies". To correctly measure the performance of these two translated sentences we need a method to decide how good these translations are [13]. Camgöz et al. [8] are using BLEU score and WER for this purpose.

4.5.1 BLEU score

Papineni et al. [33], proposed a method called BLEU (BiLingual Evaluation Understudy) score. The central idea behind this method is to compare the machine translation to a professional human translation. The more similar they are, the better the translation is. In other words, this method relies on a corpus of good quality human translations, and it needs a numerical metric score for determining the closeness of the machine translation to the human translation [33].

The main part of calculating the BLEU score is to compare n-grams in the candidate sentences to n-grams in the reference sentences, and count the number of matches [33]. n-grams are simply all the samples of n consecutive words in a sentence. For instance, the 2-grams of the sentence "the dog eats cookies", are "the dog", "dog eats", "eats cookies".

The precision of the sentence is calculated for each n-gram using a slightly modified precision metric known as modified n-gram precision [33]. In modified n-gram precision, similar n-grams in the candidate sentence are only counted for the same number of occurrences in the reference sentence. Meaning if the word "the" occurs three times in the candidate sentence, but only once in the reference sentence, it is only counted once. This score is known as the clipped count [33].

$$Count_{clip} = min(Count, Max\_Ref\_Count). \tag{4.3}$$

The BLEU score is evaluated on blocks of text. This is done by summing up the clip counts for each n-gram in every candidate sentence in the block, and then dividing on the number of n-grams in total.

$$p_n = \frac{\sum\limits_{C \in \{Candidates\}} \sum\limits_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum\limits_{C' \in \{Candidates\}} \sum\limits_{n\text{-}gram' \in C'} Count_{clip}(n\text{-}gram')}. \tag{4.4}$$

The result of equation 4.4 has to be combined for different n's. As proven in the original paper, the results of p_n decays roughly exponentially with n. To take this exponential decay into account, the BLEU score is calculated using the geometric mean, rather than the arithmetic mean [33]. The geometric mean is expressed in logscale in the paper,

$$exp\left(\sum_{n=1}^{N} w_n \log p_n\right). \tag{4.5}$$

where n is 4 and w_n is $\frac{1}{4}$ [33]. For $n = 4$ the same equation can be expressed as

$$\sqrt[4]{p_1 p_2 p_3 p_4}. \tag{4.6}$$

The last part of the BLEU score is the brevity penalty (BP). The purpose of BP is to penalize candidate sentences which are shorter than the reference sentence. Without the BP, the candidate sentence "the cat is black" will match 100% with the reference sentence "the cat is black and white". The BP is defined for two cases. If c, the length of the candidate sentence is longer than r, the length of the reference sentence, BP = 1. This is because candidate sentences longer than the reference sentences are already penalized by modified n-gram precision measure [33]. The brevity penalty is computed as

$$BP = \begin{cases} 1 & \text{if } c > \text{r} \\ e^{(1-r/c)} & \text{if } c \leq \text{r} \end{cases}. \tag{4.7}$$

The final BLEU score then becomes

$$BLEU = BP \times exp\left(\sum_{n=1}^{N} w_n \log p_n\right). \tag{4.8}$$

The output of the BLEU score will range between 0 and 1. In order to receive a score of 1, the candidate sentence has to be identical to a reference sentence. In practice, few translations will receive a score of 1. Even human translations will vary a lot from the reference sentences, and will therefore rarely receive a score of 1. One thing to keep in mind is that the BLEU score will increase with the number of reference sentences [33].

The BLEU score is often given as percentage rather than decimal. In general, BLEU scores higher than 30 reflect understandable translations, while scores higher than 50 reflect good and fluent translations. [1]. Figure 4.7 gives an indication of how good a translation is based on its BLEU score.

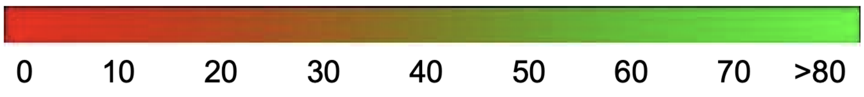

Figure 4.7: General interpretability of BLEU score [1].

4.5.2 Word Error Rate

When BLEU score is appropriate for assessing text-to-text translation systems, where the reference sentences are subjects to interpretation, the same does not apply when assessing speech-to-text applications, or sign-to-gloss. Word Error Rate (WER) is the metric that is typically used for these purposes [13].

WER compares the reference sentences to candidate sentences word by word, figuring out the number of differences between them. A difference can be one of three things. Firstly, it can be words that are present in the reference sentence, but missing from the candidate sentence, known as "deletions". Secondly, it can be words that are present in the candidate, but not in the reference sentence, known as "insertions". Thirdly, it can be altered words between the candidate and reference sentence [13].

Putting it all together, the formula for WER is the total number of changes divided by the total number of words in the reference sentence.

$$WER = \frac{Insertions + Deletions + Substitutions}{Total\ words\ in\ reference\ sentence}. \tag{4.9}$$

Word error rate is used for assessing the recognition part, the sign-to-gloss, in the model [8].

4.6 Evolved Transformer

Due to recent advances in the field of neural architecture search, So et al. [39] has come up with an improved transformer architecture, namely the Evolved Transformer (ET). They applied tournament selection architecture search on the Transformer architecture, trying to evolve it into a better and more efficient model. The search process ran through 15,000 different models on 270 TPUs (Tensor Processing Units) for almost 1 billion steps. In the interest of saving space, neural architecture search and tournament selection architecture will not be covered here.

The ET shows improved performance over the Transformer at all sizes, but especially on small, mobile friendly models with less than 7M parameters. In some cases on small sized models, the ET showed same performance as the Transformer with 37.6% less parameters [39].

There are mainly four notable differences between the original transformer and the ET: depth-wise separable convolutions, the Gated Linear Unit (GLU) layer, the use of the Swish activation function, and the branching structure. In the following subsections I will describe the separable convolutions, the GLU and the Swish function.

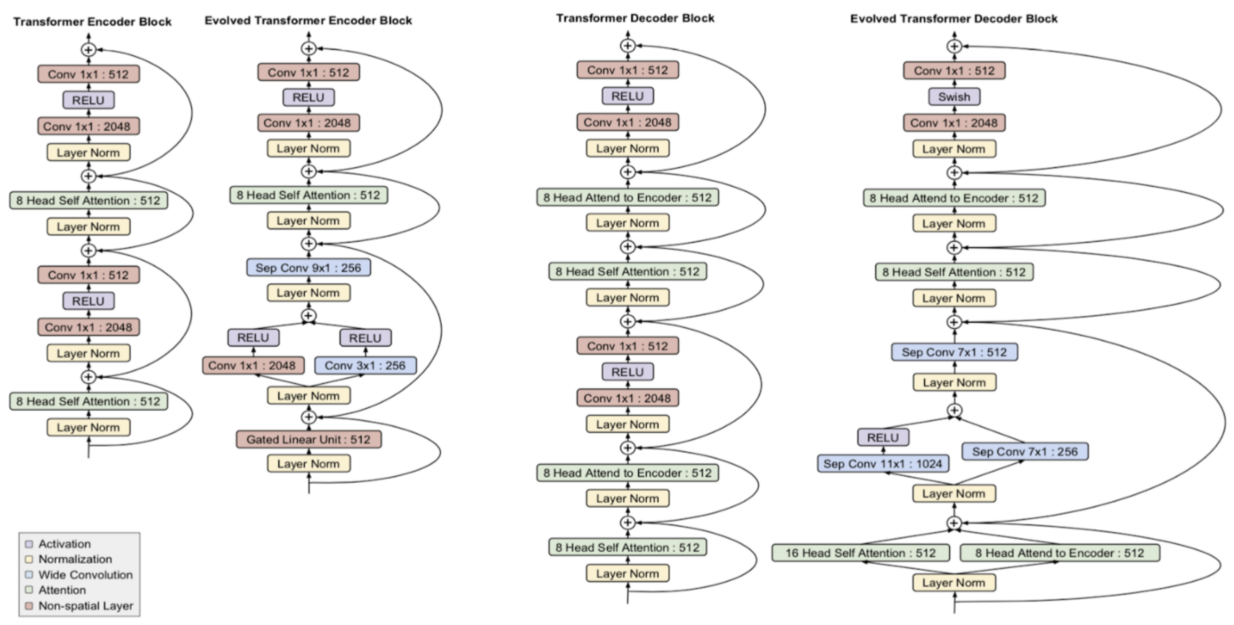

Figure 4.8: A comparison between the original Transformer architecture and the Evolved Transformer architecture [39].

4.6.1 Depth-Wise Separable Convolutions

The depth-wise separable convolutions aim to make a convolutional layer less computational costly by splitting the filter up into two smaller filters [22]. These two filters are then used in two sequential convolutional layers. Doing this results in two convolutional operations instead of one. Following are the steps needed in order to split up a filter of size $5x5x3x256$ (256 filters of dimension $5x5x3$) into two smaller filters.

The first part is the depth-wise convolution. Assuming we have an RGB image with 3 channels, the depth-wise convolution applies three different convolutions to the image in order to retain the original depth of the image. Unlike a normal convolutional layer, where all three convolutions have the same depth as the image, convolutions in a depth-wise convolutional layer only has one dimension. Each filter is applied to different depth-dimension of the image. By stacking the resulting matrices together we get an output-matrix of same depth as the input image [22]. Figure 4.9 illustrates how this works.

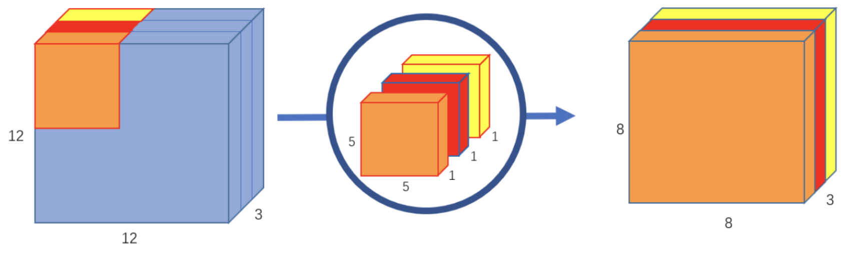

Figure 4.9: An image of dimension $12x12x3$ convolved with depth-wise convolutions of dimensions $5x5x1$, yielding an output matrix of dimension $8x8x3$ [22].

The second part is the point-wise convolution. The goal of the point-wise convolution is to change the depth of the image. It works exactly like normal convolutions, but its filter size is just $1x1$. Using the same example as above, we now have an output matrix of dimensions

$8x8x3$. Let's say we want to increase the depth to 256. We achieve this by using 256 filters of size $1x1x3$. The final output of the depth-wise and the point-wise convolutions then becomes $8x8x256$ [22].

4.6.2 Gated Linear Unit

The Gated Linear Unit (GLU) layer were introduced by Dauphin et al. [11] in 2017 and is expressed as

$$h_l(X) = (X * W + b) \otimes \sigma(X * V + c) \tag{4.10}$$

where X is the input sequence, W and V are convolutional filters (note that the symbol $*$ is the convolution operator, not the multiplication operator) and b and V are biases. σ is the Sigmoid-function, and $\otimes$ is the element-wise multiplication operator.

4.6.3 Swish Activation Function

The Swish activation function was found by Ramachandran et al. in 2017 [35], and is simply just

$$x \cdot \sigma(\beta x) \tag{4.11}$$

where σ is the Sigmoid-function, and β is either a constant, or a learnable parameter. One will often see Swish implemented as just

$$x \cdot \sigma(x).$$

This is because the most common use of the Swish function is with $\beta = 1$, making the equation similar to the Sigmoid-weighted Linear Unit (Sil) [16].

4.7 Multimodal Machine Learning

Humans are able to understand and experience the world with many different senses - we see objects, hear sounds, feel texture, smell odors etc. Furthermore, we are able to process all this information simultaneously, aligning the information perfectly together to one compound impression. The different ways in which something happens or is experienced is referred to as different *modalities*. Many machine learning problems only deal with one modality, for instance text-to-text problems. In order for machines to handle real world problems, it must be able to handle multiple modalities. The goal of multimodal machine learning (MMML) is to build models which can process and relate information from multiple modalities [2].

One of the main challenges in MMML is how to align the different modalities together. Going back to the human analogy, one can say that if a person sitting next to you claps his hands, the two modalities sound and vision are aligned if the sound is experienced at the same time as the clap. Contradictory, if the person is sitting on the other side of a soccer pitch, clapping his hands, the sound will appear later than the clap. One can say that the two modalities are not aligned properly in this case. Multimodal alignment is defined as finding relationships and correspondences between sub-components of instances from two or more modalities [2].

Sign language processing is a multimodal machine learning task as it deals with video and text sequences.

Chapter 5

Implementation

The paper I have been using as inspiration for my work is the paper "Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation" by Camgoz et al. [8]. The paper comes with a code base, called SignJoey [9], which I have been working with. The SignJoey project is highly based on the JoeyNMT project, but with modifications to realize joint continuous sign language recognition and translation. In this chapter I will cover the JoeyNMT project, the SignJoey project, and lastly, my own contribution to the project, namely the Evolved Transformer.

5.1 JoeyNMT

JoeyNMT is a neural machine translation (NMT) toolkit designed specifically for novices. The code aims to be clean, well documented and minimalist. Their approach is to identify the core features of NMT that have not changed over the last years, and to provide well documented, simple quality code. The project contains implementations of some standard network architectures, such as RNN, transformer, different attention mechanisms, input feeding, and configurable encoder/decoder bridge. It also contains standard learning techniques, such as dropout, learning rate scheduling, weight tying and early stopping criteria, and it contains tools for visualization and monitoring [29].

5.2 SignJoey

The SignJoey project is based off of the JoeyNMT project. Most of the code is the same, except some modification in order to realize joint continuous sign language recognition and translation [9]. The model is trained and evaluated on the PHOENIX14**T** corpus. In this section I will give an overview of the architecture of the "Sign Language Transformer".

The SignJoey project contains the implementation of the Sign Language Transformer (SLT) used by Camgoz et al. [8]. The SLT jointly learns to recognize and translate video sequences of sign language into sign glosses and spoken language sentences. Given a sign video sequence $\mathcal{V} = (I_1, ..., I_T)$ with T frames, the goal of the transformer is to learn the conditional probabilities $p(\mathcal{G}|\mathcal{V})$ of generating a sign gloss sequence $\mathcal{G} = (g_1, ..., g_N)$ with N glosses, and $p(\mathcal{S}|\mathcal{V})$ of generating a spoken language sentence $\mathcal{S} = (w_1, ..., w_U)$ with U words [8].

Modelling the above probabilities is a challenging sequence-to-sequence task for several reasons. First of all, the number of source tokens (number of video frames) is much greater than the number of target tokens. Furthermore, there is no one-to-one mapping from sign language to spoken language. This is because sign languages and spoken languages have different vocabularies, different grammar and different word ordering [8].

Figure 5.1 shows an overview of the model architecture. The two main components are the Sign Language Recognition Transformer (SLRT) on the left side, and the Sign Language Translation Transformer on the right side.

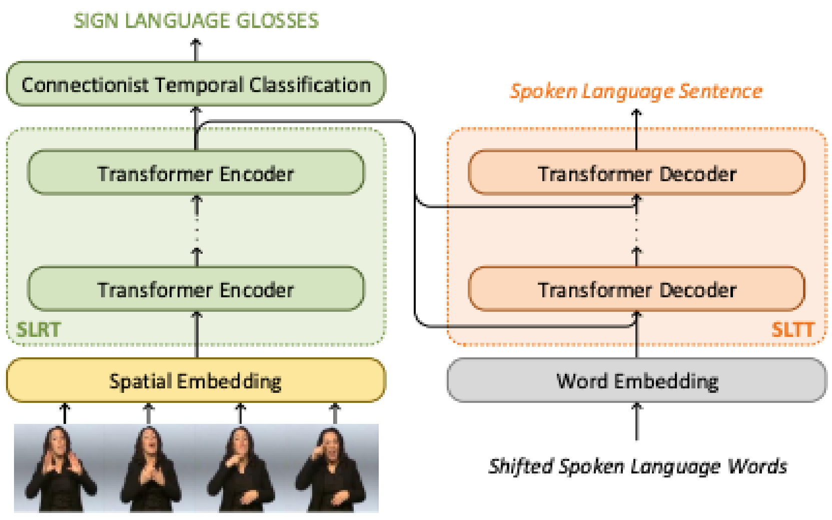

Figure 5.1: An overview of the end-to-end SLR and SLT transformer approach by Camgoz et al. [8].

5.2.1 Spatial and Word Embeddings

Both the spoken language words and the video frames are embedded before they are fed into the network. The spoken language words are embedded using a linear layer, initialized from scratch and learned during training. This is implemented in SignJoey using a PyTorch Embedding layer. The PyTorch Embedding class works as a lookup table, storing and updating embeddings of a fixed dictionary and a fixed dimension size.

The video frames, on the other hand, are embedded using the SpatialEmbedding approach from Camgoz et al. [7], which propagate each video frame through a CNN, trying to learn the non-linear frame level spatial representation [8].

Because transformer models take inputs in parallel, they have no positional information about the input. Without positional information, inputting a reversed video sequence would yield the same result as inputting the original video. To overcome this problem, positional encodings, as described in chapter 4.4.6, are added to both the word embeddings and the spatial embeddings. The final vectors are given as:

$$m_u = WordEmbedding(w_u) + PositionalEncoding(u)$$
$$f_t = SpatialEmbedding(I_t) + PositionalEncoding(t)$$

where w_u is the spoken language word at position u and I_t is the video frame at time t [8].

5.2.2 Sign Language Recognition Transformer

The Sign Language Recognition Transformer (SLRT) is shown on the left side of Figure 5.1. The goal of the SLRT is to recognize sign glosses from continuous sign language videos, and to learn meaningful spatio-temporal representations which can be forwarded to the Sign Language Translation Transformer. The SLRT takes in the positionally encoded spatial embeddings, and feeds it through a transformer encoder model [41]. The self-attention layer of the encoder learns the contextual relationship between the frames of a video. The output of the self-attention, together with a residual connection, is normalized, and passed through a non-linear point-wise feed forward layer [8].

Because signs have a spatio-temporal representation, sign glosses have a one-to-many relationship to the video frames. If we had a frame level annotated dataset, the SLRT could have been trained using a cross-entropy loss. Instead, they use a linear projection layer, followed by a softmax activation function, followed by a CTC, as described in chapter 4.2.2. We obtain $p(\mathcal{G}|\mathcal{V})$ from the CTC, and calculate the loss of the CSLR as

$$\mathcal{L}_{\mathcal{R}} = 1 - p(\mathcal{G}|\mathcal{V})$$

[8].

Th SLRT is configured in the encoders file in SignJoey, and implemented in the transformer_-layers file as "TransformerEncoderLayer". It is configured with 3 layers, an attention-head size of 8, input dimension size of 512 and a dropout of 0.1, to prevent overfitting. The point-wise feed forward layer is configured with hidden size of 2048 [8].

5.2.3 Sign Language Translation Transformer

On the right side of the figure we have the Sign Language Translation Transformer (SLTT). The SLTT is an autoregressive transformer decoder model, meaning it takes previous calculations as input when calculating the next output. The target sentence S is first prefixed with the special beginning of sentence token, $< bos >$, before the whole sentence is embedded and positionally encoded. During training, the embeddings are then passed to a masked self-attention layer, which ensures that each token only have information about its predecessors when extracting its contextual information [8].

The next attention block in the decoder learns the mapping between source and target sequences by combining the output of the encoder with the output of the masked attention layer in the decoder. The outputs from this attention block is then passed through a non-linear point-wise feed forward layer, just as in the SLRT. All these operations are also followed by normalization and residual connections, as in the SLRT [8]. This decoding process is formulated as:

$$h_{u+1} = SLTT(m_u | m_{1:u-1}, z_{1:T}), \tag{5.1}$$

where $z_{1:T}$ is the output sequence from the encoder, $m_{1:u-1}$ are the positional encoded word embeddings up to position u-1. h_{u+1} is the predicted next word in the sentence, which is fed into the next iteration of the SLTT [8].

The goal of the SLTT is to learn to generate one word at a time until it generates the special end of sentence token, $< eos >$. Its ultimate goal is to find the conditional probability $p(\mathcal{S}|\mathcal{V})$. It does so by decomposing it into ordered conditional probabilities

$$p(\mathcal{S}|\mathcal{V}) = \prod_{u=1}^{U} p(w_u|h_u). \tag{5.2}$$

These decomposed probabilities are then used to calculate the cross-entropy loss for each as:

$$\mathcal{L}_T = 1 - \prod_{u=1}^{U} \sum_{d=1}^{D} p(\hat{w}_u^d) p(w_u^d | h_u) \tag{5.3}$$

where $p(\hat{w}_u^d)$ is the probability of the ground truth word w^d at decoding step u. D is the vocabulary size of the target language. The network is then trained by minimizing a weighted joint loss of the recognition loss $\mathcal{L}_R$ and the translation loss $\mathcal{L}_T$ as:

$$\mathcal{L} = \lambda_R \mathcal{L}_R + \lambda_T \mathcal{L}_T \tag{5.4}$$

where λ_R and λ_T are hyper parameters, set in the configuration file, which decides how important each loss function is during training [8].

5.2.4 Training

The network is initialized using Xavier initialization [19], and an Adam optimizer [25] is used to train the network. The batch size is 32, and the learning rate is 10^{-3}. The network is evaluated every 100 iteration. For every 8th evaluation step without any decrease in development score (WER), the learning rate is decreased by a factor of 0.7, until it reaches below 10^{-6} [8].

5.2.5 Essential Classes and Methods

In this section I will briefly go over some of the main components of the SignJoey code base.

The *train* method in training.py is where most of the work is handled. It loads the config parameters, sets the random seed, loads the data, builds the model, sets up logging, trains the model, validates the model and tests the model.

The config-parameters are loaded from the sign.yaml file in the config-folder via $load\_config$. The *set_seed* method sets the random seed for PyTorch, NumPy and the random-library. Both $load\_config$ and *set_seed* are implemented in helpers.py.

The train, dev and test data are loaded from the data-folder together with the gloss and text vocabularies through *load_data* implemented in data.py.

The *build_model* method, implemented in model.py, initializes the spatial embedder, text embedder, transformer encoder and transformer decoder. All these are passed to a $SignModel$ object, which is initialized with a Xavier initializer in *initialize_model*, implemented in initialization.py.

$SignModel$ inherits from PyTorch's $nn.Module$, and holds the full Sign Language Transformer model, including the spatial embedding layer and word embedding layer.

The model is then passed to the $TrainManager$, which manages the training loop, validation, learning rate scheduling, early stopping and logging.

5.3 Contribution

Much of my time has gone into investigating the SignJoey codebase in order to understand the concepts, update libraries and run the code myself. The code base is large and complex, and it contains many unused classes and methods. In order to make the code base smaller and easier to understand I have removed much of the code which was not used, and done refactoring and debugging to gain understanding.

The configuration file contains many parameters from the JoeyNMT project which remain constant in the SignJoey project. In order to make the code more readable I have put many of these parameters directly into the code.

After gaining understanding of the code, and after running the original code with some different parameter variations, I tried to implement the Evolved Transformer by So et al. [39].

5.3.1 Evolved Transformer

There is an existing implementation of the evolved transformer (ET) in TensorFlow[1]. However, because almost all the code in the SignJoey project is running on PyTorch, it was convenient to create an ET implementation in PyTorch.

There were no public PyTorch-implementations of the full Evolved Transformer available. However, an ET encoder implementation marked as "work in progress" (WIP) was available at Github[2]. I used the available WIP encoder as inspiration for creating the decoder. Further investigation revealed that the reason for the WIP mark was because of the Separable Convolutions were not properly implemented. After reading the theory about Separable CNNs I made some modifications to the encoder, which yielded slightly better results. Additionally I added dropout layers with a dropout of 0.1 after both left and right branches, to avoid overfitting. This is also done in the original TensorFlow implementation.

Creating the decoder, however, was not as straight forward as it seemed when I first had the theory in place. After several attempts, comparing components to both the ET encoder, the original Transformer decoder and to the ET implementation in TensorFlow the ET decoder still produced really poor results. The result was to debug the decoder by removing parts of it until it performed better. By removing the three layers which use Separable CNNs, and their surrounding layers (normalization layers, activation function etc.), I received results close to the results of the original decoder. The findings were strange, as the same implementation of Separable CNNs were used successfully in the encoder. However, the final decoder architecture of my modified ET decoder is shown in Figure 5.2.

When comparing the Transformer to the Evolved Transformer (see Figure 4.8) we see that the Transformer repeats it selves in the comparison. This means that one layer of the ET encoder/decoder is equivalent to two layers of the Transformer encoder/decoder. Because the original implementation of the Sign Language Transformer uses 3 layers of encoder and decoder, it was not possible to divide the number of layers by 2. I have therefore used 2 layers for the ET encoder and decoder. This should be kept in mind when comparing the parameters in chapter 6.

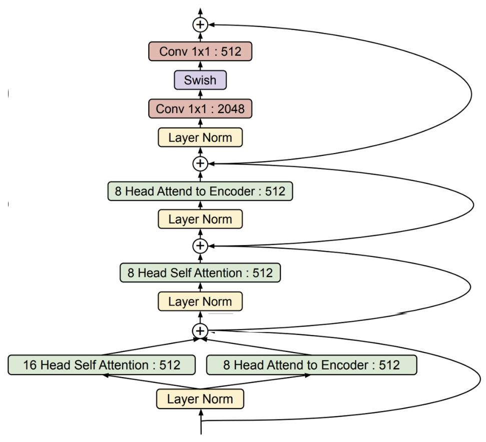
Conv 1x1 : 512
Swish
Conv 1x1 : 2048
Layer Norm
8 Head Attend to Encoder : 512
Layer Norm
8 Head Self Attention : 512
Layer Norm
16 Head Self Attention : 512
8 Head Attend to Encoder : 512
Layer Norm

Figure 5.2: The modified version of the Evolved Transformer, implemented in the SignJoey project.

[1] https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/evolved_transformer.py

[2] https://github.com/Shikhar-S/EvolvedTransformer

Chapter 6

Results

Table 6.1 shows the WER and the BLEU-4 scores that Camgoz et al. reported, compared to the my own results when running their implementation. We see that most of the best results are achieved when the recognition loss weight is 5.0. The same weight is therefore used when generating the results of table 6.2. It is interesting to see that my own results are worse than the results from Camgoz et al. The only time my own results are better is the dev results for recognition loss weight = 20. The dev results are validations done during training while tuning the model. The model is validated on a dev/validation dataset during training to test the performance along the way.

		Camgoz et al.				My own results			
Loss Weights		DEV		TEST		DEV		TEST	
λ_R	λ_T	WER	BLEU-4	WER	BLEU-4	WER	BLEU-4	WER	BLEU-4
1.0	1.0	35.13	21.73	33.75	21.22	49.53	20.08	48.84	20.13
2.5	1.0	26.99	22.11	27.55	21.37	29.76	**21.10**	29.37	20.73
5.0	1.0	**24.61**	22.12	**24.49**	**21.80**	**28.56**	20.69	**28.76**	**21.50**
10.0	1.0	24.98	**22.38**	26.16	21.32	32.13	19.36	33.32	19.66
20.0	1.0	25.87	20.90	25.73	20.93	29.76	**21.10**	29.37	20.73

Table 6.1: Comparison between Camgoz et al.'s results and my own results with the same model and parameters.

Table 6.2 shows a comparison of the BLEU-4 scores between three different models I have tested, with different embedding dimensions (d_{model}) and sizes of the feed-forward hidden layer (ff_{size}). Because the implementation of the Evolved Transformer is modified, I also tested the models running only the ET encoder. Surprisingly, it achieved better results when the dimensions were reduced.

The modified ET also achieved better results with lower dimensions in some cases. For $d_{model} = 256$ it achieved almost as good results as the best results generated with the original implementation.

Dimensions		Transformer		ET encoder only		Modified ET	
d_{model}	ff_{size}	DEV	TEST	DEV	TEST	DEV	TEST
64	256	19.65	19.59	**21.19**	**21.30**	19.39	19.06
128	512	20.97	20.31	**21.60**	**21.02**	20.88	20.18
256	1024	**20.71**	21.14	20.65	21.40	20.42	**21.46**
512	2048	**20.69**	**21.50**	20.67	20.71	20.26	19.85

Table 6.2: Comparison of results with different network dimensions. The bold results mark the best dev and test results for each dimensions.

The next table, Table 6.3 shows a comparison of the number of parameters in each model. Because one layer of ET corresponds to two layers of the Transformer the comparison would have been better if the Transformer had twice as many layers as the ET. However, in order to make the results comparable to the results from Camgoz et al. I kept 3 layers in the original architecture. The most interesting observations here is that the "ET encoder only" model achieves better results with lower dimensions. This partially proves the point of the Evolved Transformer, that it works better with less parameters than the Transformer. Partially, because I am only using the encoder.

d_{model}	ff_{size}	Transformer	ET encoder only	Modified ET
64	256	856,767	937,279	947,199
128	512	2,400,575	2,709,055	2,745,279
256	1024	7,552,575	8,759,359	8,897,343
512	2048	26,114,111	30,886,975	32,233,535

Table 6.3: Comparison of the number of parameters in the different models. The "ET encoder only" model runs with 2 layers of the encoder and 3 layers of the decoder, the Transformer runs with 3 layers in both, and the "Modified ET" runs with 2 layers in both.

The last two tables show the generated sentences and the corresponding generated gloss sequences by the "ET encoder only" model. Some interesting observations here is that it translates "dear viewers, good evening" perfectly. There are probably two reasons for this. The first being that the sentence is short, and the second one being that this is probably a much repeated line in the dataset, as this is probably the intro of many of the weather forecasts.

Also sentence 1 is translated correctly. This one is more surprising as it is a much longer sentence. However, looking at sentence 6, it seems like the same phrase is used in many of the recordings, but with different dates at the end.

Another interesting observation is the translation of sentence number 6, where it generates correctly until the last word which is the month. My best guess is that the signs for June and July are as similar as they are in spoken languages. From the generated glosses we see that the same error is also present there, meaning the error lies in the encoder.

Comparing the sign gloss sequence of entry number 4 and 5 to their corresponding sentences show that the sentences vary a lot more than the sign gloss sequences. This might be a hint of a weakly trained decoder, which is not able to translate from sign glosses to full sentences.

The English translations of the German text are acquired from Google Translate.

1	Reference	und nun die wettervorhersage für morgen mittwoch den zwölften mai
		(and now the weather forecast for tomorrow, wednesday, may twelfth)
	Generated	und nun die wettervorhersage für morgen mittwoch den zwölften mai
		(and now the weather forecast for tomorrow, wednesday, may twelfth)
2	Reference	später im westen wieder schauer und gewitter
		(later in the west again showers and thunderstorms)
	Generated	im westen und südwesten einige schauer und gewitter
		(some showers and thunderstorms in the west and southwest)
3	Reference	liebe zuschauer guten abend
		(dear viewers, good evening)
	Generated	liebe zuschauer guten abend
		(dear viewers, good evening)
4	Reference	der wind weht morgen schwach bis mäßig aus westlichen richtungen
		(the wind will blow weak to moderate from westerly directions tomorrow)
	Generated	der wind weht schwach bis mäßig aus west bis nordwest
		(the wind will be weak to moderate from west to northwest)
5	Reference	auch am donnerstag neben sonne gebietsweise heftige schauer oder gewitter
		(also on thursday, in addition to the sun, heavy showers or thunderstorms in some areas)
	Generated	am donnerstag teilweise kräftige regenfälle teilweise kräftige gewitter
		(on thursday partly heavy rainfall partly heavy thunderstorms)
6	Reference	und nun die wettervorhersage für morgen freitag den zehnten juli
		(and now the weather forecast for tomorrow, Friday, July 10th)
	Generated	und nun die wettervorhersage für morgen freitag den zehnten juni
		(and now the weather forecast for tomorrow, Friday, June 10th)

Table 6.4: Translations generated by the "ET encoder only" model, with $d_{model} = 128$ and $ff_{size} = 512$.

1	Reference	MORGEN WETTER WIE-AUSSEHEN MITTWOCH ZWOELF MAI
	Generated	MORGEN WETTER WIE-AUSSEHEN MITTWOCH ZWOELF MAI
2	Reference	WEST MEHR SCHAUER GEWITTER
	Generated	WEST MEHR REGEN GEWITTER
3	Reference	LIEB ZUSCHAUER GUT ABEND
	Generated	LIEB ZUSCHAUER GUT ABEND
4	Reference	WIND SCHWACH MAESSIG IX WEHEN
	Generated	WIND SCHWACH MAESSIG WEHEN
5	Reference	DONNERSTAG SONNE TEILWEISE REGEN STARK GEWITTER
	Generated	DONNERSTAG SONNE DANN TEILWEISE REGEN STARK GEWITTER
6	Reference	MORGEN WETTER WIE-AUSSEHEN FREITAG ZEHNTE JULI
	Generated	MORGEN WETTER WIE-AUSSEHEN FREITAG ZEHNTE JUNI

Table 6.5: Corresponding sign gloss sequences generated by the Sign Language Recognition Transformer (the encoder).

Chapter 7

Discussions

Even though there has been done a lot of good research on both Sign Language Recognition and Sign Language Translation, there are still many tasks that have to be solved before the research can be applied in real applications. In this chapter I will first discuss what I think are some of the biggest remaining challenges in the field, based on what I have learnt. Lastly, I will evaluate the architectures used.

7.1 Deaf Involvement

When working in the field of sign language translation it is important to keep in mind what the end goal of the research is. The technology is supposed to make the lives of the Deaf easier. We want to make usable systems that match the user needs of the deaf people. Most teams working on SLT are all-hearing teams, who lack the experience of being deaf. Accordingly, they lack the knowledge of use cases and contexts where sign language software must function [5]. As far as I know, this also applies to Camgoz et al.

Bragg et al. [5] state that Deaf community involvement is essential at all levels. However, because SLT is still at an early stage, I think it is not very important to have deaf members in every team. There is a lot of good literature available which describe sign language and the different aspects which have to be kept in mind when doing research. Requiring deaf members of all teams would probably limit the amount of research in the field. As SLT progresses and gets closer to an end product it is much more important to involve the Deaf, in order to make useful end products which fit their needs.

As briefly mentioned in Chapter 2, it is also important to respect the Deaf community, and their ownership to sign languages. Forcing the technology can lead to rejection from the Deaf community. There are examples of systems which have failed because of this [17]. It is therefore important to keep the Deaf community involved in the research, without needing deaf members in all teams.

7.2 Dataset

The main challenge in neural sign language translation is the lack of good available datasets. Even though Camgoz et al. achieves decent results on the PHOENIX14**T** dataset, there is still a long way to the end goal of achieving universal sign language translation. Examining the dataset against what was covered in Chapter 2.3, there are several weaknesses with PHOENIX14**T**.

First of all, the corpora is small in size compared to other machine learning datasets. The dataset contains video recordings of around 67,000 signs from a sign vocabulary of 1066 different signs distributed over around 8250 video sequences [7]. This is considered a relatively small dataset in the world of deep learning.

Secondly, the dataset lacks the variety needed in good sign language datasets. The recordings are performed by 9 different sign language interpreters [7], meaning it lacks native speakers. Additionally, there is no variation in clothing, ass all the signers wear black clothes, and there is no variation in camera quality and lightning conditions, as the recordings are performed in a studio.

Lastly, the videos are recordings of weather forecasts, meaning there is a limited domain. It is probably easier to learn a language model for a limited domain than to learn a general model for all domains. It is therefore reasonable to think that the same model would have performed worse if trained on larger datasets with no specific domain.

7.3 Annotations

No standard annotation methods exist for sign languages, which makes the production of sign language corpora hard and time consuming. In order to speed up the research in the field, a standard annotation method would have been a major contribution. Because of the many differences between spoken languages and sign languages, which make a mapping from sign language to spoken language hard, a good temporary solution would be to use gloss annotations. As long as all signs have only one gloss assigned to it, this approach could at least work well for SLR. However, in order to do full SLT we also need full sentence annotations, or a mapping from glosses to sentences.

7.4 Measurements

BLEU score and WER are two common measurements to use when measuring the performance of a neural machine translation model, and they are both used by Camgoz et al to measure gloss accuracy and spoken language translation accuracy. One problem with both of them is that they do not evaluate the meaning of the words. For a ground truth sentence "this is great", sentences like "this is good" and "this is horrible" would both yield the same BLEU-score and WER, even though the first prediction obviously is a better translation than the second one. I will give a suggestion to how this can be done in Chapter 8.1.

7.5 Sign Language Transformer

The Sign Language Transformer architecture seems like a good choice for doing SLT with the PHOENIX14**T** dataset. The key here is that the dataset contains both gloss annotations and full sentence annotations, making it natural to think of glosses as an intermediate representation. In practice, if the encoder was fully capable of recognizing sign glosses, the remaining problem would only be to map gloss sequences to sentences. This is just a text to text translation task, which is a much bigger research area. In fact, autoencoders [3] are designed for similar use cases, and could have been of interest in that case. For the purpose of this work, however, autoencoders are not of interest, and will not be described any further.

7.6 Evolved Transformer

Thinking about possible realistic sign language translation applications it is naturally thinking in the direction of mobile devices. Using efficient models with less parameters is therefore a big key for success in developing user friendly, useful applications. The Evolved Transformer was therefore an interesting model to test with the existing state-of-the-art architecture, as it has reported high performance on mobile-friendly models with fewer parameters than the Transformer. As mentioned in Chapter 4.6, the ET achieved same performance as the Transformer in some cases having 37.6% less parameters. As far as I am aware of, the ET has not been implemented in the SignJoey project before.

As implementing the ET decoder was more challenging than expected, the final decoder I ended up using was a modified version of the original one. I assume there is an error in my implementation. However, all the modules have been tested in isolation, and seem to be working. It is therefore a small possibility that the original decoder is not suitable for this task, but as the encoder works as expected, I do not see why the decoder should not work.

Looking at table 6.2 we see that especially the "ET encoder only" implementation achieves better results than the original Transformer when the dimensions and the total number of parameters decrease. As So et al. report cases where the ET performs as good as the Transformer with 37.6% less parameters, I find it interesting that my "ET encoder only" implementation achieves better results with 87.5% less parameters, compared to the Transformer implementation with $d_{model} = 256$. I have no good explanation for why this happens.

Chapter 8

Conclusions and Future Work

The purpose of this work was to gain insight into the field of neural sign language recognition and translation, understand and test the SignJoey code base by Camgoz et al., and to try to improve the performance by implementing the Evolved Transformer.

I have learnt a lot about sign languages and its complexity, and why it is a challenging neural sign language translation task. By studying the work by Camgoz et al. [8] and working with their code base SignJoey I have learnt about many new concepts in seq2seq neural networks, such as the Transformer architecture and all its components, the CTC loss function and BLEU score and WER metrics.

On my attempt to implement the Evolved Transformer I faced challenges in the decoder, which I did not manage to solve successfully. I therefore applied a modified version of the Evolved Transformer to the SignJoey project, and successfully ran experiments showing that it achieved decent results with few parameters. Experiments run with only the ET encoder achieved better results than the original Transformer for smaller dimensions. At the most it achieved a BLEU-4 score increase of more than 1.5 BLEU-4. It also partially proves that the Evolved Transformer is more efficient than the Transformer with lower dimensions.

8.1 Future Work

In Chapter 7 I mentioned that the evaluation metrics do not capture the meaning of the words. One solution to this could be to do sentiment analysis on the generated sentences and the source sentences, and compare the outcomes of these. If generated sentences are similar to the source sentences, but using different words, having a sentiment analysis can in many cases tell us if these words are somehow similar in meaning. The paper "COVID-19 sentiment analysis via deep learning during the rise of novel cases" by Chandra et al. shows how this can be done [10].

As I assume that an error in my implementation was the reason why my Evolved Transformer decoder did not work very well, a new attempt on trying to implement the full ET would also be interesting to do in the future.

For Sign Language Translation in general, the most important work in the near future is curating datasets. The bottleneck of SLT lies in the lack of good enough datasets. The Deaf community has to be more involved especially in curating dataset, but also in the research field in general.

Bibliography

[1] Alon Lavie. *Evaluating the Output of Machine Translation Systems*. AMTA. Sept. 19, 2011. URL: https://www.cs.cmu.edu/~alavie/Presentations/MT-Evaluation-MT-Summit-Tutorial-19Sep11.pdf.

[2] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. *Multimodal Machine Learning: A Survey and Taxonomy*. 2017. DOI: 10.48550/ARXIV.1705.09406. URL: https://arxiv.org/abs/1705.09406.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2020. DOI: 10.48550/ARXIV.2003.05991. URL: https://arxiv.org/abs/2003.05991.

[4] Peter Bloem. *Transformers From Scratch*. 2019. URL: http://peterbloem.nl/blog/transformers.

[5] Danielle Bragg et al. *Sign Language Recognition, Generation, and Translation: An Interdisciplinary Perspective*. 2019. DOI: 10.48550/ARXIV.1908.08597. URL: https://arxiv.org/abs/1908.08597.

[6] Jason Brownlee. *What Are Word Embeddings for Text?* Oct. 2017. URL: https://machinelearningmaster com/what-are-word-embeddings/.

[7] Necati Cihan Camgoz et al. "Neural Sign Language Translation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[8] Necati Cihan Camgoz et al. *Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation*. 2020. DOI: 10.48550/ARXIV.2003.13830. URL: https://arxiv.org/abs/2003.13830.

[9] Necati Cihan Camgoz et al. "Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[10] Rohitash Chandra and Aswin Krishna. "COVID-19 sentiment analysis via deep learning during the rise of novel cases." In: *PLOS ONE* 16.8 (Aug. 2021). Ed. by Liviu-Adrian Cotfas, e0255615. DOI: 10.1371/journal.pone.0255615. URL: https://doi.org/10.1371%5C%2Fjournal.pone.0255615.

[11] Yann N. Dauphin et al. *Language Modeling with Gated Convolutional Networks*. 2016. DOI: 10.48550/ARXIV.1612.08083. URL: https://arxiv.org/abs/1612.08083.

[12] De Coster, Mathieu and D'Oosterlinck, Karel and Pizurica, Marija and Rabaey, Paloma and Verlinden, Severine and Van Herreweghe, Mieke and Dambre, Joni. "Frozen pretrained transformers for neural sign language translation." eng. In: *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*. Online: Association for Machine Translation in the Americas, 2021, 88–97. URL: %7Bhttps://aclanthology.org/2021.mtsummit-at4ssl.10.pdf%7D.

[13] Ketan Doshi. *Foundations of NLP Explained — Bleu Score and WER Metrics*. May 2021. URL: https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b.

[14] Ketan Doshi. *Foundations of NLP Explained Visually: Beam Search, How It Works*. Apr. 2021. URL: https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24.

[15] IBM Cloud Education. *Convolutional Neural Networks.* Oct. 2020. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[16] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning.* 2017. DOI: 10.48550/ARXIV.1702.03118. URL: https://arxiv.org/abs/1702.03118.

[17] Michael Erard. *Why Sign-Language Gloves Don't Help Deaf People.* Nov. 2017. URL: https://www.theatlantic.com/technology/archive/2017/11/why-sign-language-gloves-dont-help-deaf-people/545441/.

[18] Funksjonshemmedes Fellesorganisasjon. *Norges Døveforbund.* URL: https://ffo.no/Medlemsorganisasjo N/Norges-Doveforbund/.

[19] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.* Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[20] Alex Graves et al. "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks." In: vol. 2006. Jan. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[21] Awni Hannun. "Sequence Modeling with CTC." In: *Distill* (2017). https://distill.pub/2017/ctc. DOI: 10.23915/distill.00008.

[22] Chi-Feng Wang Karani. *A Basic Introduction to Separable Convolutions.* Aug. 2018. URL: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728.

[23] Dhruvil Karani. *Introduction to Word Embedding and Word2Vec.* Sept. 2018. URL: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa.

[24] Amirhossein Kazemnejad. "Transformer Architecture: The Positional Encoding." In: *kazemnejad.com* (2019). URL: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[25] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[26] Sang-Ki Ko et al. *Neural Sign Language Translation based on Human Keypoint Estimation.* 2018. DOI: 10.48550/ARXIV.1811.11436. URL: https://arxiv.org/abs/1811.11436.

[27] Kiprono Elijah Koech. *Cross-Entropy Loss Function.* Oct. 2020. URL: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[28] Oscar Koller, Jens Forster, and Hermann Ney. "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers." In: *Computer Vision and Image Understanding* 141 (2015). Pose & Gesture, pp. 108–125. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2015.09.013. URL: https://www.sciencedirect.com/science/article/pii/S1077314215002088.

[29] Julia Kreutzer, Jasmijn Bastings, and Stefan Riezler. "Joey NMT: A Minimalist NMT Toolkit for Novices." In: (2019). DOI: 10.48550/ARXIV.1907.12484. URL: https://arxiv.org/abs/1907.12484.

[30] Rick Mangan. *Signs & their Glosses.* 2002. URL: https://www2.bellevuecollege.edu/artshum/materials/lang/Mangan/Winter2006/103/GlossExplained.pdf.

[31] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[32] Joseph Murray. "Celebrating thriving deaf communities to sign for Human Rights." In: *UNtoday* (Sept. 1, 2021). URL: https://untoday.org/celebrating-thriving-deaf-communities-to-sign-for-human-rights/ (visited on 05/06/2022).

[33] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

[34] Christophe Pere. *What are Loss Functions?* June 2020. URL: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904.

[35] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. DOI: 10.48550/ARXIV.1710.05941. URL: https://arxiv.org/abs/1710.05941.

[36] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. "Sign Language Recognition: A Deep Survey." In: *Expert Systems with Applications* 164 (2021), p. 113794. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113794. URL: https://www.sciencedirect.com/science/article/pii/S095741742030614X.

[37] Harald Scheidl. *An Intuitive Explanation of Connectionist Temporal Classification*. June 2018. URL: https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c.

[38] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. DOI: 10.48550/ARXIV.1912.05911. URL: https://arxiv.org/abs/1912.05911.

[39] David R. So, Chen Liang, and Quoc V. Le. *The Evolved Transformer*. 2019. DOI: 10.48550/ARXIV.1901.11117. URL: https://arxiv.org/abs/1901.11117.

[40] Ekin Tiu. *Understanding Latent Space in Machine Learning*. Feb. 2020. URL: https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d.

[41] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[42] Wanshun Wong. *What is Residual Connection?* Dec. 2021. URL: https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55.

[43] Aston Zhang et al. "Dive into Deep Learning." In: *arXiv preprint arXiv:2106.11342* (2021).

[44] Thomas Zimmerman et al. "A hand gesture interface device." In: vol. 17. May 1986, pp. 189–192. DOI: 10.1145/30851.275628.