Μετάφραση βίντεο σε νοηματική γλώσσα

Video translation to sign language

Antonios Alexos

Department of Electrical and Computer Engineering

University of Thessaly

Supervisor

Gerasimos Potamianos

Co-Supervisors

Dimitrios Tsaopoulos, George Stamoulis

In partial fulfillment of the requirements for the degree of

Diploma in Engineering and Computer Engineering

August 6, 2020

Acknowledgements

First of all, I would like to thank my thesis supervisor, Dr. Gerasimos Potamianos for his insight and guidance through my journey; as well as for motivating me and pushing me to my limits for the completion of this project.

Secondly, I would like to thank Katerina Papadimitriou for her insight and help in my thoughts and ideas. Without her I would have lost a lot of precious time, drifting on some ideas that would not help in the implementation of the project. I would also like to thank Dr. Panos Toulis, for his insight and guidance.

Moreover, I would like to thank my parents and my friends for believing in my and supporting me.

Last but not Least, I would like to thank myself for all this hard work and for not giving up in the darkest of times. And for always trying to push myself further into achieving the impossible.

This Thesis is dedicated to me.

Περίληψη

Σε αυτή τη διπλωματική, αντιμετωπίζουμε ένα πρόβλημα που σχετίζεται τόσο με το Όραση Υπολογιστών όσο και με την επεξεργασία φυσικής γλώσσας. Πιο συγκεκριμένα προτείνουμε διάφορες μεθόδους για τη μετάφραση της νοηματικής γλώσσας από βίντεο σε κείμενο, όπως "Encoder"-"Decoder" με μοντέλα προσοχής, το "Transformer" κ.λπ. Δοκιμάζουμε διαφορετικές παραλλαγές τους, τόσο στο πλαίσιο όσο και στην προεπεξεργασία των δεδομένων που εισέρχονται στα μοντέλα ως είσοδος.

Abstract

In this thesis, we consider a problem that has to do both with Computer Vision and Natural Language Processing. More specifically we propose various methods to translate sign language from video to text, like Encoder-Decoder with attention models, the Transformer, etc. We try different variations of them, both in the context and in the preprocessing of the data that goes into the model as an input.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Sign Language is used by deaf people as the main means of communication in their daily lives, and at the same time, it is very difficult for non-sign language speakers to understand. Deaf people encounter many obstacles in conducting a flawless and natural conversation, because they are not able to hear sounds. It is not uncommon in some social environments, that the hearing-impaired people will need help from interpreters, in order to communicate with other people. These people are also vulnerable in emergency situations which may cause health hazards. These aforementioned situations are some good examples, that may lead the hearing-impaired people into isolation and feel unwanted by the society.

Sign Language translation could provide a good solution to the problems that the hearing-impaired people are encountering in their daily lives. This kind of solution could be embedded in a lot of technologies, in order to achieve a harmonious and natural communication between hearing-impaired people and hearing-abled people. With this technologies we could help the hearing-impaired people to get integrated into the society, and most importantly to feel a part of

it.

Sign Language translation though, is a very difficult problem with a lot of challenges, because it combines both visuals and linguistics. The possible solutions are also limited by the technologies of our time. Maybe in the future new Computer Vision and Deep Learning algorithms will emerge, and help us tackle the problem more accurately and precisely.

In this work we encounter this problem by examining different approaches of translating video data to text, and providing eventually, a solution to sign language translation. Our approach includes Encoder-Decoder models with neural attention [10] and the Transformer model [7]. We present our approach in detail in the following sections.

1.2 Related Work

The problem is not new, and therefore there have been some approaches to Sign Language Translation. In [11] the authors applied the Encoder-Decoder with attention model in order to translate sign language from German weather forecasts. This sequence-to-sequence model translates spatiotemporal representation of signs into text. Another approach to sign language translation is [12], where the authors also used the attention based Encoder-Decoder model but they relied on human keypoint estimation. They implemented their model in the first Korean sign language dataset, KETI.

The other approaches in the bibliography are not directly associated with translation, but with similar tasks like sign language recognition, which are easier to solve than translation, because of their algorithms' complexities. Many of these approaches recognize individual letters of the alphabet with a single hand. This is a very simple task. In [13] the authors used a Microsoft Kinect, a gaming camera

for XBox, and achieved an accuracy of 92% with Random Forest. In [14], the authors performed two deep learning based pose estimation methods and transfer learning, for recognizing the American Sign Language (ASL). In [15], the authors added some new features based on the hand region, like distance and angle, in order to recognize Korean Sign Language. In [16] the authors used the Hidden Markov model (HMM) in order to recognize whole sentences based on ASL. They approached the problem differently by using gloves for the hands. They managed to achieve an accuracy of 99.2% with the gloves, and 84.7% without the gloves.

Moreover, there have been some interesting approaches for sign language recognition from television. In [17], the authors experimented with British Sign Language (BSL) in TV broadcasts by using simultaneously subtitle information. In [18], the authors introduced an unsupervised method to recognise signs from subtitles, with the help of some hand and head tracking on the videos. Also, in [19] the authors approached the problem with videos with subtitles from BBC TV broadcasts.

Seeing the new trend, which is recognizing sign language from television data, the authors in [20] created a large database on sign language in TV Weather broadcasts. This famous database is called RWTH-PHOENIX-Weather 2012, and after 2 years, they published a second version of it with much more data and details in [21], called RWTH-PHOENIX-Weather 2014. We have to note, that this database is very user-friendly and has a lot of useful details and features like marked glosses, time boundaries, different signers, hand positions and face expressions. The approach in [22] uses these data in a new statistical method in order to achieve continuous sign language recognition.

So far we have only mentioned work based mostly in more traditional Machine Learning methods. Now we are going to introduce some Deep Learning based methods. In [23] the authors approached sign language recognition by extracting

the signer from the video and constructing a feature vector from him and some active contours. The recognition achieved with the implementation of an Artificial Neural Network an accuracy of 93%. In [24] the authors focus more on gesture recognition, activity recognition and continuous sign language recognition. They proposed a deep recurrent CNN-BLSTM network embedded into an HMM, which corrects the weak labels of the data and manages to achieve continuous sign language recognition on the RWTH-PHOENIX-Weather 2014 dataset. In [25] the authors evaluate different Convolutional Neural Networks (CNN) architectures to predict 3D joint locations of a hand from a depth map. Another interesting classification approach to the sign language recognition problem is the one in [26]. The authors in this work proposed a classification approach with the use of bidirectional GRUs.

More recent works leverage the Transformer model in order to tackle the sign language translation problem. The authors in [27] applied the Transformer model on the RWTH-PHOENIX-Weather 2014 dataset. On the other hand, the author in [9] applied different ensembles of Transformers, on both the RWTH-PHOENIX-Weather 2014 dataset and the American Sign Language Dataset, where he achieved state-of-the-art results.

1.3 Organization of the Thesis

In this section we present the structure of the remainder of this thesis:

1. Chapter 2 provides a short introduction in Deep Learning and in the main architectures that we use. We give details about the main components of our architectures.

2. Chapter 3 gives an overview of our approach, and explains our model architecture.

3. Chapter 4 presents the Data and the Experimental Framework that we implement.

4. Chapter 5 shows the results that were drawn from our experiments, and a brief discussion on them.

5. Chapter 6 provides the conclusions of this thesis and a summary.

Chapter 2

Introduction to Deep Learning

Summary

In this Chapter we give a brief introduction to Deep Learning, and some basic models of it. The models that we present are the Artificial Neural Network, the Recurrent Neural Network, and the Neural Attention.

2.1 Deep Learning

Deep Learning is a subset of Machine Learning that is based on Artificial Neural Networks, an architecture that was inspired by the human brain, and learns from large amounts of data. The Deep Learning algorithms learn, by repeating a task a lot of times. In every repetition, the algorithms tweak a little in order to improve their result. This happens in the same way that humans learn from experience. More specifically, the characterization "deep" in deep learning, comes from the depth of the layers, which enables learning.

Moreover, that amount of data that is generated nowadays is enormous, and it is the reason that deep learning works. Deep Learning algorithms need a lot

of data in order to work properly, and the increase of data through the years is making this possible. Another factor that contributed to the era of Deep Learning is of course the computational power that is available today.

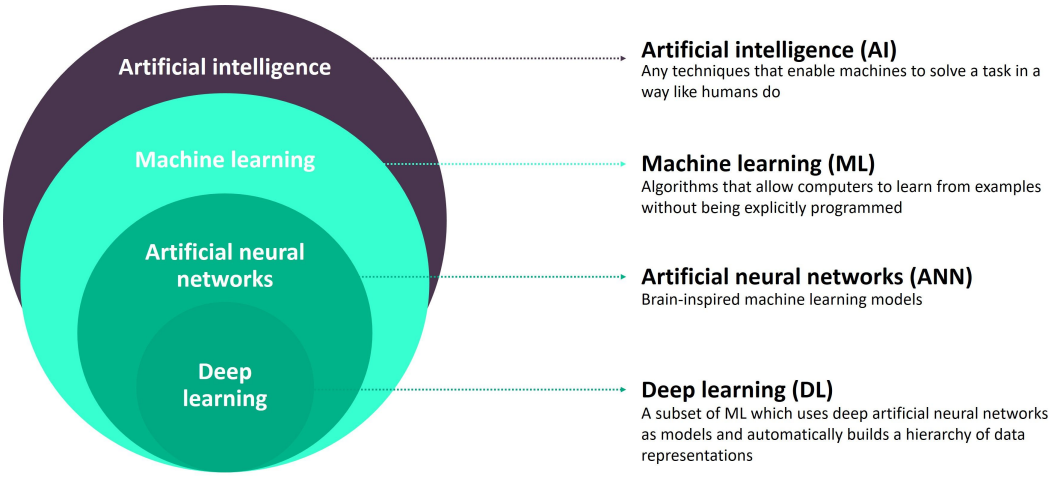

Figure 2.1: The layers of Artificial Intelligence as depicted in [1]

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) [28] are computing systems inspired by the biological neural networks inside our brains. They are based on a collection of connected units or nodes, the famous artificial neurons, which mimic the neurons of our brain. Every connection in the network transmits a signal to other neurons, where it is being processed and transferred to the connected neurons. The output of each neuron is computed by a non-linear function of the sum of its inputs. Each neuron has a weight that changes in the training process, as the neuron learns through it. The function of each weight, is that it increases or decreases the transfer of the signal at the connected neurons.

A single layer neural network is called a Perceptron, and it produces a single output.

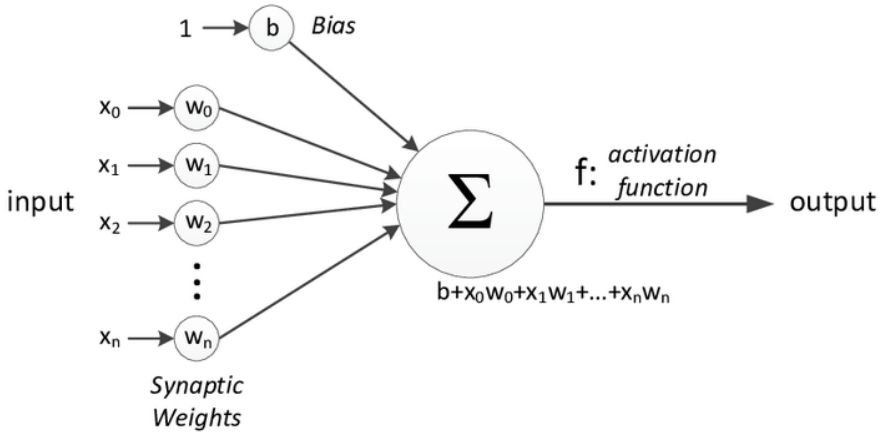

Figure 2.2: Overview of the Perceptron as depicted in [2]

In Figure 2.2, the $x_0, x_1, x_2 \ldots x_n$ represent the various inputs to the Perceptron. The symbols that we see inside the small circles, the $w_0, w_1, w_2 \ldots w_n$ are the weights, which show the strength of the node. b is called bias, and its use is to shift the activation function up or down. The products of the node are summed and go straight into the transfer function, or activation function. This function generates the result, which is the output of the Perceptron.

If we put it together mathematically, we have $x_1w_1 + x_2w_2 + \ldots ...x_nw_n = \sum x_iw_i$. Then this passes through the activation function and we have $\phi\left(\sum x_iw_i\right)$.

The activation function is important because it helps the Network to learn complicated tasks. Its purpose is to convert the neuron's signal to an output signal, which is then used as an input to the next node. The need of the activation function is to introduce non-linearity into the output of the neuron.

Without an activation function the output will always be linear. A function like that is very limited to its complexity and power, and the Network cannot learn complicated data, like videos in our case. On the other hand, non-linear functions have a curvature and a degree more than one. This is important, because we need an artificial neural network to learn and represent almost anything, like complex functions.

Here we list the most used activation functions:

1. Binary Step Function

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases} \tag{2.1}$$

   It is mostly used as a threshold function for each neuron.

2. Sigmoid Activation Function (Logistic Function)

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

   The sigmoid curve in this function ranges between 0 and 1, and therefore, sigmoid function is used for models where we need to predict the probability as an output.

3. Hyperbolic Tangent Function (tanh)

$$f(x) = \tanh x \tag{2.3}$$

   This function maps the strong negative inputs to negative outputs and zero inputs to near-zero outputs. In this way the algorithm is less likely to get stuck during training.

4. Rectified Linear Units (ReLU)

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.4}$$

   ReLU function is non-linear, and every combination of it is also non-linear. It is a very good approximator because every function can be approximated

with a combination of ReLU. It should only be applied to hidden layers of a neural network. A problem with ReLU is that some gradients are fragile during training and can die. It can cause a weight to never activate on any data point again. Basically ReLU could result in dead neurons.

2.3 Long Short-Term Memory Networks

Long Short-Term Memory Networks (LSTM) [29] are a type of Recurrent Neural Networks (RNN) [30][31][32], that can learn and remember long sequences of data, due to its gates. These gates regulate the input data of the network, and thus the information flow that goes inside the network.

The RNNs have some limitations which are solved by LSTMs. These limitations are:

1. **Short-term memory**. The RNNs forget the older information in order to remember the new data, which results in the loss of important information.

2. **Vanishing Gradient**. The weight is updated from the gradient. If the value of the gradient is very small, then it does not contribute to learning. In the vanishing gradient, the gradient value shrinks and becomes very close to zero.

3. **Exploding Gradient**. This is the opposite from the vanishing gradient problem. Here the network assigns very high values to the gradients, which means that it assigns them very high importance.

The significant difference between RNN and LSTM is that the latter uses gates in order to decide whether it should keep or forget information. As we can see in Figure 2.3, an LSTM cell has a cell state, input, forget and output gates and activation functions.

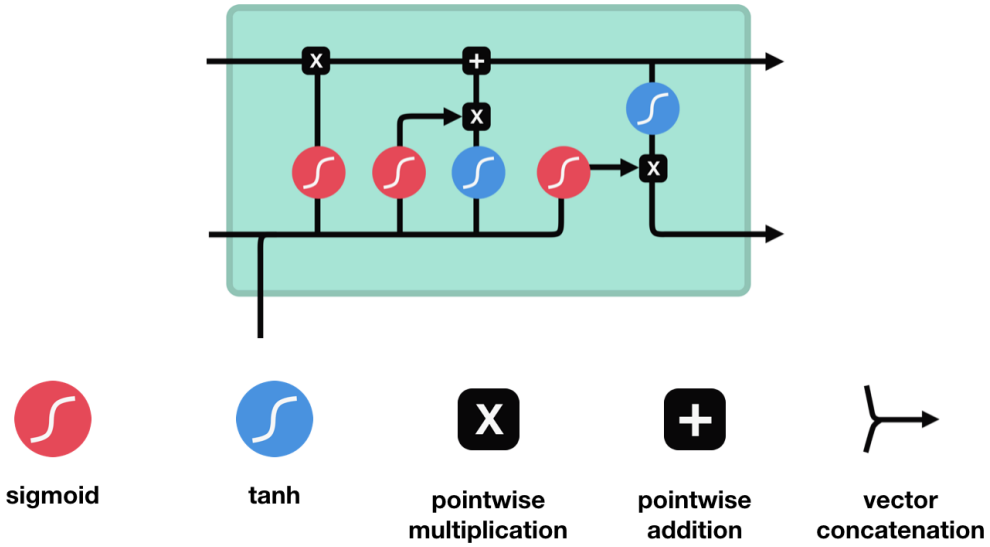

Figure 2.3: LSTM cell and its operations as depicted in [3]

An LSTM, like an RNN, processes a sequence of vectors, and it passes the previous hidden state to the next step of the sequence. The hidden state is in fact the memory of the model. It holds information on previous data that it has "seen" before. The important thing here is the calculation of the hidden state. First of all, the input and the previous hidden state form a vector. This vector goes through the tahn activation function with information from the current and the previous inputs. The result of this function is the new hidden state. Next we present briefly the components of an LSTM.

2.3.1 Tanh

The tanh activation in the LSTM regulates the input that flows in the network. As we have seen previously in Equation 2.3, it squishes the values to be between -1 and 1. This helps the network to regulate some values that may become very

high, which causes other values to vanish. It ensures that a value will stay within the boundaries.

2.3.2 Sigmoid

A sigmoid activation, as we saw in Equation 2.2 is similar to the tanh activation. It squishes the input values between 0 and 1. In that way the network either updates or forgets data. On the one hand, every value that is multiplied by 0 becomes 0, so the network forgets it. On the other hand, if it is multiplied by 1, the value stays the same, thus the network chooses to keep it.

2.3.3 Forget Gate

The Forget Gate decides what information should be thrown away or kept. Data from previous states and from the current input is passed to a sigmoid function. Because of the sigmoid function the values are between 0 and 1. Closer to 0 means that the network will forget the value, while closer to 1 means that the network will keep the value.

2.3.4 Input Gate

In order to update the cell state, we pass the previous hidden state and current input into a sigmoid function, which transforms values between 0 and 1. As we have previously mentioned, 0 is unimportant and 1 is important. The LSTM also passes the hidden state and the current input into the tanh function in order to regulate the network. Then the tanh output is multiplied with the sigmoid output. Obviously, the sigmoid output decides which information to keep from the tanh output.

2.3.5 Cell State

The cell state is pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. The output of the input gate is pointwise added, which updates the cell state to the new values that our network finds relevant and important. In that way the new cell state is produced.

2.3.6 Output Gate

The output gate decides what the next hidden state should be. The hidden state, which contains information on previous inputs, is also used for predictions. First of all, the previous hidden state and the current input are passed into a sigmoid function. Then the resulted cell state is passed to the tanh function. The output of the tanh function is multiplied with the sigmoid output, in order to decide which information the hidden state should carry. The output of this is in fact the hidden state. The new cell state and the new hidden state are then carried over to the next time step.

Chapter 3

Methodology

Summary

This Chapter presents the proposed methodology of our approach in order to tackle the sign language translation problem. We present the models that we have implemented, which are the Encoder-Decoder with neural attention model and the Transformer.

3.1 Encoder-Decoder with Neural Attention

The Encoder-Decoder model falls into the category of sequence to sequence models. Sequence to sequence models have achieved significant results on complicated tasks like machine translation [33], speech recognition [34] and video captioning [35]. This architecture can be used to tackle any sequence-based problem, and especially the ones where the inputs and outputs have different sizes and categories.

A sequence to sequence model maps a fixed-length input, to a fixed-length output where the length of the input and output may differ. For instance, translating

a sentence from English to French, which have different lengths; or translating an image to a sentence. These tasks cannot be implemented with an LSTM or a CNN, so that is why we need a sequence to sequence model for these complicated problems. These models have the ability to map sequences of different lengths to each other. In the majority of Deep Learning problems for sequence to sequence models, the inputs and the outputs are not correlated and their lengths are different.

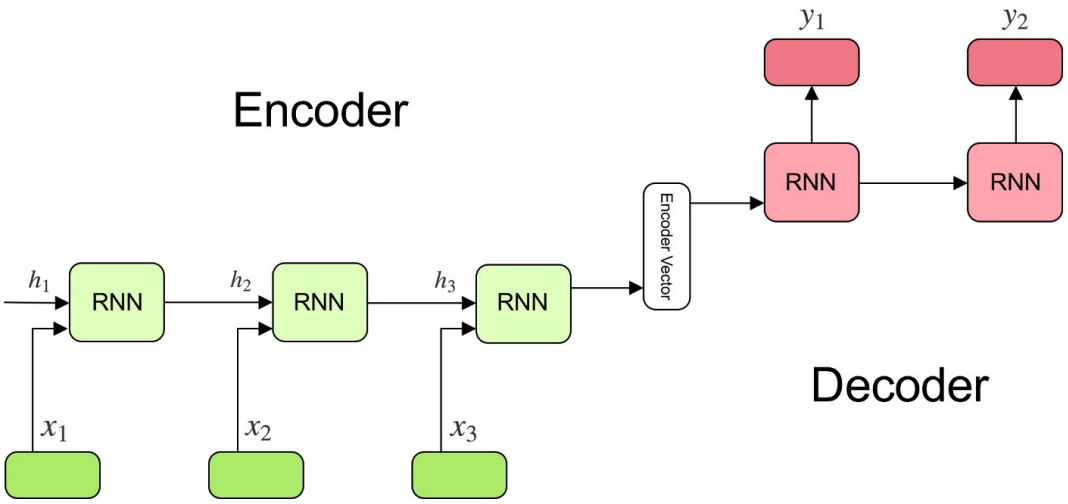

Figure 3.1: A sequence to sequence model. The Encoder and the Decoder are composed of RNNs as depicted in [4]

Figure 3.1 depicts a sequence to sequence model, consisting of 2 parts, the Encoder and the Decoder. We provide a brief overview of them.

3.1.1 Encoder

Figure 3.1 depicts an Encoder, consisting of several recurrent units, which can be either LSTMs or Gated Recurrent Units(GRUs). Each of them accepts a single element of the input sequence, it processes it and then it propagates it forward. In natural language processing problems, the input sequence is a sequence of words,

and every word is represented as x_i, where i is the order of that word. The hidden states h_i of the RNNs are computed using the following equation:

$$h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right) \tag{3.1}$$

Equation 3.1 represents the output of a Recurrent Neural Network inside the Encoder. It is clear from this equation that we apply the appropriate weights to the previous hidden state h_{t-1} and the input x_t.

The Encoder produces as an output an encoder vector, which is the final hidden state produced from the encoder and it is calculated from Equation 3.1. This output has as a goal to include the information for all the inputs of the Encoder, so that the Decoder can make accurate predictions. It is noteworthy that this vector acts as the initial hidden state of the decoder of the model.

3.1.2 Decoder

Similarly to the Encoder, the Decoder is a stack of Recurrent Neural Networks, as presented in Figure 3.1; and every network predicts an output y_t at a time step t. Every RNN accepts the hidden state from the previous unit and produces an output and its own hidden state. In Natural Language Processing problems, the output sequence is a sequence of words, and every word is represented as y_i, where i is the order of that word. The hidden state h_i of the each RNN in the Decoder is calculated from the following equation:

$$h_t = f\left(W^{(hh)}h_{t-1}\right) \tag{3.2}$$

Equation 3.2 shows that every hidden state is computed from the previous hidden state. The output y_t of every RNN in the Decoder at time step t is

computed using the following equation:

$$y_t = \text{softmax}\left(W^{(S)} h_t\right) \tag{3.3}$$

As we see in Equation 3.3, the outputs are calculated using the hidden state at the current time step together with the respective weight $W^{(S)}$. A Softmax activation is used in the output layer in order to create a probability vector that will determine the final output.

3.1.3 Embedding Layer

It may not be depicted in Figure 3.1 becuase of its simplicity, but it is common to use an Embedding Layer [36] inside a sequence to sequence model. The Embedding layer in general "turns positive integers (indexes) into dense vectors of fixed size". There are 2 main reasons to use the Embedding Layer in Natural Language Processing Problems:

- One-hot encoded vectors are high dimensional and sparse. Having a dictionary of 5000 words (for instance in an email problem) and using one-hot encoding, means that every word will be a vector containing 5000 integers (0s and 1s). 4999 of those integers would be 0, and only one of them would be 1. This is computationally expensive and becomes worse when we deal with image data.

- The vectors of each embedding get updated while training the neural network. The Embedding Layer creates a multi-dimensional space which helps to identify similarities and relationships between words, and everything that can be turned into a vector through this Layer.

In order to explain the Embedding Layer we provide a simple example. The Embedding Layer encodes a sentence by indices, which means that it assigns an

index to each unique word. Therefore, sentence "deep learning is very deep" would become "1 2 3 4 1". Then the embedding matrix is created, and we decide the number of 'latent factors' that are assigned to each index. A possible embedding matrix of the aforementioned sentence example would be the following:

Indices	Latent Factors					
1	.32	.02	.48	.21	.56	.15
2	.65	.23	.41	.57	.03	.92
3	.45	.87	.89	.45	.12	.01
4	.65	.21	.25	.45	.78	.82

Figure 3.2: Embedding Matrix

As we observe in Figure 3.2 instead of having a one-hot encoded vector, we use an embedding matrix, and we keep the size of each vector much smaller. Every row in Figure 3.2 represents a word, so "deep" is the first row of the matrix(.32, .02, .48, .21, .56, .15). More specifically, the words are not replaced by vectors in the matrix; they get replaced by the indices of the matrix. This is more computationally efficient in big datasets. These word vectors get updated during the training process of the deep neural network, and they help us explore which words are similar to each other in a multi-dimensional space. Although here we have presented a simple example of the Embedding Layer in words, we note that it is implemented in the same way for other types of data, like images.

3.1.4 Neural Attention

In this subsection we provide a brief overview of Neural Attention, a mechanism that has been used in several state-of-the-art systems and open-source toolkits.

As illustrated in Figure 3.3, the attention computation happens at every decoder time step. The attention mechanism that we present was first proposed by [6]. In the Encoder-Decoder model every decoder output is calculated based on the previous outputs and some x, where x consists of the current hidden state

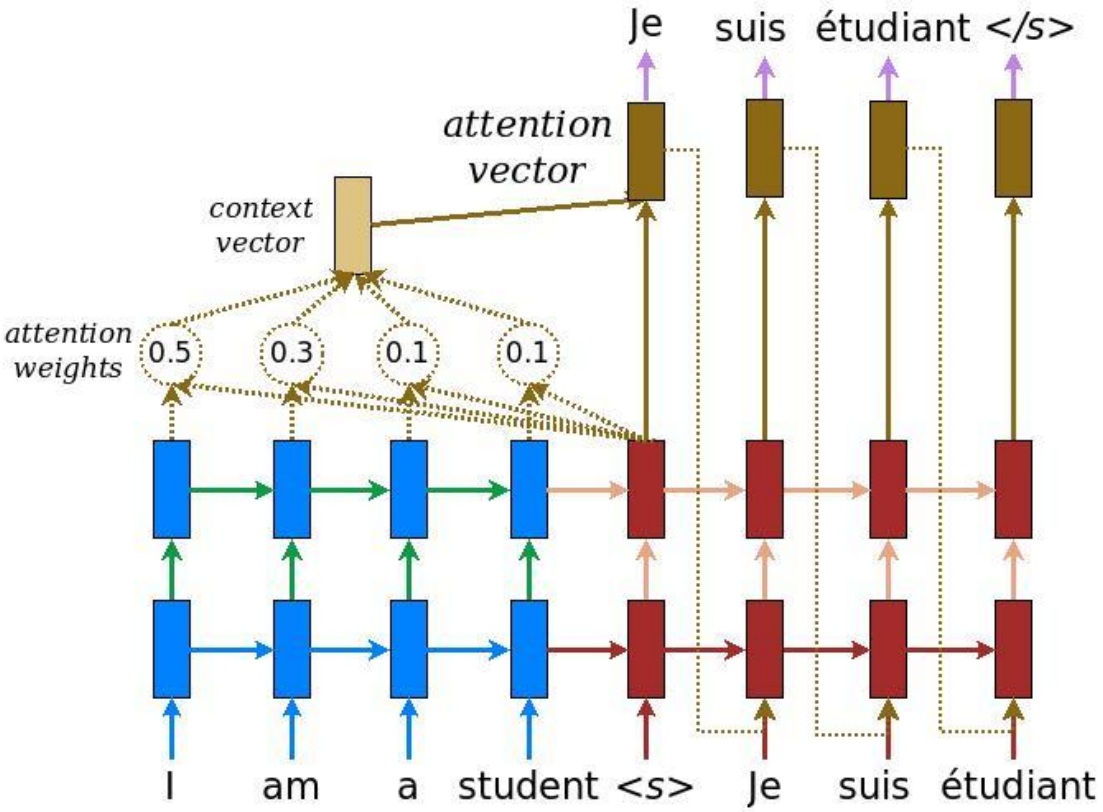

Figure 3.3: Attention mechanism – example of an attention-based NMT system as depicted in [5]. We highlight in detail the first step of the attention computation.

and the attention "context". So the decoder defines a probability over the results y by decomposing the joint probability into the ordered conditionals:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p\left(y_t | \left\{y_1, \cdots, y_{t-1}\right\}, c\right) \tag{3.4}$$

where y_t is the next prediction and $\{y_1, \cdots, y_{t-1}\}$ are all the previous predictions that the model has made so far; c is the context vector, and $\mathbf{y} = (y_1, \cdots, y_t)$. With an RNN, each conditional probability is modeled as:

$$p\left(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}\right) = g\left(y_{i-1}, s_i, c_i\right) \tag{3.5}$$

where g is a nonlinear function that outputs the probability of y_t, and s_i is the current hidden state calculated by an RNN f with the last hidden state s_{i-1}, computed by:

$$s_i = f\left(s_{i-1}, y_{i-1}, c_i\right) \tag{3.6}$$

The context vector c_i depends on a sequence of annotations $(h_1, \cdots, h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i -th part of the input sequence.

The context vector c_i is a weighted sum of all encoder outputs, where each weight a_{ij} is the amount of "attention" paid to the corresponding encoder output h_j.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.7}$$

The weight α_{ij} of each annotation h_j is normalized and computed by:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)} \tag{3.8}$$

where

$$e_{ij} = a\left(s_{i-1}, h_j\right) \tag{3.9}$$

is an alignment model which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state S_{i-1} and the j -th annotation h_j of the input sentence. Here in our proposed approaches we used the attention mechanism from [6]. Another famous attention mechanism was proposed by [5]. The main difference between [5] and [6], is in the output of the decoder model, as we observe in Figure 3.4:

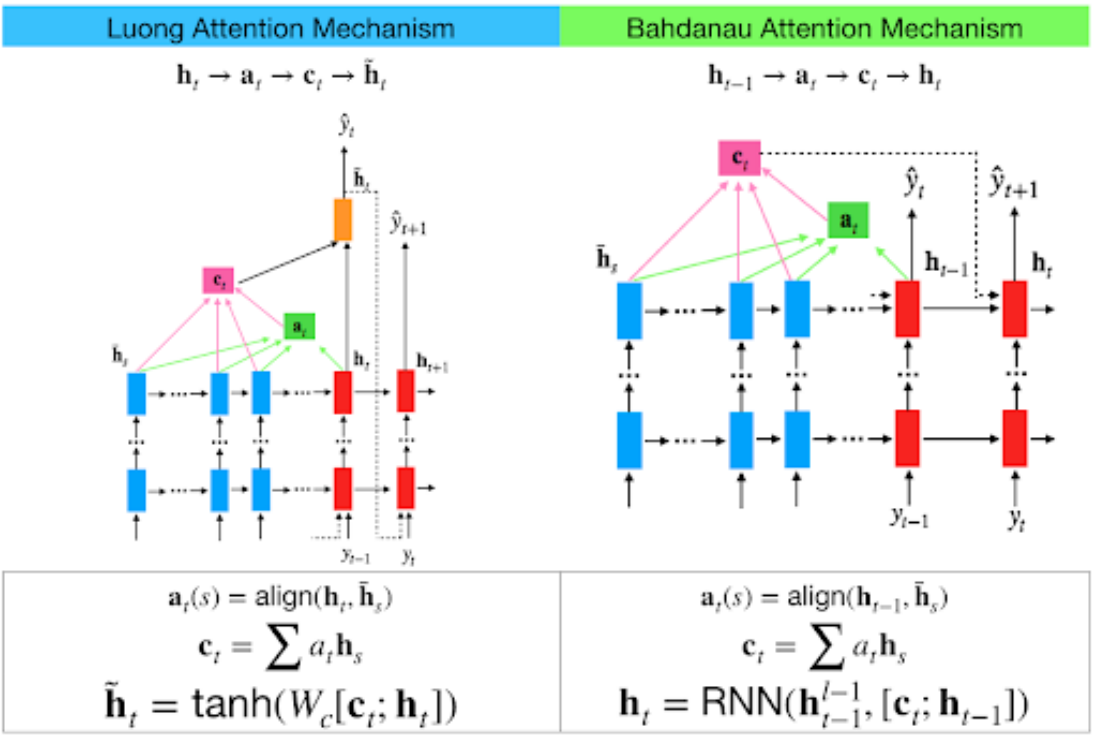

Figure 3.4: The 2 attention mechanisms as depicted in [5] and [6].

3.1.5 Transformer

The Transformer model has self-attention, which is the ability to attend to different positions of the input sequence to compute a representation of that sequence [7]. The Transformer creates stacks of self-attention layers inside the model, and it handles variable-sized input using these stacks of self-attention layers instead of RNNs or CNNs. It has several advantages:

1. It does not make assumptions about the temporal/spatial relationships across the data. In that way we can effectively process a set of objects [37].

2. Layer outputs can be calculated in parallel, instead of a series like an RNN. In this way it can run faster than a normal Encoder-Decoder model.

3. Distant input data can affect each other's output without passing through many RNN-steps, or convolution layers [38].

4. It can learn long-range dependencies because of the attention layers, which is a challenge in many sequence tasks.

The Transformer architecture, apart from the advantages has also disadvantages:

1. In time-series problems, the output for each time-step is calculated from the entire history, instead of only the inputs and current hidden state. This may be less efficient when our goal is to predict only one value at a specific time.

2. If the input data has a temporal relationship like text, some positional encoding must be added or the model will effectively see a bag of words. As a result the neural attention mechanism will lose its properties.

3.1.5.1 Positional Encoding

The Transformer does not contain any recurrence or convolution, so positional encoding is added as a vector, in order to give the model some information about the relative position of the words in the sentence. The positional encoding vector is added to the embedding vector. Embeddings represent a token in a d-dimensional space where tokens with similar meaning will be closer to each other, as we have mentioned earlier in Subsection 3.1.3. The Embeddings do not encode the relative position of words in a sentence. So after adding the positional encoding, words will be closer to each other based on the similarity of their meaning and their position in the sentence, in the d-dimensional space. The encoding vector is a vector of sines and cosines at each position, wehre each sine-cosine pair rotates

at a different frequency. The formula for calculating the positional encoding is the following:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{model}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{model}}\right) \end{aligned} \tag{3.10}$$

3.1.5.2 Masking

Masking is an important process of the Transformer architecture. We mask all the pad tokens in the batch of sequence. This process ensures that the model does not treat padding as the input. The mask indicates where pad value 0 is present: it outputs a 1 at those locations, and a 0 otherwise.

The look-ahead mask is used to mask the future tokens in a sequence. This means that the mask indicates which entries should not be used. For instance, in order to predict the third word in a sentence, only the first and the second words will be used. On the same scope, in order to predict the fourth word, only the first, the second and the third words will be used. It is noteworthy to mention that masking is a unique process in the Transformer architecture, and does not occur in the classical Encoder-Decoder models.

3.1.5.3 Scaled Dot Product Attention

The attention function used by the Transformer takes three inputs: Q (query), K (key), V (value). The equation used to calculate the attention weights is the following:

$$\text{Attention}(Q, K, V) = \text{softmax}_k\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.11}$$

As depicted in Equation 3.11 the dot-product attention is scaled by a factor of square root of the depth. This happens because, for large values of depth, the

dot product grows large in magnitude and pushes the softmax function where it has small gradients resulting in a very hard softmax. So attention would not work properly otherwise.

For instance, we assume that Q and K have a mean of 0 and variance of 1. Their matrix multiplication in this case, would have a mean of 0 and a variance of d_k. Therefore, the square root of d_k is used for scaling because the matrix product of Q and K should have a mean of 0 and variance of 1. In this way we get a gentler sofrmax.

We should note that the mask is multiplied with -1e9 (a very small number, close to negative infinity). This is done because the mask is summed with the scaled matrix multiplication of Q and K and is applied immediately before a softmax. The goal is to zero out these cells, and large negative inputs to softmax are near zero in the output.

As the softmax normalization is done on K, its values decide the amount of importance given to Q. The output represents the multiplication of the attention weights and the V(value) vector. The method guarantees that the words you want to focus on are kept as-is and the irrelevant words are flushed out in the end. Figure 3.5 presents the Scaled dot product attention, as depicted in [7].

3.1.5.4 Multi-head Attention

Multi-head attention consists of four parts as we observe in Figrue 3.6:

1. Linear layers and split into heads.
2. Scaled dot-product attention.
3. Concatenation of heads.
4. Final linear layer.

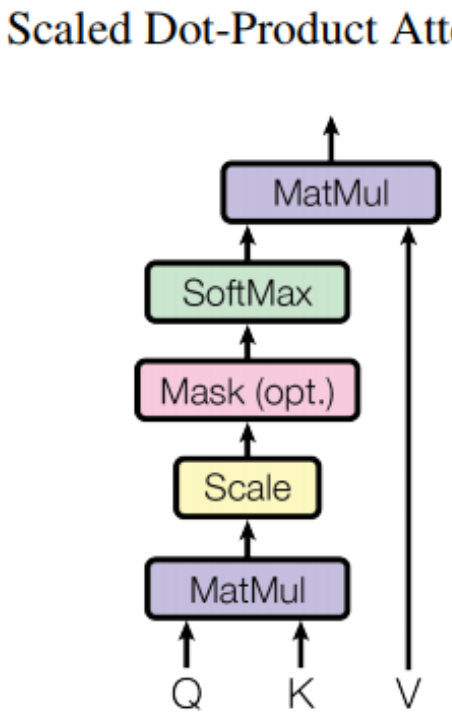

Figure 3.5: Scaled dot product attention, as depicted in [7]

Each multi-head attention block gets three inputs; Q (query), K (key), V (value). These are put through linear (Dense) layers and split up into multiple heads. The Scaled Dot Product Attention described in Subsection 3.1.5.3 is applied to each head, and it is broadcasted for efficiency. An appropriate mask is used in the attention step. The attention output for each head is then concatenated as we see in Figure 3.6 and put through a final Dense layer.

Instead of one single attention head, Q, K, and V are split into multiple "heads", since it allows the Transformer to jointly attend to information at different positions from different representational spaces. After the split on the previous step, each head has a reduced dimensionality, so the total computation

cost is the same as a single head attention with full dimensionality.

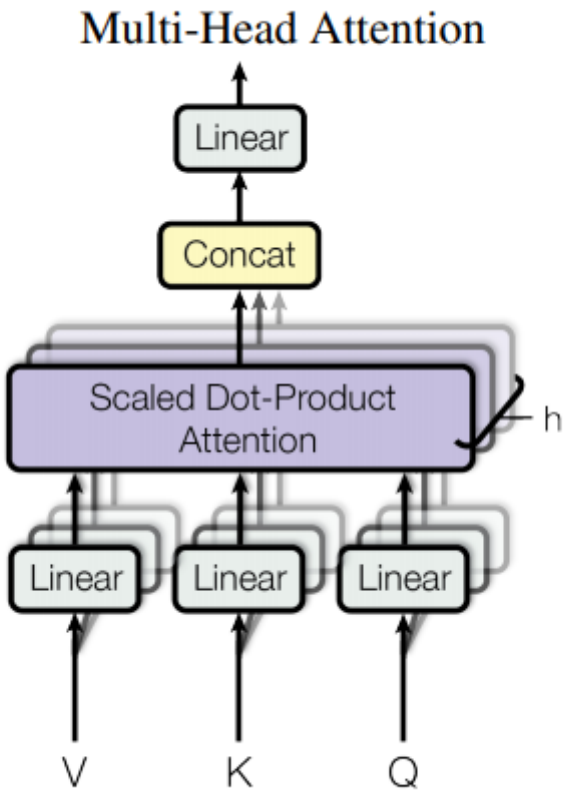

Figure 3.6: Multi-head attention, as presented in [7]

The transformer model follows the same general pattern as a standard sequence to sequence with attention model (Encoder-Decoder model), that we described in this Section. This means that in an NLP problem, the input sentence is passed through N encoder layers, and every one of them generates an output for every word in the sentence. Then the decoder attends the encoder's output and its own input (self-attention) to predict the next word. Each layer in the Encoder consists of the sublayers Multi-head attention (with padding mask) and point-wise feed forward networks. Each of these sublayers has a residual connection around it followed by a layer normalization. Residual connections help in

avoiding the vanishing gradient problem in deep networks.

The Decoder on the other hand consists of 3 different sublayers. The first one is the Masked multi-head attention (with look-ahead mask and padding mask). The second one is the Multi-head attention (with padding mask). V (value) and K (key) receive the encoder output as inputs. Q (query) receives the output from the masked multi-head attention sublayer. The last one is the point-wise feed forward networks. Each of these sublayers has a residual connection around it followed by a layer normalization. As Q receives the output of the decoder's first attention block, and K receives the encoder output, the attention weights represent the importance given to the decoder's input based on the encoder's output. This means that the decoder predicts the next word by looking at the encoder output and self-attending to its own output. This is how the attention inside the Transfromer works. Figure 3.7 shows the architecture of the Transformer model, as originally depicted in [7].

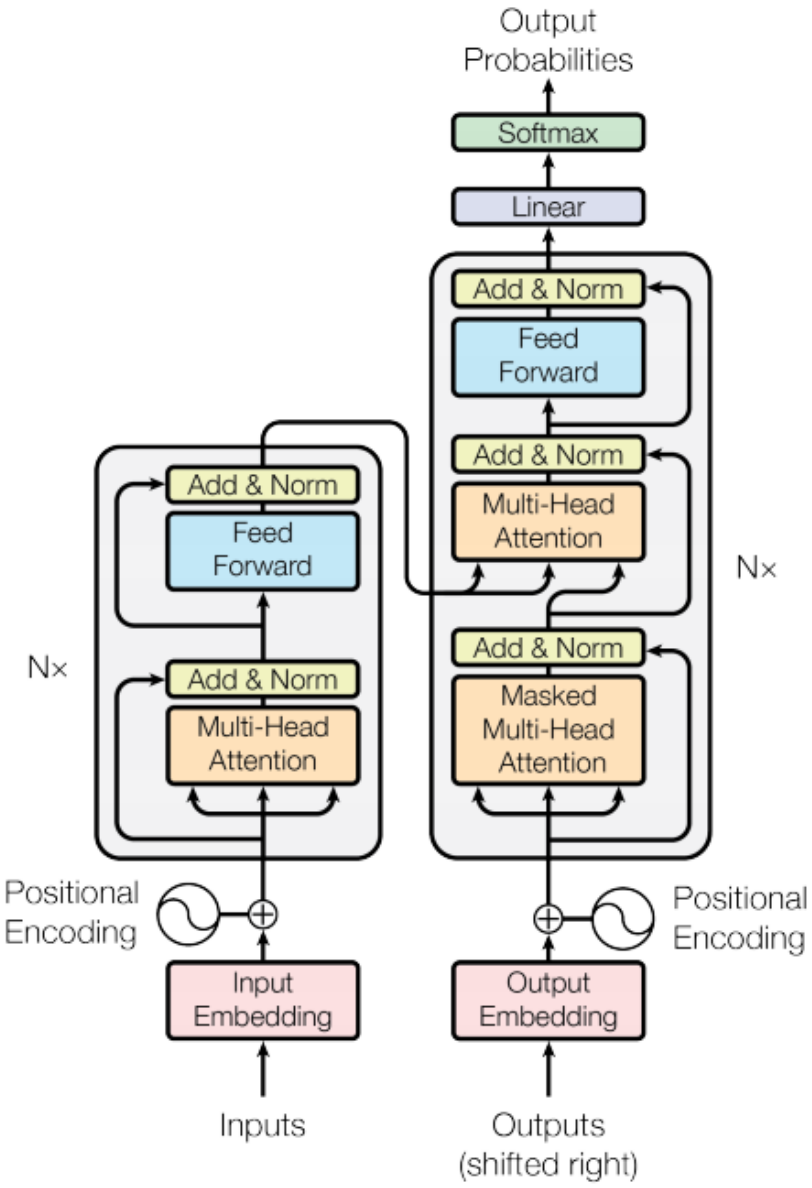

Figure 3.7: The Transformer architecture, as depicted in [7]

Chapter 4

Data and Experimental Framework

Summary

This Chapter presents the datasets that we have used in our approach and the experimental setup of the proposed models.

4.1 Data

For our experiments we have used the well-known RWTH-PHOENIX-Weather 2014T dataset. This dataset provides German sign language videos, regarding weather forecasts. Regarding more details about the data, the signing was recorded by a stationary color camera placed in front of the sign language interpreters. The interpreters wear dark clothes in front of an artificial grey background with color transition. All recorded videos are at 25 frames per second and the size of the frames is 210 by 260 pixels. Each frame shows the interpreter box only [8]. Table 4.1 presents some statistics about the dataset we used. Figure 4.1 shows example images of different signers and the percentage of images that correspond to them.

	RWTH-PHOENIX-Weather 2014T
#signers	9
#editions	645
duration[h]	10.73
#frames	965,940
#sentences	8,767
#run. glosses	123,532
vocab. size	2,589
#singletons	531

Table 4.1: Statistics of the RWTH-PHOENIX-Weather corpus, as depicted in [8]

The RWTH-PHOENIX-Weather corpus is suitable for the implementation and evaluation of various automatic recognition systems for continuous and isolated sign language. The spatial annotations allow for the evaluation of hand tracking and face tracking systems as well as for face detection systems. It is also very useful for the creation of sign language translation systems and algorithms between the signs and the German language. In our case we use this dataset for the latter purpose.

In order to address the problem of sign language translation, the authors [8] have defined two setups, corresponding to the single signer and multi signer setup of the sign language recognition corpus. So in that way, the output of the sign language recognition system can be passed to the translation system, creating a video-to-text translation pipeline. The statistics of the corpus for the sign language translation task are shown in Table 4.2

The identity of the signer hardly affects the sign language translation system, so all of the signers have been included in the train set. For the test set, there is one signer featured in the single signer setup, and several signers in the multi signer setup. Both single signer and multi-signer setups are a challenge to computer vision algorithms because of the low resolution and the motion blur of

		Glosses	German
Train:	sentences	8495	
	running words	99207	134927
	vocabulary	1580	3047
	singletons/voc	35.8%	37.9%
Dev:	sentences	250	
	running words	2573	3293
	OOVs (running)	1.3%	1.8%
Test:	sentences	2×73	
(single signer)	running words	487	921
	OOVs (running)	1.6%	(5.4%)
Test:	sentences	2×135	
(multi signer)	running words	946	1753
	OOVs (running)	0.7%	(2.4%)

Table 4.2: Overview of RWTH-PHOENIX-Weather Sign Language Translation Corpus as depicted in [8]

the images. In building both setups from single-view video data, approaches to sign language recognition can be evaluated in real-life scenarios where additional information due to RGB-D cameras or stereo-depth information is not available.

4.1.1 Choice of the target translation

In this subsection we explain the target that we chose for the final translation. As we have mentioned the RWTH-PHOENIX-Weather corpus provides the sign videos, the glosses and the text translation. The ultimate goal of Sign Language Translation is to translate directly from continuous sign videos to spoken language text, without going through an intermediary representation step, like glosses for instance.

Glosses are textual representations of multi-channel temporal signals, so they represent an information bottleneck for any translation system. Moreover glossing is not when we are trying to interpret a language, but when we are trying to transcribe it or represent it in general. Glossing has to do with the fact that the

Figure 4.1: Example images and percentage of data performed by signer in RWTH-PHOENIX-Weather corpus. Top, left to right signers 1 to 5, bottom signers 6 to 9 as shown in [8]

target language, which in our case is text language may not have equivalent words to represent the sign language. Hence glosses are imprecise for sign language translation.

Direct Sign Language to Text Translation is more difficult than Gloss Sign Language to Text Translation and Sign Language to Gloss Translation. But we believe that with the use of an advanced model mechanism we can achieve good performance on direct Sign Language to Text Translation.

4.2 OpenPose

For some experiments we have used the images from RWTH-PHOENIX-Weather 2014T in their raw form, while in others we have used the keypoints derived from OpenPose [39][40][41][42]. OpenPose, was developed by researchers at the

Carnegie Mellon University and can be considered as the state of the art approach for real-time human pose estimation.

A common approach is to follow a two-step framework which uses a human detector and solve the pose estimation for each human. This approach running time tends to grow with the number of people in the image and make the real-time performance a challenge. In OpenPose, the authors provide a bottom-up approach where the body parts are detected by the model and a final parsing is used to extract the pose estimation results. This approach can decouple the running time complexity from the number of people in the image.

The steps of the overall OpenPose pipeline are the following:

- First the image is passed through a baseline network to extract feature maps [43]. For this step the OpenPose authors use the first 10 layers of VGG-19 model [44][45].
- The feature maps that were extracted from the aforementioned network, are processed with multiple stages CNN to generate: (i) a set of Part Confidence Maps, (ii) a set of Part Affinity Fields (PAFs).
- Finally, the Confidence Maps and Part Affinity Fields are processed by a greedy algorithm to obtain the poses for each person in the image.

In order to explain OpenPose better we analyze the last step of it, the greedy algorithm. To provide a better explanation, the Part Confidence Maps are a set of 2D confidence maps for body part locations. Each joint location has a map. The Part Affinity Fields (PAFs) are a set of 2D vector fields L which encodes the degree of association between parts. This step is used to parse poses of multiple people from confidence maps and part affinity fields. We note that this is a difficult step from a mathematical approach.

- Step 1: Find all joints locations using the confidence maps.
- Step 2: Find which joints go together to form limbs (body parts) using the part affinity fields and joints in step 1.
- Step 3: Associate limbs that belong to the same person and get the final list of human poses.

Regarding the first step from above, finding all joints locations using the confidence maps, the algorithm tries to find 18 joints in total. For each joint we get the corresponding 2D heat-map for the joint in confidence maps, and then we find the peaks by thresholding the 2D heat-map. For each heat-map we take a patch around the peak in the heap, and we scale up the patch using the up-sampling scale. After that we get the maximum peak location in the scaled up patch, and we add the peak information to the list peaks of the joint.

The process for step 2, that is finding which joints go together to form limbs (body parts) using the part affinity fields and joints from the first step, is more complicated than the process in the first step. Firstly we scale up the RAFs to the input size using the difference in width/height of input image and the PAFs maps. Then for each limb type we do the following procedure:

- We get all source joint peaks and destination joint peaks.
- If there are no source or destination peaks we skip this limb.
- For each source peak and destination peak:
    1. We get the direction vector by subtracting the destination and the source locations.
    2. We normalize the direction vector to a unit vector.

3. We get the PAFs' values at each intermediate points between the source and destination peaks.
4. We calculate the score of the current limb connection by averaging the PAF's values.
5. We add a score to penalize the long distance limb.
6. Finally, we add the current limb connections to the limb connection candidates.

- We sort the limb connection candidates.
- Finally for each limb connection candidates, we add the connection to the final list if the source and destination is not selected for any connection.

For the third and last step, we associate limbs that belong to the same person and get the final list of human poses. For this step, we first find the persons that associated with either joint of the current connection, for each limb types and for each connection in connected limbs list of that type. Then, if there is no person we create a new person with the current connection. If there is already 1 person, we add the current connection to that person. If there are 2 persons, we merge them into 1 person. If a person has very few joints, we just remove it.

Although OpenPose jointly detects human body, hand, facial, and foot keypoints (in total 135 keypoints) on single images, for our purpose we detect only the keypoints in the upper body and hands. Figure 4.2 depicts some examples OpenPose applied on data from RWTH-PHOENIX-Weather 2014T.

4.3 Experimental Setup

In this section we explain in detail the implementation setup for our experiments for training and inference. We have conducted 2 different series of experiments;

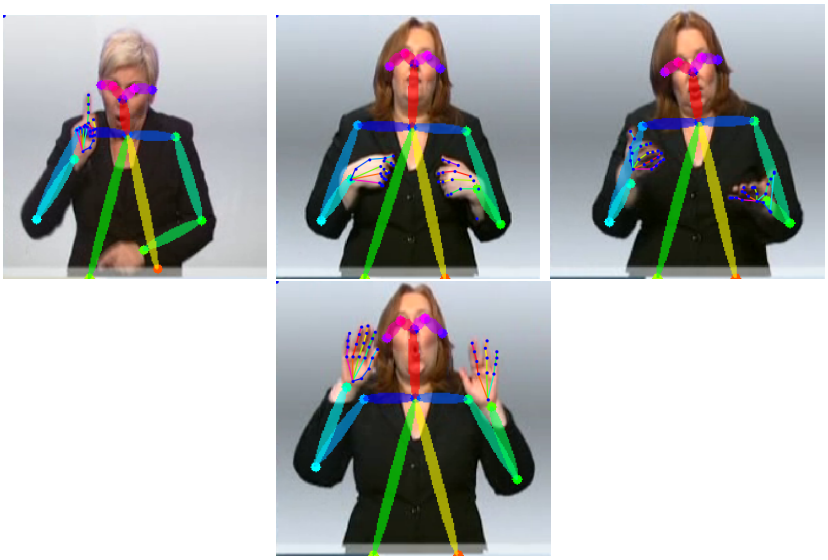

Figure 4.2: Example images of RWTH-PHOENIX-Weather 2014T with OpenPose keypoints

one with the classical Encoder-Decoder with neural attention model and one with the Transformer.

4.3.1 Setup of the Encoder-Decoder with neural attention

For the Encoder-Decoder with attention model we have implemented the classical RNN-based model. More specifically, the Encoder consists of 2 LSTM layers and 2 Dense layers. The Decoder consists of 1 attention layer, 1 embedding layer, 1 GRU layer and 2 dense layers. A more concise image of this model is depicted in Figure 4.3. Getting into more details we decided to use the attention model as described in [6]. For another Encoder-Decoder with neural attention, we implemented the same model setup, but instead of LSTMs we used GRUs. While this is the main setup for the Encoder-Decoder model we have implemented different experiments by changing some parameters of the main model, in order to see which combination of them produces the best performance.

Regarding the data input, we extract from OpenPose the keypoints from the

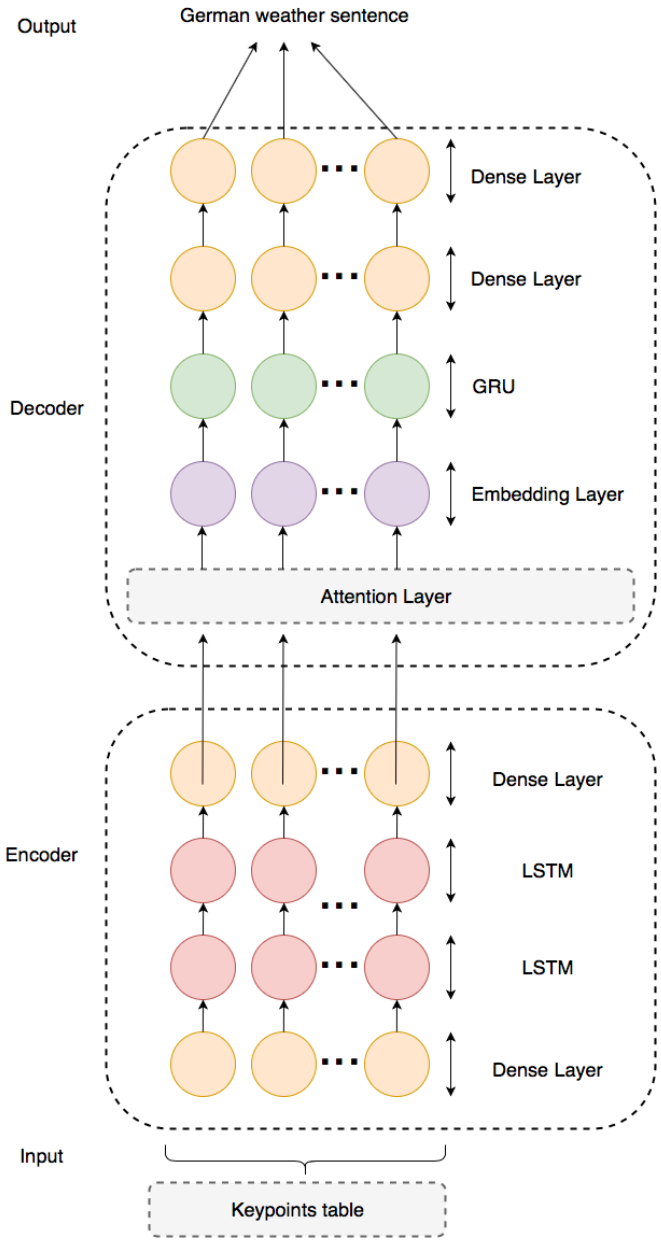

Figure 4.3: The LSTM-based Encoder-Decoder with neural attention model

upper body and hand for every frame of each video. We store them in an table, in a way that every table corresponds to a video. Every table has dimensions of 120x475. 120 are the keypoints from OpenPose, and 475 are the number of rows,

which represents the number of frames in every video. Most of the videos have a smaller number of frames, but we need to feed the model with a standard input shape. So we chose 475 as the biggest number of frames in a video in the RWTH-PHOENIX-Weather 2014T dataset. We set the batch size to 32, the buffer size to 1000, the embedding dimension to 1000 and the units of the Neural Networks to 512. We found out that by choosing a small number of the last 3 variables, the network produces worse results. These are the details for this model setup.

For the training setup we used the Adam optimizer [46] in order to optimize the model during training time with a learning rate of 0.001. As a loss function we chose the Sparse Categorical Crossentropy [47][48]. The LSTM-based model was trained for approximately 800 epochs, and the GRU-based model for 1100. Both of the models did not seem to converge.

4.3.2 Setup of the Transformer

As we have previously mentioned in Section 3.1.5, the Transformer is a sequence-to-sequence encoder-decoder network. The recurrent networks are replaced here with self-attention layers. The architecture that we used is presented in Figure 4.4 and it is the same as the model in [9]. Our model has 2 same Encoder layers and 2 same Decoder layers, which are the same as in the original Transformer model. Our Transformer model was implemented with a word embedding size of 512 hidden units, and 2048 feed-forward layers. We optimize the training of it with Adam, with 0.9 β_1 and 0.998 β_2, as well as Noam learning rate schedule, 0.1 dropout, gradient clipping with 0 threshold, and 0.1 label smoothing. In our series of experiments we have tried different values for some hyperparameters of the Transformer, in order to see which combination produces the best results.

During training time, the Transformer model is evaluated on the DEV set at each half-epoch, with early stopping with patience equal to 5, in order to

stop the training and save the model. For Decoding we used beam search with a beam width of 5, and during the decoding, the generated "$< unk >$" tokens for unknown words are also replaced by the source token having the highest attention weight. The latter is very important when these tokens correspond to proper nouns that can be directly transposed between languages. Although the recommended batch size for the Transformer is 4096, we used a batch size of 2048 in order to fit into our GPU memory.

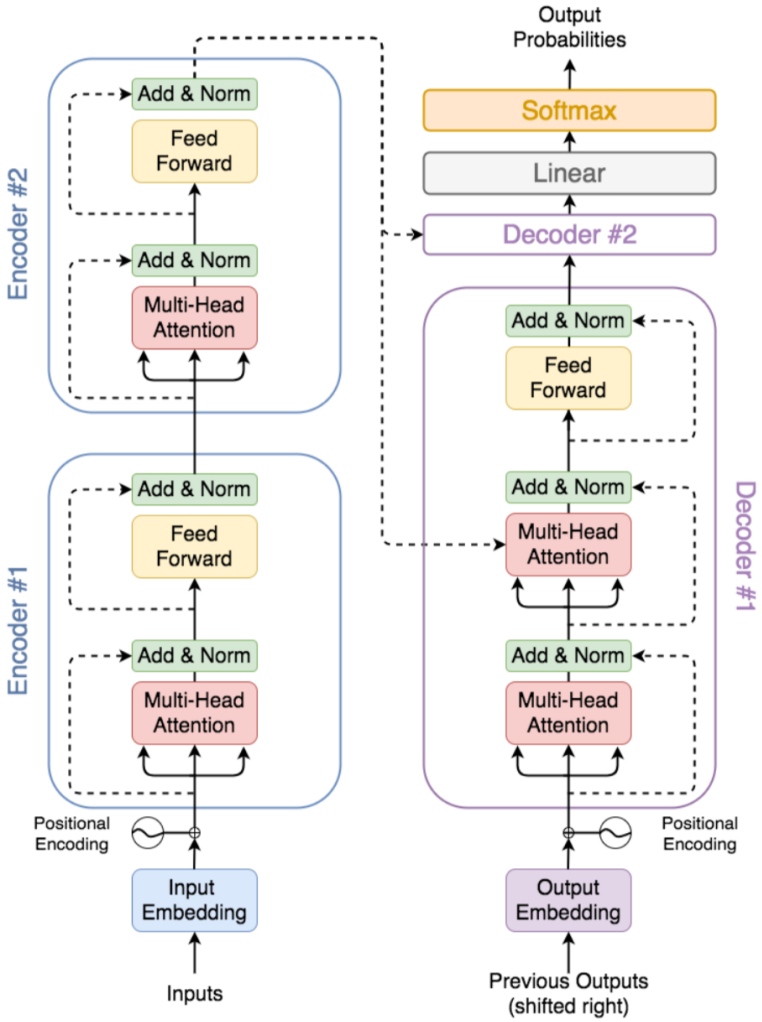

Figure 4.4: The Transformer model, as depicted in [9]

Figures 4.5 and 4.6 show the whole process flow that we follow with the Encoder-Decoder with neural attention model and the Transformer. As we have aforementioned, for the Encoder-Decoder with attention model the data go through OpenPose in order to extract the keypoints of the hands and the upper body of each signer. On the other hand for the Transformer we focus on simplicity and the images go inside the Transformer in their original form.

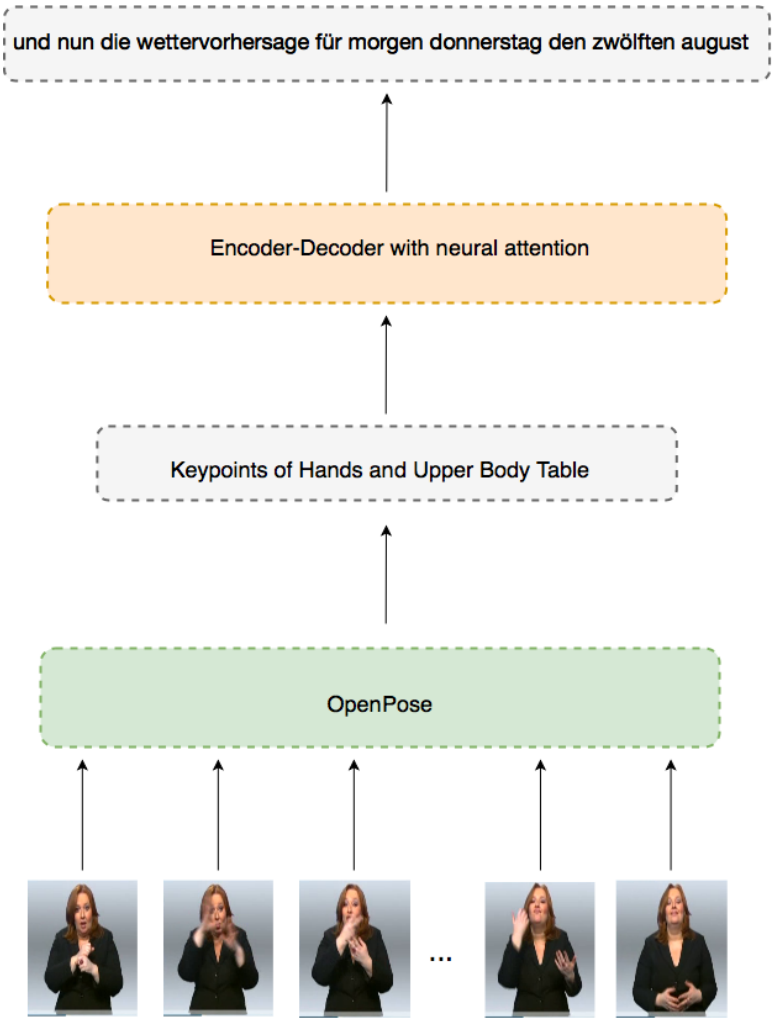

Figure 4.5: The Experimental Setup Flow of the Encoder-Decoder with neural attention model

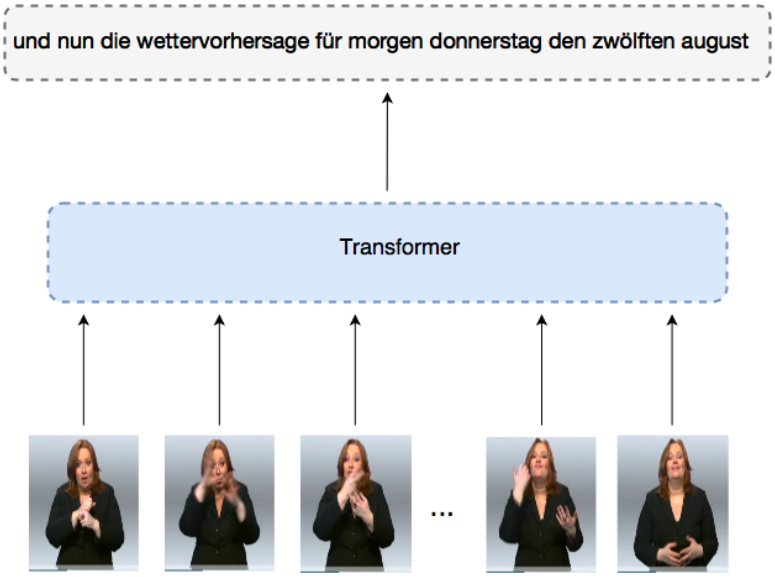

Figure 4.6: The Experimental Setup Flow of the Transformer

4.3.3 Hardware and Software Framework

The experiments ran on 4 workstations in total. A workstation with a i7-7820X CPU (3.60GHz), 64 GB RAM and an NVidia Quadro P5000 (16gb RAM), a workstation with an NVidia Tesla K40c (12gb RAM) and an Xeon E5-2609 CPU (2.5 GHz), and a workstation with an i9-7940X (3.10GHz) CPU with an NVidia RTX 2080Ti (12 gb RAM). Last but not least, due to time constrains I rented a server with 4 NVidia RTX 2080Ti (12 gb RAM) GPUs from Vast.ai a startup which is basically the Airbnb for servers. The Encoder-Decoder with neural attention models were implemented in Tensorflow [49]. The OpenPose was implemented in Caffe [50] and PyTorch [51]. The Transformer model was implemented in OpenNMT [52] and PyTorch.

Chapter 5

Experimental Results and Discussion

Summary

This Section presents the results of our experiments. Firstly we present the results of both the simple Encoder-Decoder with neural attention models and the Transformer. We decided to emphasize more on the Transformer and do more experiments with that because it produces better results than the simple Encoder-Decoder with neural attention models. You can see in the figures of this chapter, experiments of the effect of the different parameters of the Transformer on the BLEU-4 score. Finally, we provide a short discussion and observations of these results, and we try to give more insight on the explanations of them.

5.1 BLEU-4 score

For the evaluation metric we used the BLEU-4 score, a famous metric for NLP tasks [53]. BLEU (bilingual evaluation understudy) is an algorithm for evaluating

the quality of text which has been machine-translated from one natural language to another. BLEU's output is always a number between 0 and 1, and the closer the score is to 1, the better for our translation. It is very simple in the implementation, and it is computed using the couple of ngram modified precision as in the following equation:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5.1}$$

where p_n is the modified precision for n gram, the base of log is the natural base e, w_n is weight between 0 and 1 for $\log p_n$ and $\sum_{n=1}^{N} w_n = 1$, and BP is the brevity penalty to penalize short machine translations.

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r \end{cases} \tag{5.2}$$

where c is the number of unigrams (length) in all the candidate sentences, and r is the best match lengths for each candidate sentence in the corpus. Here the best match length is the closest reference sentence length to the candidate sentences.

5.2 Results

Table 5.1 depicts the final results of our experiments. We observe that the Transformer model produced the best results among the other models. This is no surprise, since neural attention is superior to the Recurrent Neural Network architectures. LSTMs were a clever bypass technique and the attention mechanism showed that normal networks could be replaced by averaging networks influenced by a context vector. Regarding the simple Encoder-Decoder with attention models we tried 2 combinations of 1 or 2 layers for the Encoder, and the use of either LSTM or GRU for the Encoder. We don't observe major differences on their

performance. Next we try to explain better the superiority of the Transformer for the sign language translation task.

Architecture	BLEU-4 score
2-layer LSTM Encoder-Decoder with attention	11.4%
1-layer LSTM Encoder-Decoder with attention	10.1%
1-layer GRU Encoder-Decoder with attention	9%
2-layer GRU Encoder-Decoder with attention	9.7%
2-layer Transformer	**20.8%**

Table 5.1: Final Results of our experiments

Table 5.2 depicts the performance of the Transformer with different number of layers for the Encoder and the Decoder of the model. We note that the Encoder and the Decoder have the same number of layers as we have mentioned before. From Table 5.2 we observe that the Transformer with 2 layers achieves the highest performance, which uses less memory and needs less time for training and inference than the bigger model with 6 layers for instance. It is logical to produce better results with a smaller model, since the dataset is relatively small, in comparison to other Deep Learning datasets that are used in Neural Machine Translation.

Number of Layers	BLEU-4 score
1	19.1%
2	**20.8%**
4	20.1%
6	20.4%

Table 5.2: Results of the Transformer with different number of layers for the Encoder and the Decoder

Figure 5.1 depicts the BLEU-4 score results for various batch sizes. We have

started from batch size 1 and moved to a batch size of 2048. We already know that the optimal batch size for the Transformer is 2048 [54]. We observe in Figure that a small batch size does not produce good results and only when the batch size is equal or bigger than 64 we start seeing higher BLEU-4 scores, which is anticipated [54]. Furthermore, the best performance is achieved when batch size is 2048, which is anticipated as aforementioned.

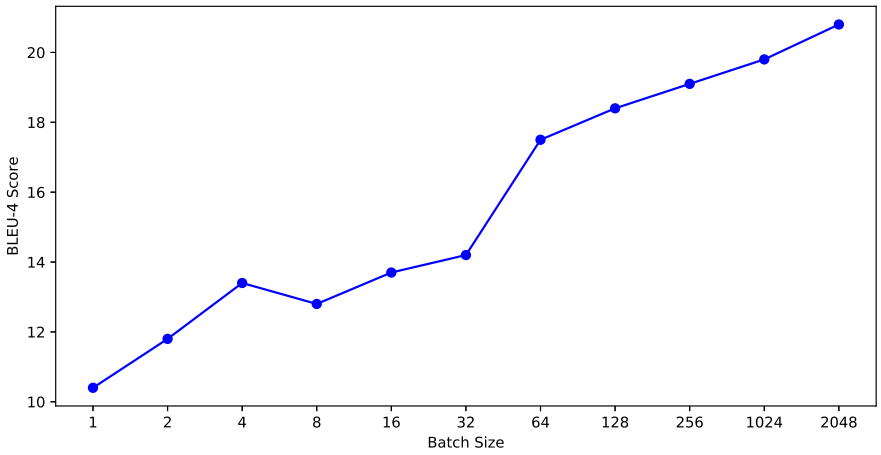

Figure 5.1: The BLEU-4 score results for different batch sizes for the Transformer

Table 5.3 presents some sample translations of the Transformer. We chose some of the best translations that the model produced. We have observed that the model predicts the small sentences with high accuracy and often mistakes small irrelevant words which are pronouns or prepositions. The model also has encounters problems into translating big sentences or sentences that have rare words in them.

5.3 Discussion

RNNs and LSTMs based architectures use mainly sequential processing over time. This means that long-term information has to sequentially travel through all cells before getting to the present processing cell. In this process it can be easily corrupted by being multiplied many time by small numbers smaller than 0. This is the cause of vanishing gradients, and consequently losing valuable information of the data.

On the other hand, neural attention units "look-back", since most of the time we deal with real-time causal data where we know the past and want to affect future decisions. A better way to look into the past is to use attention modules to summarize all past encoded vectors into a context vector C_t. Neural attention has the ability to map long-term relations between the data. The form of dynamics of dependencies in the data is adaptive due to the attention mechanism. The attention mechanism allows the model to see long-term dependencies. This relaxes the strict sequential assumption and allows the model to form any kind of dependencies in the vector. And in a dynamic fashion, because different observations lead to different kinds of dependencies.

Moreover, we note that the difference of the score between the classic Encoder-Decoder model with attention and the Transformer is caused also from the input form of the data. OpenPose may be very robust, but because some images are blurred in the dataset, it does not perform very well with all of the photos. In many images we observed that it does not detect the keypoints with success. This is a major drawback of these particular experiments. Furthermore for the Encoder-Decoder model we passed the images in the model as a table dataframe and not as images, like we did with the Transformer. This seems that it does not work properly and the model does not respond well. Some long-term dependencies vanish because of the input form of the data. For the Transformer we input the

images as they are, without any important preprocessing. In this way, the images maintain their characteristics and properties.

Figure 5.2 presents the convergence of loss during training of the 1-layer LSTM Encoder-Decoder with neural attention model. We observe from this plot that the simple Encoder-Decoder with neural attention model failed to optimize its loss so it did not train well.

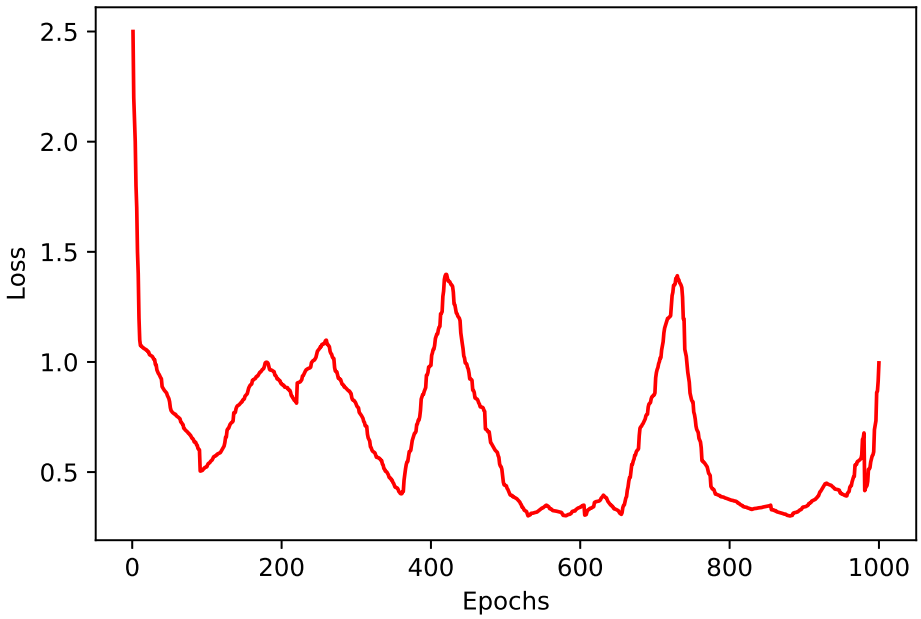

Figure 5.2: The convergence of loss during training of the 1-layer LSTM Encoder-Decoder with neural attention model.

Figure 5.3 presents the convergence of loss during training of the Transformer. We observe from this plot that the Transformer optimized its loss so there is success on its training. The same phenomena were observed in almost all of the Transformer experiments. Moreover in comparison to the Encoder-Decoder with attention's plot, we observe that the Transformer trained better and recognized the patterns in the sign language videos, as we have seen from the final results in

Table 5.1. So Figures 5.3 and 5.2 prove the superiority of the Transformer model.

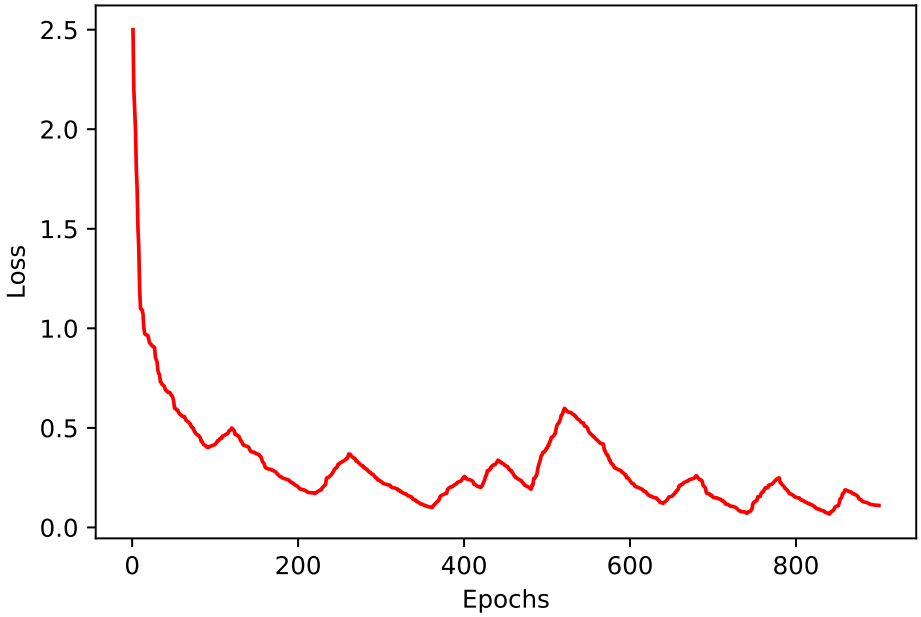

Figure 5.3: The convergence of loss during training of the Transformer.

Test	und nun die wettervorhersage für morgen samstag den zweiten april
Ours	und nun den wettervorhersage vor morgen samstag den zweiten april
Test	es kann noch leicht schneien in den nächsten stunden in sachsen oder auch am alpenrand und in bayern
Ours	es kann noch leicht schneien in den nächsten stunden in sachsen auch am alpenrand und in munchen
Test	am tag elf grad im vogtland und einundzwanzig grad am oberrhein
Ours	am tag elf grad im vogtland und zwanzig grad am vogtland
Test	richtung osten ist es meist sonnig
Ours	im osten bleibt es meist sonnig
Test	am sonntag im norden und an den alpen mal sonne mal wolken und ab und an schauer sonst ist es recht freundlich
Ours	am sonntag im norden und an den alpen mal sonne mal wolken und an schauer sonst sind es freundlich
Test	am donnerstag regen in der nordhälfte in der südhälfte mal sonne mal wolken ähnliches wetter dann auch am freitag
Ours	am donnerstag regen in der nordhälfte in der südhälfte mal sonne mal wolken ähnliches wetter denn am freitag
Test	liegt morgen unter hochdruckeinfluss der die wolken weitgehend vertreibt
Ours	liegt morgen unter drucken der wolken weitgehend vertreibt
Test	ortlich schauer oder gewitter die heftig sein konnen
Ours	ortlich schauer oder gewitter die heftig sein konnen
Test	in den nächsten tagen geht es auf jeden fall winterlich weiter immer wieder mal mit etwas schnee
Ours	in die nächsten tagen geht es auf jeden fall winterlich immer wieder mal mit etwas sonne
Test	spater breiten sich aber nebel oder hochnebelfelder aus
Ours	spater breiten sich oder nebel aber hochnebelfelder
Test	temperaturen nur so um die null grad am kältesten im norden bis minus fünf grad
Ours	temperaturen nur so die null grad am kältesten im norden zum minus kunft grad
Test	im westen ist es freundlich
Ours	im westen ist es freundlich
Test	in der nacht sinken die temperaturen auf vierzehn bis sieben grad
Ours	heute die nacht sinken die temperaturen auf funfzehn bis sieben grad
Test	hnliches wetter dann auch am donnerstag
Ours	hnliches wetter denn auch am der donnerstag

Table 5.3: Translation Examples. On the first line of every block we have the expected output and on the second one we have our output.

Chapter 6

Conclusions

In this thesis we tackled the problem of sign language translation by proposing a state-of-the-art solution for it. We presented an approach of extracting the keypoints of sign language frames with OpenPose, and then channeling them in an Encoder-Decoder with neural attention model. This did not seem to produce good results since OpenPose failed in many frames by either falsely detecting or not detecting the keypoints on the hands at all.

In order to overcome this limitation we implemented a 2 layer Transformer model and the input of the model were the images in their original form. The Transformer consists only of self-attention mechanisms, which are known to be superior in comparison to RNNs. This is also observed in this Thesis, where the proposed Transformer model produced higher results by approximately **9%** BLEU-4 score points. Moreover we conducted various experiments with different hyperparameter values for the Transformer, in order to observe the behavior of the Transformer with different hyperparameter combinations.

Of course, there are still a lot of directions to explore, for tackling the Sign Language Translation problem. Furthermore, there are also other datasets of sign language, which are easier for detection and translation. We note that the RWTH-PHOENIX-Weather 2014T is considered one of the most difficult datasets

for sign language translation.

For future directions there are new approaches that we did not implement in this thesis. An idea is to use an Ensemble of Transformers, which is considered state-of-the-art and a new idea in the Deep Learning field. Except for that, another direction would be to try a more modern system for detection of human body and hand keypoints. But even with these cases we still have the limitation of the small amout of data. The RWTH-PHOENIX-Weather 2014T dataset, is much smaller than those used in standard machine translation tasks.

References

[1] Mattab, "Artificial intelligence, enough of the hype! what is it?," *Hewlett Packard Enterprise Blog*, 2019. vii, 7

[2] N. S. Chauhan, "Introduction to artificial neural networks(ann)," *Medium*, 2019. vii, 8

[3] M. Phi, "Illustrated guide to lstms and grus: A step by step explanation," *Medium*, 2018. vii, 11

[4] S. Kostadinov, "Understanding encoder-decoder sequence to sequence model," *Medium*, 2019. vii, 15

[5] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015. vii, vii, 19, 20, 21

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. vii, 18, 20, 21, 36

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017. vii, vii, vii, 2, 21, 24, 25, 26, 27, 28

[8] J. Forster, C. Schmidt, O. Koller, M. Bellgardt, and H. Ney, "Extensions of the sign language recognition and translation corpus rwth-phoenix-weather.," in *LREC*, pp. 1911–1916, 2014. vii, ix, ix, 29, 30, 31, 32

[9] K. Yin, "Sign language translation with transformers," *arXiv preprint arXiv:2004.00588*, 2020. viii, 4, 38, 39

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. 2

[11] N. Cihan Camgoz, S. Hadfield, O. Koller, H. Ney, and R. Bowden, "Neural sign language translation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7784–7793, 2018. 2

[12] S.-K. Ko, C. J. Kim, H. Jung, and C. Cho, "Neural sign language translation based on human keypoint estimation," *Applied Sciences*, vol. 9, no. 13, p. 2683, 2019. 2

[13] C. Dong, M. C. Leu, and Z. Yin, "American sign language alphabet recognition using microsoft kinect," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 44–52, 2015. 2

[14] S. Gattupalli, A. Ghaderi, and V. Athitsos, "Evaluation of deep learning based pose estimation for sign language recognition," in *Proceedings of the 9th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, p. 12, ACM, 2016. 3

[15] T. Kim and S. Kim, "Sign language translation system using latent feature values of sign language images," in *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 228–233, IEEE, 2016. 3

[16] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *Motion-Based Recognition*, pp. 227–243, Springer, 1997. 3

[17] P. Buehler, A. Zisserman, and M. Everingham, "Learning sign language by watching tv (using weakly aligned subtitles)," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2961–2968, IEEE, 2009. 3

[18] H. Cooper and R. Bowden, "Learning signs from subtitles: A weakly supervised approach to sign language recognition," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2568–2574, IEEE, 2009. 3

[19] T. Pfister, J. Charles, and A. Zisserman, "Large-scale learning of sign language by watching tv (using co-occurrences).," in *BMVC*, 2013. 3

[20] J. Forster, C. Schmidt, T. Hoyoux, O. Koller, U. Zelle, J. H. Piater, and H. Ney, "Rwth-phoenix-weather: A large vocabulary sign language recognition and translation corpus.," in *LREC*, pp. 3785–3789, 2012. 3

[21] J. Forster, C. Schmidt, O. Koller, M. Bellgardt, and H. Ney, "Extensions of the sign language recognition and translation corpus rwth-phoenix-weather.," in *LREC*, pp. 1911–1916, 2014. 3

[22] O. Koller, J. Forster, and H. Ney, "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers," *Computer Vision and Image Understanding*, vol. 141, pp. 108–125, 2015. 3

[23] P. Kishore, A. Sastry, and A. Kartheek, "Visual-verbal machine interpreter for sign language recognition under versatile video backgrounds," in *2014 First International Conference on Networks & Soft Computing (ICNSC2014)*, pp. 135–140, IEEE, 2014. 3

[24] O. Koller, S. Zargaran, and H. Ney, "Re-sign: Re-aligned end-to-end sequence modelling with deep recurrent cnn-hmms," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4297–4305, 2017. 4

[25] M. Oberweger, P. Wohlhart, and V. Lepetit, "Hands deep in deep learning for hand pose estimation," *arXiv preprint arXiv:1502.06807*, 2015. 4

[26] S.-K. Ko, J. G. Son, and H. Jung, "Sign language recognition with recurrent neural network using human keypoint detection," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, pp. 326–328, ACM, 2018. 4

[27] N. C. Camgoz, O. Koller, S. Hadfield, and R. Bowden, "Sign language transformers: Joint end-to-end sign language recognition and translation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10023–10033, 2020. 4

[28] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958. 7

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 10

[30] M. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proc. of the Eighth Annual Conference of the Cognitive Science Society (Erlbaum, Hillsdale, NJ), 1986*, 1986. 10

[31] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 263–269, 1989. 10

[32] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural computation*, vol. 1, no. 3, pp. 372–381, 1989. 10

[33] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014. 14

[34] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition.," in *Interspeech*, pp. 939–943, 2017. 14

[35] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence-video to text," in *Proceedings of the IEEE international conference on computer vision*, pp. 4534–4542, 2015. 14

[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. 17

[37] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, *et al.*, "Alphastar: Mastering the real-time strategy game starcraft ii," *DeepMind blog*, p. 2, 2019. 21

[38] K. Fang, A. Toshev, L. Fei-Fei, and S. Savarese, "Scene memory transformer for embodied agents in long-horizon tasks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 538–547, 2019. 22

[39] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *arXiv preprint arXiv:1812.08008*, 2018. 32

[40] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7291–7299, 2017. 32

[41] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4724–4732, 2016. 32

[42] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *CVPR*, 2017. 32

[43] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic press, 2019. 33

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 33

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015. 33

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 38

[47] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016. 38

[48] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012. 38

[49] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016. 41

[50] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, 2014. 41

[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, pp. 8026–8037, 2019. 41

[52] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "Opennmt: Open-source toolkit for neural machine translation," *arXiv preprint arXiv:1701.02810*, 2017. 41

[53] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002. 42

[54] M. Popel and O. Bojar, "Training tips for the transformer model," *The Prague Bulletin of Mathematical Linguistics*, vol. 110, no. 1, pp. 43–70, 2018. 45

[55] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.