

Autoformalization of Mathematics

Markus Wich

University Erlangen-Nuremberg, Germany

Report for Seminar "Wissensrepräsentation und -verarbeitung"

Abstract—Autoformalization describes the process of translating informal mathematics into a formal mathematical language. Recently there has been a novel approach of autoformalization of mathematics by using neural machine translation. State-of-the-art natural language translation models were used to translate mathematics written in LaTeX to statements of the Mizar language. In this report I want to give an overview over this new approach as well as explain the different machine learning technologies used in more detail. This work includes a brief overview of Mizar as well as a short introduction to neural networks. The different neural machine learning models used during the experiments get explained in detail. The datasets used are described as well as a data augmentation mechanism. The results of the experiments are evaluated and judged.

1. Introduction

For many years automated/interactive theorem proofing has been a dream of mathematicians and AI researchers alike and today there exist many tools, for example proof assistants, that support this goal. But all of these tools require the formulas used during the proofing process to be available to the machine in a computer understandable format, which means they have to be written in a formal language.

This becomes a particularly big problem when dealing with mathematics, because the vast majority of mathematicians publish their work as LaTeX PDF files, which are written in an informal language and thus not readable by computers. In the past, there have been efforts to solve this issue by translating the mathematical corpus to a formal language manually. This process is called formalization. One of the most notable examples of this work is the Mizar Math Library, which contains most of undergraduate mathematics. But formalization by hand requires domain experts for the mathematical topic to be formalized and the target language. That is a very small group of people and as a result the speed of formalization is slower than the increase of mathematical papers released.

A possible solution to increase the speed of formalization is to automate the process, which is then called autoformalization. The consensus in the scientific community was that autoformalization is a very hard or even impossible task, but recently a new approach [1] [2] was developed by uti-

lizing recent advancements in neural machine translation. The idea behind this approach is to treat autoformalization as a language translation problem, where the informal language is the source language and the formal language the target language. This allows the use of already existing machine translation tools. All the best translation models right now are based on neural machine translation (NMT) [3]. NMT means translating text or speech using large neural networks.

Kaliszyk et al. [2] used three state-of-the-art NMT models to test their approach and compare them to each other. The first one is the seq2seq model from Luong et al. [4], which is a supervised model and therefore needs an aligned dataset. The other two are the unsupervised NMT model by Lample et al. [5] and the cross-lingual model from Lample and Conneau [6].

Additionally they prepared three different informal-to-formal datasets to evaluate the models. An aligned synthetic LaTeX - Mizar dataset to train and test the supervised model and two unaligned ProofWiki - Mizar datasets used to run additional evaluations for the unsupervised models. They also created a data augmentation mechanism in order to increase the quality of a dataset. For this they created an optimization feedback-loop to create new entries for a dataset and therefore increase the model's performance. The researchers also created a fourth Mizar - TPTP dataset to test this mechanism.

In this work I want to explain the general topic of autoformalization, the experiments of autoformalization conducted by Kaliszyk et al. [2] and the technologies, particularly the different neural networks, that were used in their approach. Section 2 gives a short overview over the history of autoformalization. Section 3 explains the idea behind the neural machine translation approach. A short introduction to Mizar follows in section 4. Section 5 is a very brief introduction to neural networks and different terminology used in machine learning and machine translation. Improvements and advancements to the classical neural network structure that are used for the NMT models are explained here, too. Section 6 gives an in depth look at the three neural networks used during the experiments. The datasets that were created and used to train and test the models are introduced in section 7 with the feed-back loop used for data augmentation being further explained in section 8. Then the researchers' evaluation gets presented in section 9. And at the end is a conclusion and assessment

of the experiments judging the results as well as sum up the positives and negatives of this approach in its current state.

2. Previous Work on Autoformalization

The idea of autoformalization is quite old. The first one who hinted at the possibility of autoformalization of mathematics was Wang in 1954 [7]. But no concrete approaches were made at that time. John McCarthy was the first one to formulate an idea for a system capable of autoformalization in 1962 [8]. Donald Simon made another attempt for an autoformalization system that was designed to handle elementary number theory proofs that were written in english in 1990 [9]. In 1994 the QED group released the QED manifesto [10] in which the anonymous authors published their proposal for a database of all mathematics strictly formalized, but nothing came out of it and the project was dissolved shortly after. Kaliszyk et al. cite the vision layed out in the QED manifesto as the goal of their current work. There also exist similar projects with more modest goals. For example the Mizar Mathematical Library [11] has successfully formalized most of undergraduate mathematics. In 2004 Zinn [12] parsed text books with the use of domain discourse theory, but his work is often used to discourage people from similar endeavours. The consensus in the scientific community was that autoformalization is a very hard or even impossible task. In the last couple of years, there has been renewed interest in the topic and the team around Kaliszyk et al. made multiple attempts at creating an autoformalization system and they are currently spearheading the research in this area. Kaliszyk et al. were able to parse informal latex sentences with probabilistic context free grammars into HOL light [13] [14] and this was furthermore used to parse Mizar-like formulas [15]. They also conducted first experiments with using neural machine translation, but only a supervised model was used and they focused more on testing different hyperparameter configurations [1].

3. Autoformalization via Neural Machine Translation

The reasoning behind the claim that autoformalization is a potentially impossible task, is that an autoformalization tool would need to be able to emulate the human thinking process that is used when translating informal mathematics to a formal language. Some people hinted that this would require the program to be capable of advanced natural language understanding combined with mathematical reasoning and some argue that an AI achieving result comparable to humans in this domain would take decades [16]. Kaliszyk et al. [2] however argue that the problem can be simplified by treating autoformalization as a language translation problem. Their idea is to give an informal sentence to the translation tool and it translates it to a formal sentence. The generated formal sentence can

then be further formalized and given to a proof assistant. This would allow the formalization of a large number of mathematics and would significantly increase the capabilities of proof assistants. Treating autoformalization as a language translation problem allows the use of already existing machine translation tools that have made major advancements in the last couple of years. The researchers tested three different models.

The seq2seq model from Luong et al. [4] is a well established model for machine translation. It has an encoder-decoder structure consisting of two RNNs that can be trained with an aligned bilingual dataset.

To circumvent the requirement of needing aligned data the researchers tested the unsupervised translation model (UNMT) based on Lample et al. [5]. It can be trained with two unaligned monolingual corpora by using backtranslation.

The cross-lingual pretraining model (XLM) by Lample and Conneau [6] was also used in the experiments as an improved version of the UNMT model. While in the UNMT model the vector representation of words is obtained by word2vec [17], XLM uses the recently invented technique of pretraining to create an initial vector representation that is than further improved by training the model unsupervised.

Because this is a novel idea, no usable datasets containing informal and formal mathematics existed prior to the researchers work. Therefore, they had to create their own. As the informal language LaTeX was used and for the formal language the Mizar language was chosen. Three LaTeX - Mizar datasets with various sizes and contents were created. One of the datasets is fully aligned, while the other two are unaligned and only used for training and testing the unsupervised models.

In order to experiment with data augmentation, the researchers created a feedback loop including an elaborator for TPTP formulas to generate new sentences and thus improve the models' performance. To train and test this mechanism another dataset containing aligned Mizar statements and TPTP formulas was created.

4. Mizar Language

The Mizar project [11] was started around 1973 with the goal to create a digital version of a mathematical vernacular. The result of this attempt is the Mizar language. The main design principle behind the language was to be both easily readable for mathematicians and sufficiently expressive to allow computers to process and verify mathematics written in the language. A script written in the Mizar language is called a Mizar article. The Mizar language is implemented in the form of the Mizar system, which allows users to write Mizar articles and also includes a verifier to check them. After the Mizar language and the Mizar system were created the main activity of the Mizar project shifted towards the creation of a database for formal mathematics. This effort resulted in the Mizar Mathematical Library (MML) which contains

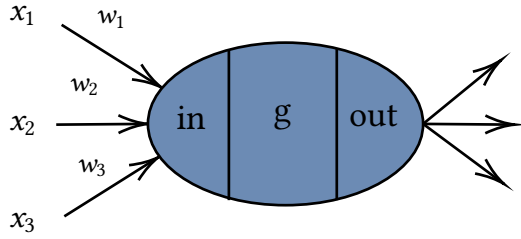


Figure 1: McCulloch-Pitts unit

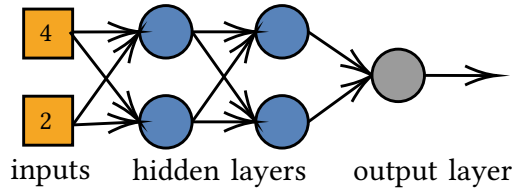


Figure 2: Architecture of a simple feedforward neural network

9400 definitions of mathematical concepts and more than 49000 theorems.

5. Neural Networks

Artificial neural networks, or for short neural networks, are inspired by the biological neural network of animal brains. An animal brain has brain cells called neurons and synapses connecting those neurons. Signals are transmitted via electrical potential.

To model this structure, artificial neural networks have computational units also called neurons and edges connecting the neurons together. The output of one unit gets forwarded along these edges to the following neuron. The neurons and edges make up a directed graph, which has some input and some output units.

The simplest model for a neuron is the McCulloch-Pitts unit and was created in 1943 [18]. Their structure is shown in figure 1. These units first multiply all inputs with their respective weights and then sum the products up. This sum is then used as the input to the neurons' activation function. Common activation functions are threshold functions or the logistic function. The result of the activation function is the output of the neuron and gets sent onwards.

$$out = g\left(\sum_{i=1}^N x_i \cdot w_i\right)$$

A feed-forward network as shown in figure 2 is the simplest form of a neural network. They contain no cycles and the input is only forwarded through the network. These networks are organized in layers with neurons only having connections to neurons of the next layer. The first layer is called input layer and the last layer called output layer. The layers in between are called hidden layers. When we have multiple hidden layers, we speak

of deep learning. The units are called input, output or hidden units correspondingly. The input units only produce a value and forward it, while the output units still do the computation described above, but might use a different activation function in order to produce a particular output format, for example perceptrons units produce either 0 or 1 as output.

The capabilities of a neural network are influenced by its number of layers as well as the number of neurons per layer. Usually the performance increases and the network can handle more complex tasks the bigger the network is. But with increasing size also comes an increase in computational resources needed and a decrease in transparency. The parameter of a model, e.g. number of layers and number of cells per layer, are called hyperparameters. The output of a model depends, apart from the input, on the values of the weights. So in order to better model the relationship between the input and output data, the weights are changed. The process of modifying the weights is called training and the specific algorithm used is called backpropagation. First input data is given to the neural network, which computes an output. This output is then compared to the wanted value and a loss function, for example the squared error, computes a loss value. With this loss value gradient descent is used to calculate new weights. Then a new input is given to the network and the output is now computed by using the new improved weights. This process is then repeated until all training data is processed or an abortion criterion is reached.

During the training process a dataset is required that maps inputs to correct outputs. This is called an aligned dataset. There are two problems one needs to be aware of when training neural networks. The first one is underfitting. A model underfitting means it is too simple to describe the underlying relationship of the data. The second problem is called overfitting and occurs when the model is too focused on the training data and as a result doesn't generalize well for previously unseen data. To spot these problems, the dataset gets divided into three parts. The training set, the validation set and the testing set. The training set is used for training and during that process the testing set is used to evaluate the model. The final model is then evaluated with the validation set, which was to this point unused and thus allows the model's performance to be checked on previously unseen data. If the model has a high testing and validation error it underfits. If it has a small testing and a high validation error it overfits and if it has a low testing and a slightly higher validation error it is in the optimal region.

While the theoretical foundations of neural networks were already created in the 1980s, it took a long time for neural network based applications to achieve competitive results. The reasons for this were the lack of computational power for training the models and the lack of large datasets. When in 2012 AlexNet [19] was released it was an important milestone, because it was capable of using the GPU during training and was trained using the sizeable ImageNet dataset. This sparked other people to use GPUs for training

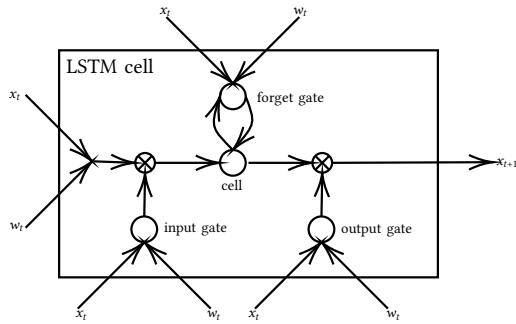


Figure 3: Structure of a Long Short-Term Memory cell

and accelerated the advancements in machine learning.

5.1. Recurrent Neural Networks

When using neural networks for machine translation usually recurrent neural networks (RNN) are chosen. The defining characteristic of a recurrent neural network is that it can have connections to nodes of a previous layer or connections between nodes of the same layer, unlike a feedforward NN where nodes can only have connections to subsequent layers. The big advantage of this type of NN is that it can keep information of previous inputs, which makes it particularly useful when dealing with sequences as inputs.

5.2. Long Short-Term Memory

For machine translation very large neural networks are needed. Usually multiple layers with over 1000 nodes each. When training such large networks with backpropagation, two problems arise, when using traditional cells. During backpropagation an error based on the output of the network and the target value gets calculated. Then the error gets propagated backwards through the network and the weights are adjusted accordingly. During this process the error gets repeatedly multiplied by the network's weights. When those weights have values between 0 and 1 the propagated error becomes smaller and smaller until it is zero. The rest of the weights then don't get modified and only a part of the network is effectively trained. This phenomenon is called vanishing gradients.

The opposite happens, when the weights are bigger than 1. In this case the backpropagated error becomes exponentially high, which makes learning unstable. This is called exploding gradients.

To fix these two issues, the seq2seq model uses a technology called long short-term memory (LSTM) cells first discovered by Hochreiter and Schmidhuber [20]. LSTM memory cells use three gates in addition to the cell to regulate how much new information should be passed to cell, how much information should be passed through and how long information should stay in the cell as seen in figure 3.

Input gate: This gate decides how much a new value should

influence the cell.

Forget gate: Decides how long a value should stay in the cell or how fast it should be forgotten.

Output gate: Decides how much the value of a cell should be passed on and influence the next cell.

6. NMT Models

The researchers tested three different state-of-the-art NMT models for their experiments. One of them, the seq2seq model, is a supervised model, meaning that it needs an aligned corpus, where the sentences from the source language are combined with the corresponding sentence from the target language, for training. The other two are unsupervised models, that only need unaligned monolingual corpora from each language. In the following sections the technologies behind each of these models are explained.

6.1. seq2seq Model

The first model tested is the seq2seq model developed by Luong et al. [4]. It is implemented using the Tensorflow [21] framework. It is a supervised model, which means it needs a dataset with sentence pairs from the source and target language that are aligned with each other. The seq2seq model is based on the sequence-to-sequence structure developed by Sutskever et al. [22] that uses an encoder-decoder-architecture as seen in figure 4. Both the encoder and the decoder are multi-layer recurrent neural networks with LSTM cells.

During the training process of the model, a sentence in the source language is given to the encoder sequentially word for word with the words being represented by a word vector. After the sentence is processed by the encoder, it generates a numerical vector called the "thought" vector or context vector as an output. This vector represents the meaning of the source sentence in a numerical form. The context vector is then given to the decoder and it translates the content of the vector to the target language.

The actual architecture of the seq2seq model is shown in figure 5. Note that this figure shows a rolled out representation of the network. This means that the network is displayed over multiple timesteps, which is very useful when showing the processing of sequences. The blue and red boxes stacked on top of each other represent the network in one timestep with the following timestep being towards the right side. The first layer of the encoder is an embedding layer that transforms the word input into a vector of fixed size usable by the main neural network, because the initial word representation is usually not easily usable for a neural network, e.g. a one-hot vector over the entire vocabulary. After the first word is processed by the embedding layer, the encoder network uses its layers to generate the context vector which is then given to the decoder. In the seq2seq model this isn't actually a vector, but the hidden state of the encoder is passed to the decoder after the last input word is processed by the encoder. The

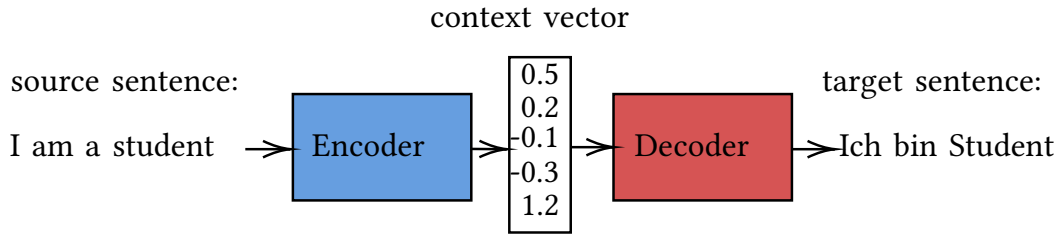


Figure 4: Sequence-to-sequence NMT with an encoder-decoder-architecture

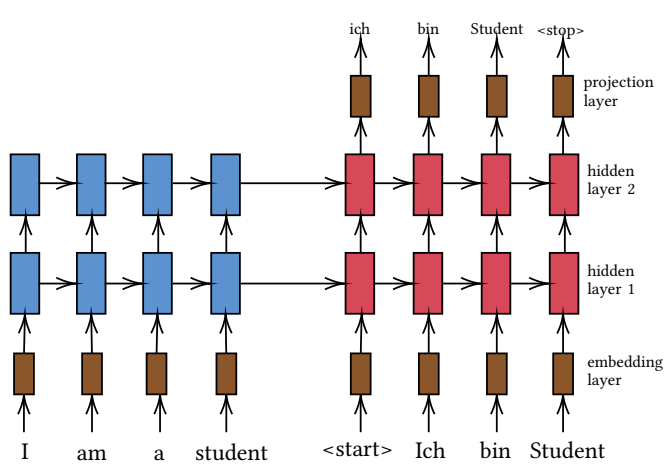


Figure 5: Training of the seq2seq NMT model

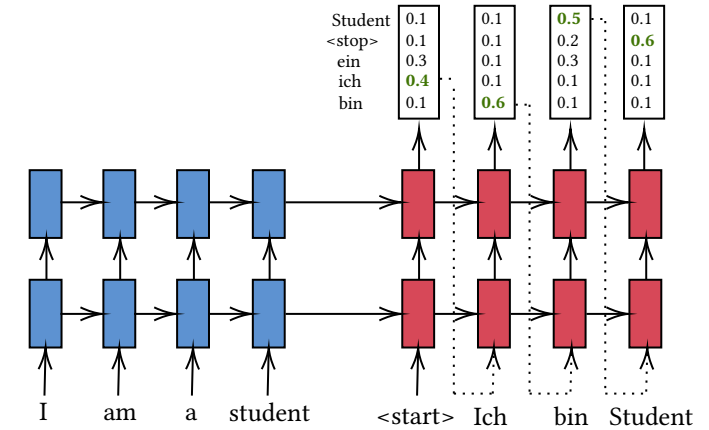


Figure 6: Inference process of the seq2seq NMT model (embedding and projection layers are omitted for clarity)

decoder also has an embedding layer to transform the words from the target language into usable vectors. The output of the decoder neural networks gets processed by a projection layer, which projects the output vectors to logit vectors that are the size of the vocabulary of the target language. Each entry in such a logit vector represents the likelihood that the corresponding word to the entry is the next word in the sentence.

Training of the model is performed by giving a sentence of the source language to the encoder word for word. The RNN of the encoder produces the context that is then given to the decoder like described above. The decoding process starts by sending a `<start>` token as an input to the decoder. The decoder then processes this token with its RNN and produces the output vector containing predictions for the first word of the target sentence. During the next timestep the first word from the correct target sentence is given to the decoder as input and the decoder tries to predict the second word in the sentence. This process is repeated until the entire correct target sentence was given to the decoder. The loss value of the generated output sentence is then calculated and backpropagation is performed to improve the models weights.

An example of output during the training process for this model can be seen in the appendix table A1. Please note, that this example isn't from the experiments described here, but from an earlier work [1] and more details can be found there.

After the model is trained, it can be used to translate previously unseen sentences from the source language. This process is called inference and an example is shown in figure 6. The encoder works like during training and takes a source sentence sequentially and calculates a context vector that is given to the decoder. The decoder starts again with a `<start>` token and produces the probabilities for the first word of the target sentence. With these probabilities greedy search or beam search can be used to pick the next word(s). In figure 6 greedy search is shown and only the word with the highest probability gets picked. When using beam search with width n multiple different words are picked and n output sentences are generated. Beam search is used in section 8 for the feedback loop. The word is then given to the decoder in the next timestep as input and the decoder tries to predict the next word again. This process is repeated until a `<stop>` marker is produced.

6.1.1. Attention. A problem with compressing the entire meaning of the source sentence into a single context vector is that the vector does not contain any information about the position of the meaning in the sentence. This becomes a problem when translating long sentences, because the decoder then does not know if the meaning from the context vector comes from the beginning or the end of the sentence.

A solution for this problem is the attention mechanism which was first introduced to NMT models by Luong et

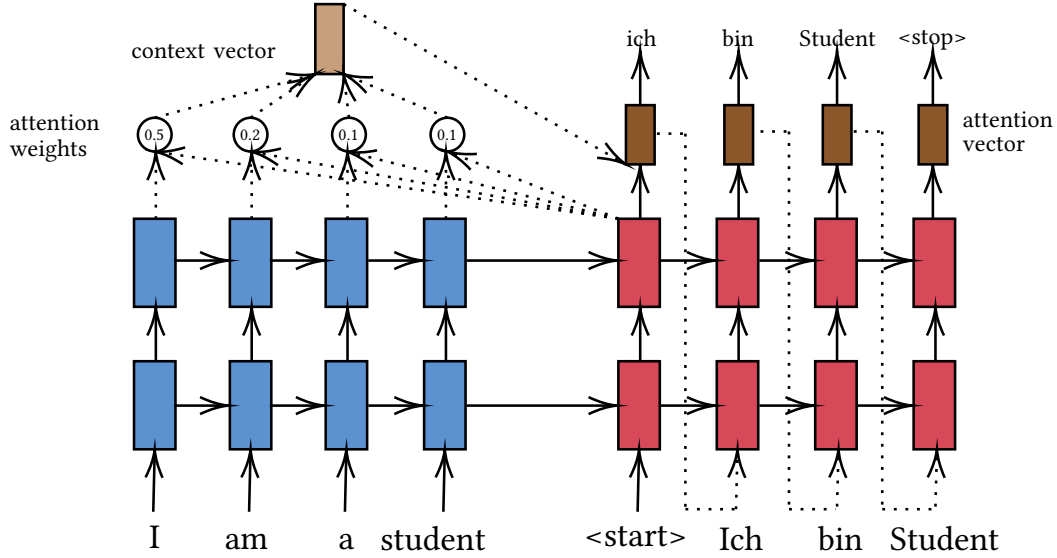


Figure 7: Attention mechanism connecting the encoder to the decoder (embedding and projection layers are omitted for clarity)

al. [23]. This mechanism is a side network connecting the encoder to the decoder. It has a new vector called context vector that records the hidden state of the network after each input word. The hidden states have weights associated with them and the states that are closer to the position of the word have bigger weights. In figure 7 you can see the calculation of the attention mechanism at the first step of the decoding process. Because the decoder wants to predict the first word, the attention weights are higher at the beginning of the sentence. The context vector is then combined with the output of the decoder into the attention vector, from which the predictions for words are made. This gives the decoder additional positional information.

6.2. UNMT Model

The second model the researchers included in their experiments is the unsupervised NMT model from Lample et al. [5], which is implemented using PyTorch [24]. The problem with using supervised models for machine translation is that they need a very large corpus of aligned data, usually well over a million entries long. Producing such a corpus with high-quality data is a very tedious, time-consuming and expensive task. This becomes a particularly big problem when dealing with unique language pairs like it is necessary to do in this autoformalization approach. Doing machine translation as an unsupervised task tries to remove the need for creating such a corpus. While doing machine translation completely unsupervised is an ill-posed problem, the unsupervised task can be turned into a series of supervised tasks.

The model architecture now consists of one shared encoder for both languages and two decoders, one for each language, as shown in figure 8. Before training the model, the two monolingual corpora are combined and turned into

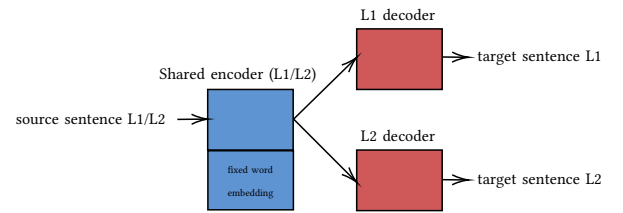


Figure 8: Architecture of the UNMT model

a bilingual word embedding with the word2vec [17] tool fastText [25]. Word2vec generates vector representations for words by processing unstructured text. The vector representation is a very high dimensional vector space with each unique word having a corresponding vector. These vectors are positioned in such a way that words that appear in the same context are closer together. This word embedding is then given to the encoder and stays fixed during the entire training and evaluation process.

Before training a denoising step is performed to initialize the neural networks' parameter values. In this step, sentences from both corpora are corrupted by changing and removing words randomly. These corrupted corpora are aligned with their non-corrupted counterparts. The corrupted corpus is then used as the input language and non-corrupted corpus as the target language. So when the corrupted sentences is given to the model, it tries to translate it to the uncorrupted sentence. One of those aligned corpora pairs is used to initialize the parameters of the shared encoder and the L1 decoder, while the other pair is used to improve the parameters of the encoder and initialize the parameters of the L2 decoder. This denoising process is shown in the figures 9a and 9b.

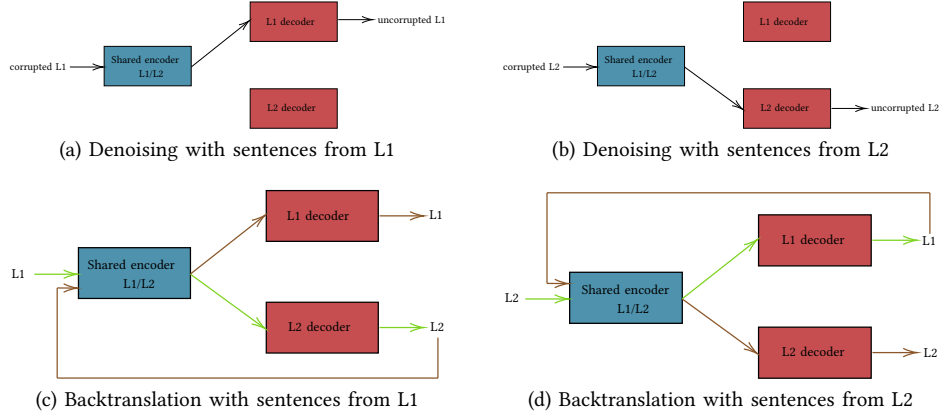


Figure 9: Denoising and backtranslation steps for UNMT and XLM

After denoising the parameters are further improved by training with backtranslation [26]. When training with this technique, a single sentence from a language is taken and given to the encoder to generate a context vector. This vector is then given to the decoder of the other language, which produces a sentence from that language. This newly generated sentence is then given to the encoder again, a context vector is generated again and is given to the decoder of the first language. The result of this process is a sentence in L1. This produces a sequence of $L1 \rightarrow L2 \rightarrow L1$ and $L2 \rightarrow L1 \rightarrow L2$ data respectively. The process flow through the model is shown in the figures 9c and 9d. The green arrows mark the flow during the first pass-through and the brown arrow during the second one. The generated sentence can now be compared to the input sentence of the same language and a loss value can be computed to optimize the network and therefore this is a supervised problem now. The same process can be done with input sentences from L2. The model is trained by repeatedly executing this process, while alternating the language of the input sentence.

When using this model for inference, the sentence from the source language is given to the encoder and only the decoder of the target language is used to translate the sentence to that language.

While this model can use recurrent neural networks as its encoder and decoders, the researchers used transformers as they allow parallel computation.

6.2.1. Transformer. One problem with recurrent neural networks is that as they compute the hidden states they always need the previous state and thus their computational steps are inherently sequentially. This characteristic prevents training examples to be parallelized, which, considering the amount of training data that needs to be processed for a machine translation model, significantly lengthens the training time. Transformers proposed by Vaswani et al. [27] fix this problem by removing the recurrency and purely rely on an attention mechanism to figure out the connection between input and output words. The resulting

architecture can therefore be parallelized and can achieve similar translation quality with significantly lower training times.

The structure of a transformer is shown in figure 10. A transformer consists of an encoding and a decoding component as well, but they are split up into multiple layers. A layer in the encoder consists of two sublayers, a normal feedforward network and a self-attention layer. The input first goes through the self-attention layer, which gives the feedforward network information about the other words from the input sentence, when encoding a single word. The feedforward network is applied to each word of the input independently and therefore this step can be parallelized. The decoder layers have the self-attention and feedforward sublayers as well. Additionally, they also have an encoder-decoder attention component that works similar in function to the attention mechanism of the seq2seq model described above.

All vector calculations are actually done by combining the vectors for all words of a sentence into a matrix and then using that matrix for calculation, but for easier understanding only the vector operations are explained here. In the self-attention part three vectors for each of the encoder's input words are computed. They are called query, key and value vector and are created by multiplying an input word vector x_i with three different matrices that were created during training.

$$q_i = x_i \cdot W^Q$$

$$k_i = x_i \cdot W^K$$

$$v_i = x_i \cdot W^V$$

The next step is to calculate scores. A score shows how much other parts of the input sentence influence the encoding of a word. The scores are the dot product of the query vector of the word the attention is computed for and the key vector of the word that gets scored. So for example

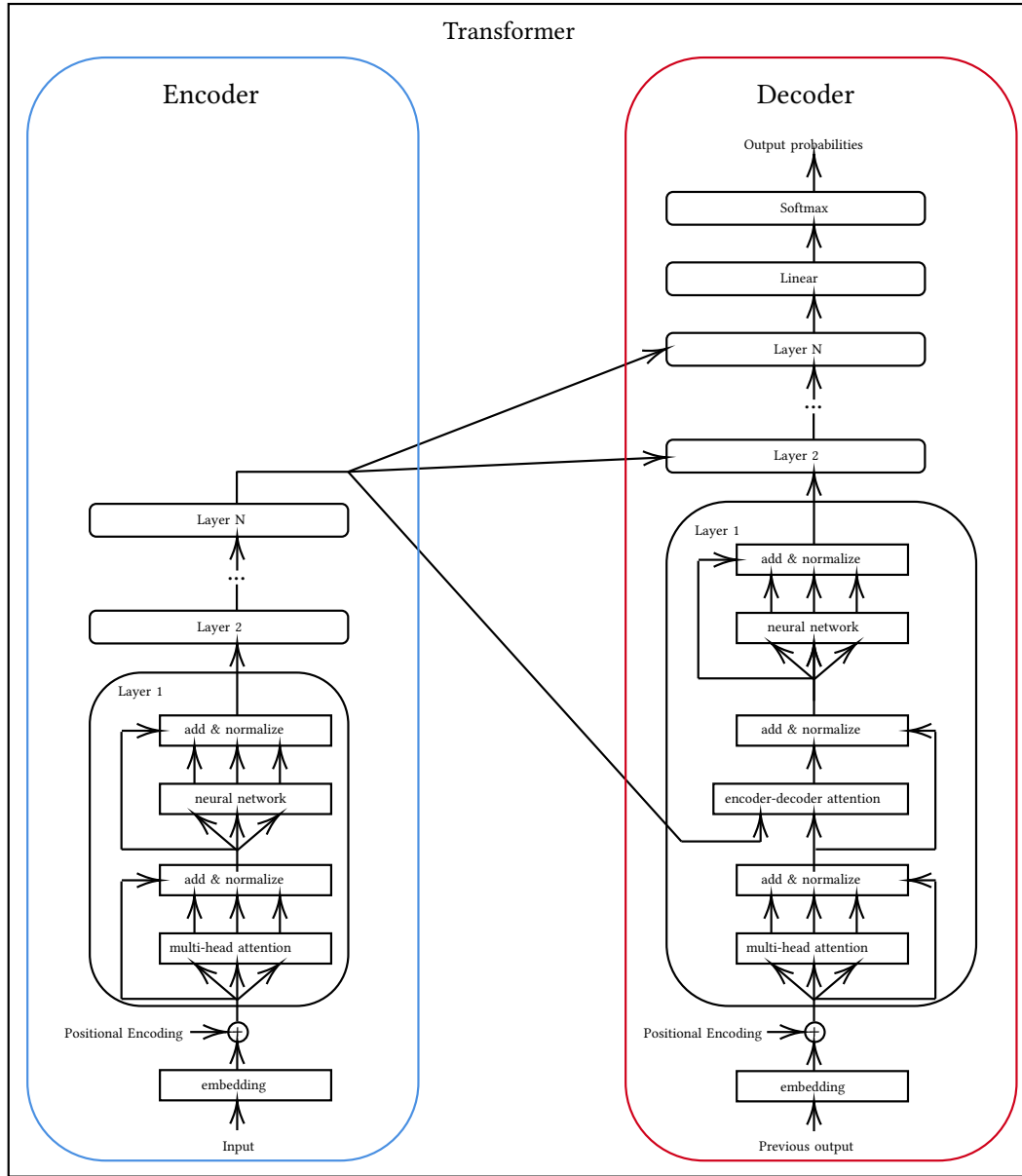


Figure 10: Architecture of a transformer

the score for the second word when calculating attention for the first word is:

$$s_2 = q_1 \cdot k_2$$

In the next step the scores get divided by the square root of the dimension of the key vectors, because it results in a higher stability, and normalized by a softmax function.

$$s'_i = \text{softmax}(s_i / \sqrt{|k|})$$

In the end the normalized scores are multiplied with their respective value vectors and the products are summed up,

which is the output z of the self-attention layer for a single word.

$$z_i = \sum_{j=1}^N s'_j \cdot v_j$$

This way outputs for all words in the input sentence are computed. These outputs are then given to their individual feedforward networks to be processed in parallel. Transformers further improve this self-attention mechanism with multi-headed attention. A multi-headed attention cell does the computation described above N times and produces N output vectors for each input word. These different vectors are then combined by multiplying them with a weight matrix that was created during training

with the product being given to the feedforward network. The output vector of the network is then used as the input vector for the next layers multi-headed self-attention sublayer.

The input of the encoder is combined with a positional encoding to give the model an understanding of the order of words. To the output of the self-attention calculation, the input is added and the sum is normalized before it gets sent to the feedforward network. The same is done with the output of the network before it gets sent onwards. The same normalization step is performed for the sublayers of the decoder as well.

After the encoding process is finished the output vectors, one for each word, are transformed into attention vectors, which are then given to the decoders encoder-decoder attention layers. The decoder works similar to the encoder with the difference that the multi-headed self-attention layer only takes previous words of the output sentence into consideration, because the following words are not yet generated. The encoder-decoder attention layer works like the multi-headed attention layer, but takes its keys and value vectors from the output of the encoder. The query vectors are still taken from the layer below. The output of the last decoder layer gets sent to a linear layer followed by a softmax layer. The linear layer consists of a simple neural network that projects its input to a much larger array, that has the size of the vocabulary with each vector entry representing the score for a word of the target language. The softmax layer normalizes the scores and turns them into probabilities. So the resulting vector contains the predicted probabilities for the next word in the target sentence.

6.3. Cross-lingual Pretrained Model

The third model the researchers tested is the cross-lingual pretrained model (XLM) created by Lample and Conneau [6] and is implemented in PyTorch [24]. It supports both supervised and unsupervised learning, but only the unsupervised capabilities were used in the experiments. This model has the same architecture as the UNMT model with one encoder and two decoders. But in contrast to the UNMT model, where the vector representation of word tokens is fixed, here a technique called pretraining is used to obtain an initial vector representation at both sentence and word level.

This initial representation can then be used to fine-tune the model through unsupervised learning.

XLM uses a technique called Masked Language Model (MLM), which was created for and is adapted from BERT [28]. BERT stands for Bidirectional Encoder Representations from Transformers and is a language representation model that pretrains deep bidirectional representations by processing unlabeled text. In MLM a sentence from the dataset is taken and corrupted, either by substituting tokens with a random token or with a mask token. The corruption process works by first selecting words with a 15% chance each. If a word was selected, it has an 80% chance

to be changed to a mask token, a 10% chance to be changed to a random token and a 10% chance to stay unchanged. The sentence is then given to a transformer that tries to predict the correct words for the mask tokens and recreate the original sentence. When using the cross-lingual model for unsupervised learning, the encoder and decoders are pretrained with this technique. After pretraining is finished, backtranslation is used like described above to fine-tune the parameters in the encoder and decoders.

7. Datasets

To train, test and evaluate the models a dataset is required. For a translation model the dataset needs to contain a collection of sentences, called a corpus, from the source language and one from the target language. An Additional requirement when using supervised learning is that the two corpora need to be aligned, meaning that sentences from one corpus are linked to their translation in the other corpus. The source corpora are composed of informal sentences in the LaTeX format and the target corpora consist of Mizar statements. The researchers chose LaTeX as the informal language, because it is the defacto standard for mathematical publications. Mizar statements were chosen as the target language, because the Mizar Mathematical Library (MML) is the largest collection of formalized mathematics and its size is comparable to a natural language corpus. Additional minor reasons for Mizar are that Mizar statements are relatively easy to read by humans and that they are ASCII encoded, which makes processing them easier. The researchers identify Mizar not being based on type theory as a potential disadvantage. To evaluate and compare the models with each other the researcher created three different informal-formal datasets. Table 1 gives an overview of the content of the datasets. To test their data augmentation mechanism for the supervised model, a fourth dataset was created. This dataset uses Mizar as the source language and TPTP prefix format as the target language.

7.1. Synthetic LaTeX - Mizar Dataset

While there is a large number of mathematical papers in the LaTeX format publicly available, extracting the relevant parts and aligning them with their Mizar language counterparts to create a dataset that is sufficiently large enough to train a NMT model is a tedious and time-consuming endeavour. The researchers didn't want to commit this much time this early into their work and created a synthetic dataset as a temporary workaround. In order to synthetically create the dataset they used a tool [29] [30] which can translate Mizar sentences to LaTeX. This tool is used to translate Mizar articles in order for them to be published in the Mizar journal. By applying this method of "informalization" to the Mizar Math Library a dataset containing 1 million aligned entries was created. In table 2 you can see an example of such a LaTeX - Mizar sentence pair.

A downside of this method is that, while the resulting sentences are grammatically correct english, they are quite artificial and use a smaller vocabulary than a human would. Because of this the resulting model doesn't generalize well for real-world data.

7.2. ProofWiki - Mizar Datasets

While the synthetic LaTeX - Mizar dataset can be used to test the unsupervised models by simply forgetting the alignment, the researchers wanted to fully explore the capabilities of the unsupervised models by testing real-world data. For the unsupervised learning task, two unaligned monolingual corpora are sufficient. For the Mizar corpus the data that belongs to the MML was used. Those are the same sentences that were used in the first dataset. To get real-world LaTeX formulas, sentences from the proofwiki.org website [31] were extracted. ProofWiki contains more than 26k theorems that are written by user in the MathJax format, which is a browser adapted LaTeX format. While no aligned data is needed for training, a small aligned corpus for testing and evaluating the models is required. From previous work the researchers already had 470 theorems that were aligned between the MML and ProofWiki and this data was used for this purpose.

When doing unsupervised learning with natural languages, usually monolingual corpora from the same topic realm are chosen, e.g. news articles from both languages. To emulate this in their tests, the researchers created another dataset which had a similar characteristic. The aligned theorems from the first ProofWiki dataset were from point-set topology and lattice theory. So for this second ProofWiki - Mizar dataset only theorems from this topic realm are chosen for both monolingual corpora in addition to those aligned sentences. Examples from the aligned portion of the datasets are shown in table 3.

7.3. Mizar - TPTP Dataset

Using Mizar as the formal language for this experiment comes with the disadvantage of not being able to use Mizar's toolchain to improve translation quality. The problem is that individual statements are very hard to compile without any context like proper environment declarations or external references. Producing such an environment declaration is a challenging task to try to automate. To circumvent this problem while further testing the capabilities of this autoformalization approach, the researchers ran a different experiment with the supervised NMT model. Instead of translating LaTeX sentences to Mizar statements, Mizar statements are translated to TPTP formulas. TPTP stands for Thousands of Problems for Theorem Provers and library containing test problems for ATP systems [32]. These formulas can be further processed and then given to a custom type elaboration tool to determine their correctness.

This dataset of aligned Mizar statements and TPTP formulas was created by using the MPTP (Mizar Problems for

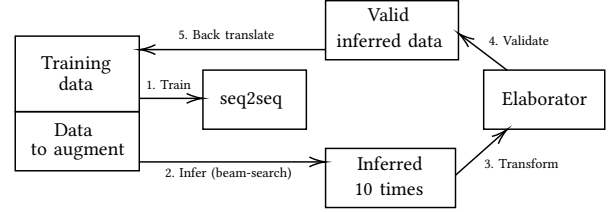


Figure 11: Feedback loop used for data augmentation

Theorem Proving) toolchain [33]. Examples are shown in table 4.

8. Data Augmentation via Feedback-loop

Data augmentation describes the process of increasing the amount of data by creating new artificial sentences or by modifying existing data. To implement this strategy in their test setup, the researchers created a feedback-loop that generates new aligned sentence pairs for the seq2seq model. The pipeline is shown in figure 11.

The process starts by training the seq2seq model with the Mizar - TPTP dataset normally. Then a subset of the training data is chosen for augmentation. The Mizar sentences from this subset are then given to the model in the inference phase. Beam-search is now used instead of greedy search to create ten new TPTP-FOF Prefix format sentences. These sentences are then transformed into TPTP-THF and given to the elaborator. The elaborator checks if a sentence is a valid formula by performing Mizar-style type checking. As this work focuses on the machine learning aspects of the autoformalization, the inner workings of the elaborator are not further discussed. The details can be found in the paper [2]. If the sentence is valid, it is back translated to the Mizar language using the toolchain and the new sentence pair is added to the dataset. The model is then trained again on the now enhanced dataset and the process repeats.

The idea behind the feedback-loop is to teach the model the correct syntactic format by creating new syntactically correct sentence pairs, that are similar to already existing ones.

9. Evaluation

The seq2seq model used 2-layer LSTM cells in both the encoder and decoder. The attention mechanism described in section 6.1.1 was used. Further information about hyperparameter configurations for this model can be found in the previous work from Kaliszyk et al. [1]. Because this model was too big for one GPU, it was trained on a CPU with a total training time of 17.9 hours.

The UNMT model was trained on one NVIDIA GeForceGTX 1080 Ti Graphics Card. The ProofWiki - Mizar topology dataset took 30 minutes to train and the ProofWiki - Mizar full dataset took 1 hour 8 minutes.

The cross lingual model was also trained on the same GPU

dataset	src	target	src sentences	target sentences	aligned
synthetic LaTeX	LaTeX	Mizar	1000000	1000000	1000000
proofwiki full	LaTeX	Mizar	200000	1000000	330
proofwiki topology	LaTeX	Mizar	30000	50000	330
TPTP	Mizar	TPTP Prefix Format	54000	54000	54000

Table 1: Contents and size of the datasets

Mizar	let T be RelStr ;
LaTeX	Let T be a relational structure .
\LaTeX	Let T be a relational structure .
Mizar	a ast (b ast t) <= b ast t ;
LaTeX	$a \text{ \textbackslash ast } (b \text{ \textbackslash ast } t) \text{ \textbackslash leq } b \text{ \textbackslash ast } t$.
\LaTeX	$a * (b * t) \leq b * t$.
Mizar	attr T is Noetherian means : Def1 : the InternalRel of T is co-well_founded ;
LaTeX	We say that $\{ T \}$ is Noetherian } if and only if (Def . 1) the internal relation of T is reversely well founded .
\LaTeX	We say that T is Noetherian if and only if (Def . 1) the internal relation of T is reversely well founded.

Table 2: Example entries of the syntactic LaTeX - Mizar dataset

Mizar	for T being non empty TopSpace for A being Subset of T st A is countable holds $A^0 = \{ \}$
LaTeX	Let $T = \left(\{S, \tau\} \right)$ be a topological space. Let A be a subset of S. Then if A is countable, then $A^0 = \text{\textbackslash varnothing}$.
\LaTeX	Let $T = (\{S, \tau\})$ be a topological space. Let A be a subset of S. Then if A is countable, then $A^0 = \emptyset$.
Mizar	for T being non empty TopSpace for A , B being Subset of T holds $(A \vee B)^0 = (A^0) \vee (B^0)$
LaTeX	Let $T = \left(\{S, \tau\} \right)$ be a topological space . Let A , B be subsets of S . Then $\left(\{A \cup B\} \right)^0 = A^0 \cup B^0$
\LaTeX	Let $T = (\{S, \tau\})$ be a topological space . Let A , B be subsets of S . Then $(\{A \cup B\})^0 = A^0 \cup B^0$

Table 3: Example entries of the ProofWiki - Mizar dataset

Mizar	for A holds A is doubleLoopStr & not A is empty implies B hold B is Scalar of A implies B is being_a_square
TPTP Prefix	iff ex C st C is Scalar of A & B = C ^2 c! b0 c=>_2 c&_2 cn16_algstr_0__1 b0 c~_1 cnv2_struct_0__1 b0 c! b1 c=>_2 cnm4_vectsp_1__2 b1 b0 c<=>_2 cnv1_o_ring_1__1 b1 c? b2 c&_2 cnm4_vectsp_1__2 b2 b0 cnr1_hidden__2 b1 cnk1_o_ring_1__1 b2

Table 4: Example entry of the Mizar - TPTP dataset

and training took 4 hours 24 minutes for the ProofWiki - Mizar topology dataset and 7 hour 53 minutes for the ProofWiki - Mizar full dataset.

9.1. Metrics

The researchers used multiple metrics to evaluate their autoformalization approach. All of them take a generated sentence and calculate a score that indicates the accuracy. The bilingual evaluation understudy (BLEU) score [34] compares two sentences by counting n-grams of various lengths from the generated sentence that appear in the reference sentences. The number of matches is called the precision of the n-gram.

$$p_n = \frac{\sum_{c \in \{candidates\}} \sum_{ngram \in c} count_{clip}(ngram)}{\sum_{c' \in \{candidates\}} \sum_{ngram' \in c'} count(ngram')}$$

This precision is then clipped to prevent to issue of the generated sentence containing an n-gram much more often than the reference. This prevents longer generated sentences to score higher simply because they contain more words. The precision scores for all n-grams of various

lengths are combined by calculating the log of the geometric mean.

$$\sum_{n=1}^N \frac{1}{N} \log(p_n) = \log\left(\prod_{n=1}^N p_n\right)^{\frac{1}{N}}$$

A brevity penalty is added for sentences that are too short. c is the candidate length and r the reference length.

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

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

This calculation gives the BLEU score for a single sentence pair. All the scores from the sentence pair in the test data are averaged to get a score for the whole corpus. The BLEU score ranges from 0 to 100 with higher values being better. Language translation models usually rank between 25-40 BLEU depending on the languages used for translation. Human result is somewhere between 25 and 34.

The perplexity score [35], which also consists of counting n-grams, but captures the idea, that words are randomly chosen before obtaining the correct one, was also used as a metric. The perplexity for a sequence of words W with

2000 samples	seq2seq	UNMT	XLM
BLEU	70.9	27.14	43.61
Perplexity	1.58	3.00	2.91
Edit distance 0	65.2%	26.8%	34.1%
Edit distance <=1	74.6%	34.4%	38.5%
Edit distance <=2	81.5%	41.8%	42.1%
Edit distance <=3	83.9%	46.3%	45.9%

Table 5: Results for synthetic LaTeX - Mizar dataset

length N is given as inverse probability of the words in the sequence.

$$perplexity(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

The probability depends on the type of the n-gram. Unigram probabilities:

$$P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2)...P(w_N)$$

Bigram probabilities:

$$P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_N|w_{N-1})$$

The probabilities of the individual words are taken from the test data. Perplexity measures how difficult it is for the model to generate the correct words in a sentence. Because the inverse probability is used, a lower score is being better here.

The Levenshtein edit distance measures the number of insertions, deletions and substitutions that the generated result is off the correct sentence.

There exists one major problem when using these metrics to evaluate this experiment. All the metrics only measure syntactic closeness, which is problematic when comparing formulas, because a single wrong negation can change the entire meaning of a sentence. The researchers decided to use them temporary as they are the defacto standard in the machine translation community and no metrics to measure the semantic closeness of formal statements exists right now.

9.2. Syntactic LaTeX - Mizar Dataset

In table 5 you can see the result for all three models being used with the synthetic LaTeX - Mizar dataset. The seq2seq model performs much better, because it benefits from the fully aligned data. One thing to note here is 65.2% of validation sentences achieving a Levenshtein edit distance of 0 which means the generated formula is an exact match. The XLM model performs better than the UNMT model, which is something that was expected and can be attributed to the use of pretraining.

9.3. ProofWiki - Mizar Datasets

Table 6 shows the result for the unsupervised models, when using the unaligned ProofWiki - Mizar datasets. The models have a significantly worse performance in all

131 samples	UNMT		XLM	
	topology	full	topology	full
BLEU	4.03	1.55	7.87	6.07
Perplexity	11.57	10.73	33.01	39.70
Edit distance 0	0%	0%	0%	0%
Edit distance <=1	0%	0%	0%	0%
Edit distance <=2	0%	0%	0.76%	0%
Edit distance <=3	9.92%	2.29%	6.11%	2.29%

Table 6: Results for unsupervised models

2000 samples	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5
BLEU	42.3	85.5	88.7	88.0	87.4
Perplexity	2.68	1.27	1.14	1.18	1.17
Edit distance 0	2.05%	2.40%	2.00%	5.15%	4.45%
Edit distance <=1	9.70%	13.4%	20.9%	21.6%	19.2%
Edit distance <=2	22.3%	25.0%	38.7%	36.9	34.6%
Edit distance <=3	32.6%	34.3%	49.3%	48.5%	45.4%

Table 7: Results for the augmentation mechanism

metrics. This is due to the much smaller size of the datasets and the fact that sentences in these datasets are generally longer. The XLM model performs generally better except the surprisingly high perplexity score. Also, the topology dataset achieves better results compared to the full dataset, which was to be expected, because of the smaller topic realm.

9.4. Evaluation of Augmentation

Table 7 shows the results of the augmentation mechanism after each iteration. Notable here is the doubling of the BLEU rate and halving of the perplexity score at iteration 2. Also note that the number of exact matches increases more than 100% at iteration 4. This improvement comes with a low number of increased training data. For example, it only has an increase of 16.4% between iteration 1 and 3. At the 4th iteration the improvement stagnates and starting with the 5th iteration the performance goes down again, which indicates that the model starts to overfit. Overall the results show that augmenting the training data with the feedback-loop improves the performance significantly.

10. Conclusion

Recent advancements in machine learning related to neural networks significantly increased the capabilities of machine translation tools. Kaliszyk et al. used these advancements to test their novel idea for an autoformalization approach by treating formalization like a language translation task. The researchers have shown that autoformalization of mathematics by using neural machine translation is definitely possible to some degree as shown by their experiments. They have also created a data augmentation mechanism to further improve the quality of the supervised model.

A limiting factor to fully judge the complete potential for this approach is the severe lack of datasets that contain formal and/or informal mathematics and are large enough to train neural machine translation models. The synthetically

created LaTeX - Mizar dataset shows that supervised learning achieves better results, but its artificial nature makes it unknown if similar results can be achieved with real-world data. Another problem is the lack of an evaluation metric that measures semantic closeness for formal statements. The researchers stated the creation of a sizeable dataset is part of their long term plan. They are exploring the possibility of combining the formal libraries of multiple proof assistants into a single embedding to increase the available data. All the models discussed only use the sequence of words itself for translation. The researchers propose the idea of giving the model additional information like the syntax tree of a formula or the environment declaration of the Mizar statements. Another idea to improve the translation quality the researchers mentioned, is to merge all the improvements from new machine learning advancements into a single model to test the maximum potential of this autoformalization approach. An idea proposed by Szegedy [36] is a pretraining method adapted to formulas. He calls them "skip-tree" models, where Instead of removing a single word, an entire subtree gets strategically deleted and during pretraining the model must try to predict the entire subtree. The still ongoing improvements of neural machine translation for natural languages is also a source of hope as they might allow better informal to formal translation, too.

References

- [1] Q. Wang, C. Kaliszyk, and J. Urban, "First experiments with neural translation of informal to formal mathematics," in *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, ser. LNCS, F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, Eds., vol. 11006. Springer, 2018, pp. 255–270. [Online]. Available: https://doi.org/10.1007/978-3-319-96812-4_22
- [2] Q. Wang, C. Brown, C. Kaliszyk, and J. Urban, "Exploration of neural machine translation in autoformalization of mathematics in mizar," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, ser. CPP 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 85–98. [Online]. Available: <https://doi.org/10.1145/3372885.3373827>
- [3] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, M. Schuster, N. Shazeer, N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, Z. Chen, Y. Wu, and M. Hughes, "The best of both worlds: Combining recent advances in neural machine translation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 76–86. [Online]. Available: <https://www.aclweb.org/anthology/P18-1008>
- [4] M. Luong, E. Brevdo, and R. Zhao, "Neural machine translation (seq2seq) tutorial," <https://github.com/tensorflow/nmt>, 2017.
- [5] G. Lample, M. Ott, A. Conneau, L. Denoyer, and M. Ranzato, "Phrase-based & neural unsupervised machine translation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 5039–5049. [Online]. Available: <https://www.aclweb.org/anthology/D18-1549>
- [6] G. Lample and A. Conneau, "Cross-lingual Language Model Pre-training," *arXiv e-prints*, p. arXiv:1901.07291, Jan. 2019.
- [7] H. Wang, "The formalization of mathematics," *The Journal of Symbolic Logic*, vol. 19, no. 4, pp. 241–266, 1954. [Online]. Available: <http://www.jstor.org/stable/2267732>
- [8] J. A. Robinson, "John mccarthy. computer programs for checking mathematical proofs. recursive function theory, proceedings of symposia in pure mathematics, vol. 5, american mathematical society, providence1962, pp. 219–227." *Journal of Symbolic Logic*, vol. 32, no. 4, p. 523–523, 1968.
- [9] D. Simon, "Checking natural language proofs," in *9th International Conference on Automated Deduction*, E. Lusk and R. Overbeek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 141–150.
- [10] "The qed manifesto," in *Proceedings of the 12th International Conference on Automated Deduction*, ser. CADE-12. Berlin, Heidelberg: Springer-Verlag, 1994, p. 238–251.
- [11] "Mizar project," <http://mizar.org>, accessed: 2021-03-20.
- [12] C. W. Zinn, "Understanding informal mathematical discourse," dissertation, University of Erlangen-Nuremberg, 2004.
- [13] C. Kaliszyk, J. Urban, and J. Vyskocil, "Learning to parse on aligned corpora (rough diamond)," in *Interactive Theorem Proving*, C. Urban and X. Zhang, Eds. Cham: Springer International Publishing, 2015, pp. 227–233.
- [14] C. Kaliszyk, J. Urban, and J. Vyskocil, "Automating formalization by statistical and semantic parsing of mathematics," in *Interactive Theorem Proving*, M. Ayala-Rincón and C. A. Muñoz, Eds. Cham: Springer International Publishing, 2017, pp. 12–27.
- [15] C. Kaliszyk, J. Urban, and J. Vyskocil, "System description: Statistical parsing of informalized mizar formulas," in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2017, pp. 169–172.
- [16] K. Grace, J. Salvatier, A. Dafoe, B. Zhang, and O. Evans, "Viewpoint: When will ai exceed human performance? evidence from ai experts," *Journal of Artificial Intelligence Research*, vol. 62, pp. 729–754, 07 2018.
- [17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, 01 2013.
- [18] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, Dec 1943. [Online]. Available: <https://doi.org/10.1007/BF02478259>
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: <https://doi.org/10.1145/3065386>
- [20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: <https://doi.org/10.1162/neco.1997.9.8.1735>
- [21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- [22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3104–3112.
- [23] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. [Online]. Available: <https://www.aclweb.org/anthology/D15-1166>

- [24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [25] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: <https://www.aclweb.org/anthology/Q17-1010>
- [26] R. Sennrich, B. Haddow, and A. Birch, "Improving neural machine translation models with monolingual data," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 86–96. [Online]. Available: <https://www.aclweb.org/anthology/P16-1009>
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.
- [28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: <https://www.aclweb.org/anthology/N19-1423>
- [29] G. Bancerek, "Automatic translation in formalized mathematics," in *Mechanized Mathematics and Its Applications* 5, 2006, pp. 19–31.
- [30] G. Bancerek and P. Carlson, "Mizar and the machine translation of mathematics documents," 1994. [Online]. Available: http://www.mizar.org/project/banc_carl93.ps
- [31] "Proofwiki," https://proofwiki.org/wiki/Main_Page, accessed: 2021-03-20.
- [32] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure," *J. Autom. Reason.*, vol. 59, no. 4, p. 483–502, Dec. 2017. [Online]. Available: <https://doi.org/10.1007/s10817-017-9407-7>
- [33] J. Urban, "MPTP 0.2: Design, Implementation, and Initial Experiments," *Journal of Automated Reasoning*, vol. 37, no. 1, pp. 21–43, Aug 2006. [Online]. Available: <https://doi.org/10.1007/s10817-006-9032-3>
- [34] 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*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: <https://www.aclweb.org/anthology/P02-1040>
- [35] G. Neubig, "Neural machine translation and sequence-to-sequence models: A tutorial," *ArXiv*, vol. abs/1703.01619, 2017.
- [36] C. Szegedy, "A promising path towards autoformalization and general artificial intelligence," in *Intelligent Computer Mathematics*, C. Benzmüller and B. Miller, Eds. Cham: Springer International Publishing, 2020, pp. 3–20.

Appendix A.

seq2seq translation example over training steps

⌘	Suppose s_8 is convergent and s_7 is convergent . Then $\lim(s_8+s_7) = \lim s_8 + \lim s_7$.
LaTeX	Suppose $\{s_{-8}\}$ is convergent and $\{s_{-7}\}$ is convergent . Then $\lim (\{s_{-8}\} + \{s_{-7}\}) = \lim \{s_{-8}\} + \lim \{s_{-7}\}$.
Correct Mizar	seq1 is convergent & seq2 is convergent implies $\lim (seq1 + seq2) = (\lim seq1) + (\lim seq2)$;
Snapshot-1000	$x \in \text{dom } f \text{ implies } (x * y) * (f(x (y (y y)))) = (x (y (y (y y))))$;
Snapshot-3000	seq is convergent & $\lim seq = 0c$ implies $seq = seq$;
Snapshot-5000	seq1 is convergent & $\lim seq2 = \lim seq2$ implies $\lim_{\text{inf}} seq1 = \lim_{\text{inf}} seq2$;
Snapshot-7000	seq is convergent & seq9 is convergent implies $\lim (seq + seq9) = (\lim seq) + (\lim seq9)$;
Snapshot-9000	seq1 is convergent & $\lim seq1 = \lim seq2$ implies $(seq1 + seq2) + (\lim seq1) = (\lim seq1) + (\lim seq2)$;
Snapshot-12000	seq1 is convergent & seq2 is convergent implies $\lim (seq1 + seq2) = (\lim seq1) + (\lim seq2)$;

Table A1: Training example from previous work [1]