Property Preserving Encoding of First-order Logic

Julian Parsert[1], Stephanie Autherith[2], and Cezary Kaliszyk[2,3]

[1] University of Oxford, United Kingdom
julian.parsert@gmail.com
[2] University of Innsbruck, Austria
{stephanie.autherith,cezary.kaliszyk}@uibk.ac.at
[3] University of Warsaw, Poland

Abstract

Logical reasoning as performed by human mathematicians involves an intuitive understanding of terms and formulas. This includes properties of formulas themselves as well as relations between multiple formulas. Although vital, this intuition is missing when supplying atomically encoded formulae to (neural) down-stream models.

In this paper we construct continuous dense vector representations of first-order logic which preserve syntactic and semantic logical properties. The resulting neural formula embeddings encode six characteristics of logical expressions present in the training-set and further generalise to properties they have not explicitly been trained on. To facilitate training, evaluation, and comparing of embedding models we extracted and generated data sets based on TPTP's first-order logic library. Furthermore we examine the expressiveness of our encodings by conducting toy-task as well as more practical deployment tests.

1 Introduction

During the first half of the 20^{th} century the concepts formal proof and proof systems were introduced [23]. Unlike a "standard" human proof, a formal proof is a proof with respect to a system of inference rules (i.e. axioms) where each step of the proof can only be the application of one of those rules. In theory, every correct human proof can be transformed to a formal proof. While these systems were not intended to be used in standard mathematics at the time, subsequent algorithmic developments and modern day computers allow for a formal approach to mathematical proofs [10]. Such developments include the proof of Kepler's conjecture [11] and the four colour theorem [8]. Formal proofs have also lead to the development of a formally verified operating system [16] and a compiler [18]. Nevertheless, due to inherent complexities involved in formal proof systems, automated or interactive theorem proving remains a cumbersome activity that requires a lot of human effort. With recent advancements in artificial intelligence, using these techniques in formal reasoning to overcome those hurdles have emerged. One hopes that these techniques will eventually lead to computers being able to derive theorems as complex as the ones derived by mathematicians in a more informal setting [4].

Human proofs do not originate in a vacuum. First, they require a familiarity with the concepts being used, i.e. the context of a statement. This may include auxiliary lemmas, alternative representations, or definitions. Sometimes it can happen that certain observations

are easier to make depending on the representation used [9]. An additional component is the experience of a mathematician who may have seen or proven similar theorems, which can be described as intuition. Once, a rough proof outline is developed, one can start filling in the necessary steps required for other mathematicians to understand the proof.

Computer provers derive theorems by manipulating syntax according to inference rules hard coded into the program. The aforementioned steps that a human goes through before finding a proof are completely irrelevant to a computer. However, this is the reason why humans are capable of deriving more involved theorems. Using deep learning we try to mimic this process. In this paper we will make the assumption that deep learning will be able to find a characterisation of mathematical objects that is good enough to reason with them efficiently. To enable this, we develop a first layer of such a deep learning approach to theorem proving, that is an embedding of formulas into a continuous vector space. That is, we develop a network that turns a character sequence (discrete sequence of symbols) that represents a mathematical statement (i.e. term, formula) into a potentially lower dimensional, continuous vector. We hope that this embedding preserves "important" information about the original formula as well as relations to other formulas. For example, when considering a distance metric, two formulas that are closely related (i.e. can be derived from each other, are unifiable, etc.) may be close in distance in the continuous vector space, even if they do not resemble each other syntactically. We consider properties (e.g. unifiability, alpha equivalence, etc.) which are important in theorem proving. We train encoding networks using such properties and with that hope to be able to capture the informal notions of "intuition" and "familiarity" which are used by human mathematicians.

Theorem Proving Problems and Artificial Intelligence In some ways automated theorem proving can be compared to a search problem where the search space can grow unboundedly depending on which decisions you make during the search. The method of estimating the "best decisions" which lead to a goal as fast as possible are called *proof guidance*. In [7] simple probabilistic models for such a task are used, while [19] connect deep learning models to an automated theorem prover. The current state of the art proof guidance uses reinforcement learning [14]. This and most other approaches (e.g. [15]) exclusively use human selected features. A related problem is that of automatically proposing interesting *conjectures and theory exploration*. Machine learning has been applied to this problem in the HipSpec system [6] which generates provable conjectures for subsequent proof searches. A task that plays a central role in interactive theorem proving is *premise selection* [17]. Here, we ask which previously proven theorems could be useful in proving our current goal. Deep learning was applied to premise selection in [1], and in [24]. The latter used embeddings based on the tree structure of formulas. Also here, human selected features are used with great success. Semantic embeddings have been proposed for related task such as [2] where the goal is deciding equivalence of propositional and polynomial expressions. Similarly, [20] presents a neural network that predicts if propositional logic formula is satisfiable. For many systems and tasks the best state of the art implementations all use human developed heuristics and features for simple learning models. We hope to be able to improve the performance in proof guidance when using deep learning embeddings that do not rely on humans to select features.

Contributions In this paper we develop a novel encoding or embedding of logical formulae and expressions. The encoding networks are trained with properties that are important in theorem proving procedures. In particular, we present the following contributions:

- We specify a number of properties relevant for theorem proving and create data sets for these properties.
- We design multiple two-part models. The first part, can be seen as a series of layers combining embedding/convolution/pooling or long short-term memory (LSTM) layers, that output an encoding of formulas and expressions. This encoding block is trained through back propagation from the second part, which consists of multiple classifiers that work purely on the generated encoding.
- These encodings are evaluated with respect to the aforementioned properties. We also train support vector machines (SVMs) based on these encodings with properties that the encoding networks were not specifically trained on. In addition, we evaluate the encodings using simple distance metrics to show that concepts such as variables were learned.

In Section 2 we will introduce some logical preliminaries and introduce the properties we will consider. We also present how the data was extracted and the structure of the data sets. In Section 3 we will present our learning framework as well as the encoding models. These are then evaluated in Section 4. Finally, we will discuss concluding remarks and future work in Section 5.

2 Logical Preliminaries and Data Sets

Our focus lies on first-order logic (FOL) formulas. In this section will introduce the syntax of first-order formulas and discuss some properties which will be used later. We will also argue that the selected properties are well suited for our purpose. This section gives only a brief overview, for a more detailed exposition we suggest a textbook on logic in computer science such as Huth and Ryan [13].

An abstract Backus-Naur Form (BNF) for FOL formulas is presented below. The two main concepts are terms (1) and formulas (2). A formula can either be an Atom (which has terms as arguments), two formulas connected with a logical connective, or a quantified variable or negation with a formula. Logical connectives are the usual connectives negation, conjunction, disjunction, implication and equivalence. In addition we have universal and existential quantifiers.

$$\text{term} := \text{var} \mid \text{const} \mid f(\text{term}, \ldots, \text{term}) \tag{1}$$

$$\begin{aligned} \text{formula} := {} & \text{Atom}(\text{term}, \ldots, \text{term}) \\ & \mid \neg\text{formula} \mid \text{formula} \wedge \text{forumla} \\ & \mid \text{formula} \vee \text{formula} \\ & \mid \text{formula} \rightarrow \text{formula} \mid \text{formula} \leftrightarrow \text{formula} \\ & \mid \exists \text{ var. formula} \mid \forall \text{ var. formula} \end{aligned} \tag{2}$$

For simplicity we omitted rules for bracketing. However, the "standard" bracketing rules apply. Hence, a formula is well-formed if it can be produced by 2 together with the mentioned bracketing rules. As theorem provers only work with well-formed formulas we expected our encodings to also reflect that the encoded formula is well formed. The parser is based on the syntax of the FOL format used in the "Thousands of Problems for Theorem Provers" (TPTP) library [21][1]. This library is very diverse as it contains data from various domains

[1] The full BNF can be found here: http://tptp.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html

including set theory, algebra, natural language processing or biology all expressed in the same logical language. Furthermore, its problems are used for the annual CASC competition for automated theorem provers. Our data sets are extracted from and presented in TPTP's format for first-order logic formulas and terms. An example for a TPTP format formula is `![D]: ![F]: (disjoint(D,F) <=> ~intersect(D,F))` which corresponds to the formula $\forall d.\ \forall f.\ \text{disjoint}(d, f) \iff \neg\,\text{intersect}(d, f)$. As part of the data extraction we developed a parser for TPTP formulas where we took some liberties. For example, we allow for occurrences of free variables, something the TPTP format would not allow. We can now introduce properties of these formulas that we will consider in subsequent sections and describe how the data was extracted.

Well-formedness: As mentioned above it is important that the encoding networks preserve the information of a formula being well-formed. The data set was created by taking TPTP formulas as positive examples and permutations of the formulas as negative examples. We generate permutations by randomly iteratively swapping two characters and checking if the formula is well-formed, if it is not, we use it as a negative examples. This ensures that the difference between well-formed formulas and non well-formed formulas is not too big.

Sub-formula: Intuitively, the sub-formula relation maps formulas to a set of formulas which comprise the original formula. Formally, the sub-formula relation is defined as follows:

$$\text{sub}(\phi) = \begin{cases} \{\phi\} & \text{if } \phi \text{ is Atom} \\ \text{sub}(\psi) \cup \{\phi\} & \text{if } \phi \text{ is } \neg\psi \\ \text{sub}(\psi_1) \cup \text{sub}(\psi_2) \cup \{\phi\} & \text{if } \phi \text{ is } \psi_1 \bullet \psi_2 \text{ and } \bullet \in \{\vee, \wedge, \rightarrow, \leftrightarrow\} \\ \text{sub}(\psi) \cup \{\phi\} & \text{if } \phi \text{ is } \forall x.\ \psi \\ \text{sub}(\psi) \cup \{\phi\} & \text{if } \phi \text{ is } \exists x.\ \psi \end{cases}$$

Notice, how we never recursively step into the terms. As the name suggests we only recurse over the logical connectives and quantifiers. Hence, $g(x)$ is not a sub-formula of $\neg f(g(x), c)$ whereas $f(g(x), c)$ is (since "$\neg$" is a logical connective of formulas). Importantly, the sub-formula property preserves the tree structure of a formula. Hence, formulas with similar sets of sub-formulas are related by this property. Therefore, we believe that recognising this property is important for obtaining a proper embedding of formulas. In the presented data set the original formulas ϕ are taken from the TPTP data set. Unfortunately, finding negative examples is not as straightforward, since each formula has infinitely many formulas that are not sub-formulas. In our data set we only provide the files as described above (positive examples). To create negative examples during training, we randomly search for formulas in which are not a sub-formula. Since, we want to have balanced training data we search for as many negative examples as positive ones.

Modus Ponens: One of the most natural logical inference rules is called *modus ponens*. The modus ponens (MP) allows the discharging of implications as shown in the inference rule (3). In other words, the consequent (right-hand side of implication) can be proven to be true if the antecedent (left-hand side of implication) can be proven.

$$\frac{P \qquad P \rightarrow Q}{Q} \tag{3}$$

Using this basic inference rule we associate two formulas ϕ and ψ with each other if ϕ can be derived from ψ in few inference steps, including modus ponens. It turns out that despite its simplicity, modus ponens makes for a sound and complete proof calculus for the (undecidable) fragment of first-order logic known as Horn Formulas [5].

Example 2.1. *We can associate the two formulas* $\phi := \forall x.\ ((P(x) \rightarrow Q(x)) \wedge P(x))$ *and* $\psi := \forall x.\ Q(x)$ *with each other, since* ψ *can be proven from* ϕ *using the modus ponens inference rule (and some others).*

Providing data for the this property required more creativity. We had two approaches: Option one involves generating data directly from the TPTP data set, while the other option comprised synthesising data ourselves with random strings. In the data set we provide both alternatives are used. First, we search for all formulas in the TPTP set that contained an implication and added the antecedent using a conjunction. We paired this formula with the formula containing only the consequent. We tried to introduce "diversity" to this data by swapping around conjuncts and even adding other conjuncts in-between. Secondly, we synthesise data using randomly generated predicate symbols.

Alpha-Equivalence: Two formulas or terms are alpha equivalent if they are equal modulo variable renaming. For example the formulas $\forall x\ y.\ P(x) \wedge Q(x, y)$ and $\forall z\ y.\ P(z) \wedge Q(z, y)$ are alpha equivalent. Alpha equivalence is an important property for two reasons. First, it implicitly conveys the notion of variables and their binding. Second, one often works on alpha equivalence classes of formulas and hence, alpha equivalent formulas need to be associated with each other.

Term vs Formula: We generally want to be able to distinguish between formulas and terms. This is a fairly simple property, especially since it can essentially be read of the BNFs 1 and 2. However, it is still important to distinguish these two concepts, and a practical embedding should be able to do so.

Unifiability: Unifiability plays an important role in many areas of automated reasoning such as resolution or narrowing [3]. Unifiability is a property that only concerns terms. Formally, two terms are unifiable if there exists a substitution σ such that $s \cdot \sigma \approx t \cdot \sigma$. Informally, a substitution is a mapping from variables to terms and the application of a substitution is simply the replacing of variables by the corresponding terms. Formally one needs to be careful that other variables do not become bound by substitutions. Example 2.2 showcases these concepts in more detail.

Example 2.2. *Substitution and Unifiability: The terms* $t = f(g(x), y))$ *and* $s = f(z, h(0))$ *are unifiable, since we can apply the substitution:* $\{z \mapsto g(x),\ y \mapsto h(0)\}$ *such that* $t \cdot \sigma = f(g(x), h(0)) = f(g(x), h(0)) = s \cdot \sigma$.

Syntactic unification, which is the type of unification described above is quite simple and can be realised with a small set of inference rules. However, even adding additional information such as associativity or commutativity can make unification an extremely complex problem [3]. Putting unification into a higher-order setting makes it even undecidable [12].

In our supplementary material we include data sets of different sizes extracted from different parts of the TPTP library. They were all extracted from the Set Theory library of TPTP. The program we provide can be used to extract bigger data-sets from any set of TPTP formulas, including the libraries of interactive theorem provers translated to the TPTP format [22] used in a number of theorem proving challenges.

3 Model(s)

We considered six different embedding models, four of which are based on Convolutional Neural Networks (CNNs) and the others are based on long short-term memory networks (LSTMs). However, the setup used for training the networks remains the same. Hence, we will discuss the general architecture first. After that we will elaborate on each neural encoding model.

Problem Formulation Given a first-order formula ϕ we wish to learn a function $\text{enc}(\cdot)$. When given the formula ϕ of length m the function $\text{enc}(\phi)$ should produce a continuous vector representation of ϕ of length n. If $m > n$ we call $\text{enc}(\cdot)$ an embedding, otherwise if $m = n$ we refer to it as encoding. The continuous vector representation of formulas $\text{enc}(\phi)$ has to preserve the properties mentioned above. That is, if two formulas ϕ and ψ are alpha equivalent then this is reflected in $\text{enc}(\phi)$ and $\text{enc}(\psi)$ and if ϕ is well formed this should be deducible from $\text{enc}(\phi)$ (and similarly for all other properties).

3.1 Training Framework

Our goal is to obtain an encoding[2] network that preserves properties. For that we develop multiple encoding networks (c.f. green box in Figure 1) that we will discuss later. All of them produce an encoding $\text{enc}(\phi)$ of a formula ϕ. This continuous vector representation is then fed into classifiers that recognise the properties discussed in Section 2. The total loss $\mathcal{L}$ is calculated by taking the sum of the losses $\mathcal{L}_P$ of each classifier of the properties $p \in \mathcal{P}$ discussed before. $\mathcal{L}$ is then propagated back into the classifiers and the encoding network. This setup is end-to-end trainable and ensures, that the resulting embedding preserves the properties discussed in Section 2. The setup of the training phase is illustrated in Figure 1. We train the network on this setup and evaluate the whole training setup (encoding network and classifiers) on unseen data in Section 4. However, since our main goal is the encoding network, we can extract the encoding network (c.f. Figure 1) and discard the classifiers after training and evaluation.

Classifiers The classifiers' purpose is to train the encoding network. This is implemented by jointly training each layer in the encoding networks and classifiers to extract continuous representations for correct property classification in the final layer. There are two philosophies that can go into designing these classifiers. The first is to make the classifiers as simple as possible, i.e. a single fully connected layer. This makes the encoding networks encode the properties in a "high-level" fashion as more complex relationships cannot be recognised by a single fully connected layer. This is advantageous if one wants to train simpler machine learning models on the encodings. On the other hand, when using multiple layers in the classifiers more complex relationships can be recognised and the encoding networks can encode more complex features without having to keep them "high-level". However, if the problems for the classifiers are too easy it could happen that only the classifier layers are trained and the encoding network layers remain "untouched" i.e. do not change the char-level encoding significantly. We chose to go a middle route by using two fully connected layers, although we believe that one could investigate further solutions to this problem (e.g. adding weights to loss).

[2]We will mostly be talking about encoding networks but the same applies for embedding networks unless specified otherwise.

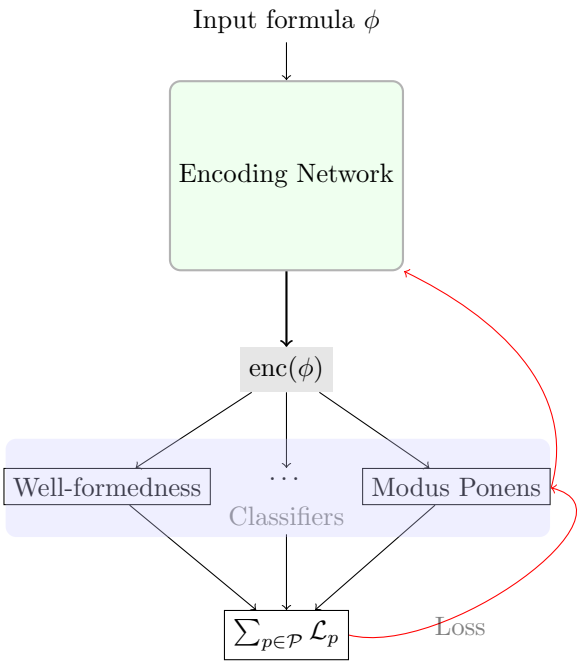

Figure 1: This graph shows the training framework we developed. The bottom area contains the classifiers that get one or more continuous representations of formulas enc(ϕ) as input. If the classifier takes two formulas as input (i.e. alpha-equivalence), we gather enc(ϕ_1) and enc(ϕ_2) separately and forward the pair (enc(ϕ_1), enc(ϕ_2)) to the classifier. The encoding networks are described subsequently (cf. Figure 2).

3.2 Embedding Models

In the previous section we treated the embedding models themselves as a green box (Figure 1) without going into more detail. We considered two different types of encoding networks. First, discuss the CNNs based models and later the ones based on LSTMs. The models' graphs are roughly depicted in Figure 2. The exact dimensions and sizes of the models are discussed in Section 4 as they are not fixed in the models.

CNN based models The first encoding model we present is the basic model based on CNNs (cf. left side in Figure 2). The first layer is a variable size embedding layer, the size of which can be changed with a command line argument. Once the formulas have been embedded, we pass them through a set of convolution and (max) pooling layers. In our current model we have 9 convolution and pooling layers with increasing filter sizes and ReLUs as activation functions. The output of the final pooling layer comprises the encoding of the input formula. In the second model we append an additional set of fully connected layers after the convolution and pooling layers. However, these do not reduce the dimensionality of the vector representation. For that we introduce a third type of models, which we call embedding models. For these the last layer is a projection layer which we tested with output dimensions 32 and 64. Note that between the last pooling layer and the projection layer one can optionally add fully connected layers like in the previous model. In Section 4 we evaluate these models.

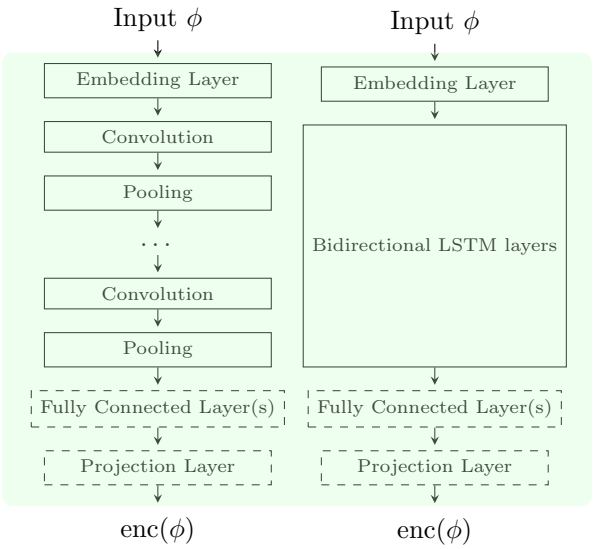

Figure 2: The encoding models we considered with the layers that the input passes through. On the left we show the CNN based models, while on the right the LSTM based models are presented. The dashed boxes describe layers that are not present in each model of that type.

LSTM based models We considered three LSTM based models (see right side of Figure 2). Much like in previous models, the first layer is comprised of an embedding layer. The output of which gets fed into bidirectional LSTM layers. The output of these layers serves as the encoding of our input formulas. As with the CNN based models we also considered models where an additional set of fully connected or projection layers is added.

4 Evaluation

As mentioned in Section 1 there are many use cases for encodings of formulas. Hence, we will discuss different possible ways of evaluation. First, we consider the properties the models have been trained with (cf. Section 2). Here, we have two different ways of obtaining evaluation and test data. We also want the encoding networks to generalise to, and preserve properties that it has not specifically been trained on. Therefore, we encode a set of formulas and expressions and train an SVM (without kernel modifications) with different properties on them. Finally, we will also discuss a more informal nearest neighbour metric that we used to evaluate the encodings.

For the first and most straightforward evaluation we use the data extracted from the Graph Theory and Set Theory library described in Section 2 as training data. In order to evaluate the models, we could split this data before training into a training set and evaluation set. Hence, the network is evaluated on unseen data. In this approach however, constants, formulas, etc. occurring in the evaluation data may have been seen before in different contexts. For example, considering the Set Theory library, terms and formulas containing `union(X,Y)`, `intersection(X,Y)`, etc. will occur in training data and evaluation data. Indeed, in applications such as premise selection such similarities and connections are actually desired, which is one of the reasons we use char level encodings. Nevertheless, we will focus on more difficult evaluation/test data. We will use data extracted from the Category Theory library as evalu-

Network	embedding dimension	sub-formula multi-label classification	binary sub-formula classification	modus ponens	term vs formula classification	unifiability	well-formedness	alpha equivalence
CNN	32	0.999	0.625	0.495	0.837	0.858	0.528	0.498
CNN	64	0.999	0.635	0.585	0.87	0.73	0.502	0.55
CNN	128	0.999	0.59	0.488	0.913	0.815	0.465	0.587
CNN with Projection to 32	64	1.0	0.662	0.992	0.948	0.81	0.748	0.515
CNN with Projection to 64	64	1.0	0.653	0.985	0.942	0.718	0.85	0.503
CNN with Fully Connected layer	32	0.999	0.64	0.977	0.968	0.78	0.762	0.5
CNN with Fully Connected layer	64	0.999	0.668	0.975	0.973	0.79	0.77	0.548
CNN with Fully Connected layer	128	0.999	0.635	0.923	0.972	0.828	0.803	0.472
CNN with Fully Connected layer Pr to 32	64	1.0	0.648	0.973	0.922	0.865	0.69	0.487
CNN with Fully Connected layer Pr to 64	64	1.0	0.662	0.968	0.967	0.898	0.762	0.497
LSTM	32	1.0	0.652	0.488	0.975	0.883	0.538	0.508
LSTM	64	0.999	0.652	0.49	0.942	0.86	0.49	0.575
LSTM	128	1.0	0.643	0.473	0.96	0.885	0.51	0.467
LSTM Pr to 32	64	1.0	0.69	0.537	0.863	0.87	0.513	0.62
LSTM Pr to 64	64	1.0	0.598	0.535	0.845	0.902	0.515	0.575
LSTM with Fully Connected layer	32	0.999	0.638	0.485	0.855	0.902	0.532	0.692
LSTM with Fully Connected layer	64	1.0	0.63	0.491	0.882	0.848	0.52	0.833
LSTM with Fully Connected layer	128	1.0	0.635	0.473	0.968	0.887	0.51	0.715
LSTM with Fully Connected layer Pr to 32	64	1.0	0.657	0.495	0.96	0.883	0.505	0.672
LSTM with Fully Connected layer Pr to 64	64	1.0	0.62	0.503	0.712	0.898	0.492	0.662

Table 1: Accuracies of classifiers working on different encoding/embedding models. The models were trained on the Graph/Set theory data set and the evaluation was done on the unseen Category Theory data set. The LSTM based models are in grey. (Pr = Projection)

ation data and the Set/Graph Theory data as training data. Hence, training and evaluation sets are significantly different and do not share terms, constants, formulas, etc. We train the models on embedding dimensions 32, 64, and 128 (we only consider 64 for projective models). The input length, i.e. the length of the formulas was fixed to 256, since this includes almost all training examples. The CNN models had 8 convolution/pooling layer pairs of increasing filters sizes (1 to 128), while the LSTM models consisted of 3 bidirectional LSTM layers each of dimension 256. In the "Fully Connected"-models we append two additional dense layers. Similarly, for the projective models we append a dense layer with a lower output dimension.

The evaluation results of the models are shown in Table 1. The multi-label sub-formula classification is not relevant for this evaluation since training and testing data is significantly different. However, the binary sub-formula classification is useful and proves to be a difficult property to learn. Surprisingly, adding further fully connected layers seems to have no major effect for this property regardless of the underlying model. In contrast, the additional dense layers vastly improve the accuracy of the modus ponens classifier (from 49% to 97% for the simple CNN based model with embedding dimension 32). It does not make a difference whether these dense layers are projective or not. Interestingly, every LSTM model even the ones with dense layers fail when classifying this property. Similar observations although with a smaller difference can be made with the term-formula distinction. Classifying whether two terms are unifiable or not seems to be a task where LSTMs perform better. Generally however, the results for unifiability are similarly good across models. When determining whether a formula is well formed CNN based models again outperform LSTMs by a long shot. In addition, a big difference in performance can be seen between CNN models with additional layers (projective or not) appended. Unsurprisingly alpha equivalence is a difficult property to learn especially for CNNs. This is the only property where LSTMs clearly outperform the CNN models. Thus combining LSTM and CNN layers into a hybrid model might proof beneficial in future works. In addition, having fully connected layers appears to be necessary in order to achieve accuracies significantly above 50%.

Generally, varying embedding dimensions does not seem to have a great impact on the performance of a model, regardless of property. As expected, adding additional fully connected

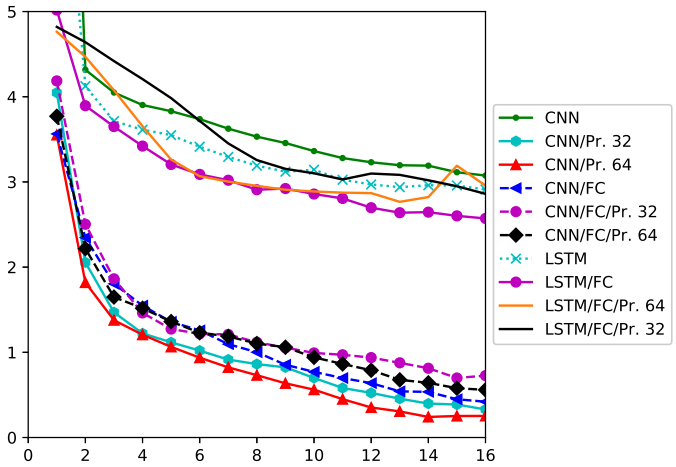

Figure 3: Cumulative loss in each epoch during training.

layers has no negative effect. This leads us to distinguish two types of the properties: Properties where additional dense layers have a big impact on the results (modus-ponens, well-formedness, alpha-equivalence), and those where the effect of additional layers is not significant (unifiability, term-formula, bin. sub-formula). It does not seem to make a big difference whether the appended dense layers are projective or not. Even the embedding models that embed the formulas to an 8^{th} of the input dimension perform very well. Another way of classifying the properties is to group properties where CNNs perform significantly better (modus-ponens, well-formedness), and conversely where LSTMs are preferable (alpha equivalence).

We also present data that shows the performance of the models during training. In Figure 3 we show the cumulative loss in each epoch.

4.1 Other Problems and Properties

We also want the encodings of formulas to retain information about the original formulas and properties that the networks have not specifically been trained on. We want the networks to learn and preserve unseen structures and relations. We conduct two lightweight tests for this. First, we train simple models such as SVMs to recognise certain structural properties (that we did not specifically train for) in the encodings of formulas. To this end we train SVMs to detected logical connectives such as conjunction, disjunction, implication, etc. These classifications are important since logical connectives were not specifically used to train the encoding networks (cf. Section 3) but are important nevertheless. Here, the SVMs correctly predict the presence of conjunctions, etc. with an accuracy of 85%. We also train an ordinary linear regression model to predict the number of occurring universal and existential quantifiers in the formulas. This regression correctly predicts the number of quantifiers with an accuracy of 94% (after rounding to the closest integer). These results were achieved by using the CNN based model with fully connected layers. We also evaluated the projective models with this method. We achieved 70% and 84% for classification and regression respectively using the CNN model with fully connected and projection layer. When using models that were trained using

Target	$union(intersection(D,F), intersection(D,I))$	D
1st	$union(intersection(D,F), difference(D,F))$	C
2nd	$unordered\_pair(empty\_set, singleton(C))$	K
3rd	$intersection(union(C,D), union(C,F))$	L

Table 2: We show the 3 formulas and terms whose continuous vector representation is closest to the vector of the target formula or term.

single layer classifiers as discussed in Section 3.1 we get better results for simple properties such as the presence of conjunction.

Furthermore, we created a nearest neighbour (based on a distance metric) evaluation that we conducted using the encoding of a set of formulas. The results of this evaluation are displayed in Table 2. In the second column we see the term closest to $union(intersection(D,F), intersection(D,I))$ is $union(intersection(D,F), difference(D,F))$. The last column shows that the closest point in space to variables are other variables, i.e. the closest term to the variable D is the variable C, K and so on. This similarity is not surprising given that their vector representations have the same structure (mostly zeroes apart from the first entry). Nevertheless, it shows that first of all, the networks don't introduce obfuscation and second of all that it does not confuse variables with single letter predicate symbols. Furthermore, the encoding network still retains similar strings even if they are in different positions of the expression. In other examples not presented in Table 2 we can observe that different formulas containing an implication with the same antecedent but different consequent are also close to each other.

5 Conclusion

We discussed several problems related to automated reasoning where modern artificial intelligence technologies can be utilised. We extracted data sets from the TPTP library that describe several properties of formulas that are relevant for theorem proving. Using these we develop a two-stage learning framework where we train an encoding network based on these properties. We consider ten different encoding models which make use of LSTMs and CNNs. In Section 4 we evaluated these models on unseen data and verify that the encoding and embedding networks do indeed preserve important syntactic and semantic properties of the input formulas. In order to verify that the networks also learn features that were not specifically trained, we train SVMs and linear regression models to recognise unseen properties on the encodings of formulas. In addition, we present a nearest neighbour evaluation.

The code and data sets are available at:

http://cl-informatik.uibk.ac.at/cek/gcai2020/

Future work can include the expanding to other types of input data. As mentioned we only considered TPTP's first-order logic. However, we hope to apply similar techniques to higher-order logic where some of the considered properties become more complex or even undecidable. Extending to higher-order logic would also enable us to directly incorporate the encodings in premise selection and proof guidance for proof assistants based on more complex and undecidable foundations including HOL, where automation has so far been much weaker than for first-order logic. We also hope to use the encodings developed in this paper in order to improve proof guidance of first-order logic theorem provers. In particular, we believe that the encodings preserve relations between logical statements that can be leveraged to support these theorem provers in their proof search.

Acknowledgements

This work has been supported by the ERC starting grant no. 714034 *SMART*.

References

[1] Alexander A. Alemi, François Chollet, Niklas Eén, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike V. Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2235–2243, 2016.

[2] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles A. Sutton. Learning continuous semantic representations of symbolic expressions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 80–88. PMLR, 2017.

[3] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.

[4] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[5] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer-Verlag Berlin Heidelberg, 1997.

[6] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 392–406, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Michael Färber and Chad E. Brown. Internal guidance for satallax. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016*, volume 9706 of *Lecture Notes in Computer Science*, pages 349–361. Springer, 2016.

[8] Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[9] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, et al. A machine-checked proof of the odd order theorem. In *International Conference on Interactive Theorem Proving*, pages 163–179. Springer, 2013.

[10] Thomas C Hales. Formal proof. *Notices of the AMS*, 55(11):1370–1380, 2008.

[11] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[12] Gérard P. Huet. Higher order unification 30 years later. In Victor Carreño, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 15th International Conference, TPHOLs 2002, Hampton, VA, USA, August 20-23, 2002, Proceedings*, volume 2410 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2002.

[13] Michael Huth and Mark Dermot Ryan. *Logic in computer science - modelling and reasoning about systems*. Cambridge University Press, 2004.

[14] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olšák. Reinforcement learning of theorem proving. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8836–8847. 2018.

[15] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3084–3090. AAAI Press, 2015.

[16] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: formal verification of an operating-system kernel. *Commun. ACM*, 53(6):107–115, 2010.

[17] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, pages 378–392, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[19] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017.

[20] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018.

[21] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[22] Josef Urban. Mptp 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1):21–43, Aug 2006.

[23] J. Van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard University Press, 1967.

[24] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 2783–2793, 2017.