Neural Theorem Prover

Arianna, Yuan
Department of Psychology
Stanford University
Stanford, CA 94305
xfyuan@stanford.edu

Abstract

In the current project, I build a neural network model to prove theorems in logical forms. Particularly, the model receives a set of axioms (premises) and a theorem to prove (goal). The model needs to select multiple axioms from the axiom list that could prove the goal theorem. I reframe it as a sequence to sequence learning problem, and use double recurrent neural networks to encode the theorem and output a sequence of axioms. I build three models and showed that the model which dynamically encodes the axioms perform much better than the model that does not do so. The current work demonstrates the power of deep neural networks in automated theorem proving and provide a cognitively plausible computational model for logical inference.

1 Introduction

In recent years, neural network models have been applied to many artificial intelligent tasks, such as relation extraction[1], question answering[2], text compression[3], machine translation[4], etc. Critically, deep neural network with unsupervised learning have successfully solved many tasks that are traditionally tackled with rule-based approaches, and they demonstrate their advantages over those previous approaches. However, few studies have applied deep neural networks on automated theorem proving, which relies heavily on logic and rules. Therefore, I want to fill this gap in the literature by showing that deep neural networks can perform theorem proving without any explicit knowledge of logic.

Particularly, I build a neural network model to prove theorems in logical forms. The model receives a set of axioms (premises) and a theorem to prove (goal). It needs to select multiple axioms from the axiom list that lead to the goal theorem. To tackle this task, I reframe it as a sequence to sequence learning problem, i.e., the model takes a sequence of words (goal theorem) as input, and generates a sequence of axioms that will prove the goal statement. Sequence to sequence learning with recurrent neural network has gained great popularity in the field of machine translation[4, 5]. However, no previous research has attempted to use such models to generate proofs.

The current approach has several innovations: 1) Contrary to traditional automated theorem proving algorithms, the current model does not involve any search. The neural network model learns to sequentially select axioms by pure feature extraction. Therefore, it is computationally less expensive. 2) In this paper, I use data in logical forms. However, the model can be easily extended to daily life language and languages other than English, since it is essentially a language model. 3) The model provides the first end-to-end theorem prover. Specifically, it takes the inputs word by word, embed them into vector representations and output the likelihood of each axioms to be selected based on their vector representations. Everything is continuous and can be trained via back propagation on a large corpus. No hand-crafted knowledge about logical inference are needed and it easily generalizes to different languages. 4) It involves pattern recognition and attention mechanisms, which resembles the theorem proving process of humans. Therefore, it offers a cognitively plausible computational model for human logical inference, which is very important from the cognitive psychology perspective.

2 Related work

Traditionally, logical inference tasks are performed using logical provers. Given a list of axioms, a logical prover uses forward-chaining to produce all the logical consequences given the axioms. It returns once the theorem is produced or refuted [6]. Logical provers are robust, but even a small axioms set would make the search space really large, which makes the algorithm very inefficient. In addition, traditional logical prover requires converting natural language to machine-readable logical forms, and then applying logical inference rules and search mechanism to find a path to reach the goal. However, this transformation is itself a non-trivial question. It does not generalize well to different languages, and requires a lot of hand-crafted rules.

Only recently did researchers began to use neural networks to tackle this task. One of the latest breakthrough in this field is Samuel Bowman and colleagues' work on logical semantics, in which they use recursive neural network to extract the logical relationship between two sentences [7, 8]. Particularly, they extended the previous recursive tree-structured model. Their model takes two sentences as input, then outputs the logical relationship between the two input sentences, namely, neutral, entailment or contradiction. The authors implemented two tree-structured models to encode sentences into fixed-dimension vector representations. The two models are Plain TreeRNNs, which uses a 2D matrix to combine word embeddings at each tree node, and tree-structured neural tensor networks (TreeRNTNs), which uses a 3D tensor to combine word embeddings at each tree node. Both networks consist of multiple composition RN(T)N layers and one comparison RN(T)N layer. All the composition layers share the same weights. Composition layers transform two input sentences into two vector embeddings and feed them to the comparison layer. Comparison layer then takes two sentence embeddings and use a softmax function to classify their logical relationships. Bowman and colleagues' work demonstrates the power of neural networks to perform logical reasoning without explicit knowledge of logic. However, their model cannot be used to select a sequence of axioms to prove a specific theorem. To my knowledge, this study is the first one using neural network to solve theorem proving in an end-to-end manner.

Another line of related research is recurrent neural network (RNN) in machine translation. Machine translation can be viewed as a sequence-to-sequence learning problem and Sutskever et al. propose a Long Short-Term Memory (LSTM) based RNN to learn machine translation[4]. The RNN takes sentences with varying lengths as inputs, i.e., word embeddings are sequentially fed into the LSTM cells until EOS is detected. Then the model returns the translated sentence word by word. LSTM cells endow the networks with great power to capture long-term dependencies and deal with the vanishing gradient problem. I use a similar approach in the current project, i.e., instead of outputting a sentence in another language, the model outputs a sequence of axioms indices. Since the sentences are not very long in my dataset, I did not use LSTM cells.

Although the data format in the current task is similar to the one in machine translation task, there is a fundamental difference between the theorem proving task and machine translation task: the current task requires the model to encode the axiom list to determine which one should be selected. To incorporate the axiom list into the model, I draw inspiration from the Neural Programmer developed by Neelakantan, Le and Sutskever [9]. In their original paper, the model performs a table-comprehension task. Given a quest, it selects relevant numeric inputs and operations sequentially. For instance, when being asked "What is the sum of the net incomes?", it first converts the query sentence and all the column names into vector representations, and then uses these vector representations to select the column "net income". Similarly, in my project, the model first converts the theorem and all the axioms into vector representations, and then use these vector representations to select the correct axioms successively.

3 Approach

I implement three sequence-to-sequence learning models and run four experiments. The first model contains two classic sequence-to-sequence RNNs and the axioms are not processed by the neural network. Therefore, the neural network does not see the sentence embedding of the axioms. It is termed as Static Double Seq2Seq RNN. The second model learns the sentence embeddings for both the axiom and the to-be-proved theorem, and uses both of them to predict which axioms to select. It is termed as the Dynamic Double Seq2Seq RNN. The third one is a control condition for the dynamic Double Seq2Seq RNN. It matches the number of layers in the dynamic Double Seq2Seq RNN, but does not include the sentence embedding of the axioms. In the following sections I describe each model in details.

3.1 Static Double Seq2Seq RNN

In all models, I use two RNNs for sequence-to-sequence learning. The first RNN is used to encode the texts (goal or axioms), termed as RNN_S. The second one is used to predict the axiom sequence, termed as RNN_P. It takes the output of RNN_S as inputs. The model structure is illustrated in Figure 1.

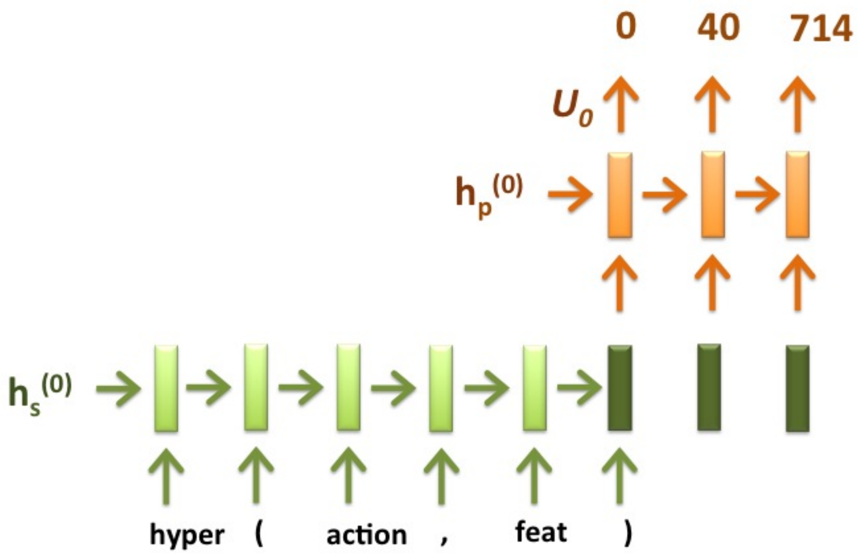

Figure 1: Static Double Seq2Seq RNN

The first recurrent neural network RNN_S transforms the to-be-proved theorems into vector representations. During the training, the word embedding matrix $V \in \mathbb{R}^{d\times|V|}$, the weight matrix W_S and the bias b_S are updated. Note that d is the word embedding size, D_h is the sentence embedding size and $|V|$ is the vocabulary size. The last hidden state of RNN_S is used as the vector representation of the sentence. The RNN_S is described by the following equations:

$$z_i = \tanh(W_S[z_{i-1}; V(w_i)] + b_S),$$
$$W_S \in \mathbb{R}^{D_h\times 2d}, b_S \in \mathbb{R}^{D_h}, z_i \in \mathbb{R}^{D_h} \qquad (1)$$

$z_S \in \mathbb{R}^{D_h}$ is the final representation of a theorem of length S, termed as q.

q is then fed into the second recurrent neural network RNN_P. At each time step t, RNN_P uses q, and the hidden state at time $t-1$ to generate a probability distribution α_t^{axioms} over all axioms. The RNN_P is described by the following equations:

$$\begin{aligned} &h_P^t = \tanh(W_P[q; h_P^{t-1}] + b_1), \\ &\alpha_P^t = softmax(U_0 h_P^t + b_2), \\ &W_P \in \mathbb{R}^{D_h \times 2D_h}, b_1 \in \mathbb{R}^{D_h}, U_0 \in \mathbb{R}^{C \times D_h}, b_2 \in \mathbb{R}^C \end{aligned} \tag{2}$$

and W_P, b_1, U_0, b_2 are updated. When training these two neural networks, we apply dropout to their input layers to avoid feature co-adaptation [10]. The dropout rate is set as 0.9.

3.2 Dynamic Double Seq2Seq RNN

In the Dynamic Double Seq2Seq RNN, RNN_S is used to encode not only the goal but also the axioms. In other words, each axiom is also fed into RNN_S to obtain its vector representations $p^{axioms} \in \mathbb{R}^{D_h}$. These p^{axioms} formed a matrix $P^a \in \mathbb{R}^{C \times D_h}$ (C denotes the number of axioms in the axiom list).

In addition, the second recurrent neural network RNN_P has an additional layer above the original hidden layer in Static Double Seq2Seq RNN, in the sense that h_P^t is multiplied by P^{axioms} before fed into the softmax function. The RNN_P is described by the following equations:

$$\begin{aligned} &h_P^t = \tanh(W_P[q; h_P^{t-1}] + b_1), \\ &\alpha_P^t = softmax(P^{axioms} \tanh(U_1 h_P^t + b_2)), \\ &W_P \in \mathbb{R}^{D_h \times 2D_h}, b_1 \in \mathbb{R}^{D_h}, U_1 \in \mathbb{R}^{D_h \times D_h}, b_2 \in \mathbb{R}^{D_h}, P^{axioms} \in \mathbb{R}^{C \times D_h} \end{aligned} \tag{3}$$

and W_P, b_1, U_1, b_2 are updated.

The model structure is illustrated in in Figure2.

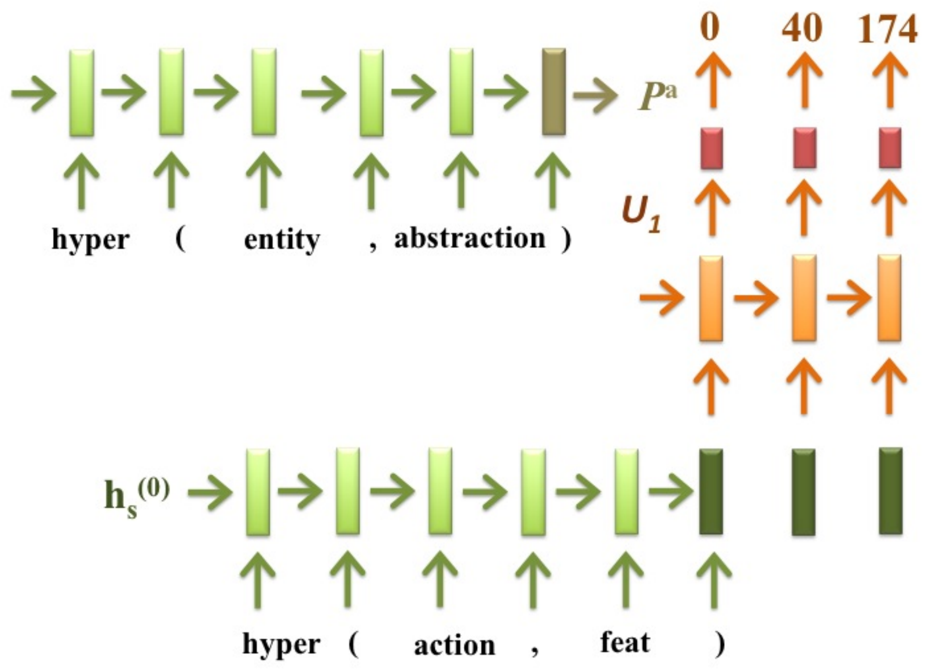

Figure 2: Dynamic Double Seq2Seq RNN

Note that the axioms and goal theorems are sentences with variable lengths, so I left-pad those sentences for batch training. After the padding, all the inputs have the same length.

3.3 Control condition of Dynamic Double Seq2Seq RNN

For the control condition of Dynamic Double Seq2Seq RNN, I replace the sentence embeddings of axioms, P^{axioms}, with an ordinary weight matrix U_3. The RNN$_P$ in this model is described by the following equations:

$$\begin{aligned} &\alpha_P^t = softmax(U_3 \tanh(U_2 h_P^t + b_2)), \\ &U_2 \in \mathbb{R}^{D_h \times D_h}, U_3 \in \mathbb{R}^{C \times D_h} \end{aligned} \tag{4}$$

and W_P, b_1, U_2, U_3, b_2 are updated.

4 Experiments

4.1 Dataset

I generate an axiom list from WordNet [11]. It includes only three binary predicates, “hypernyms”, “hyponym” and “meronym”. The axiom list is formatted in first-order logic and it contains two types of axioms. The first type is Specific Axiom. It contains examples like “hypo(perceiver, somebody) ”, “hyper(reflex, blink)” and “partOf(windscreen, plane)”. The second type is Core Axiom. It contains relationships between the predicates. There are only 3 core axioms: “ all x y z .((hyper (x , y) & hyper (y , z)) $\rightarrow$ hyper (x , z)) ”, “all x y z .((hypo (x , y) & hypo (y , z)) $\rightarrow$ hypo (x , z)) ”, “all x y z .((partOf (x , y) & hyper (y , z)) $\rightarrow$ partOf (x , z)) ”. Each axiom in the list is assigned a number as its index.

Experiment 1-3. There are 8003 axioms in total. I first examine all the axioms to see if any two of them can prove a new statement. For instance, “all x y z .((hyper (x , y) & hyper(y, z)) $\rightarrow$ hyper(x , z))” (index: 0), “hyper(action, accomplishment)” (index: 40) would prove “all z .((hyper (action, accomplishment) & hyper(accomplishment, z)) $\rightarrow$ hyper(action , z))”. These newly proved statements are treated as lemmas. I then examine all the lemmas to see if any one of them paired with one axiom in the original axiom list leads to a new statement. For instance, in the above example, the lemma “all z .((hyper (action, accomplishment) & hyper(accomplishment, z)) $\rightarrow$ hyper(action , z))” paired with the axiom “hyper(accomplishment, feat)” (index: 176) would prove that “hyper(action, feat)”. These proved statements become the to-be-proved theorems in my dataset. In other words, each training example is a [goal (X), proof (y)] pair, where goal statement (X) is given and the proof (y) is a sequence of axioms the model needs to predict. The example above would be written as [goal: “hyper(action, feat)”, proof: (“all x y z .((hyper (x , y) & hyper(y, z)) $\rightarrow$ hyper(x , z))”, “hyper(action, accomplishment)”, “hyper(accomplishment, feat)”)]. In practice, I use the axiom indices rather than the sentences to facilitate training. Therefore, a training model can be written as [X: “hyper(action, feat)”, y: (0, 40, 176)]. Note that in the current dataset, I only consider two-step proof, i.e., you first select two axioms from the axiom list, get an intermediate result (a lemma), and then you combine the lemma with a third axiom selected from the list to prove the goal. The model only needs to predict the three axioms in the correct order, and does not need to generate the lemma.

Using the method mentioned above, I generated 2313 goal-proof pairs in total, including 1850 training examples, 231 validation example and 231 test examples. This dataset are used for experiment 1-3, which test Static Double Seq2Seq RNN, Dynamic Double Seq2Seq RNN and the control condition of Dynamic Double Seq2Seq RNN, respectively.

Experiment 4. In experiment 4, I further test the power of Dynamic Double Seq2Seq RNN. Particularly, I want to know if it generalizes not only to unseen theorem but also unseen axioms. Therefore, I generate 8000 axioms for each conditions (training, validation and test). Each axiom list is used to generate its own theorems. In other words, in validation and test dataset, not only the to-be-proved theorems are new, but also the axioms are not seen in the training phase.

4.2 Evaluation metrics

When the model outputs the axiom sequence, the axioms are treated atomically. In other words, the model only needs to predict the indices of the axioms. Using the above example, the model needs to take the sentence "hyper(action, feat)" as its input and output (0, 40, 176). I use cross-entropy between the model prediction and the target sequence as the loss function and minimize the cross-entropy error. In Results session, I will show the accuracy of model predictions on training data and validation data throughout the training.

5 Results

As can be seen in Figure 3, the Static Double Seq2Seq RNN quickly overfits the training dataset, but the validation accuracy stays at 51% ("No Axioms" panel). The test accuracy is 24%.

However, Dynamic Double Seq2Seq RNN have much better performance than Static Double Seq2Seq RNN ("Axioms Seen" panel). The validation accuracy are around 77% and the test accuracy is 72%. This indicates that multiplying the output of RNN_P with P^{axioms} dramatically enhances the network's ability to prove the theorems.

Replacing the P^{axioms} with an ordinary weight matrix does not have the same boosting effect ("Axioms Seen - control" panel). Its validation accuracy stays at 51% and its test accuracy is 40%.

Finally, when the axioms are not seen ("Axioms Unseen" panel), the validation accuracies are much lower (6%). However, given that the axiom list contains 8003 axioms, it is still above chance. In addition, when we look at the accuracies for individual axioms, rather than the axiom sequence as a whole, the test accuracy is 33%, meaning that a third of the times it selects the right axioms.

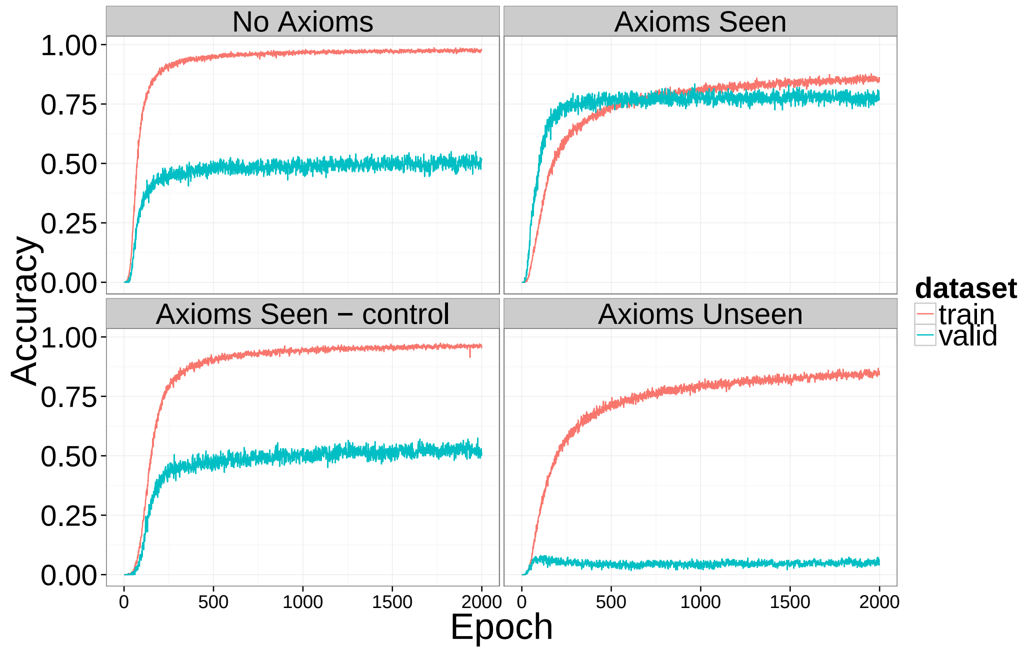

Figure 3: Accuracy on training set and validation set.

5.1 Error Analysis

To understand the lower accuracy in Experiment 4 ("Axioms Unseen" panel), I did an error analysis. It shows that the model often picks up the first axioms correctly, i.e., the core axioms, but fails to pick up the correct ones for the second and the third. This is due to the fact that both the axioms and theorems are unseen in the test phase, which contains many unknown words. The model does not have any meaningful embeddings for those Specific Axioms, thus it can not select the correct ones.

6 Conclusions

In the current project, I build a neural theorem prover and show that deep neural networks can generate proofs for given theorems, which is traditionally solved using symbolic and rule-based approaches. Particularly, I demonstrate that a neural network which dynamically encodes the content of the axioms performs better than the neural network which does not do so. This improvement can not be attributed to more layers or more parameters. The current study provides a novel approach to performing automated theorem proving with neural networks.

For future directions, it would be interesting to see how the input lengths and the proof lengths influence accuracy. I expect that increasing proof lengths and input lengths would result in lower performance, though the degree of performance drop might be different.

In addition, it is worth probing how the choice of recurrent neural network cells influences performance. It is likely that long-short-term memory (LSTM) neural network would perform better than basic recurrent neural network [12].

References

[1] T. Barnickel, J. Weston, R. Collobert, H.-W. Mewes, and V. Stümpflen, "Large scale application of neural network based semantic role labeling for automated relation extraction from biomedical texts," *PLoS One*, vol. 4, no. 7, p. e6393, 2009.

[2] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[3] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

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

[5] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

[6] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[7] S. R. Bowman, "Can recursive neural tensor networks learn logical reasoning?," *arXiv preprint arXiv:1312.6192*, 2013.

[8] S. R. Bowman, C. Potts, and C. D. Manning, "Recursive neural networks for learning logical semantics," *CoRR, abs/1406.1827*, 2014.

[9] A. Neelakantan, Q. V. Le, and I. Sutskever, "Neural programmer: Inducing latent programs with gradient descent," *arXiv preprint arXiv:1511.04834*, 2015.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[11] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

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