Michael Gump

6.S063, Special Section on Artificial Intelligence

Professor Bob Berwick

Generation within Statistical Systems

A Study of Recurrent Neural Networks

Artificial neural networks are often criticized for their propensity towards over-training and their failure to generate past their training set. Neural Networks can often become over-specified and avoiding this is a major challenge for those developing learning models. In contrast humans, in particular young humans are able to extrapolate from small data and learn overarching patterns quickly and reliably. In this paper we will investigate the generative capabilities of current open-source deep learning models when applied to rule-learning.

I implemented an RNN over a dataset of simple arithmetic problems that I complied to simulate the corpus a young child might receive when learning mathematics from a parent or teacher. I reduced the dataset of text samples to a series of sequential "timesteps" or vectors represented as sparse vectors over our vocabulary. The model receives a series of input vectors to simulate the first half of an equality and is then asked to predict the next "word" in the sequence, or in other words, solve the math problem. This method is fundamentally different than how we usually expect computers to learn math. While this seems overly verbose when applied to arithmetic, computers normally approach math through "expert systems" in which the computer follows explicit rules in response to an input. Is it possible to apply a connectionist system to arithmetic and if so what would be the advantages and disadvantages of this? Could this system of learning be applied to more complex problems and allow it to abstract in a way only humans are usually capable of?

Introduction

"How do children learn?" is a fundamental question of neuroscience, linguistics, and artificial intelligence. Children seem to, with extremely limited input, be able to learning apply complex rule systems seemingly without prior knowledge of these rules. One possible explanation to this puzzle is that children do in fact have knowledge of some of these rules. Chomsky's universal grammar postulates that innate neurological structures are responsible for the fundamental parameters of language. With this "knowledge" children can learn the complex syntactical and phonological rules of any language in a much reduced search space. Across nature's diverse list of creatures we see examples of reliable and quick learning while under a poverty of stimulus. Some of these examples might show that structures like universal grammar enable learning while others might show that nature's complexity allows it to learn complex problems through brute force. This paper will address what sorts of problems can be solved without an initial underlying structure and which can not. Specifically I will address how abstract concepts such as math can or could be learned.

Initially, I will show that given a small dataset we can train a connectionist recurrent neural network model to learn all of the equalities I show it. For example a corpus such as {'1=1','2=2','1+1=2'…. '1+1+1+1=4'}. Then we will reduce the corpus and see if the model can extrapolate to solve inequalities that it has not been shown. Investigation will attempt to determine when a model can extrapolate and generate rules to solve problems it has not been taught directly and when the problem is too hard for our current model.

The Model

Theano is an open source python deep learning library developed by a group within the University of Montreal. It is considered standard within the deep learning community. For this project I implemented a recurrent neural network based off a collection of blog posts and research articles. A full bibliography is presented at the end of this paper. Recurrent Neural Networks are often used when attempting to train generative models, be it for text, audio, or images. There are a number of advantages to RNNs that allow them to more strongly learn a corpus and generate mimicking output.

Ordinary neural networks forward propagate an input state through a series of hidden layers until it is finally transformed into the desired output state. Critically, each layer of hidden nodes is only connected to the more forward layers, preventing the network from remembering context and making standard neural networks a poor choice for analyzing sequential datasets. In a recurrent neural network, the hidden layer from past states within a sequential dataset is connected to future hidden layers, allowing the model to remember contextual data and more accurately predict future states. Another feature that can drastically improve the efficacy of RNNs is Long-Short Term Memory nodes or LSTM nodes. LSTM nodes provide more control to the network for remembering patterns through time. A problem with standard RNNs is that it is computationally intensive to obtain the proper weight matrices to relate past and future events. Therefore it is difficult to learn the relationship between past and future events that are arbitrarily large or small distances from each other. LSTMs add another layer of complexity to an RNN by allowing their weight matrices to control how often information flows in or out of the node. By making the nodes tunable through the learning process LSTM networks outperform standard RNNs and provide more interesting results.

Investigation

First, consider what is necessary to teach someone arithmetic. Small children usually learn through their parents asking them how many cookies are on a plate and as the line of questioning becomes more advanced they can learn the rules behind addition. However, what are the rules behind addition? We could say that the only rule is the formal math definition of the addition operator. However these are clearly not the rules that small children are learning when they learn to add. First the child learns the numbers, {'here is one thing','here is two things','here is three things'}. But then the child must learn that not only is 'two' when there are two things but when you have one thing and add one other thing to it. This distinction is important because it gives the child information about how you create 'two' or 'three'. An important question to ask is what rules has the child learned? The child probably couldn't come up with a formal definition of addition nor could the child come up the associative or commutative property. But I would claim that some underlying knowledge of these rules is necessary to learn what the addition operator does. For example if the child has seen {'1=1','1+1=2','2+1=3','3+1=4'} I don't think it could reliably learn '2+2=4' without seeing the example {'1+1+1=3'}. Even though the child has the definition of '4' it hasn't seen any examples of the transitive property and therefore doesn't actually have a complete picture of what ‘addition’ or how to solve problems involving addition. What examples then are necessary to teach someone addition and which examples are best?

Initially, we see that the model learns a complete set of examples very quickly and reliably. Initially I trained my model on all the addition problems involving digits less than five. It had a vocabulary of 9: [‘1’,’2’,’3’,’4’,’5’,’+’,’=‘,’SENTENCE_START’,’SENTENCE_END’]. The last two are to ensure that the program has some structure in its learning. My first real test for my system was to remove 10% of the 31 initial examples. I tested three different sets:

[['3', '+', '1', '=', '4', '.'], ['4', '+', '1', '=', '5', '.'], ['3', '+', '1', '+', '1', '=', '5', ‘.’]]

[['1', '+', '1', '+', '3', '=', '5', '.'], ['1', '+', '1', '+', '2', '+', '1', '=', '5', '.'], ['2', '+', '1', '+', '1', '+', '1', '=', '5', ‘.’]]

[['1', '+', '1', '=', '2', '.'], ['1', '+', '4', '=', '5', '.'], ['2', '+', '1', '+', '2', '=', '5', ‘.']]

In all these examples the model learned the complete 31 problems with 100% accuracy and no fall off of accuracy even after I trained well past it converged on the training set (The 28 given problems). In other words in converged quickly to a solution on the 28 givens problems that allowed it to solve the complete set of 31 problems. Then I increased the removal to 20%:

[['2', '=', '2', '.'], ['1', '+', '2', '=', '3', '.'], ['1', '+', '3', '=', '4', '.'], ['3', '+', '2', '=', '5', '.'], ['1', '+', '3', '+', '1', '=', '5', '.'], ['2', '+', '1', '+', '2', '=', '5', ‘.']]

[['1', '=', '1', '.'], ['2', '=', '2', '.'], ['2', '+', '1', '=', '3', '.'], ['2', '+', '3', '=', '5', '.'], ['4', '+', '1', '=', '5', '.'], ['1', '+', '1', '+', '1', '+', '1', '=', '4', ‘.']]

[['5', '=', '5', '.'], ['2', '+', '2', '=', '4', '.'], ['2', '+', '3', '=', '5', '.'], ['4', '+', '1', '=', '5', '.'], ['1', '+', '1', '+', '2', '=', '4', '.'], ['2', '+', '2', '+', '1', '=', '5', '.']]

This made it a bit harder for the computer. It struggles especially on equalities where multiple similar problems had been removed from it’s example set. It also became more prone to overtraining as we might expect. But it did with consistency reach 29-30 correct equalities out of the 31. For n=5 which is relatively small the accuracy fell off surprisingly little. At 40% of the set removed the model averaged 27/31 over three tests, getting eight more than it was shown and six more than it should have if it guessed randomly on unseen examples. At 60% removed the model did begin to fall off significantly and only learned around 17/31 or about five more equalities than it was shown. One interesting finding of note is that the ones the model got wrong were not just the equalities it was not shown. Instead it would fail the equalities in chunks. If one equality out of five similar equalities were removed it would learn all five but if three were removed it would fail the entire chunk. At all points however the model extrapolated to more than the number of equalities it was shown. Even better than that the model extrapolated past the number of equalities it would haven gotten if it had simply guessed from its vocabulary on equalities it was not shown. This shows that at the very least the model is learning rules past its training set even if these rules are not the rules a child would learn.

To take this investigation a step further I began training the model on complete sets up to n digits and then showing it one example past n. I learned however that some examples are not helpful for the computer. For example, when I showed it [‘1’,'+','1','+','1','+','1','+','1','=','6'] it mislearned the lesson I was trying to teach it and actually started reliably converging to another “lesson”. It classified all additions adding up to 5 or more as five except for {‘1’,’+’,’1’,’+','1','+','1','+','1','+','1','='} and {‘1’,’+’,’1','+','1','+','1','+','1','='}, which it classified as 6. In other words, from the extra example it was shown it did not learn the value of 6 in relation to the addition rule it was learning, but instead learned “If something has a lot of ones in a row it’s a six”. The next obvious step is to show it better examples so that it can get a better picture of how to define ‘6’. I showed it the following:

[[‘1’,’+’,'3','+','2','=','6'],['6','=','6'],['4','+','1','+','1','=','6'],['2','+','2','+','2','=','6'],['2','+','1','+','3','=','6'],['1','+','1','+','2','+','1','+','1','=','6'],['2','+','4','=','6'],['2','+','1','+','1','+','2','=','6']]

[[‘1’,’+’,’3','+','2','=','6'],['6','=','6'],['4','+','1','+','1','=','6']]

[[‘1’,’+’,’3’,’+','2','=','6']]

As we might expect on all examples the model was extremely susceptible to overtraining. Training for the same number of iterations as we did in the initial investigations led almost invariably to the model getting all 16 of the 16 additional problems wrong. But if we tested on the validation set early the model was able to get around half of them correct usually. What’s interesting is that showing the the model more examples adding up to six did not improve the number it got right. Showing just one example the model got 9/16 of the new examples (8/15 out of unseen examples) and showing it 3 or 7 examples to it did not improve this number. This indicates that while only one example was necessary to maximize (in relation to this model) it’s understanding of ‘6’ its maximum understanding of ‘6’ was only 50% accurate. About 50% of the examples that should have been six were incorrectly predicted as five.

An investigation of examples with larger numbers was not realistic because it was too computationally intensive to generate hundreds of thousands of examples. Additionally, increasing the largest number even by just a few seems to drastically increase the training time required for the model to learn simple relations.

Discussion

Firstly, I will address a system that could improve upon our model. One major problem is that the system has not reinforcement on whether it is learning the correct rules or overtraining. When a child learns they often ask questions and receive reinforcement as to whether their guesses are correct or not. This model has no such reinforcement, when it guesses it only gets to guess once. A system in which the model was trained on a limited dataset but was allowed to propose guesses once it had reached a certain confidence about a problem it did not know the answer to could reduce the risk of overtraining and push the model towards a more complete solution. While I don't know how such a model would be designed I could imagine training a model with 60% of a complete addition set up to 10 (1023 examples) and reserving the rest for reinforcement if the model reached a high confidence on any of the reserved equalities. I would imagine that this would be more similar to how a child would learn and it would allow the learning process to be more directed while at the same time showing the model fewer example explicitly.

When we compare the model's results between when it was given a subset of a full training set with the model's performance when given a full training set and a few examples of the next number we can clearly see that the model is better at learning when the problems that are withheld are distributed across its vocabulary. This suggests something that seems intuitive. When we teach children to add we don't show them every possible addition using small numbers and then define all the other numbers for them. We show them variety of examples spread throughout the corpus we want them to learn. This appears, empirically, on both children and my model, to be more conducive to real learning and extrapolation. Furthermore, the model appears to be more likely to overlearn its training set when it is given all of the examples. When it is given only a portion of the final set it tends to learn the rules much better then it would otherwise. This indicates how example picking could drastically effect a model's ability to learn.

Is the model learning to add? Empirical evidence that the model takes exponentially longer to train as the largest number in the dataset increases suggests that it is not picking up on underlying rules like we would expect a human to rather quickly. If the model had been as quick to learn addition with numbers up to 8 as it had been with numbers up to 5 that would indicate that the model is learning a pattern that is independent of the size of its vocabulary. Since this is not the case the patterns it is learning are likely often just convenient approximations for its training set. In some cases it does seem to be learning real lessons but often these are not exactly the lessons we would hope it would learn. In the case of giving it only the 1+1+1+1+1+1=6 example it didn't learn to add but learned that 'six' meant 'lots of ones'. This could be similar to how a small child who only knows a few numbers might just call all large quantities 'Eight!' (or some other number that seems large to the child). In the second case of showing the model more representatives examples for six it seemed to learn that large sums were simply a tossup between 'five' and 'six'. This might be a result of over training but I think it's more likely that this is just the best my model could do.

Conclusion

Could we have taught the model to learn how to add simply through examples? I would imagine that instruction such as 'addition is when we put things together' is critical to a child's education. Since this is not something I could give my model that might explain the challenges it had with learning real patterns. Clearly a fully connectionist approach does not seem to work when applied to abstract rules such as addition. There are too many hidden variables that allow models to 'mislearn' and not pick up on the true underlying patterns. An approach with more direct reinforcement and the ability to direct the learner's attention seems to be key. Our model, along with most neural network models only reinforces at the example level. Reinforcement over which lessons the model is learning could push the model towards a better 'understanding' of the problem.

Works Cited

Britz, Denny. "RECURRENT NEURAL NETWORKS TUTORIAL." Blog post. WILDML. N.p., 30 Sept. 2015. Web. 22 Dec. 2016.

Karparthy, Andrej. "The Unreasonable Effectiveness of Recurrent Neural Networks." Blog post. Andrej Karpathy Blog. N.p., 21 May 2015. Web. 22 Dec. 2016.

"Understanding LSTM Networks." Blog post. Colah Blog. N.p., 27 Aug. 2015. Web. 22 Dec. 2016.