Report: Implementing Neural Programmer-Interpreters on travelling salesmen problem

Cheng Li

Cli113@syr.edu

Abstract:

From the original paper, neural programmer- interpreters was introduced as an brilliant ways to use an algorithm to teach computer imitating programmers to write its own program. This algorithm was impressed on large scaling problem, which means the computer can deal with more hard work with simple training data. From the application guideline, I think traveling salesmen problem is very a good test to verify this idea. To write the whole model, at first a demo without any algorithm to solve travelling salesmen problem was set. Then the demo was expanded as a function of environment to generate training data. The state of environment firstly was encoded and then transfer to long short term memory to be trained. Finally the output was divided to three branches to compares verify the result. The problem of travelling salesmen is about how to find a shortest way to pass all cities, so the output of the algorithm should be the best plan of path. However, even though the encoder was used, the scale of training data was too large and computer was crashed on this problem. Also the structure introduced in original was not good enough in nested program, which is one of the main reasons that execution trace was much larger than expected one. Therefore, neural programmer-interpreters may be is good for simple programs with small execution trace, but it is not suitable and limited to solve multitasking and nested problem.

Introduction:

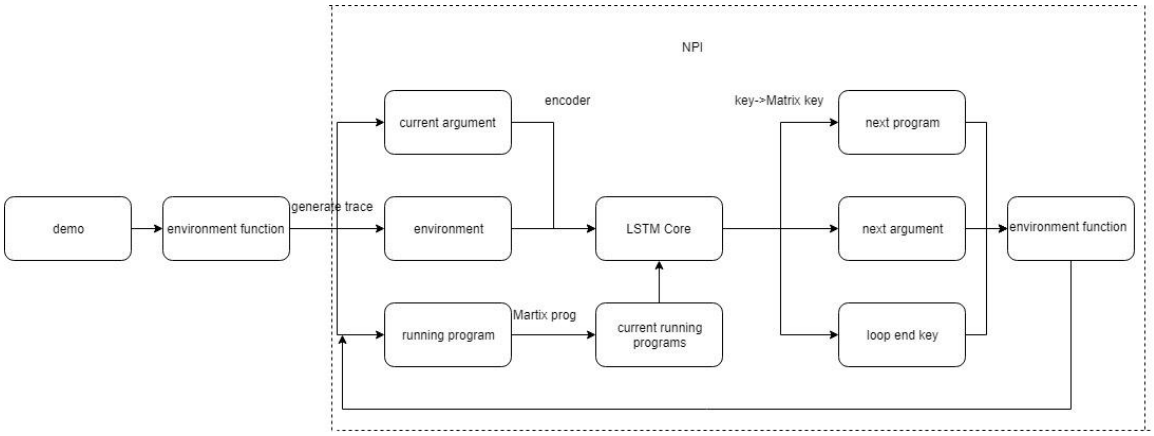

Figure 1 NPI structure

Architecture:

As shown in Figure 1, the first step to transfer any program to NPI is dividing all subprograms into small function, which is environment function in the second block. In this environment function, every function needs to be labeled and be recorded on it execution trace. From execution trace, current argument, environment and running program are generated. Current argument and environment parameters are encoded, because environment parameters are about 1600 in scale in each state and such large inputs will make LSTM training very slow. LSTM Core was form Tensorflow package and not

built by myself, because it is not the main problem in this project. For generating training data, current argument, environment, running programs, next program, next argument, current running program and loop end key are extracted from execution trace. NPI simplified model is shown in dot line zone in Figure 1, which is similar to training data and added an environment function to make a loop to run.

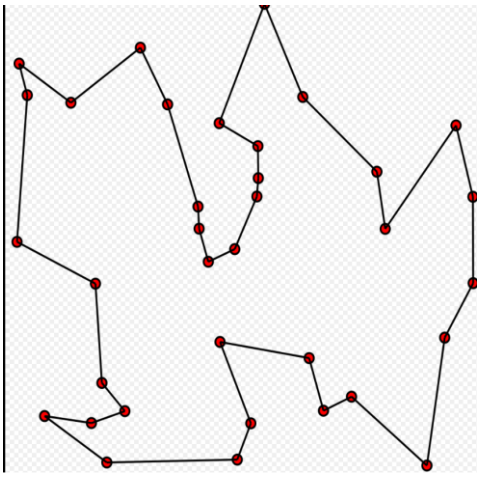

Figure 2 TSP problem

Travelling salesmen problem:

As shown in Figure 2, travelling salesmen problem is about generate a shortest path to pass all cities. Cities are simplified as red points in Figure 2. These cities will be randomly generated in this model, so that the best plans of path are different every time. As the growth of the number of cities, the total number of possible path plan will grow exponentially.

The reason why I chose NPI and travelling salesmen problem as an combination is that I think NPI maybe very interesting. When I read NPI paper first time, I think it may be the demo of any self-learning robot in the future, because it seems need a very large scale of data to be trained and can solve more complex tasks than expected, which is similar to neural program that was limited on calculation ability before. Traveling salesmen problem is different form any example given in papers, since execution trace of travelling salesmen grows exponentially but other does multiply. Therefore, I want to test whether NPI can work in travelling salesmen model which grows faster.

State of art:

In this project, I used structure introduced in Neural Programmer-Interpreters(Scott Reed & Nando de Freitas,2016). This paper explained how NPI is set and to use NPI solve three problems, which are addition, sorting and canonicalizing 3D car models. And the author shows a chart on test accuracy between LSTM and NPI model. The result shows that NPI work better than LSTM. Travelling salesman problem arises in many areas of theoretical computer science and is an important algorithm used for microchip design or DNA sequencing. Since TSP is very complicated when it scales up, genetic algorithm is a popular way to solve it (P Larranaga, 1999). Also, heuristic algorithm can work well with TPS (S. Lin, B. W. Kernighan, 1973). And Pointer Net is a creative way to find approximate best result, which uses RNN and pointer structure (Oriol Vinyals, 2015).

Compared to the research on TSP, we can find that the examples used in NPI are with small scale of environment parameters. In the original paper, the author answer about the advantage of NPI learning execution trace with same scale of environment parameters, but in TSP not only execution trace but also the scale of environment parameters will grow when the problem become more complex.

Link to my code: https://github.com/SImonleecl/Try-use-NPI-solve-TSP

Architecture:

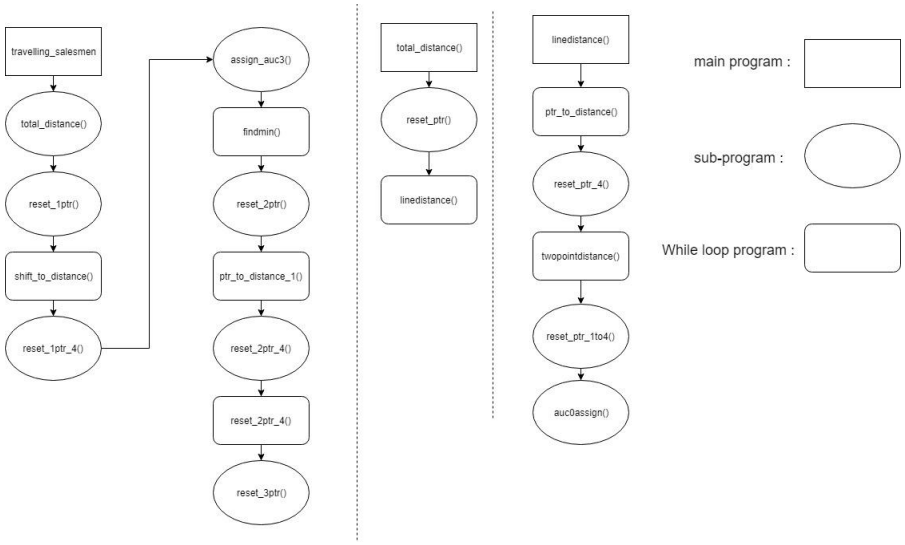

Figure 3 Details of NPI model

Figure 3 shows the architecture of TSP. Total number of program is 19 and current running program 20, because the model need one more key to determine when to stop. In this model, while loop program is determined by end loop key to be out of loop. Considering on encoder implement, some functions like permutation are set as input data to TSP model. It may be improved by better structure of model, because I hope every preset input should be a part of TSP.

Figure 4 shows the architecture of NPI, and Figure 5 shows detail structure for NPI. This model is a little different from the original one. In original version, all running program parameter should be calculated by a matrix and current running program key. The matrix is not necessary in problem and I simplified input to all running program parameters to train, because this matrix is fit for multitask problem and TSP just havs one task. Also, I used program key vector directly instead transferring to current running program parameters by a matrix. The reason why omitting the matrix is same as mentioned before that these two matrix is set for multitask models.

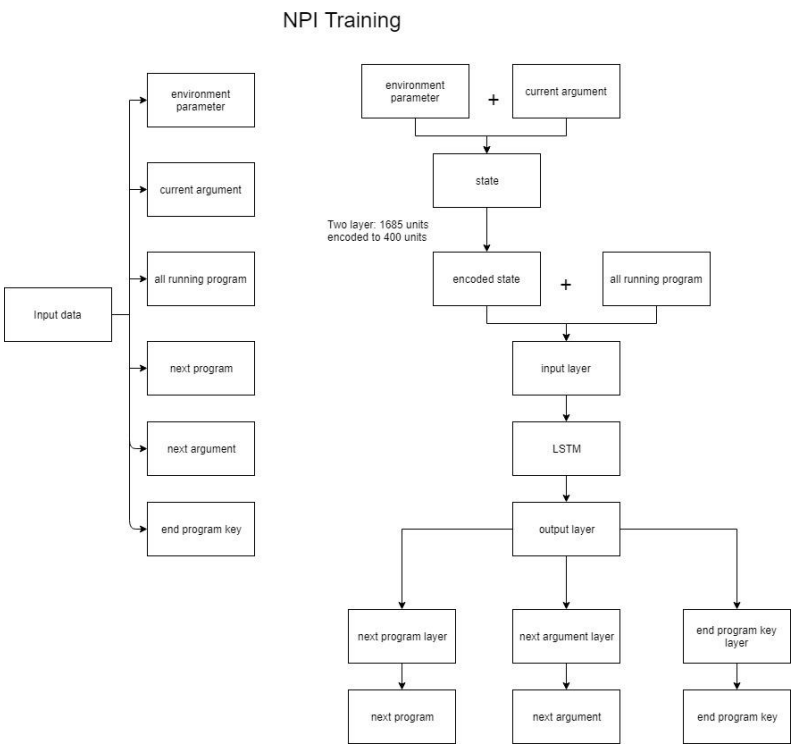

Figure 4 NPI training

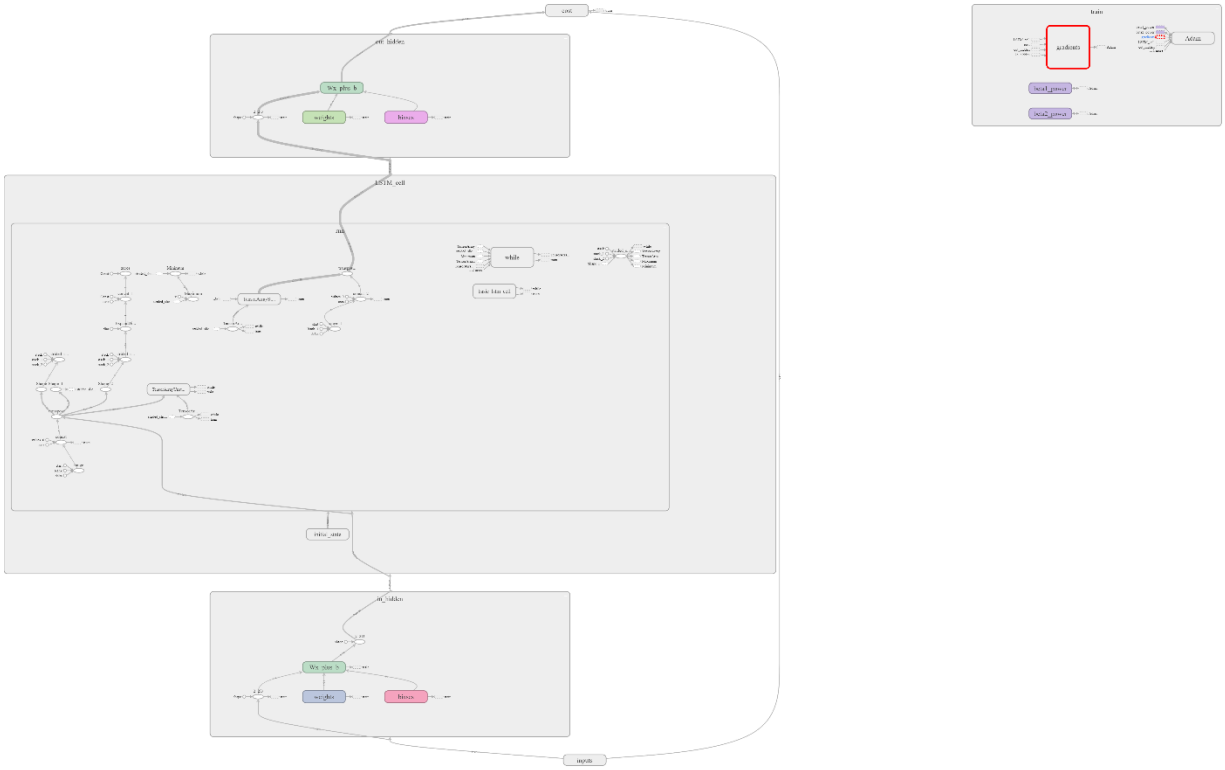

Figure 5 Training model from Tensorboard

Equations of update rule:

Programs in TSP data generating model
1: Inputs: Environment parameter *e*, current program *i*, all running program *p*, current argument *a,* stop threshold *b*
2: record(*e, i, p, a*)
3: **while** r < b **do**
4: main function()
5: output(*e2,i2,p2,a2*)
6: define(end of program key *r*)
7: record(*r*)

Neural programming inference training
1: Inputs: Environment parameter *e*, current program *i*, all running program *p*, current argument *a,* end of program key *r*, next program *i2*, next argument *a2*, encoder
2: *h*<-0, *r*<-0
3: *s*<- *encoder*(*e, a*), *h*<- flstm(*s, p, h*)
4: *h1, h2, h3<-h*
5: *r*<-sigmoid(*h1*), i*2*<-sigmoid(*h2*), *a2*<- relu(*h3*)

Neural programming inference
1: Inputs: Environment parameter *e*, current program *i*, all running program *p*, current argument *a,* stop threshold *b*
2: function RUN(*i, a*)
3: *h*<-0, *r*<-0,p<-fprog(*i*)
4: **while** *r< b* **do**
5: *s*<- *encoder*(*e, a*), *h*<- flstm(*s, p, h*)
6: *h1, h2, h3<-h*
7: *r*<-sigmoid(*h1*), i*2*<-sigmoid(*h2*), *a2*<- relu(*h3*)
8: **if** *i* == ACT **then** *e* ← fenv(*e, p, a*) .
9: else RUN(*i2, a2*)

My track is novel application track. Compared to original paper, some structure in TSP need to be adjusted. Although I do not totally finished the project, during building NPI model for sub-program, I found that my problem is more difficult and complex. So I improved the low boundary of sub-program. For example, assign function is same level in original paper as assign_all_pointer_zero in my model. And I also find that environment parameters and ACT are very different in various problems, which means that there are not enough examples and the model must be build from starting. Another problem I find that BasicLSTM from Tensorflow cannot support training multiple outputs, so change the output layer of LSTM which includes *r*, *i*2, *a*2. During building the model, I also find that some functions and matrix are very complicated to build and not necessary for some sub-program, so I simplified them.

In training procedure, there are two parts of it. The first part is encoder. I uses two dense layers as encoder and two dense layers as decoder, and Adam optimizer was used. Cost function of encoder is the difference between inputs and outputs .Training data size is (100 ,434 ,1685), where is (batch size, step

size, state size). Another part is NPI model without encoder, since state size is 4. In this part, I simplify environment parameters as integer before training. LSTM cost function is difference between output trace and predicted trace, where weight is set as one to every elements because environment parameter was simplified before. To speed up training procedure, Adam optimizer is also used. One thousand batches of data are training data and one hundred batches of data are testing data.

Experimental tasks:

I am one member group. Here is a description on problem tasks in my proposal:

1. Use simple action modules, like Addition, Shift, Assign, Save to simulation every step in TSP
2. NPI models trained by simple training data can run in more complex tasks. In this problem, it means that NPI models, trained by small numbers of cities, can run in large numbers of cities.

In TSP problem, there are many small module in main module. Here I will show how to compute distance from an ordered list of cities. In the first step, I will use a function called " Find position store distance() " to find the position store distance value, since when cities' number changes the length of environment data also changes. Then I use a function called "Compute distance()" to compute a ordered list of cities on path length and store the result. One of sub-program called " two cities' distance (e_0)" calculate path length of two cites, where e_0 is the pointer of first city and e_0 +1 is the next one.

List Distance Module	
Environment data:[[point A],[point B],….,0,0]	Last second zero stores distance value
Current argument: a_0, a_1	
Environment parameters: e_0	
Running program: *i*	
1: Running(List Distance Module):	
2: Find position store distance():	
3: while:	
4: $a_0 = a_0 + 1$	a_0 is the pointer of last second element
5: Compute distance():	
6: while:	
7: a_1 = two cities' distance (e_0)	e_0 is the pointer of a cities
8: store(a_0, a_1)	

Dataset in NPI is mainly divided into two parts, input trace and output trace. Input trace has three parts, which are environment parameters, current running program and current argument. Output trace also has three elements, which are next running program, next argument and key of end of program. Environment parameters are extracted from environment data, because latter one is very large and not all element are necessary for certain program. My dataset is online generating, because it is very large up to 1.6G in 100 branches in main program. So in this experiment part I use a subprogram as an example to show. In this example, training data is 50 branches, 22 time steps and running times is 200. Training data size are various, because I use 2 to 5 cities as examples for training. To make it easier for training, I expand all training data to same time steps, so after expansion all training data has same size.

Results:

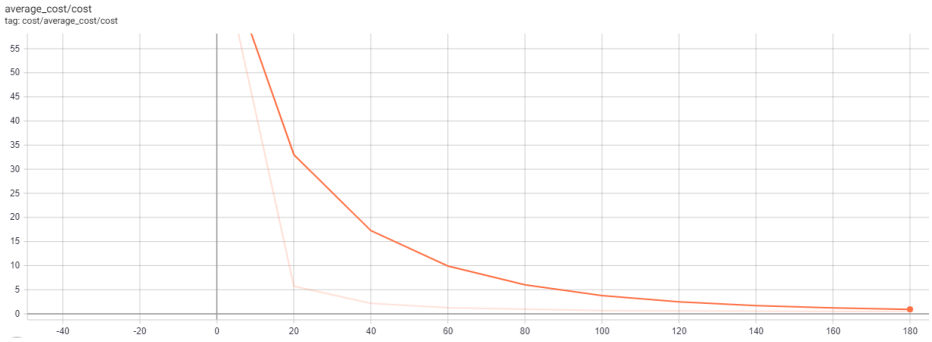

Figure 6 Learning rate

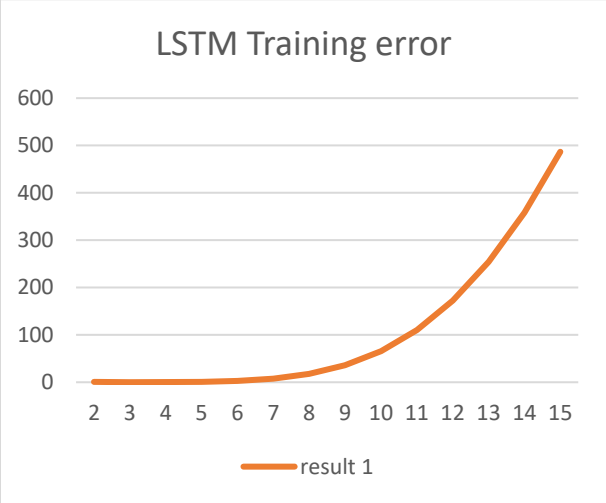

Figure7 LSTM Training error

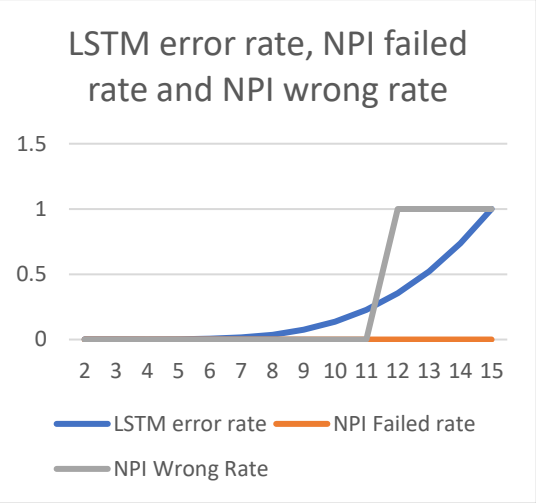

Figure8 Comparison of LSTM and NPI

Figure 6 shows learning rate of NPI, its training error is about 0.5, which is same as #2 to #5 data in Figure 7. Figure 7 shows Testing error about output trace without NPI model. Figure 8 mixes LSTM error rate, NPI failed rate and NPI wrong rate in graph so that we conclude that even though LSTM error raise a lot, NPI still can run in some complex model with high accuracy. NPI shuts down when the number of cities is 11. I find that NPI jumps out of model before right time, so the problem may be the threshold of R is too high.

Rerun:

In rerun part, I set learning rate as 0.1, 0.03, 0.01, 0.003 and 0.001, and Figure9, Figure 10 and Figure 11 shows the result. From these result, we can conclude that the model with 0.01 leaning rate has best result in training error, which means that the model need more data when learning rate become smaller. Furthermore, we can see that although the models with smaller learning rate perform better in same-sized better, the models with larger learning rate do better in LSTM error and NPI accuracy test.

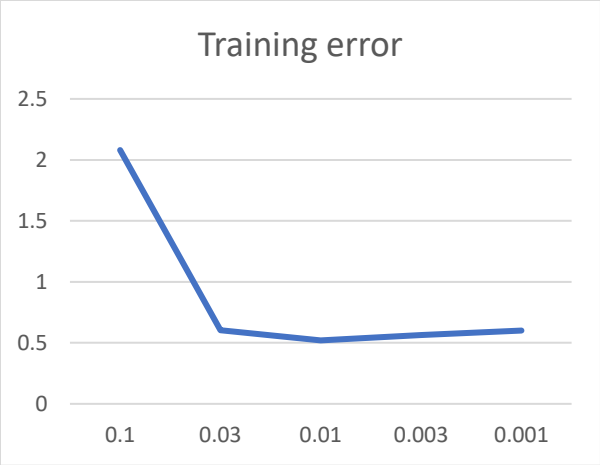

Figure 9 Training error

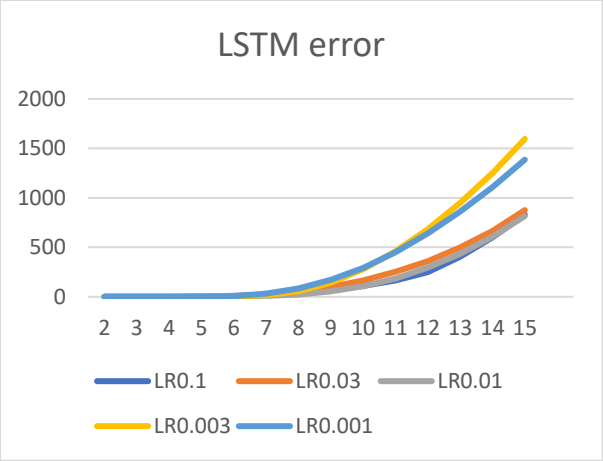

Figure10 LSTM testing error with different learning rates

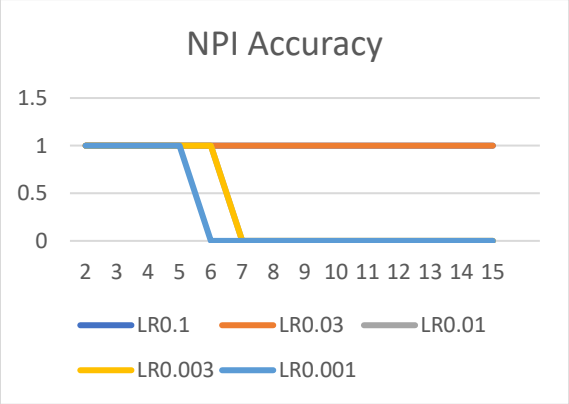

Figure 11 NPI accuracy with different learning rates

Ablation study:

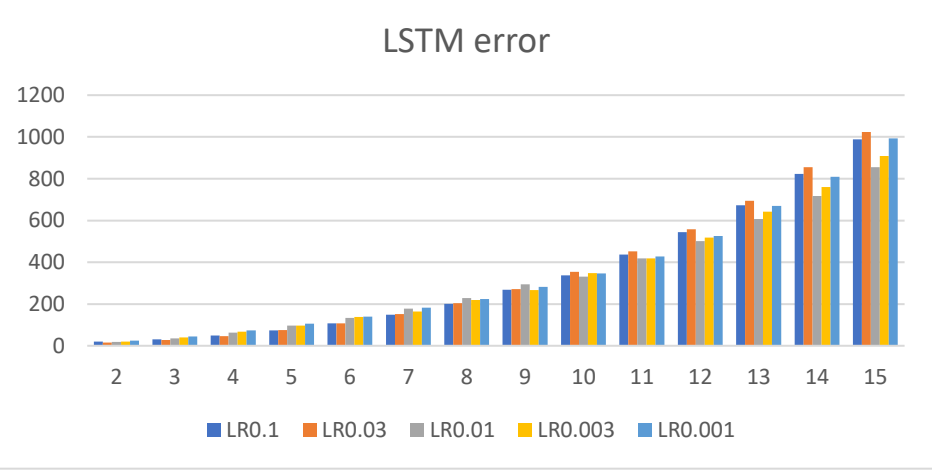

Figure 12 Ablation study of LSTM testing error in different learning rate

Figure 12 shows the result of ablation study. In this test, I shuffled data of input trace and output trace and noise to every element with the range of 0.01. Compare to Figure 9, there is two conclusions. The first result is that when testing in same-sized problem the model with larger learning rate do better. However, when testing in different-sized problem, the trend of LSTM error is similar to training error.

Discussion:

In this project, I try to use NPI model to solve TSP problem. Firstly, I built a demo without NPI model on TSP problem, and expanded this model to NPI architecture. Secondly, encoder was built to compress state parameter and I tried to build LSTM training part, but it is failed. Thirdly, I learnt some experience from failure and built a NPI model for a sub[-program of TSP. Finally, new model of this sub-program worked and I did some experiment for this model. In the experiment, it concludes that even though LSTM was trained good in same-sized problem, a large scale of training data is important to NPI model work better. If the scale of training data is limited, the model with larger learning rate may do better in different-sized problem instances than the one with small learning rate. In ablation study, it shows that the models with smaller learning rate do better in same-sized problem instances, but the models' result is similar to their training error in different-sized problem instances .

The most important result is NPI model can help problem run in more complex problem with simple training data. Even though LSTM error increase a lot, NPI model still can run in some instance with high accuracy, which means that NPI is better than simple LSTM model.

The most challenge parts of the project is how to decide the size of sub-program. NPI is more like assembly language than python. Therefore, TSP will be very complicated if the demo model is expanded totally in assembly language. So sub-programs should fuse some elementary programs. However, if sub-programs are too complicated, environment parameters will be very large. Environment parameters are extracted form environment data and size of environment parameters is depend on sub-programs, because they just need to store enough information for sub-program. One of the reasons my first model is that my environment parameters is very large and training data is up to 1.6 Gb.

I think my model may be worse than other model in same experiment task. Compare to genetic algorithms, there is some shortages of my models. My model uses brute forced random data, which means that my model is slower and less efficient and just can return accurate result not approximate one. Secondly, my model may shut down for unknown reasons ,but genetic algorithms can work in any situation. Thirdly, my model training data is execution trace, which is more complicated than the one of genetic algorithms.

Ablation study shows an evidence on my one hypotheses, which is that in limit scale of training data, the model smaller learning rate works better in same-size problem instances but not also does better in different-size problem instances. Based on ablation study, I think that the model learns same-size problem and different-size problem together, but it focus more on same-size on. If the learning error of same-size problem cannot decrease any more, the model will continue learning different-size one. This study shows that large data is very important to mu model.

To continue working on this project, the first step is to determine the size of environment parameters in every program, because training data of execution trace is very big and limitation of size of state is very necessary.

Bibliography:

1. Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.
   http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf
2. Reed S, De Freitas N. Neural programmer-interpreters[J]. arXiv preprint arXiv:1511.06279, 2015.
   https://arxiv.org/pdf/1511.06279.pdf
3. Vinyals O, Fortunato M, Jaitly N. Pointer networks[C]//Advances in Neural Information Processing Systems. 2015: 2692-2700.
   http://papers.nips.cc/paper/5866-pointer-networks.pdf
4. Larranaga P, Kuijpers C M H, Murga R H, et al. Genetic algorithms for the travelling salesman problem: A review of representations and operators[J]. Artificial Intelligence Review, 1999, 13(2): 129-170.
   https://link.springer.com/content/pdf/10.1023/A:1006529012972.pdf
5. Lin S, Kernighan B W. An effective heuristic algorithm for the traveling-salesman problem[J]. Operations research, 1973, 21(2): 498-516.
   https://www.jstor.org/stable/pdf/169020.pdf?casa_token=Rt2HJP3bJm8AAAAA:SUYQFubUgXuKVaroQGAnlnG2BBsge7CycsWXC_NCvKuMjp6coievWOialFG4xsLl9SM65ZG1yZlRAWoPqfNqYANcXt2TyvtfchPTsd7fJuwz4jxk_0E