Convergence Algorithm on NPI

Quan Wang

Abstract— **The core of automatic programming is to solve the problem of program synthesis.At present, the neural network is used to solve the problem of program synthesis.Neural program interpreters are based on input and output of training, testing.Learn the rules of the task from the input/output pairs.However, the traditional neural interpreter may not converge during training. In this paper, different convergence algorithms are used to train the neural interpreter in order to find the best convergence algorithm.**

Index Terms—**Automatic Programming; Neural Programmer-Interpreters; Artificial intelligence**

I. INTRODUCTION

The core issue of automatic programming is program synthesis. NPI (Neural Programmer - Interpreter) does not generate code snippets, but learns transformation rules from input and output pairs, and then NPI can implement tasks through these transformation rules.The task of program synthesis is to find the programs needed to satisfy some form of constraint (user intent). Unlike traditional compilers, which translate high-level code into low-level code for machines through semantic translation. Program synthesis usually searches for programs to accommodate constraints in program space. The most common constraints are input and output pairs.

II. RELATED WORK

Gulwani[2] describes an algorithm based on several new concepts that synthesizes the programs needed in the language from input and output examples. The goal of Matej Balog[3] is to train a neural network to predict the characteristics of programs that produce output from input. On the contrary, Ian j. Goodfellow[4] thinks that the main reason for the neural network's vulnerability to hostile disturbances is its linear nature. Eric Price[5] has learned to perform all arithmetic operations (and generalize to Numbers of any length). Marcin Andrychowicz[6] notes that ham-enhanced LSTM networks can learn algorithms for problems such as merging, sorting, or binary search from pure input and output examples. Łukasz Kaiser[7] proposes neural GPU is put forward.It is based on a convolutional gate controlled loop unit, which, like NTM, is computationally generic. The neural machine translation proposed by Dzmitry Bahdanau[8] usually belongs to the encoder decoder family, and the source sentence is encoded as a vector of fixed length, from which the decoder generates the translation. Arvind Neelakantan[9] came up with a neural programmer, a neural network that adds a set of basic arithmetic and logic operations that can be trained end-to-end through reverse propagation. By coupling neural networks to external memory resources, Alex Graves[10] extends the functions of neural networks that can interact through attention processes. Oriol Vinyals[11] introduced a new neural architecture to learn conditional probabilities of output sequences, where elements are discrete markers corresponding to positions in the input sequence. Kevin Ellis[12] proposes an algorithm that USES symbol solvers to effectively sample programs. Gunter[13] proposed a TERPRET model.It consists of a specification that the program represents and an interpreter that describes how the program maps input to output. Chen xinyun[14] has taken an important step in this direction. He proposed a new challenge in the field of program synthesis from the input and output examples: learning context-free parsers from paired input programs and their parse trees.

III. CONVERGENCE ALGORITHM

SGD carries out gradient update for each sample every time it updates. For large data sets, there may be similar samples. SGD only updates once at a time, so there is no redundancy, and it is faster, and new samples can be added.Random gradient descent is updated by iterating through each sample. If the sample size is large, it is possible to use only part of the sample. SGD, because it updates frequently, will cause severe oscillations in cost function.This algorithm can make larger updates to low-frequency parameters and smaller updates to high-frequency ones. Therefore, it performs well for sparse data and improves the robustness of SGD.

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right).$$

Gradient update rules:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i}.$$

Where g is the gradient of parameter given at time t
If it is a normal SGD, then the gradient update formula for groundless _i at every moment is:

$$g_{t,i} = \nabla_\theta J\left(\theta_i\right).$$

However, the learning rate literal here also changes with t and I:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}.$$

Where G_t is a diagonal matrix, the (I, I) element is the sum of the gradient squared of the parameter I at time t.
The advantage of Adagrad is to reduce the manual adjustment of learning rate
Setting value of super parameter: 0.01 is selected for general index
The downside is that the denominator accumulates, so the learning rate shrinks and eventually becomes very small.
This algorithm is an improvement on Adagrad,

Manuscript received Oct 14, 2018
Quan Wang, School of Computer Science & Software Engineering, Tianjin Polytechnic University, Tianjin, 300387, China

Compared to Adagrad, the G in the denominator is replaced by the decaying average of the past gradient squared, the exponential decaying average

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \varepsilon}} \cdot g_{t,i}$$

This denominator is equivalent to the root mean squared (RMS) of the gradient. In the statistical analysis, the square root of all values is summed, the mean value is obtained, and the square root is obtained.

$$\Delta\theta = -\frac{\eta}{RMS[g]_t} \cdot g_t$$

Where E is calculated as follows, t time depends on the average of the previous time and the current gradient:
Gradient update rules:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

In addition, there will be more changed eta for the RMS [Δ theta], in this way, we don't even need vector set in advance:

$$\Delta\theta = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t-1} = \theta_t + \Delta\theta_t$$

normally set to 0.9

Disadvantages of Adams algorithm

First, the Adams algorithm may not converge. The stochastic gradient descent method does not use the second-order momentum, so the learning rate is constant. In the actual process, the attenuation strategy of the learning rate is generally adopted to make the learning rate decrease continuously.The Adams algorithm is not like this. Second order momentum is the accumulation in the fixed time window. With the change of time window, the data encountered may change dramatically, making it possible to be large or small instead of monotonic changes.This may cause a learning rate shock at the later stage of training, resulting in the failure of the model to converge.

Secondly, Adams algorithm may miss the global optimal solution. The same optimization problem, different optimization algorithms may find different answers, but adaptive learning rate algorithms often find very poor answers.They demonstrate through a specific data example that the adaptive learning rate algorithm may overfit the features that appear in the early stage, while the features that appear in the later stage are difficult to correct the previous fitting effect.

The traditional neural program interpreter adopts Adam algorithm to accelerate convergence. The Adams algorithm is the combination of the modified Momentum and the RMSProp algorithm. In the Adams algorithm, the Momentum is directly incorporated into the first order moment estimation of gradient, that is, the exponential weighting.Because Adam's algorithm may not converge and may miss the global optimal solution, in order to improve the performance of the neural programming interpreter, a new convergence algorithm is adopted to solve these problems

AMSGrad

Input : $x_1 \in F$,step size $\{\alpha_t\}_{t=1}^{T}, \{\beta_t\}_{t=1}^{T}, \beta_2$

Set $m_0 = 0, v_0 = 0, \hat{v}_0 = 0$

For t=1 to T do

$$g_t = \nabla f_t(x_t)$$

$$m_t = \beta_{1t} m_{t-1} + (1-\beta_{1t})\tilde{g}_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_t) g_t^2$$

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

$$x_{t+1} = \prod_{F, \sqrt{\hat{v}_t}} (x_t - a_t m_t / \sqrt{\hat{v}_t})$$

End for

Figure 2 AMSGrad algorithm

IV. EXPERIMENTS

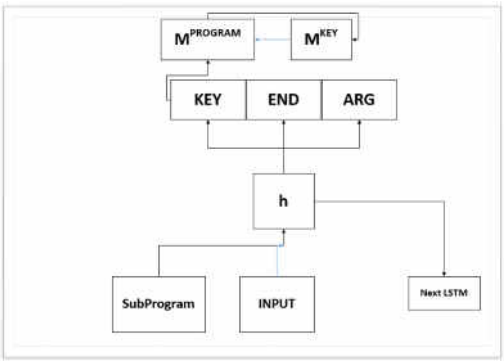

Figure 3 Addition model based on NPI

As shown in Figure 3, the arrows added to the numbers 934 and 348. The grid represents the pointer, which can be moved on the same line, LEFT (LEFT), RIGHT (RIGHT), ADD (ADD), ACT (simplified), CARRY (CARRY), WRITE (WRITE), etc. On the left side of the figure, for example, in the grid, the first subroutine ADD will be executed, after the MPL and MGU are composed, the core network function status of the current time node, Input three decoders as parameters respectively, the decoder generates embedded function keys, find the corresponding values in the space program, that is, need to execute the time node under the subroutine, here is ACT; the probability of the decoder generating the termination program, the probability is less than 0.5 . The decoder will update the function parameters of the next time node. Subsequent operations are similar to the operations in the first grid.

In the NPI addition model, this article uses an experimental environment consistent with Scott Reed, which uses two layers of LSTM, each containing 256 hidden units. For NPI training, adaptive moment estimation (AMSGrad) is employed. In practice, The NPI training has a learning rate of 0.0001 and a batch size of 1.

The task in the NPI add model is to read the numbers in two 10-digit numbers and generate the number of answers. The goal is to learn to apply addition and carry operations from right to left in this algorithm.

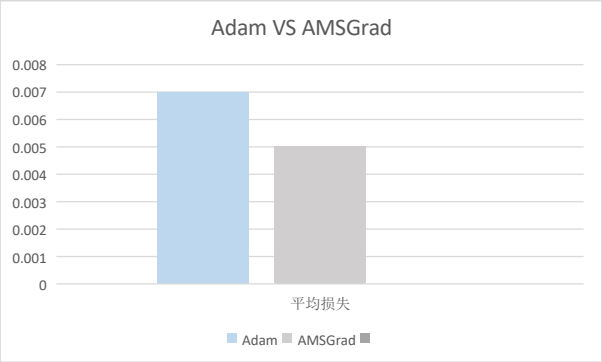

Figure 4 average loss:Adam vs AMSGrad

As shown in Figure 4, The neural programming interpreter does not use the AMSGrad algorithm, its average loss is 0.007 for training. However, the neural programming interpreters with AMSGrad algorithm reduces training time to 0.005.

CONCLUSION

Traditional neural programmer interpreters uses Adam algorithm to accelerate convergence, but Adam algorithm may not be of convergence and may miss the shortcomings of the global optimal solution to avoid these problems, we introduce a new AMSGrad algorithm, AMSGrad algorithm was applied to the neural programmer interpreters addition model, the average loss is reduced compared to the traditional neural programmer interpreters.

REFERENCES

[1] Scott Reed, Nando de Freitas , Neural Programmer-Interpreters[J] , arXiv preprint arXiv:1511.06279,2016.
[2] Sumit Gulwani ,Automating String Processing in Spreadsheets Using Input-Output Examples [J],Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages,317-330,2011.
[3] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow, DeepCoder: Learning to Write Programs[J],arXiv:1611.01989, 2016.
[4] IJ Goodfellow, J Shlens, C Szegedy, Explaining and harnessing adversarial examples[J].arXiv preprint arXiv:1607.02533, 2016.
[5] E Price, W Zaremba, I Sutskever, Extensions and Limitations of the Neural GPU[J].arXiv preprint arXiv:1611.00736, 2016.
[6] M Andrychowicz, K Kurach, Learning efficient algorithms with hierarchical attentive memory[J]. arXiv preprint arXiv:1602.03218, 2016.
[7] Ł Kaiser, I Sutskever,Neural gpus learn algorithms[J]. arXiv preprint arXiv:1511.08228, 2015.
[8] D Bahdanau, K Cho, Y Bengio,Neural machine translation by jointly learning to align and translate [J].arXiv preprint arXiv:1409.0473, 2014.
[9] A Neelakantan, QV Le, I Sutskever, Neural programmer: Inducing latent programs with gradient descent[J]. arXiv preprint arXiv:1511.04834, 2015.
[10] A Graves , G Wayne , I Danihelka, Neural Turing Machines[J]. Computer Science , 2014.
[11] Vinyals, M Fortunato, N Jaitly, Pointer networks[J]. NIPS, 2015 .
[12] K Ellis, A Solar-Lezama, J Tenenbaum,Sampling for Bayesian program learning[J].NIPS, 2016.
[13] Alexander L. Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, Daniel Tarlow,TerpreT: A Probabilistic Programming Language for Program Induction[J]. arXiv:1608.04428, 2016.
[14] X Chen , C Liu , D Song, Towards Synthesizing Complex Programs from Input-Output Examples[J].ICLR,2018.