A GENERALIZED MODEL FOR DATA SCIENCE

A PREPRINT

Ziqi zhang*  
stevezhangz@163.com

Chuanxu Zhao  
xiaoxusoon@qq.com

November 6, 2020

ABSTRACT

Neural network is a powerful tool, which is often regarded as a black box. However, different task requires different parameters to be set up or couldn’t work on, thus variable parameters might be a good solution for different tasks. We present a two-stage model based on Deep reinforcement learning as well as the pre-train method, this model could configure different parameters according to different data, improving and optimizing those parameters furthermore according to the returned loss value in each iteration. We apply this model to Boston housing pricing dataset, and it got a good result in restricted condition which was consistent with our expectations.

Keywords Generalized model, Reinforcement learning, Deep neural network, Data science

1 Introduction

In recent 10 years, the neural network has made a great success in natural language process [1], target detection[2, 3] and so on, along with this success, more and more people tends to design a good structure of neural network, rather than only study how to extract feature from data. All of this mentioned above indicate that neural network’s structure and function are closely connected. We present a two-stage model for setting up hyperparameters of neural networks. The first stage is to initialize the architecture of neural network according to the feature of the task, just like a person will make a draft when he thinks about the work he will do. And the second stage is to optimize the initialized model according to its loss value in the last iteration, just like a person will improve his work based on the previous draft. When find the optimal parameters, using the best parameters value as training data to update the parameters of the model in the first stage. After training, this model will configure good parameters for neural network quickly when be applied to a new task which has some similarities with the task it has processed before. We applied the two-stage model on the Boston housing price dataset, and our model gains a good result which was consistent with our expectation.

2 Related Work

Our work has a little bit of similarities with hyperparameter searching but not completely the same. Let’s review the work of hyperparameters searching first, the hyperparameters searching method gradually change from random searching[4, 5] to natural neural architecture searching[6]. Especially, the natural architecture searching even could find new neural architectures. However, that method was only based on the accuracy return from the last neural network, but couldn’t build a relationship between data and config parameters. Different from previous work, we focus on how to create a general model to be applied to different works, rather than only perform well on a specific work, so we adopt a two-stage method, using the first stage to initialize the structure of neural network according to different tasks and use the downstream stage to optimize the initialized model.

*Corresponding author.
Figure 1: Structure of the two-stage model. Assume that the F model is a pre-trained model, thus the F model could configure the suitable number of neural network layers. And then the reinforcement learning model will optimize the number of layers initialized to the best. Finally, we use the best number of layers to update the parameters of the F model.

3 Architecture

As shown in fig.1 our model consists of two stages. In the first stage, the input data will be input into the F model and then the F model will configure the number of the neural network’s layer, then the neural network will be built, and trained, after training or break out training process manually at a fixed time, we will gain a loss value. We use the reinforcement learning method to optimize the hyperparameters of neural network according to the loss value, and the neural network will be reset, and again and again, this process won’t stop until gain the best parameter or end of the iteration, finally, we use those parameters to train F model. When this system processes a similar task as training, the appropriate number of the layer will be provided by the F model immediately, and the reinforcement learning model just has optimized to optimized it slightly, we might get good results easily.

3.1 Algorithm

Initialize the state-policy function $\pi$ with random weights
Initialize the policy-value function $Q$ with random weights
Initialize replay buffer $\tau$ to capacity $N$
Initialize the F model with random weights
for episode 1, M do
Initialize the number of neural network’s layer, build a neural network $DNN_t$, and train it in fixed iterations $I$, return loss value $loss_t$
for $t=1,T$ do
With probability $\varepsilon$ select a random action $a_t$(Add a layer or reduce a layer)
Otherwise select $a_t = \arg\max (\pi(loss_t; \theta))$
Update layer $t$ to layer $t+1$ according to $a_t$
Rebuild the neural network to $DNN_t$, train it to $DNN_{t+1}$ in fixed iterations $I$, return loss value $loss_{t+1}$
r_t = \begin{cases} 
\frac{loss_t - loss_{t+1}}{loss_{t+1} \times 100} & \text{if } loss_{t+1} > loss_t \\
\frac{loss_{t+1} - loss_t}{loss_{t+1} \times 100} & \text{if } loss_{t+1} < loss_t 
\end{cases} 
Store transition $(loss_t, a_t, loss_{t+1}, r_t)$ in $\tau$
Sample random minibatch transitions $(loss_t, a_t, loss_{t+1}, r_t)$ from $\tau$
Update the parameters of the actor based on policy gradient according to equation 4
Update the parameters of critics according to equation 5
the end for, return the bests number of the neural networks’ layer
Update the parameters of the F model based on gradient descent
end for
Figure 2: Structure of the F model Assume F model is processing data with dimension (100, 13). F model is composed of three convolution layers, three max-pooling layers as well as three fully connected layers. F models are not complex, but it could easily converge when trained by a small size dataset. We want to use this model to build a relationship between training data and the optimal number of layers which was.

3.2 F model

3.2.1 Structure of F model

The first part is the F model (Fig. 2), which is composed of three convolution, pooling layers, and three fully-connected layers. F model could be used to configure the layer of neural networks, by using the training data which was used to training a neural network. At the same time, it’s appropriate to add or delete some convolution layers according to different datasets, here we used two convolution layers.

3.2.2 Performance of F model

We have used several datasets to test the F model, as shown in table1, table2 and fig.2, when used the Boston housing dataset as training data, at the same time set up several integers (3, 10, 50, 60, 100) as the true value which should be predicted by F model. F model can converge easily in all cases. Thus, if we could get the best numbers of layers, we can build a relationship between the best numbers of layers and training data. There is no doubt that the F model has the ability to generate an accurate prediction, and then the prediction value could be further adjusted by the reinforcement learning model.

|                | Random initialized | Train layer | Test layer | True layer |
|----------------|--------------------|-------------|------------|------------|
| Baseline1      | -12.027595         | 0.94575095  | 2.2713373  | 3.0        |
| Baseline2      | 0.94575095         | 9.329973    | 7.9445057  | 10.0       |
| Baseline3      | 9.329973           | 56.170116   | 53.22145   | 50.0       |
| Baseline4      | 56.170116          | 56.833824   | 53.10571   | 60.0       |
| Baseline5      | 56.833824          | 100.744415  | 96.43702   | 100.0      |

Table 1: F model convergence comparison table As shown in the table1, the real value set by us is on the far right side of the table. The second column of the table is the value predicted by the training data set after several rounds of training. The second column of the table is the value predicted by the test data set. It can be seen that the F model has good generalization ability and the prediction is also ideal.
Figure 3: Convergence of all kinds of integers. We can see that our model can converge well for all the integers mentioned above, which provides a strong guarantee for the establishment of data and the optimal number of layers.

|                | Random initialized | Train layer | Test layer | True layer |
|----------------|--------------------|-------------|------------|------------|
| Baseline1      | -0.22468224        | 3.519798    | 3.4251652  | 3.0        |
| Baseline2      | 11.371947          | 11.371947   | 11.142222  | 10.0       |
| Baseline3      | 11.371947          | 52.105453   | 51.38827   | 50.0       |
| Baseline4      | 52.105453          | 61.130825   | 60.287895  | 60.0       |
| Baseline5      | 61.130825          | 107.45473   | 106.21379  | 100.0      |

Table 2: Comparison table of F model convergence after data normalization. As shown in the table 2, after the training data is normalized, we can process the relevant data, and the convergence effect is more complete.

Figure 4: Generalization ability of F Model. Our model converges when use the iris dataset and Boston housing dataset to train.
3.2.3 Generalization ability of F Model

We might doubt that, whether our model has the generalization ability because we would like to apply our model on a different task, so we have to use the different datasets to train and test whether the F model could converge at the same time. We, use the iris dataset and Boston housing price dataset to train the F model in turn, it’s obvious that our model successfully converges (Fig.4), using those data to predict, we get a good result(table. 3).

|                | Boston housing data | Iris data | True layer |
|----------------|--------------------|-----------|------------|
| Baseline1      | 2.263              | 3.67      | 3.0        |
| Baseline2      | 8.9445             | 9.384     | 10.0       |
| Baseline3      | 63.66              | 48.84     | 50.0       |
| Baseline4      | 59.50              | 58.20     | 60.0       |
| Baseline5      | 107.63             | 102.85    | 100.0      |

Table 3: Generalization ability of F model.

3.2.4 Reinforcement learning stage

We describe the architecture of the reinforcement learning stage now, which contains a value function \( Q \) and a policy function \( \pi \). The setup is the same as the agent-environment interaction model. At the beginning of the \( i \) round of training, the environment should build the neural network \( DDN_i \) according to the number of \( layer_i \) which configured by the F model. Then \( DDN_i \) will be trained by the training data same as the F model, tested by the stand value, optimized by Adam optimizer with a learning rate of 0.001, this model will return a loss value \( loss_i \) after training.

We regard \( loss_i \) as the state of the agent, and the Increase or decrease of the number of layers as an action \( a_t \) which was predicted by the policy function \( \pi \), according to the state of the agent \( loss_i \), then we can get a new number of neural network’s layer \( layer_{i+1} \) and the new \( DDN_{i+1} \) configured according to \( layer_{i+1} \) , after training of \( DDN_{i+1} \) it will return a loss value \( loss_{i+1} \). It is larger than \( loss_i \) , the object will be punished, and conversely, a reward value will be provided. Whatever punishment or reward, all be represented by \( r_t \). Then we will get the sequence data \( h_t \) that will be used to train the policy function as well as the value function.

\[
layer_t = layer_t + a_t
\]

\[
a_t = \pi(h_t; \theta)
\]

\[
h_t = \{loss_1, a_1, r_1, loss_2, a_2, r_2, ..., loss_t, a_t, r_t\}
\]

The total reward in the circulation is \( R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \) which should be maximized(equation 4,5). We use the policy gradient method to optimize the policy function \( \pi \) and we use the TD error to optimize the value function \( \nu \). Then the policy function \( \pi \) will stabilize in the best strategy function \( \pi^* \), and the value function will always give the right judgment.

\[
\nabla_\theta \pi_\theta(s_t, a_t; \theta) = (r + \gamma Q_\pi(s_{t+1}, a_{t+1}; w) - Q_\pi(s_t, a_t; w)) \nabla_\theta \log \pi_\theta(s_t, a_t; \theta)
\]

\[
loss(a_{t+1}; W) = \frac{1}{2} ||r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}; w) - Q_\pi(s_t, a_t; w)||^2
\]

4 Experiments

4.1 Model performance

4.1.1 Performance on training data

We used the top 100 sets of training data from the former Boston house prices as the training dataset. Set up the number of nodes in every neural network’s layer as 100, using the same method to initialize the weights. Set up an integer represented the numbers of layers as 5, instead of using the F model to initialize the numbers of layers, because the F model was also initialized by us. We decided when finding the best value of the neural network’s hyperparameters according to the reinforcement learning model, and then we use them to update the hyperparameters of the F model.

Every 30 iterations could be regarded as an episode(a larger iteration), and there is a total of 70 episodes. At the beginning of every episode, we reset the numbers of neural networks and their loss value accordingly. In a word, in each episode, the neural network’s hyperparameters are the same as before, and then those parameters will be optimized.
by the reinforcement learning model. We didn’t set an endpoint, because we have set a limit of the numbers of layer, the loss value should reduce theoretically with the increase of the numbers of neural network’s layer, the endpoint should be set at the maximum number of layers.

Figure 5: The last 15 training episodes’ change trend of loss. It’s obvious that our model performs well in all systems, and in the last 15 episodes, the loss value of the neural network predicted by our two-stage model could converge to a tiny value.

Figure 6: The last 15 training episodes’ change trend of layers. As can be seen from this fig 6, the two-stage model tends to select more layers to reduce the loss value, which is consistent with the assumption we made before.

However, it can’t prove this system could converge when we use another number of layers initialized at the beginning of the episode. So we set another experiment to prove this. We set up the number of layers initialized as 1, 4, 7, 9, 11, 13, 14, 15 in turn, and used our system to optimize the number of layers initialized. As shown in Fig.7 and Fig.8, the results of the system won’t be influenced by the number of layers initialized. The trained model will quickly optimize the number of layers initialized to 15, and the loss value also converges into a very small interval.
Figure 7: The changing trend of loss when the number of layers initialized as different values. It's obvious that all cases have been convergent into a very small interval.

Figure 8: The changing trend of a number of the layer when the number of layers initialized as different values. As shown in Fig.8, the reinforcement learning model will quickly tune the number of neural network’s layer to 15.

4.1.2 Performance on valid data

We used the top 100 sets of valid data from the former Boston house prices as a valid dataset. Use a pre-trained F model to initialize the number of layers first, and then optimize the initialization parameters. We run three-episode without updating the parameters of the actor as well as critics. As shown in Fig.9 and Fig.10, our two-stage model has a good performance.

4.2 Visualizing the critics and actor

As mentioned above, we used policy gradient descent to update the policy function’s parameters and used TD error to optimize the parameters of the value function. Thus, both of them should converge after several episodes. Fortunately,
the experimental results are in good agreement with the theory. As shown in Fig. 11, we can see that both of value function and policy function converges into 0, It is proved that our model has reached the optimal state.

4.3 Evaluation

This two-stage model could be applied in small data such as Boston housing, however still require a large amount of time to train, but we can’t deny this model system could find the best layer of the neural network. We can speed up this process by pre-training. This model has good generalization ability, which is reflected in invalid data. As shown in Fig. 10, F model initialize the best number of the layer immediately because it has been trained by training data before, thus this operation significantly reduces the time for the process of reaching the best parameter.
5 Conclusions

The proposed two-stage model can initialize the number of layers of the neural network, and then adjust the number of layers according to the training situation, to obtain an optimal number of layers. Its advantage is that it takes environmental factors into account in the model, which is easy to understand to let the model know what it is doing. From the above experiments, we can see that our model has good performance and generalization ability. We will go further beyond the existing knowledge framework to study more flexible autonomous regulatory models to adapt to more complex tasks.

References

[1] Devlin J , Chang M W , Lee K , et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. 2018.
[2] Ren S , He K , Girshick R , et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. IEEE Transactions on Pattern Analysis Machine Intelligence, 2017, 39(6):1137-1149.
[3] Redmon J , Divvala S , Girshick R , et al. You Only Look Once: Unified, Real-Time Object Detection[C]// Computer Vision Pattern Recognition. IEEE, 2016.
[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. JMLR, 2012.
[5] James Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. Algorithms for hyper-parameter optimization. In NIPS, 2011.
[6] James Bergstra, Daniel Yamins, and David D Cox. Making a science of model search: Hyper parameter optimization in hundreds of dimensions for vision architectures. ICML, 2013.
[7] Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. arXiv preprint arXiv:1511.06732 2015.