Course Report: Logic Neural Network

Yaoyao Ding

yyding@sjtu.edu.cn
Shanghai Jiao Tong University

Abstract. This is a course report of Computer Science: Advanced Topics. This project is a team work and the other team member is Yutong Xie. In this project, we propose another kind of neural network layer, which can extract and calculate fuzzy logic expressions.

1 Introduction

Currently, there are all kinds of layers in the neural network, such as convolution layer, pooling layer and fully-connected layer. But these components in the network are mainly designed for computer vision task. In other tasks other than computer vision, the main layer we use is fully-connected layer. Fully-connected layer harnesses dot product to recognize pattern. In our work, we harness the logic kernel to extract the fuzzy logic expressions and calculate the truth values of fuzzy logic expressions. A logic neural network can be got by stacking the logic layers. A logic layer consists of many logic kernels. In the synthetic data set, logic neural network can achieve better accuracy and better generalization ability. However, in the real data set of a Click-Through-Rate task, logic neural network suffers from the problem that it need much more epochs of training to converge.

2 Fuzzy Logic

The logic we use in our network is fuzzy logic. In fuzzy logic, the truth values of variables are real number between 0 and 1, which allows us to differential operation. In logic neural network, there are mainly two kinds of logic operations: fuzzy negation and fuzzy disjunction. In fact, we can express any logical expressions by the two logic operations.

Fuzzy Negation Assume p is a fuzzy logical variable, we can define $1-p$ as the negation of p. But sometimes we do not want to negate p completely. That is, we want to partially negate p. So we employ a negation indicator α to show how likely we want to negate p. And we define the fuzzy negation of p with negation indicator α as

$$\alpha(1-p)+(1-\alpha)p,$$

where $\alpha, p \in [0,1]$

	Definition	Example
Variable	p	0.7
Negation Indicator	α	0.8
Fuzzy Negation	$\alpha(1-p)+(1-\alpha)p$	0.38

Table 1. Example of Fuzzy Negation

Fuzzy Disjunction Assume $p_1, p_2, \ldots, p_n$ are n variables that we want to calculate the disjunction of them. We can simply use the maximum of the truth values of them to represent the disjunction result. So we define the disjunction of $p_1, p_2, \ldots, p_n$ as

$$\max(p_1, p_2, \ldots, p_n),$$

where $p_1, p_2, \ldots, p_n \in [0, 1]$.

	Definition	Example
Variable	p_1, p_2, p_3	$0.7, 0.8, 0.1$
Disjunction	$\max(p_1, p_2, p_3)$	0.8

Table 2. Example of Disjunction

3 The Model of Logic Neural Network

3.1 Logic Neural Kernel

We combine the negation and disjunction and put them into the logic neural kernel. Assume the input of the logic neural kernel is n logical variables:$p_1, p_2, \ldots, p_n$ (Let $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$). And there are two trainable parameters in the kernel: the negation indicator α and selector $\beta(\boldsymbol{\alpha}, \boldsymbol{\beta} \in [0, 1]^n)$. There are 3 steps of calculation in the kernel.

Firstly, we do the negation operation:

$$\boldsymbol{o}_1 = (1 - \boldsymbol{\alpha})\boldsymbol{p} + \boldsymbol{\alpha}(1 - \boldsymbol{p})$$

Secondly, we do the selection operation:

$$\boldsymbol{o}_2 = \boldsymbol{\beta} \cdot \boldsymbol{o}_1$$

Finally, we do the disjunction operation:

$$q = \max_{i=1,\ldots,n} \boldsymbol{o}_{2i}$$

Figure 1 contains two examples of the logic neural kernel.

If we use f to represent the logic neural kernel, then it is simple to represent the kernel by

$$q = f(\boldsymbol{p})$$

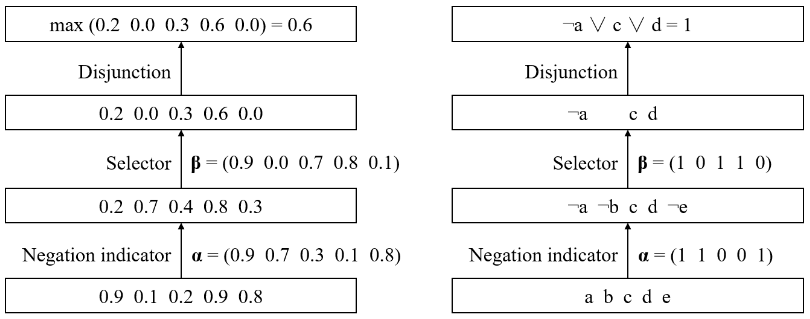

Fig. 1. Left: A concrete example of the logic neural kernel. All the truth values of variables are real numbers between 0 and 1. Right: All the variable are represented by symbols and we can figure out what happened after every step.

3.2 Logic Neural Layer

Assume the input of a logic neural layer is n logic variables:$\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$. We put m logic neural kernels together to get a logic neural layer. These kernels share the same input and the outputs of each kernel form the output of the layer. Assume the kernels are $f_1, f_2, \ldots, f_m$ and their outputs are $q_1, q_2, \ldots, q_m$. Then the output of the layer is $\boldsymbol{q} = (q_1, q_2, \ldots, q_m)$. If we use g to represent the logic neural layer, then it is simple to represent the layer by

$$\boldsymbol{q} = g(\boldsymbol{p})$$

Figure 2 illustrate the logic neural layer.

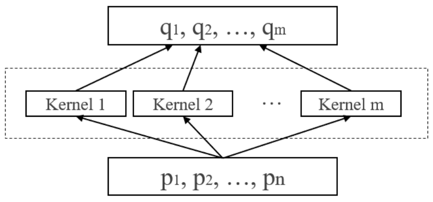

Fig. 2. Illustration of logic neural layer.

3.3 Logic Neural Network

The input of the logic neural network is the multi-hot input. That is, all the input variables are 0 or 1. Then we can stack the logic neural layers to get the logic neural network. We can also use the logic neural layers as the first few layers of a network and the consequent layers are fully-connected layers. Figure 3 illustrate the two kinds of networks.

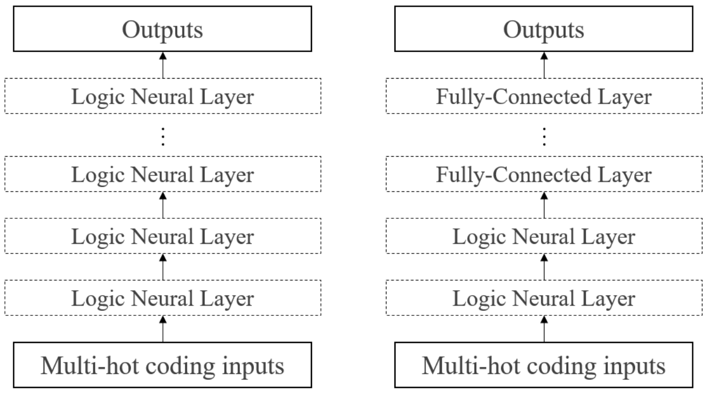

Fig. 3. Two kinds of networks within which logic layers are used.

4 Experiments

We have conducted three experiments. The first experiment is on synthetic data. The second experiment is also on synthetic data. And we want to know whether the logic neural layer has learned logic expressions from data. The third experiment is using the logic neural network to solve the click-through-rate prediction problem.

4.1 Synthetic Experiment

We randomly choose a logic proposition in advance. The number of logic variables is 100. For each sample, we generate the truth values of the variables. And then check whether the proposition is true with respect to the generated truth values. The result determines the label of this sample. We generate many samples to get the synthetic data.

We use three networks in this experiment. Their architectures are in Figure 4. All the networks have three layers in each of them. The first network has three logic neural layers. The number of kernels of each layer is 100. The second network has two logic nerual layers as the first two layers. The third layer of the second network is the fully-connected layer whose units is 100. The thrid network has three fully-connected layers and the number of units in each layer is 100.

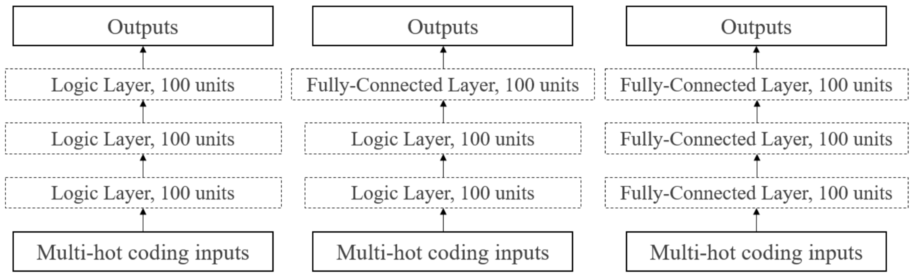

Fig. 4. Three Networks. Left: Pure logic neural network; Middle: Mixture network; Right: Pure fully-connected network.

Figure 5, 6, 7 are the results on three different networks. The auc metrics is the area under the roc curve, which is the metrics we care about most.

From this experiment, we find that the mixture network has better performance compared with pure logic neural network and pure fully-connected network. The pure logic neural network need more epochs to converge. And the pure fully-connected network suffers from the over-fitting problem. The logic layer need more epochs to train. And it has better generalization ability on synthetic data. The activation function of all the fully-connected layer is the rectified linear unit(Relu) activation function.

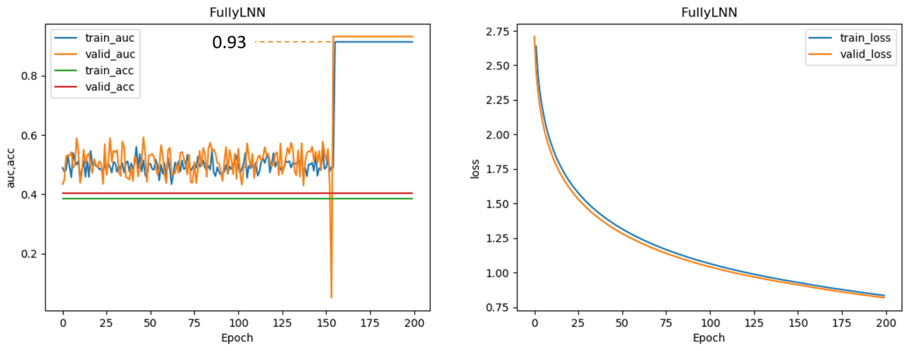

Fig. 5. Pure Logic Neural Network. Left: Accuracy and AUC score in both training phase an validation phase. Right: Loss in both training phase and validation phase.

4.2 Interpretability

We do this experiment on small data and small network. The network has three layers. First two layers are the logic neural layers and the third layer is the fully-

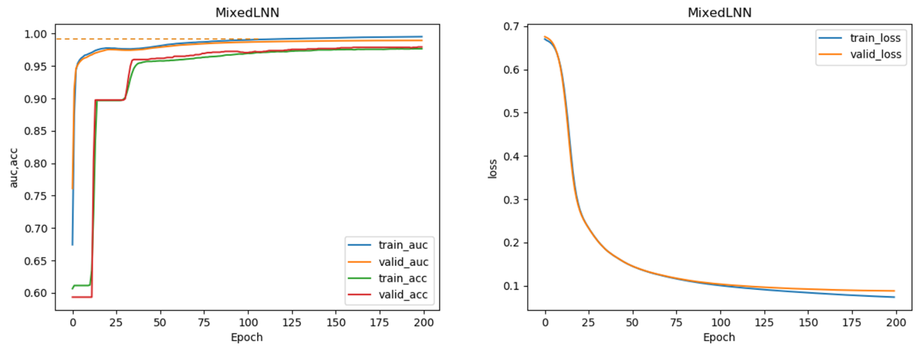

Fig. 6. Mixture Network. Left: Accuracy and AUC score in both training phase an validation phase. Right: Loss in both training phase and validation phase.

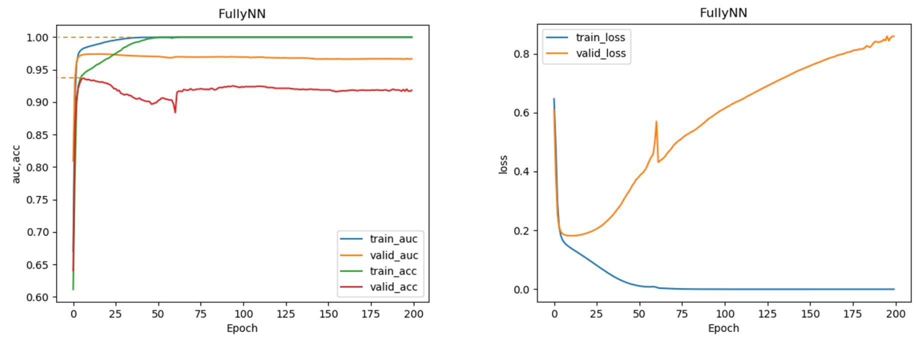

Fig. 7. Fully-Connected Network. Left: Accuracy and AUC score in both training phase an validation phase. Right: Loss in both training phase and validation phase.

connected layer. The number of kernels in each of the logic neural layer is 6 and the units in fully-connected layer is also 6. The proposition of the data is

$$(p_2 \wedge p_6 \wedge p_{10}) \vee (p_3 \wedge \neg p_6 \wedge \neg p_9) \vee (\neg p_5 \wedge \neg p_{10}).$$

And by examining the kernels in the first layer, we find that the kernels express the following logic expressions: $\neg p_3 \wedge \neg p_6 \neg p_7$, $\neg p_5$, $\neg p_{10}$, $p_1 \wedge p_2 \wedge \neg p_6 \wedge \neg p_9$ and $p_2 \wedge p_6 \wedge p_{10}$. We can find some sub-parts of the proposition we use to generate data in these logic expressions. It shows that the logic neural kernel can extract some useful logic expressions from data.

4.3 On CTR Task

We conduct this experiment by using the network to solve the click-through-rate prediction problem. The input of samples in the data is multi-hot input. We use two networks to do prediction. The first one is the mixture network and the second one is the pure fully-connected network. Figure 8 and 9 shows the results of both networks. We can see that the mixture network can be used in this task, though it has worse auc compared with the fully-connected network. We can also notice that it need more epochs to train.

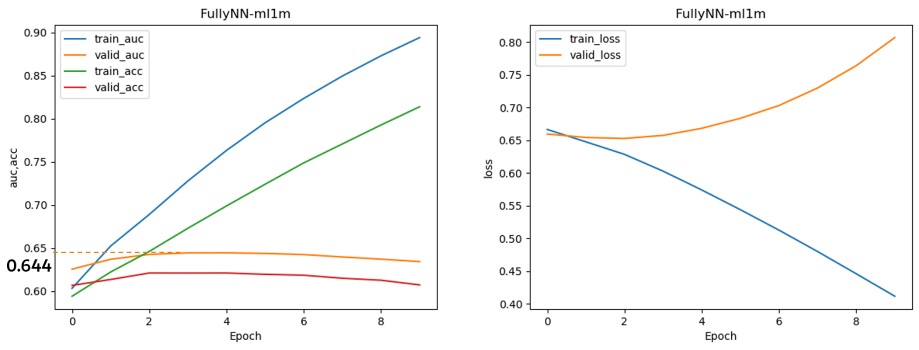

Fig. 8. Pure Fully-Connected Network. Left: Accuracy and AUC score in both training phase an validation phase. Right: Loss in both training phase and validation phase.

5 Future Work

There are some interesting questions that need us to answer.

- Why the network need so many epochs to train. In other words, why the training efficiency is so low?
- Can we modify the structure of the kernel to reduce the number of epochs required to converge?

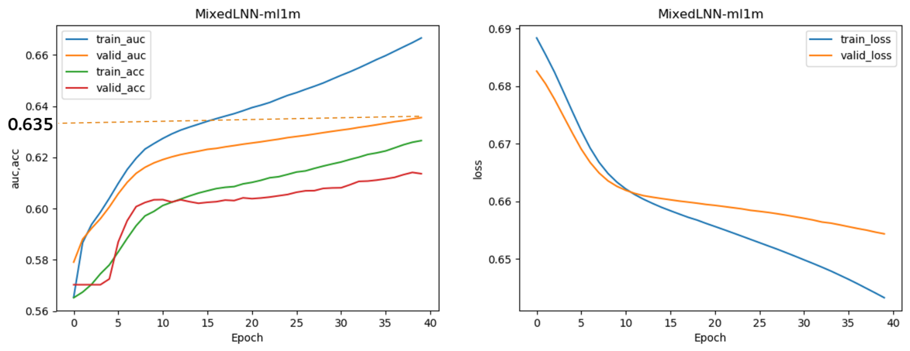

Fig. 9. Mixture Network. Left: Accuracy and AUC score in both training phase an validation phase. Right: Loss in both training phase and validation phase.

– How to use the logic layer in the middle of fully-connected layers? So we can use the logic neural layer in more places.