A study on different neural network structure for eatable and poisonous mushroom classification

Pargorn Puttapirat (ID 3117999011), Chan Min Wai (ID 3117999104), Lee Hue (ID 3117999241)
***BDDL Group 5**, School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, China*
**All members has equal contribution*

***Abstract*—This project is an implementation of artificial neural networks in binary classification task. The aim of the project is to distinguish edible from poisonous mushrooms. The features used in this experiment are based on The Mushroom Dataset from UCI Machine Learning Repository which is a publicly available dataset. In this report, we will discuss the training and classification performance of the classification model based on different optimization methods, loss functions, and different network architectures. Our findings agree with the best practices knowledges on machine learning. Our proposed method can generalize to the dataset and correctly classify all the samples in the dataset.**

Keywords—binary classification, deep learning, mushroom, neural network architecture

I. INTRODUCTION

Nowadays, it is hard for the new generations to recognize the edible mushroom from the poisonous mushroom. It is more challenges for human to classify all kinds of mushroom that is edible and poisonous. Death from mushroom poisoning could be devastating. Development of an algorithm to solve this problem could help to save lives. Thus, this paper investigates the implementation of artificial neural networks on an edible and poisonous mushroom classification task.

A. Introduction to The Mushroom Data Set

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mush-rooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous while the unknown edibility and not recommended is treated as poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy [1]. In our experiment, we modify the original dataset be truncating some features that does not have a clear explanation on the values. The modification is that we removed the attribute "veil-type" because it has the same value throughout all samples. The original dataset could be downloaded from The UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Mushroom).

1) Attribute Information

The input to the network is constructed from most of the attributes in the Mushroom Dataset. In Table 2, we have noted all the attributes available from the dataset which we have picked. Twenty-one attributes are considered to be the input features of the network. From the table, we could observe the "Number of Values" column to see the variety of values in each attribute.

TABLE I. ALL ATTRIBUTES AND ITS VALUES IN THE INPUT DATASET

Attributes	Values	Number of Values
Cap shape	bell, conical, convex, flat, knobbed, sunken	6
Cap surface	fibrous, grooves, scaly, smooth	4
Cap color	brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow	10
Bruises	bruises, no	2
Odor	almond, anise, creosote, fishy, foul, musty, none, pungent, spicy	9
Gill attachment	attached, descending, free, notched	4
Gill spacing	close, crowded, distant	3
Gill size	broad, narrow	2
Gill color	black, brown, buff, chocolate, gray, green, orange, pink, purple, red, white, yellow	12
Stalk shape	enlarging, tapering	2
Stalk root	bulbous, club, cup, equal, rhizomorphs, rooted, missing	7
Stalk surface above ring	fibrous, scaly, silky, smooth	4
Stalk surface below ring	fibrous, scaly, silky, smooth	4
Stalk color above ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow	9
Stalk color below ring	brown, buff, cinnamon, gray, orange, pink, red, white, yellow	9
Veil color	brown, orange, white, yellow	4
Ring number	none, one, two	3
Ring type	cobwebby, evanescent, flaring, large, none, pendant, sheathing, zone	8
Spore print color	black, brown, buff, chocolate, green, orange, purple, white, yellow	9
Population	abundant, clustered, numerous, scattered, several, solitary	6
Habitat	grasses, leaves, meadows, paths, urban, waste, woods	7

2) Class Distribution

There are 8124 instances of data available in total. In machine learning task especially in binary classification, it is important that the number of samples in the two classes should be balance or equal. Otherwise the network could be biased to one of the class. From our analysis, 51.8% of positive instances and 48.2% of negative instances which is quite balance as depicted in Fig. 1.With randomly generated mini-batches of training data from this whole dataset, the imbalance could not affect the performance of the network and cause some biases.

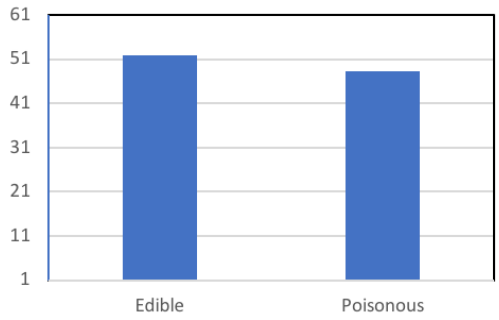

Fig. 1. Class distribution of the dataset

B. Introduction to TensorFlow

TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

C. Binary Classification and Network Design

The binary classification problem is one type of classification task in deep learning. The two approaches shown in Fig. 2 are widely adopted. The network output from the network with two output nodes can be used as a probability of the first and the second condition matching. For example, if there are two classes, the first class has the value of 0.2 and the second class has the value of 0.8, not only that we can be more confident that the prediction should match the second class, but we can also know that 20% match the condition of the first class. This kind of network architecture can also be easily expanded to higher number of classes. Our proposed method will pursuit the network that has one output node. To advise the network to output the classification results correctly, we use binary values in training set which are 1s and 0s for each class. The decision to choose this kind of network in our proposed method is arbitrary. We suspect that the network would have the same performance with another network architecture.

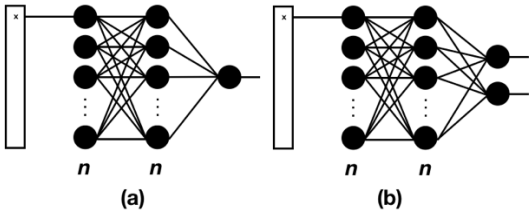

Fig. 2. Different binary classification neural network structures, (a) network with 1 node in the output layer and (b) with 2 nodes in the output layer.

II. Proposed Method

A. Data preprocessing

In the data preprocessing step, we treat each attribute of the data as one input. The value within the attribute is treated as continuous value. For example, the odor attribute has 9 different values which are almond, anise, creosote, fishy, foul, musty, none, pungent, and spicy. The numbers 0 to 8 are assigned to each value respectively and the input value is normalized to [0,1] before it is fed to the network. Noted that for some attributes that have only two possible values, we treat that attributes as a binary value.

B. Network design

Our proposed method is to use fully connected or dense neural network in every layer to solve binary classification problem. Our network will predict whether the specific mushroom is edible (positive class) or poisonous (negative class). We have decided to use the network with 3 layers in total including one input layer, one hidden layer, and one output layer as shown in Fig. 3. Considering our situation as binary classification network, the number of nodes in the output layer should be one or two and, in our case, we choose output layer with one node. However, there is no established method to decide the number of nodes in the input and hidden layer in our network. This brings us to experiment on the number of nodes in these two layers.

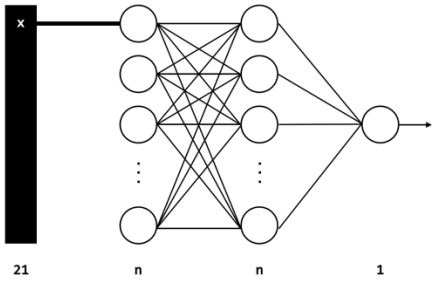

Fig. 3. Proposed Fully Connected Neural Network Model. There are 21 inputs. There are n nodes in the input and hidden layer and 1 node in the output layer.

To express the proposed network as mathematical equations, we consider 21 inputs as column-vector $p^{(1)}$, weight matrix W, and bias for each node as b. In the first layer we shall calculate the output $a^{(1)}$ which is equivalent to the input of hidden layer, $p^{(2)}$. At first, the activation function, *f(•)*, that will be used in the network is not specified

$$a^{(1)} = f\left(W^{(1)}p^{(1)} + b^{(1)}\right) \quad (1)$$

$$a^{(2)} = f\left(W^{(2)}a^{(1)} + b^{(2)}\right) \quad (2)$$

$$a_{output} = a^{(3)} = f\left(W^{(3)}a^{(2)} + b^{(3)}\right) \quad (3)$$

III. Experiment

1) Experiment technical setups

In our project, we implement 4 main machine learning and machine learning-related packages which are Pandas, TensorFlow, Keras, and TensorBoard. Since the size of our data is not so large and the network is also not too complex. Our experiments are carried out on a laptop computer.

Our aim is that our model should be able to achieve highest possible accuracy and as fast as possible. Thus, the accuracy is the first priority and the speed is the second priority. We have run 22 experiments in total. The differences between

TABLE II. EXPERIMENT SETUPS

Experiment	Optimizer	Activation function (at hidden layer)	Batch size	Epochs	Comparison
1	SGD	Relu	50	100	Accuracy is about 0.5. (See figure 2)
2	SGD	sigmoid	50	100	Experiment 2 is faster and higher accuracy than experiment 1. So we keep experiment 2.
3	SGD	Relu	50	100	Experiment 2 is faster than experiment 3, but lower accuracy. So we keep experiment 3 because the accuracy is first priority.
4	SGD	hard_sigmoid	50	100	Experiment 4 is faster than experiment 3, but slightly lower accuracy. At this time, we keep both because we could not decide which one is better and so compare them again.
5	SGD	hard_sigmoid	25	100	From experiment 3 and experiment 4 which have the same convergence speed, we try to reduce the batch size and repeat the experiments to see the difference clearly. In this experiment 5 which was experiment 4, we reduce batch size from 50 to 25.
6	SGD	Relu	25	100	Experiment 6 (which was experiment 3 reducing batch size) is faster than experiment 5 (which was experiment 4 reducing batch size). So we keep experiment 6.
7	SGD	Relu	50	100	To reduce the training time, we increase batch size in experiment 7 which was experiment 6.
8	SGD	linear	50	100	Experiment 7 is faster and higher accuracy than experiment 8. So we keep experiment 7.
9	SGD	selu	50	100	Experiment 9 is higher accuracy than experiment 7. So we keep experiment 9.
10	SGD	softmax	50	100	Experiment 10 has no accuracy. So we keep experiment 9.
11	SGD	softplus	50	100	Experiment 11 decreases accuracy. So we keep experiment 9.
12	SGD	softsign	50	100	Experiment 12 is faster, but very slightly lower accuracy than experiment 9. But we keep both experiment 9 and 12.
13	SGD	tanh	50	100	Keeping experiment 9 and experiment 12, we try another experiment 13. So now we have three experiments to compare.
14	SGD	tanh	50	300	To decide which one of three experiments (experiment 9, 12 and 13) is better, we increase epochs from 100 to 300 to see the convergence. This experiment 14 was experiment 13.
15	SGD	softsign	50	300	Experiment 14 is slightly higher accuracy than experiment 15. So we keep experiment 14.
16	SGD	selu	50	300	Experiment 14 is slightly higher accuracy than experiment 16. So we keep experiment 14.
17	SGD	tanh	50	100	To reduce the training time, we reduce epochs from 300 to 100 in this experiment 17 which was experiment 14.
18	Adadelta	tanh	50	100	Experiment 17 is faster and higher accuracy than experiment 18. So we keep experiment 17.
19	Adagrad	tanh	50	100	Experiment 17 is faster and higher accuracy than experiment 19. So we keep experiment 17.
20	SGD	tanh	25	100	In experiment 20 which was experiment 17, we reduce batch size and increase network size from [1 1 1] to [2 2 1] to check whether it is better or not. Fortunately experiment 20 is better than experiment 17. So we keep experiment 20.
21	SGD	tanh	25	300	In experiment 21 which was experiment 20, we increase epochs from 100 to 300 to see the convergence. Experiment 21 is also faster and higher accuracy than experiment 20 and it still doesn't reach 100% of accuracy. But we keep experiment 21 and try to change network size from [2 2 1] to [3 3 1].
22	SGD	tanh	25	300	In experiment 22 which was experiment 21, we change network size [2 2 1] to [3 3 1]. And experiment 22 is faster and higher accuracy than experiment 21 and it also reaches to 100% of accuracy.

each experiment can be seen in Table 2. Every experiment in Table 2 uses 80:20 train-to-test ratio, 0.01 learning rate, and mean squared error as a loss function. The first experiment to the 19th experiment, we use the proposed network structure with n=1, and we use n=2 for 20th and 21st experiments and n=3 for 22nd experiment. It can be confusing if we compare every experiment at a time. So, we picked and kept the better experiment after each two experiments to avoid the complexity. Those step by step experiments can be seen in the Table 2. In the 1st to 17th experiment, we tried 17 different activation functions without changing the other parameters in order to know which activation function is better. After 17th experiment, we know that tanh activation function is the best for our project (see activation function in the Table 2 and result in Figure 4). Then we tried to change the other two different optimizers ('Adadelta' and 'Adagrad' in experiment 18 and 19). Notice that we used SGD optimizer from experiment 1 through 17. After comparing experiment 1 to 17 with experiment 18 and 19, we can conclude that SGD optimizer is still the best for our project (See Table 2: Optimizer and result in Figure 4). In the 20th experiment, we tried to vary the network sizes and then fortunately we can notice that the performance is better than the previous experiments as shown in Table 2. In experiment 21 to 22, we increased epochs to see the convergence and we can also notice that it converges faster with higher accuracy than the previous experiment, but it could not reach to 100% accuracy (see epochs in the table 2 and result in figure 4). Finally, in the 22nd experiment we decide to adjust the network size. That experiment 22 shows that our model with network [3 3 1] is the best for our project.

IV. RESULT

In this part, we show the training of the experiments number 1, 17, 19, 20, 21, and 22 which are key experiments that have brought us to the final model configurations. All configurations are shown in the proposed method section. The

1st experiment's configuration is arbitrary decision with some background knowledge, therefore, we choose simple and well-known loss function and activation function. With the 17th experiment, we conclude that "tanh" is a suitable activation function for our network. For the 19th and 20th experiment, we were trying different optimizers. For 21st and 22nd experiment (Fig. 6), we add more nodes, so the network can generalize and reach higher accuracy. The two networks have slightly different configuration, yet they behave differently as shown in Fig. 6, this reflects that they are still not stable enough. The comparison of accuracy and loss for the mentioned experiments is shown in Fig. 6.

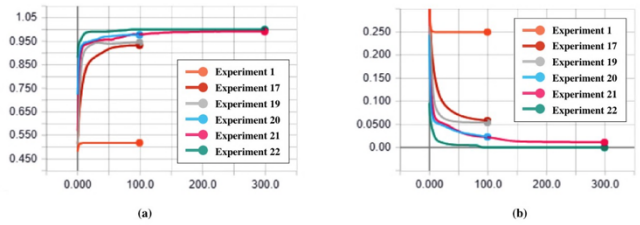

Fig. 4. (a) accuracy and (b) loss comparison among experiment 1, 17, 19, 20, 21, 22

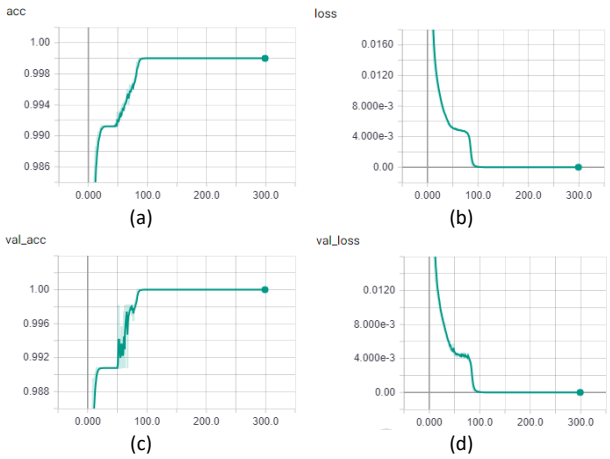

Fig. 5. Training (a) accuracy with (b) loss of experiment 22, and validation (c) accuracy with (d) loss of experiment 22

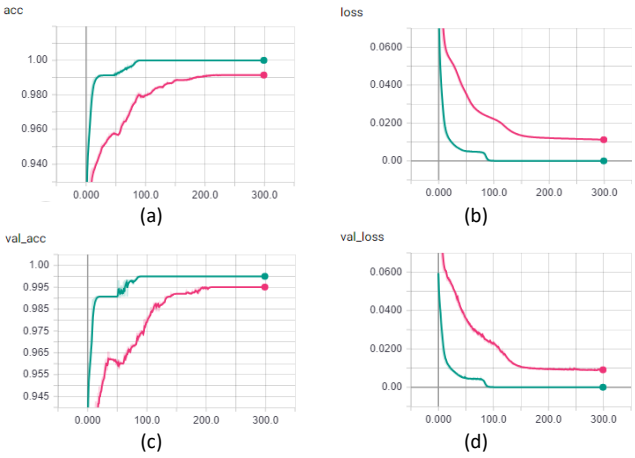

Fig. 6. Difference between training (a) accuracy with (b) loss and validation (c) accuracy with (d) loss of experiment 21 and experiment 22

Due to the randomization of initialization of weights and biases, the network with n=2 cannot achieve 100% accuracy and diverge after some iterations. This behavior can be seen in Fig. 7.

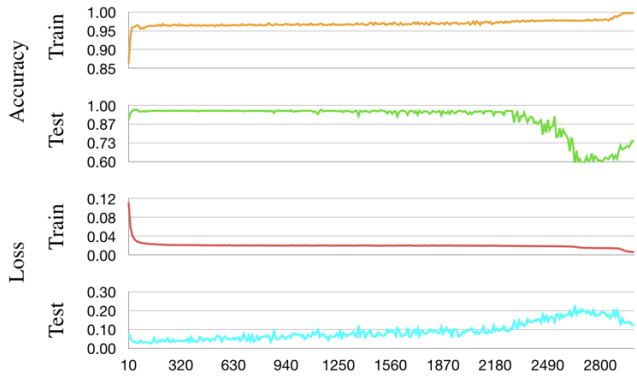

Fig. 7. An example of overfitting behavior of n=2 network.

With all information we have learned, we draw a final conclusion that the network structure should be the proposed network with n≥3. The SGD optimization method with mean squared error gives the best performance. In our case the final network configuration converges at around 160 iterations. Even though the network with n=11 converges at 90 iterations which is about 1.7 times faster, it has 4.7 times more parameters than the n=3 network.

V. Discussion

A. *The simplest network that generalized*

In deep learning, to achieve the best performance in the framework, it is logical to work with the simplest network possible. The network size and number of parameters often grow with the complexity of the problem at hand. In our experiment, we have found that the network with n=3 is the simplest network that can classify all the samples correctly. Furthermore, the networks that have more parameters than this can also generalize to our problem. From Fig. 8, we can see that the network with more complexity will converge faster. Where complexity or total number of parameters, p, in the our fully connected network could be calculated by $p = (21{\times}n) + n^2 + 4$.

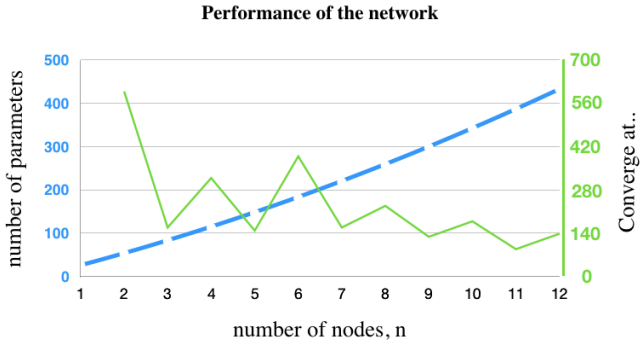

Fig. 8. Performance of the network by number of parameters and the iterations where the network converges

B. *Input features reduction*

To find out whether these 21 attributes are indispensable or not, we could reduce the number of input features in our network and train the network among the reduced features. Since our network has many features and output variation is quite small, true and false, we suspect that reduction of input features to some degree would not affect the network

performance very much. In this matter, brute-force elimination could be implemented to find out which feature is less important. Other feature selection techniques such as principle component analysis could also be used.

C. *Different approach for output layer in binary classification task*

We have noticed and explained in the introduction that the output layer of the network can be done in two ways. We assume that the classification accuracy for both approaches would be the same. To prove this assumption, experiment could be done as a future work.

VI. Conclusion

We have developed an algorithm which is a realization of the artificial neural networks for mushroom classification task. The Mushroom Dataset with 8,124 instances of data is used for training. According to the several experiments, the system can perform with 100% of accuracy. While the training data has 51.80% of edible and 48.20% of poisonous mushroom, therefore, our network could perform better than the baseline at 51.80%. This system can be a helpful way for people to classify the edibility of the mushroom which is unknown to them. If this kind of algorithm could be implemented in the real world, deaths from mushroom poisoning may be prevented.

VII. Acknowledgement

The mushroom records are drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf. The donor of the mushroom dataset is Jeff Schlimmer (jeffrey.schlimmer@a.gp.cs.cmu.edu) and the dataset has been downloaded from UCI Machine Learning Repository.

In this project we implement TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems as our machine learning system. We thank all developers involved in this project.

References

[1] A. Butalia, D. Shah, and R. . Dharaskar, “Mushroom Plant Analysis through Reduct Technique,” *Int. J. Comput. Appl. (0975 – 8887)*, vol. 1, no. 5, pp. 54–58, 2010.

Other references:

Neural network activation function: https://www.tensorflow.org/api_docs/python/tf/keras/activations

Optimizer: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

Loss function: https://www.tensorflow.org/api_docs/python/tf/losses/mean_squared_error