

## Exercise 1

Topic: Toy dataset.

**1. For each dataset (the four boxes under DATA) explore the best features that make the network converge faster.**

I assume that when the testing and training loss lost stays the same for more than five seconds, the network is regarded as converged.

For dataset 1,  $x_1^2$  and  $x_2^2$  are the best features. For dataset 2,  $x_1$ ,  $x_2$  and  $x_1x_2$  are the best features. For dataset 3,  $x_1$  and  $x_2$  with the linear activation can work very well since the data is linear dividable. For dataset 4,  $x_1$ ,  $x_2$ ,  $\sin(x_1)$  and  $\sin(x_2)$  are the best features due to the spiral distribution of the data. Converged results are shown from Fig.1 to Fig.4 for each dataset.

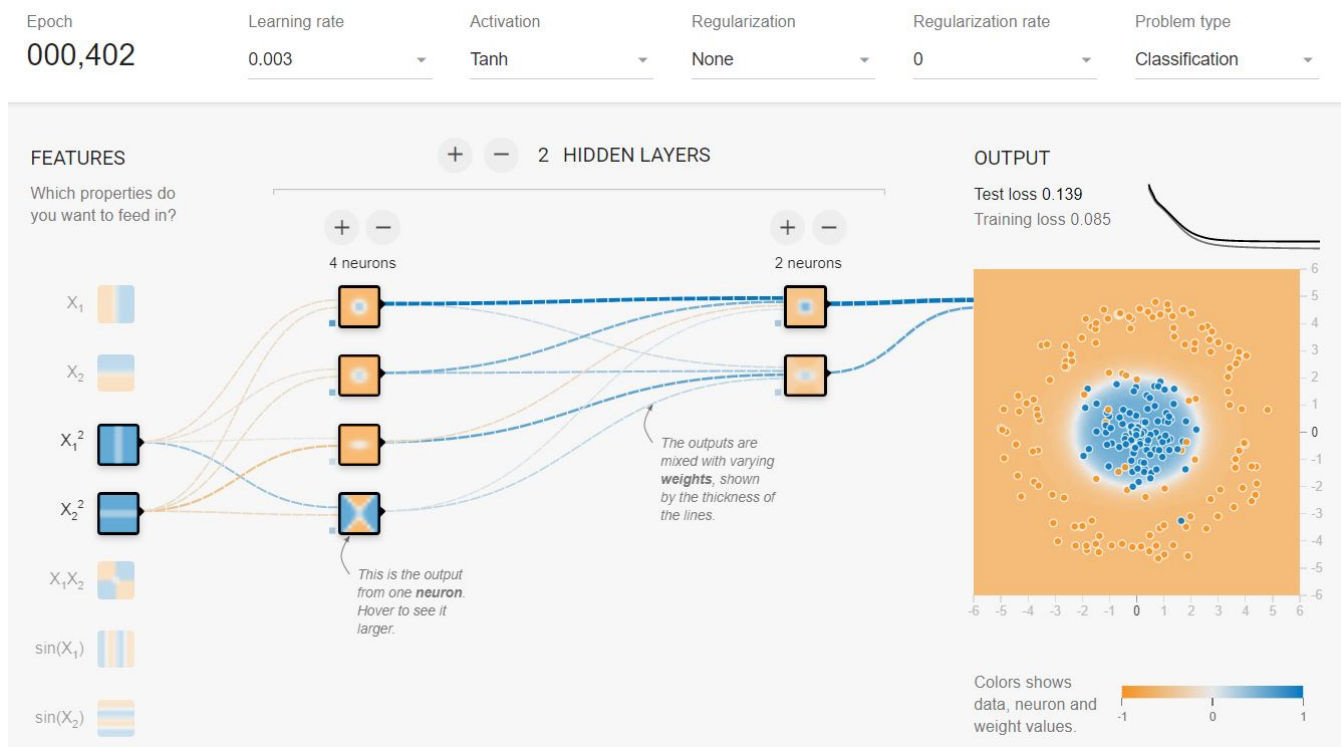


Figure 1: Best feature for dataset 1

**2. For the spiral dataset and default network how does the batch size influence the results? How does it relate with the learning rate ?**

As shown in Table 1, using the default network, none of the parameter settings can achieve a good enough result compared to adding  $\sin(x)$  feature to the input. Batch size=20 and learning rate=0.03 is the best parameter setting among them. When learning rate is too big, there would be some zigzags on the loss curve, which means the step of gradient descend is so big that the network keeps searching back and forth and fails to reach the global optima. When the learning rate is too small (which is not shown in the table), the gradient descend needs too many steps to reach the valley so it's too time-consuming. The batch size is also related to the number of batch, which determine the number of weight optimization together with the epoch number. However, from the experiments, I can only tell that the batch size should be not too larger nor too small in order to have low loss and fast converge.

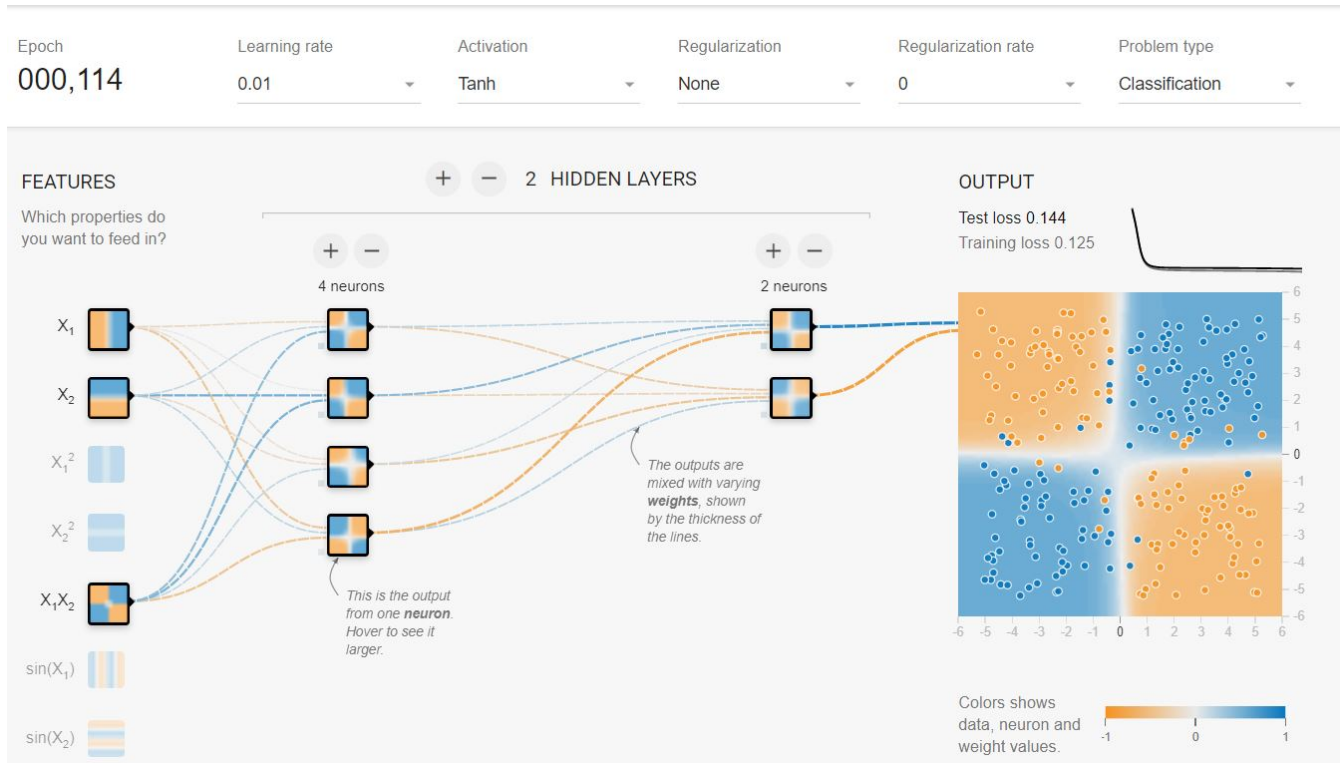


Figure 2: Best feature for dataset 2

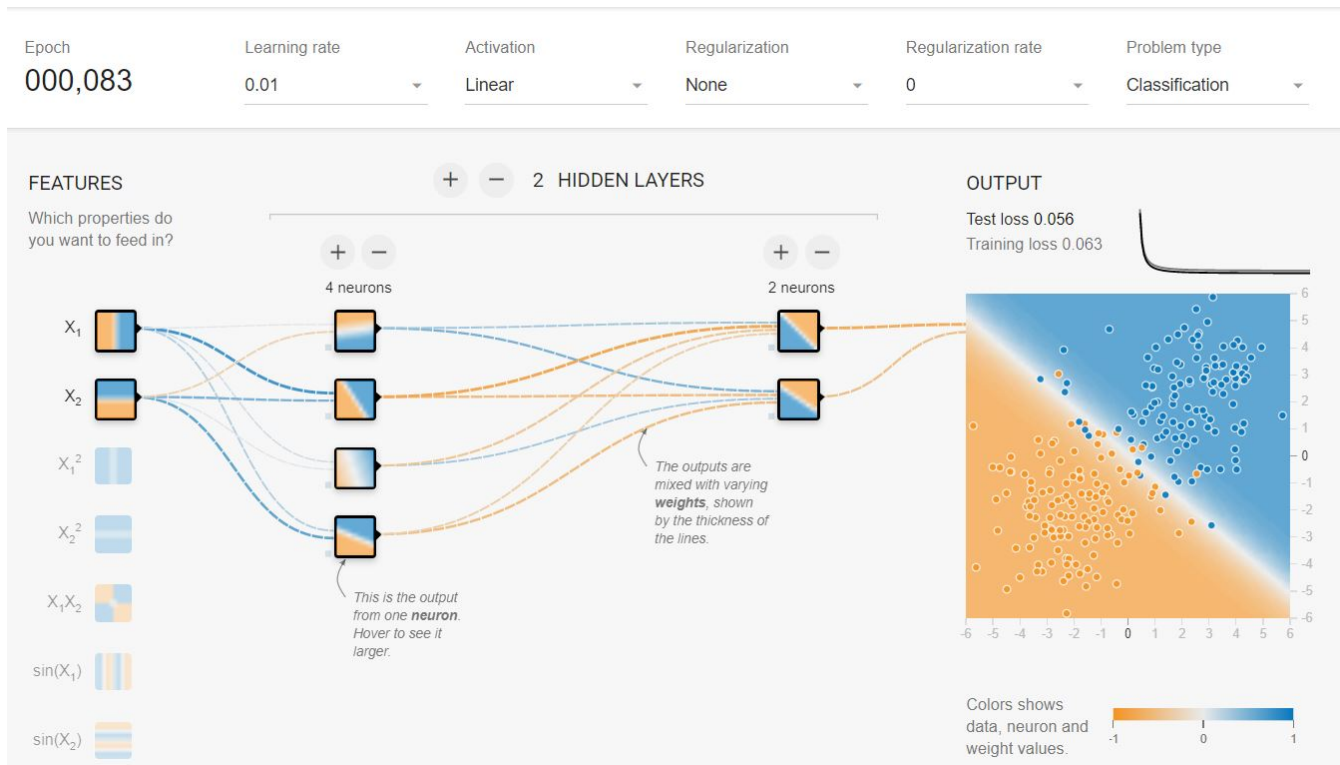


Figure 3: Best feature for dataset 3

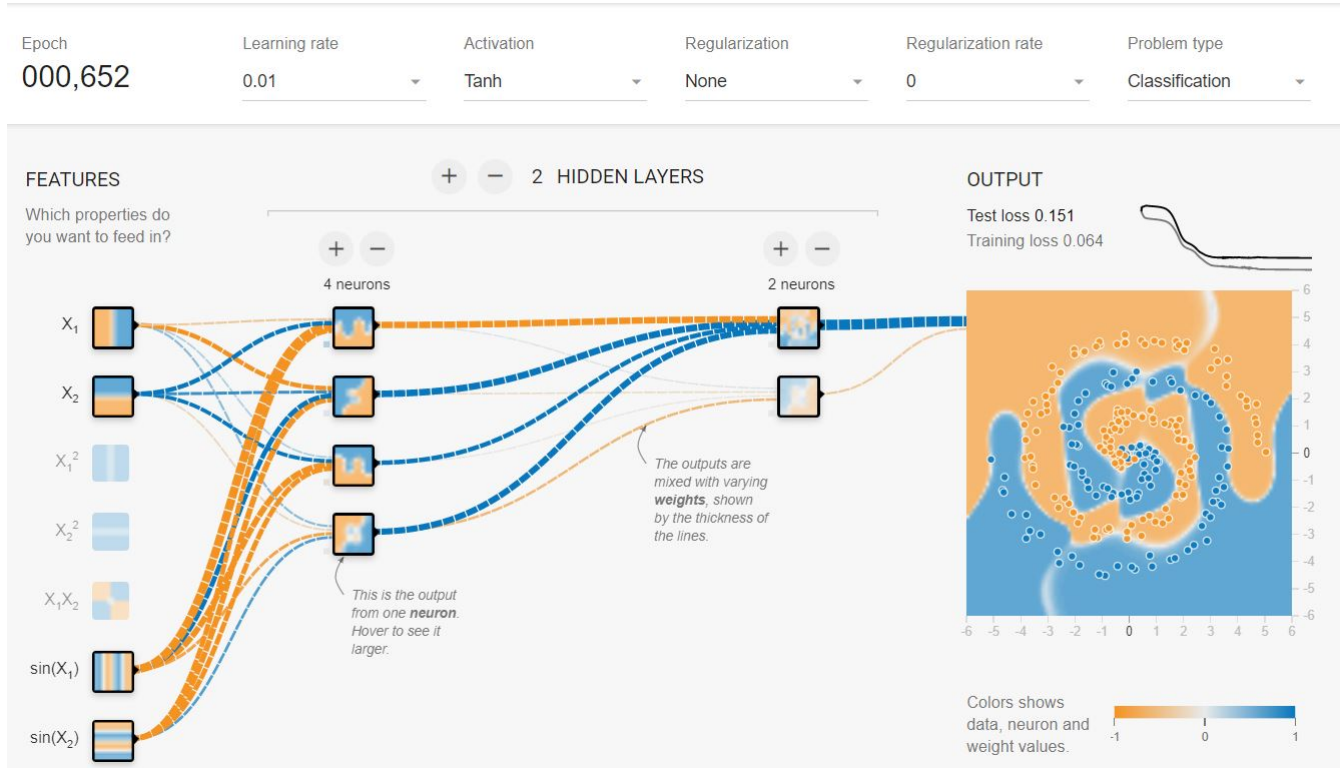


Figure 4: Best feature for dataset 4

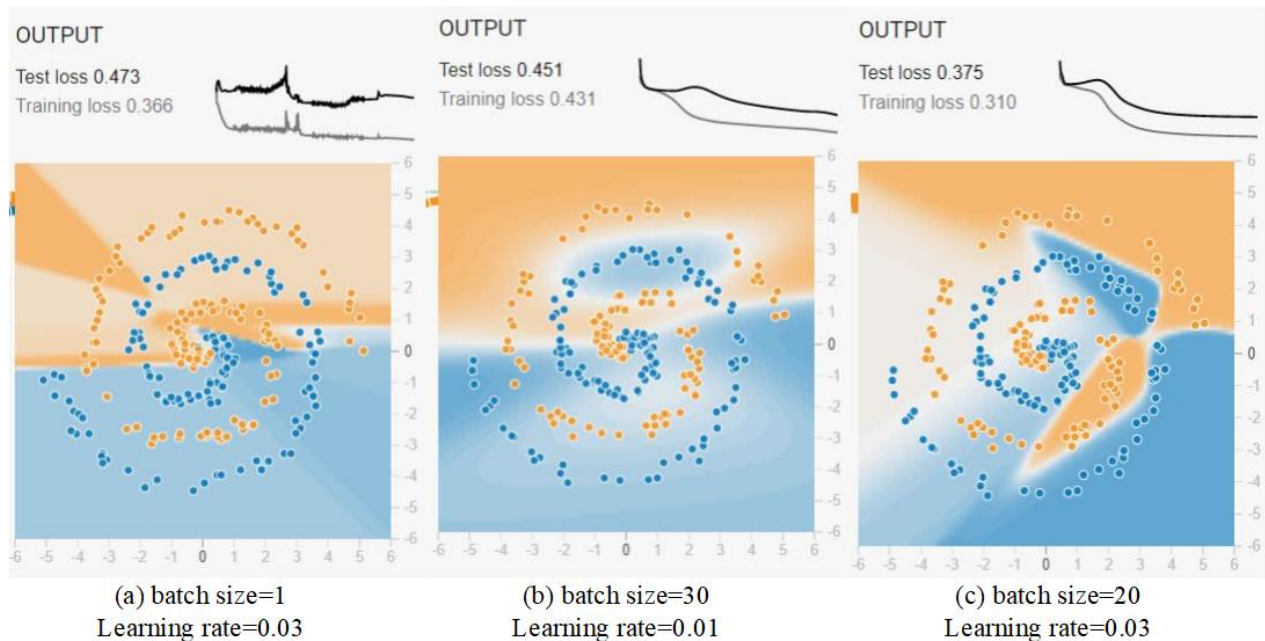


Figure 5: Some example of result for spiral dataset w.r.t different batch size and learning rate

### 3. Try all the different activation layers, what are their differences? Why?

Plots of different activation functions are shown in Fig.6. Linear activation actually is not a kind of (non-linear) activation so it only works for the linear separable dataset 3. Tanh is the best activation function

Table 1: Convergence speed and loss w.r.t different batch size and learning rate

Batch size	Learning rate	converged epoch	converged test_loss	converged train_loss
1	0.03	/	0.47	0.36
10	0.03	2300	0.44	0.31
20	0.03	2600	<b>0.37</b>	<b>0.31</b>
30	0.03	4000	0.46	0.44
1	0.01	/	0.45	0.29
10	0.01	5500	0.44	0.4
20	0.01	6000	0.4	0.28
30	0.01	7000	0.45	0.43

for the other datasets with regard to the final accuracy and converge speed. Sigmoid is a bit slower than Tanh. ReLU is not faster than Tanh on these simple fully connected networks because all those neurons with negative input would have a zero output, thus creating dead neurons that never get activated.

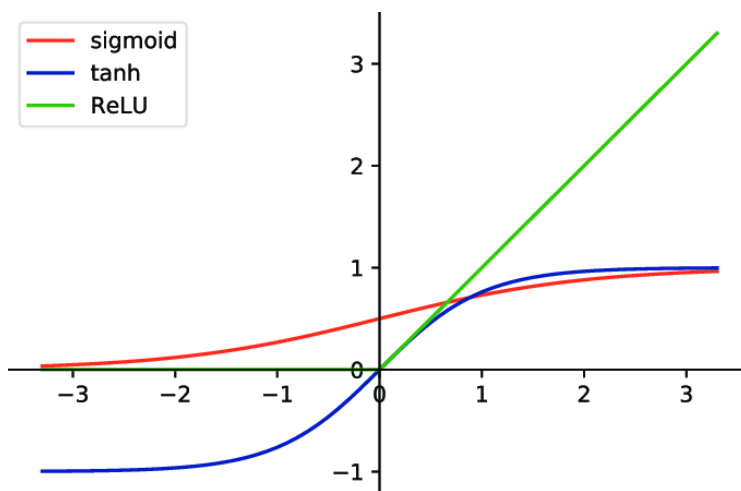


Figure 6: Plot of different activation functions

#### 4. How does the number of hidden layers and the number of neuron per layer affect the results in terms of quality (test loss) and convergence speed?

When using just the two naive feature  $x_1$  and  $x_2$ , more hidden layers and more number of neuron per layer would reach better result (lower loss) for those non-linear separable datasets. This means that with a deeper and complex enough neural network with some kind of non-linear activation function, we can fit all kinds of non-linear functions with regard to the input feature (in this case,  $x_1$  and  $x_2$ ). However, compare to using some simple non-linear features such as  $x^2$  and  $\sin(x)$  with a shallow and simple network, the former one would be more time-consuming and more likely to overfit. The results are shown in Fig.7 and Fig.8.

#### 5. Compare the performance (convergence rate and accuracy) of a network for regression and classification? Explain the differences.

Unfortunately, the datasets for regression and classification are not the same so I generally divide them into linear and non-linear problem. For the linear problem (dataset 3 for classification and dataset 1 for regression), the convergence rate and accuracy is similar for classification and regression. But for the non-linear problem



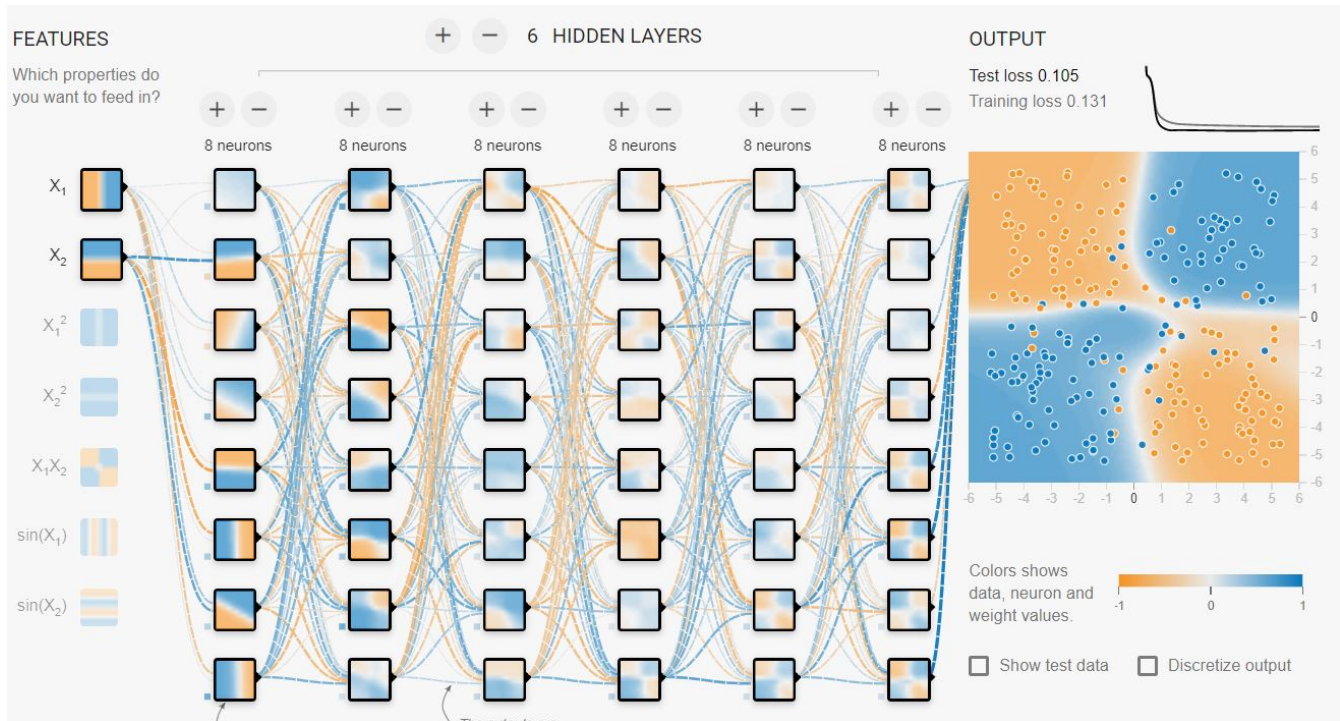


Figure 7: Use a six-layer (each has 8 neurons) network with only  $x_1$  and  $x_2$  features to train the NOR dataset

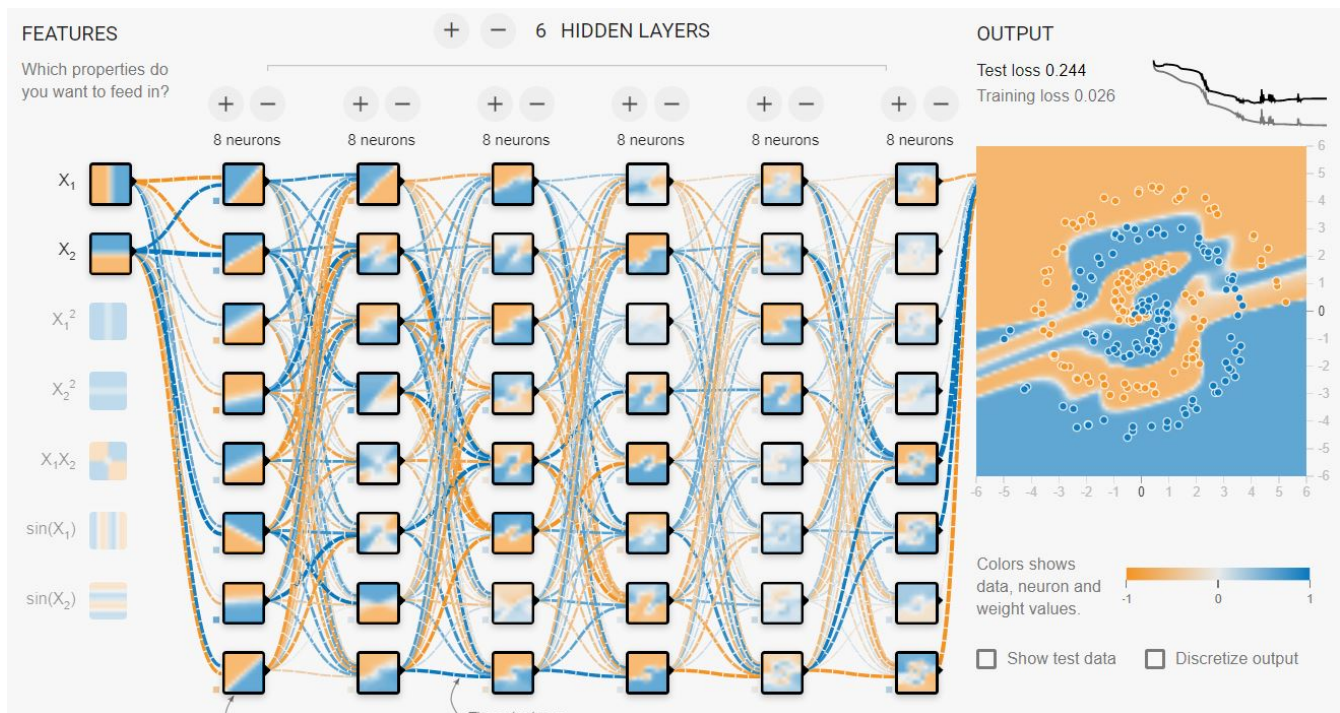


Figure 8: Use a six-layer (each has 8 neurons) network with only  $x_1$  and  $x_2$  features to train the spiral dataset

(all the rest datasets), the network has better performance on classification than regression. Figures showing the result of regression are shown in Fig.9 and Fig.10.

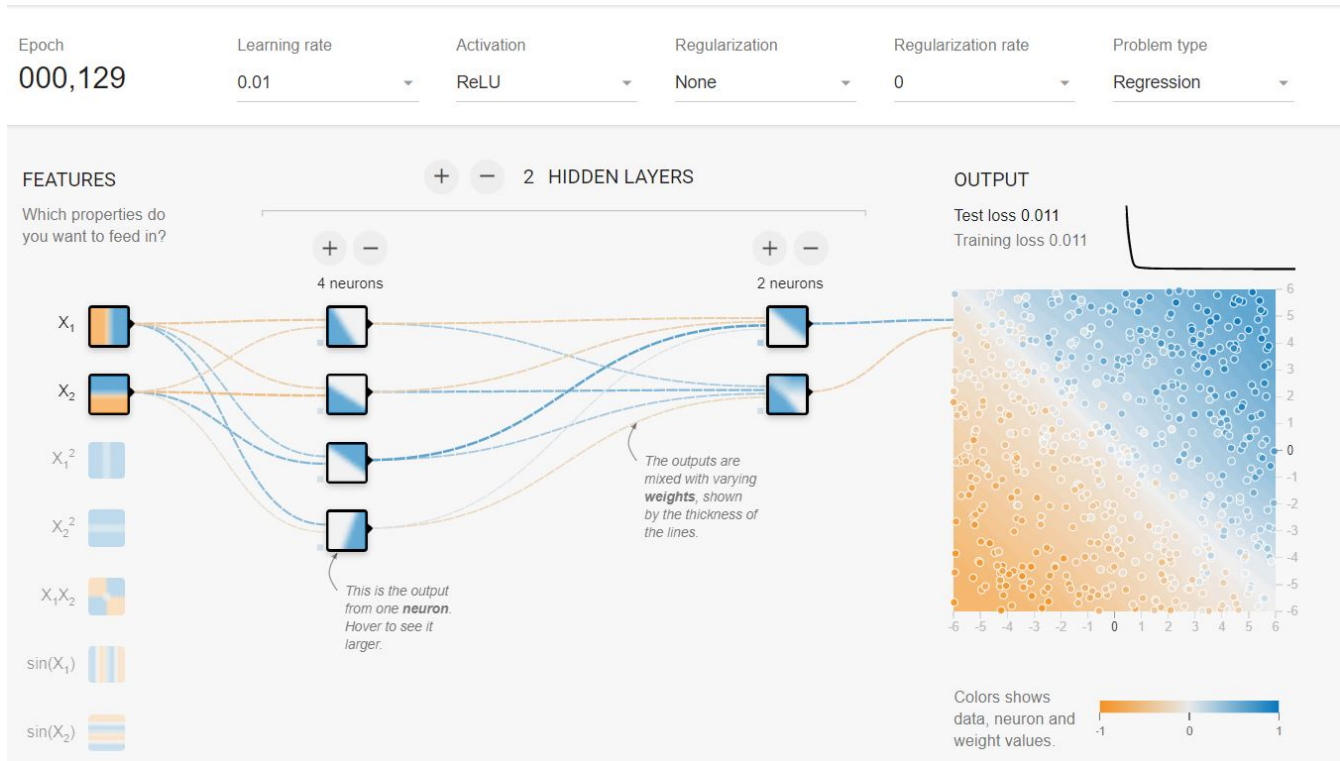


Figure 9: Regression result for regression dataset 1

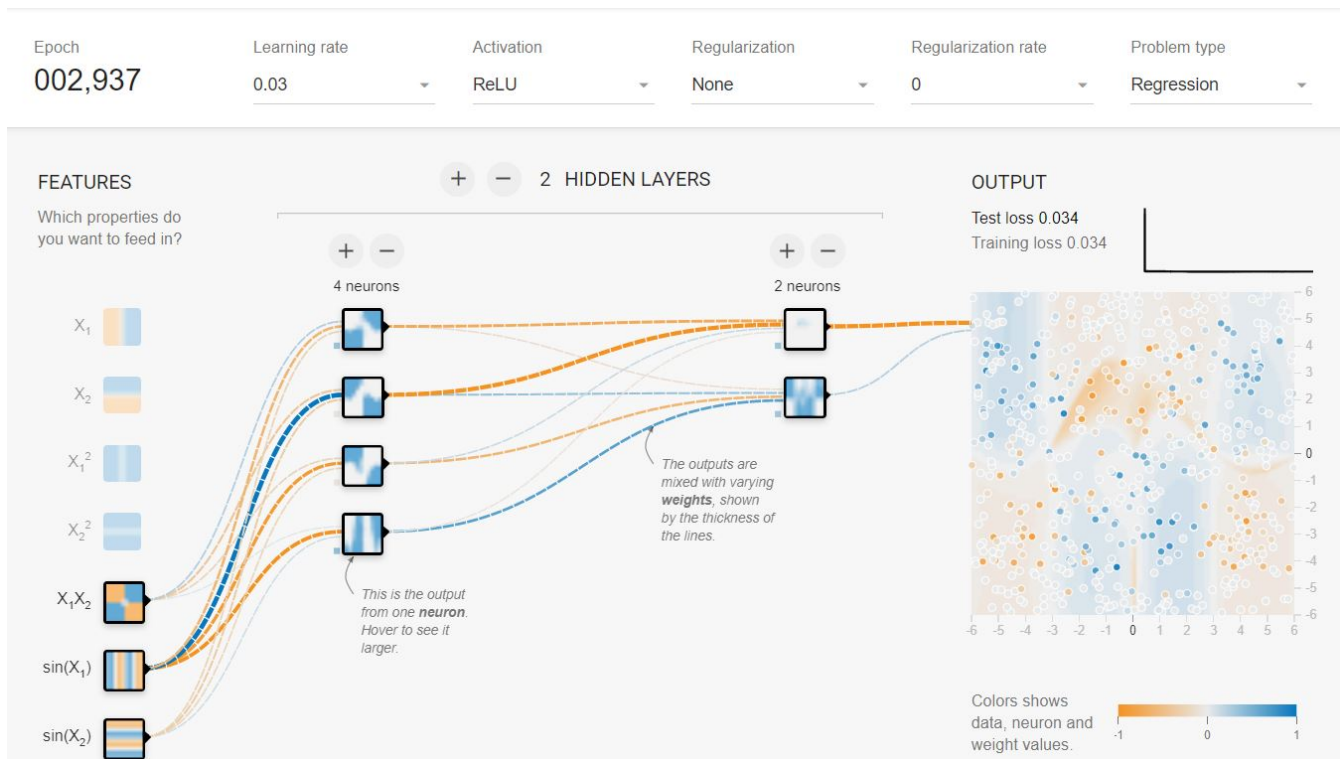


Figure 10: Regression result for regression dataset 2

6. Play with different types and rates of regularization (top slider) and explain the changes you observe and the influence on the results.

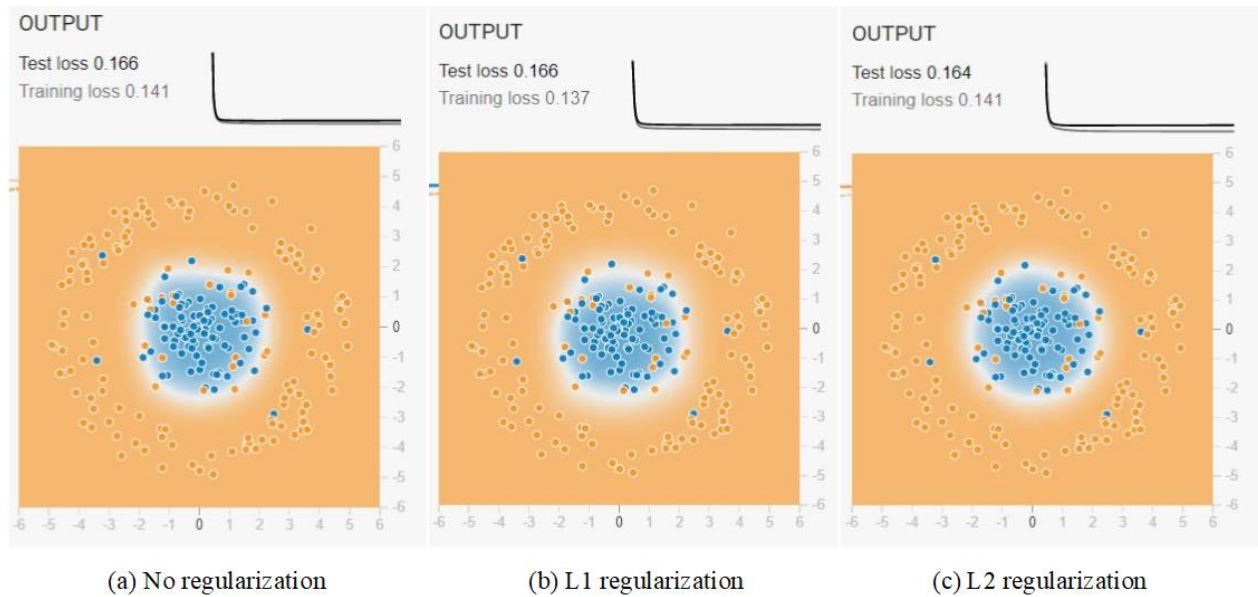


Figure 11: Different type of regularization's result on circle dataset

According to Fig.11, by using L2 regularization, the testing loss decreases a bit. By using L1 regularization, the training loss decrease a bit. Theoretically, regularization can effectively ease the problem of overfitting. However, in my experiment, the result is not significant.

## Exercise 2

Topic: CNN for MNIST dataset

For this exercise, please refer to *ii\_cnn\_mnist.py*.

I train a simple CNN to solve this problem and achieve a final testing accuracy as high as 0.993, which is higher than the basic requirement of 0.99. The accuracy and lost during training and testing are shown in Fig.13.

The architecture of my CNN is shown in Fig.12. Two convolution layers and two max-pooling layers are involved in the network. Then a fully connected layer is adopted to get a 256 dimensional feature vector, followed by another fully connected layer to get the 10-class prediction probability. The activation function used in this network is ReLu. Besides, softmax is used to project the prediction score to a categorical probability.

For other parameters, the batch size is set as 128 and the epoch number is set as 16. An adaptive training optimizer Adam is adopted to train the network self-adaptively and efficiently. Besides, I trained the network on GPU (Nvidia Quadro M1200, 4G memory, Compute Capability 5.0).

According to Fig.13, the accuracy keeps increasing and the loss keeps decreasing during the training process. For larger epoch number, the testing accuracy and loss almost dose change any more, which means the neural network has converged. An interesting observation is that the test accuracy is not always lower than the training accuracy as our intuition. For the first several epochs, the testing accuracy is a bit higher than the



training accuracy. There are mainly two reasons for this phenomenon: 1. Training accuracy is the mean accuracy of this training epoch while the test is done after the training of this epoch. When the network begin to converge, the training accuracy would gradually surpass testing accuracy. 2. Some regularization operations like dropout may also cause the training accuracy lower.

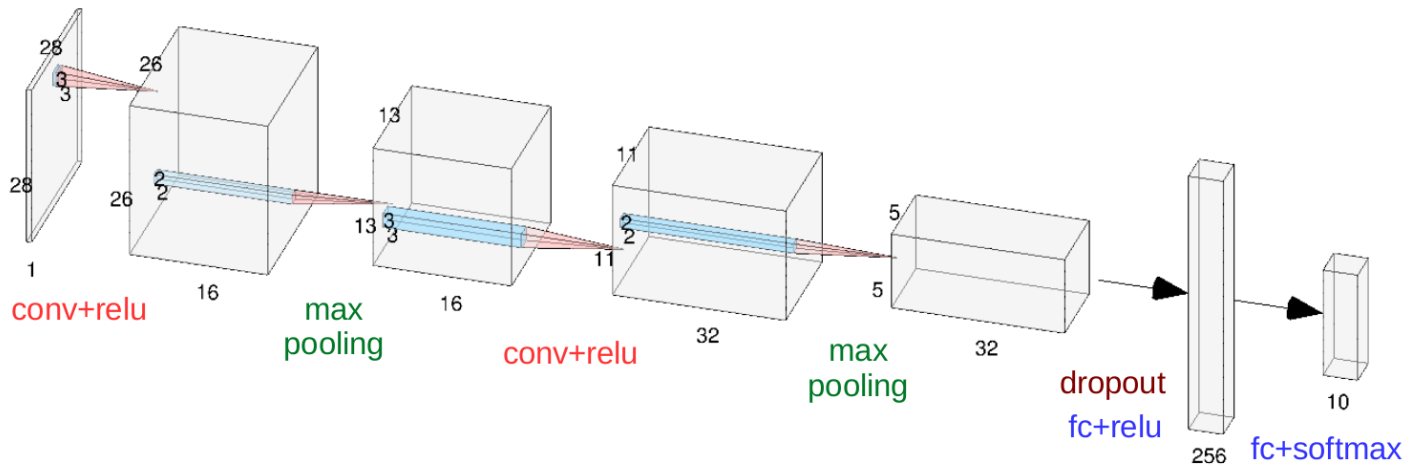


Figure 12: My network's architecture

## Exercise 3

Topic: Kaggle Dogs vs Cats Competition

For this exercise, please refer to *ii\_prepare\_data\_kaggle.py*, *ii\_pretrain\_kaggle.py* and *ii\_transfer\_learning\_kaggle.py*. Up to 20/11/2019 23:55, my submission is at the top of the leaderboard with an accuracy of 0.99519, as shown in Fig.14.

In this competition, I take advantage of the idea of transfer learning.

Firstly, I prepared and divided the data. Then, I used four well-known neural networks (VGG19, Resnet50, InceptionV3 and Xception) pretrained on ImageNet to extract feature vectors for all the training, validation and testing images. After that, for each image, I concatenated the feature vectors predicted by the 4 networks of each image into a longer feature vector. Finally I trained a simple two-layer fully connected network with dropout layer (for easing the problem of overfitting) and take the feature vector as input to do the binary classification. A sigmoid activation function help project the result to 0-1, thus accomplishing the classification. This classification neural network is fast and easy to train and predict. It only took about 15 seconds to train this network and can predict the label of testing data in just 2 seconds.



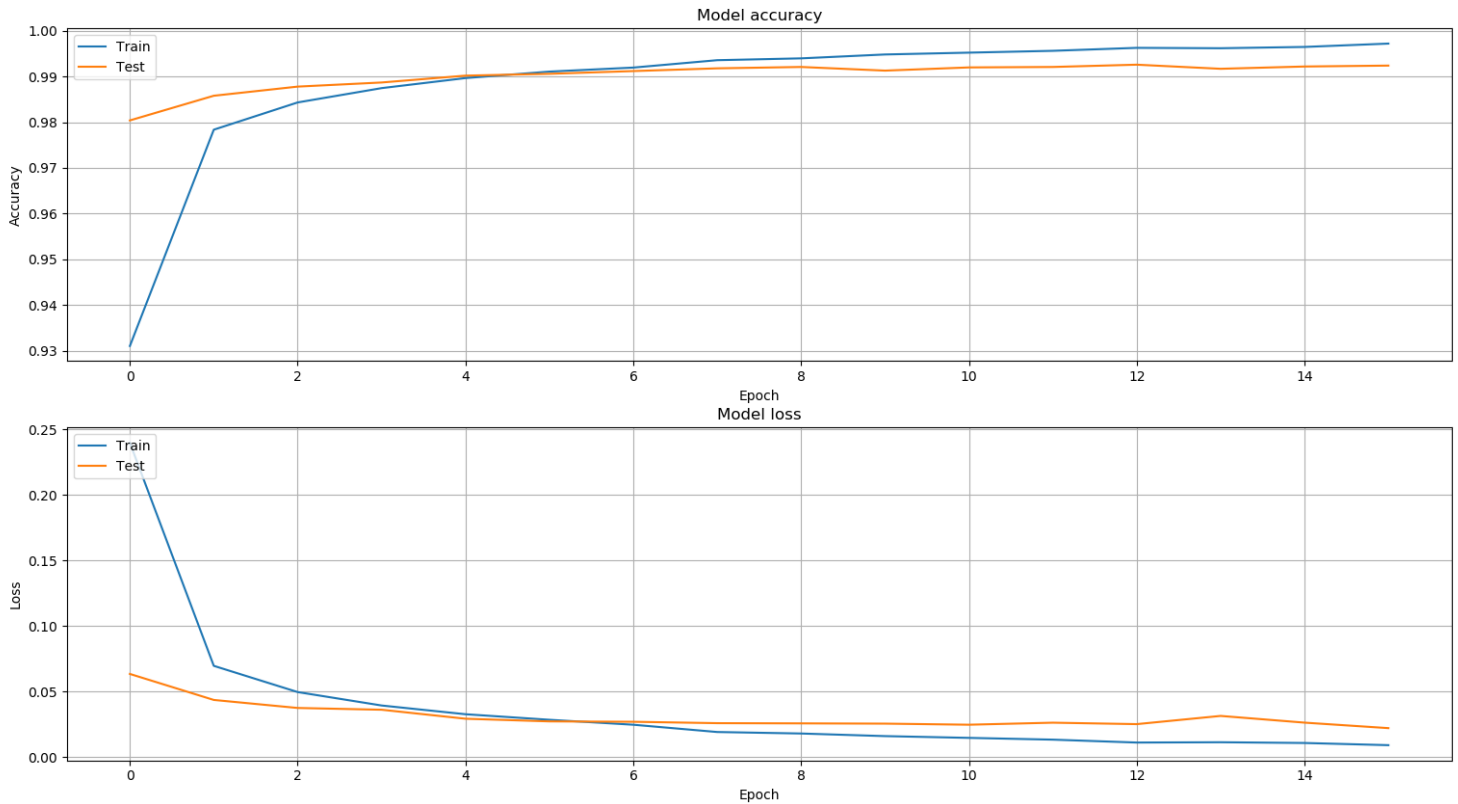


Figure 13: Test and training accuracy and lost w.g.t epoch number




#	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	Yue Pan			0.99519	5	1d
<b>Your Best Entry ↑</b> Your submission scored 0.99519, which is an improvement of your previous score of 0.99466. Great job! <a href="#">Tweet this!</a>						
2	hansun			0.99413	7	9h
3	Nicholas Meyer			0.99092	4	2h

Figure 14: Leaderboard of the competition at 20/11/2019 23:55