



Assignment Title

CS 440/640 Programming assignment 1

Name: Ziran Min

Email: minziran@bu.edu

BU ID: U59274427

Date: 02/27/2018

Problem Definition

After having learned the basic knowledge of Logistic Regression and Neural Network in class, I'm interested in using real datasets to build simple neural networks. Now I get three data sets, the linear one, the nonlinear one, and the hand-written digits recognition one for challenge. In this project, I will study on how the number of hidden layers, the number of nodes in hidden layers, and the value of learning rate affect the performance of a neural network. This is very important because neural network can greatly help us to build models and make predictions, and I think these three factors are crucial for the accuracy of prediction.

Before I start the project, I hypothesize that adding hidden layers into a neural network can improve the accuracy of a prediction, and adding more nodes into one hidden layer and lowering learning rate can also improve the performance of a neural network. My conclusion maybe different with my hypothesis.

Furthermore, the anticipated difficulties are how to implement forward propagation, back propagation, cost function, etc., in programming language in Python, and how to display the effects of different parameters on a neural network. Also, I'm worried about the underlying problem of overfitting. I will try to use L2 regularization to deal with it.

Method and Implementation

To build a neural network, I need to do the following few main steps:

1. Load and visualize the data
2. Initialize the weights of inputs and bias in each layer
3. Initialize the model
4. Calculate the cost of the model and visualize the decision boundary
5. Use forward propagation and back propagation under gradient descent to train the model
6. Visualize the decision boundary again
7. Compute confusion matrix with accuracy of the model and also the cost again

The main functions I need are weight initialization function, cost function, predict function, fit function, and decision boundary function. Furthermore, to add hidden layer into a neural work, I need to make changes on every function we have except the function for plotting decision boundary.

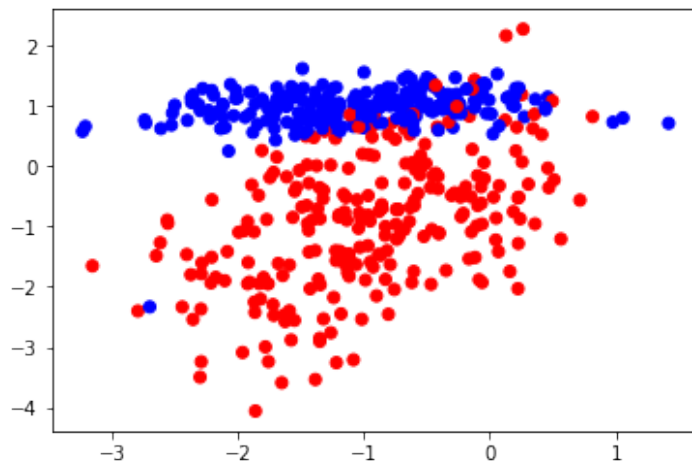
To see how the number of nodes in a hidden layer and learning rate affect a model, I decide to build for loops to compute the accuracy and cost of a model at different number of nodes and different values of learning rate for several times and compute each average, then make plots for better visualization.

Experiments & Results

Step 1: Linear data in 2-Layer Neural Network

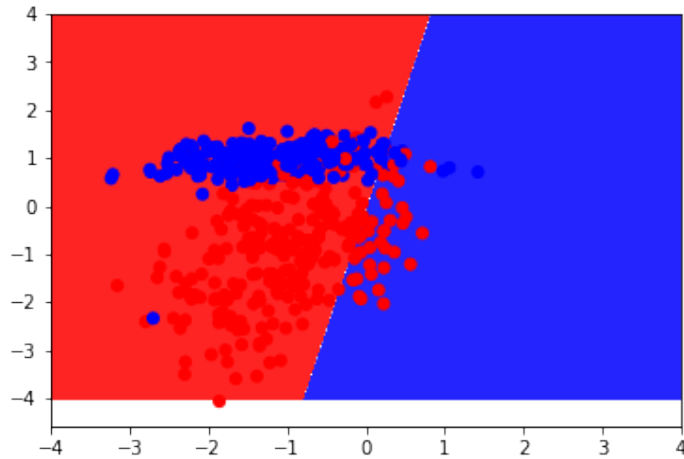
First, I use linear dataset (DATA/LinearX.csv, DATA/LinearY.csv) to build a 2-layer neural network (no hidden layer). I build a LogisticRegression class with initialization function, cost function, predict function, fit function, and also a decision boundary function for the 2-layer neural network.

I load and plot data, then get following plot:



This seems that I can use a straight line to separate two classes, but I will see this more careful later.

I initialize the 2-layer neural network model with the data and check the cost, accuracy, and decision boundary for a first glance:



ACCURACY: 0.172

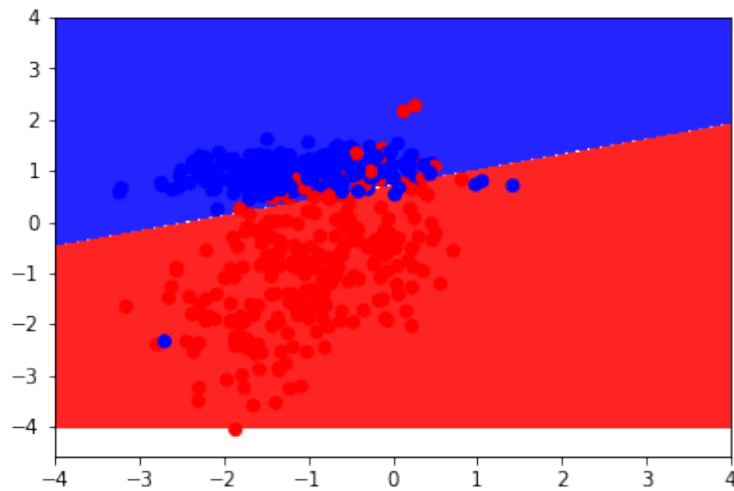
CONFUSION MATRIX:

```
[[ 2. 166.]
 [248. 84.]]
```

1.75243052997

The decision boundary shows a very bad classifier. The cost per sample is 1.75 which is almost the double of (1-0) and the accuracy is 0.172 out of 1 which is very low.

Now I need to train my model. After training, I get the following :



ACCURACY: 0.93

CONFUSION MATRIX:

```
[[ 244. 29.]
 [ 6. 221.]]
```

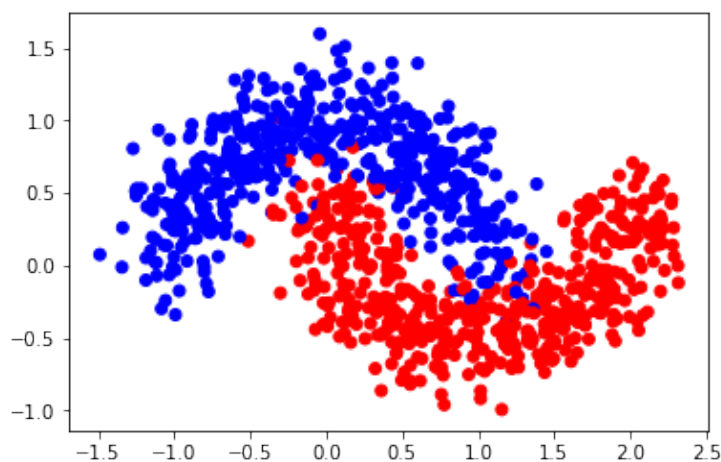
0.207171887793

Now I can see most of the blue dots and red dots are classified separately in two groups. The accuracy rises to 0.93 and the cost decreases to 0.2. I think I build a very good 2-layer neural network without hidden layer.

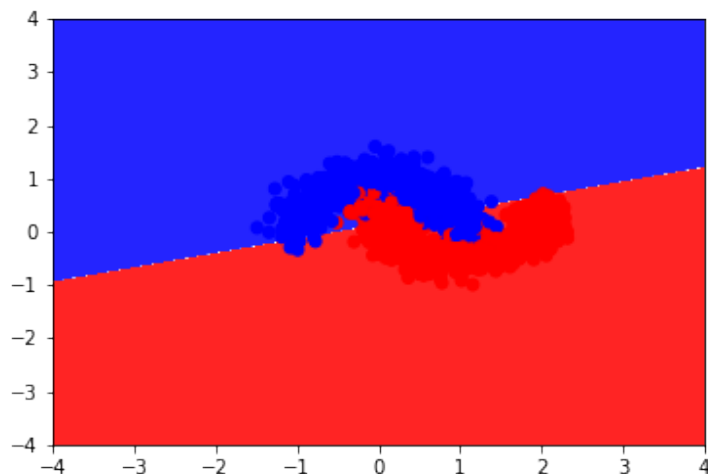
Step 2: Nonlinear data in 2-Layer Neural Network

Now I'm considering whether this kind of 2-layer neural network without hidden layer can predict nonlinear data as well. So we do the same thing in Step 1 for the nonlinear dataset (DATA/NonlinearX.csv, DATA/NonlinearY.csv).

I load and plot data, then get following plot:



After I train the model, I get the following decision boundary, accuracy, and cost:



ACCURACY: 0.875

CONFUSION MATRIX:

```
[[ 437.   62.]  
 [  63.  438.]]
```

0.290882470401

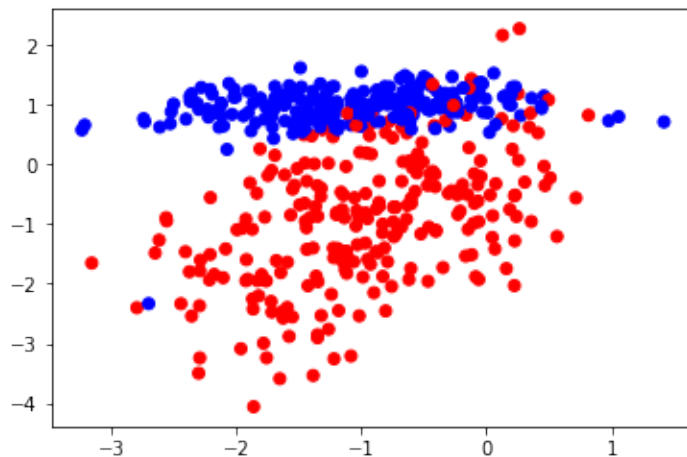
From the data plot, I know that this nonlinear datasets needs a nonlinear decision boundary to separate two classes. However, my 2-layer neural network model can't do that perfectly because without any hidden layers it can only compute a straight line as decision boundary. Therefore, I'm encouraged to build a 3-layer neural network with one hidden layer to get better decision boundary for the dataset.

Step 3: Neural Network with one Hidden Layer

To build a 3-layer neural network with one hidden layer, I need to create a new class called `NeuralNetwork` with functions that are slightly different from the functions `LogisticRegression` class because I need to add nodes in the hidden layer, assign weights to the new edges, and also compute new activation function.

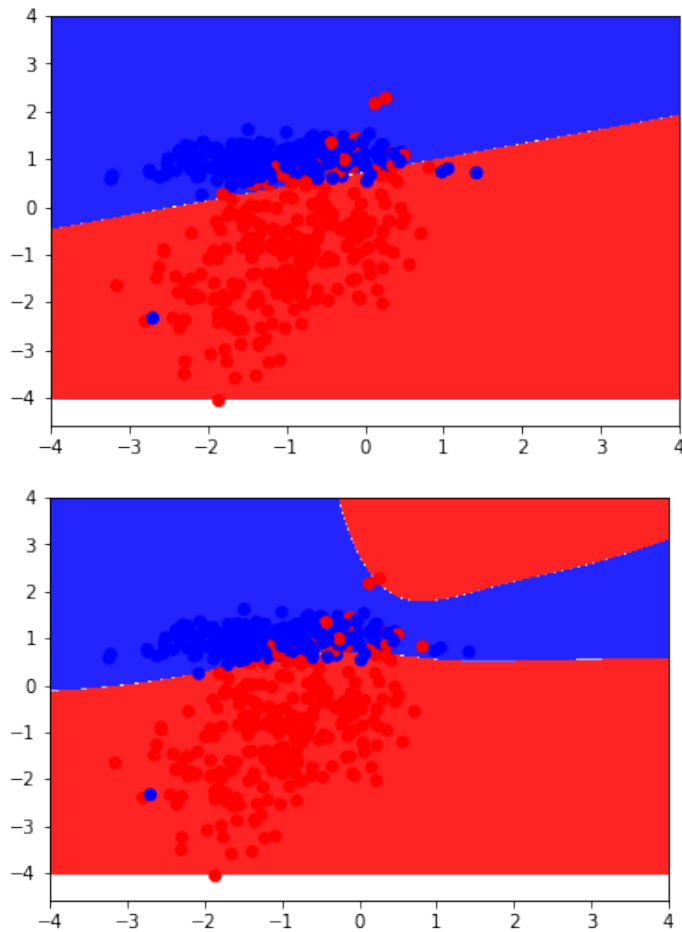
Note: For the 3-layer neural networks I build in this step, I add 20 nodes in the hidden layer by default.

I first build 3-layer neural network for the linear dataset.



computed in Step 1 and here:

Compare the decision boundary



The new accuracy and cost:

```

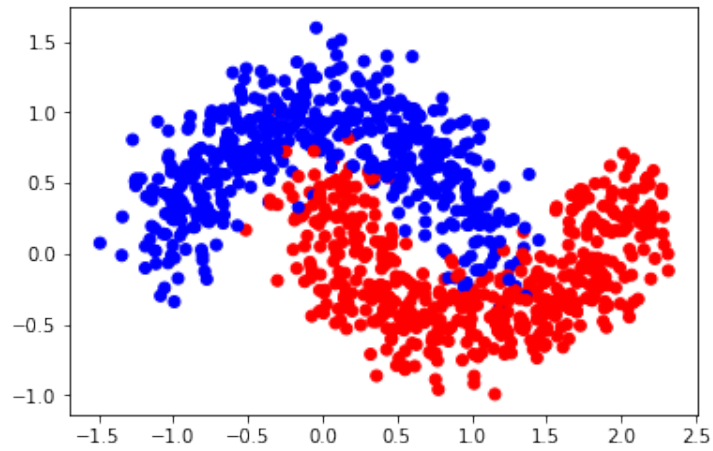
ACCURACY:  0.948
CONFUSION MATRIX:
[[ 244.   20.]
 [   6.  230.]]

```

```
0.135282074823
```

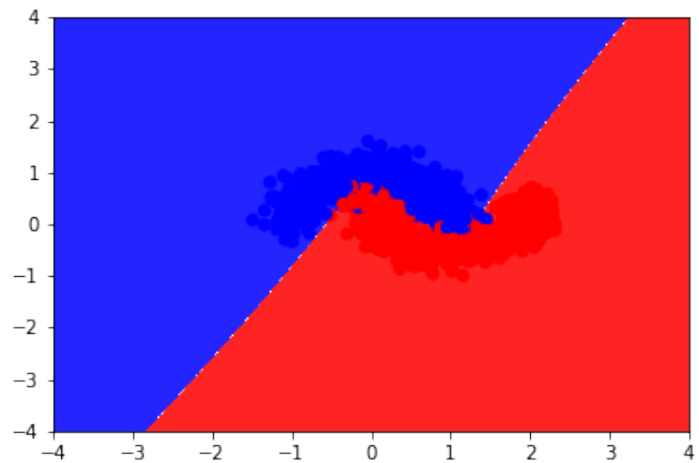
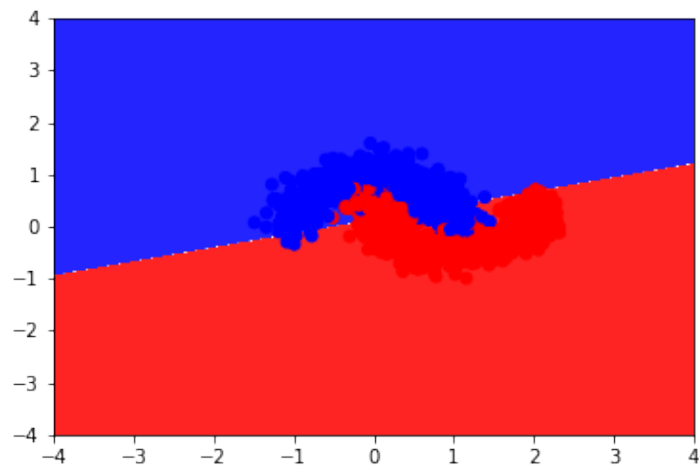
By looking at the decision boundary produced by our 3-layer neural network, I find that it's very different with the one generated by the 2-layer neural network. The one produced by my 3-layer neural network has a specific red region to classify the few red dots above the blue dots but my 2-layer neural network doesn't have that. Furthermore, I find that the accuracy rises and the cost decreases. So I think that adding a hidden layer into a neural network can make a better model to do classification.

Then I decide to train the 3-layer neural network for the nonlinear dataset to see whether I can have the same conclusion I get above.



Compare the decision boundary

computed in Step 2 and here:



The new accuracy and cost:

```

ACCURACY:  0.967
CONFUSION MATRIX:
[[ 483.   16.]
 [  17.  484.]]

```

0.0832015287631

Compared with the decision boundary produced by our 2-layer neural network in Step 2, our 3-layer neural network computes a nonlinear decision boundary that can better separate the two classes. The accuracy rises to 0.967 and the cost decreases to 0.08.

Therefore, I conclude that a 3-layer neural network with one hidden layer can learn non-linear decision boundary better than a 2-layer neural network without hidden layer, and the former can produce higher accuracy and lower cost than the latter.

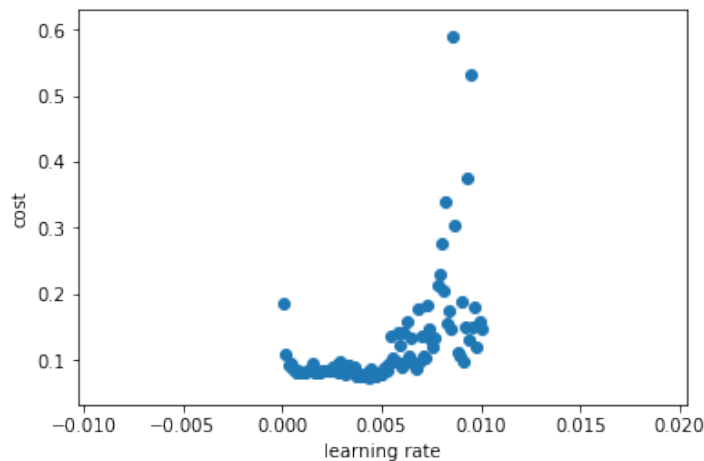
The main reason is that sometimes a neural network can't directly understand the inputs well and adding hidden layers can transform original inputs into more useful inputs for different activation functions that can produce more accurate output. For example we want to build a picture classifier to determine whether the input picture is a bus, if we have wheel detector (to help tell you it's a vehicle) and a box detector (since the bus is shaped like a big box) and a size detector (to tell you it's too big to be a car) as nodes in a hidden layer, they can help the model better identify busses.

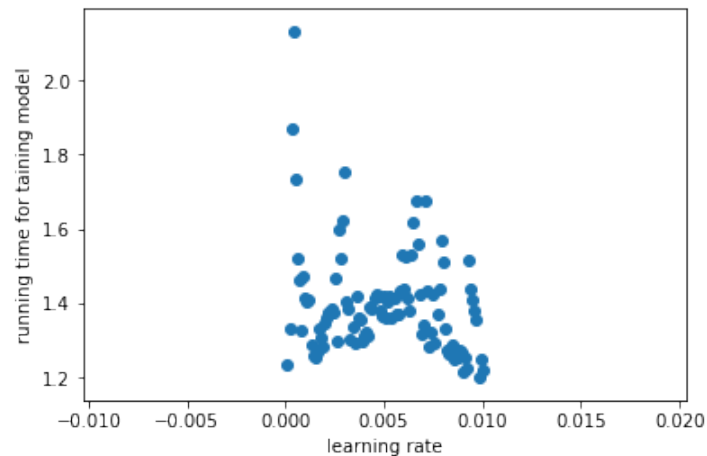
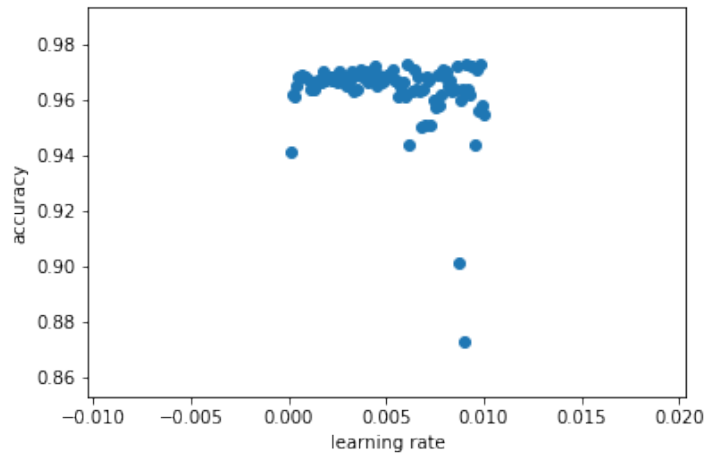
Step 4: The Effect of Learning Rate

Now I'm interested in what effect the learning rate has on the 3-layer neural network I trained. I choose to use the nonlinear dataset and I want to see how the cost and the accuracy change by varying the value of learning rate. I will use for loops to assign different learning rate and compute each cost and accuracy the model computes, and then plot them to see the trend.

I choose learning rates as 100 numbers from 0.0001 to 0.01 (like 0.0001, 0.0002, ..., 0.0099, 0.01).

After running my algorithm, I get the plots of learning rate vs. cost, learning rate vs. accuracy, and learning rate vs. running time for training model as following:





From the first two plots above, I find that when the learning rate gets larger, the cost will get larger (seems exponentially) and the accuracy diverges more (higher probability of having a lower accuracy).

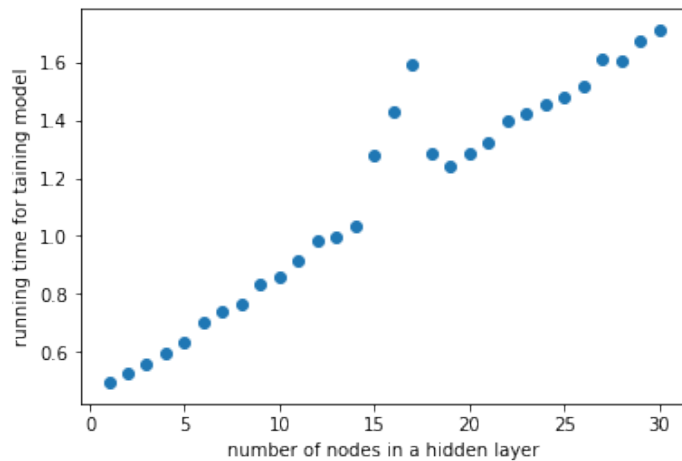
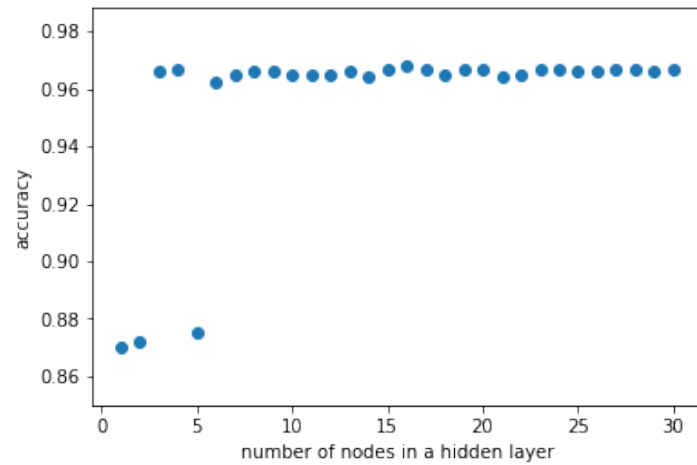
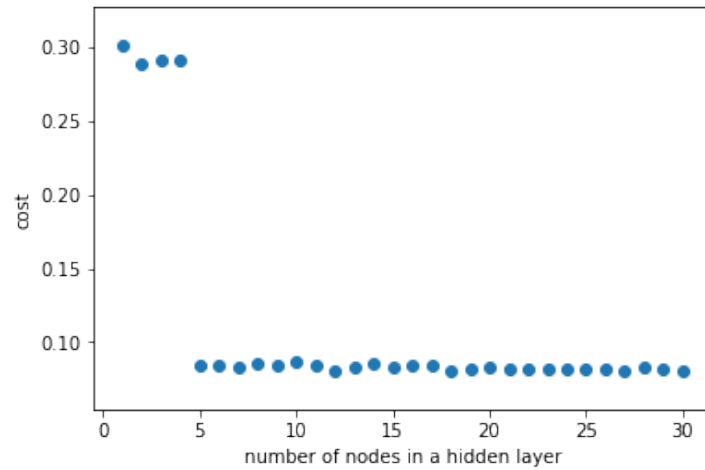
According to the Machine Learning course taught by Andrew Ng on Coursera, the reason why high learn rate results very high cost and low accuracy is that, when we training a model, gradient descent can overshoot the minimum if high learning rate is too high. It may fail to converge to the minimum or even diverge. Therefore, we should keep rate low enough. However, if the learning rate is too low, the running time for gradient descent to find minimum will be too long (this is also why in the third plot, the running time could be very high as learning rate becomes very close to zero). So we should keep learning rate neither too high nor too low.

Step 5: The Effect of Number of Nodes in a Hidden Layer

To move on, now I want to analyze how the number of nodes in a hidden effect the 3-layer neural network I trained.

I still use the nonlinear dataset and use for loops to compute neural networks with different numbers of node, then see how the decision boundary, accuracy, cost, and running time change.

After running my algorithm, I get the plots of number of nodes in a hidden layer vs. cost, number of nodes in a hidden layer vs. accuracy, and number of nodes in a hidden layer vs. running time for training model as following:



From the first two plots above, I notice that when the number of nodes is less (or equal to) than five the model has relatively higher cost and lower accuracy. However, once the number of nodes is above 5, the cost decreases to a level and sticks around there as number of nodes increases. Similarly, once the number of nodes is above 5, the accuracy increases to a level and sticks around there as number of nodes increases. Therefore, I think I don't need to initialize the number of nodes in hidden layers as 20, maybe 5 or 6 is good enough. Along with the third plot which shows the more nodes I add, the longer it takes to train the model. I become more confident that I don't need to have 20 nodes in the hidden layer. That is too many.

Step 6: Overfitting

The effect of the number of nodes on a neural networks reminds me the problem of overfitting when we do modeling. Overfitting means that we use so many features as inputs that we build a model that fits too closely to the training dataset but it doesn't fit well to the testing dataset, or any other samples.

According to the lecture slides, there are three ways to reduce overfitting:

1. Use “wrapper” to enumerate models h according to model size (e.g., number of nodes in neural net h). Select model with smallest error.
2. Feature selection: Simplify model by discarding irrelevant attributes (dimensionality reduction).
3. Minimum description length: Select model with smallest number of bits required to encode program and data.

According to the Machine Learning course taught by Andrew Ng on Coursera, we can reduce overfitting by:

1. Reducing the number of features:
 - manually select which features to keep
 - use model selection algorithm
2. Regularization:
 - keep all features but reduce magnitude/values of each parameters θ s; this works well when we have a lot of features and regularization makes each of them contribute a bit to predict y .

Step 7: L2 Regularization

Now I want to use L2 regularization based on my Step 3 to see whether it can improve my 3-layer neural network.

To compute L2 regularization, I need to add norm 2 penalizing term in my cost function and also make some change in my fit function under the NeuralNetwork class. So I create the NeuralNetwork_L2 class.

This time, in order to enlarge the problem of overfitting, I split the nonlinear dataset into training set and testing set. Furthermore, to illustrate how L2 regularization works and reduces overfitting, I will use the unchanged NeuralNetwork class to compute the model's accuracy and cost on testing set. Then I will use the modified NeuralNetwork_L2 class to get the accuracy and cost, and compare them.

By training the model under unchanged NeuralNetwork class, I get accuracy and cost:

```
ACCURACY: 0.452
CONFUSION MATRIX:
[[ 120.    3.]
 [   7.  120.]]

0.0923680263168
```

By training the model under unchanged NeuralNetwork_L2 class, I get accuracy and cost:

```
ACCURACY: 0.48
CONFUSION MATRIX:
[[   0.    0.]
 [ 127.  123.]]

0.00277298694851
```

Now we can see that by using L2 regularization, in terms of evaluating how my 3-layer neural network fitting the testing dataset, the accuracy rising from 0.45 to 0.48, and the cost decreases from 0.92 to 0.00277. Therefore, L2 regularization indeed can reduce overfitting. Furthermore, we can change the value of Lambda to modify the penalizing level for each input features.

Step 8: Hand-written Digits Recognition






In this last step of my project, I challenge myself to use the 3-layer neural network under NeuralNetwork class to realize hand-written digits recognition. By having the pixels of the picture of each hand-written digit as inputs, I want my model to compute which digit (0, 1, 2,..., 9) each input picture represents.

After loading training and testing set, and initializing and training model, I get the following confusion matrix:

```
ACCURACY: 0.9454949944382648
CONFUSION MATRIX:
[[ 86.  0.  0.  0.  2.  0.  0.  0.  0.  0.]
```

```
[ 0. 82. 0. 0. 1. 0. 1. 1. 3. 0.]
[ 0. 0. 84. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 1. 2. 81. 0. 0. 0. 0. 1. 1.]
[ 1. 0. 0. 0. 86. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 3. 0. 87. 0. 0. 3. 2.]
[ 1. 0. 0. 0. 0. 1. 90. 0. 1. 0.]
[ 0. 0. 0. 1. 0. 0. 0. 85. 0. 0.]
[ 0. 2. 0. 6. 0. 0. 0. 0. 80. 0.]
[ 0. 6. 0. 0. 3. 3. 0. 3. 0. 89.]]
```

I can also show the images of first five inputs in testing set and their predicted output and actual output.

input image	predict output	actual output
	8	8
	6	8
	4	4
	9	9
	0	0

As the above shows, my model has the accuracy of about 0.945 which is very close to 1, and in the confusion matrix, most of the numbers are lying on the diagonal. Therefore, I can conclude that my 3-layer neural network can work very well for hand-written digit recognition.

Discussion

Q: What are the strengths and weaknesses of your method?

I think my strength is having good understanding of the fundamental of neural networks both from class and by online self-study and transforming abstract knowledge into well-structured codes in Python. By using the functions I build, I can easily initialize and train any kind of neural networks I want. Furthermore, I know how to use decision boundary, accuracy, cost, and running time to evaluate the performance of a neural network model. Therefore, by familiarizing how neural network model works, how to train a model through gradient descent with forward and

backward propagation, and how to reduce overfitting by regularization fundamentally, I have the advantage to implement them technically through programming.

My weakness is in Step 4 and Step 5 where I use for loops to see how accuracy and cost change at 100 different values of learning rate and 30 different amount of nodes in hidden layer. The plots I get are good enough for me to make conclusion but there my for loops take long time to compute plots and every time there are few going points or outliers that don't follow the cluster trends. Maybe there are better algorithm for Step 4 and Step 5 (problem 4 and 5).

Q: Do your results show that your method is generally successful or are there limitations?

The result of neural network for handwritten digits recognition is quite successful. It ends with a high accuracy about 94.5%. Maybe recognizing single digit number is not a difficult job, so I would like to conduct more complicated image recognition through neural network.

On the other hand, although the L2 regularization reduces overfitting in the step 6 of my experiment, the model with L2 regularization doesn't improve so much compared with the model without L2 regularization. This may because overfitting is indeed a hard problem solve or I should improve my functions involved with L2 regularization or increase the value of Lambda.

Q: Describe what you expected to find in your experiments, and how that differed or was confirmed by your results.

I expect that neural network with hidden layer can better predict linear and especially nonlinear data than a neural network without hidden layer, and L2 regularization can reduce overfitting. My experiment confirms my those two expectation.

I also expect that the lower the learning rate is and also the more nodes a hidden layer has, the better the performance a neural network should have. However, my experiment results show that for both learning rate and node number, we should keep them neither too high nor too low.

Q: Potential future work. How could your method be improved? What would you try (if you had more time) to overcome the failures/limitations of your work?

As I mention above, in Step 4 and Step 5 (problem 4 and 5), I create for loops to see how accuracy, cost, running time differ as learning rate and nodes number change. However, I ignore that every time when I initialize a new neural network, all weights are random initialized and so are different every time. That means, even I fix the learning rate and node numbers, my result of accuracy, cost, running time will be different. Therefore, to realize scientific control as perfect as possible, I should get the accuracy, cost, running time as each earning rate and nodes number for n times (integer n should be as larger as possible), and calculate each average then plot those average numbers. In this way, I should get better plots to analyze. However, that method may take much longer running time.

Conclusions

In all, through my project, I can conclude that:

1. A neural network with hidden layer can much better predict nonlinear data than a neural network without hidden layer.
2. When we train a model, we should use a learning rate that is neither too high (causes divergence or failure of finding minimum cost) nor too low (causes long running time).
3. In a hidden layer, if the number of nodes is too few, the neural network may have low accuracy and high cost. And when we add nodes to a specific amount in a hidden layer, there is no need to add more because the performance of the network has reached a maximum level.
4. 2L regularization indeed can reduce overfitting in a neural network.
5. The 3-layer neural network I build can realize handwritten digits recognition very well with an accuracy about 94.5% through model testing.

In all, neural network is a strong machine learning technique for me to solve many data science problems and I will definitely go deeper into it in my future academic study.

Reference

Accessed the following web for explanation of hidden layers on 02/25/2018:

<https://stats.stackexchange.com/questions/63152/what-does-the-hidden-layer-in-a-neural-network-compute>

Accessed the following online machine learning course from January to February in 2018:

<https://www.coursera.org/learn/machine-learning>