EECS 349 (Machine Learning) Homework 7

Xinyi Chen

Problem 1

A)

No, it is because that perceptron's definition is a linear function, so one-layer perceptron cannot give a non-linear decision surface.

B)

No, a multi layer perceptron with linear activation functions cannot represent a non-linear decision surface. It is because if a multilayer perceptron has a linear activation function in all neurons, then it is easily proved that any number of layers can be reduced to the standard two-layer input-output model, this simple model cannot represent non-linear decision surface.

C)

Yes, sigmoid is easier to differentiate than linear function, so the parameters to the multi-layer net can be learned easier by sigmoid.

D)

This would make back propagation of error training method cannot work. Because back propagation of error training method requires that the activation function used by the artificial neurons must be differentiable(a differentiable function of one real variable is a function whose derivative exists at each point in its domain), but that activation doesn't have a derivative at 0.

Problem 2

A)

A Restricted Boltzmann Machine is a stochastic neural network consisting of one layer of visible units, one layer of hidden units and a bias unit, with the restriction that each visible unit is connected to all the hidden units (this connection is undirected, so each hidden unit is also connected to all the visible units), and the bias unit is connected to all the visible units and all the hidden units. Most importantly,

there are no connections between nodes within a group(no visible unit is connected to any other visible unit and no hidden unit is connected to any other hidden unit).

B)

DBNs are probabilistic graphical models consisting of stacked layers of RBMs, DBNs can be viewed as a composition of simple, unsupervised networks such as RBMs where each sub-network's hidden layer serves as the visible layer for the next. Also the training algorithm for DBNs needs to use RBMs.

Problem 3

A)

Kernel Machines are shallow architectures, in which one large layer of simple template matchers is followed by a single layer of trainable coefficients, and kernel machines are non-parametric learning models, which make apparently weak assumptions on the form of the function to be learned.

One of the limitations of Kernel Machines is based on the well-known depth-breadth tradeoff in circuits design. This suggests that KM has lower efficiency of the representation of many function compared other systems. And architectures relying on local kernels also can be very inefficient at representing functions that have many variations because of mathematical consequences of the curse of dimensionality. The other limitation pertains to the computational cost of learning. In theory, the convex optimization associated with kernel machine learning yields efficient optimization and reproducible results, but most current algorithms are (at least) quadratic in the number of examples. The most serious limitation pertains to inefficiency in representation. Shallow architectures and local estimators are simply too inefficient to represent many abstract functions of interest.

B)

Deep architectures are compositions of many layers of adaptive non-linear components, in other words, they are cascades of parameterized non-linear modules that contain trainable parameters at all levels. It best exemplified by multi-layer neural networks with several hidden layers.

Deep architectures have the potential to generalize in non-local ways and that this is crucial in order to make progress on the kind of complex tasks. Deep architectures can represent certain families of functions more efficiently (and with better scaling properties). Many functions can be much more efficiently represented with deeper architectures, often with a modest number of levels such as logarithmic in the number of inputs. Deep architectures allow the representation of wide families of functions in a more compact form than shallow architectures, because they can trade space for time (or breadth for depth) while making the time-space product smaller. Also, combining a deep architecture with a Kernel Machine that takes the higher level learned representation as input can be quite powerful.

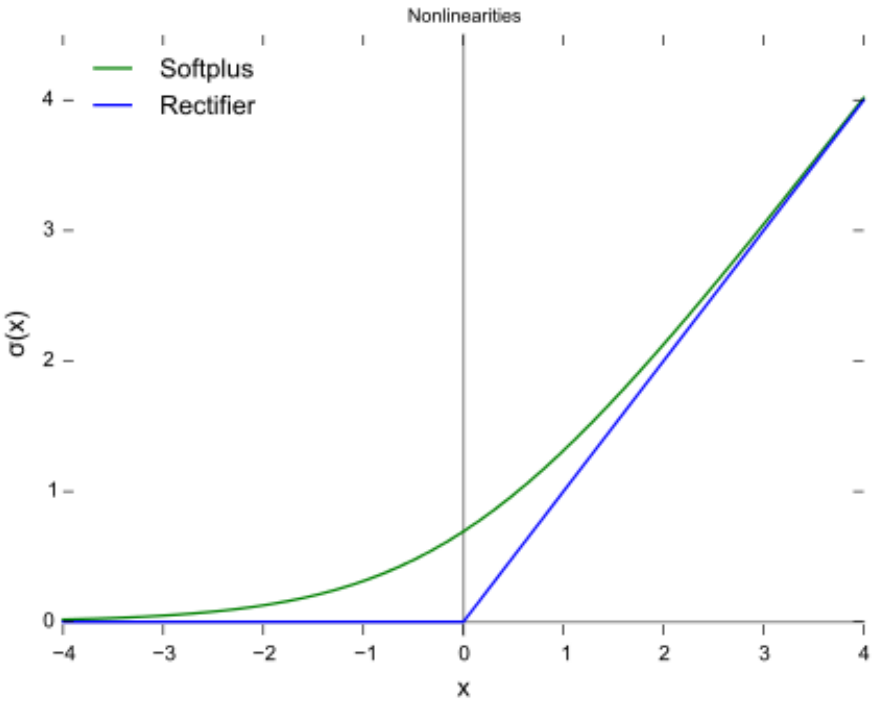
Problem 4

A)

ReLU activation function (sources: wikipedia:Rectifier (neural networks)):

$$f(x) = \max(0, x),$$

Qualitative Plot (sources: wikipidea:Rectifier (neural networks)):



B)

Softmax activation function:(sources: wikipedia: Softmax function)

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

The softmax activation function is useful predominantly in the output layer of a clustering system. Softmax functions convert a raw value into a posterior probability. This provides a measure of certainty.(sources: wikipedia: Softmax function)

Problem 5

A)

The architecture of the neural network: It has 4 layers, one input layer(6 nodes), two hidden layers(each of them has 32 nodes, activation functions both are default linear), one output layer(2 nodes, activation function is softmax). This network train for 10 epochs.

B)

Step 2. Final accuracy: 78.75%

C)

When epochs = 20, Final accuracy: 79.38%. When epochs = 100, Final accuracy: 81.39%.

D)

Chance that Leonardo DiCaprio survives: 13.02%. Chance that Kate Winslet survives: 95.08%.

Problem 6

B)

First because cross validation is too time consuming, in order to save time, I've trained without test data sets. I've trained on epoch numbers from 2, 4, 8, … , 128, 256, and batch sizes from 2, 4, 8, …, 32, 64, totally 48 combinations, find out that the top 10 combinations are as follows(all the data can be seen in linear.csv, the first column is epoch, the second is batch size, the last is accuracy without test):

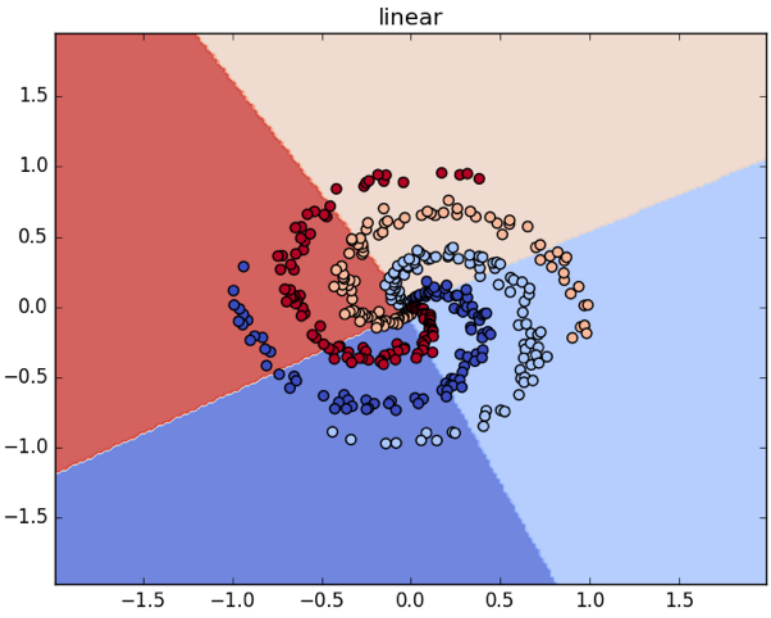
1	8	64	0.3725
2	4	8	0.3375
3	4	16	0.325
4	8	16	0.325
5	8	8	0.3225
6	2	2	0.32
7	4	4	0.3175
8	16	32	0.3175
9	2	4	0.315
10	4	2	0.315

So I do 10 folds cross validation to these top 10 combinations (*problem6_crossvalidation.py*), and the final result can be seen in *linear_cross.csv*(the first column is epoch, the second is batch size, the last is mean accuracy of 10 folds cross validation):

1	4	4	0.325
2	4	2	0.3175
3	8	64	0.3075
4	2	4	0.3075
5	2	2	0.305
6	16	32	0.305
7	4	16	0.3
8	8	16	0.295
9	8	8	0.295
10	4	8	0.2875

So the best combination is epoch = 4, batch size = 4, accuracy of the classifier after training is 32.5%.

plot:

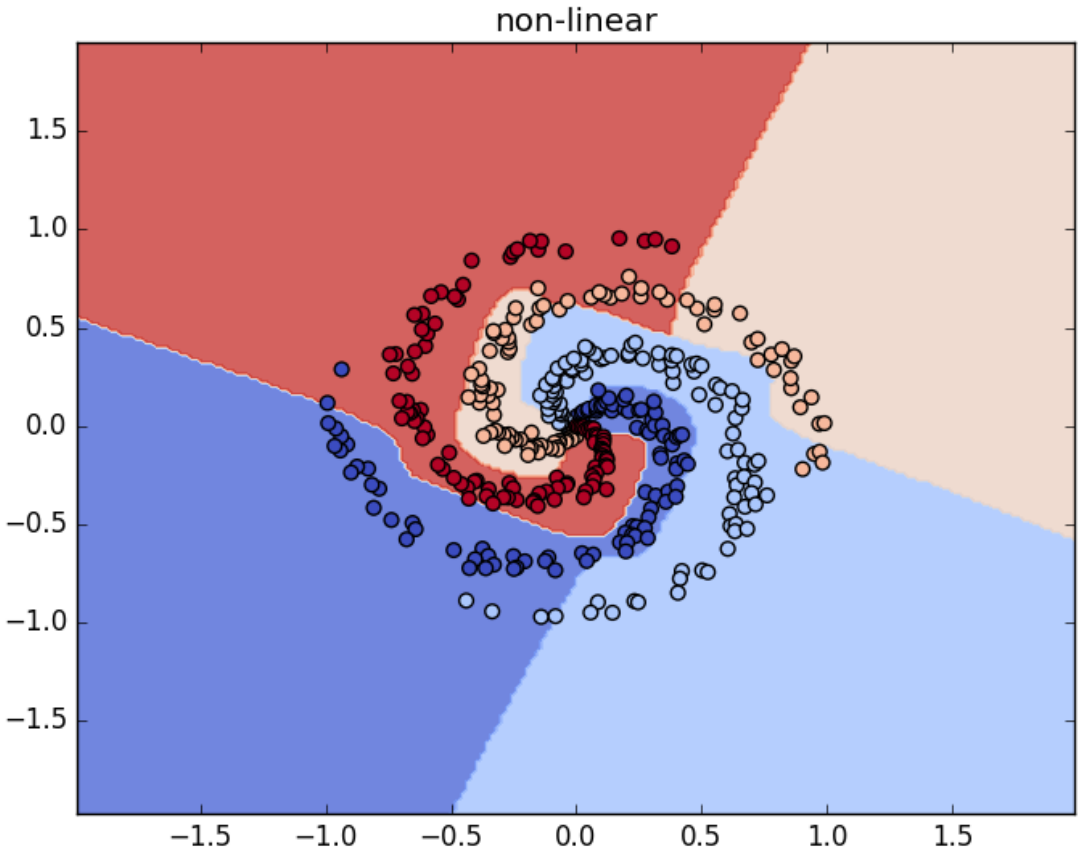


Is it impossible to correctly classify all of the data points with just one layer, because these data points are non-linear separable, one layer linear neural network can't deal with it because the definition of linear neural network limited the output is linear, so it cannot give a non-linear decision surface.

D)

In the training process I find out that when number of epochs goes up, the accuracy will be better, but when its around 200, the accuracy become stable, doesn't change better anymore.

The best parameters I use are node number for hidden layer are 64, activation function for hidden layer is sigmoid, number of epochs is 256, and batch size is 4, the mean accuracy of 10 folds cross validation is 99%, without using test data the accuracy is 96%.



E)

I did the same train in B), first using the parameters node numbers are chosen from[2 4 8 16 32 64], epochs are chosen from[2 4 8 16 32 64 128 256], batch sizes are chosen from [2 4 8 16 32 64], activations are chosen from [‘linear’,'tanh','sigmoid','softmax','relu'], and find out the best top 10 combinations without using test data, which are (the first column is node number, the second is activation, the third is epoch, the last column is accuracy without test data):

1	64	sigmoid	256	4	0.9625
2	64	sigmoid	256	2	0.9625
3	32	relu	256	16	0.96
4	64	relu	256	16	0.9575
5	64	sigmoid	256	8	0.9475
6	32	sigmoid	256	4	0.9475
7	64	relu	256	8	0.945
8	16	tanh	256	16	0.94
9	16	tanh	256	32	0.9275
10	16	relu	256	8	0.91

Then I do 10 folds cross validation on these top ten parameters, the result(the first column is node number, the second is activation, the third is epoch, the last column is mean accuracy of 10 folds cross validation):

1	64	sigmoid	256	4	0.99
2	64	sigmoid	256	2	0.9825
3	64	relu	256	8	0.9825
4	64	relu	256	16	0.98
5	64	sigmoid	256	8	0.9775
6	16	tanh	256	16	0.9775
7	32	sigmoid	256	4	0.9725
8	16	relu	256	8	0.9725
9	32	relu	256	16	0.9725
10	16	tanh	256	32	0.9725

As we can see the best is node number for hidden layer are 64, activation function for hidden layer is sigmoid, number of epochs is 256 and batch size is 4, the mean accuracy of 10 folds cross validation is 99%.