CS412

Introduction to Machine Learning

Homework Assignment - 3

Submitted By

Saikrishna Kalahasti Karthik

UIN: 655128301

Due Date: 11th March 2019

1 Neural Network with Backpropagation

Please find the submission in **src/FeedForward_NeuralNetwork.py** and **src/FeedForward_NeuralNetwork.ipynb**

2 Neural Network Testing

For the blood transfusion data, give the cross validation errors in a table form:

number_of_hidden_layers	number_of_neurons	cross_val_score
1	2	0.663387
1	5	0.550667
1	10	0.667387
2	2	0.710054
2	5	0.709351
2	10	0.747261
5	2	0.710054
5	**5**	**0.753946**
5	10	0.727351

Report the number of hidden layers and nodes per layer which provides the highest accuracy

I obtained maximum cross-validation score for **number of layers = 5** and **number of neurons = 5**.

For the digits dataset, for each of the 20 models, report the 10-fold cross validation error and the runtime in milliseconds:

Number of Hidden Layers	Number of Neurons	Run Time (ms)	cross_val_score
1	2	11461.69	0.923891
1	**5**	**8427.286**	**1**
1	**10**	**7329.898**	**1**
1	**50**	**5245.475**	**1**
1	**100**	**5272.016**	**1**
2	2	15261.4	0.964516
2	**5**	**8510.662**	**1**
2	**10**	**6404.298**	**1**
2	**50**	**7743.356**	**1**
2	**100**	**9153.54**	**1**
5	2	10160.85	0.706532
5	5	11806.83	0.925
5	**10**	**6547.332**	**1**
5	**50**	**5532.591**	**1**
5	**100**	**8845.635**	**1**
10	2	10578.38	0.631532
10	5	6969.519	0.703407
10	10	6939.218	0.889516
10	**50**	**10569.92**	**1**
10	100	13490.27	0.996875

Short answer

a) How does runtime scale with the number of layers and the number of nodes per layer? Do each have a similar effect?

Number of Nodes per Layer: At every layer, the running time is high when the number of nodes is 2. It decreases when the number of nodes is 5, 10 and 50. It increases again when the number of nodes become 100. I find a U shaped curve for running time vs number of nodes.

Number of Layers: Everytime number of layers increase, there is a significant difference in the running time. The running time vs number of layers seems to have an increasingly proportional relation.

b) What is the number of layers and nodes per layer that gives an optimum result?

We get an optimum result (CV Score = 1) for several combinations of number of layers and nodes per layer. I have **bolded** those in the table above.

c) For your optimum model, try increasing and decreasing the learning rate from the default (0.001). Discuss the tradeoff here between runtime and accuracy?

I have chosen the following model for this analysis out of several optimum models:

- **Number of Hidden Layers = 5**
- **Nodes per Layer = 10**

learning_rate	run_time (ms)	cross_val_score
0.001	6346.054	1
0.001833	3951.827	1
0.00336	2947.792	0.996875
0.006158	1725.746	1
0.011288	1713.527	0.996875
0.020691	1405.628	0.996875
0.037927	790.4911	0.959375
0.069519	850.8842	1
0.127427	558.2962	0.996875
0.233572	583.0452	0.996667
0.428133	723.0101	0.922708
0.78476	667.0139	0.662782
1.43845	774.8179	0.631532
2.636651	813.3581	0.631532
4.83293	992.491	0.631532
8.858668	1008.017	0.604866
16.23777	1327.956	0.631532
29.76351	1574.039	0.619032
54.55595	656.9831	0.544583
100	456.9573	0.499167

Run Time: As the learning rate goes higher, the run time goes down. This is because the solution converges quickly. This is because the higher the learning rate is, more the loss value oscillates around the minima. The consecutive loss values would not improve significantly for high learning rates. Therefore the loss function is optimized quickly. This is the reason for lower run time for higher learning rates.

Accuracy: When the learning rate is high, the optimization functions such as gradient descent, Adam cannot find the minima of the loss function. The algorithm would converge in some point much larger than the minima. Since the loss function is not minimized well as the learning rate goes up, it is natural that the accuracy of the model goes increasingly poor.

d) Is the neural network finding the same solution every time? Why or why not? Does this have an impact on the expected fit?

No. It does not. Here is the weights of the last hidden layer after the data is fit for 3 different executions of the same MLPClassifier with the same attributes.

```
[array([0.40413]),          [array([1.06677]),          [array([0.11537]),
  array([0.4672]),            array([-0.66866]),          array([-0.99262]),
  array([0.24687]),           array([0.65685]),           array([0.10557]),
  array([0.39373]),           array([-0.64589]),          array([1.00164]),
  array([0.8687]),            array([0.47357]),           array([-0.36797]),
  array([-0.69721]),          array([0.21951]),           array([0.50706]),
  array([0.13567]),           array([-0.85999]),          array([-0.65387]),
  array([0.53513]),           array([-0.32023]),          array([0.6053]),
  array([-0.86064]),          array([-0.54763]),          array([0.47159]),
  array([0.52742])],          array([-0.61148])],         array([0.65128])]
```

As you can observe, each time we get a different solution. I can think of two reasons for this behavior.

1. The “shuffle” parameter is set as True, and each time the dataset is shuffled. So the weights are calculated and corrected in the shuffled order of data points, and it is quite possible that the direction of descent towards the minima is different at every step.
2. We are also doing a batch processing. Thus, the weights correction after the backpropagation is totally dependent on the set of input vectors we received.

In short, the final weights are dependent on the order of data points processed. However, the direction of descent is always towards the minima and the solution is always converged at some reasonable neighborhood of the minima (Again this is true if the learning rate is sufficiently low).

e) Experiment with early stopping on neural networks that have a large number of internal nodes. Does this cause the model to under or over fit the data? Why?

Early stopping makes the model to stop in fewer iterations if the validation accuracy does not improve. Please find an experiment with CV Score and Early stopping below:

Internal Nodes	CV Score (early stopping= False)	Number of Iterations to Converge	Loss (early stopping= False)	CV Score (early stopping= True)	Number of Iterations to Converge	Loss (early stopping= True)
20	**0.70468**	**12**	**17.54644**	**0.64872**	**24**	**0.76696**
60	0.73656	44	0.58837	0.75539	24	1.24423
200	0.65912	25	1.44914	0.76205	12	2.84747
620	0.66555	19	3.65116	0.76739	12	4.85375
2000	**0.60184**	**40**	**2.27836**	**0.76205**	**12**	**4.94309**

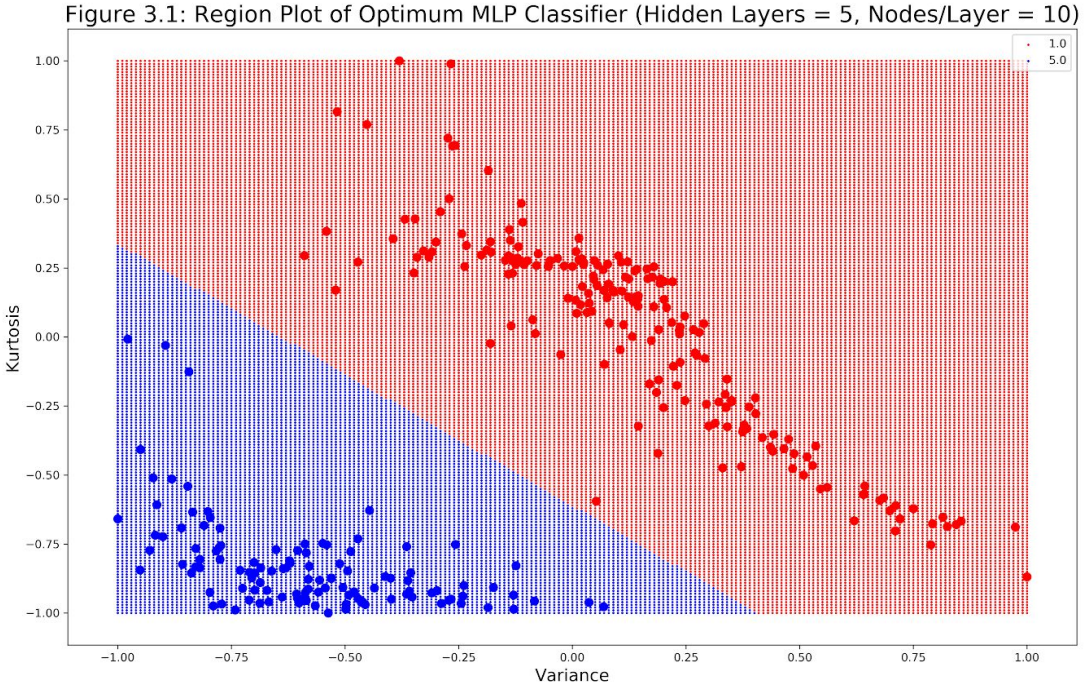
For small number of nodes, early stopping results in lesser cross validation score. However, for large number of nodes, early stopping results in better cross validation score.

Why?

In case of large network, when the model goes through more and more iterations on a limited dataset, it performs increasingly well on the training data. This increases the variance of the model and it performs poorly on the test data. Since in early stopping we test the training weights on the held-out validation set, we can stop when the validation accuracy stays the same or starts to decrease. This is how early stopping helps in preventing overfitting.

Draw the 2D region for your optimal neural network. Label this Figure 3.1

I have chosen the 2D features from HW1 and HW2 to plot this graph. The chosen neural network model has **number of layers = 5** and **number of neurons = 10**.



Extra Credit

Experiment with neural networks and bagging. Since bagging is a variance reduction strategy, you should make sure to compile all of the variances when you collect your cross-validation data

The following remain same for every experiments in this section:

Assumption:

1. We are dealing with a constant amount of data for models of every complexity.
2. The random_state is the same for every trial.

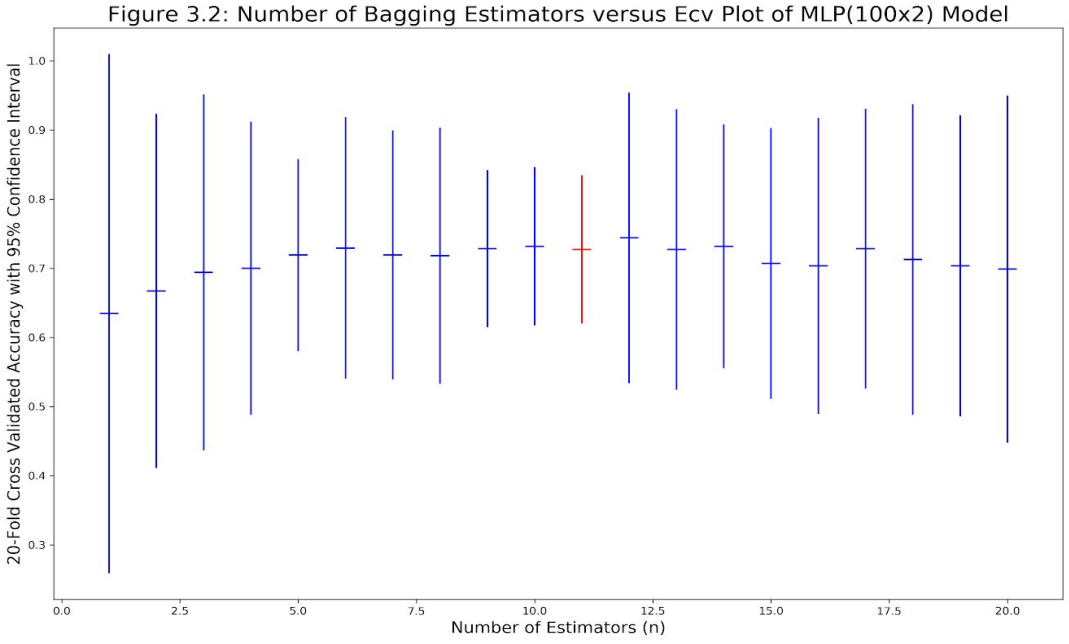
Dataset Used: UCI Blood Transfusion Dataset, but with only 200 data points to reduce the running time of the experiments.

CV Scoring: The optimality is checked against the 20-Fold Cross Validated score in 95% confidence interval.

Experiment 1: Finding the optimal number of bagging estimators.
Model Used: BaggingClassifier with number of estimators varied from 1 to 20 on the base estimator.
Base Estimator: MLPClassifier.
Parameters:

- hidden_layer_sizes=tuple([100]*2),
- activation='relu',
- solver='adam',
- epsilon=0.001,
- max_iter=10000,
- learning_rate_init=0.01, (To reduce the running time)
- alpha=0

Please find the results of the experiment below.



Does the optimum number of bagged neural networks change with the complexity of those networks?

Experiment 2: Finding the optimal number of bagging estimators for different complexities.

Model Used: BaggingClassifier with number of estimators varied from 1 to 20 on the base estimator.

Base Estimator: MLPClassifier.

Parameters:

- hidden_layers=varying in [2,5]
- nodes_per_layer=varying in [2,5,10,100]
- activation='relu',
- solver='adam',
- epsilon=0.001,
- max_iter=10000,
- learning_rate_init=0.01, (To reduce the running time)
- alpha=0

Please find the results of the experiment below:

Number Of Hidden Layers	Nodes Per Hidden Layer	Optimal Number of Bagging Estimators
2	2	1
2	5	20
2	10	3
2	100	11
5	2	1
5	5	17
5	10	18
5	100	9

Yes, they do. The optimum number of bagged neural networks makes sense only when the model has a certain level of complexity. The optimum number of bagged neural networks changes with respect to the complexity of the model.
Why? - For models with high number of internal nodes, the variance of the model is high. (As shown in the table in page 3). This causes a higher number of bagged estimators to achieve an optimal CV score for the models.

Do the more complex models provide better solutions when applied in a bagging ensemble model?

YES. Why? - As already mentioned, for highly complex models, the variance in the accuracy of the validation set is high. Since bagging ensemble model is a variance minimization technique, the results of using it can be more clearly observed in a highly complex model. The bagging ensemble technique does not do much to simple models. Thus, it is safe to use highly complex models with bagging ensemble techniques for more generalization and reduced variance.

What about neural networks makes them react well to bagging?

1. Bagging performs well on unstable learning algorithms such as neural networks, decision tree and regression tree (from lecture notes). These algorithms result in a big change in predictions when there is even a small change in the training set.
2. In neural networks, each training results in different set of weights due to the nature of the underlying solvers. This means that each training could possibly result in different local minima (if learning rate isn't sufficiently low) or somewhere around the global minima (when the learning rate is sufficiently low). Because of this, when we train the model (particularly the ones with higher complexity) using the bagging ensemble technique by bootstrapping the training data, we see that the model responds quite well

Would you choose a bagging model as your "optimum" neural network model?

It depends on the data that we are dealing with. For this dataset, the answer is NO because single MLP classifier does not perform very differently than the ensemble model (Both have CV scores of ~75% but the bagging model is costlier to compute). However, for a larger dataset, for a dataset with a high input space, for a noisy dataset or for a high complexity model using a bagging technique could result in an optimum model.