cs577 Assignment 3

Sahil Riyaz Sheikh(A20518693)

Department of Computer Science
Illinois Institute of Technology

October 25, 2022

Abstract

In this assignment I'll select a multi-class classification and a regression dataset from UCI repository. I will be making a fully connected neural network. I will then use hyper-parameter search to optimize the hyper parameters. I'll train the network for various loss and activation function. I'll compute the graph for training loss and validation loss, training accuracy and validation accuracy. In the end I'll summarize my results with observed performance results.

1. Problem Statement:

I had shortlisted two datasets to from UCI repository. Abalone Classification and Computer Hardware dataset. The Abalone dataset, received on December 1995, consist of 8 attributes and 4177 instances with no missing values and the target consisting of 29 classes whereas the Computer Hardware dataset also known as Relative CPU Performance Data, which was published in October, 1987 consisting of only 10 attributes and 209 instances. The goal this dataset is to predict the performance of CPU depending upon its relative attributes and the goal for Abalone dataset is to classify the abalone to its respective class depending upon its attributes.

2. Proposed Solution:

I will be creating a simple model for the computer performance dataset as it only consists of 209 instances and 10 attributes. I'll try to make a network which is not too deep. The output layer will have a 'linear' activation function as we need to predicted values which are not approximated for example how sigmoid or Softmax estimates the values. The other layers will have Relu as activation function. I believe MSE and uber loss will be most suitable this regression network.

For the Abalone dataset, I'll be using Softmax as the activation function in the output layer and since there are more than two classes in target, I need to one-hot-encode the target values. The other dense layers could have either sigmoid or Relu as activation function depending upon the performance of the network.

The output layer will have Softmax as activation function and the hidden layers will consist of sigmoid activation. I will use Categorical Cross Entropy loss and Kullback–Leibler divergence for loss function .

3. Implementation Detail:

The first major issue I faced was that implementing "hyperas" in my code as in the notes I saw hyperas and "hyperopt" libraries were directly installed in the Jupiter Notebook, I tried to recreate this but this ended up with unable to run my code. The optim.minimize() part of executed but the model was training. I tired all possible ways to I could find on the internet but none of them were able to solve the issue. Finally, I contacted my TA, Lawrence Amadi, during his office hours. He helped me and solved the issue I was facing; it was related to improper installation of the libraries and me having python 3.9 which is a little unstable. To make use of hyperas and optim.minimize(), I made a two separate functions "def load_cl_dataset (): and def load_rg_dataset" which does the following:

1. Load the respective datasets from repositories using pd.read_csv("hyperlink-to-csv_file")
2. There was no need to clean the dataset as it no missing values and it was already in row and column so there was no need to vectorized it again.
3. I dropped the columns which were unpredictive, for abalone I dropped the first column "M/F" and for computer hardware dataset I dropped the first two columns "vendor name and model name"
4. Separating the data(features) and target from the respective datasets using then dataset.iloc .

5. I normalized the data by calculating the mean of features by using data.mean (data,axis=0) function and subtracted the features by the so the mean of data is 0 and then calculated standard deviation of data and divided it by data so the standard deviation of the dataset is 1.
6. I one hot encoded the targets from abalone dataset by using to_categorical (). After encoding the targets, the dimension of the target was transformed to 30, so now I must have 30 nodes in the output layer for this dataset.
7. I split this normalized data into training set, testing set, training targets and testing targets. This was done by utilizing the function "train_test_split" with parameters test_size =0.1 and random_state=69 for Abalone dataset and for other dataset the split was making use of test_size =0.08.
8. Then I split this training data into training and validation using the same function as specified above.
9. The respective load function will return the training set, training targets, validation set, validation targets, test set and test targets.
10. Finally these load functions will be called in optim.minimize() .
11. I'll create the corresponding neural networks inside a function. In total I created four functions, consisting of two versions of two models. They are as follows:
    a. Regression model with MSE loss
    b. Regression model with Huber loss
    c. Classification model with Cross Entropy loss
    d. Classification model with Kullback–Leibler divergence loss
12. The model for both Classifications will same, it will just have different loss functions
13. Similarly I followed for the Regression models, they just have different loss functions.
14. Now I make use of optim.minimize() to optimize the hyper-parameters for all four of my models.
15. After getting the best_model for every model, I evaluate the these models on respective test sets to produce the results.
16. Now I saved the weights for these models using model.sav_weights("filename.h5")
17. Now for final performance the I used the best_models and fitted them with the entire training data and evaluated their performance on test set.
18. Now I created a four new models with same architecture of pervious but only consisting of one change. I incorporated regularization I these models and overfitted them to analysis their performance.

4. Results:

For the Computer Hardware dataset these are the previous results of the evaluations performed on this dataset:

Past Usage:

1. Ein-Dor and Feldmesser (CACM 4/87, pp 308-317)
   -- Results:
      -- linear regression prediction of relative cpu performance
      -- Recorded 34% average deviation from actual values
2. Kibler,D. & Aha,D. (1988). Instance-Based Prediction of Real-Valued Attributes. In Proceedings of the CSCSI (Canadian AI) Conference.
   -- Results:
      -- instance-based prediction of relative cpu performance
      -- similar results; no transformations required

- Predicted attribute: cpu relative performance (numeric)

For the Abalone dataset, these are the previous usage performance evaluations:

Sam Waugh (1995) "Extending and benchmarking Cascade-Correlation", PhD thesis, Computer Science Department, University of Tasmania.

-- Test set performance (final 1044 examples, first 3133 used for training):
    24.86% Cascade-Correlation (no hidden nodes)
    26.25% Cascade-Correlation (5 hidden nodes)
    21.5% C4.5
    0.0% Linear Discriminate Analysis
    3.57% k=5 Nearest Neighbour
  (Problem encoded as a classification task)

-- Data set samples are highly overlapped. Further information is required to separate completely using affine combinations. Other restrictions to data set examined.

David Clark, Zoltan Schreter, Anthony Adams "A Quantitative Comparison of Dystal and Backpropagation", submitted to the Australian Conference on Neural Networks (ACNN'96). Data set treated as a 3-category classification problem (grouping ring classes 1-8, 9 and 10, and 11 on).

-- Test set performance (3133 training, 1044 testing as above):
- 64% Backprop
- 55% Dystal

-- Previous work (Waugh, 1995) on same data set:
- 61.40% Cascade-Correlation (no hidden nodes)
- 65.61% Cascade-Correlation (5 hidden nodes)
- 59.2% C4.5
- 32.57% Linear Discriminate Analysis
- 62.46% k=5 Nearest Neighbour

Here are the results for the Parameter search for :

1. Regression model with MSE loss

```
def get_space():
    return {
        'optimizer': hp.choice('optimizer', ['adam','sgd']),
        'epochs': hp.choice('epochs', [5,10]),
        'batch_size': hp.choice('batch_size', [5,10,15]),
    }
```

```
22:
23:     return {'loss': validation_loss, 'status': STATUS_OK, 'model': model}
```

```
8]: 1 print("Evalutation of best performing model:")
    2 print(best_model.evaluate(test_d2,tar_d2))
    3 print("Best performing model chosen hyper-parameters:")
    4 print(best_run)
```

```
Evalutation of best performing model:
1/1 [==============================] - 0s 16ms/step - loss: 1341.4757 - mse: 1341.4757
[1341.4757080078125, 1341.4757080078125]
Best performing model chosen hyper-parameters:
{'batch_size': 0, 'epochs': 1, 'optimizer': 0}
```

2. Regression model with Huber Loss

```
def get_space():
    return {
        'optimizer': hp.choice('optimizer', ['adam','sgd','rmsprop']),
        'epochs': hp.choice('epochs', [5,6,7,8,10]),
        'batch_size': hp.choice('batch_size', [10,12,14,15]),
    }
```

```
11: std_d2 = data_d2.std()
12: data_d2 /= std_d2
13: # splitiing dataset1 into train, validation and test: DATASET2
```

```
[33]: 1 print("Evalutation of best performing model:")
      2 print(best_model1.evaluate(test_d2,tar_d2))
      3 print("Best performing model chosen hyper-parameters:")
      4 print(best_run)
```

```
Evalutation of best performing model:
1/1 [==============================] - 0s 16ms/step - loss: 19.2698 - mse: 672.6489
[19.269817352294922, 672.64892578125]
Best performing model chosen hyper-parameters:
{'batch_size': 0, 'epochs': 1, 'optimizer': 0}
```

3. Classification model with Cross Entropy loss

```
def get_space():
    return {
        'optimizer': hp.choice('optimizer', ['adam','sgd']),
        'epochs': hp.choice('epochs', [10,15,20]),
        'batch_size': hp.choice('batch_size', [10,20,30]),
    }
```

```
>>> Data
  1:
  2: dataset 1 = pd.read csv("https://archive.ics.uci.edu/ml/machine-learni
```

```
In [14]: 1 print("Evalutation of best performing model:")
         2 print(best_model2.evaluate(test_d1,tar_d1))
         3 print("Best performing model chosen hyper-parameters:")
         4 print(best_run2)
```

```
Evalutation of best performing model:
14/14 [==============================] - 0s 2ms/step - loss: 2.0586 - accuracy: 0.2679
[2.0585880279541016, 0.2679425776004791]
Best performing model chosen hyper-parameters:
{'batch_size': 1, 'epochs': 2, 'optimizer': 0}
```

4. Classification model with KL- loss

```
def get_space():
    return {
        'optimizer': hp.choice('optimizer', ['adam','sgd']),
        'epochs': hp.choice('epochs', [10,15,20]),
        'batch_size': hp.choice('batch_size', [10,20,30]),
    }

>>> Data
```

[17]:
```
print("Evalutation of best performing model:")
print(best_model3.evaluate(test_d1,tar_d1))
print("Best performing model chosen hyper-parameters:")
print(best_run3)
```

```
Evalutation of best performing model:
14/14 [==============================] - 0s 3ms/step - loss: 2.0309 - accuracy: 0.2847
[2.0309011936187744, 0.2846890091896057]
Best performing model chosen hyper-parameters:
{'batch_size': 1, 'epochs': 2, 'optimizer': 0}
```

Here is the final performance evaluation with fitting the respective best_models with entire training data(training + validation) and evaluating them on the test set. I created a separate function to return just training data, training target, test set and test target for both dataset to evaluate their performance.

Final performances of the above models, fitting with entire train dataset and evaluating with test dataset

[23]:
```
best_model.fit(X_d2,Y_d2,verbose=0)
print(best_model.evaluate(test_d2,tar_d2))
best_model1.fit(X_d2,Y_d2,verbose=0)
print(best_model1.evaluate(test_d2,tar_d2))
best_model2.fit(X_d1,Y_d1,verbose=0)
print(best_model2.evaluate(test_d1,tar_d1))
best_model3.fit(X_d1,Y_d1,verbose=0)
print(best_model3.evaluate(test_d1,tar_d1))
```

```
1/1 [==============================] - 0s 23ms/step - loss: 1084.0194 - mse: 1084.0194
[1084.0194091796875, 1084.0194091796875]
1/1 [==============================] - 0s 21ms/step - loss: 31.3728 - mse: 5497.9072
[31.372831344604492, 5497.9072265625]
14/14 [==============================] - 0s 2ms/step - loss: 2.0189 - accuracy: 0.2512
[2.01889705657959, 0.2511961758136749]
14/14 [==============================] - 0s 3ms/step - loss: 2.0128 - accuracy: 0.2703
[2.012828826904297, 0.270334929227829]
```

Where best_model: Regression model with MSE loss
best_model1: Regression model with Huber Loss
best_model2: Classification model with Cross Entropy loss
best_model3: Classification model with KL- divergence loss

```
def train_test_d1():
    dataset_1 = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data", header=None)
    # For dataset 1 I will drop the first column, the M/F as we don't need it
    data_1 = dataset_1
    data1 = data_1.drop(data_1.columns[[0]],axis=1)
    label_d1 = data1.iloc[:,7:]
    data_d1 = data1.iloc[:,:7]
    #normalize if by subtracting mean and dividing by standard devidation
    mean_d1 = data_d1.mean(axis=0)
    data_d1 -= mean_d1
    std_d1 = data_d1.std()
    data_d1 /= std_d1
    # one hot encoding
    label_d1 = to_categorical(label_d1)
    # splitiing dataset1 into train and test: DATASET1
    X_d1,test_d1,Y_d1,tar_d1 = train_test_split(data_d1,label_d1,test_size=0.1, random_state=69)
    return X_d1,test_d1,Y_d1,tar_d1
```

```
X_d1,test_d1,Y_d1,tar_d1 = train_test_d1()
```

```
def train_test_d2():
    dataset_2 = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/cpu-performance/machine.data", header
    # For dataset 2 I will drop the first two columns as they are non predictive
    data_2 = dataset_2
    data2 = data_2.drop(data_2.columns[[0,1]],axis=1)
    data_d2 = data2.iloc[:,:7]
    label_d2 = data2.iloc[:,7:]
    #for 2nd dataset
    mean_d2 = data_d2.mean(axis=0)
    data_d2 -= mean_d2
    std_d2 = data_d2.std()
    data_d2 /= std_d2
    # splitiing dataset1 into train and test: DATASET2
    X_d2,test_d2,Y_d2,tar_d2 = train_test_split(data_d2,label_d2,test_size=0.08)
    return X_d2,test_d2,Y_d2,tar_d2
```

Here are the results for overfitted models with regularization:

Model 1

```
Epoch 49/50
20/20 - 0s - loss: 1254.1016 - mse: 1253.6470 - 59ms/epoch - 3ms/step
Epoch 50/50
20/20 - 0s - loss: 880.1080 - mse: 879.6502 - 35ms/epoch - 2ms/step
1/1 [==============================] - 0s 109ms/step - loss: 858.8485 - mse: 858.3898
[858.8485107421875, 858.3898315429688]
```

Model 2

```
Epoch 50/50
14/14 - 0s - loss: 21.5169 - mse: 2391.9236 - 43ms/epoch - 3ms/step
1/1 [==============================] - 0s 109ms/step - loss: 11.3093 - mse: 399.3094
[11.309321403503418, 399.3093566894531]
```

Out[271]: a

Model 3

```
Epoch 50/50
376/376 - 1s - loss: 1.9755 - accuracy: 0.2745 - val_loss: 1.8920 - val_accuracy: 0.3285 - 806ms/epoch - 2ms/step
14/14 [==============================] - 0s 1ms/step - loss: 2.0840 - accuracy: 0.2608
[2.084015130996704, 0.26076555252075195]
```

Model 4

```
Epoch 50/50
301/301 - 1s - loss: 1.9911 - accuracy: 0.2780 - val_loss: 1.9604 - val_accuracy: 0.3019 - 718ms/epoch - 2ms/step
14/14 [==============================] - 0s 1ms/step - loss: 2.0799 - accuracy: 0.2679
[2.079861640930176, 0.2679425776004791]
```

For this I can summarize the following results:

1. Huber Loss performs better than MSE as Huber loss I less sensitive to outlines and can act as a combination of both MSE and MAE.
2. KL – divergence performs slightly better than Categorical Cross entropy. KL divergence can be defined as relative entropy/ difference between cross entropy and entropy.

5. References:

Lecture notes of Lesson 12 :

https://drive.google.com/file/d/1ao3Bd6n2GfGAJnO7v8ET60hTVtYciosp/view?usp=sharing

https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/

https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

(the below one to solve the issue related to hyperas)

https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f