CO395 Coursework 2 Report

Authors: Peizhen Wu, Jiajun Chen, Kai Zhang, Zhen Hao

- The output of the model including loss values, evaluations, illustrations can be found in CO395_CW2.ipynb.

Q2.1: Implement the forward model of the ABB IRB 120

First, we load the dataset, preprocess it, and split the data into train sets and validation sets. (80% train sets vs. 20% validation set).

Then we begin to build the architecture. The neural network includes 2 layers, which has 16, 3 neurons in each layer. The first layer uses ReLU as the activation function, and the last layer uses identity function as its activation function to get its regression result.

To avoid underfitting and overfitting problems, we use SGD to nudge the weights of the network. By setting batch size to 8, epoch times to 100 and learning rate to 0.01, the network gradually changes the weights in an ideal rate and finally reaches the convergence.

The performance of the neural network can be assessed by its loss value using mean squared error and the mean absolute error. By observing the output, we can see the loss value is decreasing every 10 epoches mostly. After 100 epoches, both the final train loss and validation loss are low, which indicates that the model does not underfit nor overfit. The mean absolute error score 0.0488 and R2 score 0.90 also indicates the predictions on the initial model are surprisingly well. The hyperparameter search stage will be illustrated later and the fine-tuned model will be shown later.

```
epoch 0 loss is 0.018464615320078336
epoch 10 loss is 0.02515483445036398
epoch 20 loss is 0.01328622468722141
epoch 30 loss is 0.019157347856253518
epoch 40 loss is 0.005662229602430007
epoch 50 loss is 0.0018869746255628331
epoch 60 loss is 0.007393693846548497
epoch 70 loss is 0.006545949221863557
epoch 80 loss is 0.00336025598912866
epoch 90 loss is 0.0049650753769076295
Train loss =  0.004157004873717038
Validation loss =  0.004279450929647508
Validation Score MAE: 0.04883736492760886
Validation Score R2: 0.9004138694816639
```

Q2.2: Evaluate your architecture

In order to have the opportunity to train the model on different datasets, we used k-fold cross-validation to evaluate the performance of the model. Since the dataset is large enough, it is acceptable to use the holdout method if time is not enough.

In each fold, the data set is divided into train set and test set, the ratio is 7: 2.5. The model trains on the train set. The generalization ability is examined on the test set.

Since the task is a regression, MSE or MAE can be used as a performance indicator, and the two are very similar, we chose MAE as the indicator. In addition, R^2 score is computed for each fold to be a more intuitive metric.

Since the neural network is a stochastic model, multiple times of evaluation were executed to know the distribution of the model's performance.

We used sklearn metrics module. Scores of all r2 scores are averaged, weighted by the variance of each individual output, which is 0.9997. All mean absolute errors are averaged with uniform weight, which is 0.002501. The evaluate_architecture function accepts the hyperparameters of the neural network and the dataset for the evaluate as arguments and outputs the performance of the model on each fold.

The code for evaluate architecture can be found on CO395-CW2.ipynb notebook

Q2.3: Fine tune your architecture

There are two sets of hyper-parameters, hyper-parameters that define the architecture such as number of layers, number of neurons in each layer and each layer 's activation function, hyper-parameters that control the training process such as learning rate and batch size. We want to search the best performing hyper-parameters for both sets.

We use a package called hyperopt which gives functionalities to suggest and choose the best hyper-parameters.

First, we specify a searching space where all the hyper-parameters can take their value. Number of hidden layers can be {1,2 or 3}. Number of neurons in each layer can be {16,32,64,128,256 or 512}. Neurons in each layer can have an activation function of either relu or sigmoid.Learning rate is uniformly selected from [1e-6, 1e-2]. Batch size is selected from {8,16,32} per batch.

We define a function which will return the loss after 50 epochs of training with a combination of hyper-parameters randomly selected from the space. Iterating this process 100 times and return the best performing (with the smallest loss)set of hyper-parameters.

The best performing hyper-parameters are three hidden layers with 512,128, 32 neurons in each layer, all layers using relu activation, batch size is 8 and learning rate is 0.01. The result of the fine-tuned model is shown below:

```
Train loss =  3.315632346107977e-05
Validation loss =  3.618522521702499e-05
Validation Score MAE: 0.0042840834555129045
Validation Score R2: 0.9991608023355171
```

The MAE score reduce to 0.00428 and R2 score increase to 0.999, indicating that the fine-tuned model has pretty well performance.

The code for hyperparameters search can be found on CO395-CW2.ipynb notebook

Q3.1: Implement the ROI detector

As Q2.1, we build the architecture in a similar way.

First, we load the dataset, preprocess it, and split the data into train sets and validation sets. (80% train sets vs. 20% validation set).

Then we begin to build the architecture. The neural network includes 2 layers, which has 16, 4 neurons in each layer. The first layer uses ReLU as the activation function, and the last layer uses identity function as its activation function in order to get its result.

To avoid underfitting and overfitting problems, we use SGD to nudge the weights of the network. By setting batch size to 8, epoch times to 100 and learning rate to 0.01, the network gradually changes the weights in an ideal rate and finally reaches the convergence.

The performance of the neural network can be assessed by its loss value using cross entropy and the accuracy. By observing the output, we can see the loss value is decreasing at most time. After 100 epoches, both the final train loss and validation loss are low, which indicates that the model does not underfit nor overfit. The accuracy, 95.1% also indicates the predictions on the initial model are quite accurate. We follow the same procedure as in part 2, the hyperparameter search and fine-tuned model will be discussed later.

```
epoch 70 loss is 0.055356117754015596
epoch 80 loss is 0.068438362710265
epoch 90 loss is 0.003624277871447856
Train loss =  0.7996862365283309
Validation loss =  0.8055854138227556
Validation accuracy: 0.95136
```

Q3.2: Evaluate your architecture

Like Q2.2, we used k-fold cross-validation to measure the performance of the classifier. We used four indicators to measure model performance, accuracy, unweighted average of recall, precision and f1 score, and output the confusion matrix in each fold to visualise the performance and allow easy identification of confusion. Finally returns the average performance of all folds. Unweighted score of recall and precision for each class are computed to avoid misleading results. F1 score is to have a simple measure of the performance of the classifier, the average of 10 folds is 0.9648.

Since the neural network is a stochastic model, multiple times of evaluation were executed to know the distribution of the model's performance.

The evaluate_architecture function accepts the hyperparameters of the neural network and the dataset for the evaluate as arguments and outputs the performance of the model on each fold. The code for evaluate architecture can be found on CO395-CW2.ipynb notebook

Q3.3: Fine tune your architecture

The strategy used is similar to q2.3. We first specify a space of hyper-parameters, randomly choose a combination of hyperparameters, train the network and evaluate the loss. Choose the best hyperparameters within 100 trials.

This process gives us the fine-tuned hyperparameters for the classification problem. Our network has 4 layers each with (32,32,256,128) neurons, ReLU as activation function for every layer.learning rate is 0.073, batch size is 32.

The code for hyperparameters search can be found on CO395-CW2.ipynb notebook and the result of fine-tuned model is as followed:

```
Train loss =  0.7575916329874425
Validation loss =  0.7622201929650209
Validation accuracy: 0.9824
```

The fine tuned model increase the accuracy of prediction to 98.4%, which is considerably well.