Ontario Tech University

Machine Learning and Data Mining

Mini Project 4

Deadline: Sunday, April 5, 11:59 PM

Instructor: Dr. Shahryar Rahnamayan, PEng, SMIEEE

Using ANN for Data Classification

Team Members	Student Numbers
Jasindan Rasalingam	100584595
Ashwin Kamalakannan	100584423

Design

The majority of the source code for this project was taken from our previous miniproject where we also made a cnn classifier. The main difference for these tasks is how we clean the data to work with our model. For our implementation, we chose to use Keras, which is a TensorFlow based tool used to create neural networks. We chose to use python for quick development. For our neural net, we chose to use a sequential model with 3 layers. This means that each layer passes its outputs as the next layers inputs sequentially. For our layers we used dense layers, meaning that all neurons are fully connected. For our first 2 layers we used ReLU (rectified linear unit) activation function. This is the default activation function and is most commonly used. For our output function, we used softmax which is a probability distribution function. For task 2 and 3, we changed the number of layers as well as the number of neurons, tweaking it as necessary to generate better results. We think the second one required the most number of layers because there were many columns. When compiling the model, we have to specify the cross entropy loss function and optimizer function. The loss function that we chose in this case is commonly used for numerical data sets such as ours. The optimizer function we chose is fast and has many benefits when using large data sets.

For our data, we downloaded it from the sources provided and cleaned it to work better with our algorithms. We replaced strings with digits representing what they are and we added column headers. In some cases, we also removed rows with empty values.

In order to calculate the confusion matrix for our predictions, we used scikit learn's confusion matrix module. We pass our input data without the target label as 1 tf array and the predictions as another array. The confusion matrix is generated based on the error vs correct classifications made by our model. In our case, we saw that our numbers were slightly off. This is likely due to us incorrectly inputting values to the confusion matrix and we weren't sure how to fix this.

```
# Create keras model
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(3, activation='softmax')
])
```

```
# Create keras model
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='softmax')
])
```

```
# Create keras model
model = tf.keras.Sequential([
    feature_layer,
    layers.Dense(64, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax')
])
```

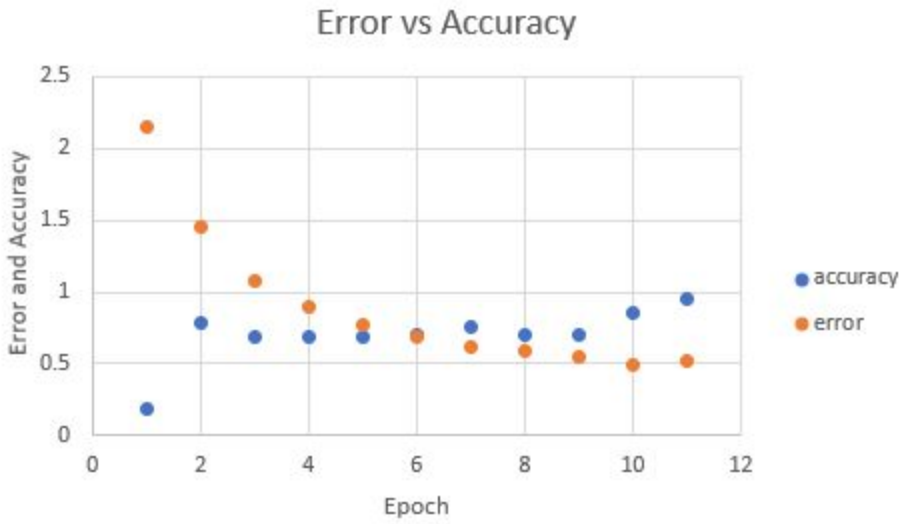
```
# Compile keras Model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'],
)

# Train the model
model.fit(train_ds, epochs=15)
```

```
# get confusion matrix
prediction = np.argmax(model.predict(df_to_dataset(dataframe, batch_size=batch_size)), axis=1)
matrix = confusion_matrix(dataframe['class'], prediction)
print(matrix)
```

Test Set 1: Iris Data Set

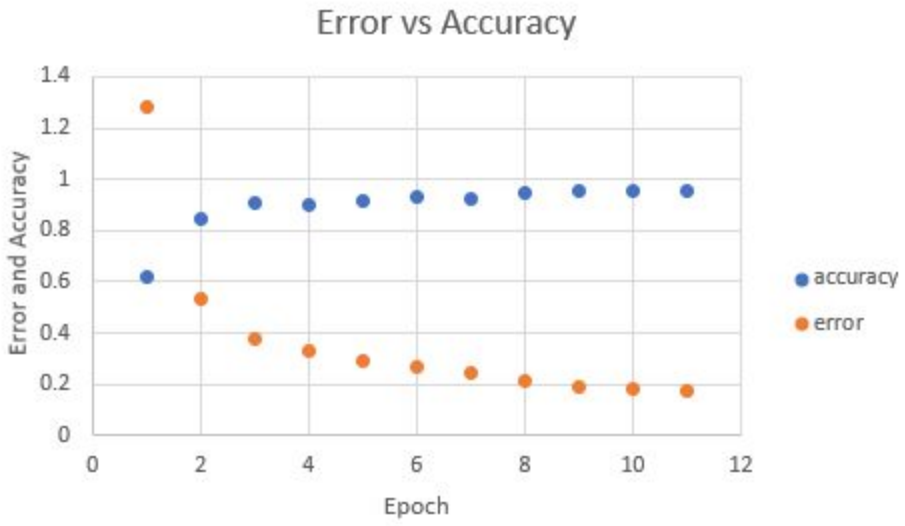
```
Epoch 1/10
4/4 [==============================] - 0s 17ms/step - loss: 2.1442 - acc: 0.1810
Epoch 2/10
4/4 [==============================] - 0s 1ms/step - loss: 1.4602 - acc: 0.7810
Epoch 3/10
4/4 [==============================] - 0s 1ms/step - loss: 1.0844 - acc: 0.6857
Epoch 4/10
4/4 [==============================] - 0s 1ms/step - loss: 0.8963 - acc: 0.6857
Epoch 5/10
4/4 [==============================] - 0s 907us/step - loss: 0.7751 - acc: 0.6857
Epoch 6/10
4/4 [==============================] - 0s 912us/step - loss: 0.6846 - acc: 0.7048
Epoch 7/10
4/4 [==============================] - 0s 1ms/step - loss: 0.6123 - acc: 0.7619
Epoch 8/10
4/4 [==============================] - 0s 948us/step - loss: 0.5833 - acc: 0.6952
Epoch 9/10
4/4 [==============================] - 0s 953us/step - loss: 0.5461 - acc: 0.6952
Epoch 10/10
4/4 [==============================] - 0s 946us/step - loss: 0.4863 - acc: 0.8571
2/2 [==============================] - 0s 13ms/step - loss: 0.5214 - acc: 0.9556
Accuracy 0.95555556
```




```
[[13 20 17]
 [19 15 16]
 [18 12 20]]
```

Test Set 2: Breast Cancer Wisconsin Data Set

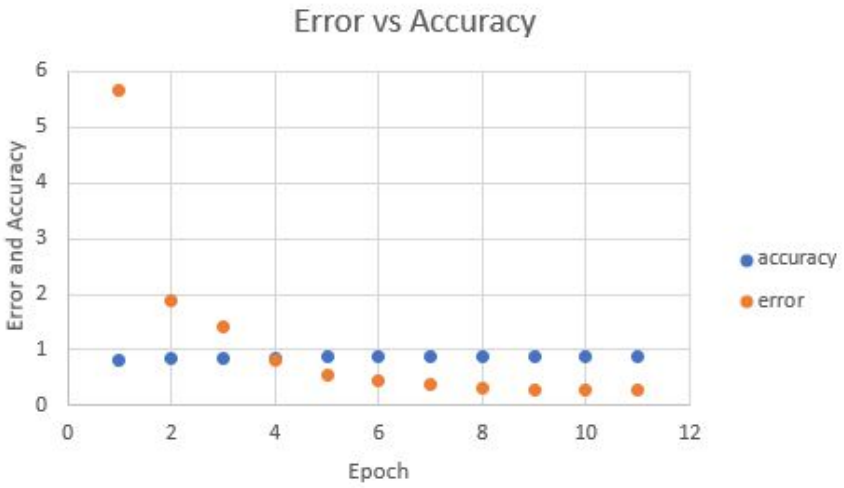
```
Epoch 1/10
15/15 [==============================] - 0s 6ms/step - loss: 1.2847 - acc: 0.6200
Epoch 2/10
15/15 [==============================] - 0s 1ms/step - loss: 0.5343 - acc: 0.8434
Epoch 3/10
15/15 [==============================] - 0s 1ms/step - loss: 0.3799 - acc: 0.9040
Epoch 4/10
15/15 [==============================] - 0s 1ms/step - loss: 0.3283 - acc: 0.8998
Epoch 5/10
15/15 [==============================] - 0s 1ms/step - loss: 0.2921 - acc: 0.9165
Epoch 6/10
15/15 [==============================] - 0s 981us/step - loss: 0.2687 - acc: 0.9332
Epoch 7/10
15/15 [==============================] - 0s 910us/step - loss: 0.2411 - acc: 0.9248
Epoch 8/10
15/15 [==============================] - 0s 899us/step - loss: 0.2099 - acc: 0.9478
Epoch 9/10
15/15 [==============================] - 0s 872us/step - loss: 0.1922 - acc: 0.9562
Epoch 10/10
15/15 [==============================] - 0s 907us/step - loss: 0.1782 - acc: 0.9582
7/7 [==============================] - 0s 6ms/step - loss: 0.1756 - acc: 0.9510
Accuracy 0.95098037
```




```
[[293 151]
 [155  84]]
```

Test Set 3: Bank Marketing Data Set

```
Epoch 1/10
989/989 [==============================] - 1s 1ms/step - loss: 5.6803 - acc: 0.8200
Epoch 2/10
989/989 [==============================] - 1s 1ms/step - loss: 1.8950 - acc: 0.8450
Epoch 3/10
989/989 [==============================] - 1s 1ms/step - loss: 1.4243 - acc: 0.8527
Epoch 4/10
989/989 [==============================] - 1s 1ms/step - loss: 0.8081 - acc: 0.8612
Epoch 5/10
989/989 [==============================] - 1s 1ms/step - loss: 0.5532 - acc: 0.8682
Epoch 6/10
989/989 [==============================] - 1s 997us/step - loss: 0.4285 - acc: 0.8733
Epoch 7/10
989/989 [==============================] - 1s 1ms/step - loss: 0.3723 - acc: 0.8802
Epoch 8/10
989/989 [==============================] - 1s 992us/step - loss: 0.3032 - acc: 0.8832
Epoch 9/10
989/989 [==============================] - 1s 993us/step - loss: 0.2819 - acc: 0.8878
Epoch 10/10
989/989 [==============================] - 1s 1ms/step - loss: 0.2772 - acc: 0.8867
424/424 [==============================] - 0s 912us/step - loss: 0.2732 - acc: 0.8907
Accuracy 0.8907321
```




```
[[  147  5142]
 [ 1178 38744]]
```

Conclusion Remarks

In this mini project, we learned how to adapt different types of datasets to cnn models. The main challenge we faced was ensuring that our data could be read consistently. For example, we had to convert strings into text, add headers and remove rows with empty values. We did this because we weren't sure how to use tensorflow to map our data for us, so instead we pre cleaned the data. We also tweaked the cnns we developed in the last miniproject to work better for each test case in this project. For example, the bank problem had a higher complexity in inputs so we used more neurons and the breast cancer diagnostic data had many inputs so we used another layer than usual. The other challenge was figuring out how to apply the confusion matrix for our implemented cnn. Overall, we were able to successfully get over 89% accuracy in all of our test cases for the 3 different datasets.