Team9_Report

December 16, 2021

1 Team 9 Project Report: Eri Kim, Arman Saduakas, Hoang Phan Pham

1.1 Description of Algorithm:

Our team created a classification neural network model that consists of multiple layers using different activation functions to predict a digit that is written in the image.
The activation functions used are as follows:
1. Relu: Rectified linear activation function is a linear function that outputs the input directly if it is positive, otherwise, zero. It has become the default activation function for many types of neural networks because it is easier to train and achieves better performance. One of the advantages is that its gradient is always equal to 1, which enables the model to pass the maximum amount of the error through the network during back-propagation. The relu functions are usually used as input and hidden layers because they describe variation of data since the output can be any number between 0 and positive infinity.

2. Sigmoid: Sigmoid function is one of the most widely used non-linear activation function. It transforms values between 0 and 1. It is mostly used for models when predicting probability is needed. Since probability can be only between 0 and 1, sigmoid is used in that case.

3. Softmax: Softmax function is used as an activation function in the output layer for a model that predicts a multi-class classification problems where class membership is required on more than two class labels. The network is configured to output N values, and the softmax function is used to normalize the outputs, converting each of them from weighted sum values into probabilities that sum to one. Each output of the softmax activation function can be interpreted as the probability of membership for each class label.

1.1.1 Three Different Approaches to Optimize the Model

We tried to use other activation functions such as tanh; however, we decided to use sigmoid instead since our output of the model would only require 0 to 1 instead -1 to 1. We also tried adjusting the epoch size. We increased the size to optimize the model, however, increasing the epoch size started over fitting after size = 35, which resulted in decreasing the test performance. Lastly, we tried different numbers of neurons in each layer. We attempted to change the number of neurons in each layer to optimize the test performance, and the numbers we ended up with provided us the best performance, which gave us 96.63% accuracy on the test data. We also tried many different combinations of the ReLu function and various kernel initializers and found out that Random Uniform worked the best. In addition, Glorot kernel initializer performed the best with the Sigmoid activation function.

1.2 Imports

[ ]:
```python
import pandas as pd
data_train = pd.read_csv('./train.csv')
```

2 Data description

The dataset was obtained from Kaggle:
https://www.kaggle.com/animatronbot/mnist-digit-recognizer
The data has 42000 rows, each row represents 1 column for label and 784 columns for pixel of each image.

[ ]:
```python
print(data_train.shape)
data_train.head()
```

```
(42000, 785)
```

[ ]:
```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[ ]:
```python
import tensorflow as tf

images = data_train.drop(columns=['label'])
labels = data_train.label
labels = tf.keras.utils.to_categorical(labels)
print(images.shape)
print(labels.shape)
```

```
(42000, 784)
(42000, 10)
```

We split the data into training, validation, and testing.
Training: 60%
Validation: 15%
Test: 25%

[ ]:
```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    images, labels, test_size=0.40, random_state=101, stratify=labels
)
x_test, x_validation, y_test, y_validation = train_test_split(
    x_test, y_test, test_size=0.375, random_state=202, stratify=y_test
)
print('Training has', y_train.shape[0], 'digits')
print('Validation has', y_validation.shape[0], 'digits')
print('Testing has', y_test.shape[0], 'digits')
```

```
Training has 25200 digits
Validation has 6300 digits
Testing has 10500 digits
```

[ ]:
```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(750, input_shape=(28 * 28,),␣
 ↪kernel_initializer="random_uniform", activation="relu"))
model.add(Dense(500, kernel_initializer="random_uniform", activation="relu"))
model.add(Dense(250, kernel_initializer="random_uniform", activation="relu"))
model.add(Dense(50, kernel_initializer="glorot_uniform", activation="sigmoid"))
model.add(Dense(10, kernel_initializer="he_normal", activation="softmax"))

model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[ ]:
```python
model_training = model.fit(
    x_train,
    y_train,
    validation_data=(x_validation, y_validation),
    epochs=35,
    batch_size=128,
)
```

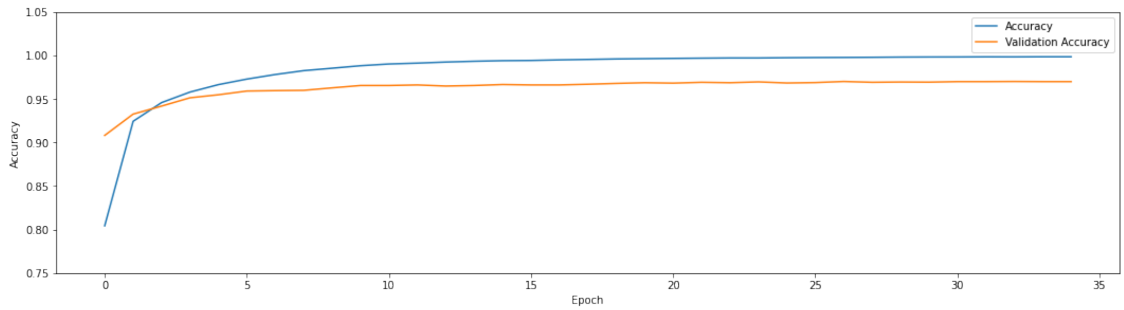
```
Epoch 1/35
197/197 [==============================] - 2s 7ms/step - loss: 0.9037 -
```

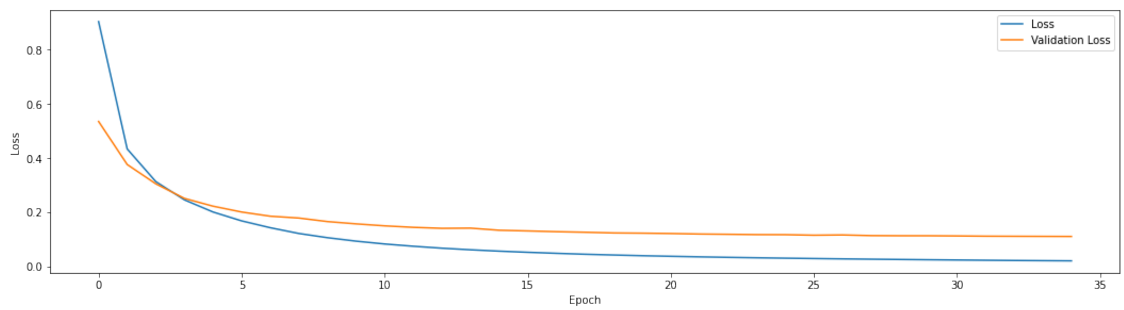
```
accuracy: 0.8043 - val_loss: 0.5350 - val_accuracy: 0.9081
Epoch 2/35
197/197 [==============================] - 1s 6ms/step - loss: 0.4335 -
accuracy: 0.9243 - val_loss: 0.3761 - val_accuracy: 0.9325
Epoch 3/35
197/197 [==============================] - 1s 7ms/step - loss: 0.3130 -
accuracy: 0.9458 - val_loss: 0.3048 - val_accuracy: 0.9417
Epoch 4/35
197/197 [==============================] - 1s 6ms/step - loss: 0.2456 -
accuracy: 0.9579 - val_loss: 0.2510 - val_accuracy: 0.9513
Epoch 5/35
197/197 [==============================] - 1s 6ms/step - loss: 0.2007 -
accuracy: 0.9664 - val_loss: 0.2222 - val_accuracy: 0.9548
Epoch 6/35
197/197 [==============================] - 1s 7ms/step - loss: 0.1680 -
accuracy: 0.9727 - val_loss: 0.2006 - val_accuracy: 0.9590
Epoch 7/35
197/197 [==============================] - 1s 7ms/step - loss: 0.1426 -
accuracy: 0.9780 - val_loss: 0.1852 - val_accuracy: 0.9595
Epoch 8/35
197/197 [==============================] - 1s 7ms/step - loss: 0.1215 -
accuracy: 0.9825 - val_loss: 0.1785 - val_accuracy: 0.9598
Epoch 9/35
197/197 [==============================] - 1s 6ms/step - loss: 0.1059 -
accuracy: 0.9852 - val_loss: 0.1654 - val_accuracy: 0.9627
Epoch 10/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0931 -
accuracy: 0.9881 - val_loss: 0.1569 - val_accuracy: 0.9654
Epoch 11/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0826 -
accuracy: 0.9901 - val_loss: 0.1496 - val_accuracy: 0.9654
Epoch 12/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0742 -
accuracy: 0.9911 - val_loss: 0.1440 - val_accuracy: 0.9660
Epoch 13/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0671 -
accuracy: 0.9923 - val_loss: 0.1404 - val_accuracy: 0.9648
Epoch 14/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0612 -
accuracy: 0.9932 - val_loss: 0.1411 - val_accuracy: 0.9654
Epoch 15/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0562 -
accuracy: 0.9939 - val_loss: 0.1333 - val_accuracy: 0.9665
Epoch 16/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0520 -
accuracy: 0.9941 - val_loss: 0.1308 - val_accuracy: 0.9660
Epoch 17/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0482 -
```

```
accuracy: 0.9948 - val_loss: 0.1281 - val_accuracy: 0.9660
Epoch 18/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0447 -
accuracy: 0.9953 - val_loss: 0.1260 - val_accuracy: 0.9668
Epoch 19/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0419 -
accuracy: 0.9959 - val_loss: 0.1234 - val_accuracy: 0.9678
Epoch 20/35
197/197 [==============================] - 2s 8ms/step - loss: 0.0393 -
accuracy: 0.9962 - val_loss: 0.1226 - val_accuracy: 0.9686
Epoch 21/35
197/197 [==============================] - 2s 9ms/step - loss: 0.0371 -
accuracy: 0.9965 - val_loss: 0.1213 - val_accuracy: 0.9681
Epoch 22/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0351 -
accuracy: 0.9968 - val_loss: 0.1194 - val_accuracy: 0.9690
Epoch 23/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0332 -
accuracy: 0.9970 - val_loss: 0.1184 - val_accuracy: 0.9686
Epoch 24/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0317 -
accuracy: 0.9970 - val_loss: 0.1173 - val_accuracy: 0.9695
Epoch 25/35
197/197 [==============================] - 1s 7ms/step - loss: 0.0302 -
accuracy: 0.9973 - val_loss: 0.1170 - val_accuracy: 0.9683
Epoch 26/35
197/197 [==============================] - 2s 8ms/step - loss: 0.0289 -
accuracy: 0.9975 - val_loss: 0.1151 - val_accuracy: 0.9687
Epoch 27/35
197/197 [==============================] - 2s 8ms/step - loss: 0.0276 -
accuracy: 0.9976 - val_loss: 0.1161 - val_accuracy: 0.9700
Epoch 28/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0265 -
accuracy: 0.9978 - val_loss: 0.1136 - val_accuracy: 0.9690
Epoch 29/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0254 -
accuracy: 0.9980 - val_loss: 0.1131 - val_accuracy: 0.9694
Epoch 30/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0244 -
accuracy: 0.9981 - val_loss: 0.1130 - val_accuracy: 0.9692
Epoch 31/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0234 -
accuracy: 0.9982 - val_loss: 0.1124 - val_accuracy: 0.9698
Epoch 32/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0225 -
accuracy: 0.9983 - val_loss: 0.1115 - val_accuracy: 0.9698
Epoch 33/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0217 -
```

```
accuracy: 0.9983 - val_loss: 0.1111 - val_accuracy: 0.9700
Epoch 34/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0210 -
accuracy: 0.9984 - val_loss: 0.1108 - val_accuracy: 0.9698
Epoch 35/35
197/197 [==============================] - 1s 6ms/step - loss: 0.0204 -
accuracy: 0.9984 - val_loss: 0.1105 - val_accuracy: 0.9698
```

[ ]:
```python
import matplotlib.pyplot as plt

_, axis = plt.subplots(figsize=(18, 10))
axis = plt.subplot(2, 1, 1)
axis.plot(
    range(len(model_training.history.get("accuracy"))),
    model_training.history.get("accuracy"),
    label="Accuracy",
)
axis.plot(
    range(len(model_training.history.get("val_accuracy"))),
    model_training.history.get("val_accuracy"),
    label="Validation Accuracy",
)
axis = plt.gca()
axis.set_ylim([0.75, 1.05])
axis.legend(loc="best")
axis.set_xlabel("Epoch")
axis.set_ylabel("Accuracy")
plt.show()

_, axis = plt.subplots(figsize=(18, 10))
axis = plt.subplot(2, 1, 2)
axis.plot(
    range(len(model_training.history.get("loss"))),
    model_training.history.get("loss"),
    label="Loss"
)
axis.plot(
    range(len(model_training.history.get("val_loss"))),
    model_training.history.get("val_loss"),
    label="Validation Loss",
)
axis.legend(loc="best")
axis.set_xlabel("Epoch")
axis.set_ylabel("Loss")
plt.show()
```





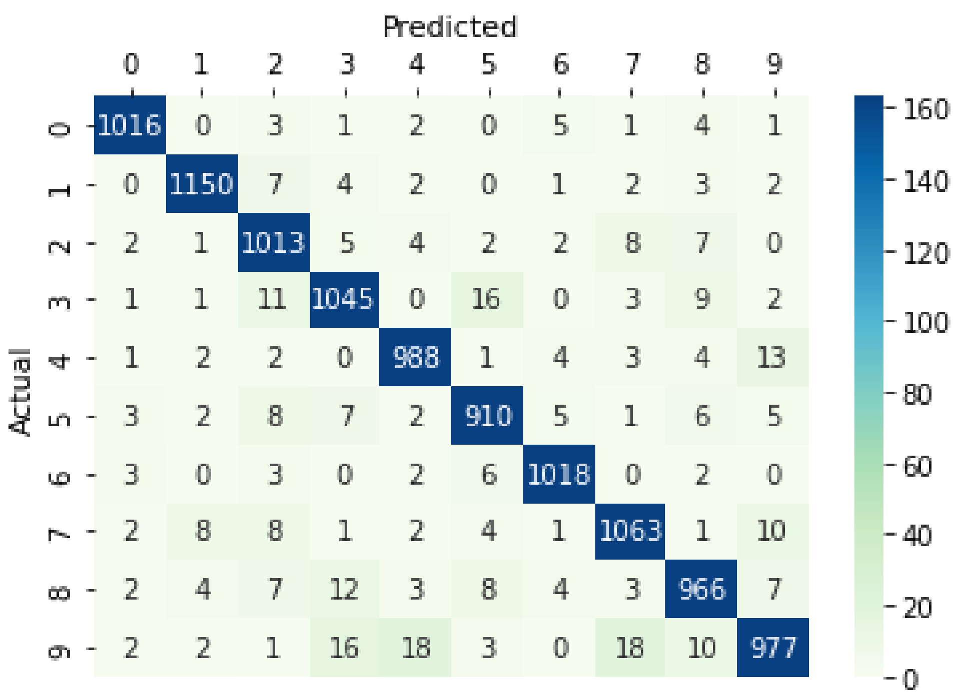
[ ]:
```
prediction = model.predict(x_test)
scores = model.evaluate(x_test, y_test, verbose=0)
accuracy = scores[1] * 100
error = 100 - scores[1] * 100
print("Accuracy: %.2f%%" % accuracy)
print("Error: %.2f%%" % error)
```

```
Accuracy: 96.63%
Error: 3.37%
```

[ ]:
```
import numpy as np

predicted, actual = [], []
for p in prediction:
    max = float(0)
    index, max_i = 0, -1
    for n in p:
        if float(n) > max:
            max = float(n)
            max_i = index
        index += 1
    predicted.append(max_i)

for digit in y_test:
```

```
    actual.append(np.argmax(digit))

y_actual = pd.Series(actual, name="Actual")
y_predict = pd.Series(predicted, name="Predicted")
```

[ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix

cf_matrix = confusion_matrix(y_actual, y_predict)
axis = sns.heatmap(cf_matrix, vmin=0, vmax=163, annot=True,fmt="d",␣
 ↪linewidths=0, cmap="GnBu")
axis.xaxis.tick_top()
axis.set_title('Predicted', size=11)
axis.set_ylabel('Actual', loc="center", size=11)
plt.show()
```



[ ]:
```
count = 0
for i in range(len(prediction)):
    if np.argmax(prediction[i]) != np.argmax(y_test[i]):
        count += 1
print("Number of wrong classified digits: " + str(count))
```

```
Number of wrong classified digits: 354
```