HAND WRITTEN DIGIT RECOGNITION

A COURSE PROJECT REPORT

By

SINDHU KALEESWARAN(RA2011026010082)

APARNA SURESH(RA2011026010099)

CHEREDDY SOWMYA SRI(RA2011026010113)

Under the guidance of

Mrs.A.Jackulin Mahariba

In partial fulfilment of the course

18CSE388T – ARTIFICIAL NEURAL NETWORKSIn CINTEL

2022-2023 ODD Semester, November 2022

Department of Computational Intelligence

School of Computing

SRMIST

Kattankulathur

Submitted to : Mrs.A.Jackulin Mahariba

BONAFIDE CERTIFICATE

Certified that this mini project report "**Hand Written Digit Recognition**" is the bonafide work of **Sindhu Kaleeswaran(RA2011026010082),Aparna Suresh(RA2011026010099) and Chereddy Sowmya Sri(RA2011026010113)** who carried out the project work under my supervision.

SIGNATURE

Mrs. A. JACKULIN MAHARIBA

Assistant Professor

CINTEL

SRM Institute of Science and Technology

TABLE OF CONTENTS

1.Problem Statement:

A neural network is used to recognize ten handwritten digits, 0-9. This is a multiclass classification task where one of n choices is selected. Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks.

MNIST ("Modified National Institute of Standards and Technology") is considered an unofficial computer vision "hello-world" dataset. This is a collection of thousands of handwritten pictures used to train classification models using Machine Learning techniques.

As a part of this problem statement, we will train a multi layer perceptron using Tensorflow -v2 to recognize the handwritten digits.

Following are the constraints faced when computers approach to recognize handwritten digits:

1. The Handwritten digits are not always of the same size, width, orientation and justified to margins as they differ from writing of person to person.

2. The similarity between digits such as 1 and 7, 5 and 6, 3 and 8, 2 and 7 etc. So, classifying between these numbers is also a major problem for computers.

3. The uniqueness and variety in the handwriting of different individuals also influence the formation and appearance of the digits.

2.Introduction:

We are implementing digit recogniton using tensorflow,taking hand-drawn images of the numbers 0-9 and build and train a neural network to recognize and predict the correct label for the digit displayed.

This model is implemented using Tensorflow implementation.Tensorflow models are built layer by layer. We specify a layer's output dimensions and this determines the next layer's input dimension.

Activation functions used in this neural network model are Relu activation function and linear activation function.Relu activation function is implemented in the input layer and the hidden layer.Linear activation function is used in output layer.

The ReLU provides a continuous linear relationship. Additionally it has an 'off' range where the output is zero. The "off" feature makes the ReLU a Non-Linear activation.

It is common because it is both simple to implement and effective at overcoming the limitations of other previously popular activation functions, such as Sigmoid and Tanh. Specifically, it is less susceptible to vanishing gradients that prevent deep models from being trained, although it can suffer from other problems like saturated or “dead” units.

The linear activation function is also called “identity” (multiplied by 1.0) or “no activation.”

This is because the linear activation function does not change the weighted sum of the input in any way and instead returns the value directly.

A multiclass neural network generates N outputs. One output is selected as the predicted answer. In the output layer, a vector **z** is generated by a linear function which is fed into a softmax function. The softmax function converts **z** into a probability distribution as described below. After applying softmax, each output will be between 0 and 1 and the outputs will sum to 1. They can be interpreted as probabilities. The larger inputs to the softmax will correspond to larger output probabilities.

Dataset:

Training,Testing and Validation:

```
history=model.fit(
          X,y,
          epochs=10,
          batch_size=10,
          verbose=1, validation_data=(X,y))
```

```
Epoch 1/10
500/500 [==============================] - 1s 3ms/step - loss: 0.1259 - accuracy: 0.9678 - val_loss: 0.1007 - val_accuracy:
0.9738
Epoch 2/10
500/500 [==============================] - 1s 2ms/step - loss: 0.1130 - accuracy: 0.9702 - val_loss: 0.0970 - val_accuracy:
0.9756
Epoch 3/10
500/500 [==============================] - 1s 2ms/step - loss: 0.1069 - accuracy: 0.9704 - val_loss: 0.0843 - val_accuracy:
0.9812
Epoch 4/10
500/500 [==============================] - 1s 3ms/step - loss: 0.0978 - accuracy: 0.9744 - val_loss: 0.0806 - val_accuracy:
0.9788
Epoch 5/10
500/500 [==============================] - 1s 3ms/step - loss: 0.0908 - accuracy: 0.9764 - val_loss: 0.0743 - val_accuracy:
0.9838
Epoch 6/10
500/500 [==============================] - 1s 3ms/step - loss: 0.0814 - accuracy: 0.9796 - val_loss: 0.0640 - val_accuracy:
0.9844
Epoch 7/10
500/500 [==============================] - 1s 3ms/step - loss: 0.0775 - accuracy: 0.9778 - val_loss: 0.0613 - val_accuracy:
0.9866
Epoch 8/10
500/500 [==============================] - 1s 2ms/step - loss: 0.0695 - accuracy: 0.9802 - val_loss: 0.0590 - val_accuracy:
0.9862
Epoch 9/10
500/500 [==============================] - 1s 2ms/step - loss: 0.0623 - accuracy: 0.9828 - val_loss: 0.0599 - val_accuracy:
0.9818
Epoch 10/10
500/500 [==============================] - 1s 3ms/step - loss: 0.0600 - accuracy: 0.9842 - val_loss: 0.0566 - val_accuracy:
0.9846
```

Source: http://yann.lecun.com/exdb/mnist/

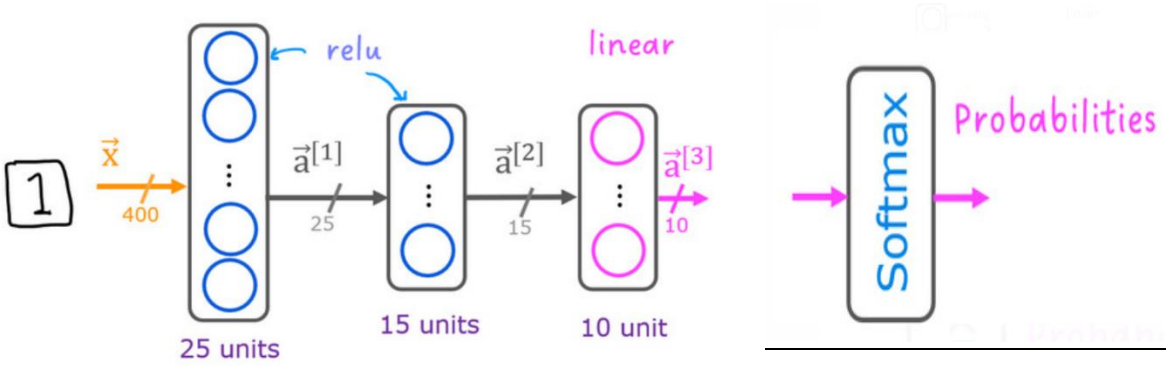
3.Methodology:

We have implemented a Neural Network with 1 hidden layer having 15 activation units (excluding bias units). The data is loaded from the MNIST data set.And the features(X) and labels(y) were extracted. Feedforward is performed with the training set for calculating the hypothesis and then backpropagation is done in order to.

The input layer is where the features of the input features (X) is fed, the Hidden layer will be the weighted sum of the inputs with parameters (W) followed by an activation function. There will always be bias (b) for each hidden layer. The output of the neural network will be the weighted sum of outputs of previous hidden layer followed by an activation function.

The output is a vector of values that may need further post-processing to convert them to business related values. For example, in a classification problem, the output is a set of probabilities that needs to be mapped to the corresponding business classes.

Neural network architecture:



Data flow in the neural network:

Layer-1(input layer)comprises of 25 units.
Layer-2(hidden layer)comprises of 15 units.
Layer-3(output layer)comprises of 10 units.
Layer-1 and Layer-2 are implemented using Relu activation.
Layer-3 is implemented using Linear activation.

Model Summary:

```
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 L1 (Dense)                  (None, 25)                10025

 L2 (Dense)                  (None, 15)                390

 L3 (Dense)                  (None, 10)                160

=================================================================
Total params: 10,575
Trainable params: 10,575
Non-trainable params: 0
```

Model Description:

An image is been extracted from the dataset and it is converted to 5x5 2-D matrix such that the pixel value ranges from 0 to 255.Then the 2-D matrix is flatend out to 1-D matrix and fed into the input layer consisting of 25 neurons.

An Input Layer that takes as input the raw data and passes them to the rest of the network.

The Hidden Layer that are intermediate layers between the input and output layer and process the data by applying complex Relu activation function to them. These layers are the key component that enables a neural network to learn complex tasks and achieve excellent performance.

An Output Layer that takes as input the processed data and produces the final results.

The output layer is the final layer in the neural network where desired predictions are obtained. There is one output layer in a neural network that produces the desired final prediction. It has its own set of weights and biases that are applied before the final output is derived.

The activation function for the output layer is linear activation function for digit recognition.Softmax activation is used to derive the final classes in a classification problem.Numerical stability is improved if the softmax is grouped with the loss function rather than the output layer during training. This has implications when building the model and using the model.

4.Implementation and Results:

Code:

```
import numpy as np
import tensorflow as tf
(X_train,Y_train),(X_test,Y_test)=keras.datasets.mnist.load_data()
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import linear, relu, sigmoid
%matplotlib widget
import matplotlib.pyplot as plt
plt.style.use('./deeplearning.mplstyle')
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)
from public_tests import *
from autils import *
from lab_utils_softmax import plt_softmax
np.set_printoptions(precision=2)
plt_act_trio()
def my_softmax(z):
    N = len(z)
    a = np.zeros(N)
    ez_sum = 0
    for k in range(N):
```

```
        ez_sum += np.exp(z[k])
    for j in range(N):
        a[j] = np.exp(z[j])/ez_sum
    return(a)
z = np.array([1., 2., 3., 4.])
a = my_softmax(z)
atf = tf.nn.softmax(z)
print(f"my_softmax(z):         {a}")
print(f"tensorflow softmax(z): {atf}")

test_my_softmax(my_softmax)

plt.close("all")
plt_softmax(my_softmax)
# load dataset
X, y = keras.datasets.mnist.load_data()
m, n = X.shape
fig, axes = plt.subplots(8,8, figsize=(5,5))
fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91]) #[left, bottom, right, top]
widgvis(fig)
for i,ax in enumerate(axes.flat):
    random_index = np.random.randint(m)
    X_random_reshaped = X[random_index].reshape((20,20)).T
    ax.imshow(X_random_reshaped, cmap='gray')
    ax.set_title(y[random_index,0])
    ax.set_axis_off()
    fig.suptitle("Label, image", fontsize=14)
```

```
tf.random.set_seed(1234) # for consistent results
model = Sequential(
    [

        tf.keras.Input(shape=(400,)),
        Dense(25, activation='relu', name = "L1"),
        Dense(15, activation='relu',  name = "L2"),
        Dense(10, activation='linear', name = "L3"),

    ], name = "my_model"
)

test_model(model, 10, 400)

[layer1, layer2, layer3] = model.layers

W1,b1 = layer1.get_weights()
W2,b2 = layer2.get_weights()
W3,b3 = layer3.get_weights()
print(f"W1 shape = {W1.shape}, b1 shape = {b1.shape}")
print(f"W2 shape = {W2.shape}, b2 shape = {b2.shape}")
print(f"W3 shape = {W3.shape}, b3 shape = {b3.shape}")
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics = ['accuracy'])
```

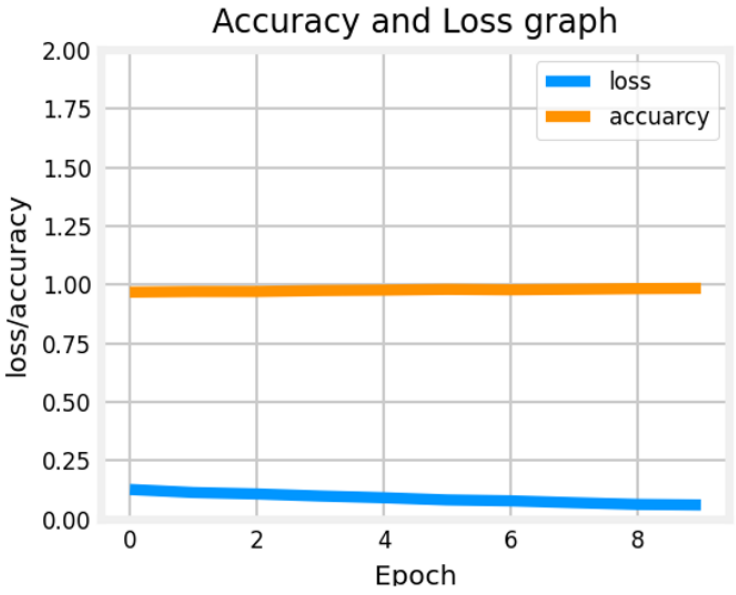
```
history=model.fit(
        X,y,
        epochs=10,
        batch_size=10,
        verbose=1, validation_data=(X,y))

plot_loss_tf(history)
history.history.keys()
plt.title(“Accuracy and Loss graph”)
plt.plot(history.history['accuracy'])
plt.legend(['loss','accuarcy'])
plt.ylabel('loss/accuracy')

image_of_two = X[1015]
display_digit(image_of_two)
prediction = model.predict(image_of_two.reshape(1,400))
print(f" predicting a Two: \n{prediction}")
print(f" Largest Prediction index: {np.argmax(prediction)}")
prediction_p = tf.nn.softmax(prediction)
print(f" predicting a Two. Probability vector: \n{prediction_p}")
print(f"Total of predictions: {np.sum(prediction_p):0.3f}")
yhat = np.argmax(prediction_p)
print(f"np.argmax(prediction_p): {yhat}")
```

```
m, n = X.shape
fig, axes = plt.subplots(8,8, figsize=(5,5))
fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91])
widgvis(fig)
for i,ax in enumerate(axes.flat):
    random_index = np.random.randint(m)
    X_random_reshaped = X[random_index].reshape((20,20)).T
    ax.imshow(X_random_reshaped, cmap='gray')
    prediction = model.predict(X[random_index].reshape(1,400))
    prediction_p = tf.nn.softmax(prediction)
    yhat = np.argmax(prediction_p)
    ax.set_title(f"{y[random_index,0]},{yhat}",fontsize=10)
    ax.set_axis_off()
fig.suptitle("Label, yhat", fontsize=14)
plt.show()
print( f"{display_errors(model,X,y)} errors out of {len(X)} images")
```

Graph:



Result:

Thus,the neural network model has been successfully implemented to recognise hand written digits.

Output:

```
image_of_two = X[1013]
display_digit(image_of_two)

prediction = model.predict(image_of_two.reshape(1,400))  # prediction

print(f" predicting a Two: \n{prediction}")
print(f" Largest Prediction index: {np.argmax(prediction)}")
```

```
 predicting a Two:
[[-10.18 -11.09   9.05   2.27 -18.21  -8.06 -12.45 -11.88   2.01  -8.29]]
 Largest Prediction index: 2
```

```
image_of_two = X[4078]
display_digit(image_of_two)

prediction = model.predict(image_of_two.reshape(1,400))  # prediction

print(f" predicting a Eight: \n{prediction}")
print(f" Largest Prediction index: {np.argmax(prediction)}")
```

```
 predicting a Eight:
[[-8.78 -1.29 -1.27 -0.36 -5.52 -4.87 -9.34 -6.38  3.34 -3.08]]
 Largest Prediction index: 8
```

```
image_of_two = X[276]
display_digit(image_of_two)

prediction = model.predict(image_of_two.reshape(1,400))  # prediction

print(f" predicting a Zero: \n{prediction}")
print(f" Largest Prediction index: {np.argmax(prediction)}")
```

```
 predicting a Zero:
[[  3.75 -18.78 -11.64  -5.57 -13.33  -1.75 -11.89  -4.37  -7.01  -1.02]]
 Largest Prediction index: 0
```

5.Inference :

We have gained knowledge on how the neural network works and how to implement a neural model.

6.References:

1.https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

2. https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/

3. http://neuralnetworksanddeeplearning.com/chap1.html

4. https://stats.stackexchange.com/questions/218542/which-activation-function-for-output-layer

5. https://becominghuman.ai/simple-neural-network-on-mnist-handwritten-digit-dataset-61e47702ed25