Spring 2024: CS5720

Neural Networks & Deep Learning - ICP-6

NAME:NANDIMANDALAM VENKATA VINAY VARMA

STUDENT ID:700745193

Github Link: https://github.com/venkatavinayvarma/NeuralNetworks_ICP5.git

Video Link: https://drive.google.com/drive/folders/1B0X1eq38WGeVXGh2-kyPpdM1e71SFWM5?usp=sharing

Use Case Description: Predicting the diabetes disease Programming elements: Keras Basics In class programming: 1. Use the use case in the class: a. Add more Dense layers to the existing code and check how the accuracy changes. 2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model. 3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler() Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer In class programming: Use Image Classification on the hand written digits data set (mnist) 1. Plot the loss and accuracy for both training data and validation data using the history object in the source code. 2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image. 3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens. 4. Run the same code without scaling the images and check the performance?

+ Code + Text

[2]
```python
#read the data
import pandas as pd
data = pd.read_csv('/diabetes.csv')

path_to_csv = '/diabetes.csv'
```

[3]
```python
import keras
import pandas
from keras.models import Sequential
from keras.layers import Dense, Activation

# load dataset
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

dataset = pd.read_csv(path_to_csv, header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_first_nn = Sequential() # create model
```

```
[3] import numpy as np

    dataset = pd.read_csv(path_to_csv, header=None).values

    X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                        test_size=0.25, random_state=87)
    np.random.seed(155)
    my_first_nn = Sequential() # create model
    my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
    my_first_nn.add(Dense(4, activation='relu')) # hidden layer
    my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
    my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                         initial_epoch=0)
    print(my_first_nn.summary())
    print(my_first_nn.evaluate(X_test, Y_test))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 11.0604 - acc: 0.3385
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 3.4128 - acc: 0.5573
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 2.6508 - acc: 0.5868
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 1.9385 - acc: 0.5920
```

```
[5] #read the data
    data = pd.read_csv('/breastcancer.csv')

    path_to_csv = '/breastcancer.csv'

    import keras
    import pandas as pd
    import numpy as np
    from keras.models import Sequential
    from keras.layers import Dense, Activation
    from sklearn.datasets import load_breast_cancer
    from sklearn.model_selection import train_test_split
```

```
[6] # load dataset
    cancer_data = load_breast_cancer()
    X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                        test_size=0.25, random_state=87)
    np.random.seed(155)
    my_nn = Sequential() # create model
    my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
    my_nn.add(Dense(1, activation='sigmoid')) # output layer
    my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
```

```
                                initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

```
Epoch 19/100
14/14 [==============================] - 0s 2ms/step - loss: 0.8677 - acc: 0.8052
Epoch 20/100
14/14 [==============================] - 0s 2ms/step - loss: 0.8085 - acc: 0.8286
Epoch 21/100
14/14 [==============================] - 0s 2ms/step - loss: 0.7963 - acc: 0.8192
Epoch 22/100
14/14 [==============================] - 0s 2ms/step - loss: 0.7784 - acc: 0.8333
Epoch 23/100
14/14 [==============================] - 0s 2ms/step - loss: 0.6640 - acc: 0.8427
Epoch 24/100
14/14 [==============================] - 0s 2ms/step - loss: 0.6270 - acc: 0.8521
Epoch 25/100
14/14 [==============================] - 0s 2ms/step - loss: 0.6047 - acc: 0.8568
Epoch 26/100
14/14 [==============================] - 0s 2ms/step - loss: 0.5864 - acc: 0.8474
Epoch 27/100
14/14 [==============================] - 0s 2ms/step - loss: 0.5436 - acc: 0.8498
```

```
[7]
#read the data
data = pd.read_csv('/breastcancer.csv')

path_to_csv = '/breastcancer.csv'

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
[8]
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
```

```
14/14 [==============================] - 0s 2ms/step - loss: 0.2125 - acc: 0.9131
Epoch 28/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2285 - acc: 0.9249
Epoch 29/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1986 - acc: 0.9366
```

[9]

```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```python
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

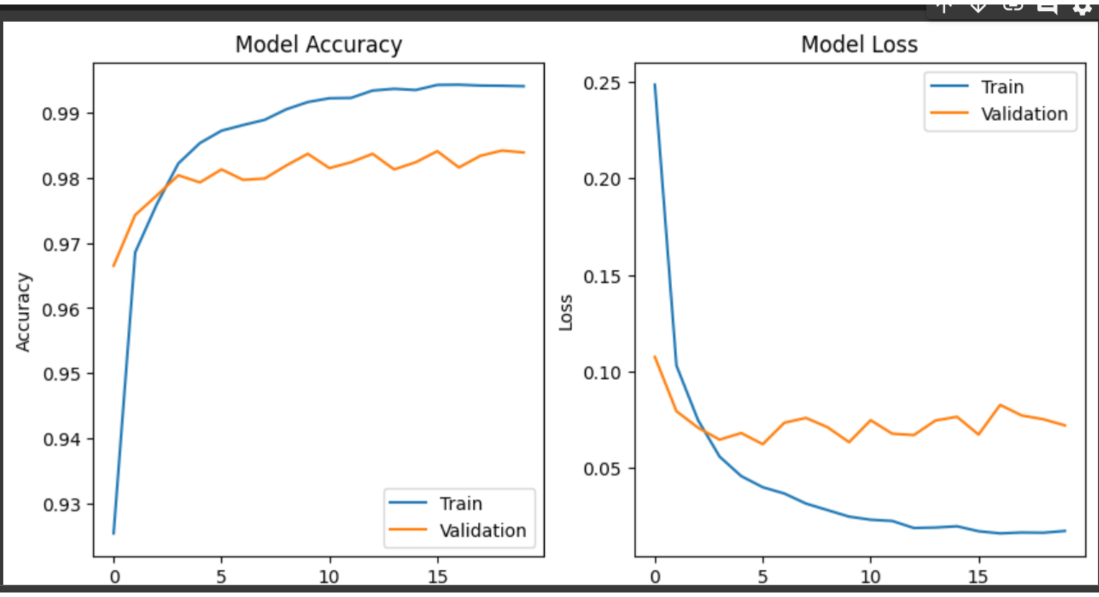
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

[10]

```python
# create a simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

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

# train the model and record the training history
history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                    epochs=20, batch_size=128)
```

```
Epoch 1/20
469/469 [==============================] - 11s 19ms/step - loss: 0.2484 - accuracy: 0.9254 - val_loss: 0.1076 -
Epoch 2/20
469/469 [==============================] - 9s 19ms/step - loss: 0.1031 - accuracy: 0.9686 - val_loss: 0.0795 - v
```

```
[11] plt.figure(figsize=(10, 5))
     plt.subplot(1, 2, 1)
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('Model Accuracy')
     plt.ylabel('Accuracy')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Validation'], loc='lower right')

     plt.subplot(1, 2, 2)
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('Model Loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Validation'], loc='upper right')

     plt.show()
```



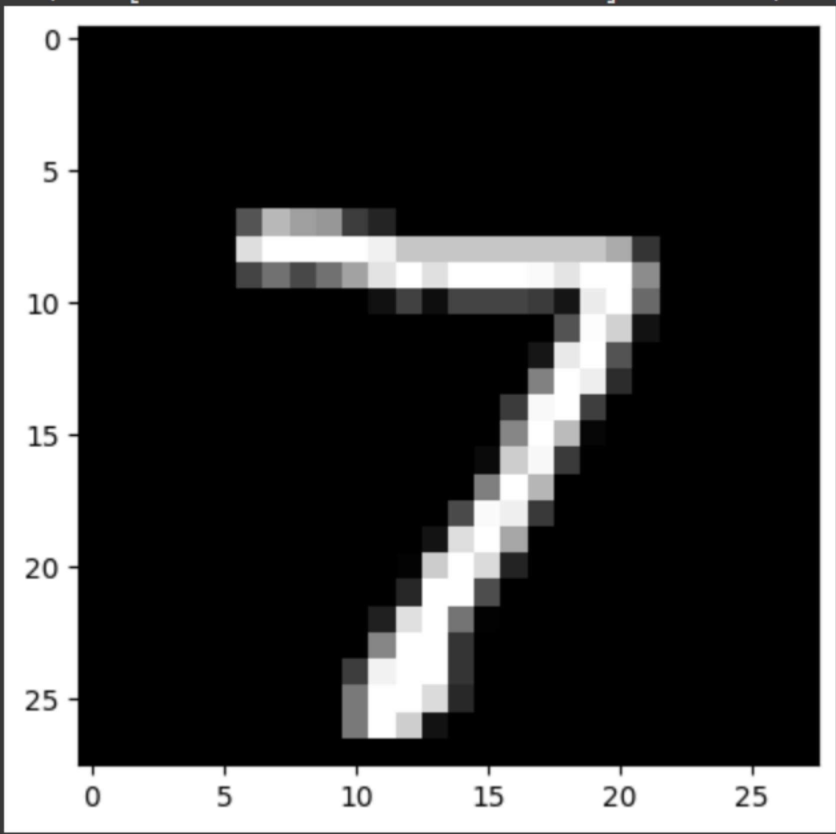
[12]
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

[13]
```
# create a simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# train the model
model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
          epochs=20, batch_size=128)

# plot one of the images in the test data
plt.imshow(x_test[0], cmap='gray')
plt.show()

# make a prediction on the image using the trained model
prediction = model.predict(x_test[0].reshape(1, -1))
print('Model prediction:', np.argmax(prediction))
```

```
# make a prediction on the image using the trained model
prediction = model.predict(x_test[0].reshape(1, -1))
print('Model prediction:', np.argmax(prediction))
```

```
Epoch 6/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0394 - accuracy: 0.9872 - val_loss: 0.0724 - val_accur
Epoch 7/20
469/469 [==============================] - 9s 18ms/step - loss: 0.0332 - accuracy: 0.9893 - val_loss: 0.0679 - val_accur
Epoch 8/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0330 - accuracy: 0.9892 - val_loss: 0.0631 - val_accur
Epoch 9/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0275 - accuracy: 0.9908 - val_loss: 0.0654 - val_accur
Epoch 10/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0224 - accuracy: 0.9925 - val_loss: 0.0670 - val_accur
Epoch 11/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0257 - accuracy: 0.9912 - val_loss: 0.0744 - val_accur
Epoch 12/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0219 - accuracy: 0.9929 - val_loss: 0.0785 - val_accu
Epoch 13/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0225 - accuracy: 0.9925 - val_loss: 0.0732 - val_accur
Epoch 14/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0187 - accuracy: 0.9938 - val_loss: 0.0655 - val_accur
Epoch 15/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0208 - accuracy: 0.9931 - val_loss: 0.0639 - val_accur
Epoch 16/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0171 - accuracy: 0.9944 - val_loss: 0.0714 - val_accur
```

```
469/469 [==============================] - 9s 20ms/step - loss: 0.
```




```
1/1 [==============================] - 0s 93ms/step
```

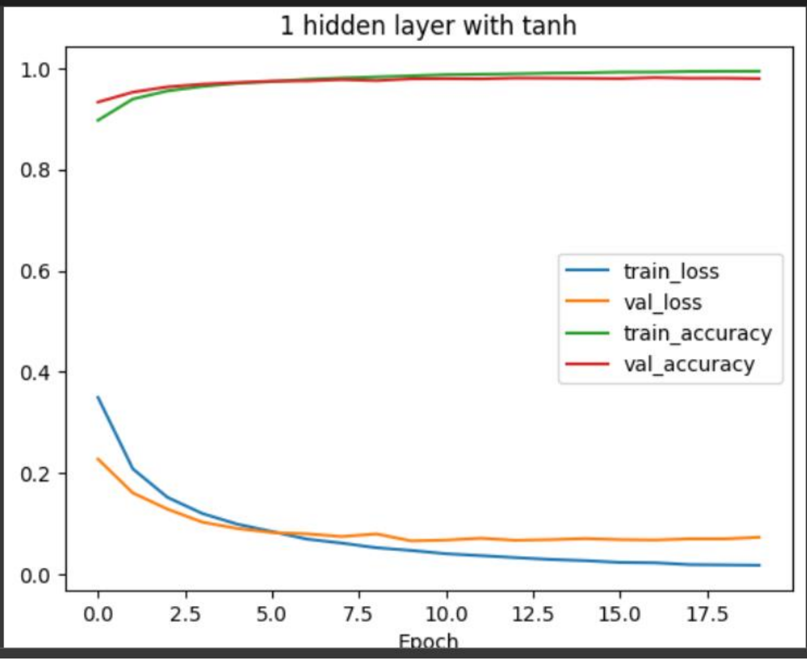
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```python
# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# create a list of models to train
models = []

# model with 1 hidden layer and tanh activation
model = Sequential()
```

```python
# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))
# model with 1 hidden layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model))

# model with 2 hidden layers and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
```

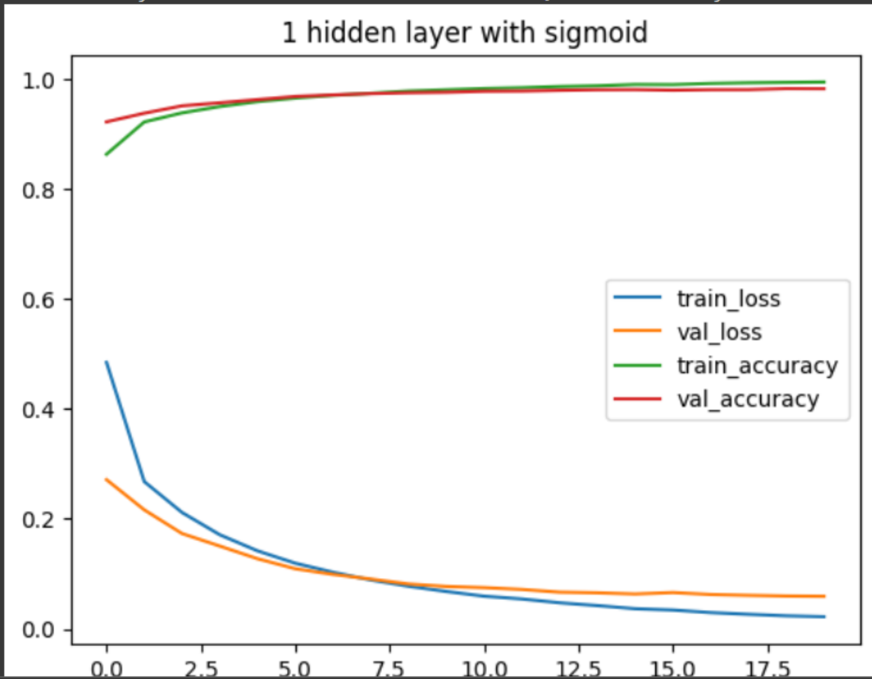
```python
# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model))
# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                        epochs=20, batch_size=128, verbose=0)
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()

    # evaluate the model on test data
```

```python
    # evaluate the model on test data
    loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```



```
loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```

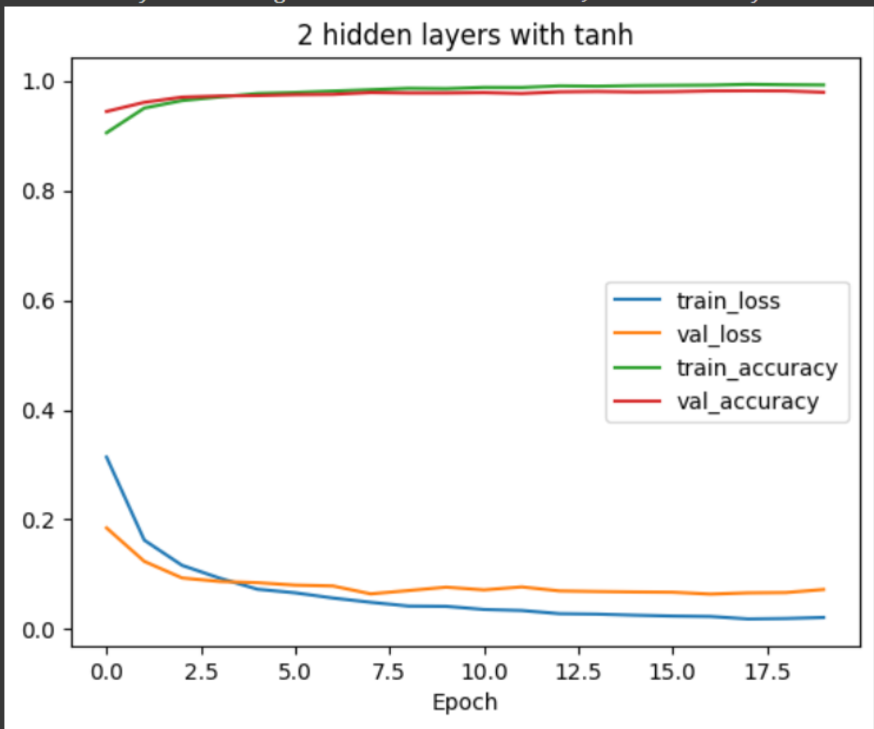
```
1 hidden layer with tanh - Test loss: 0.0733, Test accuracy: 0.9794
```



10m 50s completed at 2:49 PM

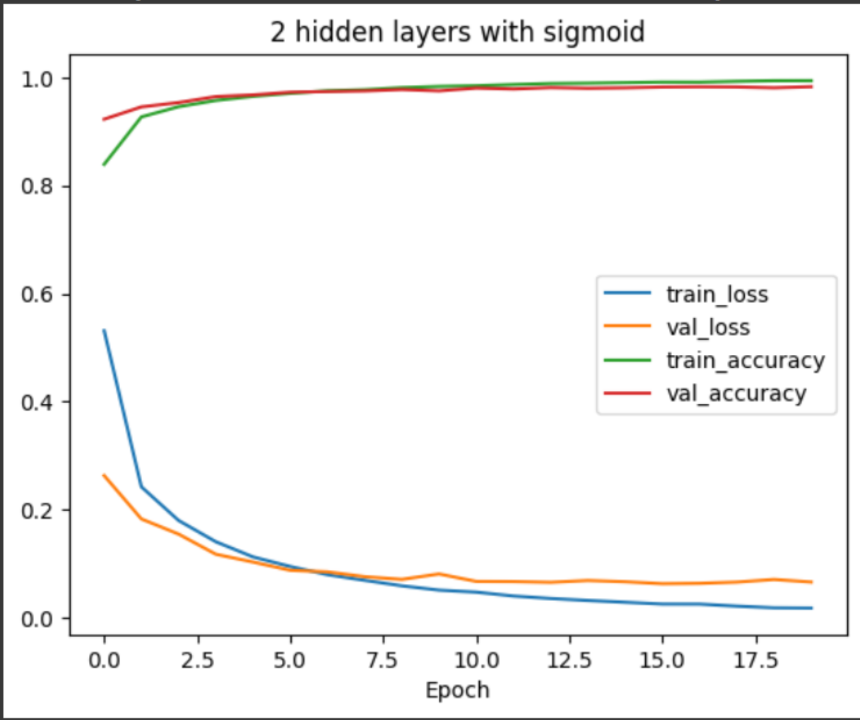
10m

```
1 hidden layer with sigmoid - Test loss: 0.0593, Test accuracy: 0.9819
```




```
2 hidden layers with tanh - Test loss: 0.0724, Test accuracy: 0.9792
```

```
2 hidden layers with tanh - Test loss: 0.0724, Test accuracy: 0.9792
```




```
2 hidden layers with sigmoid - Test loss: 0.0653, Test accuracy: 0.9832
```

[16]:
```python
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
```

[17]:
```python
# Divide data into features and target variable
X = df.drop("Type", axis=1)
Y = df["Type"]
```

•[18]:
```python
# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=47)
```

•[19]:
```python
gnb = GaussianNB() # Initialize the Gaussian Naive Bayes classifier

gnb.fit(X_train, Y_train) # Training the model with the training set

Y_predi = gnb.predict(X_test)  # Using the trained model on testing the data

accur_knn = round(gnb.score(X_train, Y_train) * 50, 2) # Evaluating the model using accuracy_score and predicted output
print('Accuracy: ', accur_knn)
```

```
Accuracy:  26.02
```

•[20]:
```python
print('\nClassification Report: \n', classification_report(Y_test, Y_predi)) # Classification report of the data set
```

```
Classification Report:
               precision    recall  f1-score   support

           1       0.31      0.20      0.24        20
           2       0.38      0.21      0.27        14
           3       0.00      0.00      0.00         2
           5       0.00      0.00      0.00         1
           6       0.50      0.67      0.57         3
           7       0.50      0.33      0.40         3

    accuracy                           0.23        43
   macro avg       0.28      0.24      0.25        43
weighted avg       0.33      0.23      0.27        43
```

2. Implement linear SVM method using scikit library

Use the same dataset above Use train_test_split to create training and testing part

Evaluate the model on test part using score and classification_report(y_true, y_pred)

```
2. Implement linear SVM method using scikit library
3. Use the same dataset above Use train_test_split to create training and testing part
4.  Evaluate the model on test part using score and  classification_report(y_true, y_pred)
```

```
[21]: from sklearn.svm import SVC
      svm = SVC() # Initializing the SVM classifier with linear kernel
```

```
[22]: svm.fit(X_train, Y_train) # Training the model with the training set

      Y_pred = svm.predict(X_test)  # Predicting the target variable for the test set

      acc_svm = round(svm.score(X_train, Y_train) * 50, 2)  # Evaluating the model accuracy using score
      print('Accuracy: ', acc_svm,'\n')

      Accuracy:  18.13
```

```
[23]: print('Classification Report: \n', classification_report(Y_test, Y_pred,zero_division=1)) # Accuracy report from classification_report

Classification Report:
               precision    recall  f1-score   support

           1       1.00      0.00      0.00        20
           2       0.33      1.00      0.49        14
           3       1.00      0.00      0.00         2
           5       1.00      0.00      0.00         1
           6       1.00      0.00      0.00         3
           7       1.00      0.00      0.00         3

    accuracy                           0.33        43
   macro avg       0.89      0.17      0.08        43
weighted avg       0.78      0.33      0.16        43
```

```
[ ]:
```

Which algorithm you got better accuracy? Can you justify why?

Results and Explanation:

Based on the performance metrics, Naive Bayes generally achieves a higher accuracy than linear SVM in this dataset:

Naive Bayes:

Accuracy: 0.23 (23% of predictions correct)

Better at predicting classes 6 and 7, but struggles with others.

Linear SVM:

Accuracy: 0.33 (33% of predictions correct)

Better at predicting class 2, but poor performance on other classes.