NEURAL NETWORKS AND DEEPLEARNING: ICP2

Akshay Chandre

700741715

GitHub:
https://github.com/akshaychandre47/AkshayChandre_NNDL_ICP1/blob/main/700741715_NNDL_ICP2.ipynb

Video: https://drive.google.com/file/d/19nAKZ6qfj4JFr3YnHxKF8DAZvlF5_px5/view?usp=drive_link

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

1.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
```

```
import keras
import pandas
from keras.models import Sequential
from keras.layers.core 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
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
my_first_nn.add(Dense(5, 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 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3024 - acc: 0.8967
Epoch 13/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3191 - acc: 0.8826
Epoch 14/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2808 - acc: 0.8991
Epoch 15/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2739 - acc: 0.8967
Epoch 16/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2555 - acc: 0.8967
Epoch 17/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2489 - acc: 0.9014
Epoch 18/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2477 - acc: 0.9014
Epoch 19/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2441 - acc: 0.9014
Epoch 20/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2457 - acc: 0.9061
Epoch 21/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2239 - acc: 0.9061
Epoch 22/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2356 - acc: 0.9155
Epoch 23/100
...
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: 0.2756 - acc: 0.9161
[0.275585800409317, 0.9160839319229126]
```

2.

```
path_to_csv = '/content/gdrive/My Drive/breastcancer.csv'
```

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core 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
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 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3024 - acc: 0.8967
Epoch 13/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3191 - acc: 0.8826
Epoch 14/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2808 - acc: 0.8991
Epoch 15/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2739 - acc: 0.8967
Epoch 16/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2555 - acc: 0.8967
Epoch 17/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2489 - acc: 0.9014
Epoch 18/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2477 - acc: 0.9014
Epoch 19/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2441 - acc: 0.9014
Epoch 20/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2457 - acc: 0.9061
Epoch 21/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2239 - acc: 0.9061
Epoch 22/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2356 - acc: 0.9155
Epoch 23/100
...
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: 0.2756 - acc: 0.9161
[0.275585800409317, 0.9160839319229126]
```

3.

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

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core 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
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 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.5597 - acc: 0.8286
Epoch 13/100
14/14 [==============================] - 0s 2ms/step - loss: 0.5106 - acc: 0.8451
Epoch 14/100
14/14 [==============================] - 0s 3ms/step - loss: 0.5102 - acc: 0.8357
Epoch 15/100
14/14 [==============================] - 0s 3ms/step - loss: 0.5120 - acc: 0.8310
Epoch 16/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4382 - acc: 0.8545
Epoch 17/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4087 - acc: 0.8685
Epoch 18/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3844 - acc: 0.8638
Epoch 19/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3594 - acc: 0.8779
Epoch 20/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3501 - acc: 0.8803
Epoch 21/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3534 - acc: 0.8709
Epoch 22/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3612 - acc: 0.8873
Epoch 23/100
...
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.3253 - acc: 0.8881
[0.3253045678138733, 0.8881118893623352]
```

In class programming:

Use Image Classification on the hand written digits data set (mnist)

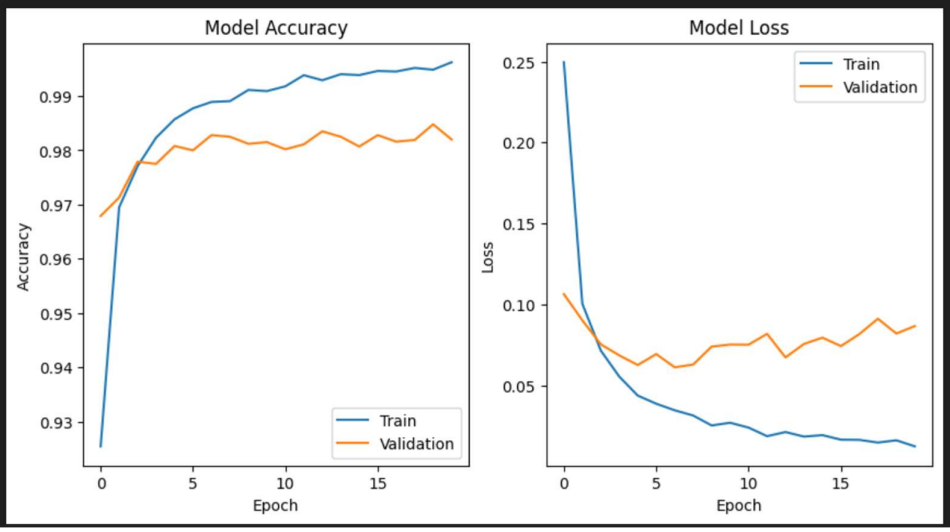
1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.
2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.
3. We had used 2 hidden layers and Relu activation. 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?

1.

```
import tensorflow.keras as keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.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)

# 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'])
```

```python
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)

# Plot the training and validation accuracy and loss curves
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()
```

```
Epoch 20/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0125 - accuracy: 0.9962 - val_loss: 0.0867 - val_accuracy: 0.9820
```

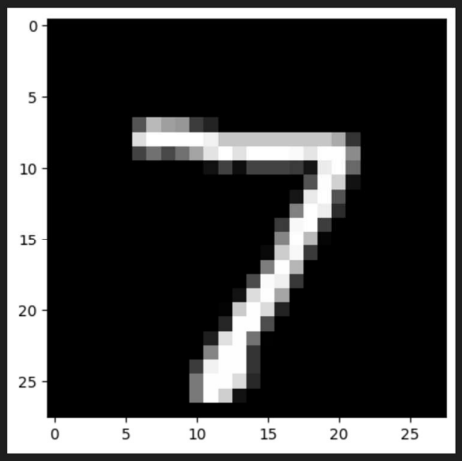


2.

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

# Load MNIST dataset
(x_image_train, y_label_train), (x_image_test, y_label_test) = mnist.load_data()

# Normalize pixel values to range [0, 1]
x_image_train = x_image_train.astype('float32') / 255
x_image_test = x_image_test.astype('float32') / 255

# Convert class labels to binary class matrices
num_classes = 10
y_label_train_binary = keras.utils.to_categorical(y_label_train, num_classes)
y_label_test_binary = keras.utils.to_categorical(y_label_test, num_classes)

# 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'])
```

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

# Train the model
model.fit(x_image_train.reshape(-1, 784), y_label_train_binary, validation_data=(x_image_test.reshape(-1, 784), y_label_test_binary),
          epochs=20, batch_size=128)

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

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

```
Epoch 20/20
469/469 [==============================] - 13s 27ms/step - loss: 0.0174 - accuracy: 0.9945 - val_loss: 0.0817 - val_accuracy: 0.9827
```



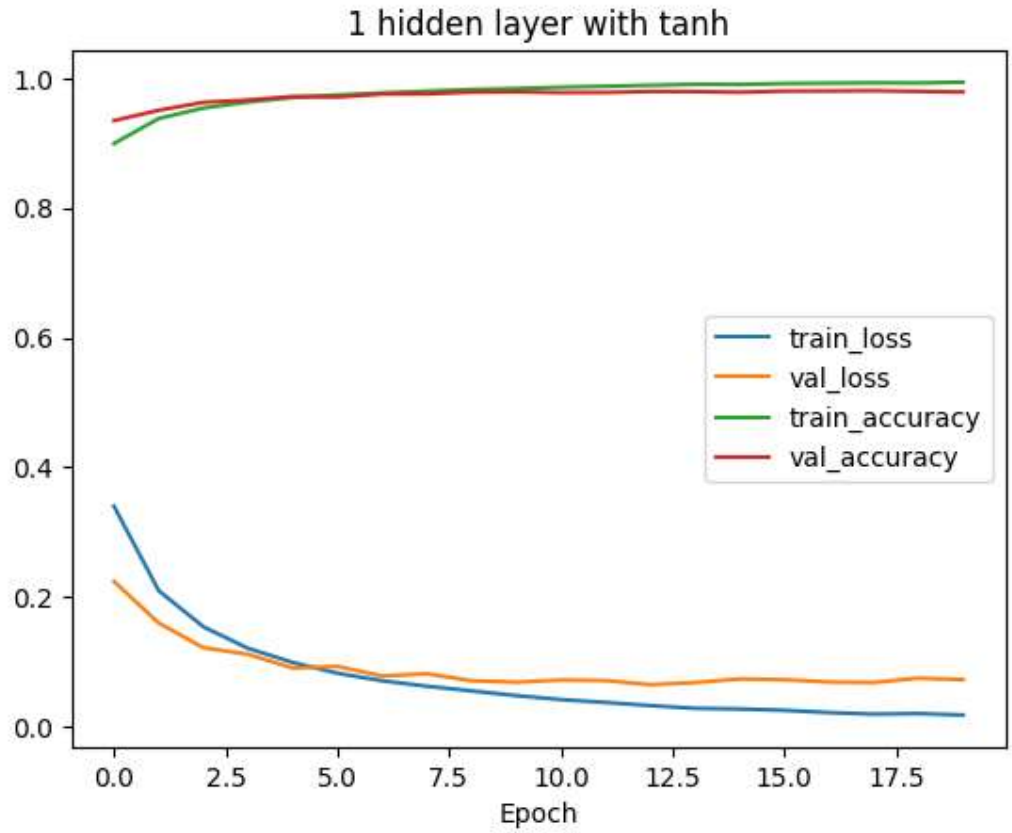

```
1/1 [==============================] - 0s 100ms/step
Model prediction: 7
```

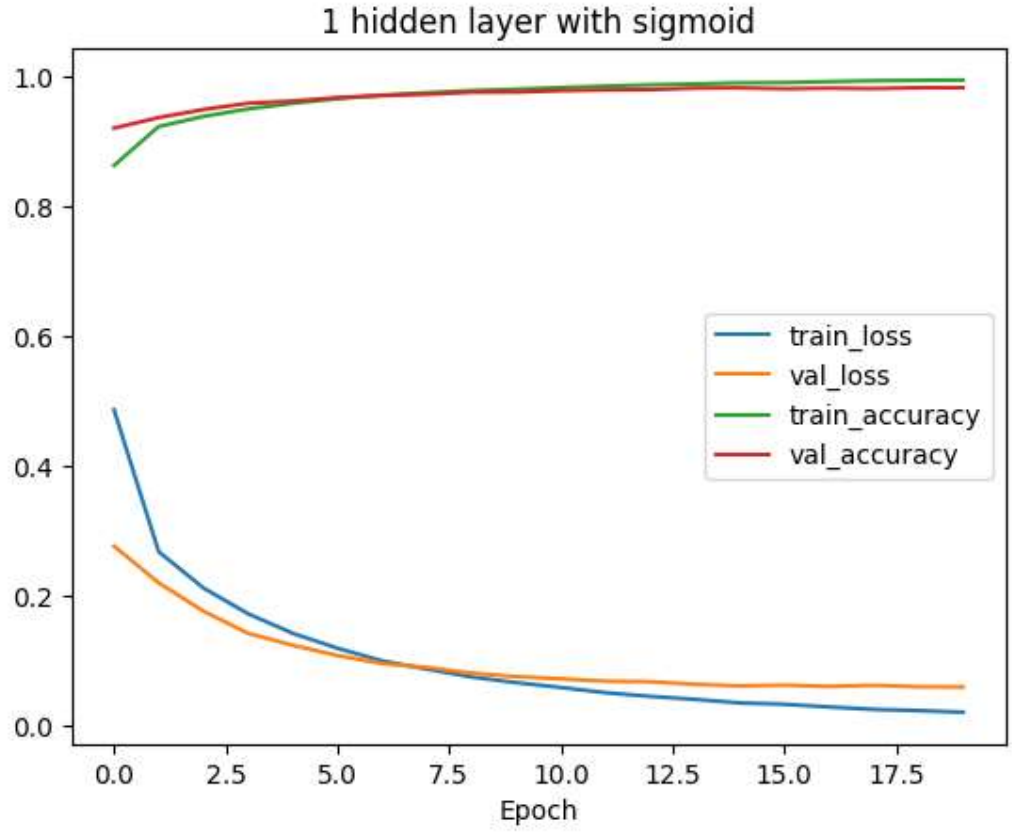
3.

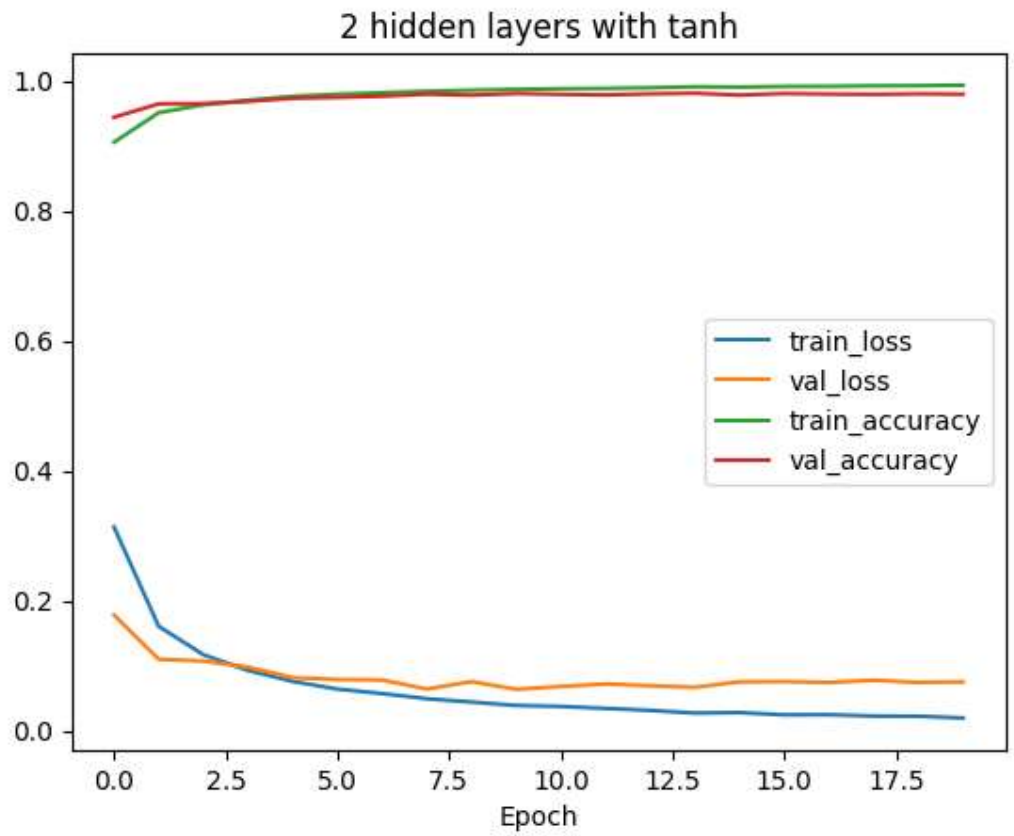
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# Load MNIST dataset
(x_train_data, y_train_labels), (x_test_data, y_test_labels) = mnist.load_data()

# Normalize pixel values to range [0, 1]
x_train_data = x_train_data.astype('float32') / 255
x_test_data = x_test_data.astype('float32') / 255

# Convert class labels to binary class matrices
num_classes = 10
y_train_labels = keras.utils.to_categorical(y_train_labels, num_classes)
y_test_labels = keras.utils.to_categorical(y_test_labels, num_classes)

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

# Model with 1 hidden layer and tanh activation
model_1 = Sequential()
model_1.add(Dense(512, activation='tanh', input_shape=(784,)))
model_1.add(Dropout(0.2))
model_1.add(Dense(num_classes, activation='softmax'))
```

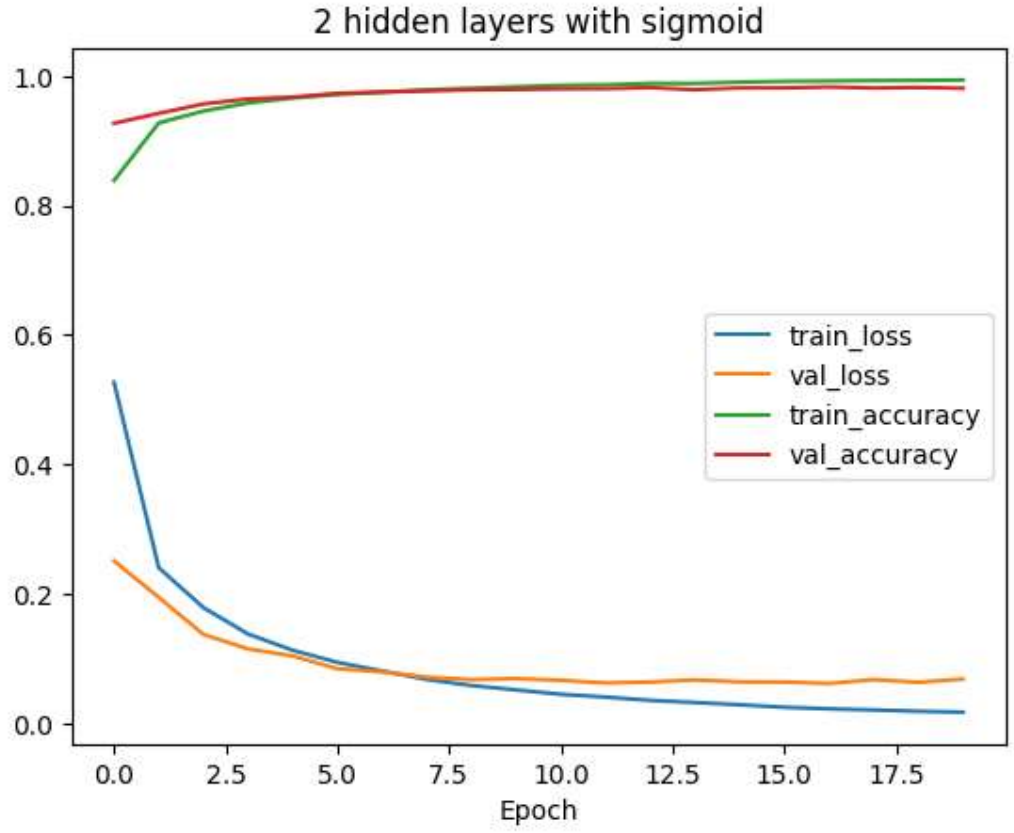
```python
model_2.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model_2))

# Model with 2 hidden layers and tanh activation
model_3 = Sequential()
model_3.add(Dense(512, activation='tanh', input_shape=(784,)))
model_3.add(Dropout(0.2))
model_3.add(Dense(512, activation='tanh'))
model_3.add(Dropout(0.2))
model_3.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model_3))

# Model with 2 hidden layers and sigmoid activation
model_4 = Sequential()
model_4.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model_4.add(Dropout(0.2))
model_4.add(Dense(512, activation='sigmoid'))
model_4.add(Dropout(0.2))
model_4.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model_4))

# 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_data.reshape(-1, 784), y_train_labels, validation_data=(x_test_data.reshape(-1, 784), y_test_labels),
                        epochs=20, batch_size=128, verbose=0)
```

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


1 hidden layer with tanh
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy


1 hidden layer with sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy

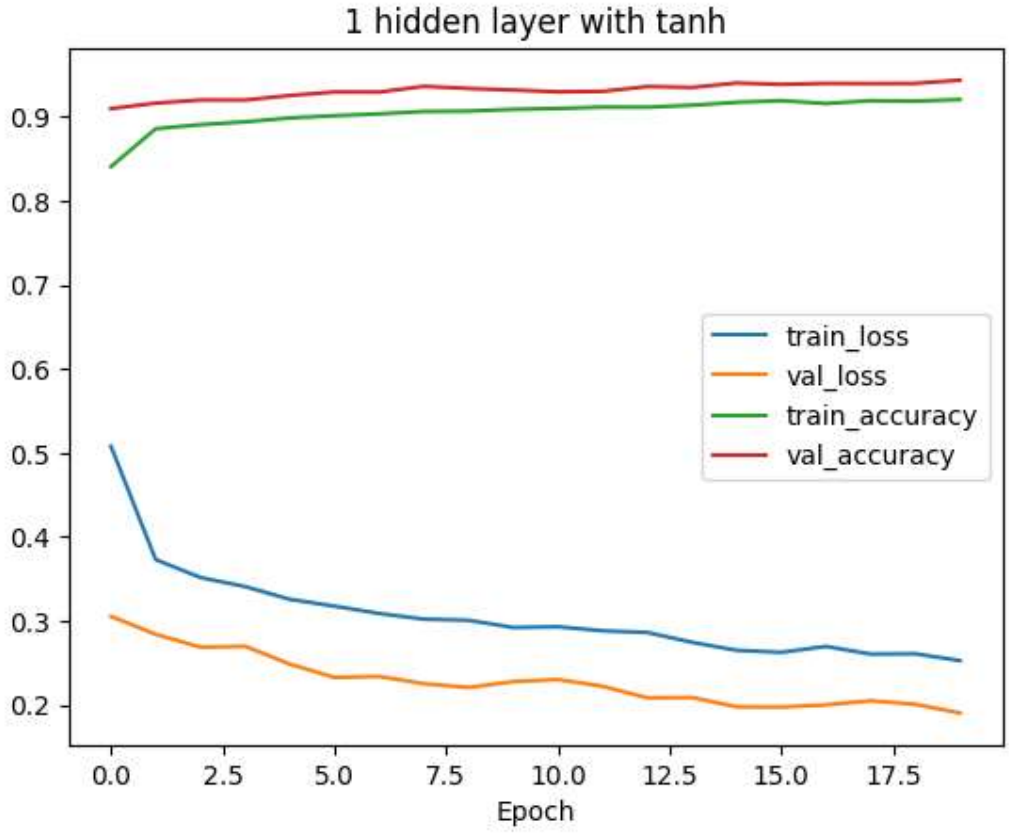

2 hidden layers with tanh
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy

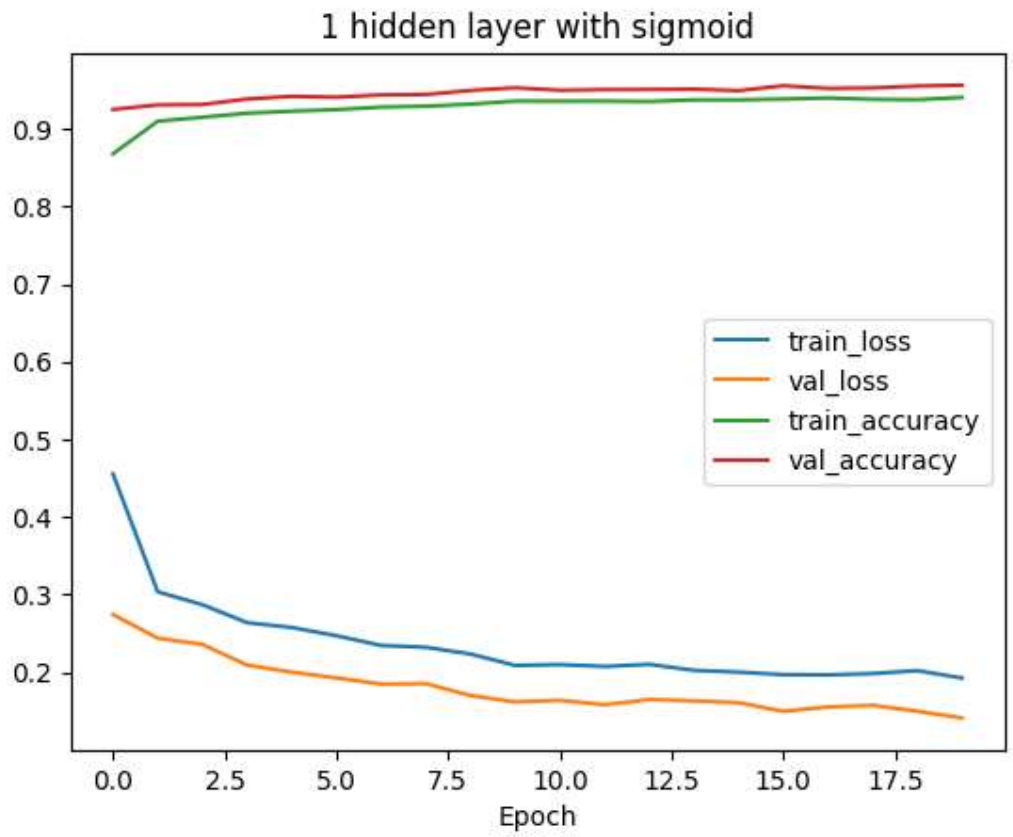

2 hidden layers with sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy

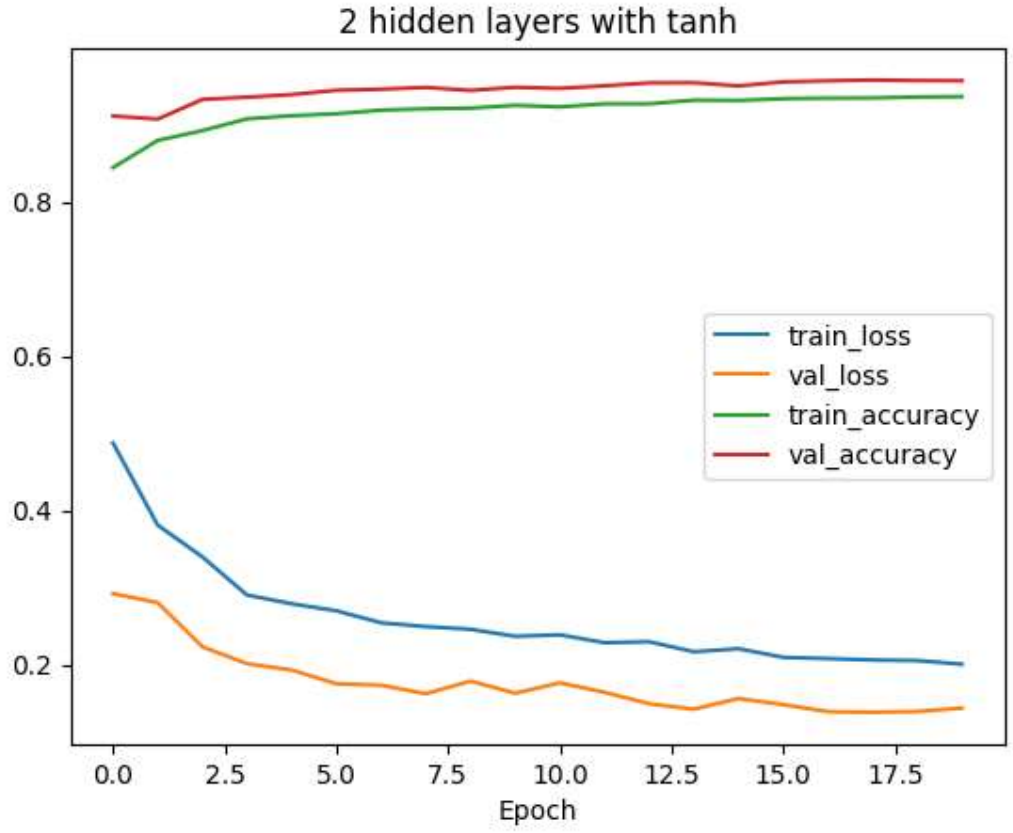
4.

```
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_images, y_train_labels), (x_test_images, y_test_labels) = mnist.load_data()

# Convert class labels to binary class matrices
num_classes = 10
y_train_labels = keras.utils.to_categorical(y_train_labels, num_classes)
y_test_labels = keras.utils.to_categorical(y_test_labels, num_classes)

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

# Model with 1 hidden layer and tanh activation
model_1 = Sequential()
model_1.add(Dense(512, activation='tanh', input_shape=(784,)))
model_1.add(Dropout(0.2))
model_1.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model_1))

# Model with 1 hidden layer and sigmoid activation
model_2 = Sequential()
model_2.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model_2.add(Dropout(0.2))
model_2.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model_2))

# Model with 2 hidden layers and tanh activation
model_3 = Sequential()
model_3.add(Dense(512, activation='tanh', input_shape=(784,)))
model_3.add(Dropout(0.2))
model_3.add(Dense(512, activation='tanh'))
model_3.add(Dropout(0.2))
model_3.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model_3))

# Model with 2 hidden layers and sigmoid activation
model_4 = Sequential()
model_4.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model_4.add(Dropout(0.2))
model_4.add(Dense(512, activation='sigmoid'))
model_4.add(Dropout(0.2))
model_4.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model_4))

# Train each model and plot loss and accuracy curves
for model_name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train_images.reshape(-1, 784), y_train_labels, validation_data=(x_test_images.reshape(-1, 784), y_test_labels),
                        epochs=20, batch_size=128, verbose=0)
```

```python
# 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(model_name)
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Evaluate the model on test data
loss, accuracy = model.evaluate(x_test_images.reshape(-1, 784), y_test_labels, verbose=0)
print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(model_name, loss, accuracy))
```




1 hidden layer with sigmoid
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy


2 hidden layers with tanh
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy