NAME :

NET ID :

SUBMISSION DATE :

Instructions for submission

HOW TO DO THE ASSIGNMENT?

```
>> Download this jupyter notebook file to your local computer.
>> Rename it by adding your netid. 'CS4347-Assighnment4-NetId.ipynb
>> Now start working on assignment
```

HOW TO SUBMIT THE ASSIGNMENT?

```
>> Before you submit make sure you have the final version of your
work.
>> Submit the jupyter notebook ('CS4347-Assighnment4-NetId.ipynb)
with all cells' running results to Canvas.
>> Save the jupyter notebook with all cells' running results as a
pdf, submit the pdf file to Canvas as well.
```

ASSIGNMENT 4 - INTRO TO MACHINE LEARNING - Nerual Networks

> **FULL MARKS = 120 points**

In this assignment we will ...

In [47]:
```
# import all the packages that you will need during this assignment.
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
```

Exercise 1: Neural Networks for Handwritten Digit Recognition, Binary

In this exercise, you will use a neural network to recognize the hand-written digits zero and one.

Neural Networks

Previously, you implemented logistic regression. This was extended to handle non-linear boundaries using polynomial regression. For even more complex scenarios such as image recognition, neural networks are preferred.

Problem Statement

In this exercise, you will use a neural network to recognize two handwritten digits, zero and one. This is a binary classification task. Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks. You will extend this network to recognize all 10 digits (0-9) in a future assignment.

This exercise will show you how the methods you have learned can be used for this classification task.

Dataset

You will start by loading the dataset for this task.

- The `load_data()` function shown below loads the data into variables `X` and `y`
- The data set contains 1000 training examples of handwritten digits [1], here limited to zero and one.
  - Each training example is a 20-pixel x 20-pixel grayscale image of the digit.
    - Each pixel is represented by a floating-point number indicating the grayscale intensity at that location.
    - The 20 by 20 grid of pixels is “unrolled” into a 400-dimensional vector.
    - Each training example becomes a single row in our data matrix `X`.
    - This gives us a 1000 x 400 matrix `X` where every row is a training example of a handwritten digit image.

$$X = \begin{pmatrix} --- (x^{(1)}) --- \\ --- (x^{(2)}) --- \\ \vdots \\ --- (x^{(m)}) --- \end{pmatrix}$$

- The second part of the training set is a 1000 x 1 dimensional vector `y` that contains labels for the training set
  - `y = 0` if the image is of the digit `0`, `y = 1` if the image is of the digit `1`.

[1] This is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/)

In [48]:
```python
def load_data():
    X = np.load("data_ex1/X.npy")
    y = np.load("data_ex1/y.npy")
    X = X[0:1000]
    y = y[0:1000]
    return X, y
```

In [49]:
```python
# load dataset
X, y = load_data()
```

View the variables

Let's get more familiar with your dataset.

- A good place to start is to print out each variable and see what it contains.

The code below prints elements of the variables `X` and `y`.

In [50]:
```python
print ('The first element of X is: ', X[0])
print ('The first element of y is: ', y[0,0])
print ('The last element of y is: ', y[-1,0])
```

```
The first element of X is:  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.0000
0000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  8.56059680e-06
  1.94035948e-06 -7.37438725e-04 -8.13403799e-03 -1.86104473e-02
 -1.87412865e-02 -1.87572508e-02 -1.90963542e-02 -1.64039011e-02
 -3.78191381e-03  3.30347316e-04  1.27655229e-05  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  1.16421569e-04  1.20052179e-04
 -1.40444581e-02 -2.84542484e-02  8.03826593e-02  2.66540339e-01
  2.73853746e-01  2.78729541e-01  2.74293607e-01  2.24676403e-01
  2.77562977e-02 -7.06315478e-03  2.34715414e-04  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.28335523e-17 -3.26286765e-04 -1.38651604e-02
  8.15651552e-02  3.82800381e-01  8.57849775e-01  1.00109761e+00
  9.69710638e-01  9.30928598e-01  1.00383757e+00  9.64157356e-01
  4.49256553e-01 -5.60408259e-03 -3.78319036e-03  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  5.10620915e-06
  4.36410675e-04 -3.95509940e-03 -2.68537241e-02  1.00755014e-01
  6.42031710e-01  1.03136838e+00  8.50968614e-01  5.43122379e-01
  3.42599738e-01  2.68918777e-01  6.68374643e-01  1.01256958e+00
  9.03795598e-01  1.04481574e-01 -1.66424973e-02  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  2.59875260e-05
 -3.10606987e-03  7.52456076e-03  1.77539831e-01  7.92890120e-01
  9.65626503e-01  4.63166079e-01  6.91720680e-02 -3.64100526e-03
 -4.12180405e-02 -5.01900656e-02  1.56102907e-01  9.01762651e-01
  1.04748346e+00  1.51055252e-01 -2.16044665e-02  0.00000000e+00
  0.00000000e+00  0.00000000e+00  5.87012352e-05 -6.40931373e-04
 -3.23305249e-02  2.78203465e-01  9.36720163e-01  1.04320956e+00
  5.98003217e-01 -3.59409041e-03 -2.16751770e-02 -4.81021923e-03
  6.16566793e-05 -1.23773318e-02  1.55477482e-01  9.14867477e-01
  9.20401348e-01  1.09173902e-01 -1.71058007e-02  0.00000000e+00
  0.00000000e+00  1.56250000e-04 -4.27724104e-04 -2.51466503e-02
  1.30532561e-01  7.81664862e-01  1.02836583e+00  7.57137601e-01
  2.84667194e-01  4.86865128e-03 -3.18688725e-03  0.00000000e+00
  8.36492601e-04 -3.70751123e-02  4.52644165e-01  1.03180133e+00
  5.39028101e-01 -2.43742611e-03 -4.80290033e-03  0.00000000e+00
  0.00000000e+00 -7.03635621e-04 -1.27262443e-02  1.61706648e-01
  7.79865383e-01  1.03676705e+00  8.04490400e-01  1.60586724e-01
 -1.38173339e-02  2.14879493e-03 -2.12622549e-04  2.04248366e-04
 -6.85907627e-03  4.31712963e-04  7.20680947e-01  8.48136063e-01
  1.51383408e-01 -2.28404366e-02  1.98971950e-04  0.00000000e+00
```

```
 0.00000000e+00 -9.40410539e-03  3.74520505e-02  6.94389110e-01
 1.02844844e+00  1.01648066e+00  8.80488426e-01  3.92123945e-01
-1.74122413e-02 -1.20098039e-04  5.55215142e-05 -2.23907271e-03
-2.76068376e-02  3.68645493e-01  9.36411169e-01  4.59006723e-01
-4.24701797e-02  1.17356610e-03  1.88929739e-05  0.00000000e+00
 0.00000000e+00 -1.93511951e-02  1.29999794e-01  9.79821705e-01
 9.41862388e-01  7.75147704e-01  8.73632241e-01  2.12778350e-01
-1.72353349e-02  0.00000000e+00  1.09937426e-03 -2.61793751e-02
 1.22872879e-01  8.30812662e-01  7.26501773e-01  5.24441863e-02
-6.18971913e-03  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00 -9.36563862e-03  3.68349741e-02  6.99079299e-01
 1.00293583e+00  6.05704402e-01  3.27299224e-01 -3.22099249e-02
-4.83053002e-02 -4.34069138e-02 -5.75151144e-02  9.55674190e-02
 7.26512627e-01  6.95366966e-01  1.47114481e-01 -1.20048679e-02
-3.02798203e-04  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00 -6.76572712e-04 -6.51415556e-03  1.17339359e-01
 4.21948410e-01  9.93210937e-01  8.82013974e-01  7.45758734e-01
 7.23874268e-01  7.23341725e-01  7.20020340e-01  8.45324959e-01
 8.31859739e-01  6.88831870e-02 -2.77765012e-02  3.59136710e-04
 7.14869281e-05  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  1.53186275e-04  3.17353553e-04 -2.29167177e-02
-4.14402914e-03  3.87038450e-01  5.04583435e-01  7.74885876e-01
 9.90037446e-01  1.00769478e+00  1.00851440e+00  7.37905042e-01
 2.15455291e-01 -2.69624864e-02  1.32506127e-03  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  2.36366422e-04
-2.26031454e-03 -2.51994485e-02 -3.73889910e-02  6.62121228e-02
 2.91134498e-01  3.23055726e-01  3.06260315e-01  8.76070942e-02
-2.50581917e-02  2.37438725e-04  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  6.20939216e-18  6.72618320e-04 -1.13151411e-02
-3.54641066e-02 -3.88214912e-02 -3.71077412e-02 -1.33524928e-02
 9.90964718e-04  4.89176960e-05  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
The first element of y is:  0
The last element of y is:  1
```

In [51]:

```python
print ('The shape of X is: ' + str(X.shape))
print ('The shape of y is: ' + str(y.shape))
```

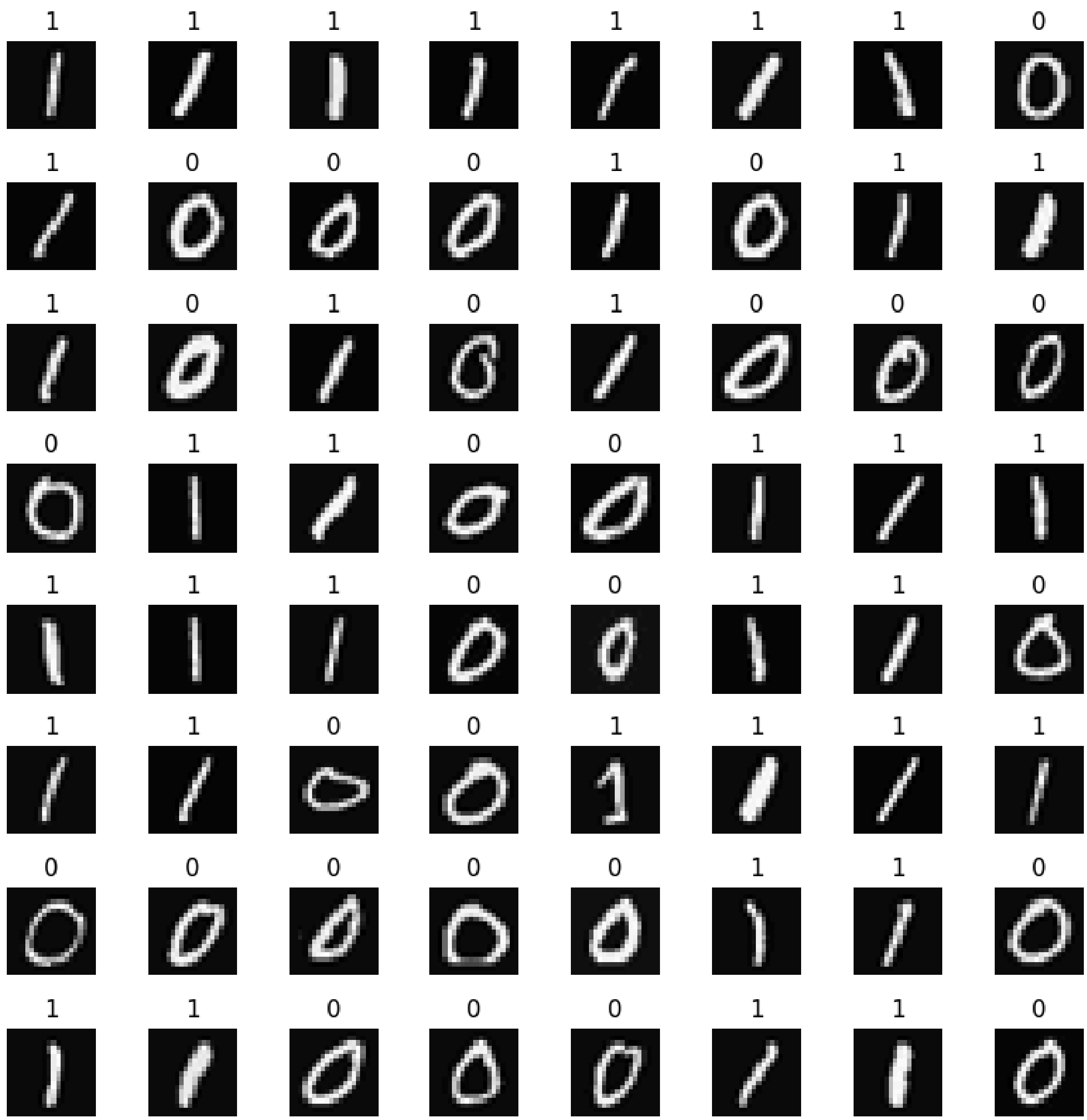
```
The shape of X is: (1000, 400)
The shape of y is: (1000, 1)
```

Visualizing the Data

You will begin by visualizing a subset of the training set.

- In the cell below, the code randomly selects 64 rows from `X`, maps each row back to a 20 pixel by 20 pixel grayscale image and displays the images together.
- The label for each image is displayed above the image

In [52]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

m, n = X.shape

fig, axes = plt.subplots(8,8, figsize=(8,8))
fig.tight_layout(pad=0.1)

for i,ax in enumerate(axes.flat):
    # Select random indices
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T

    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Display the label above the image
    ax.set_title(y[random_index,0])
    ax.set_axis_off()
plt.show()
```



Exercise 1 [20 points]

Below, using Keras Sequential model and Dense Layer with a sigmoid activation to construct the network described below.

- The neural network has three dense layers with sigmoid activations.
- Since the images are of size 20×20, this gives us 400 inputs
- The parameters have dimensions that are sized for a neural network with 25 units in layer 1, 15 units in layer 2 and 1 output unit in layer 3.

In [53]:

```
# GRADED CELL: Sequential model

model = Sequential(
    [
        ### START CODE HERE ###
```

```
        tf.keras.Input(shape=(400,)),
        Dense(25, activation="sigmoid", name="layer1"),
        Dense(15, activation="sigmoid", name="layer2"),
        Dense(1, activation="sigmoid", name="layer3")
        ### END CODE HERE ###
    ], name = "my_model"
)
```

In [54]:
```
model.summary()
```

Model: "my_model"

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 25)	10,025
layer2 (Dense)	(None, 15)	390
layer3 (Dense)	(None, 1)	16

Total params: 10,431 (40.75 KB)

Trainable params: 10,431 (40.75 KB)

Non-trainable params: 0 (0.00 B)

▶ **Expected Output (Click to Expand)**

The parameter counts shown in the summary correspond to the number of elements in the weight and bias arrays as shown below.

In [55]:
```
L1_num_params = 400 * 25 + 25  # W1 parameters  + b1 parameters
L2_num_params = 25 * 15 + 15   # W2 parameters  + b2 parameters
L3_num_params = 15 * 1 + 1     # W3 parameters  + b3 parameters
print("L1 params = ", L1_num_params, ", L2 params = ", L2_num_params, ",  L3 params
```

```
L1 params =  10025 , L2 params =  390 ,  L3 params =  16
```

Let's further examine the weights to verify that tensorflow produced the same dimensions as we calculated above.

In [56]:
```
[layer1, layer2, layer3] = model.layers
```

In [57]:
```
#### Examine Weights shapes
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}")
```

```
W1 shape = (400, 25), b1 shape = (25,)
W2 shape = (25, 15), b2 shape = (15,)
W3 shape = (15, 1), b3 shape = (1,)
```

In [58]:
```
print(model.layers[2].weights)
```

```
[<Variable path=my_model/layer3/kernel, shape=(15, 1), dtype=float32, value=[[ 0.449
1232 ]
 [-0.0729351 ]
 [-0.21013978]
 [-0.26162514]
 [ 0.00764054]
 [-0.32960477]
 [ 0.60828525]
 [ 0.04044491]
 [-0.21457413]
 [ 0.3757875 ]
 [ 0.21734071]
 [ 0.39721137]
 [-0.15484482]
 [ 0.10201937]
 [-0.38693586]]>, <Variable path=my_model/layer3/bias, shape=(1,), dtype=float32, va
lue=[0.]>]
```

The following code will define a loss function and run gradient descent to fit the weights of the model to the training data.

In [59]:
```
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(0.001),
)

model.fit(
    X,y,
    epochs=20
)
```

```
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.6749
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.5231
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.3889
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2790
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.2018
Epoch 6/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1509
Epoch 7/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1172
Epoch 8/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0942
Epoch 9/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0779
Epoch 10/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0658
Epoch 11/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 968us/step - loss: 0.0567
Epoch 12/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 0.0497
Epoch 13/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 887us/step - loss: 0.0442
Epoch 14/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1000us/step - loss: 0.0398
Epoch 15/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 984us/step - loss: 0.0362
Epoch 16/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 0.0333
Epoch 17/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0308
Epoch 18/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 968us/step - loss: 0.0287
Epoch 19/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0269
Epoch 20/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 903us/step - loss: 0.0253
```

Out[59]:
```
<keras.src.callbacks.history.History at 0x16d71b75280>
```

To run the model on an example to make a prediction, use Keras `predict`. The input to `predict` is an array so the single example is reshaped to be two dimensional.

In [60]:
```python
prediction = model.predict(X[0].reshape(1,400))  # a zero
print(f" predicting a zero: {prediction}")
prediction = model.predict(X[500].reshape(1,400))  # a one
print(f" predicting a one:  {prediction}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
 predicting a zero: [[0.01557071]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
 predicting a one:  [[0.97938555]]
```

The output of the model is interpreted as a probability. In the first example above, the input is a zero. The model predicts the probability that the input is a one is nearly zero. In the second example, the input is a one. The model predicts the probability that the input is a one is nearly one. As in the case of logistic regression, the probability is compared to a threshold to make a final prediction.

In [61]:
```
if prediction >= 0.5:
    yhat = 1
else:
    yhat = 0
print(f"prediction after threshold: {yhat}")
```

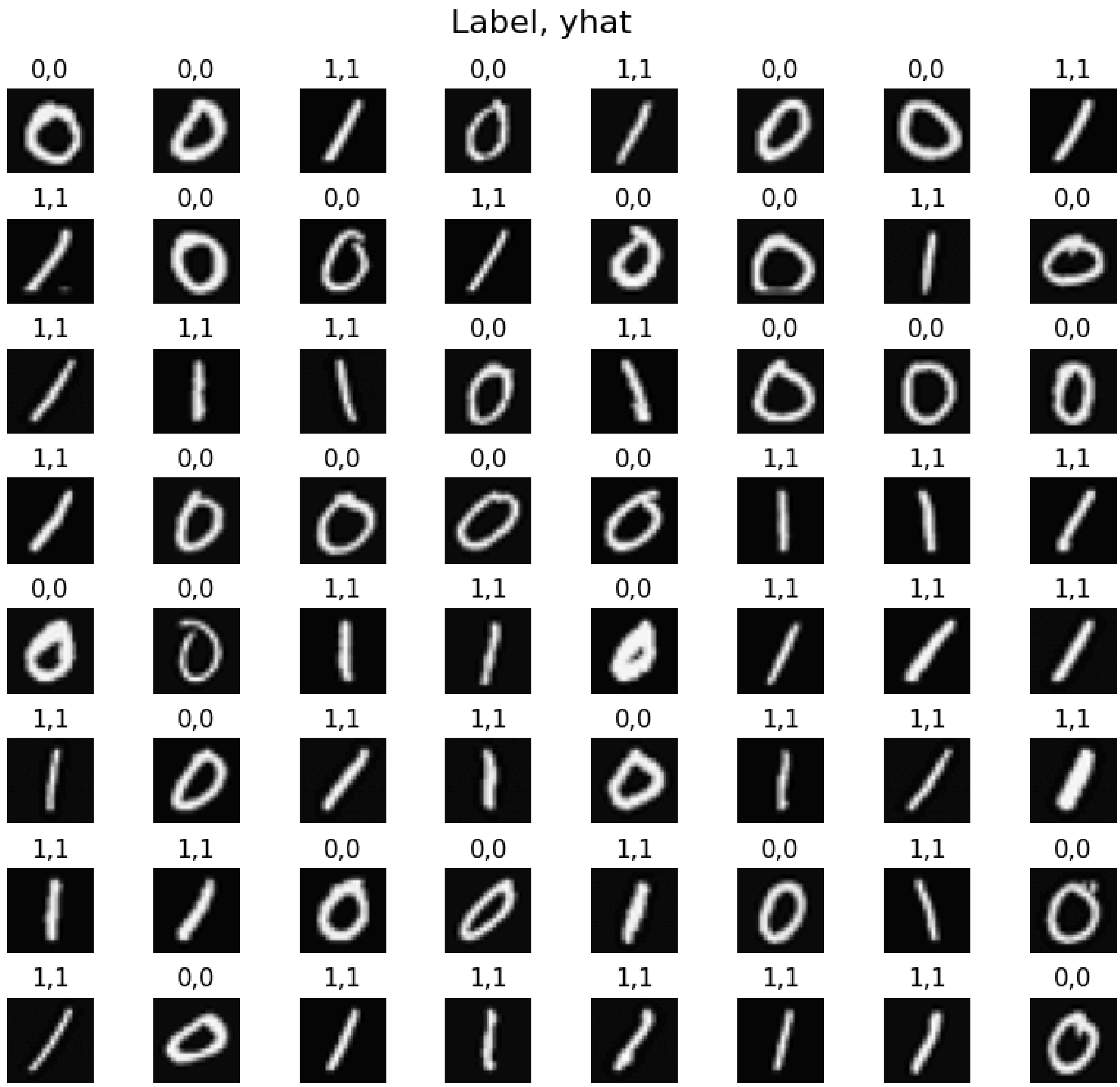
```
prediction after threshold: 1
```

Let's compare the predictions vs the labels for a random sample of 64 digits. This takes a moment to run.

In [62]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

m, n = X.shape

fig, axes = plt.subplots(8,8, figsize=(8,8))
fig.tight_layout(pad=0.1,rect=[0, 0.03, 1, 0.92]) #[left, bottom, right, top]

for i,ax in enumerate(axes.flat):
    # Select random indices
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T

    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Predict using the Neural Network
    prediction = model.predict(X[random_index].reshape(1,400))
    if prediction >= 0.5:
        yhat = 1
    else:
        yhat = 0

    # Display the label above the image
    ax.set_title(f"{y[random_index,0]},{yhat}")
    ax.set_axis_off()
fig.suptitle("Label, yhat", fontsize=16)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



Exercise 2:Neural Networks for Handwritten Digit Recognition, Multiclass

In this exercise, you will use a neural network to recognize the hand-written digits 0-9.

In [63]:
```
from tensorflow.keras.activations import linear, relu, sigmoid
```

Softmax Function

A multiclass neural network generates N outputs. One output is selected as the predicted answer. In the output layer, a vector $\mathbf{z}$ is generated by a linear function which is fed into a softmax function. The softmax function converts $\mathbf{z}$ 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.

The softmax function can be written:

$$a_j = \frac{e^{z_j}}{\sum_{k=0}^{N-1} e^{z_k}} \tag{1}$$

Where $z = \mathbf{w} \cdot \mathbf{x} + b$ and N is the number of feature/categories in the output layer.

Exercise 2.1 [10 points]

Let's create a NumPy implementation of softmax function:

In [64]:
```
def my_softmax(z):
    """ Softmax converts a vector of values to a probability distribution.
    Args:
      z (ndarray (N,))  : input data, N features
    Returns:
      a (ndarray (N,))  : softmax of z
    """
    ### START CODE HERE ###
    z_x = np.exp(z - np.max(z))
    a = z_x / np.sum(z_x)
    ### END CODE HERE ###
    return a
```

In [65]:
```
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}")
print("my_softmax(z) and tensorflow softmax(z) should provide the same results")
```

```
my_softmax(z):         [0.0320586  0.08714432 0.23688282 0.64391426]
tensorflow softmax(z): [0.0320586  0.08714432 0.23688282 0.64391426]
my_softmax(z) and tensorflow softmax(z) should provide the same results
```

In [66]:
```
print(f"The sum of my_softmax(z) is {sum(a)}")
print(f"The sum of tensorflow softmax(z) is {sum(atf)}")
```

```
The sum of my_softmax(z) is 1.0
The sum of tensorflow softmax(z) is 1.0
```

Neural Networks

In Exercise 1, you implemented a neural network to do binary classification. In Exercise 2 you will extend that to multiclass classification. This will utilize the softmax activation.

Problem Statement

In this exercise, you will use a neural network 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.

Dataset

You will start by loading the dataset for this task.

- The `load_data()` function shown below loads the data into variables `X` and `y`
- The data set contains 5000 training examples of handwritten digits [1].
  - Each training example is a 20-pixel x 20-pixel grayscale image of the digit.
    - Each pixel is represented by a floating-point number indicating the grayscale intensity at that location.
    - The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector.
    - Each training examples becomes a single row in our data matrix `X`.
    - This gives us a 5000 x 400 matrix `X` where every row is a training example of a handwritten digit image.

$$X = \left(\begin{array}{cc} --- (x^{(1)}) --- \\ --- (x^{(2)}) --- \\ \vdots \\ --- (x^{(m)}) --- \end{array}\right)$$

- The second part of the training set is a 5000 x 1 dimensional vector `y` that contains labels for the training set
  - `y = 0` if the image is of the digit `0`, `y = 4` if the image is of the digit `4` and so on.

[1] This is a subset of the MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/)

In [67]:
```
def load_data():
    X = np.load("data_ex2/X.npy")
    y = np.load("data_ex2/y.npy")
    return X, y
```

In [68]:
```
# load dataset
X, y = load_data()
```

In [69]:
```
print ('The first element of X is: ', X[0])
```

```
The first element of X is:  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.0000
0000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  8.56059680e-06
  1.94035948e-06 -7.37438725e-04 -8.13403799e-03 -1.86104473e-02
 -1.87412865e-02 -1.87572508e-02 -1.90963542e-02 -1.64039011e-02
 -3.78191381e-03  3.30347316e-04  1.27655229e-05  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  1.16421569e-04  1.20052179e-04
 -1.40444581e-02 -2.84542484e-02  8.03826593e-02  2.66540339e-01
  2.73853746e-01  2.78729541e-01  2.74293607e-01  2.24676403e-01
  2.77562977e-02 -7.06315478e-03  2.34715414e-04  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.28335523e-17 -3.26286765e-04 -1.38651604e-02
  8.15651552e-02  3.82800381e-01  8.57849775e-01  1.00109761e+00
  9.69710638e-01  9.30928598e-01  1.00383757e+00  9.64157356e-01
  4.49256553e-01 -5.60408259e-03 -3.78319036e-03  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  5.10620915e-06
  4.36410675e-04 -3.95509940e-03 -2.68537241e-02  1.00755014e-01
  6.42031710e-01  1.03136838e+00  8.50968614e-01  5.43122379e-01
  3.42599738e-01  2.68918777e-01  6.68374643e-01  1.01256958e+00
  9.03795598e-01  1.04481574e-01 -1.66424973e-02  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  2.59875260e-05
 -3.10606987e-03  7.52456076e-03  1.77539831e-01  7.92890120e-01
  9.65626503e-01  4.63166079e-01  6.91720680e-02 -3.64100526e-03
 -4.12180405e-02 -5.01900656e-02  1.56102907e-01  9.01762651e-01
  1.04748346e+00  1.51055252e-01 -2.16044665e-02  0.00000000e+00
  0.00000000e+00  0.00000000e+00  5.87012352e-05 -6.40931373e-04
 -3.23305249e-02  2.78203465e-01  9.36720163e-01  1.04320956e+00
  5.98003217e-01 -3.59409041e-03 -2.16751770e-02 -4.81021923e-03
  6.16566793e-05 -1.23773318e-02  1.55477482e-01  9.14867477e-01
  9.20401348e-01  1.09173902e-01 -1.71058007e-02  0.00000000e+00
  0.00000000e+00  1.56250000e-04 -4.27724104e-04 -2.51466503e-02
  1.30532561e-01  7.81664862e-01  1.02836583e+00  7.57137601e-01
  2.84667194e-01  4.86865128e-03 -3.18688725e-03  0.00000000e+00
  8.36492601e-04 -3.70751123e-02  4.52644165e-01  1.03180133e+00
  5.39028101e-01 -2.43742611e-03 -4.80290033e-03  0.00000000e+00
  0.00000000e+00 -7.03635621e-04 -1.27262443e-02  1.61706648e-01
  7.79865383e-01  1.03676705e+00  8.04490400e-01  1.60586724e-01
 -1.38173339e-02  2.14879493e-03 -2.12622549e-04  2.04248366e-04
 -6.85907627e-03  4.31712963e-04  7.20680947e-01  8.48136063e-01
  1.51383408e-01 -2.28404366e-02  1.98971950e-04  0.00000000e+00
```

```
  0.00000000e+00 -9.40410539e-03  3.74520505e-02  6.94389110e-01
  1.02844844e+00  1.01648066e+00  8.80488426e-01  3.92123945e-01
 -1.74122413e-02 -1.20098039e-04  5.55215142e-05 -2.23907271e-03
 -2.76068376e-02  3.68645493e-01  9.36411169e-01  4.59006723e-01
 -4.24701797e-02  1.17356610e-03  1.88929739e-05  0.00000000e+00
  0.00000000e+00 -1.93511951e-02  1.29999794e-01  9.79821705e-01
  9.41862388e-01  7.75147704e-01  8.73632241e-01  2.12778350e-01
 -1.72353349e-02  0.00000000e+00  1.09937426e-03 -2.61793751e-02
  1.22872879e-01  8.30812662e-01  7.26501773e-01  5.24441863e-02
 -6.18971913e-03  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -9.36563862e-03  3.68349741e-02  6.99079299e-01
  1.00293583e+00  6.05704402e-01  3.27299224e-01 -3.22099249e-02
 -4.83053002e-02 -4.34069138e-02 -5.75151144e-02  9.55674190e-02
  7.26512627e-01  6.95366966e-01  1.47114481e-01 -1.20048679e-02
 -3.02798203e-04  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -6.76572712e-04 -6.51415556e-03  1.17339359e-01
  4.21948410e-01  9.93210937e-01  8.82013974e-01  7.45758734e-01
  7.23874268e-01  7.23341725e-01  7.20020340e-01  8.45324959e-01
  8.31859739e-01  6.88831870e-02 -2.77765012e-02  3.59136710e-04
  7.14869281e-05  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.53186275e-04  3.17353553e-04 -2.29167177e-02
 -4.14402914e-03  3.87038450e-01  5.04583435e-01  7.74885876e-01
  9.90037446e-01  1.00769478e+00  1.00851440e+00  7.37905042e-01
  2.15455291e-01 -2.69624864e-02  1.32506127e-03  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  2.36366422e-04
 -2.26031454e-03 -2.51994485e-02 -3.73889910e-02  6.62121228e-02
  2.91134498e-01  3.23055726e-01  3.06260315e-01  8.76070942e-02
 -2.50581917e-02  2.37438725e-04  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  6.20939216e-18  6.72618320e-04 -1.13151411e-02
 -3.54641066e-02 -3.88214912e-02 -3.71077412e-02 -1.33524928e-02
  9.90964718e-04  4.89176960e-05  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

In [70]:

```python
print ('The first element of y is: ', y[0,0])
print ('The last element of y is: ', y[-1,0])
```

```
The first element of y is:  0
The last element of y is:  9
```

In [71]:

```python
print ('The shape of X is: ' + str(X.shape))
print ('The shape of y is: ' + str(y.shape))
```

```
The shape of X is: (5000, 400)
The shape of y is: (5000, 1)
```

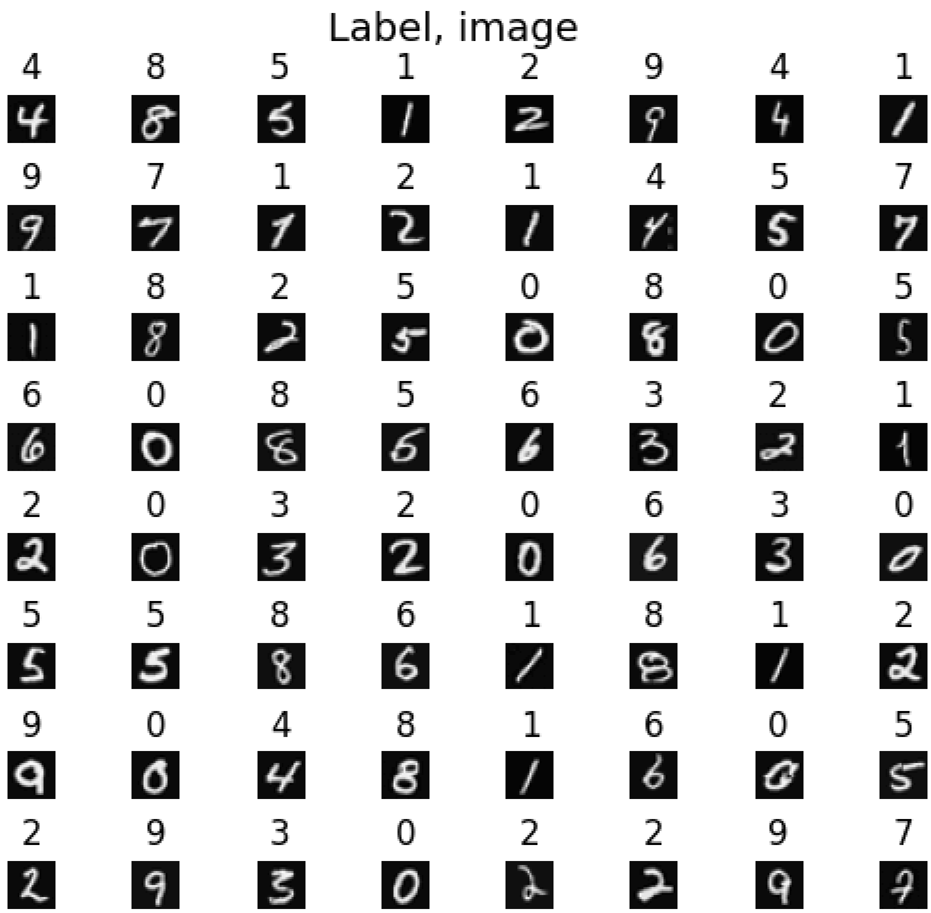
Visualizing the Data

You will begin by visualizing a subset of the training set.

- In the cell below, the code randomly selects 64 rows from `X`, maps each row back to a 20 pixel by 20 pixel grayscale image and displays the images together.
- The label for each image is displayed above the image

In [75]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

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]

for i,ax in enumerate(axes.flat):
    # Select random indices
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T

    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Display the label above the image
    ax.set_title(y[random_index,0])
    ax.set_axis_off()
    fig.suptitle("Label, image", fontsize=14)
```



Exercise 2.2 [20 points]

Below, using Keras Sequential model and Dense Layer with a ReLU activation to construct the three layer network described below.

- The neural network has two dense layers with ReLU activations followed by an output layer with a linear activation.
- Since the images are of size 20×20, this gives us 400 inputs
- The parameters have dimensions that are sized for a neural network with 25 units in layer 1, 15 units in layer 2 and 10 output units in layer 3, one for each digit.

In [78]:
```
tf.random.set_seed(1234) # for consistent results
model = Sequential(
    [
        ### START CODE HERE ###
        tf.keras.Input(shape=(400,)),
        Dense(25, activation="relu", name="layer1"),
        Dense(15, activation="relu", name="layer2"),
        Dense(10, activation="linear", name="outputLayer")
        ### END CODE HERE ###
    ], name = "my_model"
)
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

Softmax placement

As described in the lecture and the optional softmax lab, 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.
Building:

- The final Dense layer should use a 'linear' activation. This is effectively no activation.
- The `model.compile` statement will indicate this by including `from_logits=True`.

`loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`

- This does not impact the form of the target. In the case of SparseCategorialCrossentropy, the target is the expected digit, 0-9.

Using the model:

- The outputs are not probabilities. If output probabilities are desired, apply a softmax function.

In [79]:
```
model.summary()
```

Model: "my_model"

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 25)	10,025
layer2 (Dense)	(None, 15)	390
outputLayer (Dense)	(None, 10)	160

Total params: 10,575 (41.31 KB)
Trainable params: 10,575 (41.31 KB)
Non-trainable params: 0 (0.00 B)

▶ **Expected Output (Click to expand)**

In [80]:
```
[layer1, layer2, layer3] = model.layers
```

In [81]:
```
#### Examine Weights shapes
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}")
```

```
W1 shape = (400, 25), b1 shape = (25,)
W2 shape = (25, 15), b2 shape = (15,)
W3 shape = (15, 10), b3 shape = (10,)
```

The following code:

- defines a loss function, `SparseCategoricalCrossentropy` and indicates the softmax should be included with the loss calculation by adding `from_logits=True`)
- defines an optimizer. A popular choice is Adaptive Moment (Adam) which was described in lecture.

In [82]:
```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

history = model.fit(
    X,y,
    epochs=100
)
```

```
Epoch 1/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 1.8681
Epoch 2/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - loss: 0.6610
Epoch 3/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - loss: 0.4437
Epoch 4/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.3653
Epoch 5/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.3187
Epoch 6/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.2857
Epoch 7/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.2601
Epoch 8/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - loss: 0.2396
Epoch 9/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.2220
Epoch 10/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.2074
Epoch 11/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.1949
Epoch 12/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.1838
Epoch 13/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - loss: 0.1736
Epoch 14/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.1646
Epoch 15/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 930us/step - loss: 0.1562
Epoch 16/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - loss: 0.1489
Epoch 17/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 603us/step - loss: 0.1418
Epoch 18/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - loss: 0.1352
Epoch 19/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 0.1291
Epoch 20/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - loss: 0.1228
Epoch 21/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - loss: 0.1172
Epoch 22/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.1118
Epoch 23/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 620us/step - loss: 0.1069
Epoch 24/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 631us/step - loss: 0.1018
Epoch 25/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - loss: 0.0974
Epoch 26/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0928
Epoch 27/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 0.0883
Epoch 28/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 0.0839
```

```
Epoch 29/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - loss: 0.0796
Epoch 30/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0756
Epoch 31/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - loss: 0.0717
Epoch 32/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 689us/step - loss: 0.0678
Epoch 33/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 636us/step - loss: 0.0642
Epoch 34/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0610
Epoch 35/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 0.0573
Epoch 36/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0546
Epoch 37/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 615us/step - loss: 0.0515
Epoch 38/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 656us/step - loss: 0.0485
Epoch 39/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - loss: 0.0461
Epoch 40/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 631us/step - loss: 0.0432
Epoch 41/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0412
Epoch 42/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0384
Epoch 43/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - loss: 0.0364
Epoch 44/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0341
Epoch 45/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 850us/step - loss: 0.0326
Epoch 46/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 917us/step - loss: 0.0307
Epoch 47/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 748us/step - loss: 0.0290
Epoch 48/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 0.0275
Epoch 49/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - loss: 0.0261
Epoch 50/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - loss: 0.0243
Epoch 51/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 0.0227
Epoch 52/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - loss: 0.0216
Epoch 53/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 0.0205
Epoch 54/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - loss: 0.0188
Epoch 55/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 0.0177
Epoch 56/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.0168
```

```
Epoch 57/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 0.0154
Epoch 58/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 770us/step - loss: 0.0143
Epoch 59/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 0.0133
Epoch 60/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0121
Epoch 61/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step - loss: 0.0112
Epoch 62/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 705us/step - loss: 0.0101
Epoch 63/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0092
Epoch 64/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.0081
Epoch 65/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 0.0072
Epoch 66/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 0.0063
Epoch 67/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 0.0056
Epoch 68/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 679us/step - loss: 0.0049
Epoch 69/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0045
Epoch 70/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - loss: 0.0041
Epoch 71/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - loss: 0.0037
Epoch 72/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 679us/step - loss: 0.0034
Epoch 73/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 692us/step - loss: 0.0031
Epoch 74/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 0.0028
Epoch 75/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - loss: 0.0025
Epoch 76/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 0.0023
Epoch 77/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 0.0021
Epoch 78/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 719us/step - loss: 0.0020
Epoch 79/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - loss: 0.0018
Epoch 80/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0016
Epoch 81/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0015
Epoch 82/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - loss: 0.0013
Epoch 83/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.0012
Epoch 84/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 0.0011
```

```
Epoch 85/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0010
Epoch 86/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 9.5326e-04
Epoch 87/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 8.6864e-04
Epoch 88/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 8.1763e-04
Epoch 89/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - loss: 7.3379e-04
Epoch 90/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 6.9603e-04
Epoch 91/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 6.1776e-04
Epoch 92/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - loss: 5.7698e-04
Epoch 93/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - loss: 5.2421e-04
Epoch 94/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 4.8595e-04
Epoch 95/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 731us/step - loss: 4.4513e-04
Epoch 96/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 731us/step - loss: 4.0612e-04
Epoch 97/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 3.7389e-04
Epoch 98/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 3.4354e-04
Epoch 99/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 3.1863e-04
Epoch 100/100
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 2.8963e-04
```

Epochs and batches

In the `compile` statement above, the number of `epochs` was set to 100. This specifies that the entire data set should be applied during training 100 times. During training, you see output describing the progress of training that looks like this:

```
Epoch 1/100
157/157 [==============================] - 0s 1ms/step - loss:
2.2770
```

The first line, `Epoch 1/100`, describes which epoch the model is currently running. For efficiency, the training data set is broken into 'batches'. The default size of a batch in Tensorflow is 32. There are 5000 examples in our data set or roughly 157 batches. The notation on the 2nd line `157/157 [====` is describing which batch has been executed.

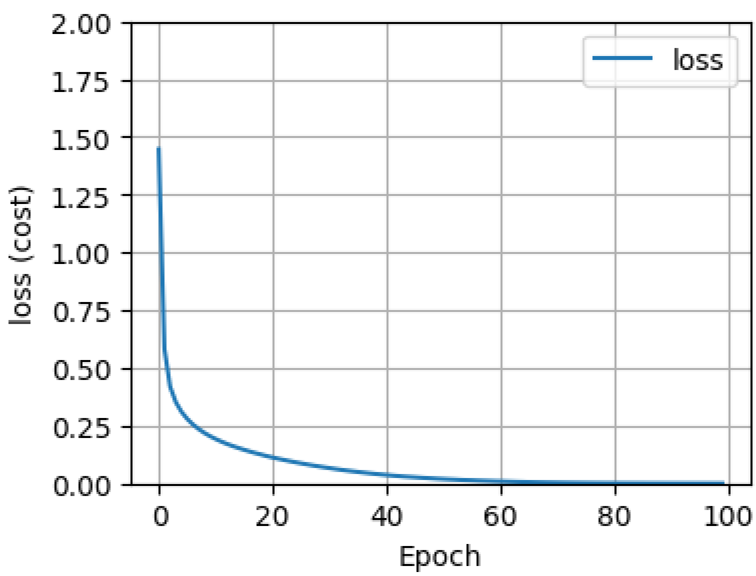
Loss (cost)

In the class, we learned to track the progress of gradient descent by monitoring the cost. Ideally, the cost will decrease as the number of iterations of the algorithm increases.

Tensorflow refers to the cost as `loss`. Above, you saw the loss displayed each epoch as `model.fit` was executing. The .fit method returns a variety of metrics including the loss. This is captured in the `history` variable above. This can be used to examine the loss in a plot as shown below.

In [83]:
```
def plot_loss_tf(history):
    fig,ax = plt.subplots(1,1, figsize = (4,3))
    ax.plot(history.history['loss'], label='loss')
    ax.set_ylim([0, 2])
    ax.set_xlabel('Epoch')
    ax.set_ylabel('loss (cost)')
    ax.legend()
    ax.grid(True)
    plt.show()
```

In [84]:
```
plot_loss_tf(history)
```



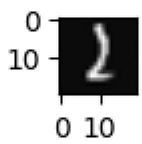
Prediction

To make a prediction, use Keras `predict`. Below, X[1015] contains an image of a two.

In [85]:
```
def display_digit(X):
    """ display a single digit. The input is one digit (400,). """
    fig, ax = plt.subplots(1,1, figsize=(0.5,0.5))
    X_reshaped = X.reshape((20,20)).T
    # Display the image
    ax.imshow(X_reshaped, cmap='gray')
    plt.show()
```

In [86]:
```
image_of_two = X[1015]
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)}")
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
 predicting a Two:
[[-10.7005       4.8022423   11.940644     0.25595793 -15.865273
   -5.9875355  -17.10808      3.203707    -9.729481   -14.672356  ]]
 Largest Prediction index: 2
```

The largest output is prediction[2], indicating the predicted digit is a '2'. If the problem only requires a selection, that is sufficient. Use NumPy argmax to select it. If the problem requires a probability, a softmax is required:

In [87]:
```
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}")
```

```
 predicting a Two. Probability vector:
[[1.4677708e-10 7.9325604e-04 9.9903798e-01 8.4137009e-06 8.3873614e-13
  1.6348279e-08 2.4203656e-13 1.6039037e-04 3.8758474e-10 2.7650529e-12]]
Total of predictions: 1.000
```

To return an integer representing the predicted target, you want the index of the largest probability. This is accomplished with the Numpy argmax function.

In [88]:
```
yhat = np.argmax(prediction_p)

print(f"np.argmax(prediction_p): {yhat}")
```

```
np.argmax(prediction_p): 2
```

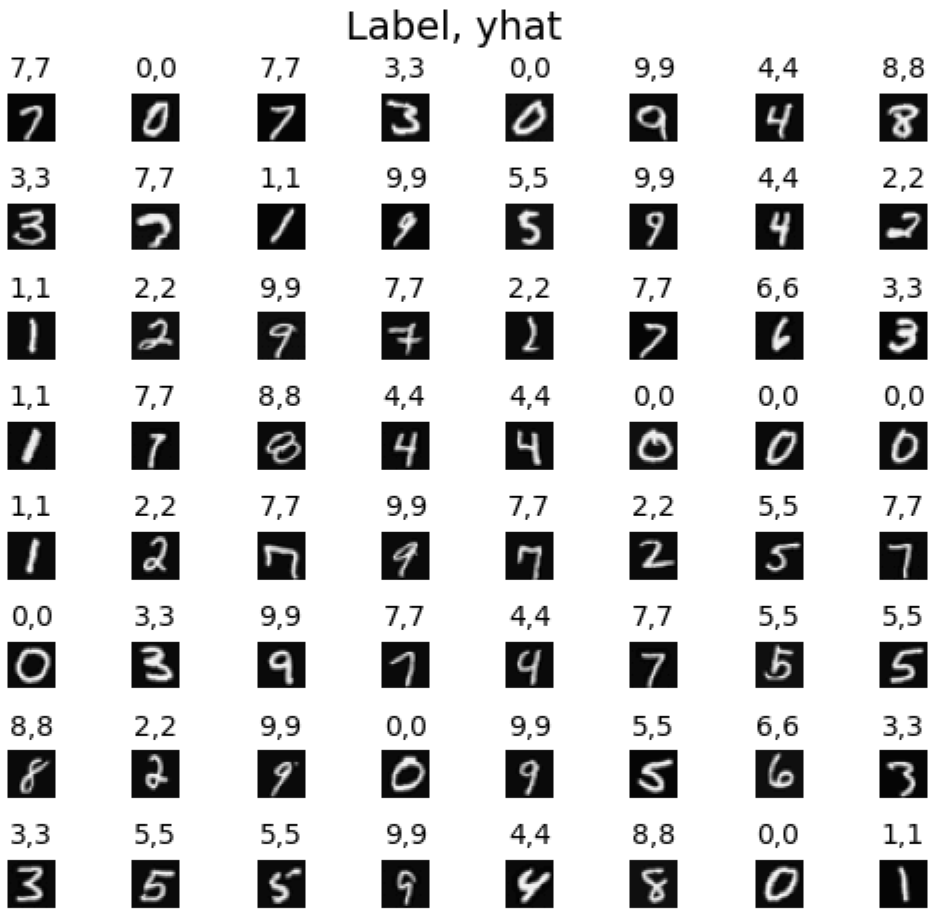
Let's compare the predictions vs the labels for a random sample of 64 digits. This takes a moment to run.

In [89]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

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]
for i,ax in enumerate(axes.flat):
    # Select random indices
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T
```

```
    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Predict using the Neural Network
    prediction = model.predict(X[random_index].reshape(1,400))
    prediction_p = tf.nn.softmax(prediction)
    yhat = np.argmax(prediction_p)

    # Display the label above the image
    ax.set_title(f"{y[random_index,0]},{yhat}",fontsize=10)
    ax.set_axis_off()
fig.suptitle("Label, yhat", fontsize=14)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```



In [90]:

```
def display_errors(model,X,y):
    f = model.predict(X)
    yhat = np.argmax(f, axis=1)
    doo = yhat != y[:,0]
    idxs = np.where(yhat != y[:,0])[0]
    if len(idxs) == 0:
        print("no errors found")
    else:
        cnt = min(8, len(idxs))
        fig, ax = plt.subplots(1,cnt, figsize=(5,1.2))
        fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.80]) #[left, bottom, right, t

        for i in range(cnt):
            j = idxs[i]
            X_reshaped = X[j].reshape((20,20)).T

            # Display the image
            ax[i].imshow(X_reshaped, cmap='gray')
```

```
            # Predict using the Neural Network
            prediction = model.predict(X[j].reshape(1,400))
            prediction_p = tf.nn.softmax(prediction)
            yhat = np.argmax(prediction_p)

            # Display the label above the image
            ax[i].set_title(f"{y[j,0]},{yhat}",fontsize=10)
            ax[i].set_axis_off()
            fig.suptitle("Label, yhat", fontsize=12)
    return(len(idxs))
```

In [91]:
```
print( f"{display_errors(model,X,y)} errors out of {len(X)} images")
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 0s 427us/step
no errors found
0 errors out of 5000 images
```

Exercise 3 [70 points]

Below you will build and train five additional models using variations of the provided base code. Each model will have at least one significant difference in architecture, hyperparameters, or training approach. After training, compare the models' performance on the test set to see how these changes affect accuracy and generalization.

In [100…
```
#reload the data
def load_data():
    X = np.load("data_ex2/X.npy")
    y = np.load("data_ex2/y.npy")
    return X, y
X,y = load_data()
print(X.shape)
print(y.shape)
```

```
(5000, 400)
(5000, 1)
```

In [101…
```
# Let's shuffle the data and get 80% and training set and 20% as testing set
# Suppose X is shape (5000, 400) and y is shape (5000, 1)
indices = np.arange(len(X))        # Create an array of indices [0..4999]
np.random.shuffle(indices)         # Shuffle the indices in-place

# Reorder X and y according to the shuffled indices
X_shuffled = X[indices]
y_shuffled = y[indices]

# Calculate the split point (80% = 0.8)
split_index = int(0.8 * len(X_shuffled))

# Create the train/test splits
X_train = X_shuffled[:split_index]
y_train = y_shuffled[:split_index]
X_test  = X_shuffled[split_index:]
y_test  = y_shuffled[split_index:]
```

3.1 Vary the Number of Hidden Layers [10 points]

Goal: Observe how adding or removing hidden layers impacts training time and performance.

In [102…

```
# Additional more hidden layers than the model you build in Exercise 2
model_1 = tf.keras.models.Sequential([
    tf.keras.Input(shape=(400,)),
    Dense(25, activation="relu", name="layer1"),
    Dense(20, activation="relu", name="layer2"),
    Dense(15, activation="relu", name="layer3"),
    Dense(10, activation="linear", name="outputLayer")
], name="model_1")


model_1.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

history_1 = model_1.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 819us/step - loss: 2.1063
Epoch 2/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 0.8958
Epoch 3/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 0.4941
Epoch 4/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - loss: 0.3945
Epoch 5/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 0.3363
Epoch 6/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 0.2958
Epoch 7/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 0.2644
Epoch 8/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 0.2388
Epoch 9/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 0.2159
Epoch 10/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 0.1965
Epoch 11/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - loss: 0.1800
Epoch 12/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.1648
Epoch 13/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 0.1502
Epoch 14/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - loss: 0.1370
Epoch 15/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.1250
Epoch 16/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.1141
Epoch 17/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - loss: 0.1052
Epoch 18/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 0.0966
Epoch 19/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 0.0886
Epoch 20/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - loss: 0.0812
Epoch 21/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0746
Epoch 22/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 659us/step - loss: 0.0685
Epoch 23/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 672us/step - loss: 0.0621
Epoch 24/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - loss: 0.0571
Epoch 25/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 0.0520
Epoch 26/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 0.0469
Epoch 27/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 0.0421
Epoch 28/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0380
```

```
Epoch 29/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 0.0344
Epoch 30/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 0.0315
Epoch 31/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 0.0285
Epoch 32/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 0.0260
Epoch 33/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.0236
Epoch 34/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - loss: 0.0210
Epoch 35/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step - loss: 0.0187
Epoch 36/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0164
Epoch 37/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 667us/step - loss: 0.0147
Epoch 38/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 0.0128
Epoch 39/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.0114
Epoch 40/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 0.0102
Epoch 41/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 0.0092
Epoch 42/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0084
Epoch 43/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0076
Epoch 44/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0069
Epoch 45/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 0.0061
Epoch 46/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.0055
Epoch 47/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 0.0048
Epoch 48/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 699us/step - loss: 0.0043
Epoch 49/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0039
Epoch 50/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 0.0035
Epoch 51/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0032
Epoch 52/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 0.0028
Epoch 53/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 770us/step - loss: 0.0026
Epoch 54/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 0.0023
Epoch 55/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 712us/step - loss: 0.0021
Epoch 56/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 0.0020
```

```
Epoch 57/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0018
Epoch 58/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.0016
Epoch 59/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 0.0015
Epoch 60/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 654us/step - loss: 0.0014
Epoch 61/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 683us/step - loss: 0.0013
Epoch 62/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0012
Epoch 63/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 0.0011
Epoch 64/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0010
Epoch 65/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 9.4388e-04
Epoch 66/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 8.7042e-04
Epoch 67/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 8.1192e-04
Epoch 68/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - loss: 7.3383e-04
Epoch 69/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 6.6308e-04
Epoch 70/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 6.0407e-04
Epoch 71/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - loss: 5.4447e-04
Epoch 72/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 4.9416e-04
Epoch 73/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - loss: 4.5059e-04
Epoch 74/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 671us/step - loss: 4.1019e-04
Epoch 75/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 3.7560e-04
Epoch 76/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 3.4192e-04
Epoch 77/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 3.1485e-04
Epoch 78/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 2.8608e-04
Epoch 79/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.6393e-04
Epoch 80/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 2.4066e-04
Epoch 81/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 2.2302e-04
Epoch 82/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 658us/step - loss: 2.0296e-04
Epoch 83/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - loss: 1.8676e-04
Epoch 84/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 1.7146e-04
```

```
Epoch 85/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 1.5767e-04
Epoch 86/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 1.4473e-04
Epoch 87/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 1.3267e-04
Epoch 88/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 1.2164e-04
Epoch 89/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 1.1171e-04
Epoch 90/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 1.0231e-04
Epoch 91/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 9.4033e-05
Epoch 92/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 8.6150e-05
Epoch 93/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 668us/step - loss: 7.8868e-05
Epoch 94/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 7.2622e-05
Epoch 95/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 6.7402e-05
Epoch 96/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 6.1799e-05
Epoch 97/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 5.6874e-05
Epoch 98/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 5.2449e-05
Epoch 99/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 4.8388e-05
Epoch 100/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 671us/step - loss: 4.4711e-05
```

3.2 Increase the Number of Units Per Layer [10 points]

Goal: See if a larger model can learn more complex patterns and whether it might overfit.

In [103…

```python
#add more units(neurons) in each hidden layer on the model that you build in Exerci
model_2 = tf.keras.models.Sequential([
    tf.keras.Input(shape=(400,)),
    Dense(40, activation="relu", name="layer1"),
    Dense(35, activation="relu", name="layer2"),
    Dense(30, activation="relu", name="layer3"),
    Dense(25, activation="linear", name="outputLayer")
], name="model_2")

model_2.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

history_2 = model_2.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 766us/step - loss: 2.4363
Epoch 2/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 691us/step - loss: 0.6464
Epoch 3/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.3984
Epoch 4/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 0.3078
Epoch 5/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 0.2551
Epoch 6/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 688us/step - loss: 0.2169
Epoch 7/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 0.1872
Epoch 8/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 0.1630
Epoch 9/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1427
Epoch 10/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 0.1259
Epoch 11/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 0.1114
Epoch 12/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 0.0991
Epoch 13/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 0.0882
Epoch 14/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 655us/step - loss: 0.0789
Epoch 15/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.0692
Epoch 16/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.0611
Epoch 17/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.0537
Epoch 18/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 679us/step - loss: 0.0461
Epoch 19/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 670us/step - loss: 0.0406
Epoch 20/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 0.0354
Epoch 21/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 0.0309
Epoch 22/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0268
Epoch 23/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 0.0238
Epoch 24/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 0.0202
Epoch 25/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 0.0183
Epoch 26/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 0.0151
Epoch 27/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 0.0140
Epoch 28/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0116
```

```
Epoch 29/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 0.0105
Epoch 30/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 0.0093
Epoch 31/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 998us/step - loss: 0.0185
Epoch 32/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 944us/step - loss: 0.0322
Epoch 33/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 0.0219
Epoch 34/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0156
Epoch 35/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0101
Epoch 36/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 919us/step - loss: 0.0055
Epoch 37/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0043
Epoch 38/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0033
Epoch 39/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 0.0027
Epoch 40/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 0.0023
Epoch 41/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 0.0020
Epoch 42/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 0.0017
Epoch 43/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 0.0016
Epoch 44/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 0.0014
Epoch 45/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 721us/step - loss: 0.0013
Epoch 46/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.0012
Epoch 47/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 0.0011
Epoch 48/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 9.6596e-04
Epoch 49/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 946us/step - loss: 8.8892e-04
Epoch 50/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 8.1777e-04
Epoch 51/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 7.6508e-04
Epoch 52/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 6.9759e-04
Epoch 53/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 6.6232e-04
Epoch 54/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - loss: 6.0403e-04
Epoch 55/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 5.5460e-04
Epoch 56/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 5.0853e-04
```

```
Epoch 57/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 4.7131e-04
Epoch 58/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 4.3629e-04
Epoch 59/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 4.0394e-04
Epoch 60/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 680us/step - loss: 3.7233e-04
Epoch 61/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 3.4307e-04
Epoch 62/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 3.1689e-04
Epoch 63/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 2.9115e-04
Epoch 64/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 2.7094e-04
Epoch 65/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 2.4967e-04
Epoch 66/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 696us/step - loss: 2.3214e-04
Epoch 67/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 2.1330e-04
Epoch 68/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 1.9785e-04
Epoch 69/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 1.8300e-04
Epoch 70/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 1.6931e-04
Epoch 71/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 1.5624e-04
Epoch 72/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - loss: 1.4497e-04
Epoch 73/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 1.3419e-04
Epoch 74/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 1.2403e-04
Epoch 75/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 704us/step - loss: 1.1551e-04
Epoch 76/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 1.0742e-04
Epoch 77/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 9.9378e-05
Epoch 78/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 704us/step - loss: 9.2404e-05
Epoch 79/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 675us/step - loss: 8.5665e-05
Epoch 80/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - loss: 7.9846e-05
Epoch 81/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 7.3936e-05
Epoch 82/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - loss: 6.8524e-05
Epoch 83/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 6.3748e-05
Epoch 84/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 5.9163e-05
```

```
Epoch 85/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 5.5244e-05
Epoch 86/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 5.1106e-05
Epoch 87/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 4.7468e-05
Epoch 88/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 4.4122e-05
Epoch 89/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 4.0982e-05
Epoch 90/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 3.7961e-05
Epoch 91/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - loss: 3.5521e-05
Epoch 92/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 684us/step - loss: 3.2872e-05
Epoch 93/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 3.0542e-05
Epoch 94/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 2.8361e-05
Epoch 95/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 2.6399e-05
Epoch 96/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 2.4482e-05
Epoch 97/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 2.2770e-05
Epoch 98/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 2.1162e-05
Epoch 99/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 1.9679e-05
Epoch 100/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 1.8268e-05
```

3.3 Adjust the Learning Rate [10 points]

Goal: Understand how the learning rate influences convergence speed and final accuracy.

In [105…

```python
model_3 = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer((400,)),
    tf.keras.layers.Dense(25, activation="relu", name="L1"),
    tf.keras.layers.Dense(15, activation="relu", name="L2"),
    tf.keras.layers.Dense(10, activation="linear", name="Output")
], name="model_3")

# Lower learning rate to 0.00001
model_3.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
)

history_3 = model_3.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 2.3414
Epoch 2/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 2.3253
Epoch 3/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - loss: 2.3114
Epoch 4/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 2.2990
Epoch 5/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 2.2876
Epoch 6/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 2.2768
Epoch 7/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 2.2665
Epoch 8/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 2.2561
Epoch 9/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - loss: 2.2457
Epoch 10/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 2.2354
Epoch 11/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.2250
Epoch 12/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 2.2141
Epoch 13/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 677us/step - loss: 2.2026
Epoch 14/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 2.1905
Epoch 15/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 2.1776
Epoch 16/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 644us/step - loss: 2.1637
Epoch 17/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 2.1489
Epoch 18/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - loss: 2.1332
Epoch 19/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 2.1166
Epoch 20/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - loss: 2.0992
Epoch 21/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 2.0812
Epoch 22/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 2.0628
Epoch 23/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.0441
Epoch 24/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.0254
Epoch 25/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - loss: 2.0066
Epoch 26/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 1.9878
Epoch 27/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 1.9690
Epoch 28/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 1.9502
```

```
Epoch 29/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 691us/step - loss: 1.9313
Epoch 30/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 1.9123
Epoch 31/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - loss: 1.8933
Epoch 32/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 687us/step - loss: 1.8743
Epoch 33/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 1.8552
Epoch 34/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 685us/step - loss: 1.8361
Epoch 35/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 1.8170
Epoch 36/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - loss: 1.7979
Epoch 37/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 1.7790
Epoch 38/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 1.7603
Epoch 39/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - loss: 1.7419
Epoch 40/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.7237
Epoch 41/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.7057
Epoch 42/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - loss: 1.6880
Epoch 43/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 949us/step - loss: 1.6705
Epoch 44/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.6533
Epoch 45/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 1.6363
Epoch 46/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - loss: 1.6195
Epoch 47/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 1.6028
Epoch 48/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 919us/step - loss: 1.5864
Epoch 49/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.5702
Epoch 50/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 969us/step - loss: 1.5541
Epoch 51/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 1.5382
Epoch 52/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 1.5225
Epoch 53/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1.5070
Epoch 54/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 679us/step - loss: 1.4917
Epoch 55/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 1.4765
Epoch 56/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 1.4615
```

```
Epoch 57/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 1.4467
Epoch 58/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 1.4321
Epoch 59/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 1.4176
Epoch 60/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 1.4032
Epoch 61/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.3890
Epoch 62/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 1.3750
Epoch 63/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 1.3612
Epoch 64/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 1.3475
Epoch 65/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 1.3340
Epoch 66/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 1.3206
Epoch 67/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 692us/step - loss: 1.3075
Epoch 68/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 674us/step - loss: 1.2945
Epoch 69/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 1.2817
Epoch 70/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 948us/step - loss: 1.2691
Epoch 71/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 1.2566
Epoch 72/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.2443
Epoch 73/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 1.2322
Epoch 74/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 1.2203
Epoch 75/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 1.2085
Epoch 76/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 1.1969
Epoch 77/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - loss: 1.1854
Epoch 78/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.1741
Epoch 79/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.1630
Epoch 80/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - loss: 1.1520
Epoch 81/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 1.1413
Epoch 82/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 1.1307
Epoch 83/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 1.1203
Epoch 84/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 1.1100
```

```
Epoch 85/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 1.0999
Epoch 86/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 1.0900
Epoch 87/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 1.0803
Epoch 88/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 1.0707
Epoch 89/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 1.0614
Epoch 90/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 1.0522
Epoch 91/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 706us/step - loss: 1.0431
Epoch 92/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 1.0342
Epoch 93/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 1.0254
Epoch 94/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 1.0168
Epoch 95/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 1.0083
Epoch 96/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 1.0000
Epoch 97/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.9918
Epoch 98/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 0.9838
Epoch 99/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 0.9759
Epoch 100/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 0.9681
```

3.4 Use a Different Number of Epochs [10 points]

Goal: Check if training for fewer or more epochs changes the performance significantly (e.g., underfitting vs. overfitting).

In [107…

```
model_4 = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer((400,)),
    tf.keras.layers.Dense(25, activation="relu", name="L1"),
    tf.keras.layers.Dense(15, activation="relu", name="L2"),
    tf.keras.layers.Dense(10, activation="linear", name="Output")
], name="model_4")

model_4.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

# Train with fewer epochs (e.g., 10)
history_4 = model_4.fit(X_train, y_train, epochs=200)
```

```
Epoch 1/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 2.0571
Epoch 2/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 0.8658
Epoch 3/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.4903
Epoch 4/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.3805
Epoch 5/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 0.3227
Epoch 6/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - loss: 0.2848
Epoch 7/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 0.2563
Epoch 8/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 0.2337
Epoch 9/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 0.2146
Epoch 10/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 0.1984
Epoch 11/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.1836
Epoch 12/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.1706
Epoch 13/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 0.1592
Epoch 14/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.1487
Epoch 15/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 635us/step - loss: 0.1390
Epoch 16/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 0.1304
Epoch 17/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 0.1217
Epoch 18/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - loss: 0.1142
Epoch 19/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.1067
Epoch 20/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 0.1001
Epoch 21/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.0942
Epoch 22/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 634us/step - loss: 0.0882
Epoch 23/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.0830
Epoch 24/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0779
Epoch 25/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0732
Epoch 26/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 899us/step - loss: 0.0687
Epoch 27/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.0644
Epoch 28/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 0.0605
```

```
Epoch 29/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 0.0569
Epoch 30/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - loss: 0.0532
Epoch 31/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 609us/step - loss: 0.0497
Epoch 32/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 627us/step - loss: 0.0468
Epoch 33/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.0435
Epoch 34/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 622us/step - loss: 0.0407
Epoch 35/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - loss: 0.0382
Epoch 36/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 0.0354
Epoch 37/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - loss: 0.0332
Epoch 38/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step - loss: 0.0309
Epoch 39/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.0286
Epoch 40/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 642us/step - loss: 0.0268
Epoch 41/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - loss: 0.0250
Epoch 42/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 0.0231
Epoch 43/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 0.0216
Epoch 44/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - loss: 0.0201
Epoch 45/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 623us/step - loss: 0.0185
Epoch 46/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step - loss: 0.0172
Epoch 47/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.0160
Epoch 48/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 630us/step - loss: 0.0149
Epoch 49/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - loss: 0.0137
Epoch 50/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.0128
Epoch 51/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 0.0118
Epoch 52/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 0.0109
Epoch 53/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 624us/step - loss: 0.0102
Epoch 54/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 0.0094
Epoch 55/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 0.0086
Epoch 56/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - loss: 0.0080
```

```
Epoch 57/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 618us/step - loss: 0.0075
Epoch 58/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 0.0069
Epoch 59/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 0.0063
Epoch 60/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.0059
Epoch 61/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0054
Epoch 62/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 0.0050
Epoch 63/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - loss: 0.0047
Epoch 64/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 0.0043
Epoch 65/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 0.0040
Epoch 66/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 0.0037
Epoch 67/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 0.0034
Epoch 68/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 0.0032
Epoch 69/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 0.0029
Epoch 70/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0027
Epoch 71/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 0.0025
Epoch 72/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0023
Epoch 73/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 0.0022
Epoch 74/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0020
Epoch 75/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 0.0019
Epoch 76/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 613us/step - loss: 0.0018
Epoch 77/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 856us/step - loss: 0.0016
Epoch 78/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 0.0015
Epoch 79/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 0.0014
Epoch 80/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 628us/step - loss: 0.0013
Epoch 81/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 0.0012
Epoch 82/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 0.0012
Epoch 83/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 0.0011
Epoch 84/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 9.9315e-04
```

```
Epoch 85/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 681us/step - loss: 9.2254e-04
Epoch 86/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 8.5683e-04
Epoch 87/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 8.0538e-04
Epoch 88/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 7.4283e-04
Epoch 89/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 679us/step - loss: 6.8875e-04
Epoch 90/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 6.4689e-04
Epoch 91/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 5.9832e-04
Epoch 92/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 5.5866e-04
Epoch 93/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 5.2113e-04
Epoch 94/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 4.8410e-04
Epoch 95/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 666us/step - loss: 4.4854e-04
Epoch 96/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 4.1960e-04
Epoch 97/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - loss: 3.8808e-04
Epoch 98/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 956us/step - loss: 3.5956e-04
Epoch 99/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 3.3973e-04
Epoch 100/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 3.1491e-04
Epoch 101/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 621us/step - loss: 2.9150e-04
Epoch 102/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - loss: 2.7346e-04
Epoch 103/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 601us/step - loss: 2.5360e-04
Epoch 104/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 593us/step - loss: 2.3591e-04
Epoch 105/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 605us/step - loss: 2.1858e-04
Epoch 106/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 618us/step - loss: 2.0383e-04
Epoch 107/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 1.9069e-04
Epoch 108/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 936us/step - loss: 1.7640e-04
Epoch 109/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 664us/step - loss: 1.6534e-04
Epoch 110/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 1.5288e-04
Epoch 111/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 1.4304e-04
Epoch 112/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 1.3262e-04
```

```
Epoch 113/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 1.2367e-04
Epoch 114/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 694us/step - loss: 1.1536e-04
Epoch 115/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 1.0704e-04
Epoch 116/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 9.9598e-05
Epoch 117/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 899us/step - loss: 9.3086e-05
Epoch 118/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 8.6558e-05
Epoch 119/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 756us/step - loss: 8.0670e-05
Epoch 120/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 689us/step - loss: 7.5054e-05
Epoch 121/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 7.0051e-05
Epoch 122/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 617us/step - loss: 6.5232e-05
Epoch 123/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 6.0867e-05
Epoch 124/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 674us/step - loss: 5.6677e-05
Epoch 125/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 5.2884e-05
Epoch 126/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 4.9472e-05
Epoch 127/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 4.6100e-05
Epoch 128/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 4.2908e-05
Epoch 129/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 4.0009e-05
Epoch 130/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 691us/step - loss: 3.7459e-05
Epoch 131/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 686us/step - loss: 3.4810e-05
Epoch 132/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 3.2514e-05
Epoch 133/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 3.0384e-05
Epoch 134/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 2.8196e-05
Epoch 135/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 2.6328e-05
Epoch 136/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 2.4693e-05
Epoch 137/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 2.3030e-05
Epoch 138/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 2.1383e-05
Epoch 139/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 2.0047e-05
Epoch 140/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step - loss: 1.8650e-05
```

```
Epoch 141/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 899us/step - loss: 1.7455e-05
Epoch 142/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 672us/step - loss: 1.6290e-05
Epoch 143/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 1.5298e-05
Epoch 144/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 1.4165e-05
Epoch 145/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 669us/step - loss: 1.3299e-05
Epoch 146/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 1.2450e-05
Epoch 147/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - loss: 1.1603e-05
Epoch 148/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 1.0840e-05
Epoch 149/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.0097e-05
Epoch 150/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 9.4796e-06
Epoch 151/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 638us/step - loss: 8.8504e-06
Epoch 152/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 8.2492e-06
Epoch 153/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 7.7518e-06
Epoch 154/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 7.2346e-06
Epoch 155/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 684us/step - loss: 6.7376e-06
Epoch 156/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 6.3424e-06
Epoch 157/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 702us/step - loss: 5.8903e-06
Epoch 158/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 5.5239e-06
Epoch 159/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 665us/step - loss: 5.1622e-06
Epoch 160/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 4.8388e-06
Epoch 161/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 4.5327e-06
Epoch 162/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 4.2231e-06
Epoch 163/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 653us/step - loss: 3.9494e-06
Epoch 164/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 3.6938e-06
Epoch 165/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - loss: 3.4555e-06
Epoch 166/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - loss: 3.2264e-06
Epoch 167/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 953us/step - loss: 3.0306e-06
Epoch 168/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.8183e-06
```

```
Epoch 169/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 2.6494e-06
Epoch 170/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 2.4725e-06
Epoch 171/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 2.3070e-06
Epoch 172/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 2.1661e-06
Epoch 173/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 924us/step - loss: 2.0248e-06
Epoch 174/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 649us/step - loss: 1.8859e-06
Epoch 175/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - loss: 1.7741e-06
Epoch 176/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 633us/step - loss: 1.6546e-06
Epoch 177/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 1.5450e-06
Epoch 178/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 1.4490e-06
Epoch 179/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 992us/step - loss: 1.3490e-06
Epoch 180/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 1.2664e-06
Epoch 181/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 687us/step - loss: 1.1863e-06
Epoch 182/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 661us/step - loss: 1.1073e-06
Epoch 183/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - loss: 1.0406e-06
Epoch 184/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 883us/step - loss: 9.7201e-07
Epoch 185/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 9.0502e-07
Epoch 186/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step - loss: 8.5533e-07
Epoch 187/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 637us/step - loss: 7.9578e-07
Epoch 188/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 657us/step - loss: 7.4332e-07
Epoch 189/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 641us/step - loss: 6.9858e-07
Epoch 190/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 6.5116e-07
Epoch 191/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 698us/step - loss: 6.0951e-07
Epoch 192/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 645us/step - loss: 5.7092e-07
Epoch 193/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 678us/step - loss: 5.3498e-07
Epoch 194/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - loss: 4.9764e-07
Epoch 195/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 4.6848e-07
Epoch 196/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 4.3891e-07
```

```
Epoch 197/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 919us/step - loss: 4.1113e-07
Epoch 198/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 3.8551e-07
Epoch 199/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 3.6138e-07
Epoch 200/200
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 919us/step - loss: 3.3897e-07
```

3.5 Train on a Subset of the Data [10 points]

Goal: Explore how dataset size impacts model training and performance (e.g., does the model overfit more easily on less data?).

In [108…

```
subset_indices = np.random.choice(len(X_train), size=int(0.3 * len(X_train)), repla
X_small = X_train[subset_indices]
y_small = y_train[subset_indices]
```

In [109…

```
model_5 = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer((400,)),
    tf.keras.layers.Dense(25, activation="relu", name="L1"),
    tf.keras.layers.Dense(15, activation="relu", name="L2"),
    tf.keras.layers.Dense(10, activation="linear", name="Output")
], name="model_5")

model_5.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

history_5 = model_5.fit(X_small, y_small, epochs=100)
```

```
Epoch 1/100
38/38  0s 906us/step - loss: 2.2045
Epoch 2/100
38/38  0s 892us/step - loss: 1.7623
Epoch 3/100
38/38  0s 839us/step - loss: 1.3070
Epoch 4/100
38/38  0s 905us/step - loss: 0.9795
Epoch 5/100
38/38  0s 852us/step - loss: 0.7679
Epoch 6/100
38/38  0s 797us/step - loss: 0.6306
Epoch 7/100
38/38  0s 824us/step - loss: 0.5360
Epoch 8/100
38/38  0s 811us/step - loss: 0.4678
Epoch 9/100
38/38  0s 878us/step - loss: 0.4169
Epoch 10/100
38/38  0s 811us/step - loss: 0.3770
Epoch 11/100
38/38  0s 865us/step - loss: 0.3447
Epoch 12/100
38/38  0s 851us/step - loss: 0.3172
Epoch 13/100
38/38  0s 865us/step - loss: 0.2938
Epoch 14/100
38/38  0s 838us/step - loss: 0.2731
Epoch 15/100
38/38  0s 824us/step - loss: 0.2549
Epoch 16/100
38/38  0s 906us/step - loss: 0.2391
Epoch 17/100
38/38  0s 1ms/step - loss: 0.2244
Epoch 18/100
38/38  0s 1ms/step - loss: 0.2111
Epoch 19/100
38/38  0s 825us/step - loss: 0.1994
Epoch 20/100
38/38  0s 798us/step - loss: 0.1880
Epoch 21/100
38/38  0s 797us/step - loss: 0.1780
Epoch 22/100
38/38  0s 814us/step - loss: 0.1684
Epoch 23/100
38/38  0s 824us/step - loss: 0.1593
Epoch 24/100
38/38  0s 824us/step - loss: 0.1513
Epoch 25/100
38/38  0s 832us/step - loss: 0.1430
Epoch 26/100
38/38  0s 851us/step - loss: 0.1351
Epoch 27/100
38/38  0s 784us/step - loss: 0.1273
Epoch 28/100
38/38  0s 798us/step - loss: 0.1203
```

```
Epoch 29/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 0.1135
Epoch 30/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.1073
Epoch 31/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - loss: 0.1010
Epoch 32/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 0.0949
Epoch 33/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 0.0897
Epoch 34/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 0.0841
Epoch 35/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 0.0794
Epoch 36/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0745
Epoch 37/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0699
Epoch 38/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 906us/step - loss: 0.0658
Epoch 39/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 959us/step - loss: 0.0619
Epoch 40/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0581
Epoch 41/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 905us/step - loss: 0.0549
Epoch 42/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 933us/step - loss: 0.0518
Epoch 43/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0488
Epoch 44/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 879us/step - loss: 0.0460
Epoch 45/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0433
Epoch 46/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0408
Epoch 47/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 892us/step - loss: 0.0387
Epoch 48/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0365
Epoch 49/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 0.0344
Epoch 50/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0325
Epoch 51/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 892us/step - loss: 0.0307
Epoch 52/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - loss: 0.0290
Epoch 53/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 904us/step - loss: 0.0273
Epoch 54/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.0259
Epoch 55/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0246
Epoch 56/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0233
```

```
Epoch 57/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 0.0220
Epoch 58/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 0.0209
Epoch 59/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 0.0198
Epoch 60/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0188
Epoch 61/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 960us/step - loss: 0.0179
Epoch 62/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 959us/step - loss: 0.0171
Epoch 63/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 987us/step - loss: 0.0162
Epoch 64/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0154
Epoch 65/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 973us/step - loss: 0.0146
Epoch 66/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 0.0140
Epoch 67/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0133
Epoch 68/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 0.0126
Epoch 69/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 0.0120
Epoch 70/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0114
Epoch 71/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0109
Epoch 72/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0104
Epoch 73/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0099
Epoch 74/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0095
Epoch 75/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 905us/step - loss: 0.0091
Epoch 76/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 905us/step - loss: 0.0087
Epoch 77/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0083
Epoch 78/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0078
Epoch 79/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 0.0074
Epoch 80/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0071
Epoch 81/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0068
Epoch 82/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0065
Epoch 83/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 0.0062
Epoch 84/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 0.0059
```

```
Epoch 85/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 0.0056
Epoch 86/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 0.0054
Epoch 87/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0052
Epoch 88/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0049
Epoch 89/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0048
Epoch 90/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0046
Epoch 91/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.0044
Epoch 92/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 0.0042
Epoch 93/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.0040
Epoch 94/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0039
Epoch 95/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 959us/step - loss: 0.0037
Epoch 96/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 906us/step - loss: 0.0036
Epoch 97/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - loss: 0.0035
Epoch 98/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 0.0033
Epoch 99/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0032
Epoch 100/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 0s 865us/step - loss: 0.0031
```

In [110…

```python
# Evaluate Model 1
logits_1 = model_1.predict(X_test)
predicted_classes_1 = np.argmax(logits_1, axis=1)
true_classes_1 = y_test.reshape(-1)       # Ensure y_test is 1D
accuracy_1 = np.mean(predicted_classes_1 == true_classes_1)
print(f"Model 1 test accuracy: {accuracy_1:.4f}")

# Evaluate Model 2
logits_2 = model_2.predict(X_test)
predicted_classes_2 = np.argmax(logits_2, axis=1)
true_classes_2 = y_test.reshape(-1)
accuracy_2 = np.mean(predicted_classes_2 == true_classes_2)
print(f"Model 2 test accuracy: {accuracy_2:.4f}")

# Evaluate Model 3
logits_3 = model_3.predict(X_test)
predicted_classes_3 = np.argmax(logits_3, axis=1)
true_classes_3 = y_test.reshape(-1)
accuracy_3 = np.mean(predicted_classes_3 == true_classes_3)
print(f"Model 3 test accuracy: {accuracy_3:.4f}")

# Evaluate Model 4
logits_4 = model_4.predict(X_test)
```

```
predicted_classes_4 = np.argmax(logits_4, axis=1)
true_classes_4 = y_test.reshape(-1)
accuracy_4 = np.mean(predicted_classes_4 == true_classes_4)
print(f"Model 4 test accuracy: {accuracy_4:.4f}")

# Evaluate Model 5
logits_5 = model_5.predict(X_test)
predicted_classes_5 = np.argmax(logits_5, axis=1)
true_classes_5 = y_test.reshape(-1)
accuracy_5 = np.mean(predicted_classes_5 == true_classes_5)
print(f"Model 5 test accuracy: {accuracy_5:.4f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Model 1 test accuracy: 0.9180
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Model 2 test accuracy: 0.9360
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Model 3 test accuracy: 0.7450
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Model 4 test accuracy: 0.9170
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Model 5 test accuracy: 0.8930
```

Question: Though these five models are different in some ways, but their performance might be still similar. Why? [10 points]

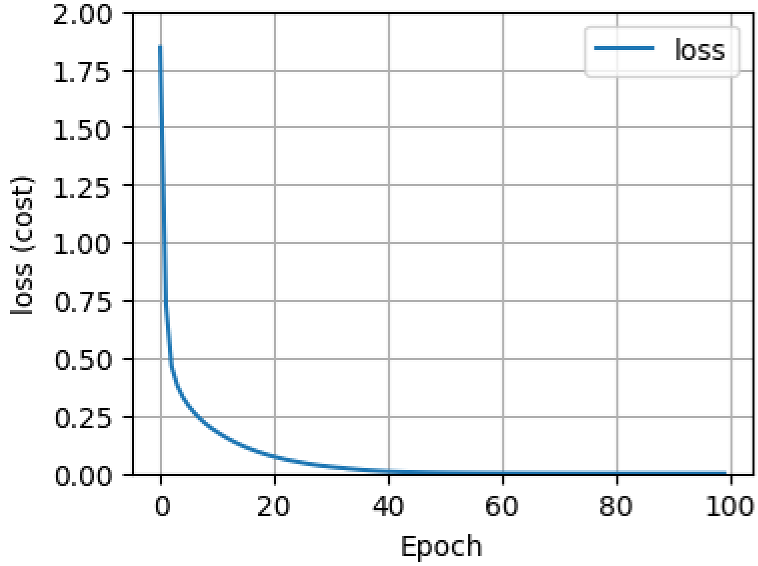
The reasons are: the epochs remain relatively the same alongside the learning rate. In model 3 the learning rate is tiny compared to the epochs allowed so the accuracy is much lower, we will need more steps to increase the accuracy. Aside from that more feature engineering is required to increase the accuracy.

In [111…

```
def plot_loss_tf(history):
    fig,ax = plt.subplots(1,1, figsize = (4,3))
    ax.plot(history.history['loss'], label='loss')
    ax.set_ylim([0, 2])
    ax.set_xlabel('Epoch')
    ax.set_ylabel('loss (cost)')
    ax.legend()
    ax.grid(True)
    plt.show()
```

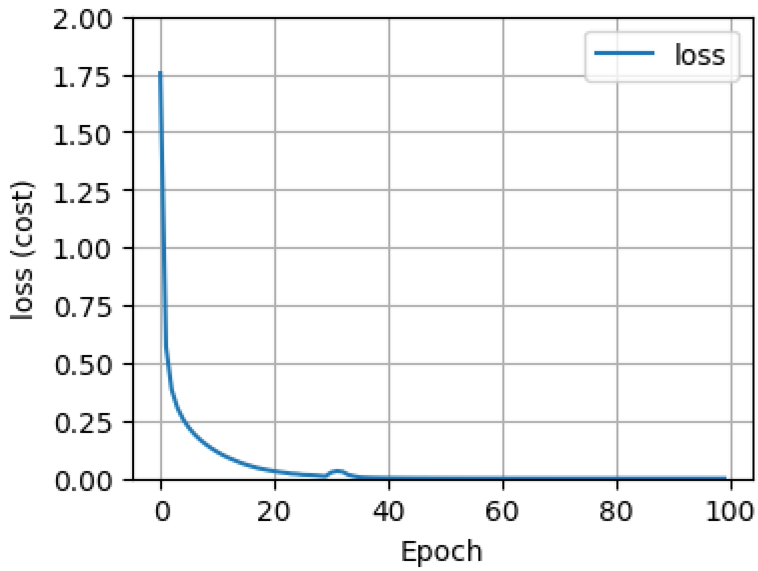
In [112…

```
plot_loss_tf(history_1) # Vary the Number of Hidden Layers
```



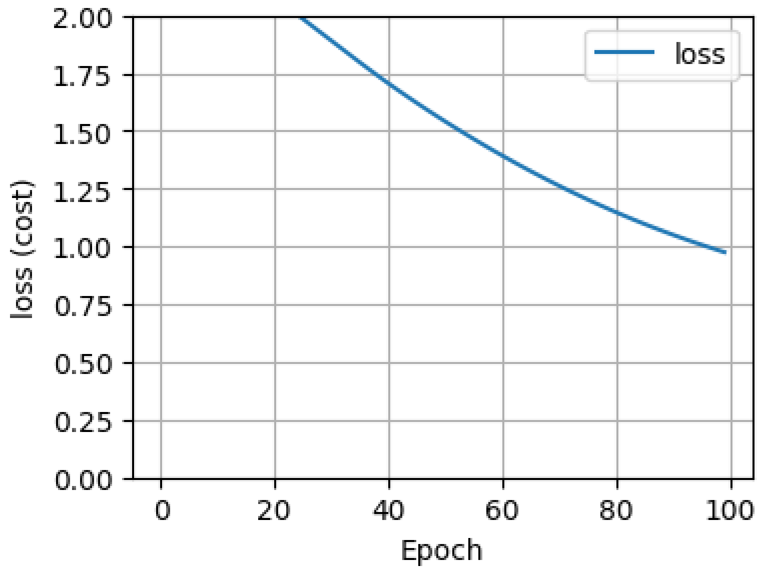
In [113…

```
plot_loss_tf(history_2) #Increase the Number of Units Per Layer
```

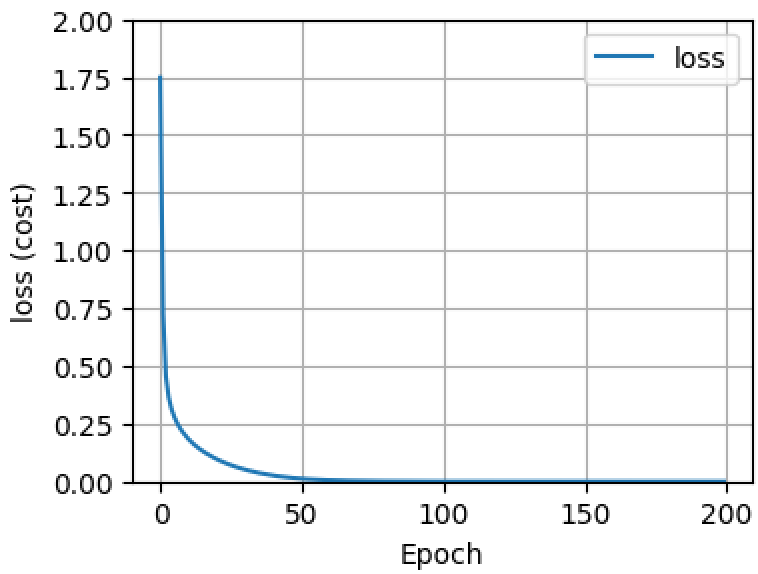


In [114…

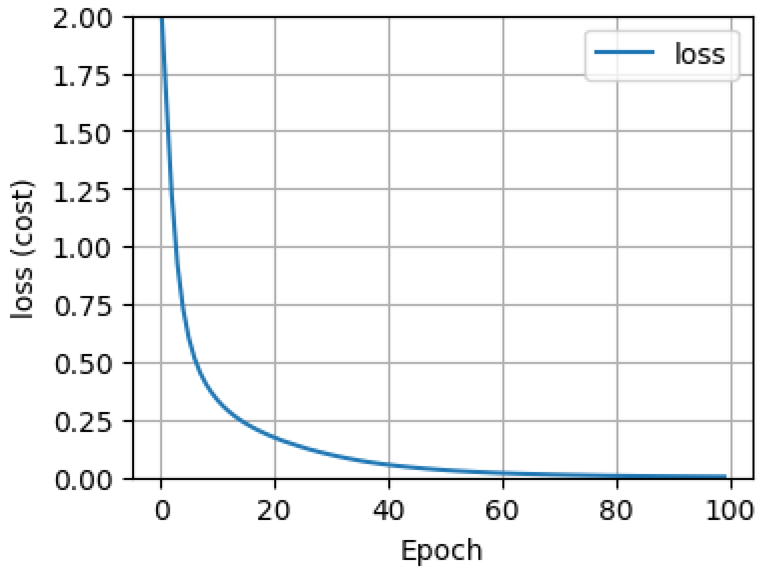
```
plot_loss_tf(history_3) #Adjust the Learning Rate
```



In [115…
```
plot_loss_tf(history_4) #Use a Different Number of Epochs
```



In [116…
```
plot_loss_tf(history_5) #Train on a Subset of the Data
```



Question: What you can find from the loss curves of these five models [10 points]

Your answers go below: The learning rate takes a huge toll on the loss curve. An effective learning rate is critical to the model.

In [ ]: