实验四 神经网络-多分类

一、实验目的

1. 掌握本地Jupyter notebook编译环境；
2. 熟悉Python编程集成环境Pychram、vs code；
3. 掌握基于numpy与Tensorflow库的神经网络模型搭建。

本实验代码于吴恩达老师Coursera中的机器学习专项课程。

二、实验内容

1. 安装并配置本地Jupyter notebook编译环境；
2. 基于numpy与Tensorflow库的神经网络多分类模型搭建。

三、实验结果

1. 完成本实验中的Exercise 1和2, 直到显示All tests passed!;

四、实验心得(500字以内)

实验心得：通过本次实验，我利用 Keras Sequential 模型和带有 ReLU 激活的 Dense Layer 构建了三层神经网络，用来实现手写体识别数字0-9。这三层网络的激活函数分别为'relu','relu','linear'。前两层采用relu函数不用sigmoid函数是因为sigmoid函数涉及e的乘方运算，计算量非常大，影响模型的效率。最后一层用linear函数不用softmax函数是因为softmax函数中也存在e的开放运算，由于float型变量存储位数有限，计算e的开方时可能存在溢出，结果存在累计误差，故采用linear函数。选择好损失函数和梯度下降策略后，进行40次epoch，就得到了0-9手写体识别的模型，然而在测试集上的测试结果其正确率并不是很理想。对比上一次实验简单的0-1识别只用了20次epoch，我猜测如果增加epoch模型应当更好。于是改换50次epoch重新测试，测试集全部通过。当然了，是不是过拟合，是不是一次偶然，希望在以后的学习中慢慢验证。

Practice Lab: Neural Networks for Handwritten Digit Recognition, Multiclass

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

Outline

1 - Packages

First, let's run the cell below to import all the packages that you will need during this assignment.

- numpy (https://numpy.org/) is the fundamental package for scientific computing with Python.
- matplotlib (http://matplotlib.org) is a popular library to plot graphs in Python.
- tensorflow (https://www.tensorflow.org/) a popular platform for machine learning.

In [182]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import linear, relu, sigmoid
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('./deeplearning.mplstyle')

import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)

from public_tests import *

from autils import *
from lab_utils_softmax import plt_softmax
np.set_printoptions(precision=2)
```

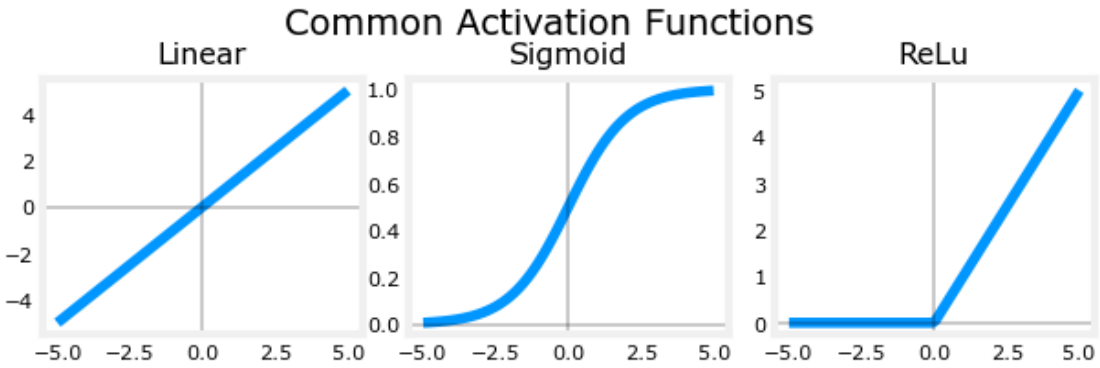
2 - ReLU Activation

This week, a new activation was introduced, the Rectified Linear Unit (ReLU).

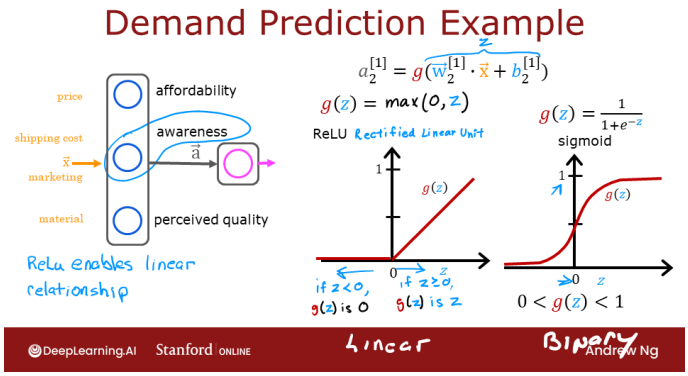
$$a = max(0, z) \qquad \text{\# ReLU function}$$

In [ ]:
```

```

In [183]:
```
plt_act_trio()
```

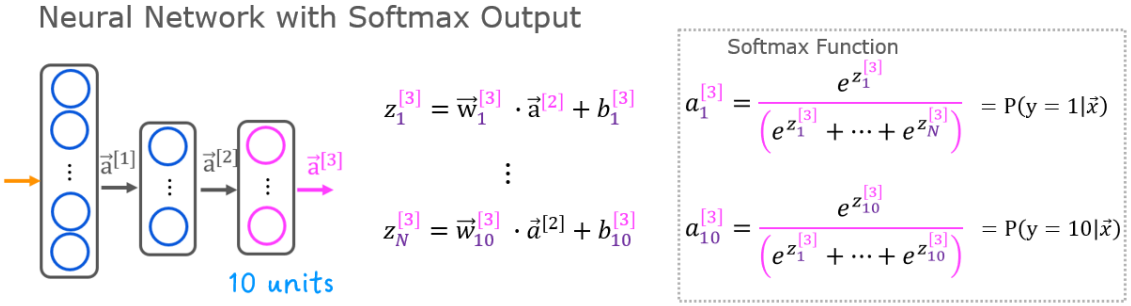


The example from the lecture on the right shows an application of the ReLU. In this example, the derived "awareness" feature is not binary but has a continuous range of values. The sigmoid is best for on/off or binary situations. The ReLU provides a continuous linear relationship. Additionally it has an 'off' range where the output is zero. The "off" feature makes the ReLU a Non-Linear activation. Why is this needed? This enables multiple units to contribute to to the resulting function without interfering. This is examined more in the supporting optional lab.



3 - 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 1

Let's create a NumPy implementation:

In [184]:
```
# UNQ_C1
# GRADED CELL: my_softmax

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 ###
    e_z=np.exp(z)
    total=np.sum(e_z)
    a=e_z/total
    ### END CODE HERE ###
    return a
```

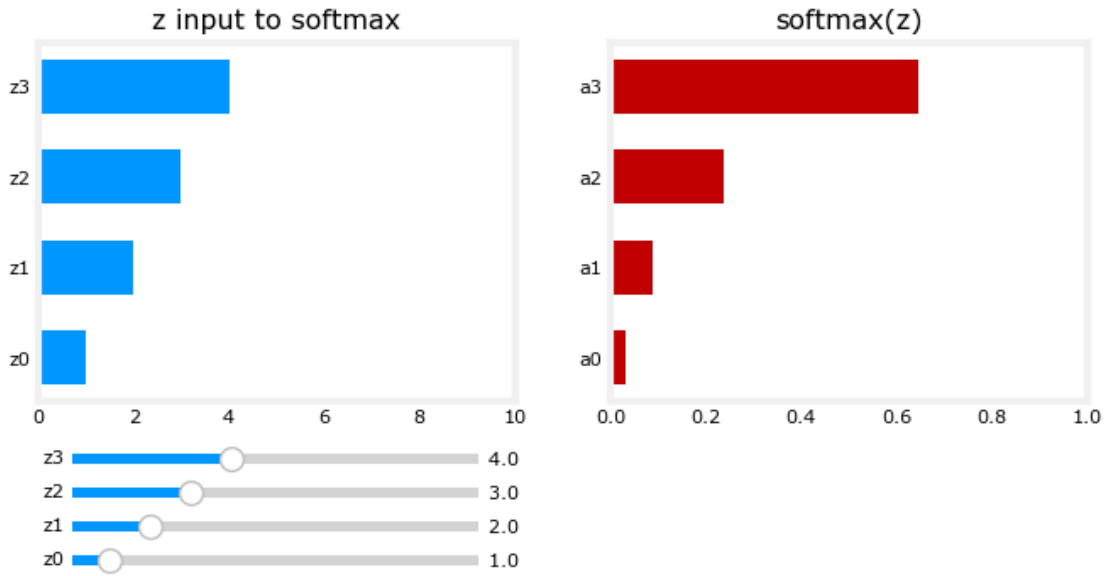
In [185]:
```
z = np.array([1., 2., 3., 4.])
a = my_softmax(z)
atf = tf.nn.softmax(z)
print(f"my_softmax(z):          {a}")
print(f"tensorflow softmax(z): {atf}")

# BEGIN UNIT TEST
test_my_softmax(my_softmax)
# END UNIT TEST
```

```
my_softmax(z):          [0.03 0.09 0.24 0.64]
tensorflow softmax(z): [0.03 0.09 0.24 0.64]
 All tests passed.
```

Click for hints

Below, vary the values of the `z` inputs. Note in particular how the exponential in the numerator magnifies small differences in the values. Note as well that the output values sum to one.

In [186]:
```
plt.close("all")
plt_softmax(my_softmax)
```



4 - Neural Networks

In last weeks assignment, you implemented a neural network to do binary classification. This week you will extend that to multiclass classification. This will utilize the softmax activation.

4.1 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.

4.2 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.

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

- 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/ (http://yann.lecun.com/exdb/mnist/))

In [187]:
```
# load dataset
X, y = load_data()
```

4.2.1 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 the first element in the variables `X` and `y`.

In [188]:
```
print ('The first element of X is: ', X[0])
```

```
The first element of X is:  [ 0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+
00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  8.56e-06  1.94e-06 -7.37e-04
 -8.13e-03 -1.86e-02 -1.87e-02 -1.88e-02 -1.91e-02 -1.64e-02 -3.78e-03
  3.30e-04  1.28e-05  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  0.00e+00  0.00e+00  1.16e-04  1.20e-04 -1.40e-02 -2.85e-02  8.04e-02
  2.67e-01  2.74e-01  2.79e-01  2.74e-01  2.25e-01  2.78e-02 -7.06e-03
  2.35e-04  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00  0.00e+00
  1.28e-17 -3.26e-04 -1.39e-02  8.16e-02  3.83e-01  8.58e-01  1.00e+00
  9.70e-01  9.31e-01  1.00e+00  9.64e-01  4.49e-01 -5.60e-03 -3.78e-03
  0.00e+00  0.00e+00  0.00e+00  0.00e+00  5.11e-06  4.36e-04 -3.96e-03
```

In [189]:
```
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
```

4.2.2 Check the dimensions of your variables

Another way to get familiar with your data is to view its dimensions. Please print the shape of `X` and `y` and see how many training examples you have in your dataset.

In [190]:
```
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)
```

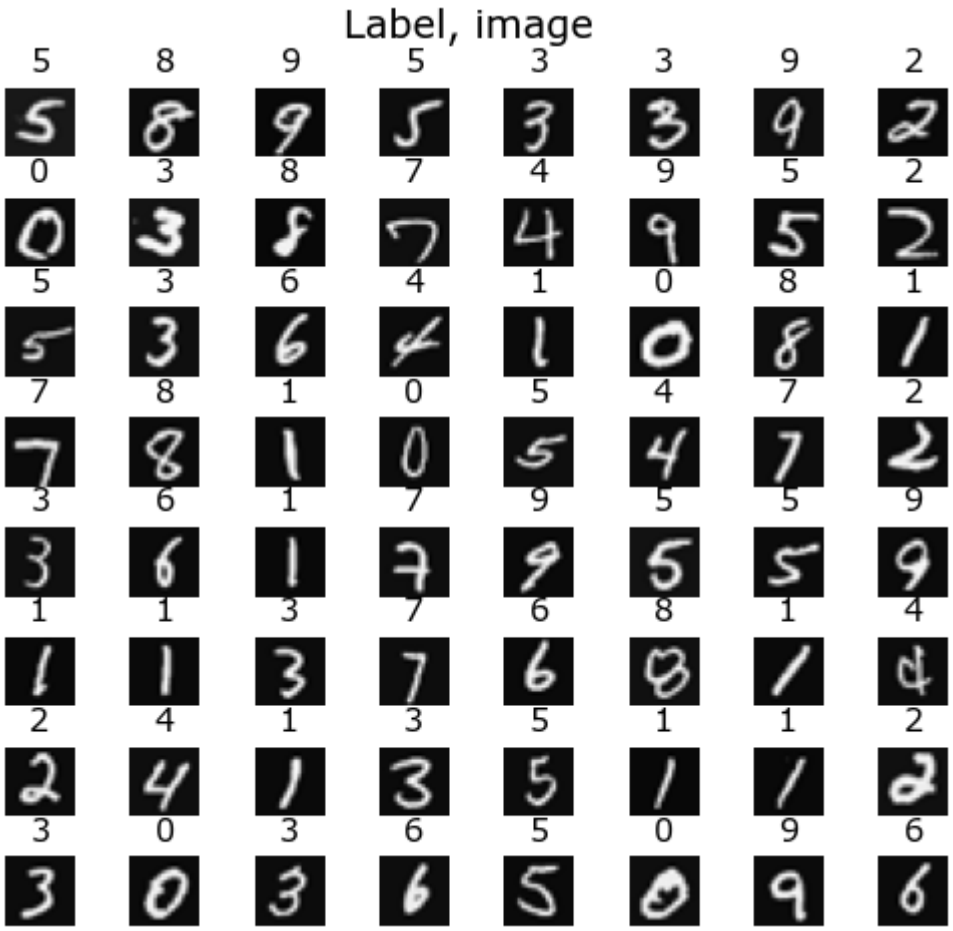
4.2.3 Visualizing the Data

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 [191]:

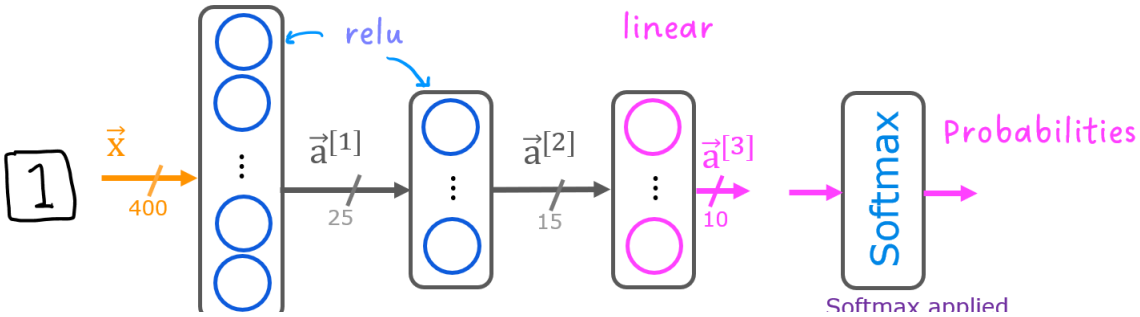
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

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]

#fig.tight_layout(pad=0.5)
widgvis(fig)
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)
```



4.3 Model representation

The neural network you will use in this assignment is shown in the figure below.

- This has two dense layers with ReLU activations followed by an output layer with a linear activation.
  - Recall that our inputs are pixel values of digit images.
  - 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.
  - Recall that the dimensions of these parameters is determined as follows:
    - If network has s_{in} units in a layer and s_{out} units in the next layer, then
      - W will be of dimension $s_{in} \times s_{out}$.
      - b will be a vector with s_{out} elements
  - Therefore, the shapes of `W` , and `b` , are
    - layer1: The shape of `W1` is (400, 25) and the shape of `b1` is (25,)
    - layer2: The shape of `W2` is (25, 15) and the shape of `b2` is: (15,)
    - layer3: The shape of `W3` is (15, 10) and the shape of `b3` is: (10,)

> **Note:** The bias vector `b` could be represented as a 1-D (n,) or 2-D (n,1) array. Tensorflow utilizes a 1-D representation and this lab will maintain that convention:

4.4 Tensorflow Model Implementation

Tensorflow models are built layer by layer. A layer's input dimensions (s_{in} above) are calculated for you. You specify a layer's *output dimensions* and this determines the next layer's input dimension. The input dimension of the first layer is derived from the size of the input data specified in the `model.fit` statement below.

> **Note:** It is also possible to add an input layer that specifies the input dimension of the first layer. For example:
> `tf.keras.Input(shape=(400,)),  #specify input shape`
> We will include that here to illuminate some model sizing.

4.5 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

Exercise 2

Below, using Keras Sequential model (https://keras.io/guides/sequential_model/) and Dense Layer (https://keras.io/api/layers/core_layers/dense/) with a ReLU activation to construct the three layer network described above.

In [192]:
```
# UNQ_C2
# GRADED CELL: Sequential model
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='layer3')
        ### END CODE HERE ###
    ], name = "my_model"
)
```

In [193]:
```
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 25)                10025

 layer2 (Dense)              (None, 15)                390

 layer3 (Dense)              (None, 10)                160

=================================================================
Total params: 10575 (41.31 KB)
Trainable params: 10575 (41.31 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Expected Output (Click to expand)

In [194]:
```
# BEGIN UNIT TEST
test_model(model, 10, 400)
# END UNIT TEST
```

```
All tests passed!
```

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

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

In [195]:
```
[layer1, layer2, layer3] = model.layers
```

In [196]:
```
#### 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,)
```

Expected Output

```
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 [197]:
```
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=50
)
```

```
Epoch 1/50
157/157 [==============================] - 0s 980us/step - loss: 1.4973
Epoch 2/50
157/157 [==============================] - 0s 886us/step - loss: 0.6038
Epoch 3/50
157/157 [==============================] - 0s 910us/step - loss: 0.4151
Epoch 4/50
157/157 [==============================] - 0s 890us/step - loss: 0.3298
Epoch 5/50
157/157 [==============================] - 0s 895us/step - loss: 0.2806
Epoch 6/50
157/157 [==============================] - 0s 923us/step - loss: 0.2453
Epoch 7/50
157/157 [==============================] - 0s 888us/step - loss: 0.2175
Epoch 8/50
157/157 [==============================] - 0s 865us/step - loss: 0.1960
Epoch 9/50
157/157 [==============================] - 0s 853us/step - loss: 0.1830
Epoch 10/50
```

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:

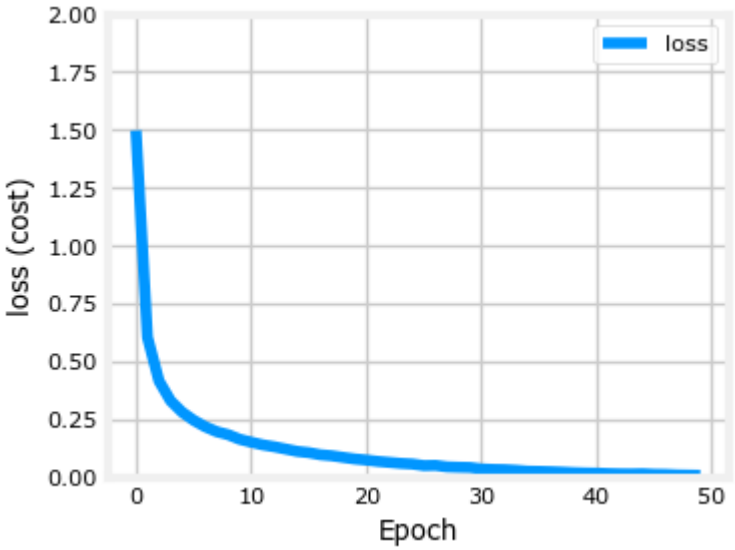
```
Epoch 1/100
157/157 [==============================] - 0s 1ms/step - loss: 2.2770
```

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 course 1, 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 (https://www.tensorflow.org/api_docs/python/tf/keras/Model) 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.

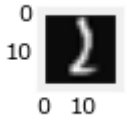
In [198]:
```
plot_loss_tf(history)
```



Prediction

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

In [199]:
```
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 37ms/step
 predicting a Two:
[[ -7.04   3.76  12.26   7.94 -13.29  -5.39  -9.93   8.02   0.29  -4.86]]
 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 (https://numpy.org/doc/stable/reference/generated/numpy.argmax.html) to select it. If the problem requires a probability, a softmax is required:

In [200]:
```
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:
[[4.04e-09 1.98e-04 9.73e-01 1.30e-02 7.84e-12 2.11e-08 2.24e-10 1.41e-02
  6.19e-06 3.59e-08]]
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 (https://numpy.org/doc/stable/reference/generated/numpy.argmax.html) function.

In [201]:
```
yhat = np.argmax(prediction_p)

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

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

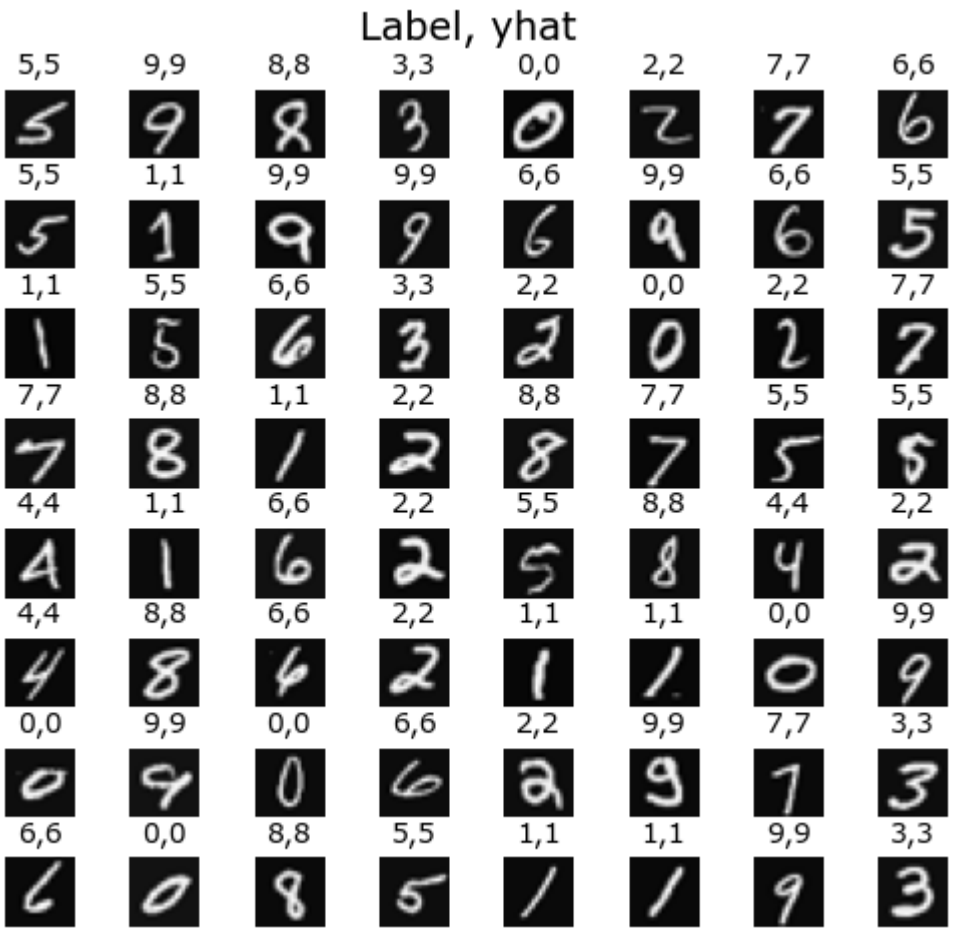
Let's compare the predictions vs the labels for a random sample of 64 digits. This takes a moment to run.

In [202]:

```
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]
widgvis(fig)
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 16ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 13ms/step
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
```



Let's look at some of the errors.

> Note: increasing the number of training epochs can eliminate the errors on this data set.

In [203]:
```
print( f"{display_errors(model,X,y)} errors out of {len(X)} images")
```

```
157/157 [==============================] - 0s 846us/step
no errors found
0 errors out of 5000 images
```

Congratulations!

You have successfully built and utilized a neural network to do multiclass classification.

In [ ]:
```
```