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
- 2 - ReLU Activation
- 3 - Softmax Function
  - Exercise 1
- 4 - Neural Networks
  - 4.1 Problem Statement
  - 4.2 Dataset
  - 4.3 Model representation
  - 4.4 Tensorflow Model Implementation
  - 4.5 Softmax placement
    - Exercise 2

NOTE: To preview errors from the autograder, you can not advance to edit or delete non-graded cells in this notebook. *Please ensure hitten Kernel->Restart and Run all cells. Once you have passed this assignment** and want to experiment with any of the non-graded code, you may follow the instructions at the bottom of this notebook.

1 - Packages

Many, with a lot of the software is import at the packages that you will need during this assignment.

- numpy is the fundamental package for scientific computing with Python.
- matplotlib is a popular library to plot graphs in Python.
- tensorflow a popular platform for machine learning.

```
In [1]: # UNCOMMENT TO RUN
        %pip install -q -U tensorflow==2.6
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Input
        from tensorflow.keras.activations import linear, relu, sigmoid
        %matplotlib widget
        import matplotlib.pyplot as plt
        plt.rcParams['axes.formatter.offset_mode'] = 'sci'

        import logging
        import warnings, os, sys, contextlib, os.setenv(logging.ERROR)
        if os.getenv("DS_ENV") in None:
            os.setenv("DS_ENV","local")

        from public_tests import *

        from utilsv2 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 [2]: plt.ioff()
        fig,ax = plt.subplots(1,1,figsize=(5,3))
        x = np.arange(-10,10,1)
        y = np.zeros(len(x))
        ax.plot(x,y,lw=1)
        ax.fill_between(x[x<=0], y[x<=0], y[x<=0], color='orange', alpha=0.5)
        ax.fill_between(x[x>=0], y[x>=0], y[x>=0], color='lightblue', alpha=0.5)
        ax.set_ylim(-.5,3)
        ax.set_xlabel('z')
        ax.set_ylabel('a')
        ax.set_title('Common Activation Functions')
        ax.legend(['linear', 'sigmoid', 'ReLU'])
        plt.show()
```

The example from the lecture on the right shows an application of the ReLU. In this example, the output of the function is a linear function of the input for positive inputs. Inputs of zero or negative gives a zero output. This is used for on/off or binary situations. The ReLU provides a continuous and linear switching. Additionally it has an "off" range where the output is zero. The "off" feature makes the ReLU a non-linear activation. Why is this non-linearity needed? It allows the model to create complex non-linear functions. Including function without flattening. This is examined more in the supporting optional lab.

3 - Softmax Function

Ordinarilyas neural networks generalize it outputs. Over output is selected as the predicted answer. In the output layer, a vector is generated by a linear function which is fed into a softmax function. The softmax function converts a 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 features/categories in the output layer.

Exercise 1

Let's create a softmaxfy implementation.

```
In [3]: # 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
            '''
            eZ = np.exp(z) # vector
            A = eZ/sum(eZ) # broadcast
            return(A)


            ### END CODE HERE ###
            return a

        z = np.array([1., 2., 3., 4.])
        a = my_softmax(z)
        atf = tf.nn.softmax(z)
        print("softmax(z) = {a}")
        print("Tensorflow softmax(z) = {atf}")

        # UNIT TESTS
        test_softmax(my_softmax, atf)
        # CHECK THIS
        np.sum(a)
        np.sum(atf)
        ### END TESTS ###
```

```
softmax(z) = [0.03 0.08 0.23 0.63]
Tensorflow softmax(z) = [0.03 0.08 0.23 0.63]
All tests passed.
```

Click for hints

Below, vary the values of the 4 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 [4]: plt.close("all")
        plt_softmax(my_softmax)
```

4 - Neural Networks

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

4.1 Problem Statement

In this exercise, you will use a neural network to recognize hand-written digits. 0-9. This is a multiclass classification task where one of n choices is selected. Automatic 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.
  - Each training example is a 20-pixel x 20-pixel pixel 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.

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

- 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 = 1 if the image is of the digit 1, and so on.

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

```
In [5]: # 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 [6]: 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 8.56e-74 2.26e-03 2.26e-03
 7.10e-74 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 1.28e-02 2.21e-01 2.36e-01 2.36e-01
 2.36e-01 2.36e-01 2.36e-01 2.36e-01 2.36
```

In [29]:
```
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:
[[1.42e-04 4.49e-02 8.98e-01 3.76e-02 3.61e-06 5.97e-06 3.03e-05 1.44e-02
  5.03e-03 3.22e-04]]
Total of predictions: 1.000
```

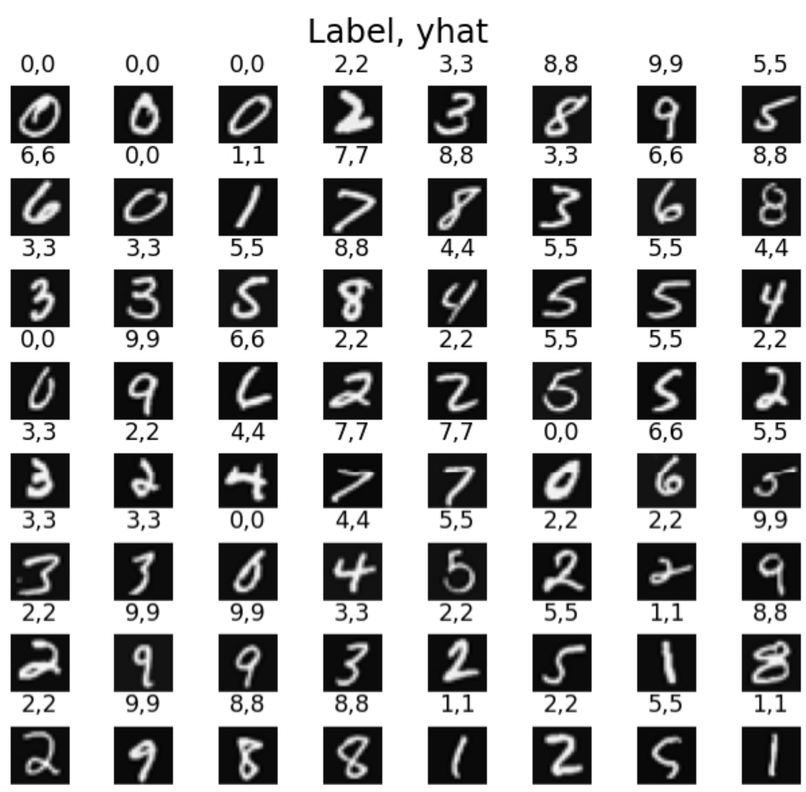
To return an integer representing the predicted target, you want the index of the largest probability. This is accomplished with the Numpy argmax function.

In [30]:
```
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.

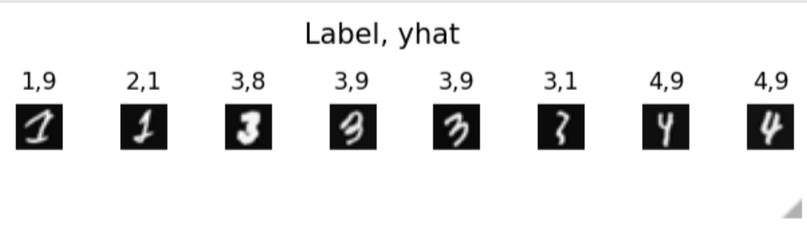
In [31]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

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()
```



Let's look at some of the errors.

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

In [32]:
```
print( f"{display_errors(model,X,y)} errors out of {len(X)} images")
```




```
15 errors out of 5000 images
```

Congratulations!

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

Please click here if you want to experiment with any of the non-graded code.