Chapter 16

1. **binary logistic regression** – output layer where each neuron can represent one of two classes
2. **sigmoid activation function** – returns a value in the range of 0 for negative infinity, through 0.5 for the input of 0, and to 1 for positive infinity
3. **softmax** – prediction of which class the network "thinks" the input represents
4. **binary cross-entropy** – Rather than only calculating negative log on the target class, we will sum the log-likelihoods of the correct and incorrect classes for each neuron separately
5. **categorical cross-entropy** – used to compare a "ground-truth" probability ("targets") and some predicted distribution
6. **sample_losses = np.mean(sample_losses, axis=-1)**

```
a. outputs = np.array([[1, 2, 3], [2, 4, 6],
                       [0, 5, 10], [11, 12, 13], [5, 10, 15], [1, 5,
   9]])
   # since axis is negative one mean is done on last dimension of
   function
   print(np.mean(outputs, axis=-1))
   # output = [ 2.  4.  5. 12. 10.]
   print(np.mean(outputs, axis=1))
   # output = [ 2.  4.  5. 12. 10.]
   # negative sign is not actually required for 2d array

   outputs = np.array([[[1, 2, 3], [2, 4, 6]],
                       [[0, 5, 10], [11, 12, 13]],
                       [[5, 10, 15], [1, 5, 9]]])
   print(np.mean(outputs, axis=-1))
   # output is the mean of each array returned in the 2x3 shape
   # output = [[ 2.  4.] [ 5. 12.] [10.  5.]]
   print(np.mean(outputs, axis=1))
   # output is the mean of each index and position so the first
   array is (1+2)/2, (2+4)/2, (3+6)/2
   # output = [[ 1.5  3.   4.5] [ 5.5  8.5 11.5] [ 3.   7.5 12. ]]
```

7. **Loss** – prediction error of Neural Net
8. **Adam** – **Adaptive Momentum**, is currently the most widely-used optimizer

Chapter 17

1. **mean squared error –** you square the difference between the predicted and true values of single outputs
2. **mean absolute error** – you take the absolute difference between the predicted and true values in a single output and average those absolute values
3. **np.std(y) / 250**

```
a. y = 9, 2, 5, 4, 12
   # Standard Deviation = mean (abs(x - x.mean())²)
   # textbook code = SD / Accuracy
   print(np.std(y) / 250)
   # output = 0.014444376068214231
   # the result is the cushion allowance for regression outputs
```

4. **np.sign() –** if array value is greater than 0 it returns 1, if array value is less than 0 it returns -1, and if array value 0 it returns 0
    a. array1 = [1, 0, -13] output = [1, 0, -1]
5. **Glorot uniform** – the fraction that multiplies the draw from the uniform distribution depends on the number of inputs and the number of neurons and is not constant like in the sample program

Chapter 18

1. **def set(self, *, loss, optimizer) –** The use of the asterisk in the parameter definitions notes that the subsequent parameters (loss and optimizer in this case) are keyword arguments. Since they have no default value assigned, they are required keyword arguments, which means that they have to be passed by names and values, making code more legible
2. **hidden layer** – defined back in first unit basically a layer the network controls
3. **model inference** – use of a machine learning model to process live input data to produce an output