**Chapter 10**

1. **Stochastic Gradient Descent (SGD)** – subtracting a fraction of the gradient for each weight and bias parameter. Historically refers to an optimizer that fits a single sample at a time
2. **Batch Gradient Descent** – optimizer used to fit a whole dataset at once
3. **Mini-Batch Gradient Descent** – used to fit slices of a dataset
4. **Batches** – slices of data
5. **mini-batches** – slices of data in the context of Stochastic Gradient Descent
6. **optimizer.update\_params(layerToUpdate) –** python code to call optimizer to update weights and biases
7. **epoch** – each full pass through the training data is referred to by this name**.** our network will be wrapped in a loop that will preform a set number of epochs that will each improve the model a bit
8. **local minimum** – from back in the first section of the book a local lowest point on a function which is not necessary the lowest point of the entire function.
9. **learning rate (LR)** – the fraction used to adjust our weights and biases
10. **global minimum –** lowest point of the entire function
11. **learning rate decay** – start with a large learning rate and decrease it over time
12. **Decay Rate –** code that reduces learning rate over time
13. **1/t decaying or exponential decaying** – reduce the learning rate after every step
14. **Learning rate decay code** - starting\_learning\_rate = 1. learning\_rate\_decay = 0.1 step = 1 learning\_rate = starting\_learning\_rate \* \ (1. / (1 + learning\_rate\_decay \* step)) print(learning\_rate)=.909
15. **pre\_update\_params** – This method, if we have a decay rate other than 0, will update our self.current\_learning\_rate
16. **momentum** – creates a rolling average of gradients over some number of updates and uses this average with the unique gradient at each step
17. **AdaGrad -** short for adaptive gradient, institutes a per-parameter learning rate rather than a globally-shared rate
18. **Epsilon –** is a hyperparameter preventing division by 0
19. **Hyperparameter** – pre-training control knob setting
20. **RMSProp –** short for **Root Mean Square Propagation**. Similar to AdaGrad, RMSProp calculates an adaptive learning rate per parameter
21. **Adam** – short for **Adaptive Momentum**, is currently the most widely-used optimizer and is built atop RMSProp, with the momentum concept from SGD added back in

**Chapter 11**

1. **Overfitting** – when the networkmemorizes the data without any understanding of it. When seeing a training example the network can predict with high accuracy however the network will struggle when shown new examples
2. **Training data** – data the network is trained on and will see many times when tuning the network
3. **Testing data** – data the network has never seen before. If the network preforms well then the network is probably generalized however if it preforms poorly then the network is probably overfitting the data
4. **Out-of-sample** – different name for testing data
5. **IMPORTANT NOTE ON TESTING DATA** – when doing a model using time series testing data that is close in time to training data can result in a false negative when looking for overfitting

**Chapter 12**

1. **Validation data** – This data is not used during training but it is occosanly used to test a models performance and learning. It seems to be like test data however it may be seen by the network more than once just not enough time for the model to learn it
2. **Cross-validation** – if we have 5 chunks, we can call them chunks A, B, C, D, and E. We may first train on A, B, C, and D, then validate on E. We’ll then train on A, B, C, E, and validate on D, doing this until we’ve validated on each of the 5 sample groups

**Chapter 13**

1. **Preprocessing** – operations preformed on training data
2. **Data augmentation** – changing existing data to generate more data. For example in analyzing photos you could rotate or crop the photos to create more data points from existing data.

**Chapter 14**

1. **Regularization methods** – methods used to avoid generalization error
2. **L1** and **L2 regularization** – calculate a number added to the loss value to penalize the model for large weights and biases.
3. **Penalty** – name of the number that is added to the loss value to penalize the model
4. **Lambda** – dictates how much of an impact we want a regularization penalty to carry
5. **np.ones\_like(weights)** – an array with the same shape as weights filled with positive and negative ones
6. **np.abs** – returns the absolute value of each element in an array

**Chapter 15**

1. **dropout layer –** This type of layer disables some neurons, while the others pass through unchanged. The reason to do this is to prevent a network from becoming to dependent on any given neuron
2. **co-adoption** – happens when neurons depend on the output values of other neurons and do not learn the underlying function on their own
3. **noise** – outliers in the training data
4. **Bernoulli distribution** – distribution where we can get a value of 1 with a probability of p and value of 0 with a probability of q