Improving Deep Neural Network

Week 1:

1. Train/dev/test sets:
   a) Make sure that your test and dev sets come from the same distribution
   b) Not having a test set might be okay
   c) Dev set for cross validation
2. Bias/Variance:
   a) Problem with high bias – underfitting model
   b) Problem with high variance – overfitting model
3. Basic recipe for ML:
   a) High bias:
      i) Make bigger network (More hidden units or layer)
      ii) Train longer
   b) High variance:
      i) Need more data
      ii) Regularization
      iii) Find more appropriate neural model
4. Regularizing NN:
   a) L2 regularization:
      i) We wish to minimize cost function, for L2 we add component that penalize large weights, where lambda is regularization parameter Now, *lambda* is a parameter than can be tuned. Larger weight values will be more penalized if the value of *lambda* is large. Similarly, for a smaller value of *lambda*, the regularization effect is smaller.
   b) Dropout regularization:
      i) More technically, At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.
5. Normalizing inputs:
   a) The first is to subtract out or to zero out the mean
   b) The second step is to normalize the variances
6. Vanishing/Exploding Gradients:
   a) In a network of *n* hidden layers, *n* derivatives will be multiplied together. If the derivatives are large then the gradient will increase exponentially as we propagate down the model until they eventually explode, and this is what we call the problem of ***exploding gradient***.
   b) Alternatively, if the derivatives are small then the gradient will decrease exponentially as we propagate through the model until it eventually vanishes, and this is the ***vanishing gradient*** problem.
7. Gradient checking:
   a) Don't use in training only to debug
   b) If algorithms fails grad check look at components to try to identify bug
   c) Does not work with dropout

Week 2:

8. Gradient descent:
   a) **In Batch Gradient Descent**, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use

that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.

b) **Stochastic Gradient Descent -** Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step. We do the following steps in **one epoch** for SGD:
   i) Take an example
   ii) Feed it to Neural Network
   iii) Calculate it's gradient
   iv) Use the gradient we calculated in step 3 to update the weights
   v) Repeat steps 1–4 for all the examples in training dataset

c) **Mini batch -** Neither we use all the dataset all at once nor we use the single example at a time. We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants we saw. (The same steps like in SGD)

9. Choosing your mini-batch size:
   a) If small train set – use Batch Gradient Descent (<= 2000)
   b) Typical mini-batch size: 64, 128,256, 512, 1024
   c) Make sure your mini-batch fit in CPU/GPU memory
10. Gradient descent with momentum:
   a) Compute dW, db on current mini-batch
   b) Use Exponentially weighting averages:
      i) Vdw = Bvdw + (1-B)dW
      ii) Vdb = Bvdb + (1-B)db
      iii) W = W – avdw, b = b-avdb
      iv) B = 0.9
11. Adam optimization algorithm:
   a) Vdw = 0, Vdb = 0, Sdw = 0, Sdb = 0
   b) Compute dW and db using current mini batch
   c) Vdw = BVdw + (1-B)dW, Vdb = BVdb +(1-B)db
   d) Sdw = B2 Vdw + (1-B)dW**2, Sdb = B2 Vdw + (1-B)db**2
   e) V(correction) = Vdw/(1-B**t), Vdb = Vd b/(1-B**t)
   f) Sd = Sdw/(1-B2**t), Sdb = Sdb(1-B2**t)
   g) W = W - alpha*(Vdw/ sqrt(Swd)+ e), b = W – alpha(Vdb/sqrt(Sdb)+e)
12. Learning rate decay:
   a) 1 epoch = 1 pass through the date
   b) Learning rate = 1/ (1+ rate-decay * epoch-num) * learning-rate[0])

Week 3:

13. Tuning process:
   a) "Coarse to Fine" usually refers to the hyperparameter optimization of a neural network during which you would like to try out different combinations of the hyperparameters and evaluate the performance of the network.However, due to the large number of parameters AND the big range of their values, it is almost impossible to check all the available combinations. For that reason, you usually discretize the available value range of each parameter into a "coarse" grid of values (i.e. val = 5,6,7,8,9) to estimate the effect of increasing or decreasing the value of that parameter. After selecting the value that seems

most promising/meaningful (i.e. val = 6), you perform a "finer" search around it (i.e. val = 5.8, 5.9, 6.0, 6.1, 6.2) to optimize even further.

14. Normalizing activations in a network:
    a) Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.
    b) It reduces overfitting because it has a slight regularization effects. Similar to dropout, it adds some noise to each hidden layer's activations.
    c) To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.
15. Softmax regression - The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0,1] which is nice because we are able to avoid binary classification and accommodate as many classes or dimensions in our neural network model. This is why softmax is sometimes referred to as a multinomial logistic regression.
16. Writing and running programs in TensorFlow has the following steps:
    a) Create Tensors (variables) that are not yet executed/evaluated. Tf.Variable()
    b) Write operations between those Tensors. Tf.add()/ Tf.multiply
    c) Initialize your Tensors. Tf.global_variable_initializers()
    d) Create a Session. With tf.session() as session
    e) Run the Session. This will run the operations you'd written above. Session.run()

17. Tf.place_holder(tf.float32, name=) - A Tensor that may be used as a handle for feeding a value, but not evaluated directly.
18. tf.one_hot(labels, C, axis=0)
19. tf.nn.relu(Z) – relu function
20. tf.nn.sigmoid(Z) – sigmoid
21. tf.train.GradientDescentOptimizer(learning_rate = learning_rate).minimize(cost)
22. tf.train.AdamOptimizer(learning_rate = learning_rate).minimize(cost)
23.