CS 725 : Machine Learning Assignment 2

Arka Sadhu - 140070011

November 3, 2017

1 General Procedure Employed

The core algorithm is based on gradient descent which requires us to calculate the gradients with respect to the weights and biases. I have explained the general procedure for the same.

- Feed Forward:
  - Feed forward is pretty straightword with all the layers having relu activation. The input of the previous layer is passed on to the next layer. The last layer has a softmax which converts the raw inputs into respective probability distribution.
- Back Propagation Derivation
  - The last layer has a softmax. Let the output of the last layer be h^l and the output of the softmax be q. The true probability distribution (in terms of one-hot vector) is p. The cross-entropy loss is
  $$L = -\sum_i p_i log(q_i)$$
  . We find that
  $$\frac{dL}{dh_i^l} = q_i - p_i$$
  where i is the index of the node (the derivation follows from chain rule).
  - The main backprop algorithm is as follows: Suppose g is the gradient vector from the next layer which is being back propagated and h_p is the output of the previous layer. First we do
  $$g = g \cdot activation\_gradients$$
  where $\cdot$ denotes the hadamard product. The activation gradient depends on the input from the feed forward input values. Now we have
  $$\nabla_b L = g$$
  and
  $$\nabla_W = gh^T$$
  - The g vector is again updated as
  $$g = gW$$
  and is then passed onto the previous layer (back proped).
  - This allows for the updates of the gradients.

2 Data Normalization

The normalization parameters obtained were as follows:

- Mean (μ) = array([2.5004525 , 7.003275 , 2.49920375, 7.00594375, 2.4987175 , 6.9897175 , 2.49958375, 6.9983725 , 2.50062125, 7.00085125])
- Standard Deviation (σ) = array([1.11753738, 3.74127408, 1.11883281, 3.7422765 , 1.11922779, 3.74010886, 1.11795676, 3.74443018, 1.11844294, 3.74208872])

3 K-Fold Validation

Number of Epochs in all cases are kept constant and equal to 10. K-fold validation is done with k=5. The average accuracy is reported.

Parameters:

- Learning rate = [1e-2, 1e-3]
- Number of hidden layers = [2, 3]
- Number of nodes in hidden layer = 100
- Regularizer lambda = [1e-6, 1e-7]
- Batch Size = 200
- Epochs = 10, 20

The regularizer values are chosen based on experiments and it was found that a regularizer of higher order didn't give any good results and hence the range is restricted to the order of 6 and 7. Similarly for learning rate it was found that learning rate = 0.1, it lead to explosion of the weights and led to run time error at cases of random initialization. For learning rate below 0.001 it took a lot more iterations and hence the results are reproduced. The number of hidden layers was constrained to 2 and 3 so as not to increase the parameters too much which would in turn lead to a lot more iterations to converge and perhaps fall victim to the local minimas created in the process.

Table 1: Hyperparameter Adjustment Table

learning rate	Regularization	Hidden Layers	Accuracy(epoch=10)	Accuracy(epoch=20)
0.01	1e-6	3	0.7813	0.97108125
	1e-7	3	0.79523125	0.98913125
0.001	1e-6	3	0.5557	0.66523125
	1e-7	3	0.55566875	0.66495625
0.01	1e-6	2	0.7589625	0.9286125
	1e-7	2	0.76018125	0.92965
0.001	1e-6	2	0.55074375	0.569125
	1e-7	2	0.55075	0.56914375

The tuned hyperparameters are :

- Learning Rate = 1e-2
- Hidden Layers = 3
- Lambda Regularizer = 1e-7

4 Gradient Updates

- Stochastic Gradient Descent:
  - The updates are made using each data point. When we see one data point and then back propagate the gradients we immediately want to change the direction of weights to be updated.
  - This often takes a lot of time because of extra computation.
  - The step size to be taken is usually annealed. It can also be computed in an exact search method which will minimize the current loss.
  - The path to the optimal point is often traversed in a zig-zag fashion.
  - It is stochastic in the sense of picking up a random data point.
- Batch Gradient Descent:

- Ideally we would like to go in the direction of gradient descent which will lead to maximum decrease in loss.
  - To get this direction we should ideally calculate the gradient direction of each of the data points.
  - Unfortunately the computations involved become exponential and hence this method is never used in practice unless data size is fairly less.

- Mini-Batch Stochastic Gradient Descent:
  - The mini-batch approach is a mid-way between the two approaches above.
  - It considers a smaller number of points in batches. The idea behind this is that the batch of data will correspond to the expected direction of the best gradient descent.
  - It basically takes average of the different gradient of the batches and hopes to go in the expected direction.
  - The path to optimal point is a lot less zig-zagged compared to stochastic gradient descent.

- Momentum Based Methods:
  - An extremely big problem in training neural networks comes because of being stuck at local minima which is usually the case with vanilla gradient descent methods.
  - To circumvent this problem momentum based methods keeps track of a velocity term.
  - In some sense velocity term keeps track of all the previous gradient directions. Therefore if the gradient direction is pretty steep and suddenly a local minimum is encountered the velocity term will force the algorithm to go ahead even though it say the local minimum.
  - The term velocity is correlated with the usual physics law.
  - In essence it allows us to get out of shallow boundaries.

Observations in this Assignment:

- The vanilla stochastic gradient descent takes way too much time to approach to the local minima.
- Mini-Batch Stochastic gradient descent takes a much more uniform path but it still takes a lot of time to converge.
- Momentum Based Method gave the best result in all cases.
- The momentum based method adds an extra hyper parameter but it is not tuned and in general it is agreed that 0.9 is a good value.

5 Activation Functions

Different activation functions are used for different reasons. In this Assignment ReLu is predominantly used but the usage of all activation functions are detailed.

- Sigmoid:
  - It gives output in the range $[0, 1]$.
  - It is an approximation for the step function and has the advantage of being continuous and differentiable and therefore being able to Backprop efficiently.
  - Unfortunately it suffers from the problem of vanishing gradients when the input is not near 0 and hence the gradients almost disappear when going to initial layers.

- Tanh:
  - It is similar to sigmoid and shares all other properties except that the output is in the range $[-1, 1]$

  - Has similar advantages and disadvantages to sigmoid, with the extra advantage of being able to go into the negative domain as well.

- ReLu:
  - It is simply the function $max(0, x)$ so the output range is in the positive real numbers.
  - The gradients can only be 0 or 1 depending on the input.
  - It is empirically proven that ReLu is able to converge much faster than both sigmoid and tanh and is predominantly used.
  - It is an approximation for the piece wise linear curves in higher dimensions.

- Leaky ReLu:
  - It is the same as ReLu for positive inputs but for negative inputs instead of all zeros, it gives a small negative number instead.
  - This allows for non-zero gradient for negative inputs as well.
  - Shares the advantages of ReLu.