SoftMax for Multi-Class Classification

1. In the intersection point, the probabilities of the x belong to Class 1, Class 2 and Class 3 is equal:

$$p(c_1|x) = p(c_2|x) = p(c_3|x) = 1 / 3$$

2. The probabilities of the points along the red line that belongs to the two classes are equal. For instance, the points in the red line divides the class 1 and class 2:

$$p(c_1|x) = p(c_2|x)$$

   Subsequently, the probabilities of the points belonging to the third class decreases as the probability of the other two classes increases. In this example, $p(c_1|x)$, $p(c_2|x)$ will increase and $p(c_3|x)$ will decrease.

3. If we move far away from the intersection point and staying in the middle of one region, the probability for the points belonging to this region will increase and it will be larger than that of the remaining two regions i.e. decrease as the point moves. For example, if we stay in the region 1 (Class 1):

$$p(c_1|x) > p(c_2|x)$$
$$p(c_1|x) > p(c_3|x)$$

Error Back Propagation

1.

$$\frac{\partial E_n(w)}{\partial a_1^{(4)}}$$

It is mentioned that for the output layer $h(a)=a$, so the output function can written as:

$$E_n(w) = \frac{1}{2}(h(a_1)^4 - t_n)^2$$
$$= \frac{1}{2}(a_1^4 - t_n)^2$$

The ~~derivation~~ derivative of this output function becomes:

$$\frac{\partial E_n(w)}{\partial a_1^{(4)}} = 2 \times \frac{1}{2}(a_1^4 - t_n)$$
$$= (a_1^4 - t_n)$$

2.

(b) $\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$ can be written as

$$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = \frac{\partial E_n(w)}{\partial a_1^{(4)}} \times \frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}}$$

inserting the result from previous question

$$= (a_1^4 - t_n) \times \frac{\partial a_1}{\partial w_{12}^{(3)}}$$

$$a_1^{(4)} = \sum_{k=1}^{3} w_{1k}^{(3)} z_k^{(3)}$$

Then,

$$\frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}} = z_2^{(3)}$$

$$a_j^L = \sum_k w_{jk}^{(L-1)} z_k^{(L-1)}$$

$$\frac{\partial a_j^L}{\partial w_{jk}^{(L-1)}} = z_k^{(L-1)}$$

we can conclude that:

$$\frac{\partial E_n(w)}{\partial w_{12}} = (a_1^{(4)} - t_n) z_2^{(3)}$$

or

$$\frac{\partial E_n(w)}{\partial w_{12}} = \delta_1^{(4)} \times z_2^{(3)}$$

3.

$$\frac{\delta E_n(w)}{\delta a_1^{(3)}} = \sum_{k=1}^{1} \frac{\delta E_n(w)}{\delta a_k^{(4)}} \times \frac{\delta a_k^{(4)}}{\delta a_1^{(3)}}$$

where $\sum_{k=1}^{1} \frac{\delta E_n(w)}{\delta a_k^{(4)}} = \delta_1^{(4)}$

Then,

$$a_k^{(4)} = \sum_{k=1}^{3} w_{1k}^{(3)} z_k^{(3)} \qquad z_k = h(a_k)$$

$$a = w_{11}^{(3)} z_1^{(3)} + w_{12}^{(3)} z_2^{(3)} + w_{13}^{(3)} z_3^{(3)}$$

$$\frac{\delta a_k^{(4)}}{\delta a_1^{(3)}} = w_{11}^{(3)} z_1^{(3)} + \cancel{w_{12} z_2^{0}} + \cancel{w_{13} z_3^{0}}$$

$$= w_{11}^{(3)} h'(a_1^{(3)})$$

So,

$$\frac{\delta E_n(w)}{\delta a_1^{(3)}} = \delta_1 \cdot \times w_{11}^{(3)} \times h'(a_1^{(3)}).$$

4.

$$\frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = \delta_1^{(3)} \times \frac{\partial a_1^{(3)}}{w_{11}^{(2)}}$$

Similarly,

$$\frac{\partial a_1^{(3)}}{w_{11}^{(2)}} = z_1^{(2)}$$

And,

$$\delta_1 = \frac{\partial E_n(w)}{\partial a_1^{(3)}} = h'(a_1^{(3)}) w_{11}^{(3)} \delta_1^{(4)}$$

So,

$$\frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = h'(a_1^{(3)}) w_{11}^{(3)} \delta_1^{(4)} z_1^{(2)}$$

5.

$$\frac{\partial E_n(w)}{\partial a_1^{(2)}} = \sum_1^3 \frac{\partial E_n(w)}{\partial a_k^{(3)}} \times \frac{\partial a_k^{(3)}}{\partial a_1^{(2)}}$$

where,

$$\sum_1^3 \frac{\partial E_n(w)}{\partial a_k^{(3)}} = \sum_1^3 \delta_k^{(3)}$$

$$a_k^{(3)} = \sum_1^3 w_k^{(2)} z_k^{(2)}$$

$$\frac{\delta a_k^{(3)}}{\partial a_1^{(2)}} = z_1^{(2)} \sum_1^3 w_k^{(2)}$$

$$= h'\left(a_1^{(2)}\right) \sum_1^3 w_k^{(2)}$$

So,

$$\frac{\partial E_n(w)}{\partial a_1^{(2)}} = h'\left(a_1^{(2)}\right) \sum_1^3 w_k^{(2)} \delta_k^{(3)}$$

6.

(+) $$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = \delta^{(2)} \times \frac{\partial a_1^{(2)}}{w_{11}^{(1)}}$$

Similarly,

$$\frac{\partial a_1^{(2)}}{w_{11}^{(1)}} = z^{(1)}$$

And,

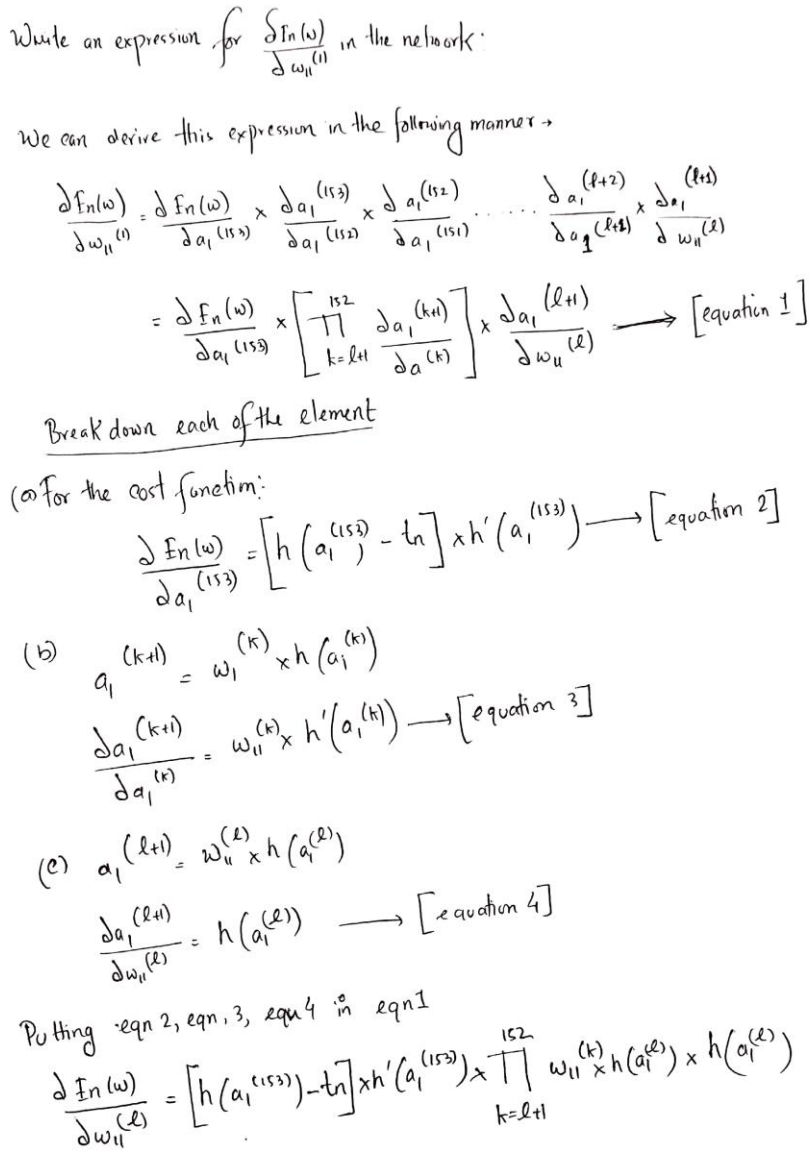
$$\delta_1^{(2)} = \frac{\partial E_n(w)}{\partial a_1^{(2)}} = h'\left(a_1^{(2)}\right) \sum_1^3 w_{1k}^{(2)} \delta_k^{(3)}$$

Then,

$$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = h'\left(a_1^{(2)}\right) z_1^{(1)} \sum_1^3 w_{1k}^{(2)} \delta_k^{(3)}$$

Vanishing Gradient

1.



2. When training a deep model using back propagation, we calculate the gradient of the output with respect to the weight which is subtracted from those respective weight to make values more accurate to give correct output. The gradient tends to get smaller as we move backward in the network. The neuron in the earlier layers learn very slowly as compared to the neurons in the later layers in the hierarchy. The sigmoid function "squeezes" any input value into an output range of (0,1) which is very useful for representation of probabilities and classification. It is observed, when the sigmoid function value is either too high or too low, the derivative becomes very small i.e. << 1. This causes vanishing gradients and poor learning for deep networks. This can occur when the

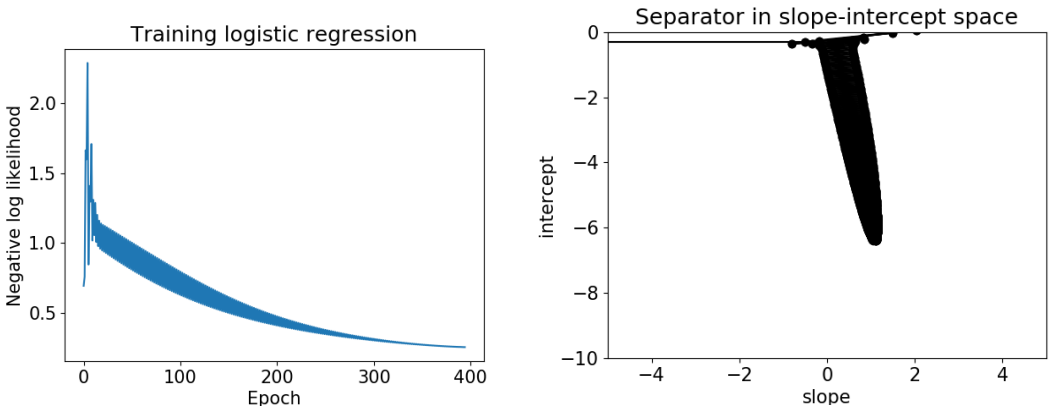
weights of our networks are initialized poorly – with too-large negative and positive values. These too-large values *saturate* the input to the sigmoid and pushes the derivatives into the small valued regions. However, even if the weights are initialized properly, and the derivatives are sitting around the maximum with many layers there will still be a vanishing gradient problem.

3. The *ReLU* activation returns its argument x whenever it is greater than zero and returns 0 otherwise. Then the first derivative of *ReLU* is equal to 1 when x is greater than zero, but otherwise it is 0. The advantage of *ReLU* activation function is that there will be no degradation of the error. The derivative is zero when x < 0 and this makes certain weights killed off. The back propagated error can be cancelled out whenever there is a negative input into a given neuron and therefore the gradient will also fall to zero.

4. For the case of bipartite graph, for the layer l +1 layer, we need to consider two nodes. So, their gradient will become zero when both $\boldsymbol{h'(a_1^{(l+1)})}$ and $\boldsymbol{h'(a_2^{(l+2)})}$ becomes zero (sum of them, which is the derivative of the cost function w.r.t $\boldsymbol{W_{11}^{l}}$ goes to zero.

Logistic Regression

1. Figure 4(a) represents the plot of separator path in slope intercept space and Figure 4(b) represents the negative log likelihood over iterations. The oscillating plot occurs due to the learning rate and the direction of the gradient descent. If the scale of the learning rate is 0.5, it will be too large, and the gradient descent will jump too large in each iteration. Since the direction is pointing exactly to the bottom, it will experience a lot of oscillation.



(a) the plot of neg. log likelihood over iterations (b) the plot of slope-intercept space

Figure 4: The plot of separator path in slope-intercept space and the plot of neg. log likelihood over iterations.

2. Figure 5 represents the log-likelihood versus iteration for the different learning rates. It can be concluded that the learning rate n grows with the speed of convergence. However, as the n becomes smaller, the regression becomes smoother but the speed decreases.

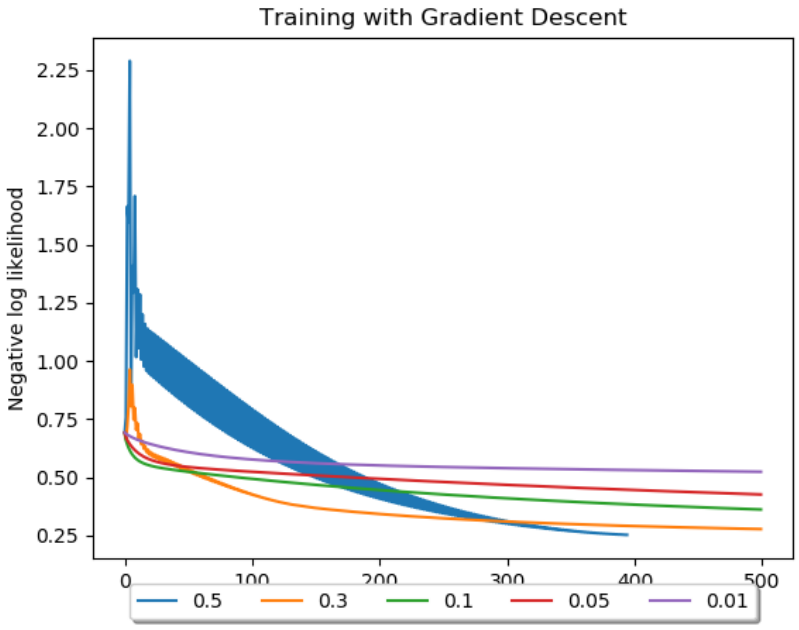


Figure 5: The plot comparing negative log-likelihood versus iteration for the different learning rates

3. Stochastic gradient descent depends on the scale of learning rate *n;* thus, it is not always faster than the gradient descent. If n is large, the stochastic descent will be slower and vice versa. Here in this scenario for all step sizes, the stochastic gradient has taken less iterations than gradient descent to reach the minimum error level.

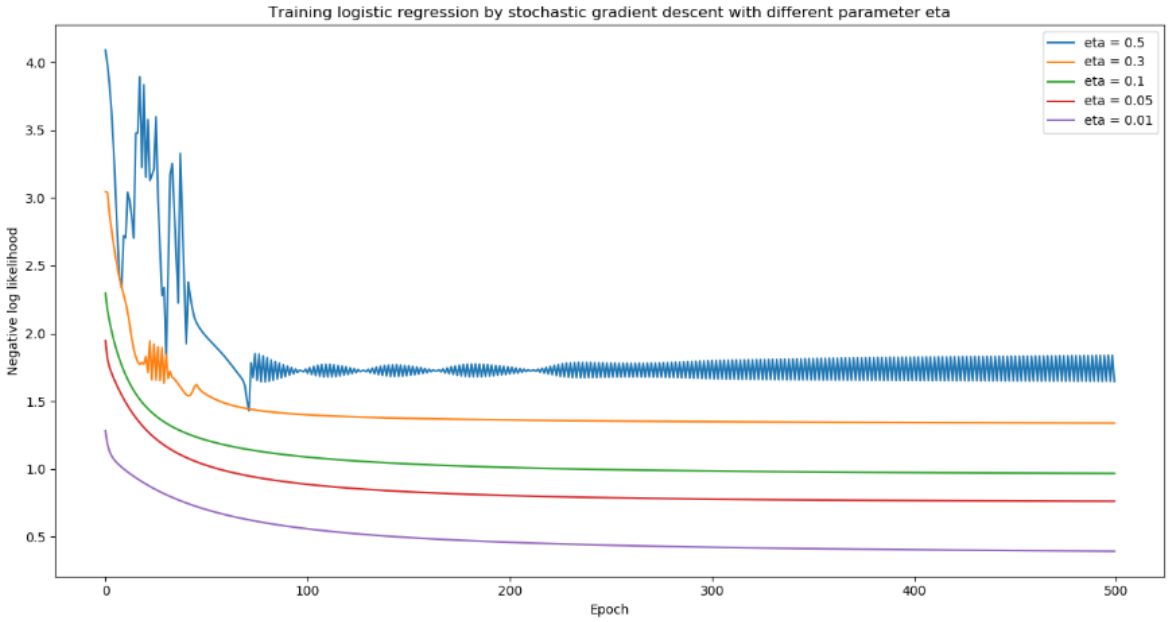


Figure 6: The stochastic gradient descent version of the plot comparing negative log-likelihood versus iteration for the different learning rates

Fine Tuning a Pre-Trained Network

Write a Python function to be used at the end of training that generates HTML output showing each test image and its classification scores. You could produce an HTML table output for example.

For the given problem, the following parameters are used:

- Dataset: CIFR10
- Number of training sample: 4500
- Number of testing sample: 500
- Number of Epochs: 5
- CUDA to train in GPU

Figure 7: below shows the precision of the network, which in this case has increased upon each epochs.

Epoch	Precision(%)
1	39.0
2	50.0
3	49.8
4	48.8
5	53.8

Figure 7: Precision for CIFR10 dataset

Figure 8: illustrates the classification scores generated from training the model. The table is created in HTML format

Test Image and Its Classification Scores

Image	P(plane)	P(car)	P(bird)	P(cat)	P(deer)	P(dog)	P(frog)	P(horse)	P(ship)	P(truck)	The prediction
dog	-1.85	-3.33	-0.5	2.49	0.29	2.58	0.36	-0.4	0.42	1.91	dog
dog	-1.05	-0.3	1.3	2.82	0.0	1.59	0.82	0.32	0.18	-2.28	cat
bird	-0.48	-0.8	2.9	-3.98	1.38	-1.99	-3.45	3.94	-1.18	-1.73	horse
car	3.18	2.29	-0.92	-2.35	-1.23	-4.46	0.47	-1.96	-0.76	3.32	truck
cat	-0.37	-2.05	0.32	1.52	-0.61	0.17	-1.26	0.82	2.85	-3.43	ship
cat	-2.41	2.19	-0.99	-1.14	-1.84	-0.86	3.95	-3.02	-1.94	1.72	frog
frog	2.01	0.26	2.67	-0.47	-0.41	-0.76	-0.44	-0.18	-1.91	0.01	bird
deer	-0.08	-1.58	1.34	-1.34	3.21	-1.11	-2.82	2.68	0.14	0.33	deer

Figure 8: Classification Scores on CIFR10 dataset