CMPT 419 Softmax and Neural Networks

Echo Liu

October 2018

1 Question 1: Softmax for Multi-Class Classification

1. We make $a_1 = a_2 = a_3$ therefore our subsitution
   for the softmax function goes as follows
   $\frac{e^{a_1}}{3*e^{a_1}}$
   simplifying our fraction
   we obtain $\frac{1}{3}$ as such this suggests that our green point
   has equal probability to be in either of the three classes.

2. We let $a_1 = a_2$ hence if we compute the activation for a_3
   we notice that our activations for a_3 is negative but same in magnitude with respect to
   a_1 computing our probabilities we end up with a probability of nearing 50 percent.

3. The same analogy applies, we would get negative activations for the class that is not in the numerator.

2 Question 2: Error Backpropogation

Consider the output layer responses:

1. Recall that the deriative of the identity is 1 Thus our expression for $\delta^{(4)} = y_k - t_k$ an alternate form for this expression is $(y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})$

2. $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l+1)} * z_i^{(l)}$
   Therefore:
   $\frac{\partial E_n}{\partial w_{12}^{(3)}} = \delta_1^{(4)} * z_2^{(3)} = (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})) * h(a_2^{(3)})$
   where:
   $h(a_2^{(3)}) = g_{logistic}(a_2^{(3)})$

Next, consider the penultimate layer of nodes responses:

1. $\frac{\partial E_n}{\partial a_1^{(3)}} = \delta_1^{(3)}$
   $\delta_j^{(l)} = h'(a_1)^{(l)} \sum_{k=1}^{k} w_{kj}^{(l)} \delta_k^{(l+1)}$
   $\delta_1^{(3)} = h'(a_1)^{(3)} \sum_{k=1}^{k} w_{k1}^{(3)} \delta_k^{(4)}$

$\delta_1^{(3)} = h'(a_1)^{(3)} \sum_{k=1}^{k} w_{k1}^{(3)} \delta_k^{(4)}$
$\delta_1^{(3)} = h'(a_1)^{(3)} * w_{11}^{(3)} \delta_1^{(4)}$
$\delta_1^{(3)} = h'(a_1)^{(3)} * w_{11}^{(3)} * (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n}))$
Where: $h'(a_1)^{(3)} = g_{logistic}(a_1^{(3)})(1 - g_{logistic}(a_1^{(3)})$

2. $\frac{\partial E_n}{\partial w_j i}^{(l)} = \delta_j^{(l+1)} * z_i^{(l)}$
   $=$
   $\frac{\partial E_n}{\partial w_{11}}^{(2)} = \delta_1^{(3)} * z_1^{(2)}$
   $\frac{\partial E_n}{\partial w_{11}}^{(2)} = w_{11}^{(3)} * (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})) * z_1^{(2)}$

Responses to "Finally, consider the weights connecting from the inputs"

1. $\frac{\partial E_n}{\partial a_1}^{(2)} = \delta_1^{(2)}$
   $\delta_j^{(l)} = h'(a_j)^{(l)} * \sum_{k=1}^{k} w_{kj}^{(l)} * \delta_k^{(l+1)}$ $\delta_1^{(2)} = h'(a_j)^{(2)} * \sum_{k=1}^{k} w_{k1}^{(2)} * \delta_k^{(3)}$ $\delta_1^{(2)} = h'(a_1)^{(2)} * \sum_{k=1}^{k} w_{k1}^{(2)} * \delta_k^{(3)}$ $\delta_1^{(2)} = h'(a_1)^{(2)} * (w_{11}^{(2)} * \delta_1^{(3)} + w_{21}^{(2)} * \delta_2^{(3)} + w_{31} * \delta_3^{(3)})$
   where
   $h'(a_1)^{(2)} = (g(a_1)^{(2)})(1 - g(a_1)^{(2)})$
   $\delta_1^{(3)} = (g(a_1)^{(3)})(1 - g(a_1)^{(3)})(w_{11}^{(3)} * (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})^{(4)})$
   $\delta_2^{(3)} = (g(a_2)^{(3)}) * (1 - g(a_2)^{(3)}) * (w_{12}^{(3)} * (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})^{(4)})$
   $\delta_3^{(3)} = (g(a_3)^{(3)}) * (1 - g(a_2)^{(3)}) * (w_{13}^{(3)} * (y(\mathbf{x_n}, \mathbf{w}) - \mathbf{t_n})^{(4)})$
   $\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \delta_j^{(l+1)} * z_i^{(l)}$
   $\frac{\partial E_n}{\partial w_{11}^{(1)}} = \delta_2^{(2)} * z_1^{(1)}$

3 Vanishing Gradients

1. $\frac{\partial E_n}{\partial w_{11}^{(l)}} = \delta_1^{(l+1)} * z_1^{(l)}$
   $\delta_j^{(l)} = h'(a_j)^{(l)} * \sum_{k=1}^{k} w_{kj}^{(l+1)} * \delta_j^{(l+1)}$
   where $k = 1$ $\delta_1^{(l)} = h'(a_j^{(l+1)}) * \sum_{k=1}^{k} w_{kj}^{(l)} * \delta_j^{(l+1)}$ Furthermore, since we only have one neuron at each hidden layer, our equation reduces to the following:
   $\delta_1^{(153)} = h'(a_1)^{(153)} * w_{11}^{(153)} * \delta_1^{(154)}$ Where $\delta_1^{(154)} = (y_k - t_k)$
   $\delta_1^{(152)} = h'(a_1)^{(152)} * w_{11}^{(152)} * \delta_1^{(153)}$
   $\delta_1^{(152)} = h'(a_1)^{(152)} * w_{11}^{(152)} * h'(a_1)^{(153)} * w_{11}^{(153)} * \delta_1^{(154)}$

I.e: We obtain a product of the form
$\delta_1^{(l)} = \prod_{i=l}^{O} h'(a_1)^{(i)} * w_{11}^{(i)} * \delta_1^{(i+1)}$ (Eq 1)
Consider a network wih n hidden layers with a single neuron in each matrix. Then we just simply take the product of all successive terms up to our output layer as evidentially seen in our deriavation of (1). Since we have $h'(a_1)^{(l)}$ as a term in our product, our deriative is then inversely proportional to the amount of terms in our product. As a result, in our particular scenerio the deriative approaches $\frac{1}{153}$ as our terms get larger and larger. This phenomena happens when we initialize our weight matrices to very small values. Likewise, the same argument applies to exploding gradients when we initialize our weight matrices to very large values. Thus, the gradients would be reasonable in magnitude when the sigmoidal function is not evaluated at its extremities.

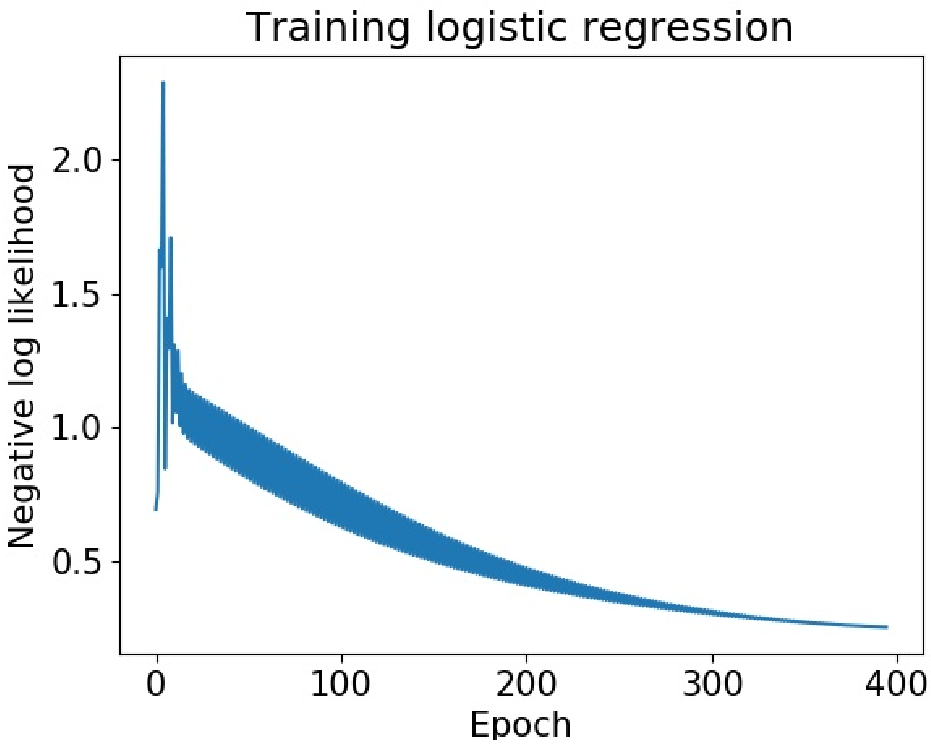
2. The weights would become zero when the deriatives of the sigmoid function are tending towards zero. In otherwords, when the value of the sigmoidal activation function is prohibitively large or prohabitively small then the weights would be saturated.

3. Same argument applies but when $x/n > 0$

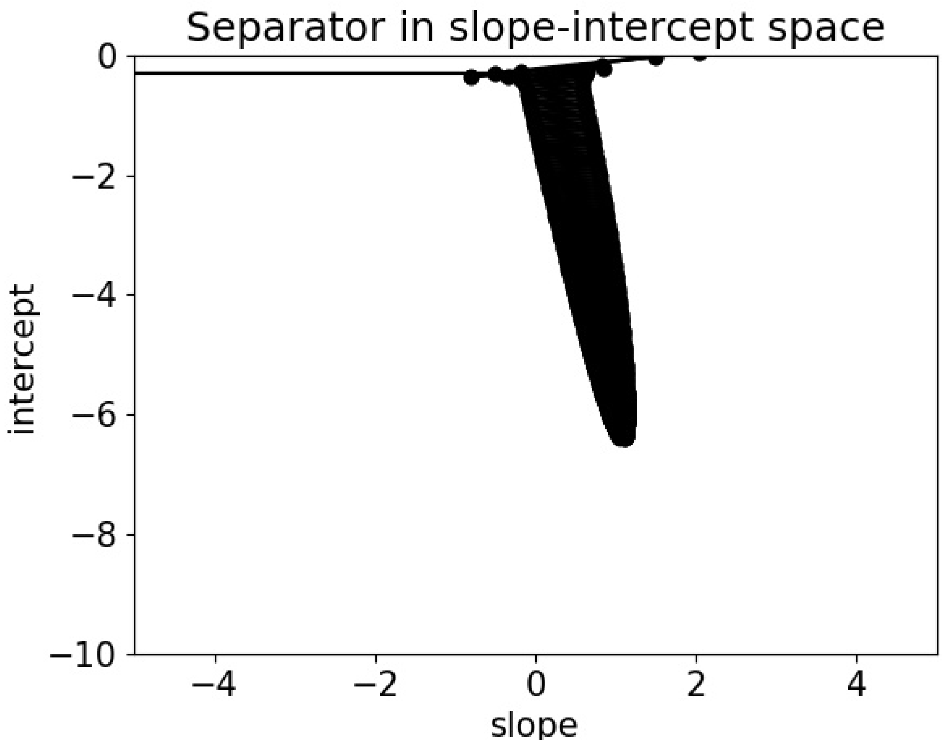
4. When any subset of the weights are saturated between zero and one

4 Logistic Regression

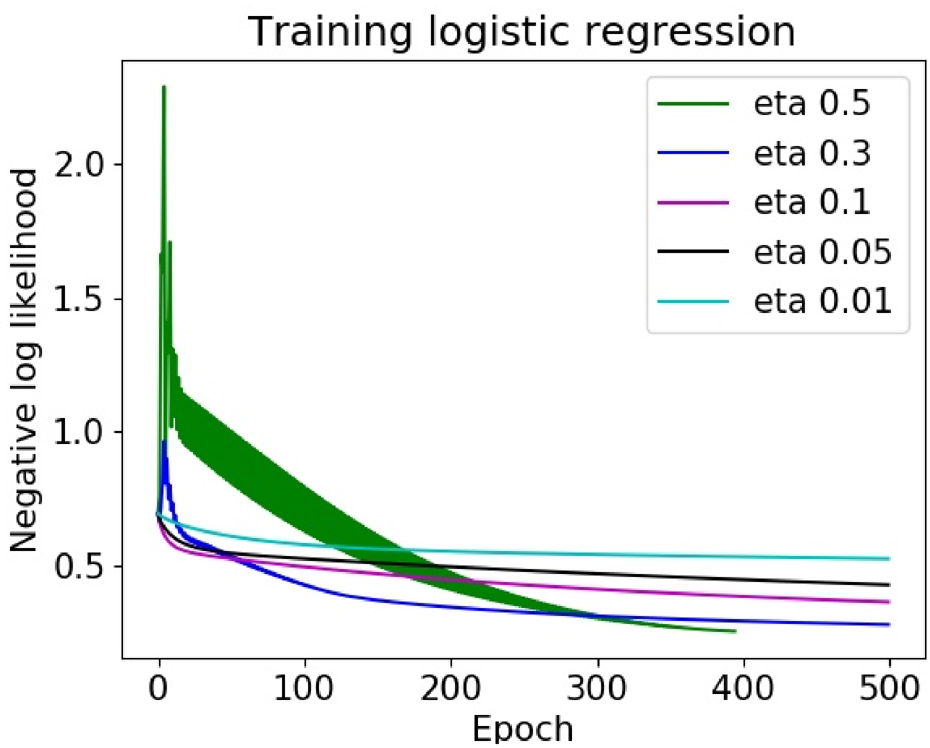
1. The reason why the errors are ossicilating are have to do with the values of the eigenvalues at each step. I.e the eigenvalues dictate the rate of descent. With this intention in mind, the reason why the errors ossicilate is due to the noise in the data.



Training logistic regression
Negative log likelihood
2.0
1.5
1.0
0.5
0
100
200
300
400
Epoch

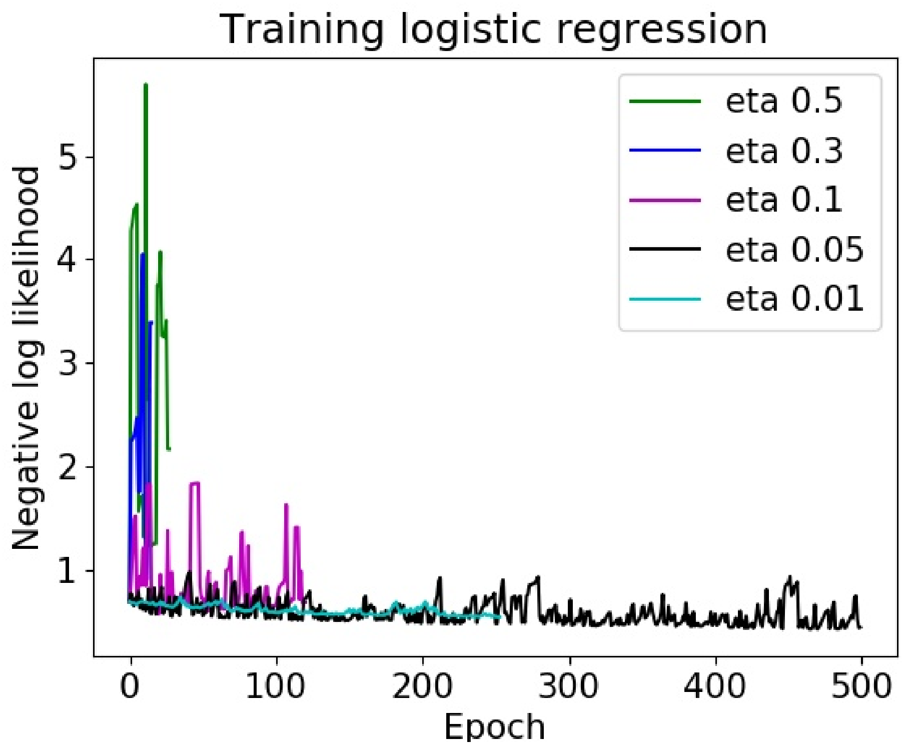


2. As seen in the following figure below:



At the 300 mark, there is no advantage between setting eta=0.5 and eta=0.3. This indicates that at that particular epoch, both eta values have approached a steady state. After that epoch point however, it can be evidentially seen that eta=0.3 provides the best learning rate.

3. Stochastic gradient in terms of error converges faster than batch gradient descent.



5 Fine-Tuning a Pre-Trained Network

What I have implemented so far: The HTML output for the images and thier associated probabilities.