Logistic and Softmax Regression for Handwritten Digits Classification

Shilin Zhu
Ph.D. student, Computer Science
UCSD
La Jolla, CA
shz338@eng.ucsd.edu

Yunhui Guo
Ph.D. student, Computer Science
UCSD
La Jolla, CA
email

1 Logistic Regression via Gradient Descent

Derive the gradient for logistic regression:
The cross-entropy cost function can be expressed as

$$E(w) = -\sum_{n=1}^{N}[t^n \ln(y^n) + (1-t^n)\ln(1-y^n)] \tag{1}$$

where t^n is the target label for example n and y^n is our prediction for this example. To perform gradient descent, we need to first compute the gradient (derivative) of the cost function with respect to the parameters. The gradient of cost function on example n is

$$-\frac{\partial E^n(w)}{\partial w_j} = \frac{\partial[t^n \ln(y^n) + (1-t^n)\ln(1-y^n)]}{\partial w_j} \tag{2}$$

where y^n is the prediction of logistic regression as

$$y^n = g(\sum_{j=0}^{m} w_j x_j^n) \tag{3}$$

and $g(\cdot)$ is the sigmoid activation function and its derivative is

$$g'(z^n) = \frac{d(\frac{1}{1+e^{-z^n}})}{dz^n} = g(z^n)(1-g(z^n)) \tag{4}$$

where $z^n = \sum_{j=0}^{m} w_j x_j^n$. According to the chain rule in calculus, we can then compute the gradient (derivative) of the cost function on example n with respect to the parameters as

$$-\frac{\partial E^n(w)}{\partial w_j} = -\frac{\partial E^n(w)}{\partial y^n}\frac{\partial y^n}{\partial z^n}\frac{\partial z^n}{\partial w_j} = (\frac{t^n}{y^n} - \frac{1-t^n}{1-y^n}) \cdot y^n(1-y^n) \cdot x_j^n = (t^n - y^n)x_j^n \tag{5}$$

Derive the gradient for regularized logistic regression:

To prevent potential overfitting, regularization is used in logistic regression. The cross-entropy cost function with regularization term can be computed as

$$E(w) = -\sum_{n=1}^{N}[t^n \ln(y^n) + (1-t^n)\ln(1-y^n)] + \lambda * C(w) \tag{6}$$

where $C(w)$ represents the complexity of the model. $L1$ and $L2$ regularizations are two most common functions people use

$$C(w) = ||w||^2 = \sum_{i,j} w_{i,j}^2 (L2) \tag{7}$$

$$C(w) = |w| = \sum_{i,j} |w_{i,j}| (L1) \tag{8}$$

Thus the gradient of cost function on example n is

$$-\frac{\partial E^n(w)}{\partial w_j} = \begin{cases} (t^n - y^n)x_j^n - 2\lambda w_j, & \text{if } L2 \\ (t^n - y^n)x_j^n - \lambda sign(w_j), & \text{if } L1 \end{cases}$$

Note that we can always add a factor of $1/N$ to scale the cost and gradient to somehow speed up the learning, and this will not change the optimization results (learned parameters).

2 Softmax Regression via Gradient Descent

2.1 Problem definition

In this problem, we need to classify MNIST datasets using softmax regression. In the experiments, we only use the first 20,000 training images and the last 2,000 test images.

2.2 Methods

We use softmax regresion for this problem.
Derive the gradient for Softmax Regression:
The cross-entropy cost function can be expressed as,

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{9}$$

Where,

$$y_k^n = \frac{\exp(a_k^n)}{\sum_{k'} \exp(a_{k'}^n)} \tag{10}$$

And,

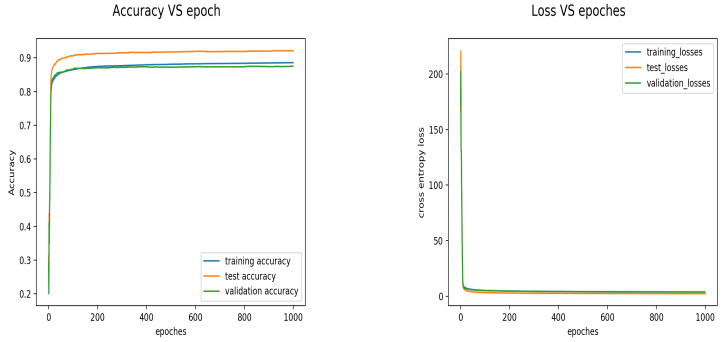
$$a_k^n = w_k^T x^n \tag{11}$$

We can calculate the gradient for softmax regression as follows,

$$\begin{aligned} -\frac{\partial E^n(w)}{\partial w_{jk}} &= -\frac{\partial E^n(w)}{\partial a_k^n}\frac{\partial a_k^n}{\partial w_{jk}} \\ &= -\sum_{k'} \frac{\partial E^n(w)}{\partial y_{k'}^n}\frac{\partial y_{k'}^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial w_{jk}} \end{aligned} \tag{12}$$

And

$$\frac{\partial y_{k'}^n}{\partial a_k^n} = y_{k'}^n \delta_{kk'} - y_{k'} y_k \tag{13}$$



(a) label 1 (b) label 2

Figure 1: 2 Figures side by side

Where $\delta_{kk} = 1$ if $k = k'$, otherwise $\delta_{kk} = 0$. And

$$\frac{\partial E^n(w)}{\partial y^n_{k'}} = -\frac{t_{k'}}{y_{k'}} \tag{14}$$

Substitute Equation (5) and Equation (6) into Equation (4) we get,

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (t_k - y_k)x^n_j \tag{15}$$

Preprossing: First, we extract the first 20,000 training images and the last 2,000 test images. Then normailize the images to make sure the pixel values are in the range of [0,1]. Convert the labels to one-hot vectors. Divide the training images into two parts, the first 10% are used for as a hold-out set and the rest 90% are used for training.
Experiments settings: To determine the best type of regurization and the best λ, we try L_2 regularization and L_1 regularization seperately.
For the L_2 regularizartion, we search the best λ in the set $\{0.01, 0.001, 0.0001\}$. If the accuracy on the hold-out set decreases for 3 epochs, we stop the algorithm and use the weights with the minimum error (highest accuracy) on the hold-out set as the final answer. For the L_1 regularization, we follow the same steps. Then we compare the results get from these two regularization methods and use it as the best final result.

2.3 Results

(a) In the experiments, we find that using L_2 regularization with $\lambda = 0.01$ obtain the best result on the validation set. With an accuracy of 0.9045% on the validation set. With such settings, the accuracy on the test set is 0.927%.

(b) In this experiement, we use L_2 regularization with $\lambda = 0.01$. The figure is shown in Fig **??**.

(c) In this experiement, we use L_2 regularization with $\lambda = 0.01$. The figure is shown in Fig **??**.

(d) We plot the results in Fig 2. We can see that the image of the weight and the corresponding image of the average digit is almost the same. The reason is that we classify the images based on the inner product of the pixesls with the weights. And the inner product is maximized when the angle between the weight and the image is zero. So we see that the image of the weight and the corresponding image of the average digit is similar.

2.4 Discussion

Acknowledgments

.

References

Images of Weights and Average Examples

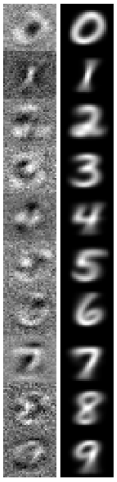


Figure 2