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

1.1 Problem Definition

In this work, we realize the handwritten digit classification using MNIST database. Our goal is to accurately and robustly recognize what number is present in a new image. In this section we will first use logistic regression to classify only two digit classes (binary classification), later in the next section we will use softmax regression to generalize logistic regression into N classes.

1.2 Mathematical Derivation

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

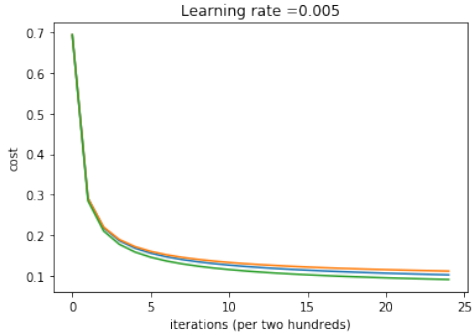
$$-\frac{\partial E^n(w)}{\partial w_j} = \frac{\partial[t^n \ln(y^n) + (1-t^n)\ln(1-y^n)]}{\partial w_j} \tag{2}$$

where y^n is the prediction of logistic regression as

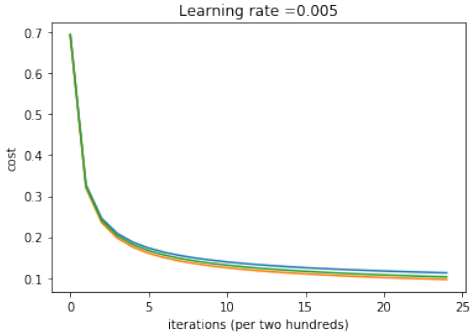
$$y^n = g(\sum_{j=0}^{m} w_j x_j^n) \tag{3}$$

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

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



(a) cost function with iterations for batch GD on classifying 2 vs. 3



(b) cost function with iterations for batch GD on classifying 2 vs. 8

Figure 1: Cost function through batch GD training process

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

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

1.3 Data Reading, Parsing and Feature Extraction

In this work we use the famous MNIST hand written digits database created by Yann LeCun. Each image in the database contains 28×28 pixels and each pixel has a grayscale intensity, so that the input feature $x \in R^{784}$ after we unroll the 2D image into an 1D vector. To include the bias term, after reading the data, we append a '1' to the beginning of each x vector so the final $x \in R^{785}$. We will use the first 20,000 training images and the last 2,000 test images. Note that for logistic regression to do binary classification, we can only use the images of two specific digit classes, so that the actual training images and test images are smaller than 20,000 and 2,000 respectively. To speed up learning, we need to apply feature normalization. For images, the most common way is to normalize the pixel values by the maximum pixel value 255. After normalization, all the values in x is now ranging from 0 to 1.

The following code shows how we can choose the example images corresponding to two specific digit classes. For the rest of the report, we choose two binary classification problems: 2 vs. 3 and 2 vs. 8.

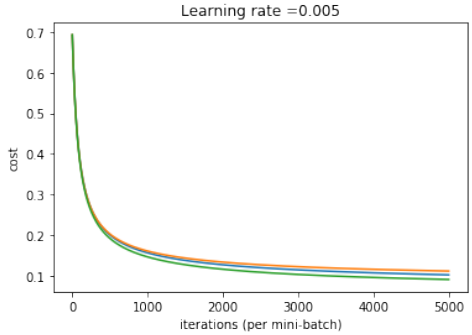
TODO: Attach code here.

1.4 Experimental Results

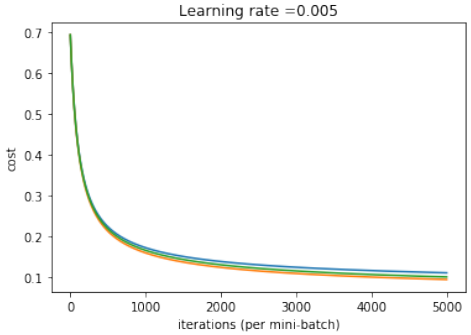
Batch gradient descent
We first apply batch gradient descent rule on logistic regression since the training set is not that huge, thus batch gradient descent can work reasonably fast.

After trying different values of learning rate, we found 0.005 is a reasonably good learning rate for this problem. Fig. **??**(a) plots the loss function over training for the training set, the hold-out validation set and the test set. From the results we can see the model tries to minimize the cost and meanwhile maximize the likelihood to perform good prediction on the data.

Mini-Batch gradient descent
We can use mini-batch gradient descent to speed up learning process. Since the one-step optimization is done on a smaller mini-batch, the cost function will not monotonically decrease as batch gradient descent. Here we change the weights after a mini-batch of roughly 10% of the entire training set. This set of mini-batch size can result in relatively smooth curve of the cost function. Fig.

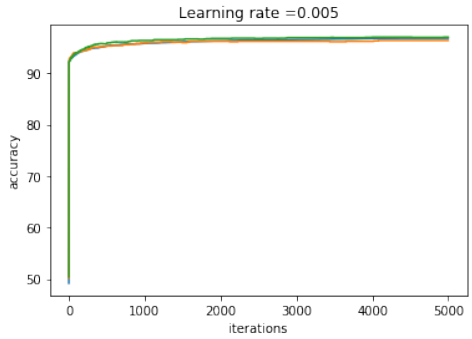


(a) cost function with iterations for mini-batch GD on classifying 2 vs. 3

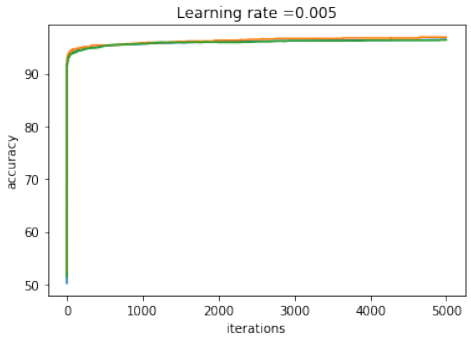


(b) cost function with iterations for mini-batch GD on classifying 2 vs. 8

Figure 2: Cost function through mini-batch GD training process



(a) accuracy with iterations on classifying 2 vs. 3



(b) accuracy with iterations on classifying 2 vs. 8

Figure 3: Accuracy through training process

??(b) plots the loss function over training for consecutive mini-batches. Since the validation set always has very similar errors as the test set, we can conclude the hold-out set work as a good stand-in for the test set and their underlying data distributions are same.

Accuracy of Prediction Using Logistic Regression Classifier

Weight Visualization

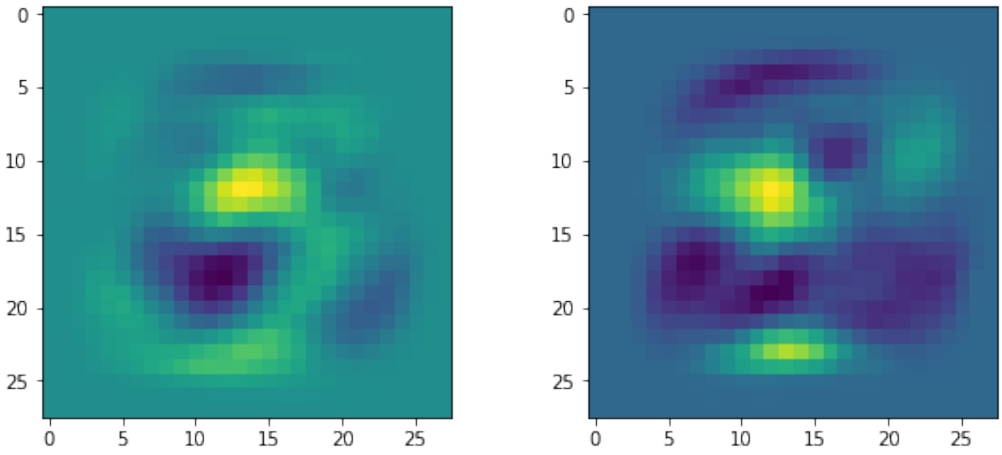
We can visualize the weights learned to see what logistic regression learns through the training process.

Computing the difference between these two different classifiers, we can get a new weight matrix visualized in Fig. **??**. This result shows that the new weights can be used to accurately classify digit 3 and digit 8.

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

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



(a) weights learned on classifying 2 vs. 3 (b) weights learned on classifying 2 vs. 8

Figure 4: Weight visualization on binary classification using logistic regression

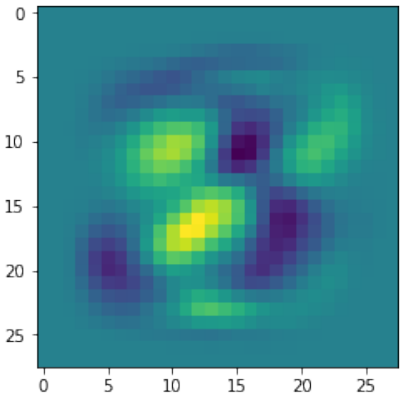


Figure 5: Difference of the weights between the two classifiers (2 vs. 3 and 2 vs. 8)

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

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

where $sign(w_j)$ is the signature function of w_j.

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

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,

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

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

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

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

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

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

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

Derive the gradient for Softmax Regression with Regularizations:

With regularization, generally the loss function can be writen as,

$$J(w) = E(w) + \lambda C(w) \tag{16}$$

If we use L_1 regularization in softmax regression,

$$\lambda C(w) = \lambda C_{L_1}(w) = \lambda\sum_{ij}|w_{i,j}| \tag{17}$$

We can compute the derivate of $\frac{\partial C}{\partial w}$ as follows,

$$\frac{\partial C_{L_1}(w)}{\partial w_{ij}} = sign(w_{i,j}) \tag{18}$$

Where $sign(x) = 1$ if $x > 0$, $sign(x) = 0$ if $x = 0$ and $sign(x) = -1$ if $x < 0$.

If we use L_2 regularization in softmax regression,

$$\lambda C(w) = \lambda C_{L_2}(w) = \lambda \sum_{ij} w_{i,j}^2 \tag{19}$$

We can compute the derivate of $\frac{\partial C}{\partial w}$ as follows,

$$\frac{\partial C_{L_2}(w)}{\partial w_{ij}} = 2w_{i,j} \tag{20}$$

In summary,

$$-\frac{\partial J^n(w)}{\partial w_{j,k}} = \begin{cases} (t_k - y_k)x_j^n - \lambda sign(w_{j,k}), & \text{if use } L_1 \text{ regularization} \\ (t_k - y_k)x_j^n - 2\lambda w_{j,k}, & \text{if use } L_2 \text{ regularization} \end{cases}$$

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: We use standard normal distributions to initilize the weights. We use an initial learning rate $\eta(0)$ = 0.004 and use equation $\eta(t) = \eta(0)/(1 + t/T)$ to anneal the learning rate by reducing it over time. t is used to index the epoch number and T is a metaparameter which is set to be 2. In the experiements, we find that above learning settings work best.
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 highest accuracy on the hold-out set as the final answer. For the L_1 regularization, we follow the same steps. We run the 1000 epochs for each setting. Then we compare the results got from these two regularization methods and use the best one as the 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. The accuracy is 0.927% on the test set under such settings.

To further examine the performance of the algorithm, we try other λs. We choose λ in the set $\{0.05, 0.005, 0.0005\}$ and use L_2 regularization. The highest accuracy on the validation set is 0.8965% with $\lambda = 0.0005$. And the test accuracy is 0.933% which is slightly higher than when $\lambda = 0.01$. So in following experiments, we fix λ to be 0.0005.

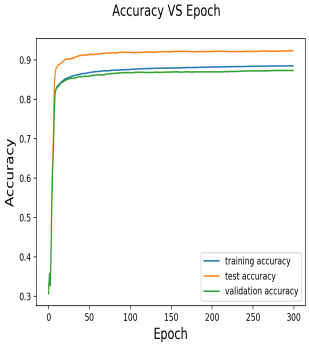
(b) In this experiement, we use L_2 regularization with $\lambda = 0.005$. The figure is shown in Fig 6a.

(c) In this experiement, we use L_2 regularization with $\lambda = 0.005$. The figure is shown in Fig 6b. Note that we use the sum of loss of individual data rather than the average one.

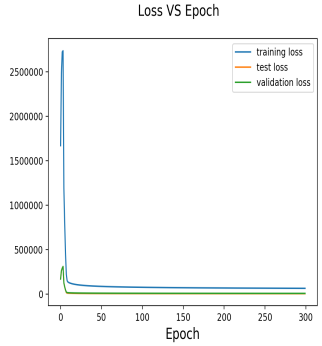
(d) We plot the results in Fig 7.

2.4 Discussion

From 6a and 6b we can see that the loss function went up before going down. And the accuracy and loss function converges quickly and then becomes smooth. The plausible reason is that the learining rates in the first few epoches maybe a little large but after "annealing", the learning rates are suitable enough to make the algorithm converge. Setting the learning rate too small initially may let the algorihtm too slow to converge or just being stucked. And we also see that there is no overfitting



(a) The percent correct over the number of training iterations for the training, hold-out and test set.



(b) The value of the loss function over the number of training iterations for the training, hold-out, and test set.

Figure 6: The performance of the algorithm over the number of training iterations.

Figure 7: Images of the weights and the average examples.

occurs. One possible reason for this is that the impact of the regularization. Another one is that the algorithm correctly learns the pattern underlying the data at hand.

From 7 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 pixels 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.

Acknowledgments

.

References