COGS 260 Spring 2018: Assignment 3

Qiaojun Feng
University of California, San Diego
qjfeng@ucsd.edu

Abstract

In this assignment we are working on the following questions:

1. *Least Square Estimation*
2. *Parabola Estimation*
3. *Perceptron Learning*
4. *Feed Forward Neural Network*
5. *Convolutional Neural Network*

1. Method(Derivation)

1.1. Least Square Estimation

$$X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times m} (m \text{ is feature dimension})$$
$$Y = [y_1, y_2, \ldots, y_n]^T \in \mathbb{R}^n$$

$$g(W) = \|X \cdot W - Y\|_2^2 = (X \cdot W - Y)^T (X \cdot W - Y)$$

(a) Compute the gradient of $g(W)$ with respect to W.

$$\begin{aligned} g(W) &= (X \cdot W - Y)^T (X \cdot W - Y) \\ &= W^T X^T X W - Y^T X W - W^T X^T Y + Y^T Y \\ &= W^T X^T X W - 2 Y^T X W + Y^T Y \end{aligned}$$

$$\begin{aligned} \frac{\partial g(W)}{\partial W} &= \frac{\partial}{\partial W}(W^T X^T X W - 2 Y^T X W + Y^T Y) \\ &= \frac{\partial}{\partial W}(W^T X^T X W - 2 Y^T X W) \\ &= 2 X^T X W - 2 X^T Y \end{aligned}$$

(b) By setting the answer of part (a) to 0, prove the following:

$$W^* = \arg\min_W g(W) = (X^T X)^{-1} X^T Y$$

$$\begin{aligned} \frac{\partial g(W)}{\partial W} &= 2 X^T X W - 2 X^T Y = 0 \\ X^T X W &= X^T Y \\ (X^T X)^{-1} (X^T X) W &= (X^T X)^{-1} X^T Y \\ W^* &= (X^T X)^{-1} X^T Y \end{aligned}$$

1.2. Parabola Estimation

$$X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \; Y = [y_1, y_2, \ldots, y_n]^T$$
$$f(x; W) = w_0 + w_1 x + w_2 x^2 = \mathbf{x}^T W$$
$$\mathbf{x} = [1, x, x^2]^T \quad W = [w_0, w_1, w_2]^T$$

(a) Consider L2 norm as your loss function:

$$g(W) = \|X \cdot W - Y\|_2^2 = \sum_{i=1}^{n} (y_i - f(x_i; W))^2$$

please use the closed form solution to compute W and plot the scatter graph of data and estimated parabola. Report the parabola function and the figure

Actually it should be exactly the same form as the solution in section 1.1:

$$W^* = (X^T X)^{-1} X^T Y$$

(b) Consider L1 norm as your loss function:

$$g(W) = \|XW - Y\|_1 = \sum_{i=1}^{n} |y_i - f(x_i; W)|$$

Derive the gradient:

$$\frac{\partial g(W)}{\partial W} = ((\textbf{sign}(XW - Y))^T X)^T$$

Use the gradient descent method to compute W and plot the scatter graph of data and estimated parabola. Report the parabola function and the figures

$$\text{sign}(x) = \begin{cases} 1 & , \quad x > 0 \\ 0 & , \quad x = 0 \\ -1 & , \quad x < 0 \end{cases}$$

and if X is a matrix, sign(X) means performing element-wise sign(x_{ij}) over all element x_{ij} in X.

1.3. Perceptron Learning

First inspect the dataset to see if it is linearly separable. Then try (mini-batch) gradient descent and get the decision boundary. Finally get some metrics to evaluate the performance and also try z-score(normalization).

1.4. Feed Forward Neural Network

Try

1. 1 hidden layer
2. 2 hidden layers
3. 1 hidden layer with regularization and momentum

1.5. Convolutional Neural Network

We implemented the network using part of the structure of VGG16. Basically we kept the first 3 blocks out of 5 blocks in total because the dataset(cifar 10) is small and we didn't want to overfit. We first used SGD on different architecture, including:

1. fully-connected
2. global-average-pooling instead of fully-connected
3. add batch-normalization
4. add dropout

For the optimizers we also enumerated the following methods

1. Stochastic Gradient Descent(SGD)
2. Adaptive Gradient(Adagrad)
3. Nesterovs Accelerated Gradient(Nadam)
4. RMSprop

2. Experiments

2.1. Least Square Estimation

The estimated functions are as below in Figure 1:

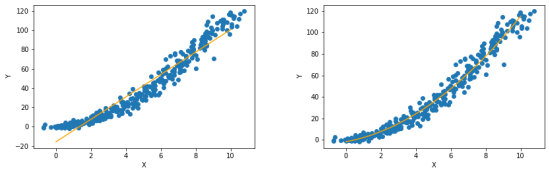

Figure 1. least square line(left) and least square parabola(right)

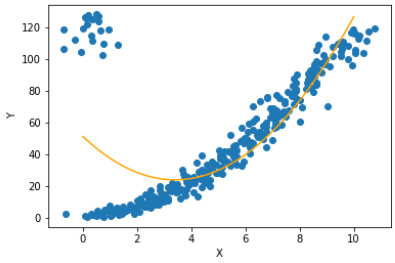

Figure 2. L_2 norm as loss function: outlines

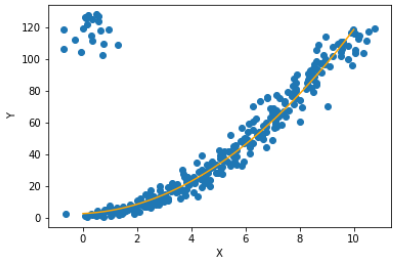

Figure 3. L_1 norm as loss function: outlines

2.2. Parabola Estimation

(a) L_2 norm loss function

The final parabola function is

$$Y = 51.07 + -16.06X + 2.36X^2$$

The plot is as below in Figure 2:

(b) L_1 norm loss function

The final parabola function is

$$Y = 2.48 + 0.77X + 1.08X^2$$

The plot is as below in Figure 3:

2.3. Perceptron Learning

2.3.1 Dataset

In Figure 4, we found that classes are linearly separable in each of the 2D feature spaces, when there are only 2 classes.

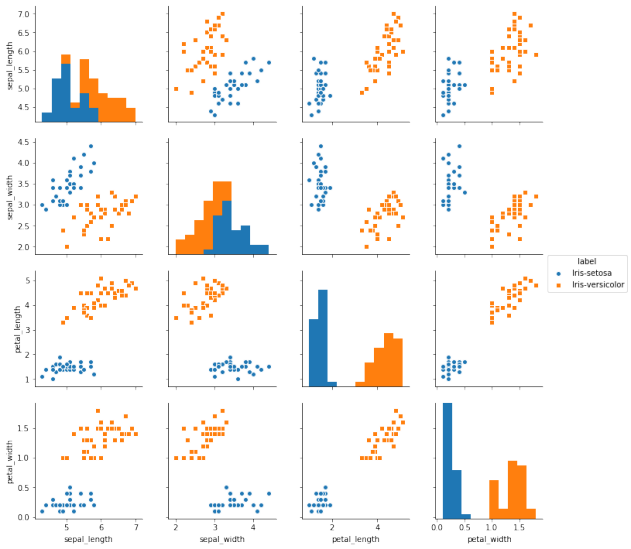

Figure 4. paired 2D feature spaces

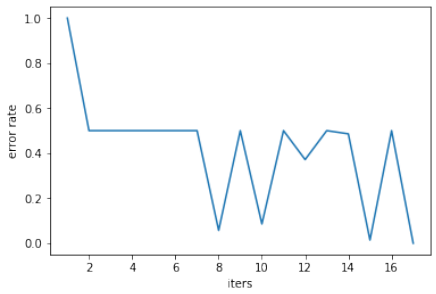

Figure 5. error rate during training

2.3.2 Programming

Plot the error rate on the training set during training as below in Figure 6
We used learning rate = 1 and after 17 iterations the error rate dropped to 0 so it converged.

2.3.3 Decision Boundary

$$Y = \begin{bmatrix} 2.14 \\ -15.07 \\ 7.66 \\ 6.91 \\ 2.32 \end{bmatrix}^T \begin{bmatrix} sepal\_length \\ sepal\_width \\ petal\_length \\ petal\_width \\ 1 \end{bmatrix}$$

2.3.4 Test

We report the metrics in Table 1

	origin	z-score
Accuracy	1	0.5
Precision	1	0.5
Recall	1	1
F-value	1	0.67

Table 1. test on perceptron learning

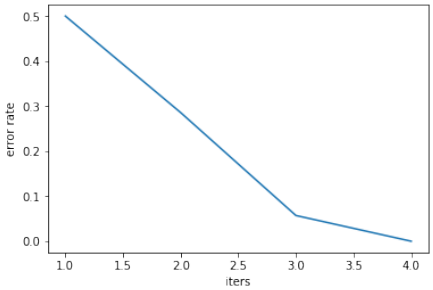

Figure 6. error rate during training(Z-score)

2.3.5 Z-score

After Z-score preprocessing, the iterations for convergence is fewer(4 other than 17, see Figure 6). The decision boundary becomes

$$Y = \begin{bmatrix} 5.46 \\ -3.56 \\ 1.39 \\ 0.23 \\ 3.19 \end{bmatrix}^T \begin{bmatrix} sepal\_length \\ sepal\_width \\ petal\_length \\ petal\_width \\ 1 \end{bmatrix}$$

So the coefficients are smaller. However the accuracy and precision have dropped(see Table 1).
Mainly because training set and test set has different feature distributions.

2.4. Feed Forward Neural Network

2.4.1 1 hidden layer

Network architecture:
Input layer, Fully-connected layer, Softmax, cross-entropy loss
Hyperparameters:
max_iterations = 1000, learning_rate = 1e-5. Loss w.r.t. iterations in Figure 7.

2.4.2 2 hidden layers

Network architecture:
Input layer, Fully-connected layer 1, Fully-connected layer 2, Softmax, cross-entropy loss
Hyperparameters:

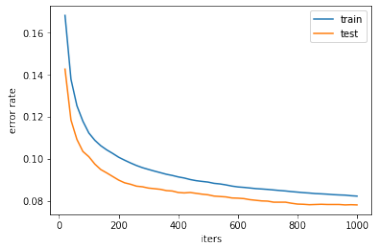

Figure 7. error rate during training(Z-score)

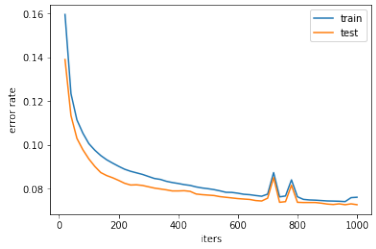

Figure 8. error rate during training(Z-score)

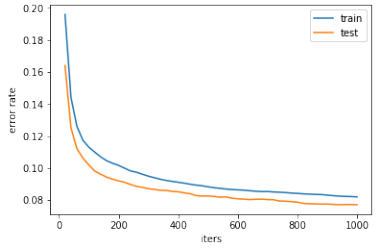

Figure 9. error rate during training(Z-score)

	1 layer	2 layers	1 layer plus
training loss	0.296	0.274	0.297
training error	0.082	0.076	0.082
test error	0.078	0.073	0.077

Table 2. Feed Forward Neural Network result

max_iterations = 1000, learning_rate = 1e-5. Loss w.r.t. iterations in Figure 8.

2.4.3 1 hidden layer with regularization and momentum

Network architecture:
Input layer, Fully-connected layer, Softmax, cross-entropy loss + regularization loss
Hyperparameters:
max_iterations = 1000, learning_rate = 1e-5. Loss w.r.t. iterations in Figure 9.

It seems that 1 layer plus can be slightly better than the origin one. 2 layers are better than only 1 layer. See Table 2.

2.5. Convolutional Neural Network

comment on the iterations for convergence, accuracy on the test set, plot of training and test loss with respect to the iterations. Also, report the network architecture

We used the structure of VGG16 network and keep the first 3 blocks out of the 5. Basically the structure is 2 convolution layers + 1 pooling layer, 2 convolution layers + 1 pooling layer and 3 convolution layers + 1 pooling layer, which are directly inherited from VGG. Afterwards we flatten the output and add 3 more fully connected layers. This is called ”fully-connected” in the Table 3.
There are some variation on the origin version. First is to replace the flatten processing with the global pooling, which is called ”global pooling”. Another version is add batch normalization after each convolution layer after the global pooling version(”+batch-normalization”). Also we added dropout layers after each pooling layer and each fully-connected layer after the batch-normalization version(”+dropout”).

Here we have 4 different optimizer: SGD: Stochastic gradient descent optimizer
RMSprop: running average of the magnitudes of recent gradients
Adagrad: Adaptive Subgradient - Adagrad optimizer
Nadam: Nesterov Adam optimizer
We used learning rate = 1e-3 for SGD, 1e-2 for Adagrad, 2e-3 for Nadam and 1e-3 for RMSprop.

The results are shown in Figure 10, 11 and Table 3. We found that in all the cases there is some kind of over-fitting. Simply use global pooling is hard to train, while adding bathc-normalization helps a lot. And also using dropout can perform well while using Adagrad and Nadam.

3. Discussion

For the first 2 parts we tried easy polynomial function fitting. And we can see that different versions of norm have different sensitivity to outliers. For the simple perceptron and feed forward neural network we tried the gradient descent and back propagation. Finally we used more complex convolution neural network to work on a little bit harder task and compared with different layer structure and optimizers. Choosing different structure can give different performances and good optimizer can accelerate the training process.

architecture	optimizer	train loss	test loss	train acc	test acc
fully-connected	SGD	0.0160	1.7450	0.9995	0.6794
global pooling	SGD	0.6510	0.9348	0.7730	0.6785
+batch-normalization	SGD	0.2180	0.9274	0.9459	0.7087
	RMSprop	0.0213	1.0131	0.9936	0.8341
+dropout	SGD	1.0658	1.3320	0.6236	0.5622
	RMSprop	0.0771	0.8476	0.9781	0.8480
	Adagrad	0.0719	0.6950	0.9779	0.8580
	Nadam	0.0663	0.6751	0.9794	0.8749

Table 3. Feed Forward Neural Network result

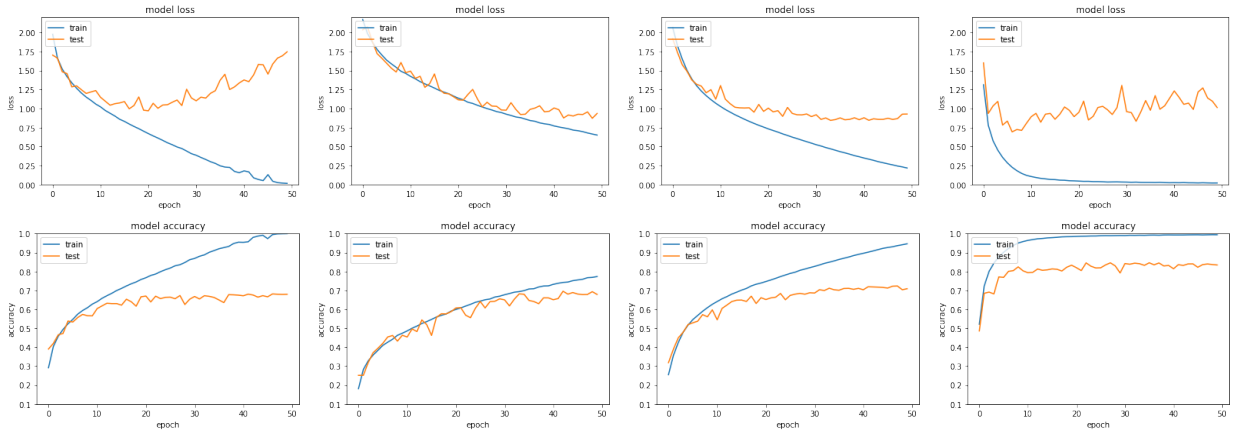

Figure 10. column1:fully-connected column2:globalpooling column3:batch-normalization SGD column4:batch-normalization RMSprop

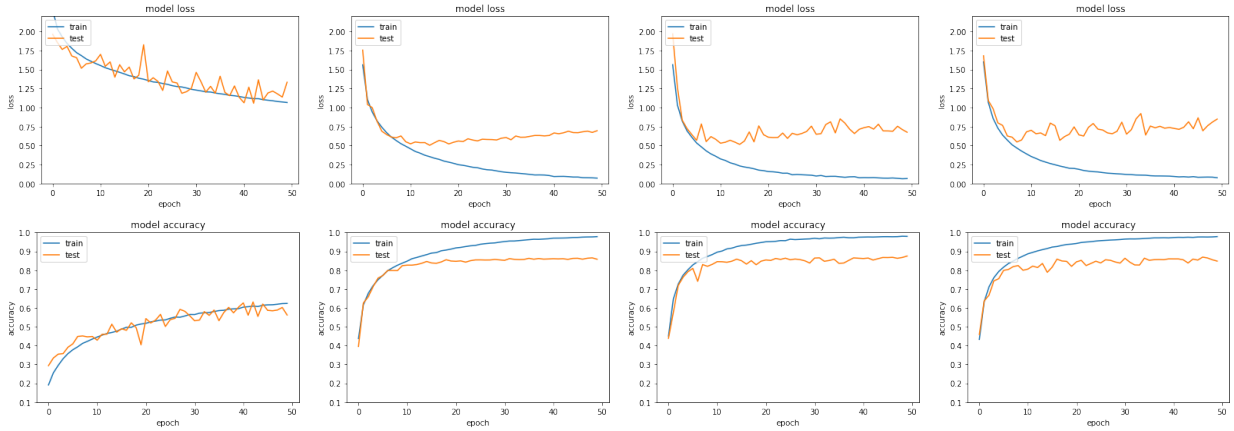

Figure 11. for dropout column1:SGD column2:Ada Gradient column3:NA Gradient column4:RMSprop