First Weekly Report

Jianyong Yuan AISIG SJTU

Version: 0.01

Last update: March 30, 2019

Abstract

Last week I focus on lesson cs231n and its homework1. As the contain is somehow repeat with cs229, so I just have a quick glance of the previous part and join the assignment 1.

1 What I Learnt

- `KNN` K-Nearest Neighbor.
- `SVM` Support Vector Machine.
- `SoftMax`
- `NN` Neural Network.

1.1 KNN

Maybe it's the easiest algorithm. Given a dataset, containing feature matrix X and the corresponding label vector y, we even don't need to train them. Instead, we just store them as X_{train} and y_{train}.
To predict newly given data $X_{predict}$ without label, firstly, we compute the distance between it and each X_{train} point (usually using L2 distance fomula).

$$distance = \|X_{train} - X_{predict}\|^2$$

Then, choose K points which are nearest the point we try to predict. And finally, find these K points' labels, and the one appears most frequently is our conclusion on the prediction.

$$y_{predict} = mode(y_{train}[min_k(distance)])$$

Though KNN is very simple, it works very well in practice. In some case such as recognizing hand-writing number, classifying users for insurance, we can consider KNN firstly for its simplicity and efficiency.

1.2 SVM

SVM is a commonly used loss function in classifying problems. Its core idea is to divide a significant boundary between the correct class and others.

So it is unstranged that its expression looks like this:

$$loss = max(0, f_{incorrect} - f_{correct} + \Delta)$$

Here f is the output of ωx^T or maybe others complicated expression. And from SVM on, we add regulization part to loss in order to punish the ω, making our algorithm more smooth.

$$loss+ = \lambda \sum_i \sum_j \omega_{ij}^2$$

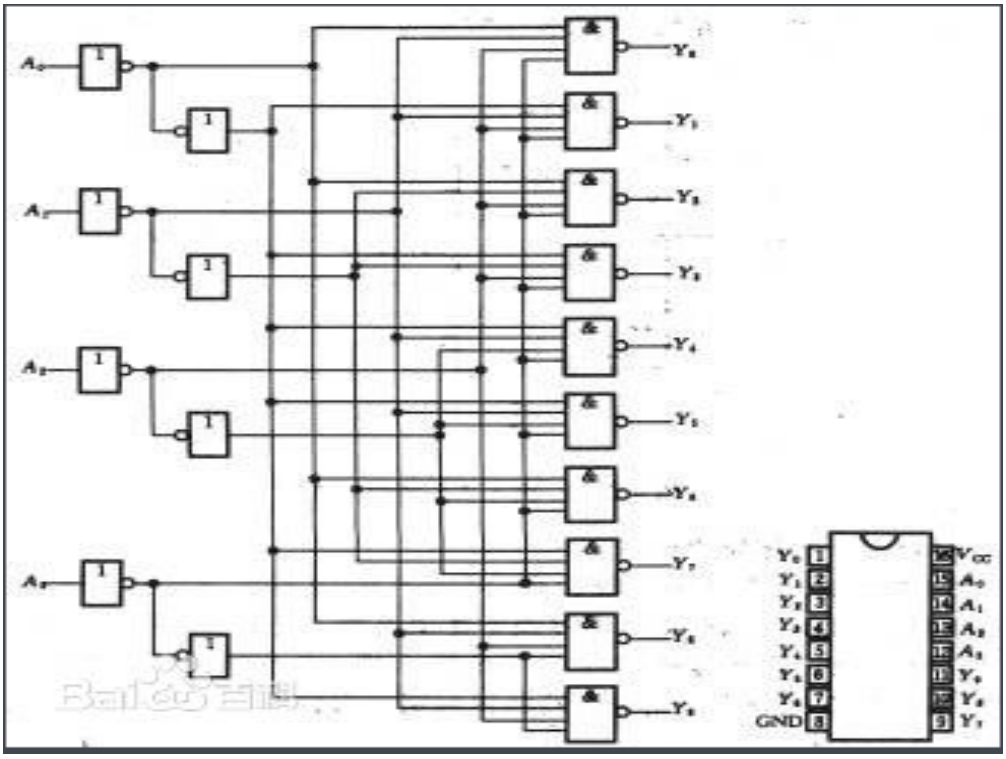
1.3 SoftMax

At the beginning, I often can't distinguish between SVM and SoftMax. In a word, SVM tries to draw a good boundary(Δ) between the correct class and others, which means that onces it has been satisfied, it doesn't care about the detail, while SoftMax is so greed that it always wants better. Its loss function looks like this:

$$loss = -log(\frac{e^{f_{correct}}}{\sum_j e^{f_j}}) + \lambda \sum_i \sum_j \omega_{ij}^2$$

1.4 Neural Network

I think this is the most interesting part and I spend most time on it. At first, I am really confused that why Neural Network makes sence, why hidden layer is needed and why we need nonlinear activation function. One day I found that it resembles the logic circuit I learnt last semester.



The weight matrix ω is like the lines which connects to the gate. The nonlinear activation function is like the gate in logic circuit. And the hidden layer is like the middle procedure of the circuit–it can be seemed as a black-box. In this figure, analogously, we can say there are four inputs, one hidden layer and ten output. (Maybe it's not accurate, after all, it's just an analogy.) If we don't use nonlinear activation function, no matter how many hidden layers we use, the output is always linear with the inputs. So nonlinear activation function plays a critical role in Neural Network.

Another inexplicable and important part of NN is back-propagation. (As it contains nonlinear activation function.) To understand it breifly and compute it more efficiently, we can make good use of the chain rule and we should have a deep understand on matrix derivation.

$$chainrule: \quad \frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial \omega_1}$$

$$simple\ matrix\ derivation: \quad f = \omega x \quad \frac{\partial f}{\partial \omega} = x^T$$

More complicated, given that (here I deliberately not use $h_1 = \omega_1 x$ to show the tiny difference. f() is an activation function. L() is a Loss function such as SVM or SoftMax.):

$$h_1 = x\omega_1$$

$$a_1 = f(h_1)$$

$$h_2 = a_1\omega_2$$

$$loss = L(h_2, y)$$

Then how could we compute the back-propagation error?

First we should use special way to compute $\frac{\partial loss}{\partial h_2} = \frac{\partial L}{\partial h_2}$ for specific loss function.

Then:

$$\frac{\partial loss}{\partial \omega_2} = \frac{\partial loss}{\partial h_2}\frac{\partial h_2}{\partial \omega_2} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial \omega_2} = a_1^T\frac{\partial L}{\partial h_2}$$

If $h_2 = \omega_2 a_1$, it will be

$$\frac{\partial loss}{\partial \omega_2} = \frac{\partial loss}{\partial h_2}\frac{\partial h_2}{\partial \omega_2} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial \omega_2} = \frac{\partial L}{\partial h_2}a_1^T$$

And

$$\frac{\partial loss}{\partial a_1} = \frac{\partial loss}{\partial h_2}\frac{\partial h_2}{\partial a_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_1} = \frac{\partial L}{\partial h_2}\omega_2^T$$

$$\frac{\partial loss}{\partial h_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_1}\frac{\partial a_1}{\partial h_1} = \frac{\partial L}{\partial h_2}\omega_2^T\frac{\partial f}{\partial h_1}$$

$$\frac{\partial loss}{\partial \omega_1} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_1}\frac{\partial a_1}{\partial h_1}\frac{\partial h_1}{\partial \omega_1} = x^T\frac{\partial L}{\partial h_2}\omega_2^T\frac{\partial f}{\partial h_1}$$

OK, that's all annoying part of Neural Network.

1.5 More about Neural Network

Optimization Ways

- Nesterov momentum update (look forward)
- Learning rate decay (let learning_rate multiply a number such as 0.95 each epoch)
- Newton's way (use second derivative to decide how much each step to go, replacing the learning_rate)
- Per-parameter adaptive learning rate methods
  - Adagrad
  - RMSprop
  - Adam

```
# Assume the gradient dx and parameter vector x
Adagrad:
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)

RMSprop:
cache = decay_rate*cache + (1 - decay_rate) * dx**2
x += -learning_rate * dx / (np.sqrt(cache) + eps)
```

```
Adam:
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += -learning_rate * m / (np.sqrt(v) + eps)
```

- Hyperparameter optimization (by using small simple to try each combination of hyperparameters and compair their performances)

1.6 Neural Network Training Process

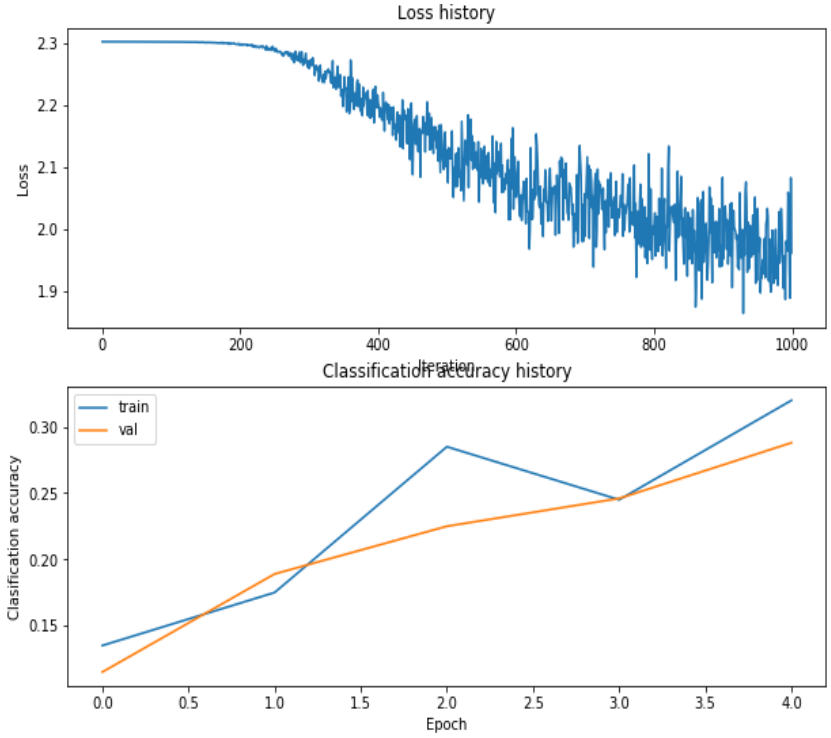


figure1 loss and the accuracy

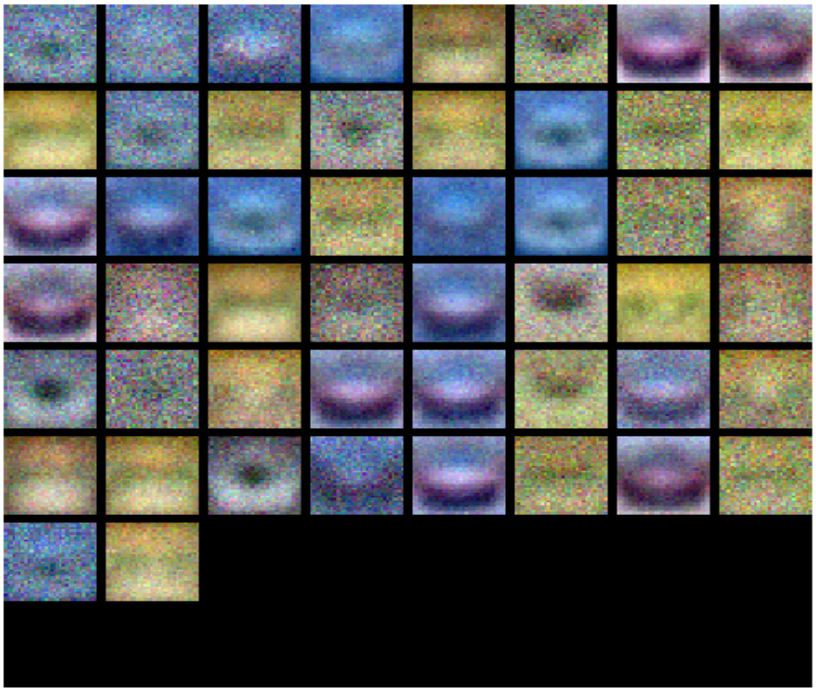


figure2 visualizing ω_1

```
learning rate: 0.0001, learning_rate_decay: 0.95,  regulization_strength: 0.5
Validation accuracy:  0.284
learning rate: 0.0001, learning_rate_decay: 0.95,  regulization_strength: 1
Validation accuracy:  0.277
learning rate: 0.0001, learning_rate_decay: 0.5,  regulization_strength: 0.25
Validation accuracy:  0.167
learning rate: 0.0001, learning_rate_decay: 0.5,  regulization_strength: 0.5
Validation accuracy:  0.195
learning rate: 0.0001, learning_rate_decay: 0.5,  regulization_strength: 1
Validation accuracy:  0.151
learning rate: 0.0005, learning_rate_decay: 0.95,  regulization_strength: 0.25
Validation accuracy:  0.441
learning rate: 0.0005, learning_rate_decay: 0.95,  regulization_strength: 0.5
Validation accuracy:  0.44
learning rate: 0.0005, learning_rate_decay: 0.95,  regulization_strength: 1
Validation accuracy:  0.444
learning rate: 0.0005, learning_rate_decay: 0.5,  regulization_strength: 0.25
Validation accuracy:  0.319
learning rate: 0.0005, learning_rate_decay: 0.5,  regulization_strength: 0.5
Validation accuracy:  0.309
learning rate: 0.0005, learning_rate_decay: 0.5,  regulization_strength: 1
Validation accuracy:  0.314
learning rate: 0.001, learning_rate_decay: 0.95,  regulization_strength: 0.25
Validation accuracy:  0.459
learning rate: 0.001, learning_rate_decay: 0.95,  regulization_strength: 0.5
Validation accuracy:  0.461
learning rate: 0.001, learning_rate_decay: 0.95,  regulization_strength: 1
Validation accuracy:  0.451
learning rate: 0.001, learning_rate_decay: 0.5,  regulization_strength: 0.25
Validation accuracy:  0.394
learning rate: 0.001, learning_rate_decay: 0.5,  regulization_strength: 0.5
Validation accuracy:  0.395
learning rate: 0.001, learning_rate_decay: 0.5,  regulization_strength: 1
Validation accuracy:  0.386
```

figure3 trying combination of hyperparameters

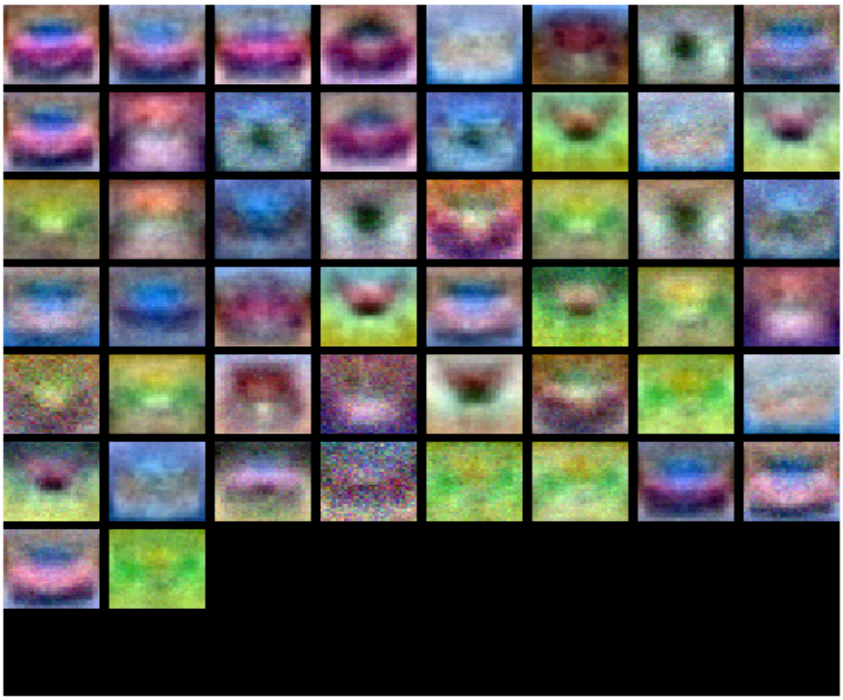


figure4 visualizing the best Network's ω_1

2 What I Am Confused

I am really confused about the Per-parameter adaptive learning rate methods although I read some paper to understand them. Maybe that's because I haven't use them in practice. Hope that I will be familar with these algorithm some day. Another thing is that I am not familar with latex (is there a standard style for a paper?), some python package like matplotlib and that my English is so poor. . . (I often confuse which word to use in order to make my paper look more professional and I have such a limited volcabulary that I frequently turn to Google)
But I still think writing such an English report is so cool!

3 What 's My Plan

I am going on cs231n, and I would like to begin the assignment2. I had spent so much time on ass1. As I get more familar with python and the environment, I believe I can go faster.

4 Conclusion

Deep learning is so cool, I often admire those who can come up with such amazing ideas, such as CNN. Can I take a seat one day? So just study hard and make a little progress every day.