CV作业三

姓名：楚逸飞

学号：2020302878

任务一：线性分类器图像分类

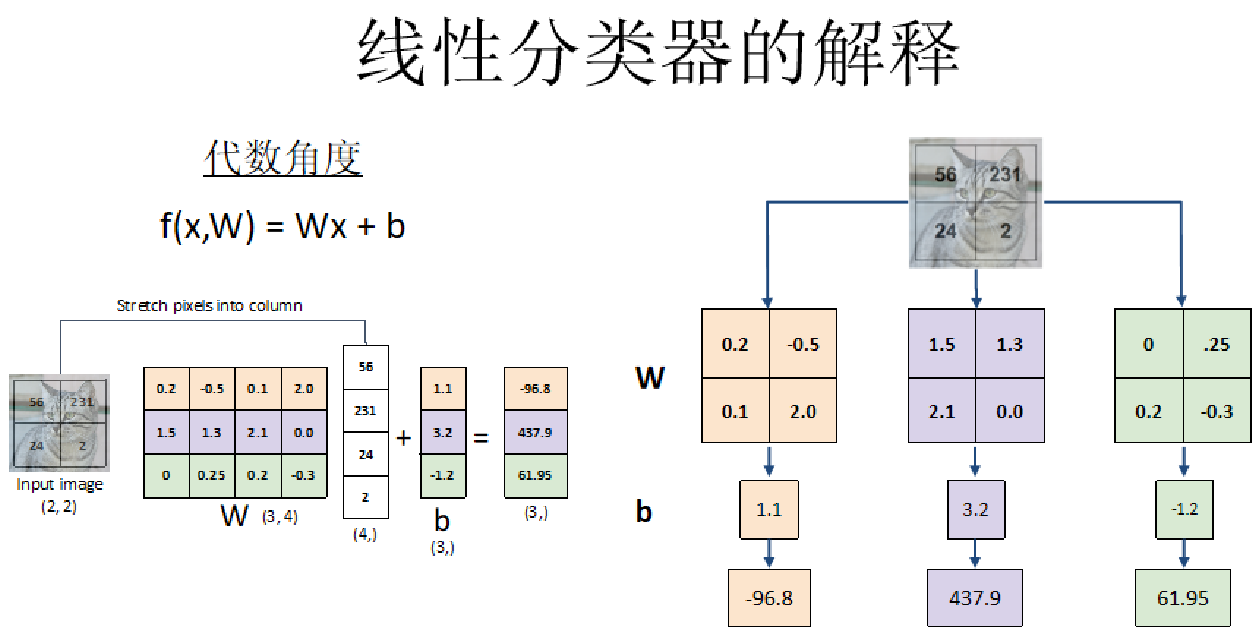
数据和要求：

数据：cifar10数据集

要求： 构建一个线性分类器实现cifar10数据分类，
其中 损失函数：交叉熵函数
优化方法：随机梯度速降法
1.使用精确率和召回率评估分类性能；
2.对数据进行划分，交叉验证评估分类性能。

实验原理：

1.线性分类器：

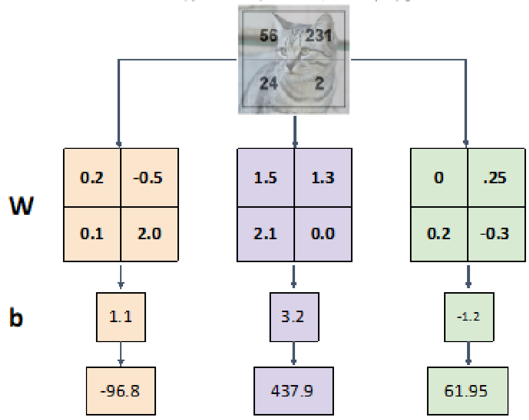


线性分类器的解释：可视化角度

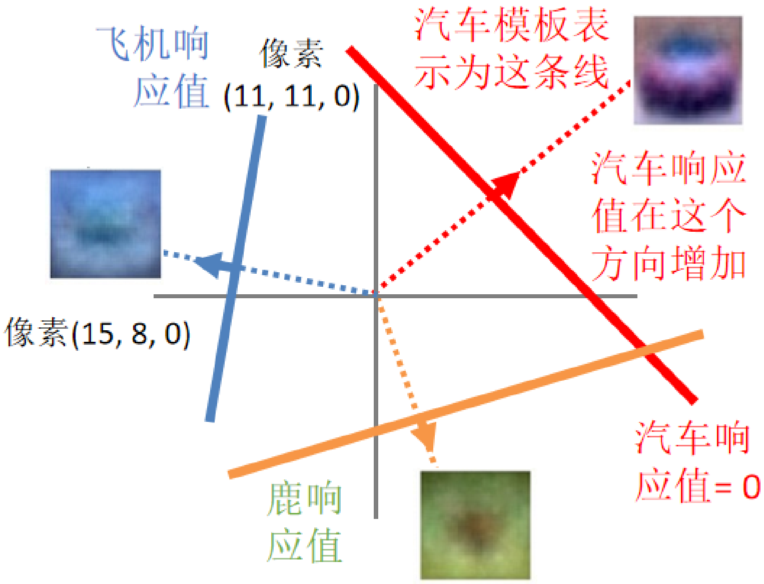
线性分类器每个类别
有一个模板

单一模板无法建模多模式数据

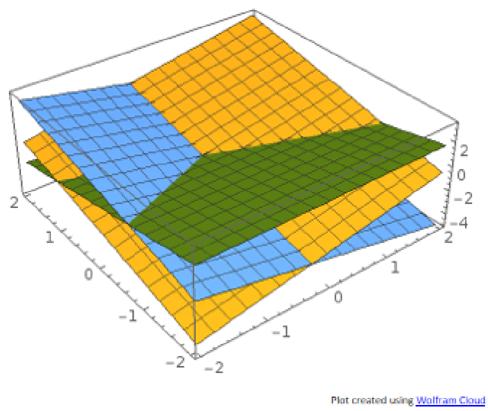
例. 马的模板有两个头!



线性分类器的解释：几何角度



分割高维空间的超平面



损失函数

损失函数告诉我们当前的分类器有多好

低损失= 好分类器
高损失= 坏分类器

(同样称为: 目标函数; 代价函数)

负向损失函数也被称为**报酬函数, 利润函数, 效用函数, 拟合度函数**, 等等

给定数据集

$$\{(x_i, y_i)\}_{i=1}^{N}$$

这里 x_i 对应图像
y_i 对应标签

一个样例的损失对应为

$$L_i(f(x_i, W), y_i)$$

最终损失为各样例损失的平均值:

$$L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)$$

多类 SVM 损失



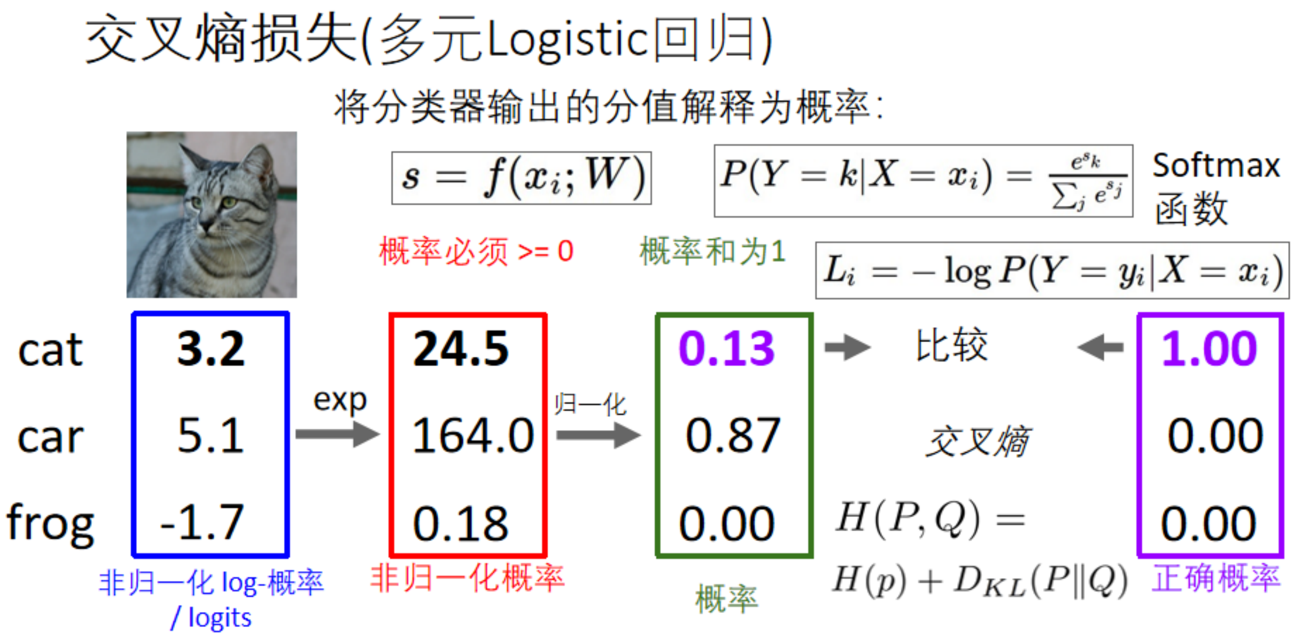
cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

给定一副图像 (x_i, y_i)
(x_i 为图像, y_i 为标签)

令 $s = f(x_i, W)$为响应分值

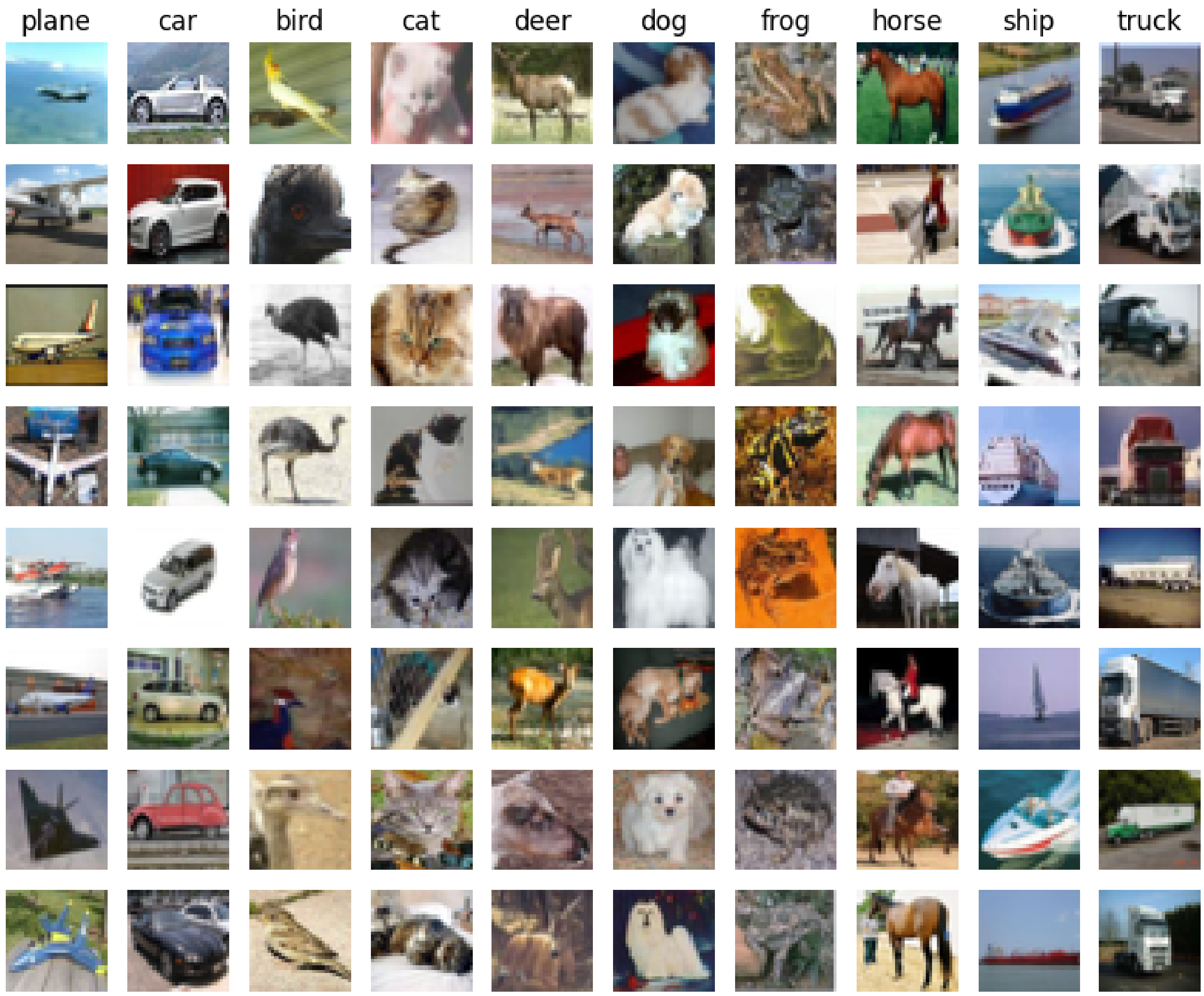
则SVM 损失定义为:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



实验结果与分析：

随机展示一部分cifar10数据集图片：



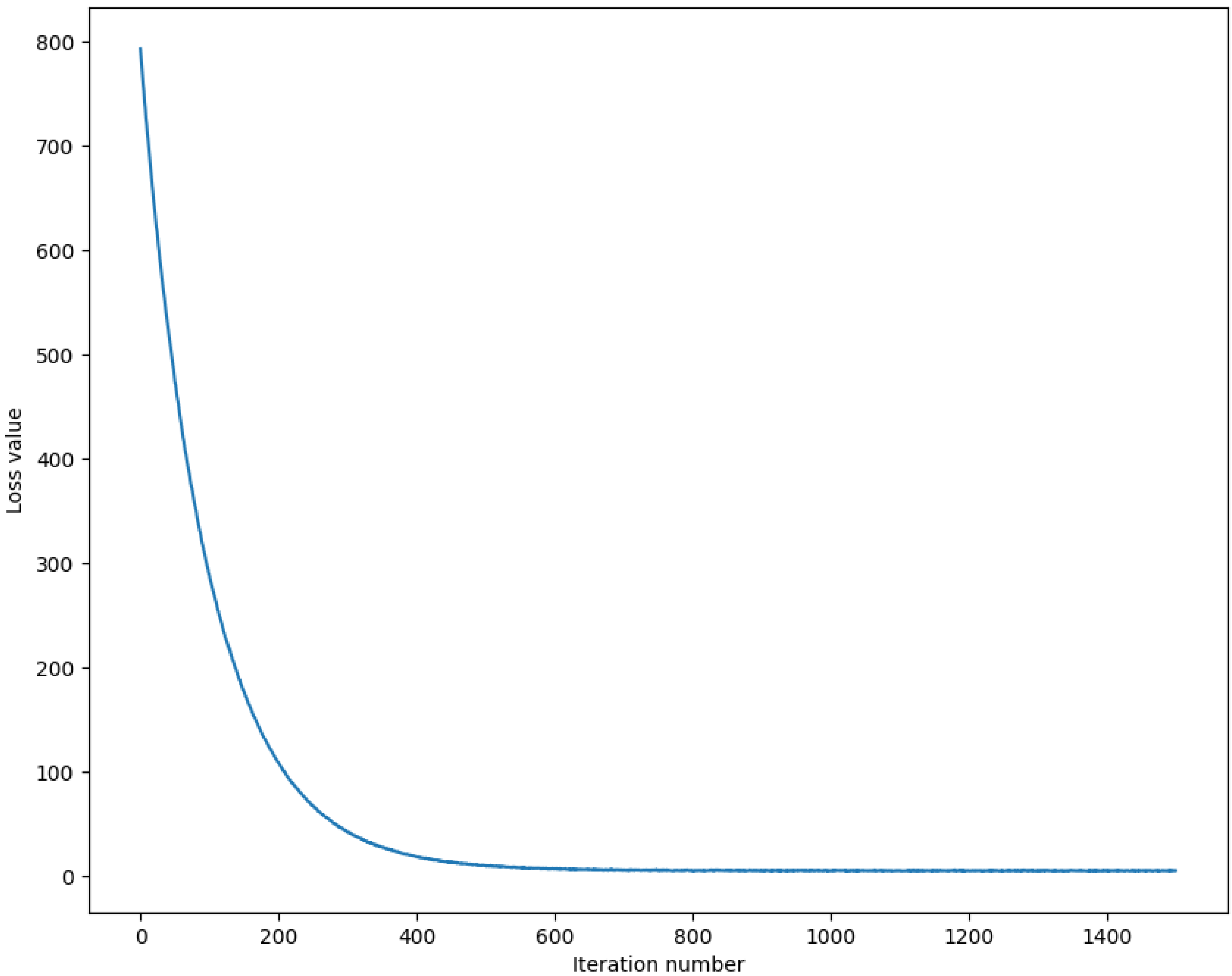
Loss收敛情况：

Iteration 0 / 1500: loss 786.185813
Iteration 100 / 1500: loss 288.658004
Iteration 200 / 1500: loss 109.034157
Iteration 300 / 1500: loss 43.267777
Iteration 400 / 1500: loss 19.184710
Iteration 500 / 1500: loss 9.985912
Iteration 600 / 1500: loss 7.019733
Iteration 700 / 1500: loss 5.583328
Iteration 800 / 1500: loss 5.167600
Iteration 900 / 1500: loss 5.081772
Iteration 1000 / 1500: loss 4.937401
Iteration 1100 / 1500: loss 5.479632
Iteration 1200 / 1500: loss 4.845214
Iteration 1300 / 1500: loss 5.641187
Iteration 1400 / 1500: loss 5.214697
Iteration 0 / 1500: loss 1544.081823
Iteration 100 / 1500: loss 208.792519
Iteration 200 / 1500: loss 32.179351
Iteration 300 / 1500: loss 9.550682
Iteration 400 / 1500: loss 6.222487
Iteration 500 / 1500: loss 5.641604
Iteration 600 / 1500: loss 5.642791
Iteration 700 / 1500: loss 5.577313
Iteration 800 / 1500: loss 5.214214
Iteration 900 / 1500: loss 5.814737
Iteration 1000 / 1500: loss 5.601292
Iteration 1100 / 1500: loss 5.016984
Iteration 1200 / 1500: loss 5.553018
Iteration 1300 / 1500: loss 6.030728
Iteration 1400 / 1500: loss 6.191609
Iteration 0 / 1500: loss 785.825779
Iteration 100 / 1500: loss over
Iteration 200 / 1500: loss over
Iteration 300 / 1500: loss over
Iteration 400 / 1500: loss over
Iteration 500 / 1500: loss over
Iteration 600 / 1500: loss over
Iteration 700 / 1500: loss over
Iteration 800 / 1500: loss over
Iteration 900 / 1500: loss inf
Iteration 1000 / 1500: loss inf
Iteration 1100 / 1500: loss inf
Iteration 1200 / 1500: loss inf
Iteration 1300 / 1500: loss inf
Iteration 1400 / 1500: loss inf

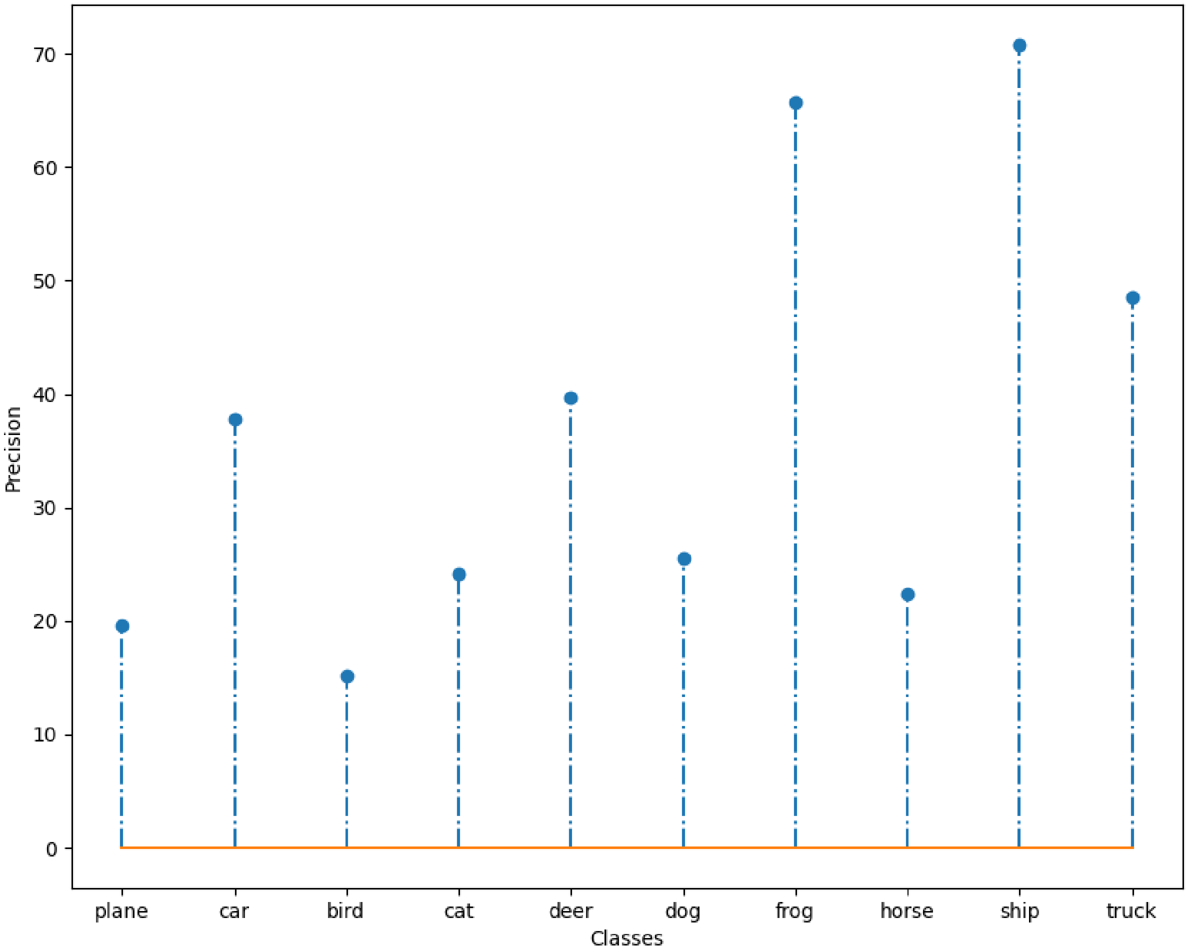
Iteration 0 / 1500: loss 1586.019987
Iteration100 / 1500: loss over
Iteration 200 / 1500: loss over
Iteration 300 / 1500: loss inf
Iteration 400 / 1500: loss inf
Iteration 500 / 1500: loss inf

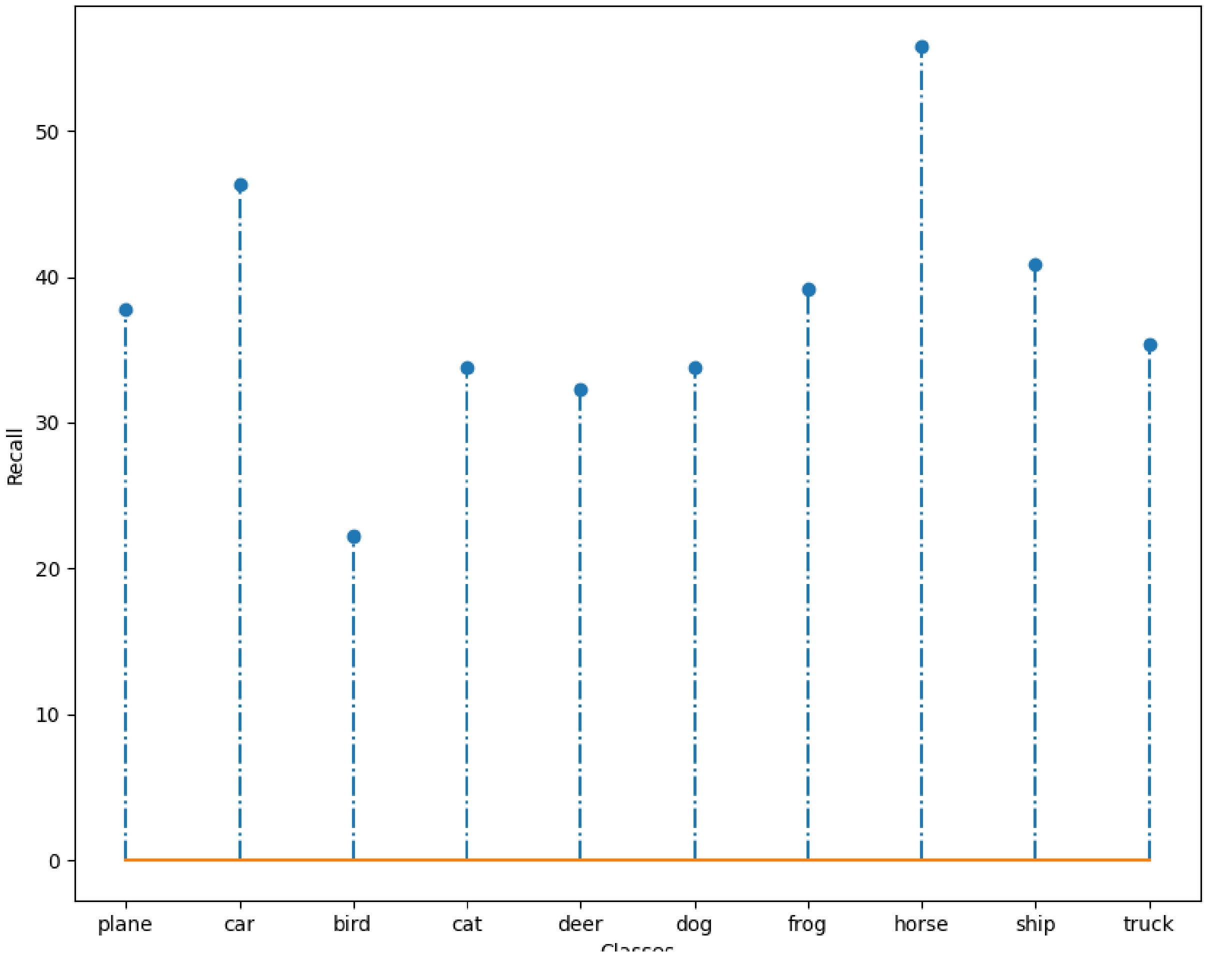
Iteration 600 / 1500: loss nan
Iteration 700 / 1500: loss nan
Iteration 800 / 1500: loss nan
Iteration 900 / 1500: loss nan
Iteration 1000 / 1500: loss nan
Iteration 1100 / 1500: loss nan
Iteration 1200 / 1500: loss nan
Iteration 1300 / 1500: loss nan
Iteration 1400 / 1500: loss nan

Iteration 0 / 1500: loss 793.377013
Iteration 100 / 1500: loss 289.058912
Iteration 200 / 1500: loss 108.570926
Iteration 300 / 1500: loss 42.434866
Iteration 400 / 1500: loss 18.715338
Iteration 500 / 1500: loss 10.399240
Iteration 600 / 1500: loss 7.235696
Iteration 700 / 1500: loss 6.072553
Iteration 800 / 1500: loss 4.942468
Iteration 900 / 1500: loss 5.843540
Iteration 1000 / 1500: loss 5.358267
Iteration 1100 / 1500: loss 5.543909
Iteration 1200 / 1500: loss 4.981700
Iteration 1300 / 1500: loss 4.886463
Iteration 1400 / 1500: loss 4.828447



精确度和召回度:


Precision
70
60
50
40
30
20
10
0
plane
car
bird
cat
deer
dog
frog
horse
ship
truck
Classes



实验中，训练集用来确定模型中的weights和biases；训练后使用验证集检验模型分类的结果，验证集只是为了确定hyper-params；最后的测试集是在整个训练都完成后评价模型效果。

在train和val的时候表现好并不代表test的时候表现也会好，因为不断地划分训练集和验证集，模型已经记住了这一些数据的特点，因而表现一般会比较好，但实际上模型并不一定真正学到了这些数据特征背后的规律，即模型可能只是对数据的“死记硬背”，这样的模型是一无是处的，在test时表现不会很好。

任务二：神经网络图像分类

数据和要求：

数据：cifar10数据集

要求： 构建一个包含两个线性层的前向神经网络模型进行数据分类，

其中 隐藏层节点：100

激活函数：第一层，线性修正单元，第二层，softmax函数

损失函数：交叉熵函数

优化方法：分批随机梯度速降法

1.使用精确率和召回率评估分类性能；

2.调节算法某一参数（参数包括批大小，学习速率，迭代轮数，隐藏层节点等），进行分类性能对比，画出性能随参数变化曲线。

实验原理：

神经网络

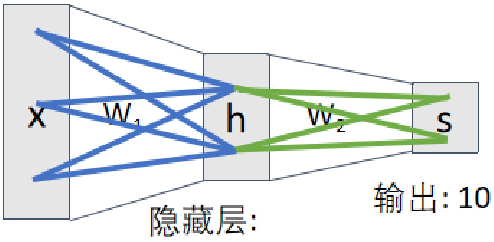
(前面) 线性响应函数: $f = Wx$

(现在) 2-层神经网络 $f = W_2 \max(0, W_1 x)$

W_1元素 (i, j) 给出了x_j对h_i的影响

x的所有元素影响h的所有元素

输入: 3072



隐藏层: 100

输出: 10

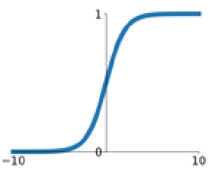
W_2元素 (i, j)给出了h_j对s_i影响

h的所有元素影响s的所有元素

全连接神经网络
也称为 “多层感知器” (MLP)

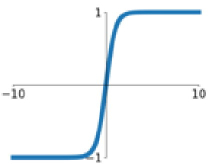
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



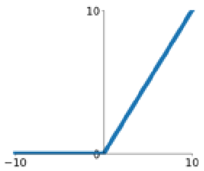
tanh

$\tanh(x)$



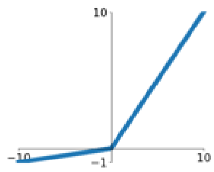
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

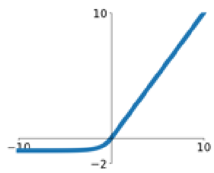


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

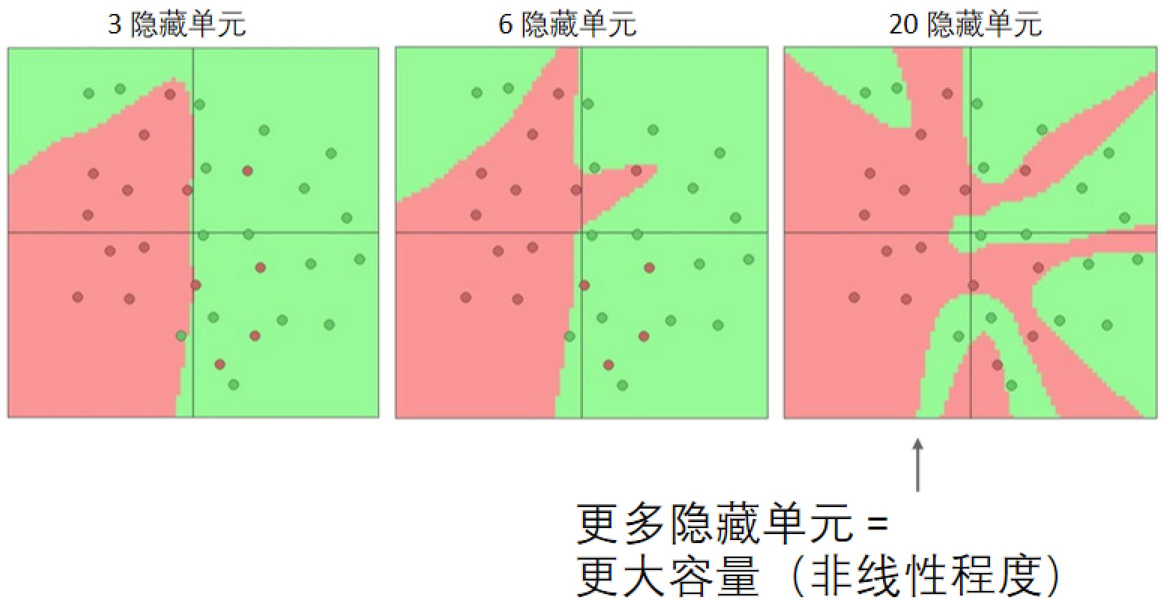
ELU

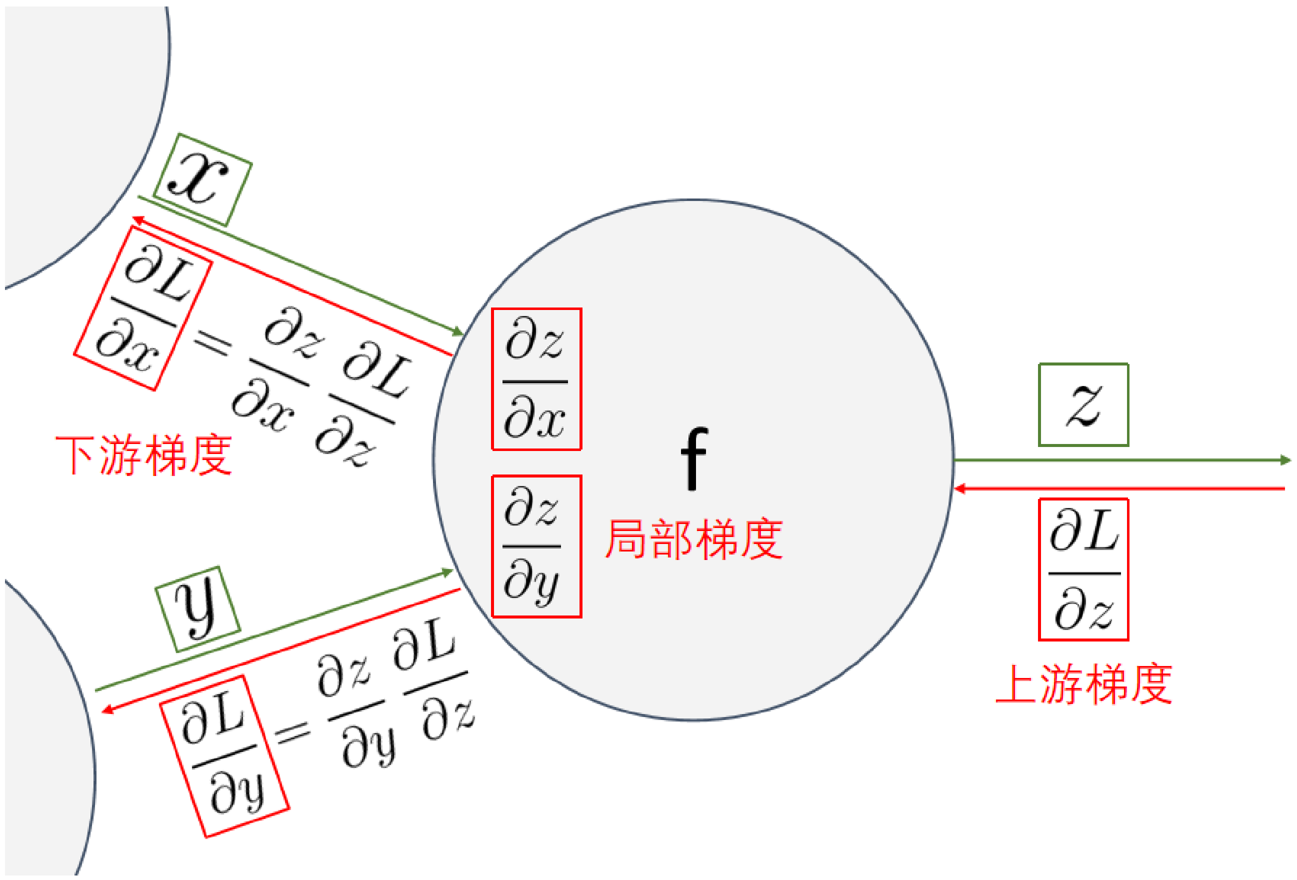
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



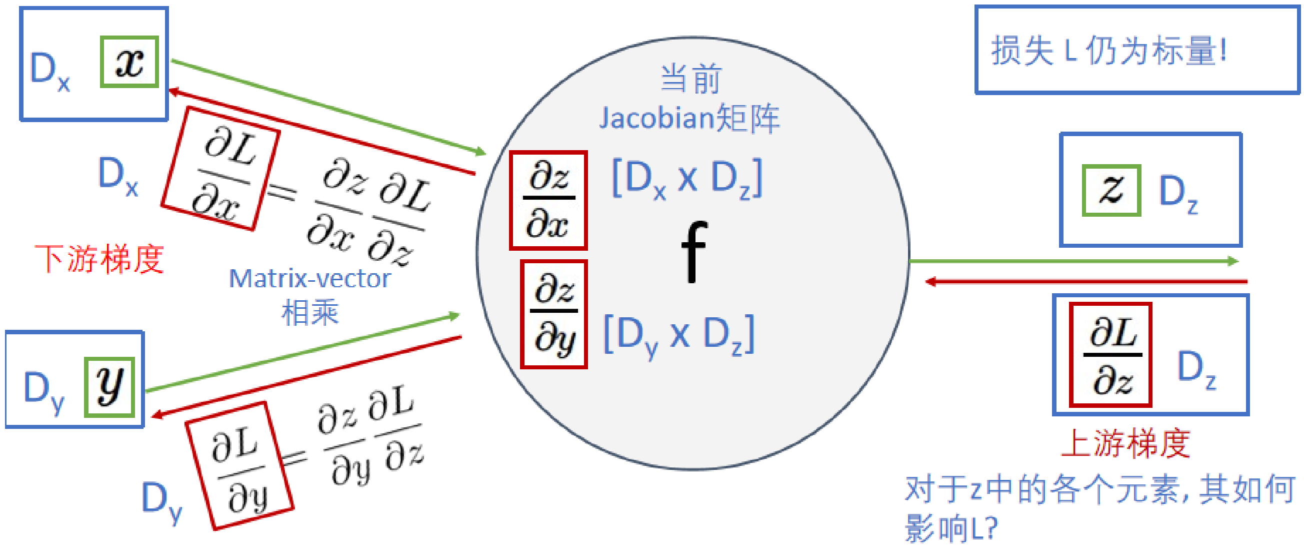
(对于大部分问题ReLU是很好的默认选择)

层数和大小设置


3 隐藏单元
6 隐藏单元
20 隐藏单元
更多隐藏单元 =
更大容量（非线性程度）


x
$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial L}{\partial z}$
下游梯度
y
$\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial L}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
局部梯度
z
$\frac{\partial L}{\partial z}$
上游梯度

矢量函数后向传播



实验结果与分析：

①Linear layer forward：

```
Testing Linear.forward function:
difference:  3.6830429120909964e-08
```

Linear layer backward：

```
Testing Linear.backward function:
dx error:   5.170864758949246e-10
dw error:   3.293741405059995e-10
db error:   5.373171200544344e-10
```

②ReLU forward：

```
Testing ReLU.forward function:
difference:  4.5454545613554664e-09
```

ReLU backward：

```
Testing ReLU.backward function:
dx error:  2.6317796097761553e-10
```

③Sandwich layers：

```
Testing Linear_ReLU.forward and Linear_ReLU.backward:
dx error:  1.4564739946431168e-09
dw error:  6.424556508465271e-10
db error:  8.915028842081707e-10
```

④Loss layers （Softmax and SVM）：

```
Testing svm_loss:
loss:  9.000430792478463
dx error:  7.97306008441663e-09

Testing softmax_loss:
loss:  2.3026286102347924
dx error:  1.0417990899757076e-07
```

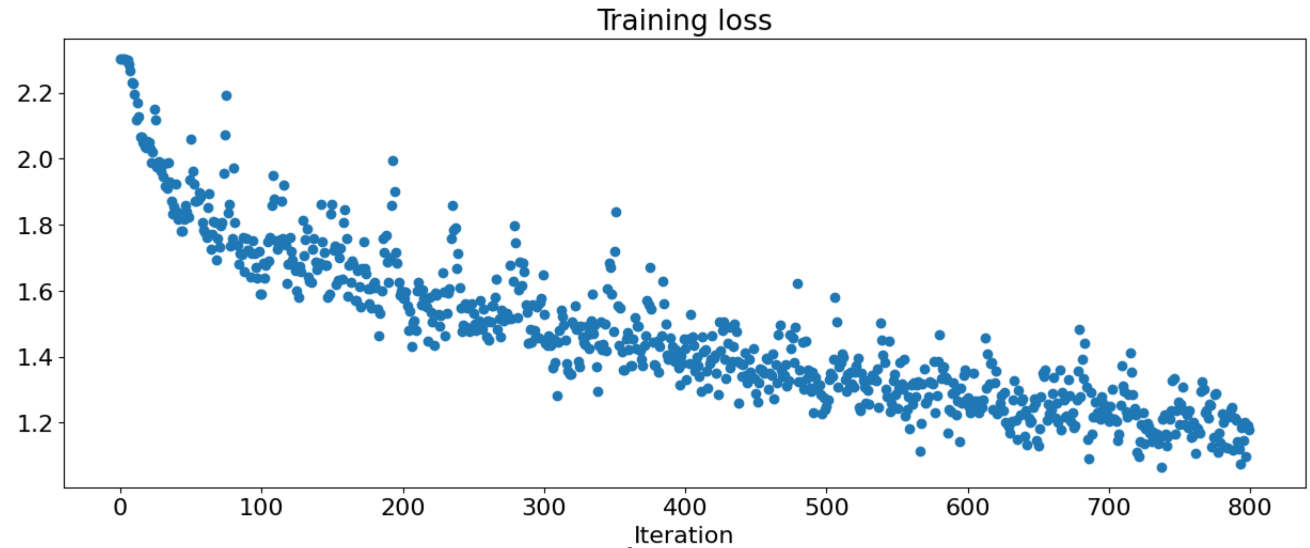
⑤2-layer network：

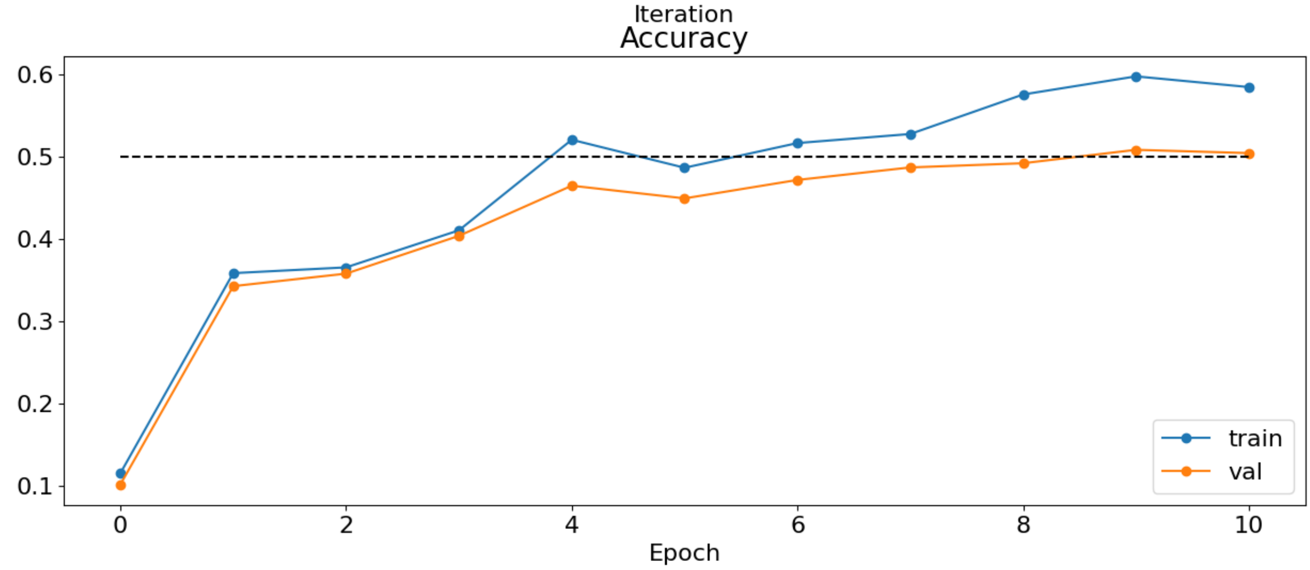
```
Testing initialization ...
Testing test-time forward pass ...
Testing training loss (no regularization)
Running numeric gradient check with reg =  0.0
W1 relative error: 2.28e-07
W2 relative error: 1.45e-09
b1 relative error: 9.91e-07
b2 relative error: 4.99e-09
Running numeric gradient check with reg =  0.7
W1 relative error: 3.16e-08
W2 relative error: 8.40e-09
b1 relative error: 6.27e-07
b2 relative error: 2.72e-08
```

⑥Solver：

train and val：

```
(Time 0.03 sec; Iteration 1 / 800) loss: 2.302587
(Epoch 0 / 10) train acc: 0.115000; val_acc: 0.101400
(Epoch 1 / 10) train acc: 0.358000; val_acc: 0.342200
(Epoch 2 / 10) train acc: 0.365000; val_acc: 0.357400
(Epoch 3 / 10) train acc: 0.410000; val_acc: 0.403300
(Epoch 4 / 10) train acc: 0.520000; val_acc: 0.464200
(Epoch 5 / 10) train acc: 0.486000; val_acc: 0.448800
(Epoch 6 / 10) train acc: 0.516000; val_acc: 0.471200
(Time 8.14 sec; Iteration 501 / 800) loss: 1.250751
(Epoch 7 / 10) train acc: 0.527000; val_acc: 0.486400
(Epoch 8 / 10) train acc: 0.575000; val_acc: 0.491500
(Epoch 9 / 10) train acc: 0.597000; val_acc: 0.507800
(Epoch 10 / 10) train acc: 0.584000; val_acc: 0.503800
```



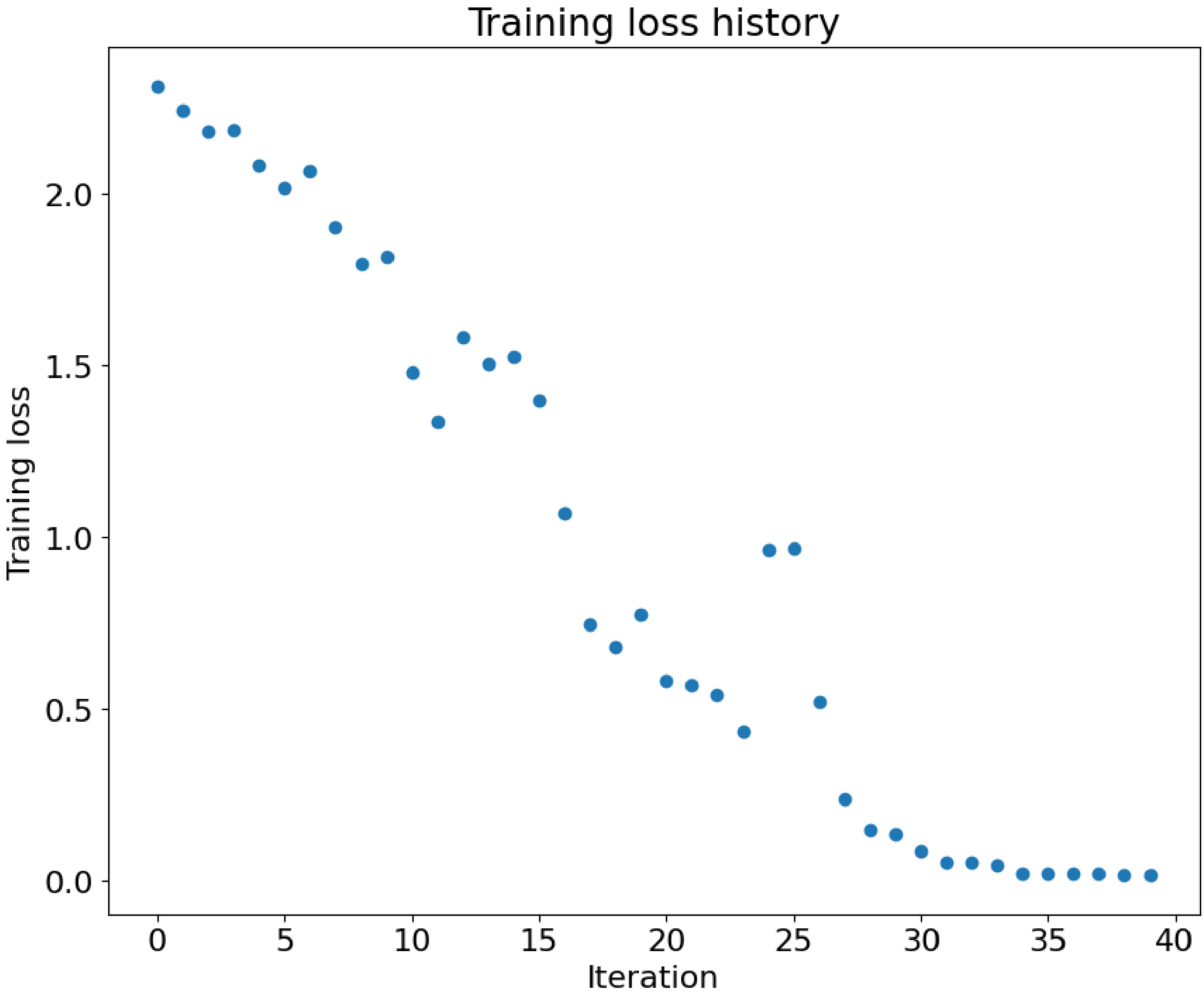


⑦Multilayer network：

```
Running check with reg =  0
Initial loss:  2.307533872340282
W1 relative error: 4.71e-08
W2 relative error: 9.16e-08
W3 relative error: 4.72e-08
b1 relative error: 9.39e-08
b2 relative error: 2.42e-08
b3 relative error: 2.39e-09
Running check with reg =  3.14
Initial loss:  7.063988454673523
W1 relative error: 7.86e-09
W2 relative error: 9.95e-09
W3 relative error: 1.05e-08
b1 relative error: 6.20e-07
b2 relative error: 1.00e-07
b3 relative error: 6.20e-09
```

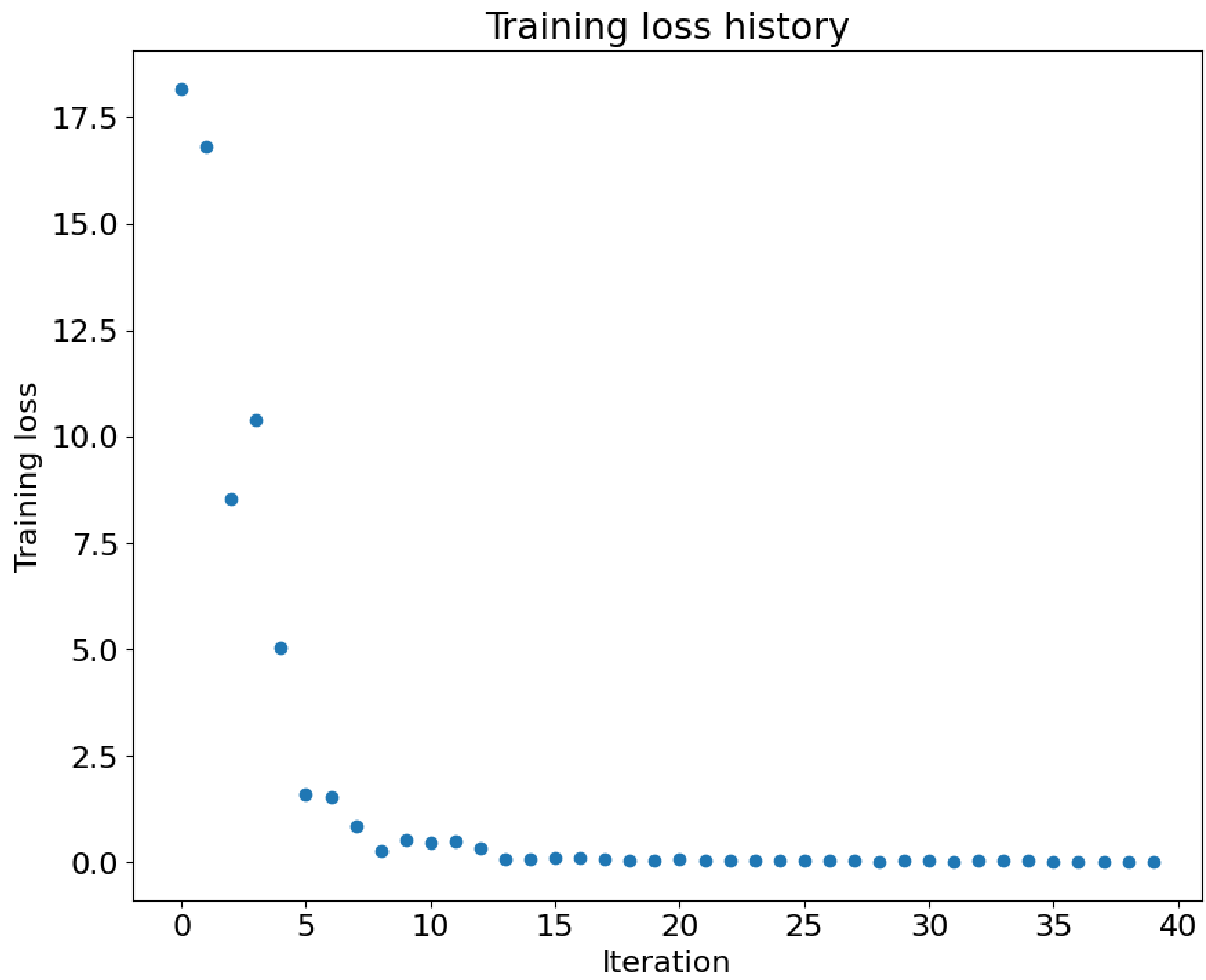
training loss：

```
(Time 0.00 sec; Iteration 1 / 40) loss: 2.312459
(Epoch 0 / 20) train acc: 0.320000; val_acc: 0.108300
(Epoch 1 / 20) train acc: 0.360000; val_acc: 0.114800
(Epoch 2 / 20) train acc: 0.240000; val_acc: 0.097100
(Epoch 3 / 20) train acc: 0.300000; val_acc: 0.145000
(Epoch 4 / 20) train acc: 0.440000; val_acc: 0.140200
(Epoch 5 / 20) train acc: 0.540000; val_acc: 0.171000
(Time 0.37 sec; Iteration 11 / 40) loss: 1.481615
(Epoch 6 / 20) train acc: 0.380000; val_acc: 0.160700
(Epoch 7 / 20) train acc: 0.440000; val_acc: 0.128800
(Epoch 8 / 20) train acc: 0.720000; val_acc: 0.170300
(Epoch 9 / 20) train acc: 0.740000; val_acc: 0.191900
(Epoch 10 / 20) train acc: 0.860000; val_acc: 0.187400
(Time 0.56 sec; Iteration 21 / 40) loss: 0.581121
(Epoch 11 / 20) train acc: 0.840000; val_acc: 0.137600
(Epoch 12 / 20) train acc: 0.700000; val_acc: 0.147300
(Epoch 13 / 20) train acc: 0.840000; val_acc: 0.196500
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.195600
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.199900
(Time 0.76 sec; Iteration 31 / 40) loss: 0.086812
(Epoch 16 / 20) train acc: 0.980000; val_acc: 0.193100
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.197500
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.197700
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.196600
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.197300
```



Training loss history
Training loss
Iteration
0.0
0.5
1.0
1.5
2.0
0
5
10
15
20
25
30
35
40

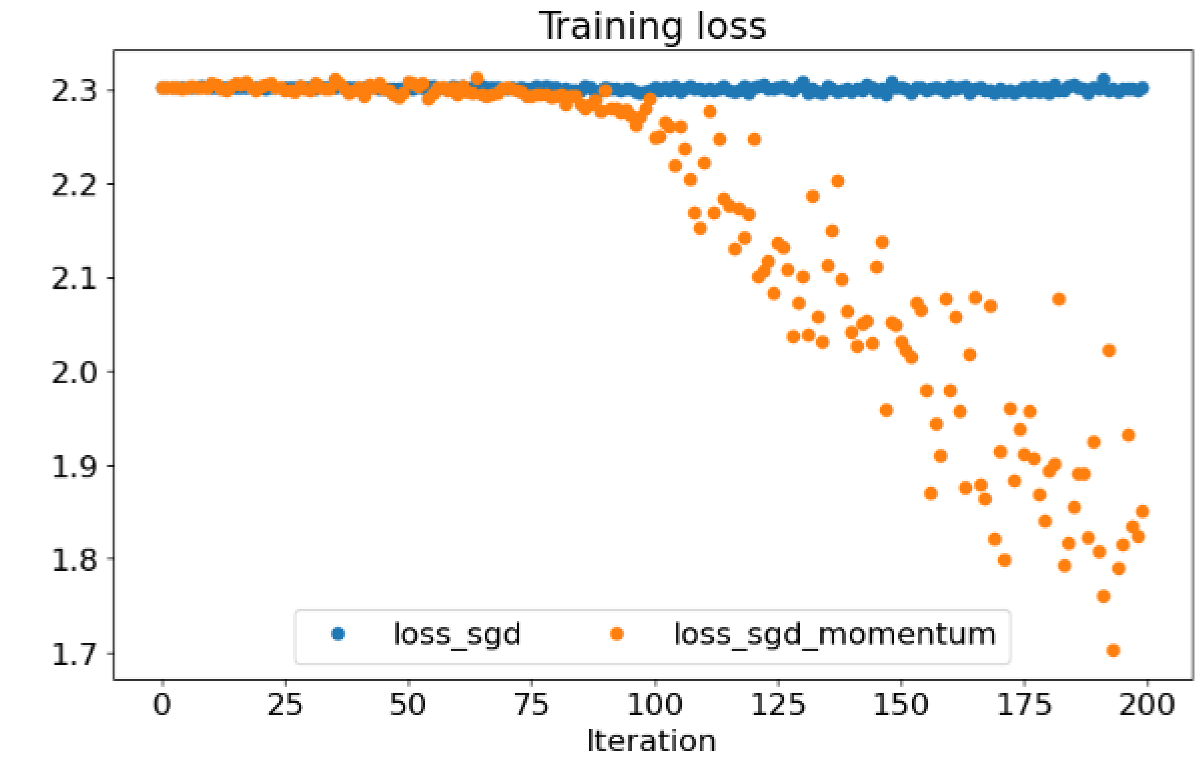
```
(Time 0.01 sec; Iteration 1 / 40) loss: 18.173244
(Epoch 0 / 20) train acc: 0.220000; val_acc: 0.103400
(Epoch 1 / 20) train acc: 0.240000; val_acc: 0.119400
(Epoch 2 / 20) train acc: 0.380000; val_acc: 0.128800
(Epoch 3 / 20) train acc: 0.700000; val_acc: 0.126600
(Epoch 4 / 20) train acc: 0.900000; val_acc: 0.125400
(Epoch 5 / 20) train acc: 0.920000; val_acc: 0.123800
(Time 0.51 sec; Iteration 11 / 40) loss: 0.451857
(Epoch 6 / 20) train acc: 0.960000; val_acc: 0.124300
(Epoch 7 / 20) train acc: 1.000000; val_acc: 0.126600
(Epoch 8 / 20) train acc: 1.000000; val_acc: 0.128300
(Epoch 9 / 20) train acc: 1.000000; val_acc: 0.129800
(Epoch 10 / 20) train acc: 1.000000; val_acc: 0.130400
(Time 0.83 sec; Iteration 21 / 40) loss: 0.062391
(Epoch 11 / 20) train acc: 1.000000; val_acc: 0.130600
(Epoch 12 / 20) train acc: 1.000000; val_acc: 0.130200
(Epoch 13 / 20) train acc: 1.000000; val_acc: 0.130700
(Epoch 14 / 20) train acc: 1.000000; val_acc: 0.132000
(Epoch 15 / 20) train acc: 1.000000; val_acc: 0.133200
(Time 1.15 sec; Iteration 31 / 40) loss: 0.027682
(Epoch 16 / 20) train acc: 1.000000; val_acc: 0.132700
(Epoch 17 / 20) train acc: 1.000000; val_acc: 0.133600
(Epoch 18 / 20) train acc: 1.000000; val_acc: 0.133600
(Epoch 19 / 20) train acc: 1.000000; val_acc: 0.134300
(Epoch 20 / 20) train acc: 1.000000; val_acc: 0.134300
```

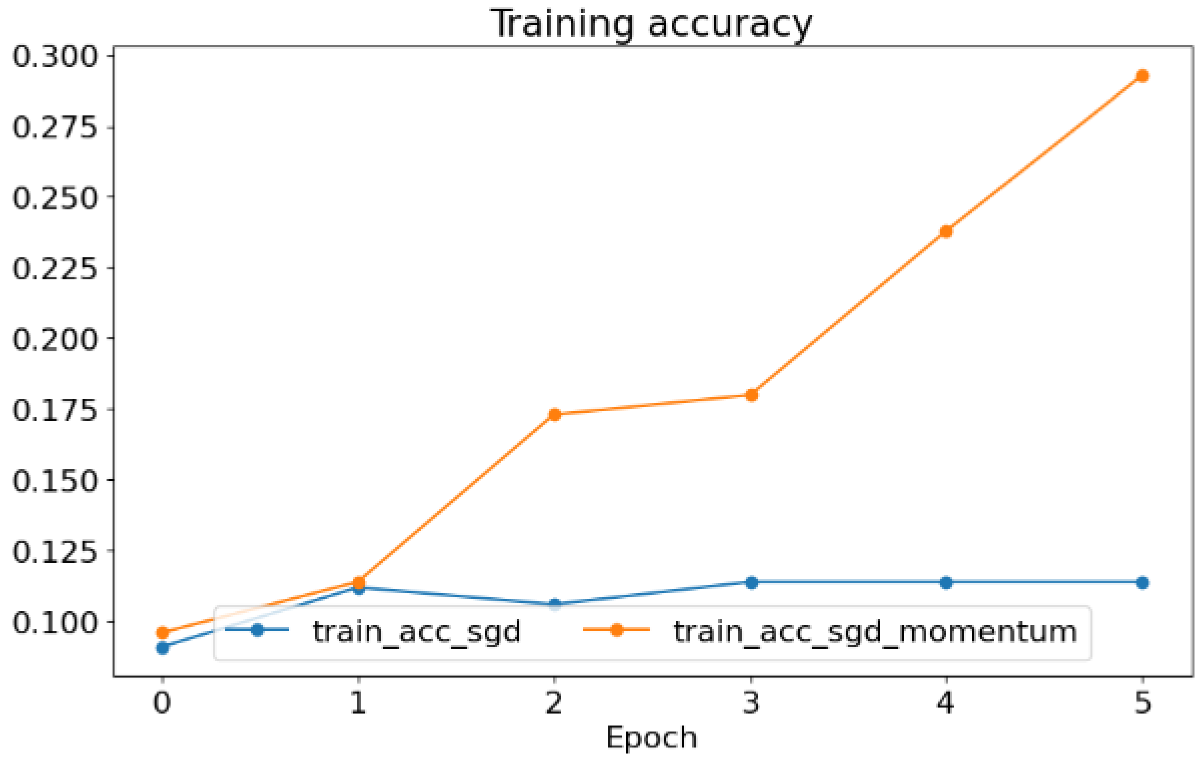


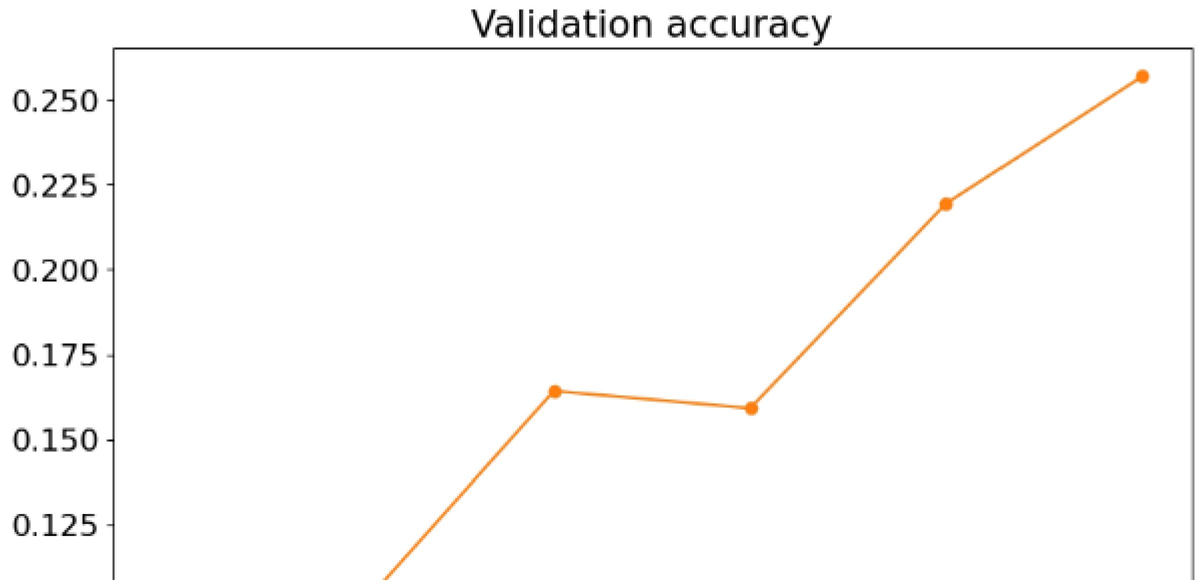
⑧SGD and Momentum：

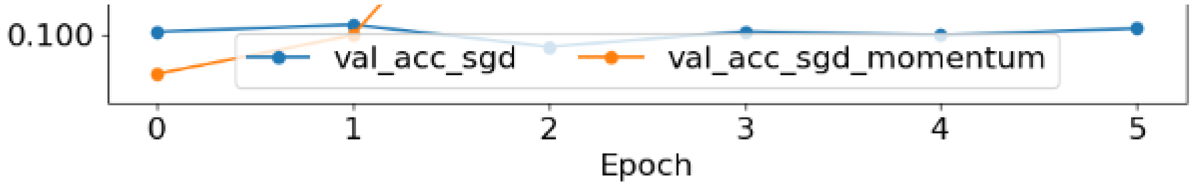
```
next_w error:  1.6802078709310813e-09
velocity error:  2.9254212825785614e-09
```

```
running with  sgd
(Time 0.00 sec; Iteration 1 / 200) loss: 2.302626
(Epoch 0 / 5) train acc: 0.091000; val_acc: 0.100700
(Epoch 1 / 5) train acc: 0.112000; val_acc: 0.102800
(Epoch 2 / 5) train acc: 0.106000; val_acc: 0.096300
(Epoch 3 / 5) train acc: 0.114000; val_acc: 0.100800
(Epoch 4 / 5) train acc: 0.114000; val_acc: 0.099700
(Epoch 5 / 5) train acc: 0.114000; val_acc: 0.101700

running with  sgd_momentum
(Time 0.00 sec; Iteration 1 / 200) loss: 2.302985
(Epoch 0 / 5) train acc: 0.096000; val_acc: 0.088300
(Epoch 1 / 5) train acc: 0.114000; val_acc: 0.099700
(Epoch 2 / 5) train acc: 0.173000; val_acc: 0.164200
(Epoch 3 / 5) train acc: 0.180000; val_acc: 0.159100
(Epoch 4 / 5) train acc: 0.238000; val_acc: 0.219500
(Epoch 5 / 5) train acc: 0.293000; val_acc: 0.257000
```


Training loss
loss_sgd
loss_sgd_momentum
Iteration


Training accuracy
train_acc_sgd
train_acc_sgd_momentum
Epoch


Validation accuracy



RMSProp：

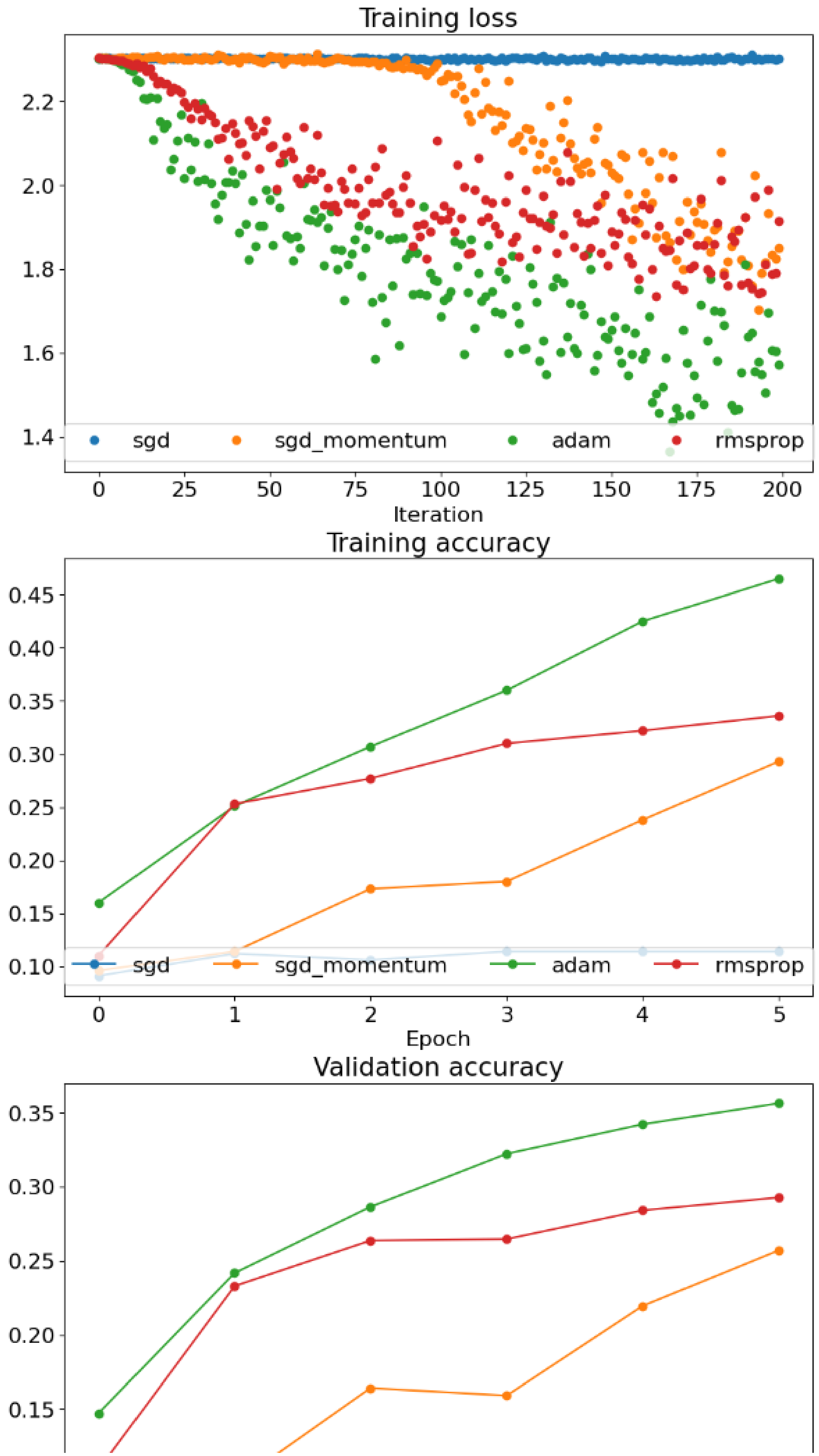
```
next_w error:  4.064797880829826e-09
cache error:  1.8620321382570356e-09
```

Adam：

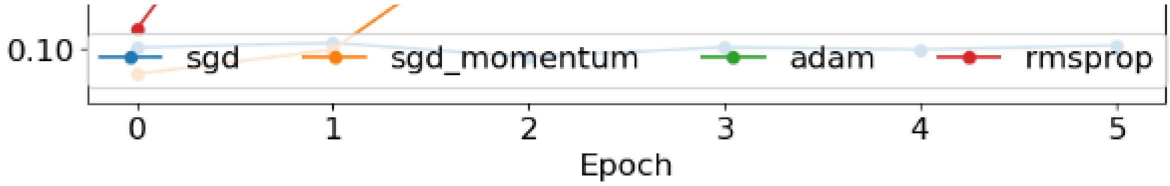
```
next_w error:  3.756728297598868e-09
v error:  3.4048987160545265e-09
m error:  2.786377729853651e-09
```

sgd_momentum:

```
running with  sgd_momentum
(Time 0.01 sec; Iteration 1 / 200) loss: 2.302626
(Epoch 0 / 5) train acc: 0.160000; val_acc: 0.147100
(Epoch 1 / 5) train acc: 0.251000; val_acc: 0.241700
(Epoch 2 / 5) train acc: 0.307000; val_acc: 0.286400
(Epoch 3 / 5) train acc: 0.360000; val_acc: 0.322200
(Epoch 4 / 5) train acc: 0.425000; val_acc: 0.342100
(Epoch 5 / 5) train acc: 0.465000; val_acc: 0.356300

running with  sgd_momentum
(Time 0.01 sec; Iteration 1 / 200) loss: 2.302985
(Epoch 0 / 5) train acc: 0.110000; val_acc: 0.109200
(Epoch 1 / 5) train acc: 0.253000; val_acc: 0.232900
(Epoch 2 / 5) train acc: 0.277000; val_acc: 0.263600
(Epoch 3 / 5) train acc: 0.310000; val_acc: 0.264600
(Epoch 4 / 5) train acc: 0.322000; val_acc: 0.284000
(Epoch 5 / 5) train acc: 0.336000; val_acc: 0.292700
```



Training loss
2.2
2.0
1.8
1.6
1.4
0
25
50
75
100
125
150
175
200
Iteration
sgd
sgd_momentum
adam
rmsprop
Training accuracy
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
1
2
3
4
5
Epoch
sgd
sgd_momentum
adam
rmsprop
Validation accuracy
0.35
0.30
0.25
0.20
0.15



⑨Dropout:

forward:

```
torch.float64
Running tests with p =  0.25
Mean of input:  9.999363323877896
Mean of train-time output:  9.990662915718602
Mean of test-time output:  9.999363323877896
Fraction of train-time output set to zero:  0.2505599856376648
Fraction of test-time output set to zero:  0.0

torch.float64
Running tests with p =  0.4
Mean of input:  9.999363323877896
Mean of train-time output:  9.974636010279736
Mean of test-time output:  9.999363323877896
Fraction of train-time output set to zero:  0.40133199095726013
Fraction of test-time output set to zero:  0.0

torch.float64
Running tests with p =  0.7
Mean of input:  9.999363323877896
Mean of train-time output:  10.012350710230168
Mean of test-time output:  9.999363323877896
Fraction of train-time output set to zero:  0.6997119784355164
Fraction of test-time output set to zero:  0.0
```

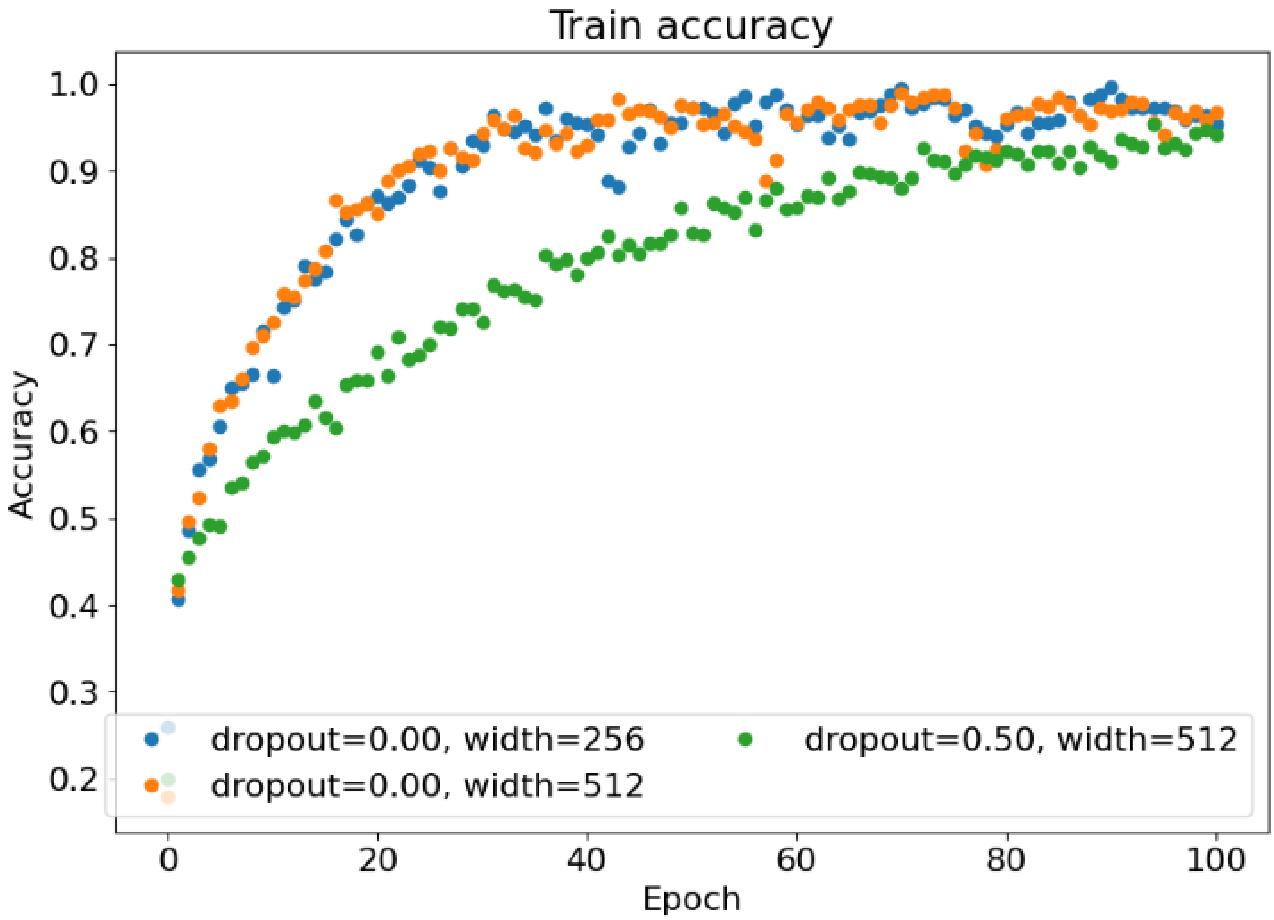
backward:

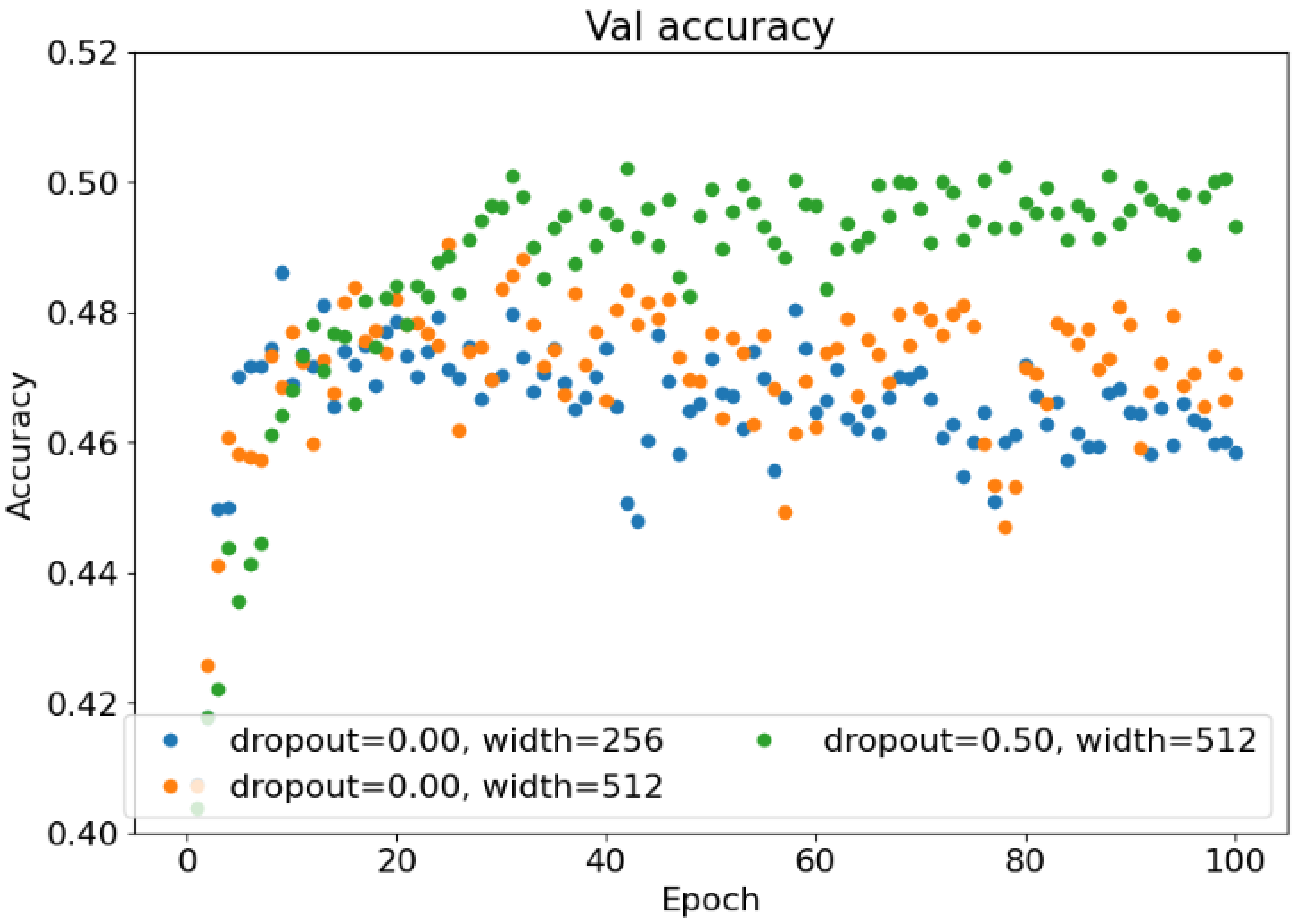
```
dx relative error:  3.914942325636866e-09
```

fc nn with dropout：

```
Running check with dropout =  0
Initial loss:  2.307533872340282
W1 relative error: 4.71e-08
W2 relative error: 9.16e-08
W3 relative error: 4.72e-08
b1 relative error: 9.39e-08
b2 relative error: 2.42e-08
b3 relative error: 2.39e-09

Running check with dropout =  0.25
Initial loss:  2.3024712491074784
W1 relative error: 6.99e-08
W2 relative error: 7.49e-08
W3 relative error: 3.67e-08
b1 relative error: 1.46e-07
b2 relative error: 1.96e-08
b3 relative error: 4.32e-09

Running check with dropout =  0.5
Initial loss:  2.3084634160711257
W1 relative error: 4.78e-08
W2 relative error: 3.19e-08
W3 relative error: 3.18e-08
b1 relative error: 2.95e-08
b2 relative error: 1.80e-08
b3 relative error: 3.25e-09
```

```
Output exceeds the size limit. Open the full output data in a text editor
Training a model with dropout=0.00 and width=256
(Time 0.02 sec; Iteration 1 / 3900) loss: 2.300956
(Epoch 0 / 100) train acc: 0.259000; val_acc: 0.245700
(Epoch 10 / 100) train acc: 0.664000; val_acc: 0.468900
(Epoch 20 / 100) train acc: 0.872000; val_acc: 0.478600
(Epoch 30 / 100) train acc: 0.929000; val_acc: 0.470400
(Epoch 40 / 100) train acc: 0.953000; val_acc: 0.474500
(Epoch 50 / 100) train acc: 0.973000; val_acc: 0.472900
(Epoch 60 / 100) train acc: 0.954000; val_acc: 0.464600
(Epoch 70 / 100) train acc: 0.995000; val_acc: 0.470900
(Epoch 80 / 100) train acc: 0.953000; val_acc: 0.472000
(Epoch 90 / 100) train acc: 0.997000; val_acc: 0.464700
(Epoch 100 / 100) train acc: 0.954000; val_acc: 0.458500

Training a model with dropout=0.00 and width=512
(Time 0.00 sec; Iteration 1 / 3900) loss: 2.305852
(Epoch 0 / 100) train acc: 0.179000; val_acc: 0.183200
(Epoch 10 / 100) train acc: 0.726000; val_acc: 0.477000
(Epoch 20 / 100) train acc: 0.851000; val_acc: 0.482000
(Epoch 30 / 100) train acc: 0.944000; val_acc: 0.483500
(Epoch 40 / 100) train acc: 0.929000; val_acc: 0.466500
(Epoch 50 / 100) train acc: 0.973000; val_acc: 0.476800
(Epoch 60 / 100) train acc: 0.955000; val_acc: 0.462400
(Epoch 70 / 100) train acc: 0.989000; val_acc: 0.480700
(Epoch 80 / 100) train acc: 0.961000; val_acc: 0.471500
...
(Epoch 80 / 100) train acc: 0.922000; val_acc: 0.496900
(Epoch 90 / 100) train acc: 0.910000; val_acc: 0.495700
(Epoch 100 / 100) train acc: 0.941000; val_acc: 0.493200
```





在使用神经网络预测类别的时候，有更多的办法保证结果的精确度，可以加入dropout防止overfitting，让模型在val和test的时候效果更好。

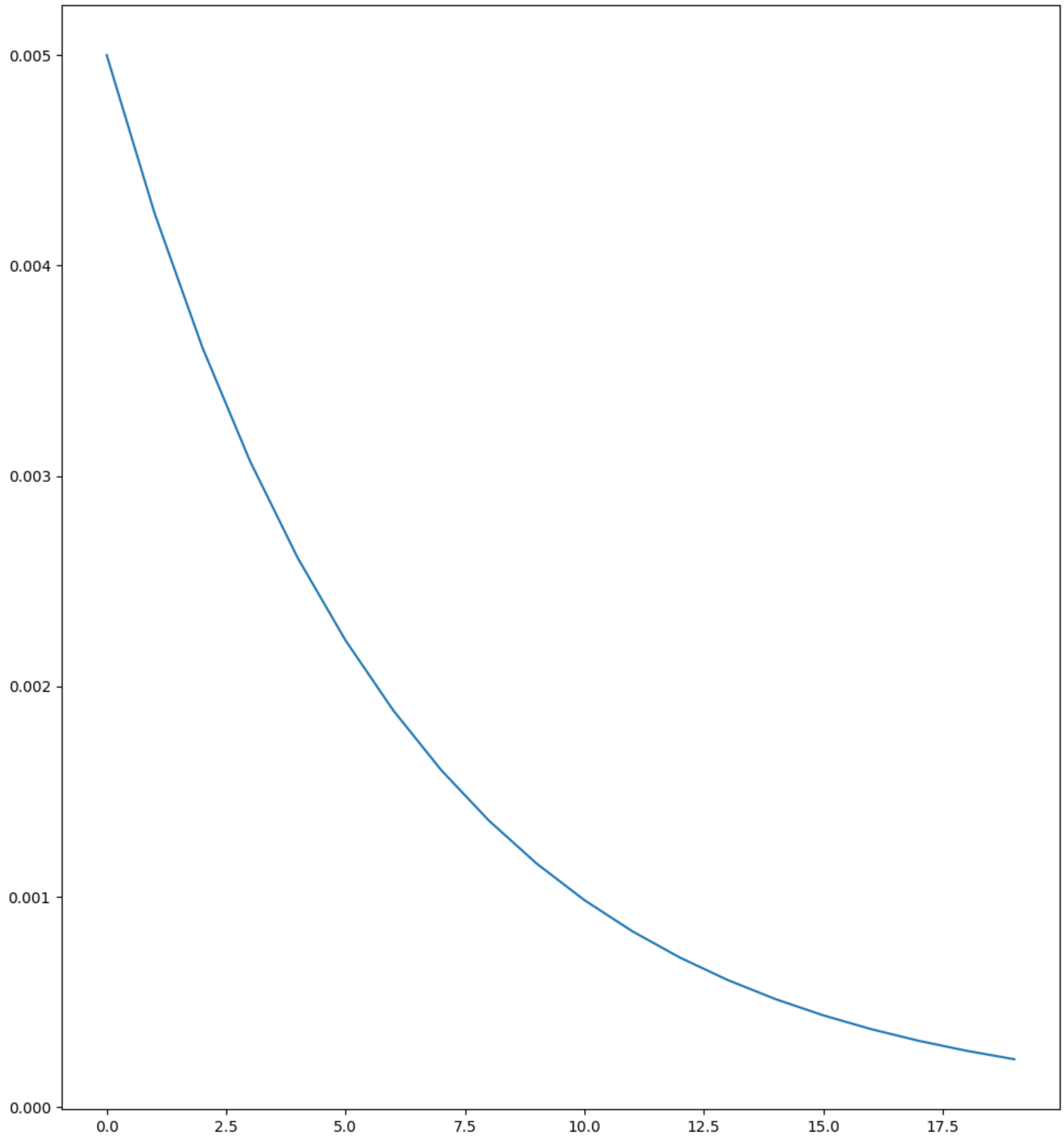
模型性能随学习率变化的情况：

```
learning_rate of 1 epoch: 0.005000
[1,  2000] loss: 1.8903
[1,  4000] loss: 1.6018
[1,  6000] loss: 1.5008
[1,  8000] loss: 1.3934
[1, 10000] loss: 1.3437
[1, 12000] loss: 1.2707
learning_rate of 2 epoch: 0.004250
[2,  2000] loss: 1.0788
[2,  4000] loss: 1.0702
[2,  6000] loss: 1.0388
[2,  8000] loss: 1.0245
[2, 10000] loss: 1.0251
[2, 12000] loss: 1.0159
learning_rate of 3 epoch: 0.003613
[3,  2000] loss: 0.7489
[3,  4000] loss: 0.7641
[3,  6000] loss: 0.7570
[3,  8000] loss: 0.8132
[3, 10000] loss: 0.8070
[3, 12000] loss: 0.7652
learning_rate of 4 epoch: 0.003071
[4,  2000] loss: 0.4992
[4,  4000] loss: 0.5184
[4,  6000] loss: 0.5332
[4,  8000] loss: 0.5629
[4, 10000] loss: 0.5730
[4, 12000] loss: 0.5633
learning_rate of 5 epoch: 0.002610
[5,  2000] loss: 0.2772
[5,  4000] loss: 0.3050
[5,  6000] loss: 0.3337
[5,  8000] loss: 0.3374
[5, 10000] loss: 0.3669
[5, 12000] loss: 0.3638
learning_rate of 6 epoch: 0.002219
[6,  2000] loss: 0.1586
[6,  4000] loss: 0.1559
[6,  6000] loss: 0.1939
[6,  8000] loss: 0.2020
[6, 10000] loss: 0.2108
[6, 12000] loss: 0.2092
learning_rate of 7 epoch: 0.001886
[7,  2000] loss: 0.0740
[7,  4000] loss: 0.0751
[7,  6000] loss: 0.0929
[7,  8000] loss: 0.0950
[7, 10000] loss: 0.1024
[7, 12000] loss: 0.0952
learning_rate of 8 epoch: 0.001603
[8,  2000] loss: 0.0322
[8,  4000] loss: 0.0309
```

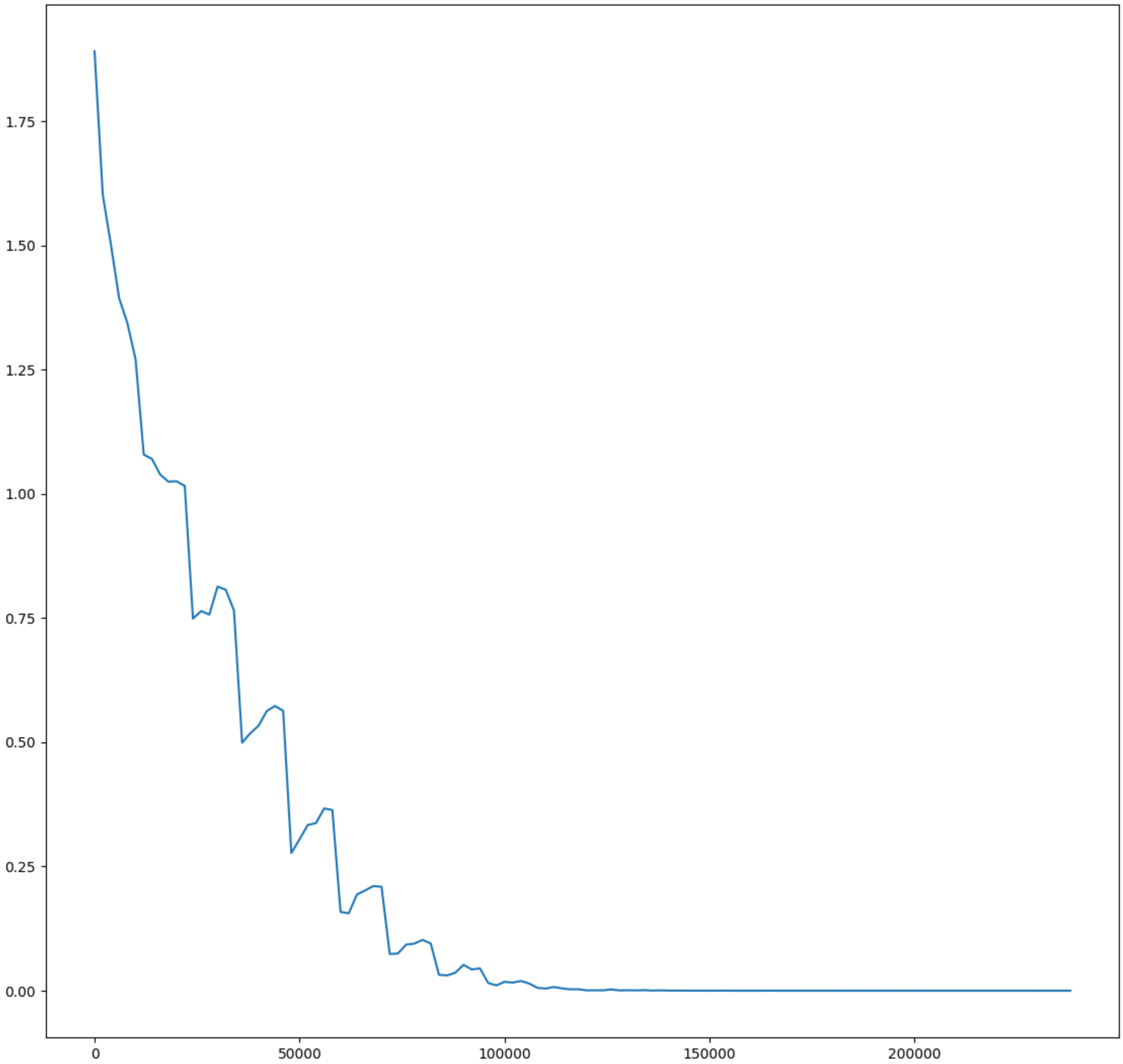
[8, 6000] loss: 0.0364
[8, 8000] loss: 0.0522
[8, 10000] loss: 0.0429
[8, 12000] loss: 0.0451
learning_rate of 9 epoch: 0.001362
[9, 2000] loss: 0.0157
[9, 4000] loss: 0.0108
[9, 6000] loss: 0.0180
[9, 8000] loss: 0.0165
[9, 10000] loss: 0.0197
[9, 12000] loss: 0.0146
learning_rate of 10 epoch: 0.001158
[10, 2000] loss: 0.0060
[10, 4000] loss: 0.0044
[10, 6000] loss: 0.0075
[10, 8000] loss: 0.0048
[10, 10000] loss: 0.0030
[10, 12000] loss: 0.0031
learning_rate of 11 epoch: 0.000984
[11, 2000] loss: 0.0009
[11, 4000] loss: 0.0009
[11, 6000] loss: 0.0009
[11, 8000] loss: 0.0027
[11, 10000] loss: 0.0008
[11, 12000] loss: 0.0010
learning_rate of 12 epoch: 0.000837
[12, 2000] loss: 0.0008
[12, 4000] loss: 0.0012
[12, 6000] loss: 0.0004
[12, 8000] loss: 0.0009
[12, 10000] loss: 0.0004
[12, 12000] loss: 0.0003
learning_rate of 13 epoch: 0.000711
[13, 2000] loss: 0.0003
[13, 4000] loss: 0.0003
[13, 6000] loss: 0.0002
[13, 8000] loss: 0.0002
[13, 10000] loss: 0.0003
[13, 12000] loss: 0.0003
learning_rate of 14 epoch: 0.000605
[14, 2000] loss: 0.0002
[14, 4000] loss: 0.0002
[14, 6000] loss: 0.0002
[14, 8000] loss: 0.0002
[14, 10000] loss: 0.0002
[14, 12000] loss: 0.0002
learning_rate of 15 epoch: 0.000514
[15, 2000] loss: 0.0002
[15, 4000] loss: 0.0002
[15, 6000] loss: 0.0002
[15, 8000] loss: 0.0002
[15, 10000] loss: 0.0002

[15, 12000] loss: 0.0002
learning_rate of 16 epoch: 0.000437
[16, 2000] loss: 0.0002
[16, 4000] loss: 0.0002
[16, 6000] loss: 0.0002
[16, 8000] loss: 0.0002
[16, 10000] loss: 0.0002
[16, 12000] loss: 0.0002
learning_rate of 17 epoch: 0.000371
[17, 2000] loss: 0.0001
[17, 4000] loss: 0.0002
[17, 6000] loss: 0.0002
[17, 8000] loss: 0.0001
[17, 10000] loss: 0.0002
[17, 12000] loss: 0.0002
learning_rate of 18 epoch: 0.000316
[18, 2000] loss: 0.0002
[18, 4000] loss: 0.0002
[18, 6000] loss: 0.0001
[18, 8000] loss: 0.0002
[18, 10000] loss: 0.0002
[18, 12000] loss: 0.0001
learning_rate of 19 epoch: 0.000268
[19, 2000] loss: 0.0001
[19, 4000] loss: 0.0001
[19, 6000] loss: 0.0001
[19, 8000] loss: 0.0002
[19, 10000] loss: 0.0001
[19, 12000] loss: 0.0001
learning_rate of 20 epoch: 0.000228
[20, 2000] loss: 0.0002
[20, 4000] loss: 0.0001
[20, 6000] loss: 0.0001
[20, 8000] loss: 0.0001
[20, 10000] loss: 0.0002
[20, 12000] loss: 0.0001

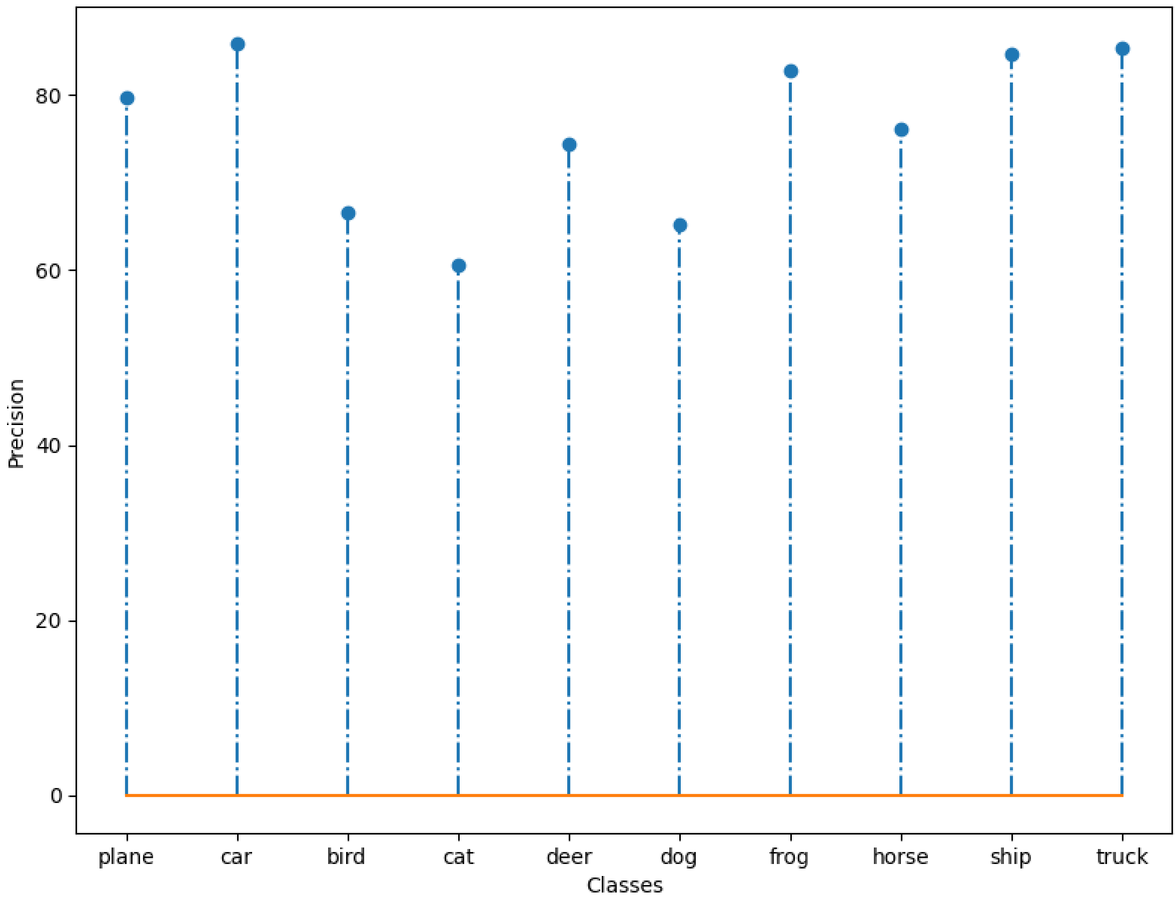
learning_rate变化：

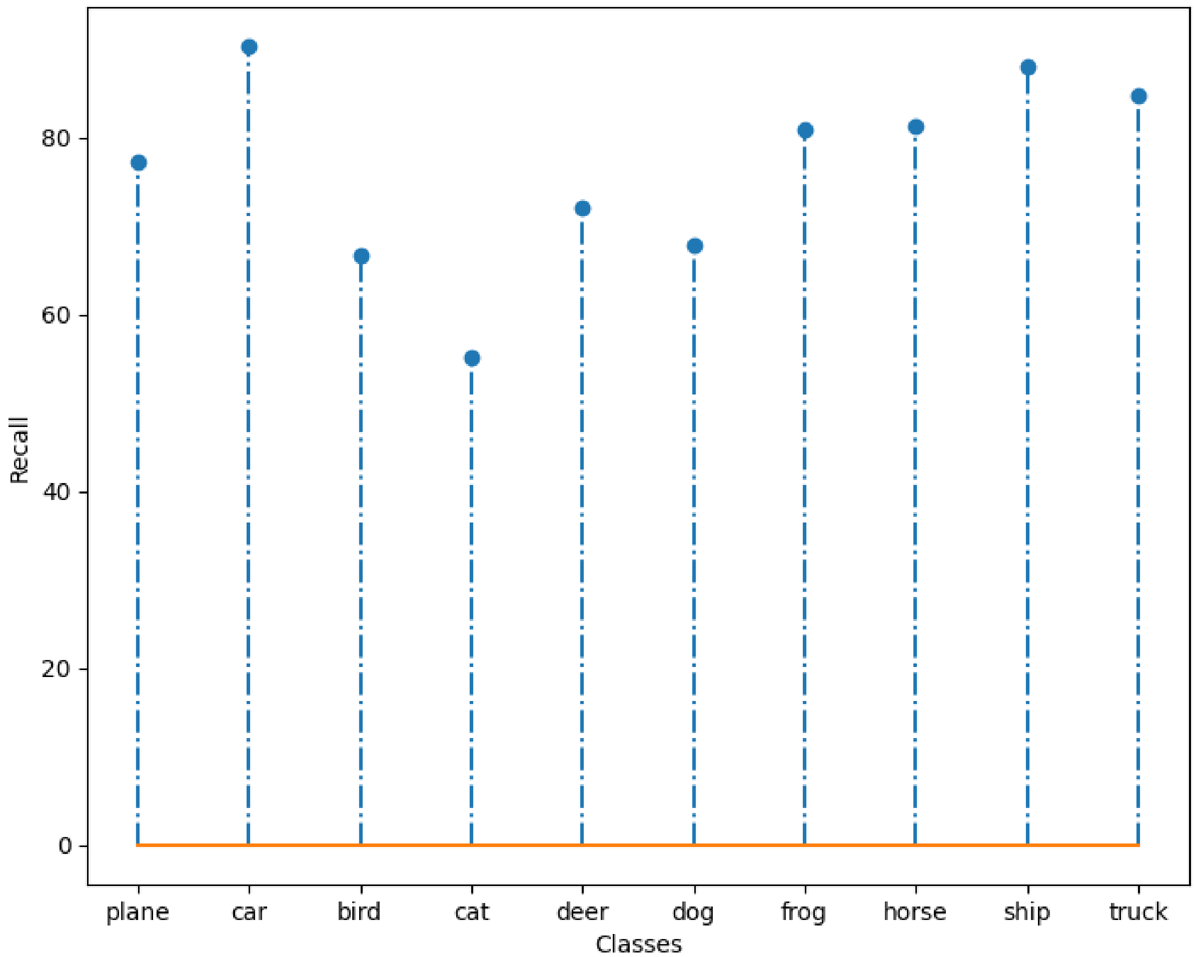


Loss：



精确率和召回率:





可以看到随着learning_rate的减小，losss值也在减小。

神经网络的分类效果相比于线性分类器的提升还是很明显的。