计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目： Regularization Batch Normalization		**学号：** 201900130015
日期： 2021	**班级：** 智能班	**姓名：** 李德锋
Email： ldf2878945468@163.com		

实验目的：

学会实现 Regularization Batch Normalization

实验软件和硬件环境：

Intel(R) Core(TM) i7-8550U CPU
华为云

实验原理和方法：

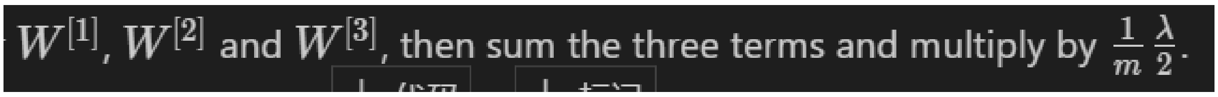
根据公式补全代码，并测试运行

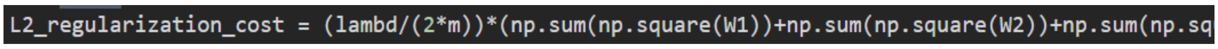
实验步骤：（不要求罗列完整源代码）

Regularization：

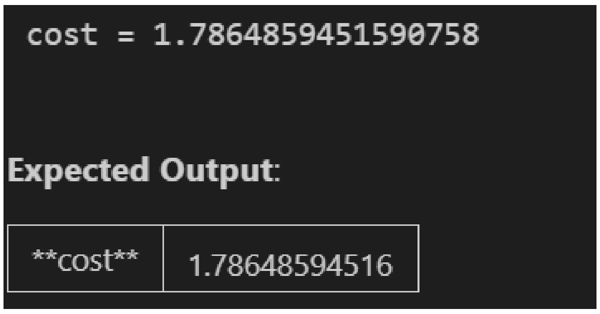
在深度学习模型中使用正则化
1. 导入所需的包
2. 根据公式补全代码

L2 Regularization：


$W^{[1]}$, $W^{[2]}$ and $W^{[3]}$, then sum the three terms and multiply by $\frac{1}{m}\frac{\lambda}{2}$.



```
L2_regularization_cost = (lambd/(2*m))*(np.sum(np.square(W1))+np.sum(np.square(W2))+np.sum(np.sq
```



```
cost = 1.7864859451590758
```

Expected Output:

cost	1.78648594516

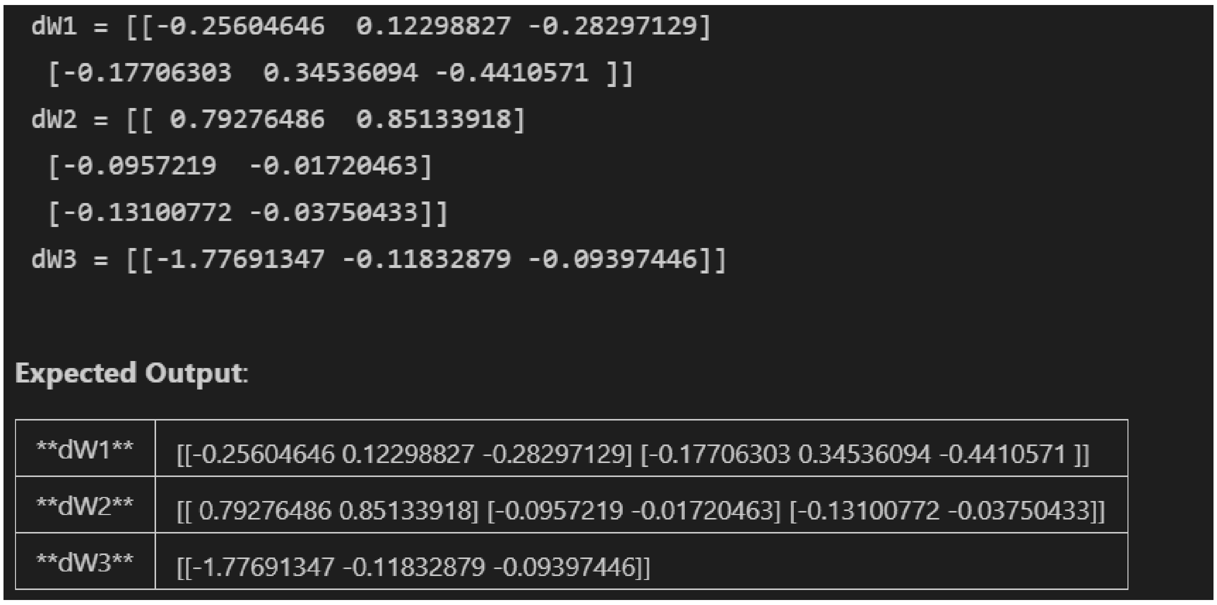
实施反向传播中的正则化

```
### START CODE HERE ### (approx. 1 line)
dW3 = 1./m * np.dot(dZ3, A2.T) + (lambd/m)*W3
### END CODE HERE ###
db3 = 1./m * np.sum(dZ3, axis=1, keepdims = True)

dA2 = np.dot(W3.T, dZ3)
dZ2 = np.multiply(dA2, np.int64(A2 > 0))
### START CODE HERE ### (approx. 1 line)
dW2 = 1./m * np.dot(dZ2, A1.T) + (lambd/m)*W2
### END CODE HERE ###
db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)

dA1 = np.dot(W2.T, dZ2)
dZ1 = np.multiply(dA1, np.int64(A1 > 0))
### START CODE HERE ### (approx. 1 line)
dW1 = 1./m * np.dot(dZ1, X.T) + (lambd/m)*W1
### END CODE HERE ###
```

结果：

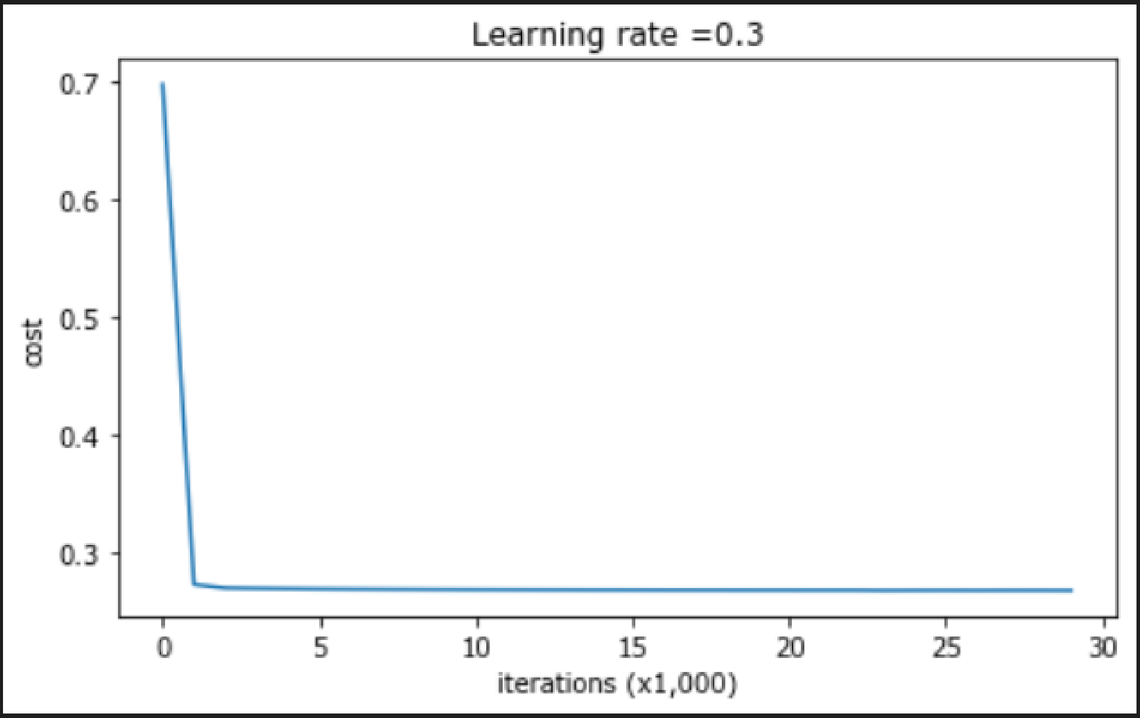


```
dW1 = [[-0.25604646  0.12298827 -0.28297129]
 [-0.17706303  0.34536094 -0.4410571 ]]
dW2 = [[ 0.79276486  0.85133918]
 [-0.0957219  -0.01720463]
 [-0.13100772 -0.03750433]]
dW3 = [[-1.77691347 -0.11832879 -0.09397446]]
```

Expected Output:

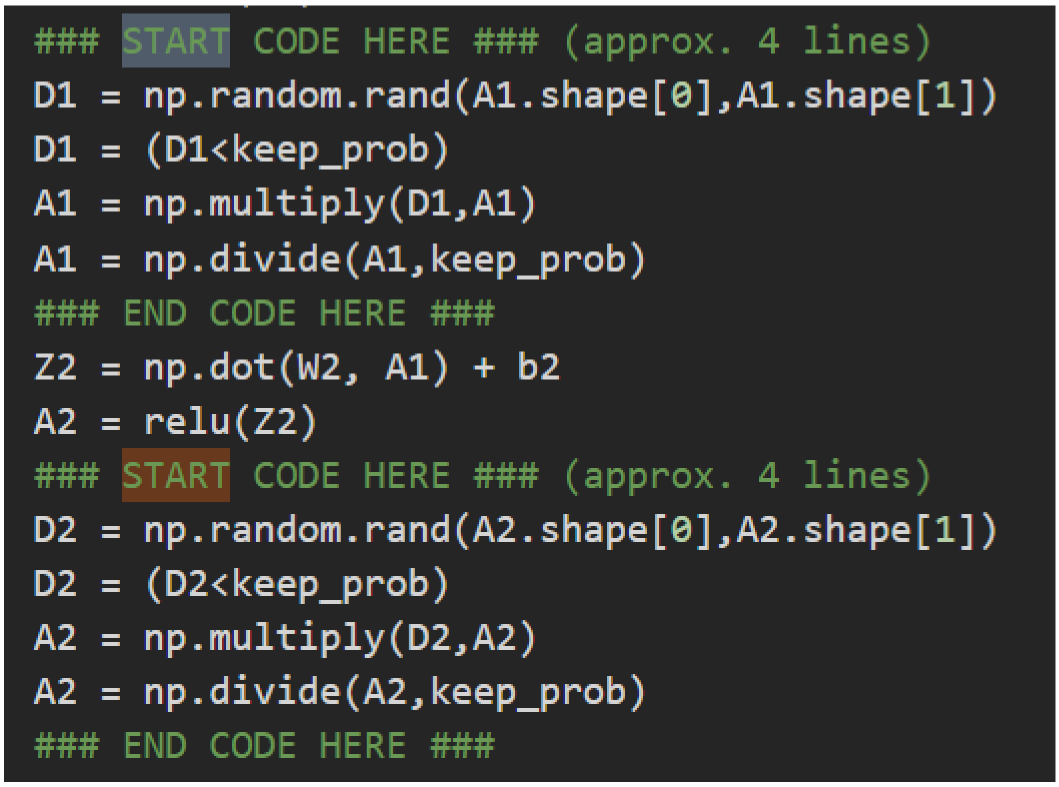
dW1	[[-0.25604646 0.12298827 -0.28297129] [-0.17706303 0.34536094 -0.4410571]]
dW2	[[0.79276486 0.85133918] [-0.0957219 -0.01720463] [-0.13100772 -0.03750433]]
dW3	[[-1.77691347 -0.11832879 -0.09397446]]

```
Cost after iteration 0: 0.6974484493131264
Cost after iteration 10000: 0.2684918873282239
Cost after iteration 20000: 0.2680916337127301
```

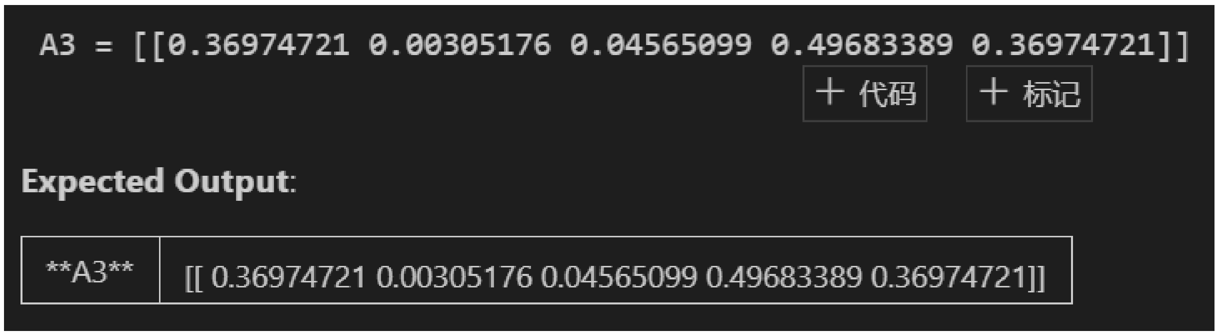


```
On the train set:
Accuracy: 0.9383886255924171
On the test set:
Accuracy: 0.93
```

实现具有丢失的正向传播



```
### START CODE HERE ### (approx. 4 lines)
D1 = np.random.rand(A1.shape[0],A1.shape[1])
D1 = (D1<keep_prob)
A1 = np.multiply(D1,A1)
A1 = np.divide(A1,keep_prob)
### END CODE HERE ###
Z2 = np.dot(W2, A1) + b2
A2 = relu(Z2)
### START CODE HERE ### (approx. 4 lines)
D2 = np.random.rand(A2.shape[0],A2.shape[1])
D2 = (D2<keep_prob)
A2 = np.multiply(D2,A2)
A2 = np.divide(A2,keep_prob)
### END CODE HERE ###
```

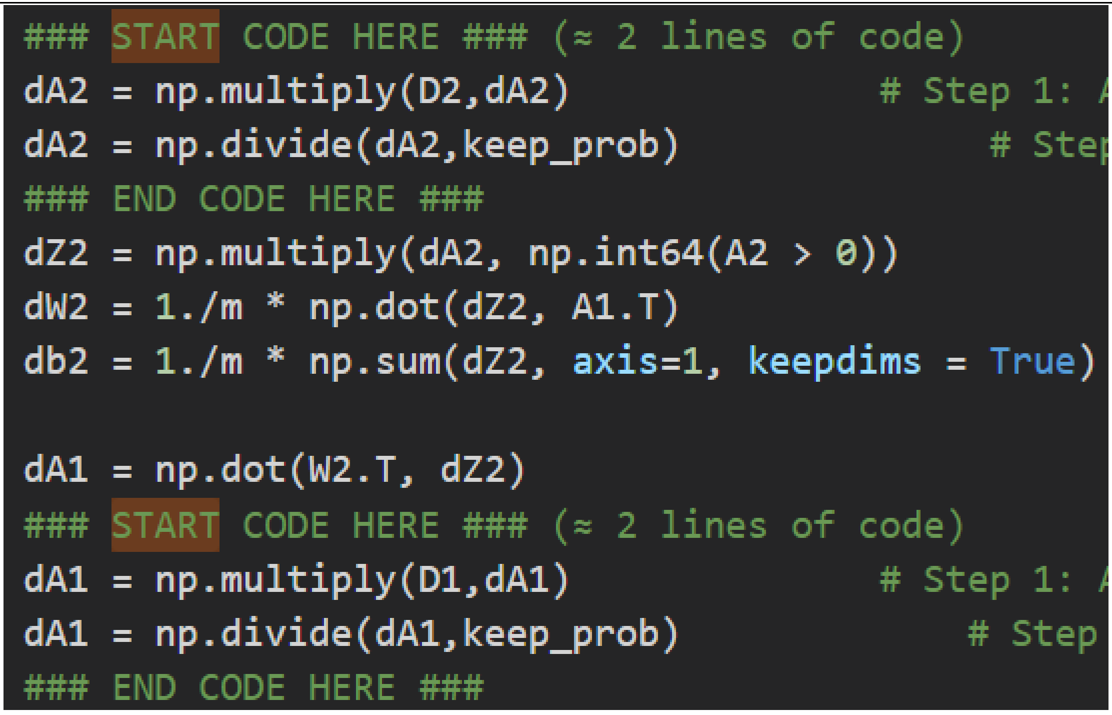


```
A3 = [[0.36974721 0.00305176 0.04565099 0.49683389 0.36974721]]
```

Expected Output:

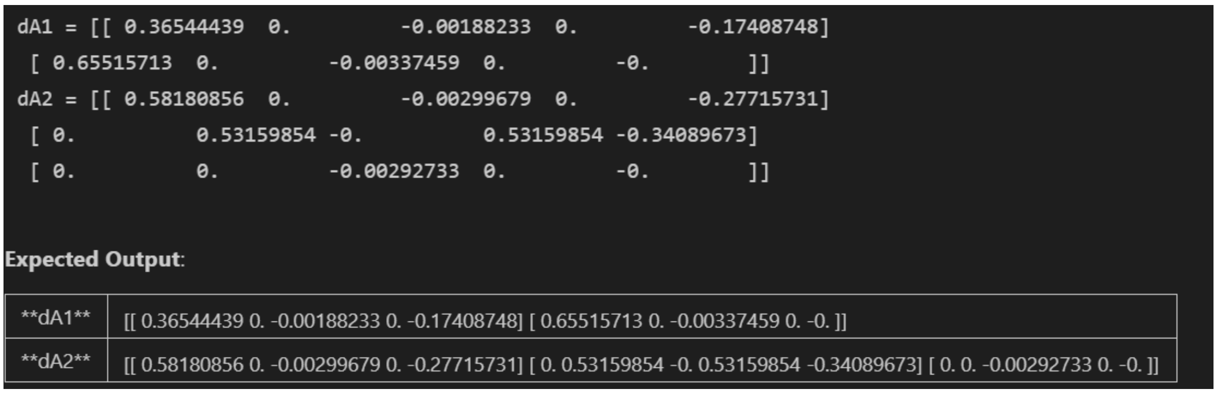
A3	[[0.36974721 0.00305176 0.04565099 0.49683389 0.36974721]]

Backward propagation with dropout



```python
### START CODE HERE ### (≈ 2 lines of code)
dA2 = np.multiply(D2,dA2)                # Step 1:
dA2 = np.divide(dA2,keep_prob)               # Step
### END CODE HERE ###
dZ2 = np.multiply(dA2, np.int64(A2 > 0))
dW2 = 1./m * np.dot(dZ2, A1.T)
db2 = 1./m * np.sum(dZ2, axis=1, keepdims = True)

dA1 = np.dot(W2.T, dZ2)
### START CODE HERE ### (≈ 2 lines of code)
dA1 = np.multiply(D1,dA1)                # Step 1:
dA1 = np.divide(dA1,keep_prob)              # Step
### END CODE HERE ###
```

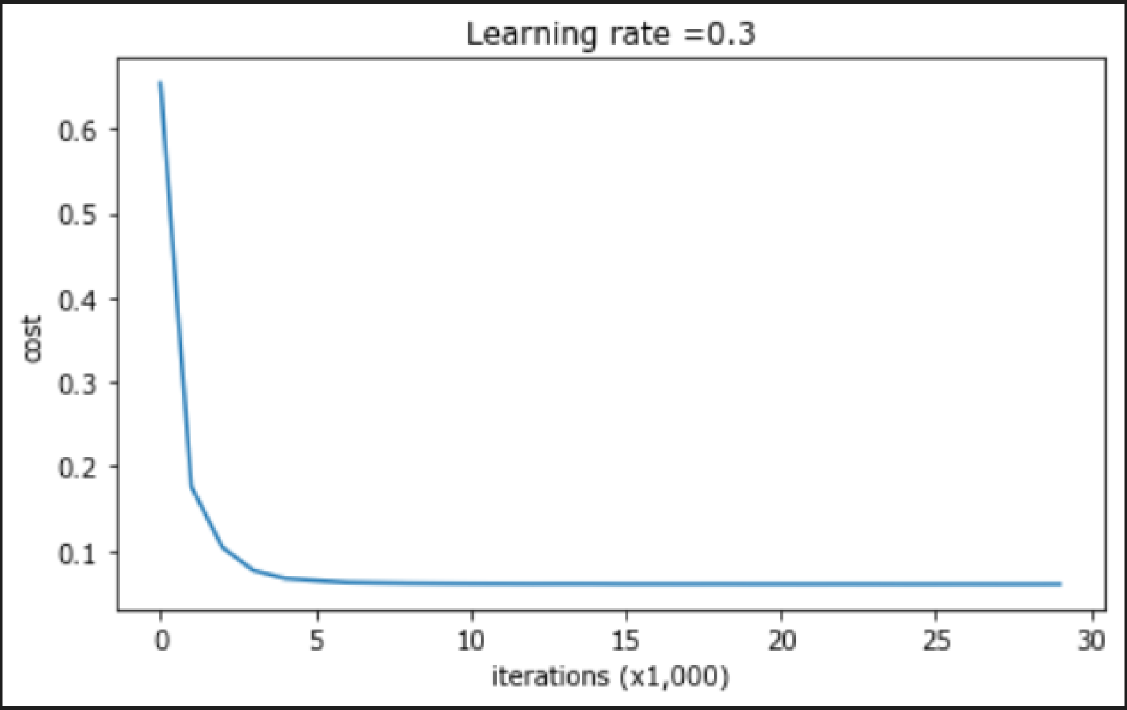


```
dA1 = [[ 0.36544439  0.          -0.00188233  0.          -0.17408748]
 [ 0.65515713  0.          -0.00337459  0.          -0.          ]]
dA2 = [[ 0.58180856  0.          -0.00299679  0.          -0.27715731]
 [ 0.          0.53159854 -0.          0.53159854 -0.34089673]
 [ 0.          0.          -0.00292733  0.          -0.          ]]
```

Expected Output:

dA1	[[0.36544439 0. -0.00188233 0. -0.17408748] [0.65515713 0. -0.00337459 0. -0.]]
dA2	[[0.58180856 0. -0.00299679 0. -0.27715731] [0. 0.53159854 -0. 0.53159854 -0.34089673] [0. 0. -0.00292733 0. -0.]]

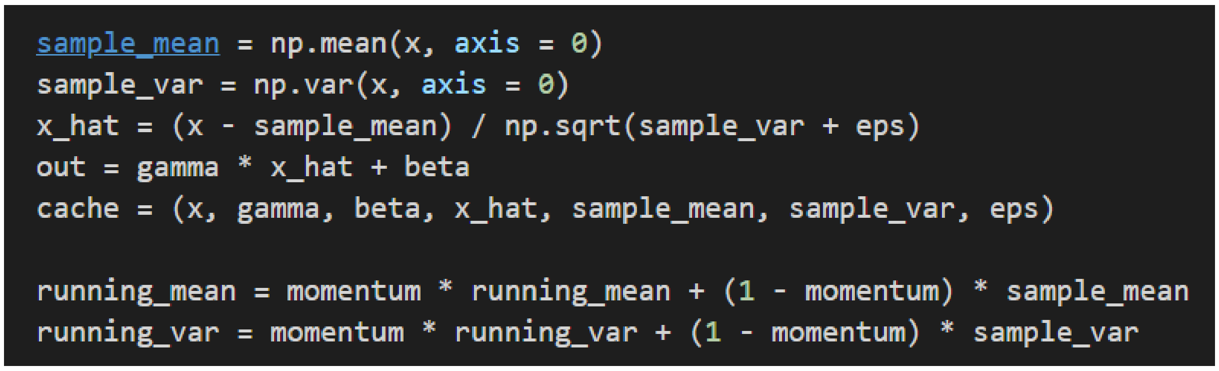
```
Cost after iteration 10000: 0.06101698657490560 5
Cost after iteration 20000: 0.060582435798513114
```



```
On the train set:
Accuracy: 0.9289099526066351
On the test set:
Accuracy: 0.95
```

Batch Normalization
在网络中插入批处理规范化层。在训练时，批处理规范化层使用数据的小批量来估计每个特征的平均值和标准偏差。这些估计的平均值和标准偏差随后被用于集中和标准化小批量的特征
批处理规范化的正向传递
mode == 'train'

$$\widehat{x}^{(k)}=\frac{x^{(k)}-\mathrm{E}\left[x^{(k)}\right]}{\sqrt{\operatorname{Var}\left[x^{(k)}\right]}}$$

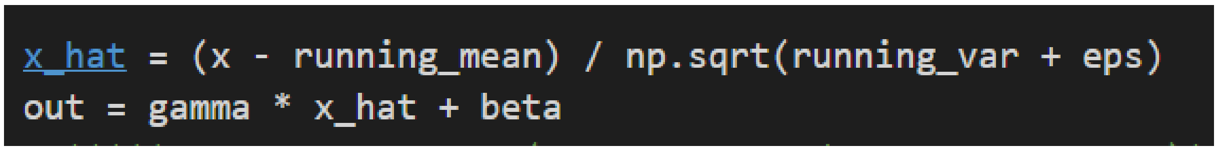


```
sample_mean = np.mean(x, axis = 0)
sample_var = np.var(x, axis = 0)
x_hat = (x - sample_mean) / np.sqrt(sample_var + eps)
out = gamma * x_hat + beta
cache = (x, gamma, beta, x_hat, sample_mean, sample_var, eps)

running_mean = momentum * running_mean + (1 - momentum) * sample_mean
running_var = momentum * running_var + (1 - momentum) * sample_var
```

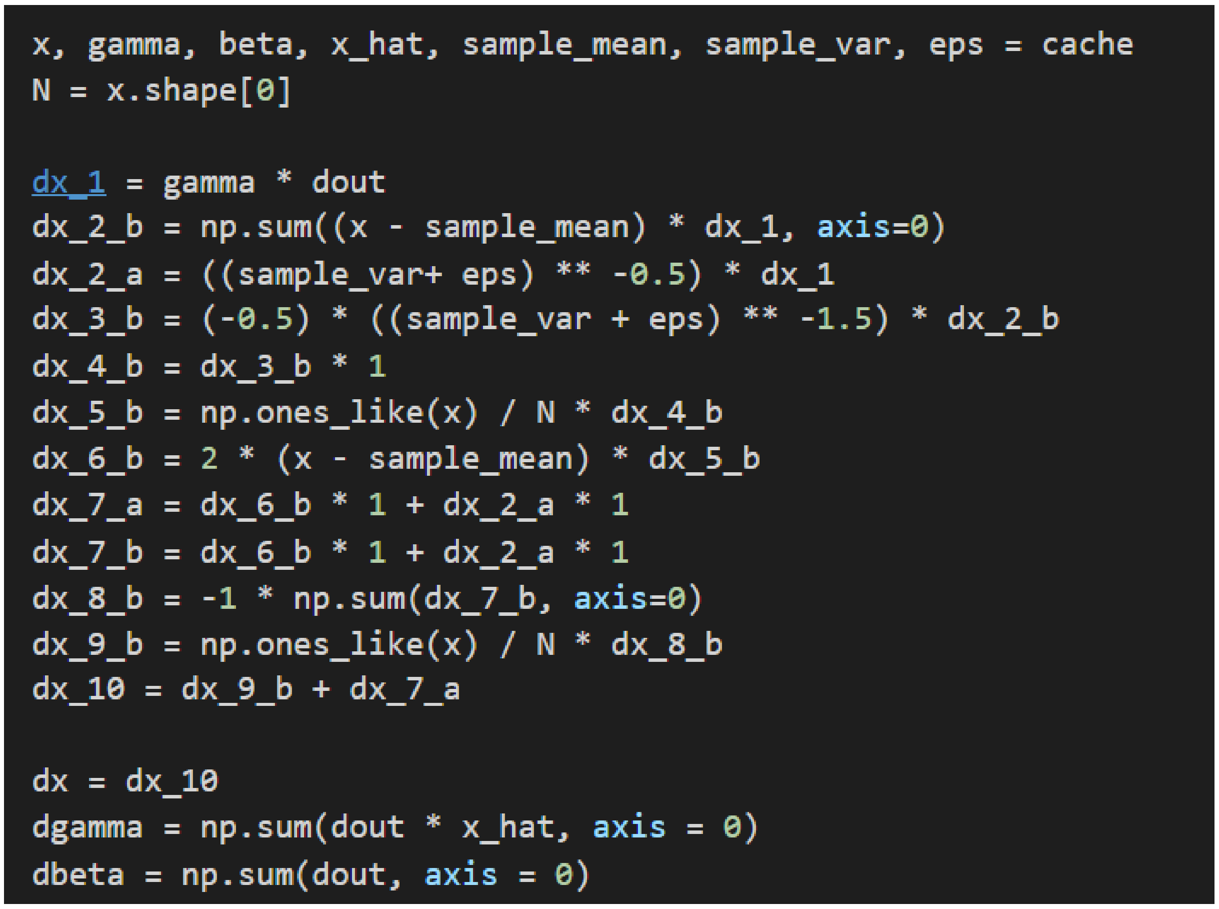
mode == ’test’

在测试数据集上使用运行时均值和方差进行计算



```
x_hat = (x - running_mean) / np.sqrt(running_var + eps)
out = gamma * x_hat + beta
```

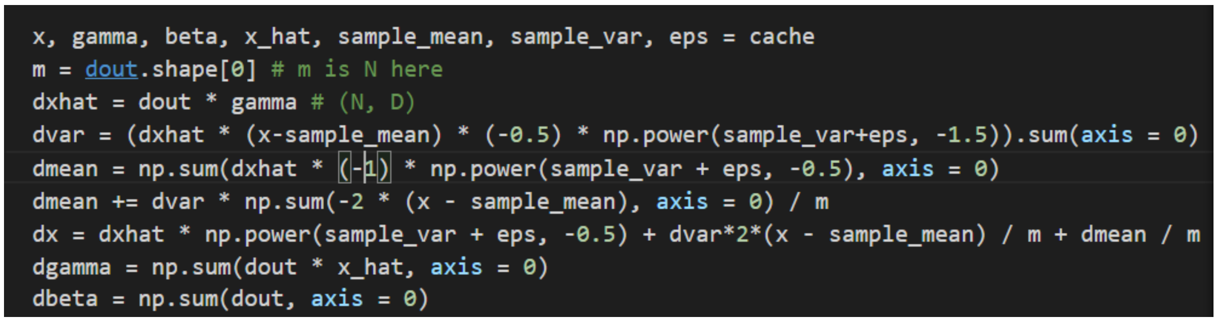
批处理规范化的反向传递



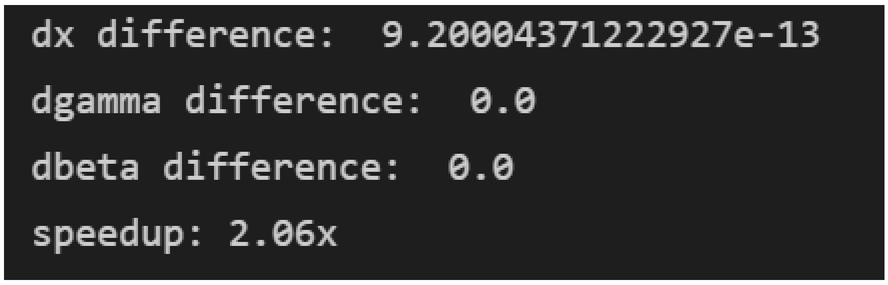
```
x, gamma, beta, x_hat, sample_mean, sample_var, eps = cache
N = x.shape[0]

dx_1 = gamma * dout
dx_2_b = np.sum((x - sample_mean) * dx_1, axis=0)
dx_2_a = ((sample_var+ eps) ** -0.5) * dx_1
dx_3_b = (-0.5) * ((sample_var + eps) ** -1.5) * dx_2_b
dx_4_b = dx_3_b * 1
dx_5_b = np.ones_like(x) / N * dx_4_b
dx_6_b = 2 * (x - sample_mean) * dx_5_b
dx_7_a = dx_6_b * 1 + dx_2_a * 1
dx_7_b = dx_6_b * 1 + dx_2_a * 1
dx_8_b = -1 * np.sum(dx_7_b, axis=0)
dx_9_b = np.ones_like(x) / N * dx_8_b
dx_10 = dx_9_b + dx_7_a

dx = dx_10
dgamma = np.sum(dout * x_hat, axis = 0)
dbeta = np.sum(dout, axis = 0)
```



```
x, gamma, beta, x_hat, sample_mean, sample_var, eps = cache
m = dout.shape[0] # m is N here
dxhat = dout * gamma # (N, D)
dvar = (dxhat * (x-sample_mean) * (-0.5) * np.power(sample_var+eps, -1.5)).sum(axis = 0)
dmean = np.sum(dxhat * (-1) * np.power(sample_var + eps, -0.5), axis = 0)
dmean += dvar * np.sum(-2 * (x - sample_mean), axis = 0) / m
dx = dxhat * np.power(sample_var + eps, -0.5) + dvar*2*(x - sample_mean) / m + dmean / m
dgamma = np.sum(dout * x_hat, axis = 0)
dbeta = np.sum(dout, axis = 0)
```



```
dx difference:  9.20004371222927e-13

dgamma difference:  0.0

dbeta difference:  0.0

speedup: 2.06x
```

结论分析与体会：

Regularization

学会了实现前向、后向传播的正则化

正则化减少过度拟合，正则化使权重降低。

L2 正则化和 Dropout 是两种非常有效的正则化技术

只在训练中使用 dropout，测试期间不要随机消除节点

在前向和后向传播过程中应用 dropout

在训练期间，用 keep_prob 划分每个脱离层，以保持激活的期望值不变

Batch Normalization

权重不同的初始化值对网络有着不同的影响

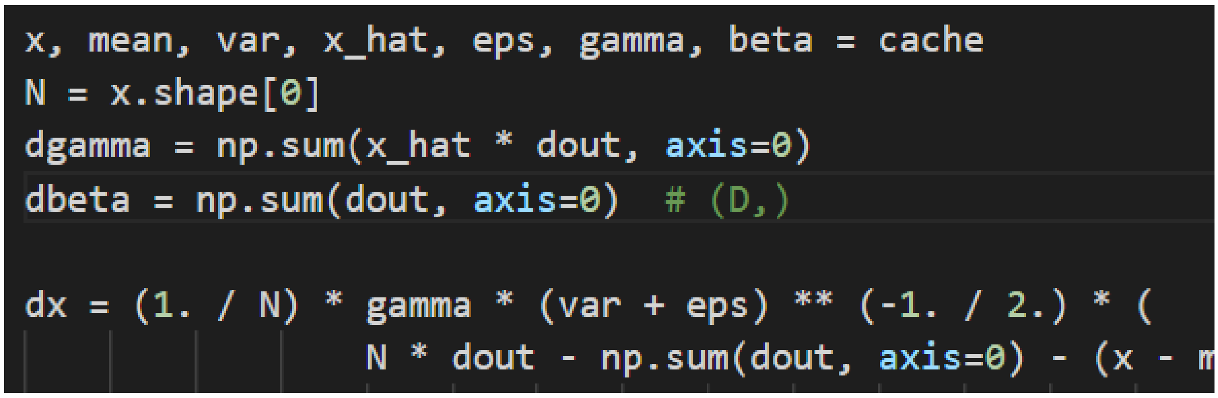
采用 batch normalization 的网络可以降低坏的初始值影响。

在一定的范围内，随着 batch size 的提高，使用 batch normalization 的网络的准确率也会提高

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1—3 道问答题：

如何实现一步的梯度计算

上网查询后实现



```
x, mean, var, x_hat, eps, gamma, beta = cache
N = x.shape[0]
dgamma = np.sum(x_hat * dout, axis=0)
dbeta = np.sum(dout, axis=0)  # (D,)

dx = (1. / N) * gamma * (var + eps) ** (-1. / 2.) * (
            N * dout - np.sum(dout, axis=0) - (x - m
```