关于使用 CS231 中 TwoLayerNet 的代码

18052223 饶晓龙

一、关于代码的理解

目标：开发具有完全连接的层的神经网络以执行分类，并在 CIFAR-10 数据集上进行测试。

```python
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt

from cs231n.classifiers.neural_net import TwoLayerNet

plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots设置图的默认大小
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

def rel_error(x, y):
    """ returns relative error 返回相对误差"""
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

先导入构图的库和 TwoLayerNet 的神经网络文件。

设置图的默认大小

返回相对误差

```python
class TwoLayerNet(object):
    """..."""

    def __init__(self, input_size, hidden_size, output_size, std=1e-4):
        """..."""
        self.params = {}
        self.params['W1'] = std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)
```

构造两层的神经网络（部分代码）

初始化模型，权重初始化为较小的随机值，然后偏差被初始化为零。权重和偏差存储在变量 self.params。

```
def loss(self, X, y=None, reg=0.0):
    """..."""
    # Unpack variables from the params dictionary
    W1, b1 = self.params['W1'], self.params['b1']
    W2, b2 = self.params['W2'], self.params['b2']
    N, D = X.shape
```

传参生成数组矩阵。

```
    # Compute the forward pass
    scores = None
    ...
    h_output = np.maximum(0, X.dot(W1)+b1)  # ReLU activation
    scores = h_output.dot(W2) + b2
    ...
    if y is None:
        return scores
```

计算前向传播

```
    # Compute the loss
    loss = None
    ...
    temp = np.transpose(np.exp(scores)) / np.sum(np.exp(scores), axis=1)
    softmax_output = np.transpose(temp)
    loss = -np.sum(np.log(softmax_output[range(N), list(y)]))
    loss /= N
    loss += 0.5 * reg * (np.sum(W1*W1) + np.sum(W2 * W2))
```

完成前向传播并利用 Softmax 分类器计算训练示例和正则化过程中的平均交叉熵 loss。

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2}\lambda\sum_k\sum_l W_{k,l}^2}_{\text{regularization loss}}$$

计算 loss 对应的公式。

```
# Backward pass: compute gradients
grads = {}
...
dscores = softmax_output.copy()   # how this come from please s
dscores[range(N), list(y)] -= 1
dscores /= N
grads['W2'] = h_output.T.dot(dscores) + reg * W2
grads['b2'] = np.sum(dscores, axis = 0)

dh = dscores.dot(W2.T)
dh_ReLu = (h_output > 0) * dh
grads['W1'] = X.T.dot(dh_ReLu) + reg * W1
grads['b1'] = np.sum(dh_ReLu, axis = 0)
...

return loss, grads
```

计算反向传播梯度，通过计算权重和偏差的导数。并把结果存储在梯度字典中。

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad\qquad L_i = -\log\left(p_{y_i}\right)$$

$$\frac{\partial L_i}{\partial f_k} = p_k - 1(y_i = k)$$

反向传播梯度计算的公式。

```python
def train(self, X, y, X_val, y_val,
          learning_rate=1e-3, learning_rate_decay=0.95,
          reg=5e-6, num_iters=100,
          batch_size=200, verbose=False):
    """Train this neural network using stochastic gradient descent...."""
    num_train = X.shape[0]
    iterations_per_epoch = max(num_train / batch_size, 1)

    # Use SGD to optimize the parameters in self.model
    loss_history = []
    train_acc_history = []
    val_acc_history = []
```

使用随机梯度下降训练该神经网络。（部分代码）

```python
# Use SGD to optimize the parameters in self.model
loss_history = []
train_acc_history = []
val_acc_history = []

for it in xrange(num_iters):
    X_batch = None
    y_batch = None

    ...
    idx = np.random.choice(num_train, batch_size, replace=True)
    X_batch = X[idx]
    y_batch = y[idx]
    ...
    loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
    loss_history.append(loss)

    ...
    self.params['W2'] += - learning_rate * grads['W2']
    self.params['b2'] += - learning_rate * grads['b2']
```

使用随机梯度下降对 self.model 中的参数进行优化。（部分代码）

```python
        if verbose and it % 100 == 0:
          print('iteration %d / %d: loss %f' % (it, num_iters, loss))

        # Every epoch, check train and val accuracy and decay learning rate.
        if it % iterations_per_epoch == 0:
          # Check accuracy
          train_acc = (self.predict(X_batch) == y_batch).mean()
          val_acc = (self.predict(X_val) == y_val).mean()
          train_acc_history.append(train_acc)
          val_acc_history.append(val_acc)

          # Decay learning rate
          learning_rate *= learning_rate_decay

      return {
        'loss_history': loss_history,
        'train_acc_history': train_acc_history,
        'val_acc_history': val_acc_history,
      }
```

对迭代的每个时期检查训练和 val 的准确性和衰减学习率。

```python
def predict(self, X):
    """..."""
    y_pred = None

    ...
    h = np.maximum(0, X.dot(self.params['W1']) + self.params['b1'])
    scores = h.dot(self.params['W2']) + self.params['b2']
    y_pred = np.argmax(scores, axis=1)
    ...

    return y_pred
```

使用经过训练的两层网络权重来预测数据点。对于每个数据点，预测每个 C 的得分类别，并将每个数据点分配给得分最高的类别。

二、具体代码运行对应的结果

```python
input_size = 4
hidden_size = 10
num_classes = 3
num_inputs = 5

def init_toy_model():
    np.random.seed(0)
    return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1)

def init_toy_data():
    np.random.seed(1)
    X = 10 * np.random.randn(num_inputs, input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

net = init_toy_model()
X, y = init_toy_data()
print(X, y)
```

```
[[ 16.24345364  -6.11756414  -5.28171752 -10.72968622]
 [  8.65407629 -23.01538697  17.44811764  -7.61206901]
 [  3.19039096  -2.49370375  14.62107937 -20.60140709]
 [ -3.22417204  -3.84054355  11.33769442 -10.99891267]
 [ -1.72428208  -8.77858418   0.42213747   5.82815214]] [0 1 2 2 1]
```

```python
scores = net.loss(X)
print('Your scores:')
print(scores)
print()
```

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]
```

```
print('correct scores:')
correct_scores = np.asarray([
  [-0.81233741, -1.27654624, -0.70335995],
  [-0.17129677, -1.18803311, -0.47310444],
  [-0.51590475, -1.01354314, -0.8504215 ],
  [-0.15419291, -0.48629638, -0.52901952],
  [-0.00618733, -0.12435261, -0.15226949]])
print(correct_scores)
print()
```

```
correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]
```

```
# The difference should be very small. We get < 1e-7
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_scores)))
```

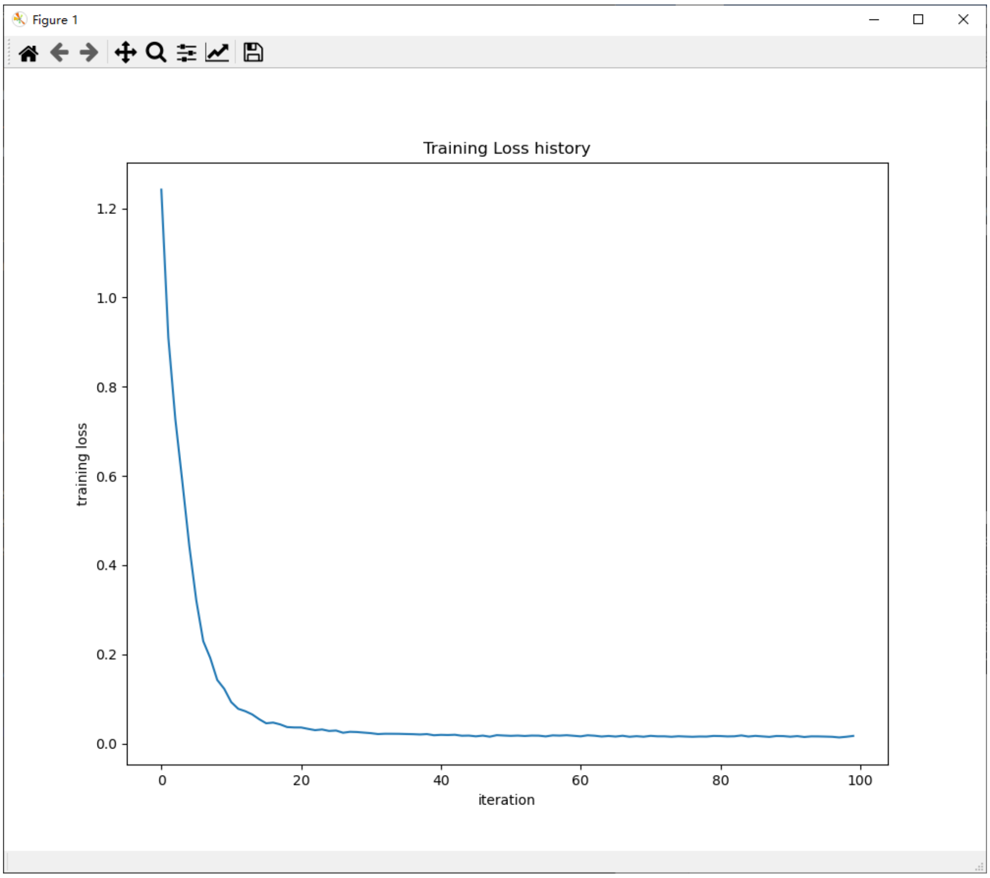
```
Difference between your scores and correct scores:
3.6802720496109664e-08
```

```
loss, _ = net.loss(X, y, reg=0.05)
correct_loss = 1.30378789133

# should be very small, we get < 1e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
Difference between your loss and correct loss:
0.018965419606062905
```

```python
# these should all be less than 1e-8 or so
for param_name in grads:
    f = lambda W: net.loss(X, y, reg=0.05)[0]
    param_grad_num = eval_numerical_gradient(f, net.params[param_name], verbose=False)
    print('%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[param_name])))

net = init_toy_model()
stats = net.train(X, y, X, y,
            learning_rate=1e-1, reg=5e-6,
            num_iters=100, verbose=False)

print('Final training loss: ', stats['loss_history'][-1])

# plot the loss history
plt.plot(stats['loss_history'])
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')
plt.show()
```

```
W2 max relative error: 3.440708e-09
b2 max relative error: 3.865028e-11
W1 max relative error: 3.561318e-09
b1 max relative error: 2.738422e-09
Final training loss:  0.017143643532923733
```



（此处为玩具数据模型的 loss 曲线）

```
# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

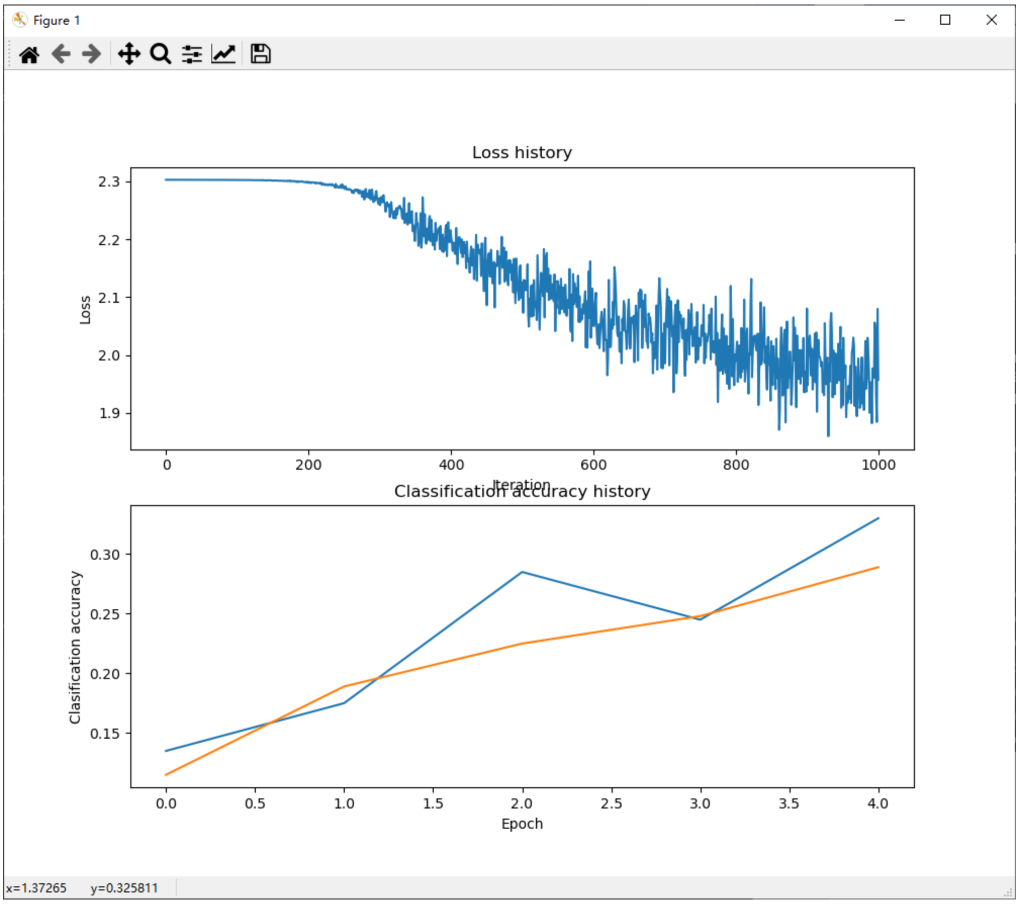
```
if verbose and it % 100 == 0:
    print('iteration %d / %d: loss %f' % (it, num_iters, loss))
```

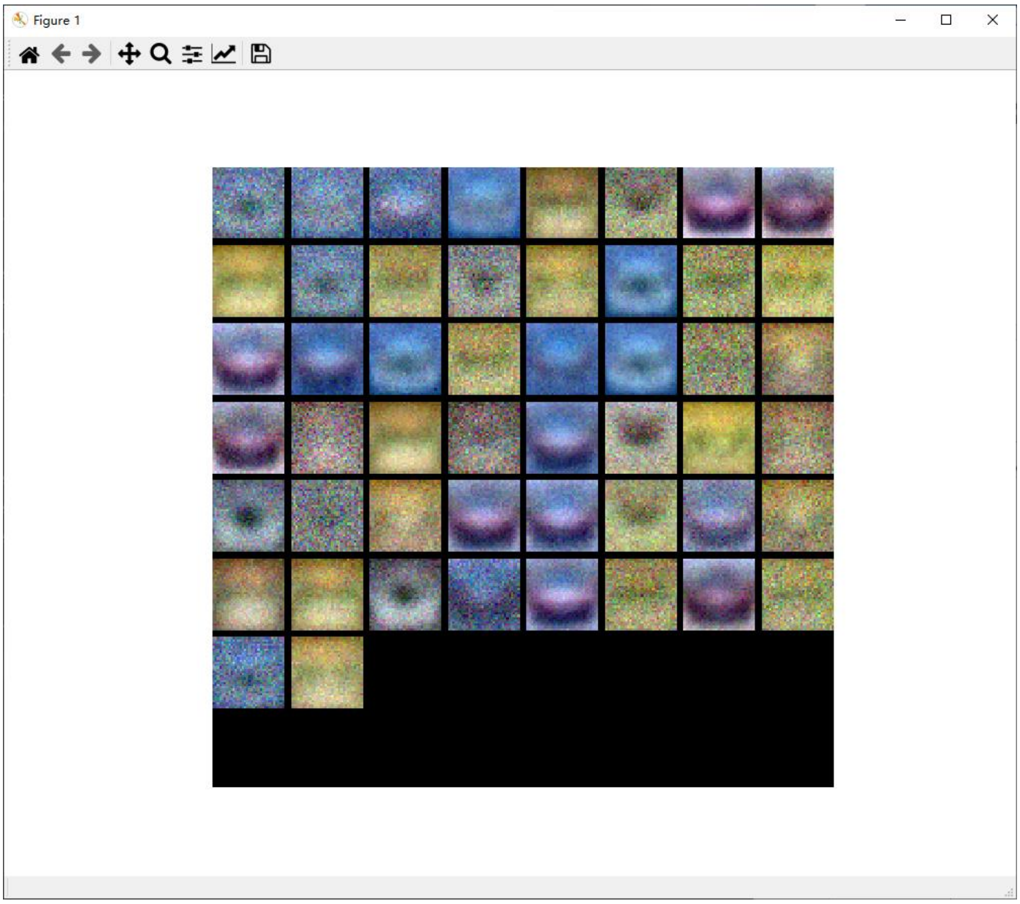
```
iteration 0 / 1000: loss 2.302762
iteration 100 / 1000: loss 2.302358
iteration 200 / 1000: loss 2.297404
iteration 300 / 1000: loss 2.258897
iteration 400 / 1000: loss 2.202975
iteration 500 / 1000: loss 2.116816
iteration 600 / 1000: loss 2.049789
iteration 700 / 1000: loss 1.985711
```

```
# Predict on the validation set
val_acc = (net.predict(X_val) == y_val).mean()
print('Validation accuracy: ', val_acc)
```

```
Validation accuracy:  0.287
```

```
# Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(stats['loss_history'])
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(stats['train_acc_history'], label='train')
plt.plot(stats['val_acc_history'], label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
plt.show()
```



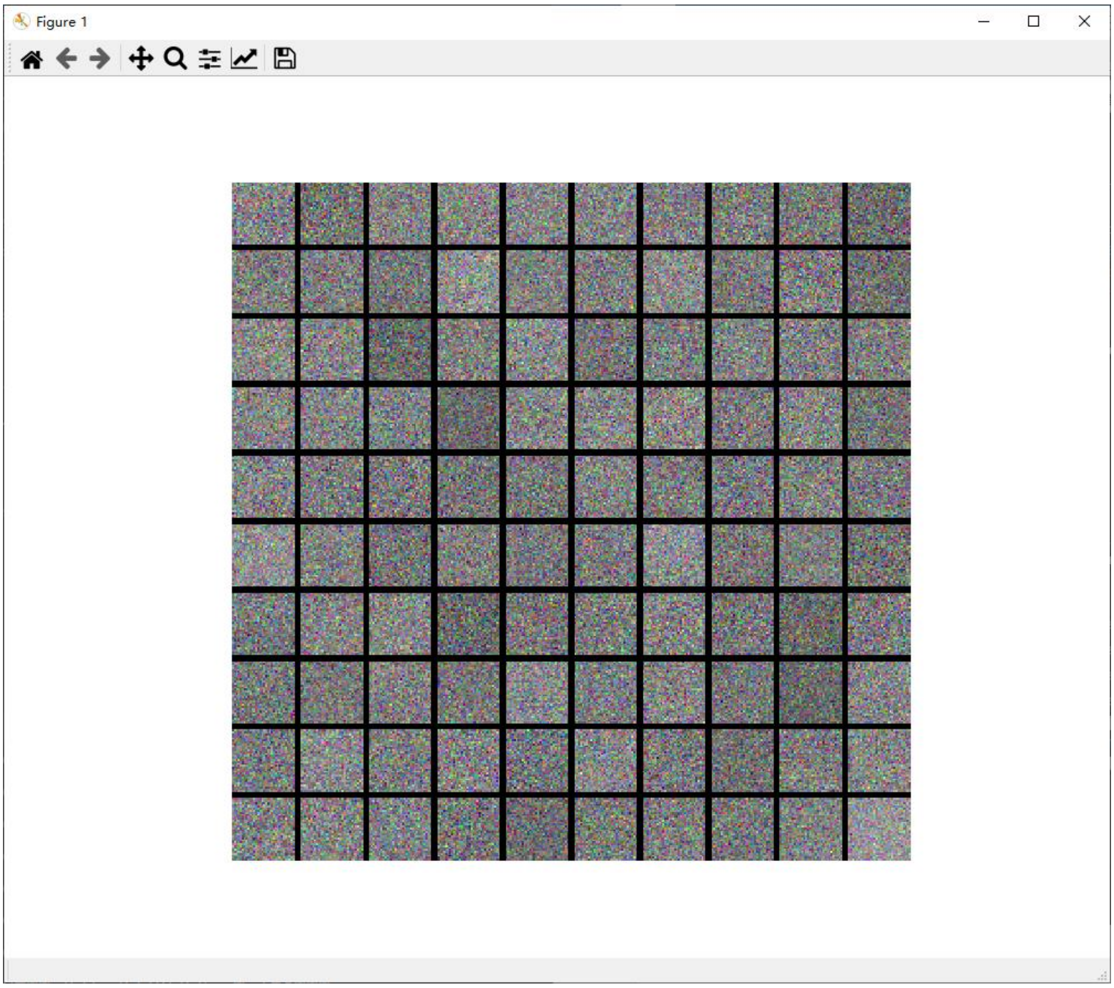



```python
print("running")
for hs in hidden_size:
    for lr in learning_rates:
        for reg in regularization_strengths:
            net = TwoLayerNet(input_size, hs, num_classes)
            stats = net.train(X_train, y_train, X_val, y_val,
                              num_iters=10, batch_size=200,
                              learning_rate=lr, learning_rate_decay=0.95,
                              reg=reg, verbose=False)
            val_acc = (net.predict(X_val) == y_val).mean()
            if val_acc > best_val_acc:
                best_val_acc = val_acc
                best_net = net
            results[(hs, lr, reg)] = val_acc

print("finished")

for hs, lr, reg in sorted(results):
    val_acc = results[(hs, lr, reg)]
    print('hs %d lr %e reg %e val accuracy: %f' % (hs, lr, reg, val_acc))

print('best validation accuracy achieved during cross-validation: %f' % best_val_acc)
```

```
running
finished
hs 75 lr 3.000000e-04 reg 5.000000e-01 val accuracy: 0.206000
hs 75 lr 3.000000e-04 reg 7.500000e-01 val accuracy: 0.148000
hs 75 lr 3.000000e-04 reg 1.000000e+00 val accuracy: 0.175000
hs 75 lr 3.000000e-04 reg 1.250000e+00 val accuracy: 0.185000
```

```
best validation accuracy achieved during cross-validation: 0.252000
```

```
# visualize the weights of the best network
show_net_weights(best_net)

test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```



（因为原代码中设置的迭代次数过大导致运行不出结果，此处改成了10，因此结果可能不是很准确。）

```
Test accuracy:  0.229
```

三、遇到的问题与解决措施

1。对于数据集的导入问题，刚开始没搞清楚问题的源头，后来发现数据集的路径没有到最后解压文件的根目录。

通过对比网上原代码中的文件路径发现并解决。

2. `from cs231n.classifiers.neural_net import TwoLayerNet` 对类似于这种导入已经写好的 py 文件的时候显示无法导入。

通过查阅网上的资料发现是 pycharm 这个软件的问题，需要手动设置根目录。

3.最后是对于代码的理解方面的问题。

首先对一些第一次看到的 numpy 函数，通过网上查阅 numpy 函数的使用和请教同学得到了一定的理解。

其次是，对神经网络中反向传播梯度的计算中的代码与原数学公式的转化，通过自己推导和请教同学得到了一定的理解。