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：a simple image classification pipeline		学号：201600181073
日期：2019.3.15	班级： 智能 16	姓名： 唐超
Email：		

实验目的：

• understand the basic Image Classification pipeline and the data-driven approach;
• implement and apply a softmax classifier;
• implement and apply a three-layer neural network classifier.

实验软件和硬件环境：

Spyder&python3.6

实验原理和方法：

1. Softmax classifier

设第 i 个 example 的第 j 个 feature 为 x_{ij}，$i=1,2,...,N$，$j=1,2,...,D$，则

$$X=\begin{pmatrix} x_{11} & x_{21} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ \vdots & \vdots & & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{pmatrix}_{N\times D}.$$

设第 j 个 feature 对下一个 layer 的第 k 个结点（第 k 个 label）的 weight 为 w_{jk}，$k=0,1,...,C-1$， 则

$$W=\begin{pmatrix} w_{10} & w_{11} & \cdots & w_{1,C-1} \\ w_{20} & w_{21} & \cdots & w_{2,C-1} \\ \vdots & \vdots & & \vdots \\ w_{D0} & w_{D1} & \cdots & w_{D,C-1} \end{pmatrix}_{D\times C}.$$

设第 i 个 example 再第 k 个 label 上的取值为 $l_{ik}=\sum_{j=1}^{D} x_{ij}w_{jk}$，则

$$L = XW = \begin{pmatrix} l_{10} & l_{11} & \cdots & l_{1,C-1} \\ l_{20} & l_{21} & \cdots & l_{2,C-1} \\ \vdots & \vdots & & \vdots \\ l_{N0} & l_{N1} & \cdots & l_{N,C-1} \end{pmatrix}_{N\times C}$$

经过 softmax，第 i 个 example 属于 label k=r, $r = 0,1,\ldots,C-1$ 的概率为

$$q_{ir} = \frac{e^{l_{ir}}}{\sum_k e^{l_{ik}}}$$

从而得到所有 examples 的预测概率矩阵：$Q = \left(q_{ik}\right)_{N\times C}$.

y 为 N 个 example 的实际 label 向量，$0 \le y_i \le C-1$，设 $\hat{y} = (\hat{y}_{ik})_{N\times C}$ 为理想的概率预测矩阵，其中 $\hat{y}_{ik} = \begin{cases} 1, & \text{k=y}_i \\ 0, & \text{others} \end{cases}$.

用交叉熵作为 $Loss$ 函数，则

$$Loss = -\sum_{i=1}^{N}\sum_{k=0}^{C-1} \hat{y}_{ik} \ln q_{ik} = -\sum_{i=1}^{N} \ln q_{i,y_i} ,$$

进一步作平均并正则化（正则化系数为 reg ）：

$$Loss = \frac{-\sum_{i=1}^{N} \ln q_{i,y_i}}{N} + \frac{1}{2} reg \cdot \sum_j \sum_k w_{jk}^2 .$$

由求导的链式法则，$Loss$ 对权重 w_{jk} 的导数为：

$$\frac{\partial Loss}{\partial w_{jk}} = \frac{1}{N} \cdot \left(-\sum_{i=1}^{N} \frac{1}{q_{i,y_i}} \cdot \frac{\partial q_{i,y_i}}{\partial l_{ik}} \cdot \frac{\partial l_{ik}}{\partial w_{jk}} \right) + \frac{\partial \frac{1}{2} reg \cdot \sum_j \sum_k w_{jk}^2}{\partial w_{jk}} ,$$

其中，$\frac{\partial q_{i,y_i}}{\partial l_{ik}} = \begin{cases} q_{ik}(1-q_{ik}), & k = \text{y}_i \\ -q_{ik} q_{i,y_i}, & k \ne \text{y}_i \end{cases}$，$\frac{\partial l_{ik}}{\partial w_{jk}} = x_{ij}$，因此

$$\frac{\partial Loss}{\partial w_{jk}} = \frac{1}{N} \cdot \left(\sum_{i=1}^{N} I(k = y_i) \cdot x_{ij} \right) + reg \cdot w_{jk} ,$$

其中 $I(k = y_i) = \begin{cases} q_{ik} - 1, & \text{k=y}_i \\ q_{ik}, & \text{k} \ne \text{y}_i \end{cases}$.

对所有 weight 分别求导的结果即为一个与 W 相同的大小的矩阵。

2. three-layer neural network

主要流程：输入数据，计算 loss，反向传播，得到参数梯度，更新参数，如此不断迭代，得到一组模型参数。

调整超参：将需要调整的超参：learning rate、hidden size、batch size、正则化系数可能数值分别罗列（网格化），用网格中每一个点所代表的参数进行训练、验证，找到最优的网格点，并在该网格点附近细化网格尝试寻找更优的参数组合。

实验步骤：（不要求罗列完整源代码）

1. Softmax classifier

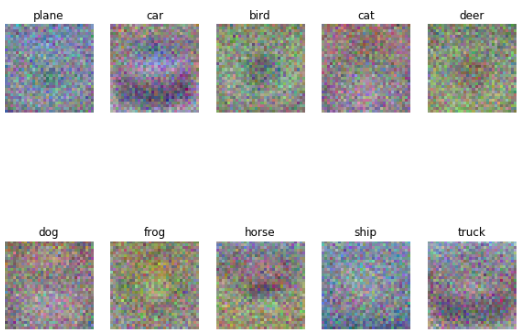
通过循环和向量化操作计算 loss 及 gradient：

```python
def softmax_loss_naive(W, X, y, reg):

  # Initialize the loss and gradient to zero.
  loss = 0.0
  dW = np.zeros_like(W)

  N = X.shape[0]

  for i in range(N):
    l = np.dot(X[i, :], W)
    l_softmax = np.exp(l)/np.sum(np.exp(l))
    loss -= np.log(l_softmax[y[i]])

    for j in range(W.shape[0]):
      for k in range(W.shape[1]):
        if k != y[i]:
          dW[j, k] += np.dot(X.T[j, i], l_softmax[k])
        else:
          dW[j, k] += np.dot(X.T[j, i], l_softmax[k] - 1)

  loss /= N
  loss += 0.5 * reg * np.sum(W**2)

  dW /= N
  dW += reg * W


  return loss, dW
```

```python
def softmax_loss_vectorized(W, X, y, reg):
  """
  Softmax loss function, vectorized version.

  Inputs and outputs are the same as softmax_loss_naive.
  """
  # Initialize the loss and gradient to zero.
  loss = 0.0
  dW = np.zeros_like(W)

  num_train = X.shape[0]  #N

  # loss
  scores = np.dot(X, W) # N*C 的矩阵 表示每个example被label为Ci的score
  exp_scores = np.exp(scores)
  prob_scores = exp_scores/np.sum(exp_scores, axis=1, keepdims=True)  # softmax
  correct_log_probs = -np.log(prob_scores[range(num_train), y])  # 交叉熵损失函数
  loss = np.sum(correct_log_probs)
  loss /= num_train
  loss += 0.5 * reg * np.sum(W**2)  # 正则化

  # grads
  dscores = prob_scores
  dscores[range(num_train), y] -= 1
  dW = np.dot(X.T, dscores)
  dW /= num_train
  dW += reg * W

  return loss, dW
```

最终测试结果如下：

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.352571 val accuracy: 0.372000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.333122 val accuracy: 0.359000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.354000 val accuracy: 0.372000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.327224 val accuracy: 0.342000
best validation accuracy achieved during cross-validation: 0.372000
softmax on raw pixels final test set accuracy: 0.360000
```



2. three-layer neural network

训练过程：

（1） 随机抽取 batch size 大小的数据

```
shuffle_indexes = np.arange(num_train)
np.random.shuffle(shuffle_indexes)
shuffle_indexes = shuffle_indexes[0:batch_size-1]  # 从训练集中随机选取b
X_batch = X[shuffle_indexes, :]
y_batch = y[shuffle_indexes]
```

（2） 正向传播，计算各类别的 scores

```
layer1 = np.dot(X, W1) + b1
layer1[layer1 < 0] = 0
layer2 = np.dot(layer1, W2) + b2
layer2[layer2 < 0] = 0
scores = np.dot(layer2, W3) + b3
```

（3） 计算 loss

```
exp_scores = np.exp(scores)
prob_scores = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)  # softmax 得到每个类别的概率
correct_log_probs = -np.log(prob_scores[range(len(y)), y])  # 交叉熵损失函数
loss = np.sum(correct_log_probs)  # 损失函数
loss /= len(y)
loss += 0.5 * reg * np.sum(W1 ** 2) + 0.5 * reg * np.sum(W2 ** 2) + 0.5 * reg * np.sum(W3 ** 2) # 正则
```

（3） 反向传播计算 gradient

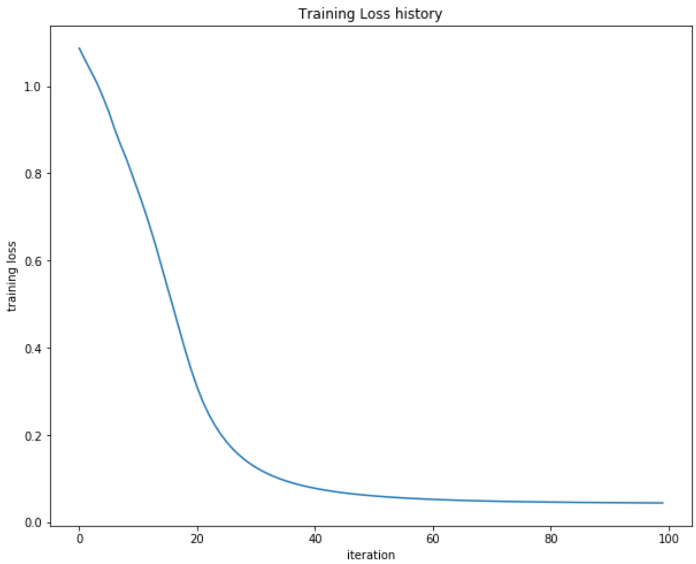
```
dscores[range(len(y)), y] -= 1
dscores /= len(y)

grads['W3'] = np.dot(layer2.T, dscores)
grads['W3'] += reg*W3
grads['b3'] = np.sum(dscores, axis=0)

dlayer2 = np.dot(dscores, W3.T)
dlayer2[layer2 < 0] = 0
grads['W2'] = np.dot(layer1.T, dlayer2)
grads['W2'] += reg*W2
grads['b2'] = np.sum(dlayer2, axis=0)

dlayer1 = np.dot(dlayer2, W2.T)
dlayer1[layer1 < 0] = 0
grads['W1'] = np.dot(X.T, dlayer1)
grads['W1'] += reg*W1
grads['b1'] = np.sum(dlayer1, axis=0)
```

（5） 更新参数

```
self.params['W1'] -= learning_rate*grads['W1']
self.params['W2'] -= learning_rate*grads['W2']
self.params['W3'] -= learning_rate*grads['W3']
self.params['b1'] -= learning_rate*grads['b1']
self.params['b2'] -= learning_rate*grads['b2']
self.params['b3'] -= learning_rate*grads['b3']
```

在 toy_model 上的做简单训练：

```
Your scores:
[[-0.03596154 -0.01613583 -0.00048556]
 [-0.09480192  0.20724618 -0.09763798]
 [-0.07015667  0.17081869 -0.07675745]
 [ 0.01473052  0.09321097 -0.0395178 ]
 [-0.05306489  0.04729807  0.01750587]]
Final training loss:  0.04526310220887222
```



调整超参：第一次设置

batch_size=[100, 150, 200];

hidden_size=[64, 128, 256, 512];

learning_rate=[0.1, 0.05, 0.01, 0.005, 0.001];

regularization_strengths = [0.1, 0.05, 0.01,0.005, 0.001],

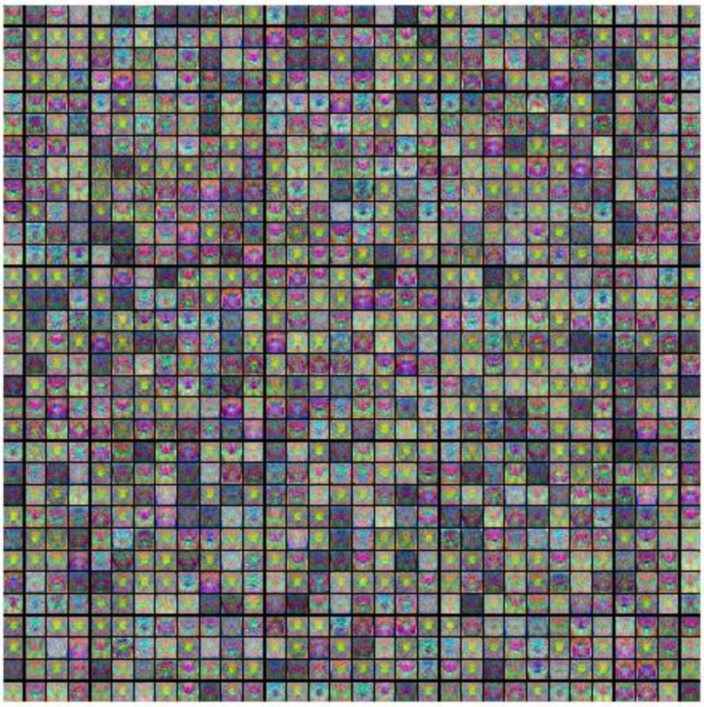
进行训练，发现 batch_size 和 hidden_size 都取最大，learning_rate 取 0.005，正则化系数取 0.01 或 0.001 时效果较好，部分结果如下：

```
Training params: batch size: 200, hidden size: 512, learning rate: 0.005,
reg strength: 0.001
iteration 0 / 2000: loss 2.302594
iteration 100 / 2000: loss 2.302618
iteration 200 / 2000: loss 2.302680
iteration 300 / 2000: loss 2.302849
iteration 400 / 2000: loss 2.302618
iteration 500 / 2000: loss 2.302615
iteration 600 / 2000: loss 2.302695
iteration 700 / 2000: loss 2.302585
iteration 800 / 2000: loss 2.300595
iteration 900 / 2000: loss 2.104757
iteration 1000 / 2000: loss 2.069014
iteration 1100 / 2000: loss 2.103755
iteration 1200 / 2000: loss 2.039024
iteration 1300 / 2000: loss 2.054273
iteration 1400 / 2000: loss 1.939294
iteration 1500 / 2000: loss 1.915237
iteration 1600 / 2000: loss 1.973038
iteration 1700 / 2000: loss 1.868560
iteration 1800 / 2000: loss 1.979711
iteration 1900 / 2000: loss 1.824305
8 train_accuracy: 0.32957142857142857 val_accuracy: 0.33
```

在上一次调参的经验基础上，在最优参数附近进一步调整参数，并提高迭代次数，得到了更高的准确率：

Training params: batch size: 250, hidden size: 1024, learning rate: 0.005, reg strength: 0.0001

```
2 train_accuracy: 0.43204081632653063 val_accuracy: 0.44
[]
```



```
Test accuracy:  0.404
```

结论分析与体会：

用简单的三层神经网络实现了图片的分类，尽管只填写了部分函数，但这一过程使我对神经网络的工作流程和原理有了更具体的认识。

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：

1. 在开始做实验时对 softmax 函数不了解，也不知道怎么求 loss 和 gradient：查找了 softmax 的相关资料并做了总结。
2. 调整超参时，没注意提高迭代次数，迭代次数过少导致 loss 并没有收敛：将迭代次数从 2000 提高到了 5000。