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：神经网络实现		学号：201600301304
日期：2019/3/16	班级：智能 16	姓名： 贾乘兴
Email：1131225623@qq.com		

实验目的：

1.实现神经网络的前向传播、反向传播算法（bp 算法），并使用多种方式进行测试

2.实现 softmax 层并进行测试（cifar-10 数据集）

3.可视化 cifar-10 各类的图像对应的参数与神经网络的参数

实验软件和硬件环境：

硬件：16GB 内存

软件：mac os 编译器 pycharm

实验原理和方法：

一．Softmax

1.softmax 的前向计算

对于输入的 H 维向量 x，以 x(j)表示其第 j 个元素，我们需要得到最终的概率 p，p 是 C 维向量，p(i)表示其第 i 个元素，同时也表示 x 属于第 i 类的概率。中间我们要经过全连接运算得到与 p 同等大小的 score，再由 score 计算得到最终的概率 p

第一步计算 score 的公式如下，计算 score 的每个元素

$$score_i = \sum_{j=1}^{H} W_{ij} x_j + b_i$$

其中 W 矩阵为 H*C 的矩阵，b 为 C 维向量，矩阵形式计算公式如下

$$score = Wx + b$$

第二步由 score 计算概率 p，考虑到 score 各个元素的数值和求导运算等的影响，我们首先进行标准化的处理，先减去最大值，后取指数，得到比重则为 p 的各个元素的分量

减去最大值的操作是使指数运算不会溢出，同时可以证明这一操作对求导等计算没有增加计算量的影响

$$p_i = \frac{e^{score_i - score_{\max}}}{\sum_{k=1}^{C} e^{score_k - score_{\max}}} = \frac{e^{score_i} \Big/ e^{score_{\max}}}{\sum_{k=1}^{C} e^{score_k} \Big/ e^{score_{\max}}} = \frac{e^{score_i}}{\sum_{k=1}^{C} e^{score_k}}$$

最终我们得到了归一化的概率 p

2.softmax 的反向传播（loss function 与梯度计算）

对于输入的向量 x，我们有它的真实类别值 y，y 的数字 c 代表属于哪一类

在训练过程中我们目的是更新参数 W，使得 softmax 在测试集的准确率得到提升，所以我们使用梯度下降的方式进行参数的更新，对于损失函数，在该分类任务上我们使用的损失函数是交叉熵损失函数

由于该多分类得到的是概率，所以我们可以视为得到一个分布去逼近真实的分布，即真实分

布与假设的分布的距离的大小计算，而计算两个分布的距离的是 KL 散度，公式如下

$$KL(p\|q)=\sum_x p(x)\log\frac{p(x)}{q(x)}$$

将 KL 散度拆开，可以得到两部分

$$KL(p\|q)=\sum_x p(x)\log\frac{p(x)}{q(x)}=-\sum_x p(x)\log q(x)+\sum_x p(x)\log p(x)$$

可知后一部分为负的真实分布的熵，该部分为确定数值，前一部分为交叉熵

$$CrossEntropy(p,q)=-\sum_x p(x)\log q(x)$$

故我们取交叉熵作为 loss function，用来计算 loss

$$loss=\sum_{k=1}^{C} y_k \log p_k$$

定义了 loss function 之后，我们需要最小化 loss，需要对参数进行优化，参数的优化如下

$$W_{ij}:=W_{ij}-\eta\frac{\partial loss}{\partial W_{ij}}$$

其中艾塔是学习率，为超参数，而偏导数可以通过链式求导法则得到

$$\frac{\partial loss}{\partial W_{ij}}=\sum_{k=1}^{C}\left(\frac{\partial loss}{\partial p_k}\frac{\partial p_k}{\partial score_i}\right)\bullet\frac{\partial score_i}{\partial W_{ij}}$$

$$\frac{\partial loss}{\partial p_k}=-\frac{y_k}{p_k}$$

$$\frac{\partial p_k}{\partial score_i}=\begin{cases}p_i(1-p_i), i=k\\-p_i p_k, i\neq k\end{cases}$$

$$\frac{\partial loss}{\partial score_i}=\sum_{k=1}^{C}\left(\frac{\partial loss}{\partial p_k}\frac{\partial p_k}{\partial score_i}\right)=p_i-y_i$$

所以我们得到最终的导数为

$$\frac{\partial loss}{\partial W_{ij}}=(p_i-y_i)x_j$$

其矩阵形式为

$$\frac{\partial loss}{\partial W}=X^T(p-y)$$

得到梯度后我们可以进行反向传播更新参数（使用随机梯度下降）

3. 将训练好的参数 W 对应每个种类可视化，即与该类连接的参数图像形式可视化

4. 数值解梯度的误差分析

对于泰勒展开，我们有

$$f(x+h)=f(x)+hf'(x)+\frac{h^2}{2}f''(x)+\frac{h^3}{6}f'''(\xi)$$

$$f(x-h)=f(x)-hf'(x)+\frac{h^2}{2}f''(x)-\frac{h^3}{6}f'''(\zeta)$$

$$f(x+h)-f(x-h)=2hf'(x)+\frac{h^3}{3}f'''(\psi)$$

整理得

$$f'(x)=\frac{f(x+h)-f(x-h)}{2h}-\frac{h^2}{2}f'''(c)$$

故本实验的数值梯度的计算精度为二次的

二．神经网络（三层神经网络）

1.前向计算

本次的神经网络的结构为

输入层→隐藏层 1→relu 函数→隐藏层 2→relu 函数→score→softmax→输出

输入为 D 维向量 x（或者 N 个样本的 N*D 矩阵 X），输出为 C 维向量 p（或者 N*C）

计算过程如下

$$h^1=W^1x+b^1$$
$$a^1=relu(h^1)$$
$$h^2=W^2a^1+b^2$$
$$a^2=relu(h^2)$$
$$h^3=W^3a^2+b^3$$
$$p=soft\max(h^3)$$

（relu 函数：relu（x）取值为 0，x<0，否则为 x）

2.损失函数与反向传播

损失函数为交叉熵函数，定义如 softmax

定义符号如下：

$$\delta^l=\frac{\partial loss}{\partial h^l}$$

可得

$$\delta_i^l=\sum_{h_j^{l+1}\in h^{l+1}}\frac{\partial loss}{\partial h_j^{l+1}}\frac{\partial h_j^{l+1}}{\partial a_i^l}\bullet\frac{\partial a_i^l}{\partial h_i^l}=\sum_{h_j^{l+1}\in h^{l+1}}\delta_j^{l+1}W_{ji}^{l+1}\bullet relu'(h_i^l)$$

整理为矩阵形式为

$$\delta^l=\left(\left(W^{l+1}\right)^T\delta^{l+1}\right)\odot relu'(h^l)$$

对 softmax 层的参数梯度计算如上，对全连接层的参数梯度计算如下

$$\frac{\partial loss}{\partial W_{ij}^{l}}=\frac{\partial loss}{\partial h_{i}^{l}}\frac{\partial h_{i}^{l}}{\partial W_{j}^{l}}=\delta_{i}^{l}a_{j}^{l-1}$$

$$\frac{\partial loss}{\partial b_{i}^{l}}=\frac{\partial loss}{\partial h_{i}^{l}}\frac{\partial h_{i}^{l}}{\partial b_{i}^{l}}=\delta_{i}^{l}$$

整理为矩阵形式为

$$\frac{\partial loss}{\partial W^{l}}=\frac{\partial loss}{\partial h^{l}}\frac{\partial h^{l}}{\partial W^{l}}=\delta^{l}\left(a^{l}\right)^{T}$$

$$\frac{\partial loss}{\partial b^{l}}=\frac{\partial loss}{\partial h^{l}}\frac{\partial h^{l}}{\partial b^{l}}=\delta^{l}$$

3. mini-batch, 在之前的梯度计算中，对 bias 求导后按行元素相加，然后所有的梯度结果处以 batch 的大小取均值

4. regularization

为了防止形成过拟合，需要进行正则化，更新的 loss function 如下

$$loss = CrossEntropy + \lambda \bullet Regularization$$

$$Regularization = \frac{1}{2}\sum_{W} W \odot W$$

对梯度计算的影响为所有的 dW 都需要再加上 lambda*W

实验步骤:（不要求罗列完整源代码）

1. 使用循环计算 softmax 的 loss 与 grad 的主要代码如下

```
N = X.shape[0]
C = W.shape[1]
z = np.dot(X, W)
loss = 0
for i in range(N):
  zx = z[i]
  zx = zx - np.max(zx)
  ex = np.exp(zx) / np.sum(np.exp(zx), axis=0)
  loss = loss - np.log(ex[y[i]])
  for j in range(C):
    dW[:, j] += np.dot(ex[j] , X[i])
  dW[:, y[i]] -= X[i]
loss = loss / N + 0.5 * reg * np.sum(W ** 2)
dW = dW / N + reg * W
```

2. 使用矩阵运算 softmax 的 loss 与 grad 的主要代码如下

```
z = np.matmul(X, W)
sz = z - np.max(z, axis=1)[:, np.newaxis]
ez = np.exp(sz)
pre_y = ez / np.sum(ez, axis=1)[:, np.newaxis]
```

```
lab_y = (np.arange(len(pre_y[0])) == y[:, None]).astype(np.integer)
loss = np.mean(- np.sum(lab_y * np.log(pre_y), axis=1) + 1/2 * reg *
np.sum(np.square(W)))
dW = - (np.matmul(X.T, (lab_y - pre_y))) / X.shape[0] + reg * W
```

3. scores 计算主要代码如下

```
def relu(h):
  index = np.where(h>0)
  h0 = np.zeros_like(h)
  h0[index] = h[index]
  return h0

def softmax(h):
  sh = h - np.max(h, axis=1)[:, np.newaxis]
  eh = np.exp(sh)
  scores = eh / np.sum(eh, axis=1)[:, np.newaxis]
  return scores

h1 = np.matmul(X,W1) + b1
h2 = np.matmul(relu(h1),W2) + b2
h3 = np.matmul(relu(h2),W3) + b3
scores = h3
```

4. loss 计算主要代码如下

```
pre_y = softmax(scores)
lab_y = (np.arange(len(pre_y[0])) == y[:, None]).astype(np.integer)
loss = -np.sum(np.log(pre_y[np.arange(N),y])) / N + 1/2 * reg *
(np.sum(W1*W1)+np.sum(W2*W2)+np.sum(W3*W3))
```

5. 梯度计算主要代码如下

```
h2_d = np.zeros_like(h2)
h2_d[relu(h2) > 0] = 1
h1_d = np.zeros_like(h1)
h1_d[relu(h1) > 0] = 1

#d3 = - lab_y + pre_y
d3 = np.zeros_like(pre_y)
d3[range(N),y] -= 1
d3 += pre_y
d2 = np.matmul(d3, W3.T) * h2_d
d1 = np.matmul(d2, W2.T) * h1_d

grads['W3'] = np.matmul(h2.T, d3) / N + reg * W3
grads['b3'] = np.sum(d3, axis=0) / N
```

```
grads['W2'] = np.matmul(h1.T, d2) / N + reg * W2
grads['b2'] = np.sum(d2, axis=0) / N
grads['W1'] = np.matmul(X.T, d1) / N + reg * W1
grads['b1'] = np.sum(d1, axis=0) / N
```

6. mini—batch 生成代码如下

```
idx = np.random.choice(num_train, batch_size, replace=True)
X_batch = X[idx]
y_batch = y[idx]
```

7. 梯度下降主要代码如下

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

8. predict 主要代码如下

```
def relu(h):
  index = np.where(h>0)
  h0 = np.zeros_like(h)
  h0[index] = h[index]
  return h0

def softmax(h):
  sh = h - np.max(h, axis=1)[:, np.newaxis]
  eh = np.exp(sh)
  scores = eh / np.sum(eh, axis=1)[:, np.newaxis]
  return scores

W1, b1 = self.params['W1'], self.params['b1']
W2, b2 = self.params['W2'], self.params['b2']
W3, b3 = self.params['W3'], self.params['b3']
h1 = np.matmul(X, W1) + b1
h2 = np.matmul(relu(h1), W2) + b2
h3 = np.matmul(relu(h2), W3) + b3
scores = softmax(h3)
y_pred = np.argmax(scores,axis=1)
```

9. 运行 softmax_train.py，得到相应结果

10. 运行 three_layer_net.py，得到相应结果

结论分析与体会：
结果：
一. softmax 分类
1. 初始化并计算梯度与梯度数值解对比，计算与数值解的相对误差
Train data shape: (49000, 3073)
Train labels shape: (49000,)
Validation data shape: (1000, 3073)
Validation labels shape: (1000,)
Test data shape: (1000, 3073)
Test labels shape: (1000,)
dev data shape: (500, 3073)
dev labels shape: (500,)

loss: 2.357751
sanity check: 2.302585
numerical: -2.049885 analytic: -2.049885, relative error: 4.645181e-08
numerical: 2.560704 analytic: 2.560704, relative error: 1.935980e-08
numerical: 2.617564 analytic: 2.617564, relative error: 1.962999e-08
numerical: -2.430297 analytic: -2.430297, relative error: 5.723656e-10
numerical: 1.687667 analytic: 1.687667, relative error: 9.290069e-09
numerical: 1.565769 analytic: 1.565769, relative error: 9.707565e-09
numerical: 0.042216 analytic: 0.042215, relative error: 9.002679e-07
numerical: 1.391542 analytic: 1.391542, relative error: 2.027629e-08
numerical: -4.272710 analytic: -4.272710, relative error: 1.026857e-09
numerical: -0.010183 analytic: -0.010183, relative error: 3.210500e-06
numerical: 3.202150 analytic: 3.202150, relative error: 8.842856e-09
numerical: -2.112007 analytic: -2.112007, relative error: 1.232407e-08
numerical: 1.346648 analytic: 1.346648, relative error: 5.132151e-09
numerical: -0.836121 analytic: -0.836121, relative error: 9.405408e-08
numerical: 0.763234 analytic: 0.763233, relative error: 1.274807e-07
numerical: -3.694332 analytic: -3.694332, relative error: 6.605912e-09
numerical: 3.337983 analytic: 3.337982, relative error: 1.235690e-08
numerical: 0.080412 analytic: 0.080412, relative error: 6.679538e-07
numerical: 1.524004 analytic: 1.524004, relative error: 4.028002e-08
numerical: 3.331715 analytic: 3.331715, relative error: 6.260818e-09
可见数值解与解析解相对误差较小，二次精度

2. 矩阵运算与循环的对比
naive loss: 2.357751e+00 computed in 0.103219s
vectorized loss: 2.357751e+00 computed in 0.004399s
Loss difference: 0.000000
Gradient difference: 0.000000
可见同样的运算精度下，使用矩阵形式计算速度要快接近 250 倍

3. 不同的 learning—rate 与正则化权重测试

lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.347184 val accuracy: 0.357000

lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.328265 val accuracy: 0.341000

lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.351306 val accuracy: 0.371000

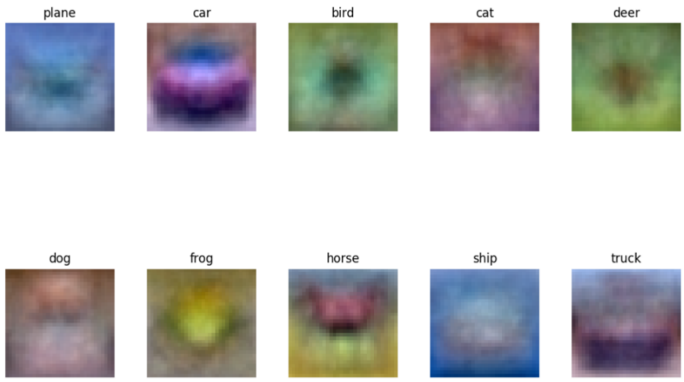
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.316633 val accuracy: 0.336000

best validation accuracy achieved during cross-validation: 0.371000

softmax on raw pixels final test set accuracy: 0.360000

正确率在 30-40%之间

可视化的各类权重为



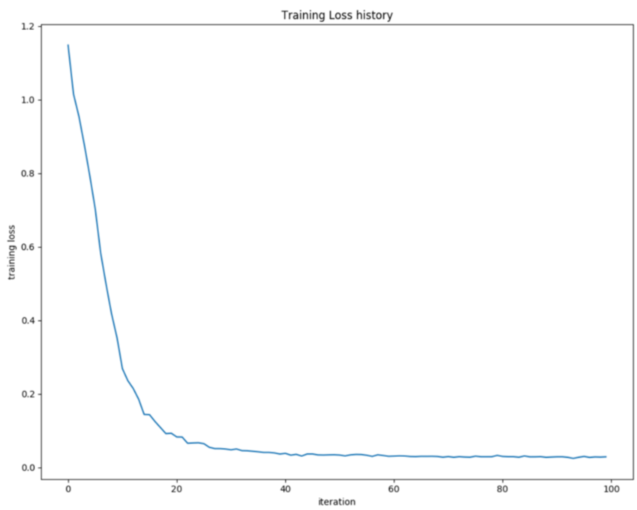
4. 三层神经网络，简单模型的计算结果（调参 H=20）

Your scores:

```
[[-0.38141594   0.0102428    0.42025884]
 [-0.31506721   0.24741116   0.45896028]
 [-0.35939606   0.19335479   0.13135735]
 [-0.13228368   0.1387217    0.07713126]
 [-0.11070252   0.01931671   0.0882341 ]]
```

Final training loss: 0.029048917813040233

训练过程中损失函数变化图像如下



5. 三层神经网络，cifar—10 的训练结果（调参）

Train data shape: (49000, 3072)

Train labels shape: (49000,)

Validation data shape: (1000, 3072)

Validation labels shape: (1000,)

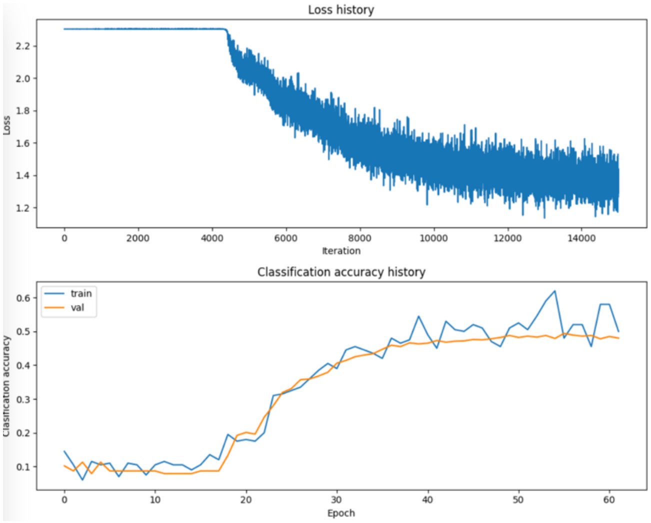
Test data shape: (1000, 3072)

Test labels shape: (1000,)

Validation accuracy: 0.489

Test accuracy: 0.478

训练过程中正确率与 loss 的变化图像



W1 的可视化如下

体会：

首先调参解决问题在参数较少时还可以解决，但结合之前使用 tensorflow 训练 cifar10，有很多方法如批标准化的重要性在这个问题得到了充分体现

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：
问题 1:softmax 求导错误，在计算中忽略了 softmax 是类似全连接的，直接对某个单元求导导致结果错误，重新计算才得到正确结果
问题 2:全连接反向传播求导错误，计算时没有对 batch 求均值，除以 n 解决之
问题 3:未调参数导致训练结果不好，甚至因为调整参数时发生了梯度爆炸现象。最终找到了较为合适的参数,同时通过正则化参数调整和学习率调整缓解了梯度爆炸的现象。
改进思路：可以加入 batch—normalization，但对求导有一定复杂度