17级种子班

数字图像处理课设

Week2-softmax总结

队伍成员：张志宇、李勉、刘羿

CONTENT

analytic grad推导及验证

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial f_i}\frac{\partial f_i}{\partial W_i} = (p_i - y_i)x$$

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}}$$

$$L = -\sum_j y_j \log p_j,$$

$$\begin{aligned}
\frac{\partial L}{\partial o_i} &= -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} \\
&= -\sum_k y_k \frac{1}{p_k}\frac{\partial p_k}{\partial o_i} \\
&= -y_i(1-p_i) - \sum_{k\neq i} y_k \frac{1}{p_k}(-p_k p_i) \\
&= -y_i(1-p_i) + \sum_{k\neq i} y_k (p_i) \\
&= -y_i + y_i p_i + \sum_{k\neq i} y_k (p_i) \\
&= p_i\left(\sum_k y_k\right) - y_i \\
&= p_i - y_i
\end{aligned}$$

```python
def numerical_grad(w):
    x = np.array([4,3,5])
    y = np.array([1,0])
    p = normalization(np.dot(x,w))
    loss = 0 - np.log(np.dot(y,p))
    return loss
```

```python
def analytic_grad(w):
    x = np.array([4,3,5])
    y = np.array([1,0])
    p = normalization(np.dot(x,w))
    return np.array(np.dot(np.mat(x).T, np.mat(p-y)))
```

analytic grad推导及验证

```
def numerical_gradient(f, x):
    fx = f(x)
    grad = np.zeros(x.shape)
    h = 0.00001
    it = np.nditer(x, flags = ['multi_index'] ,op_flags=['readwrite'])
    while not it.finished:
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h
        fxh = f(x)
        x[ix] = old_value

        grad[ix] = ((fxh - fx) / h)
        it.iternext()

    return grad
```

```
def analytic_grad(f, x):
    return f(x)
```

```
def numerical_grad(w):
    x = np.array([4,3,5])
    y = np.array([1,0])
    p = normalization(np.dot(x,w))
    loss = 0 - np.log(np.dot(y,p))
    return loss
```

```
def analytic_grad(w):
    x = np.array([4,3,5])
    y = np.array([1,0])
    p = normalization(np.dot(x,w))
    return np.array(np.dot(np.mat(x).T, np.mat(p-y)))
```

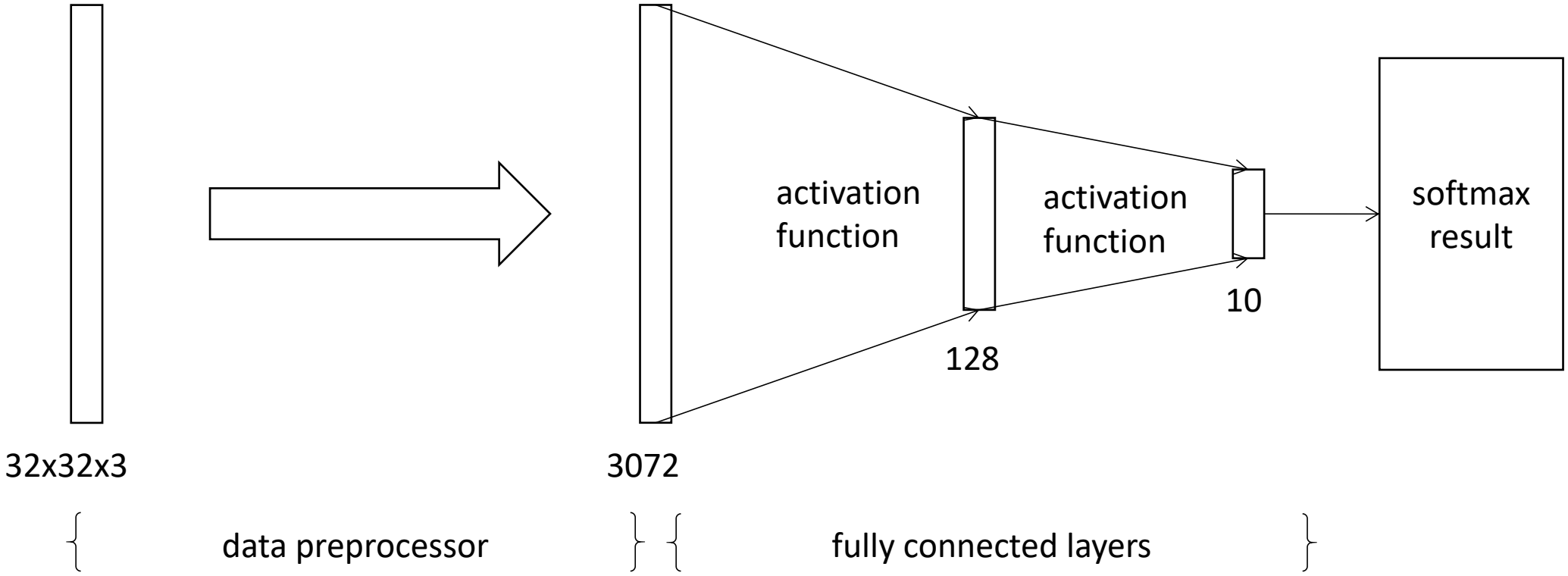
```
numerical_gradient(numerical_grad, w)
```

```
array([[-2.5365238 ,  2.53656092],
       [-1.90239633,  1.90241721],
       [-3.17064895,  3.17070696]])
```

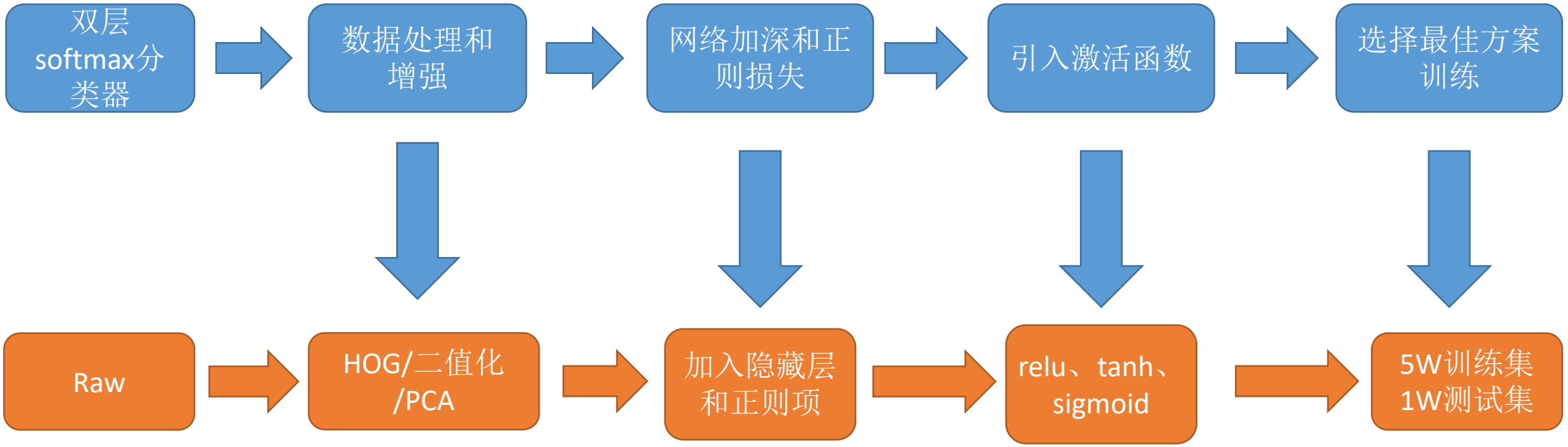
```
analytic_grad(w)
```

```
array([[-2.53654236,  2.53654236],
       [-1.90240677,  1.90240677],
       [-3.17067796,  3.17067796]])
```

整体网络结构


activation function
activation function
softmax result
10
128
32x32x3
3072
data preprocessor
fully connected layers

实验结果及改进 -- 实验流程



实验结果及改进 -- 数据处理

数据预处理

- 归一化(防止溢出)
- 二值化
- Hog
- PCA

测试基本条件：

- 用batch_1训练，batch_2测试
- 2层softmax分类器

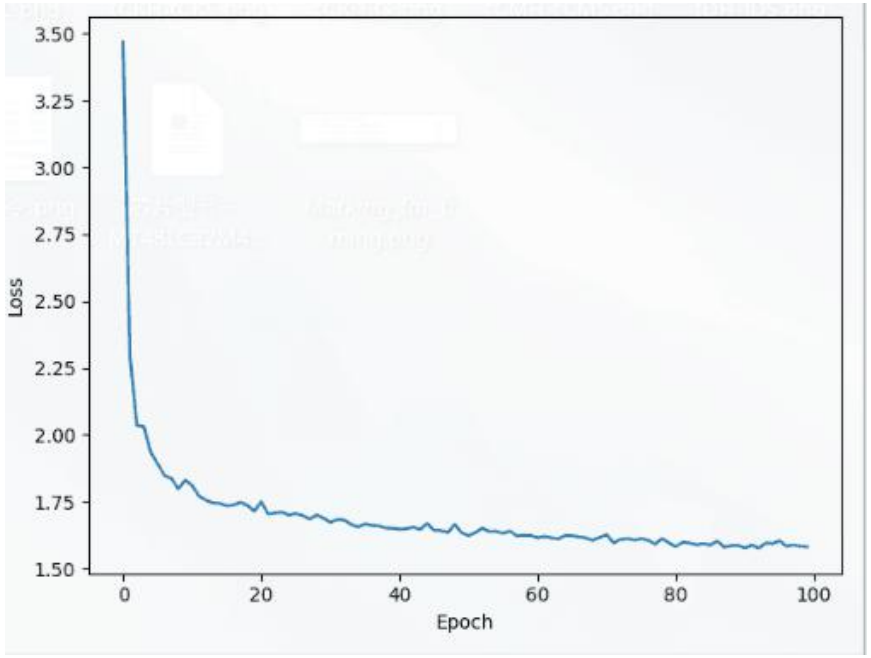
结果：

经过各种数据预处理之后准确率均有不同程度的降低，猜测是部分信息丢失导致的，因此最后只对数据做归一化。

实验结果及改进 -- 网络加深

增加隐藏层

- 增加网络参数，增强拟合效果

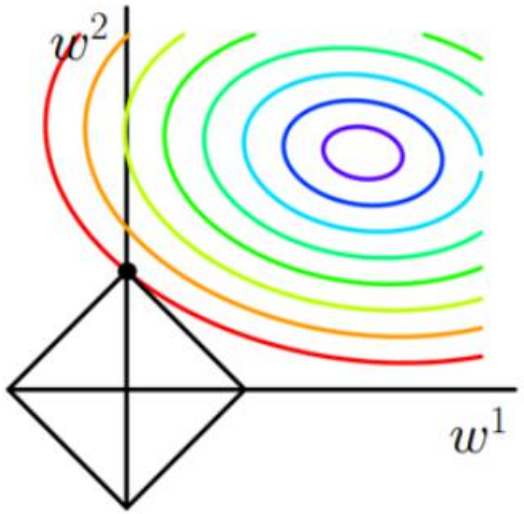



```
for epoch86, loss is 1.6028285767283748
for epoch87, loss is 1.580351269899901
for epoch88, loss is 1.5862254015838686
for epoch89, loss is 1.5877135808227425
for epoch90, loss is 1.576759931099618
for epoch91, loss is 1.5884304222772745
for epoch92, loss is 1.5764173847599152
for epoch93, loss is 1.5956746003840834
for epoch94, loss is 1.5941211526404049
for epoch95, loss is 1.6040080152158502
for epoch96, loss is 1.584372903008172
for epoch97, loss is 1.588408680028449
for epoch98, loss is 1.583926151988619
for epoch99, loss is 1.5815254256849132
acc: 0.377900
```

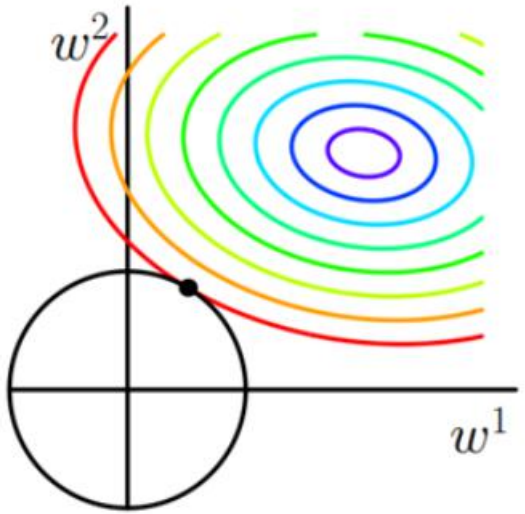
实验结果及改进 -- 正则损失

L1及L2正则化

- L1正则化产生稀疏模型
- L2正则化防止过拟合



$$J = J_0 + \alpha \sum_w |w|$$



$$J = J_0 + \alpha \sum_w w^2$$

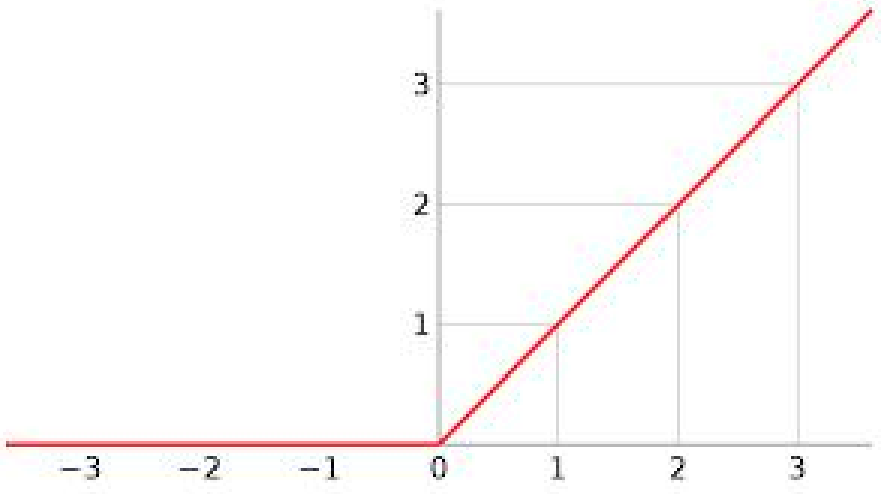
实验结果及改进 -- 激活函数

激活函数

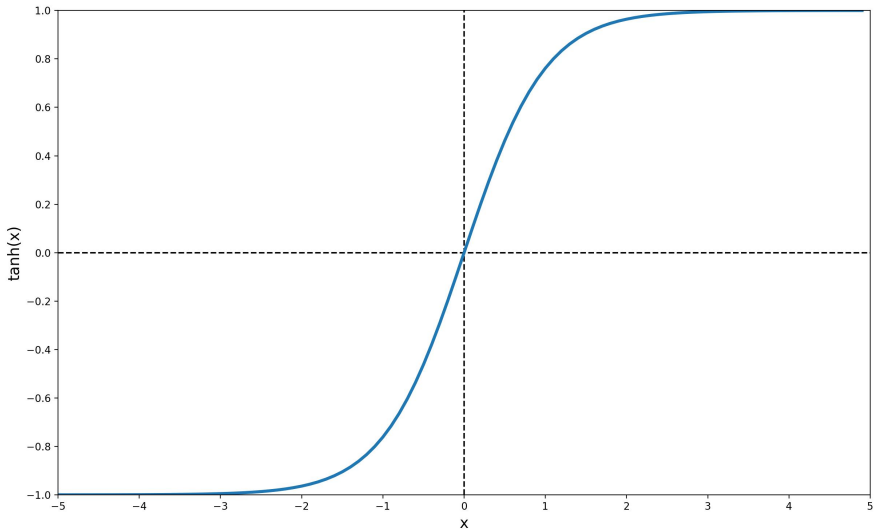
- 引入非线性性

```
act_derivative = lambda y: 1
if self._activation_method == "relu":
    act_derivative = lambda y: (y > 0)
elif self._activation_method == "tanh":
    act_derivative = lambda y: 1 - np.multiply(y, y)
```

relu函数



tanh函数



实验结果及改进

实验结果

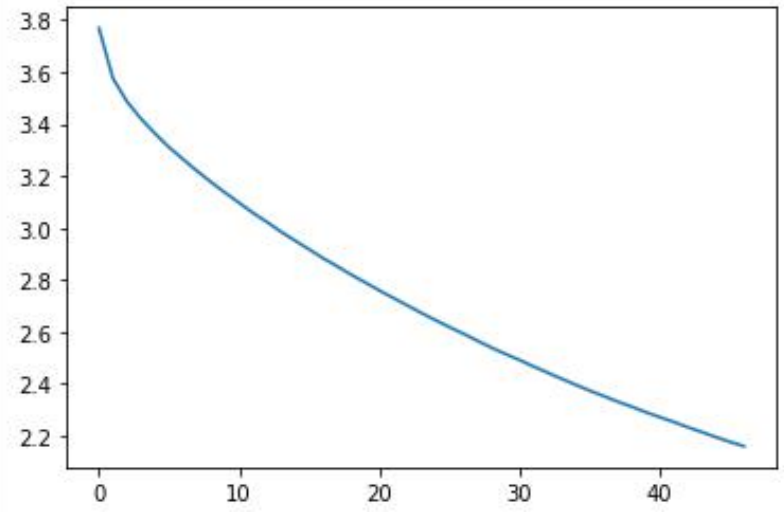
- 无正则：52% (100epoch)
- L1正则：47% (50epoch)

```
norm_method = 0
norm_ratio = 0
if norm_method == 1:
    norm_ratio = 0.0002
elif norm_method == 2:
    norm_ratio = 0.01

batch_size = 256
learning_rate = 0.03
epoch = 100

clsfir = softmax_classifier((3072, 128, 10), norm_ratio, norm_method)
loss = []
```

```
for epoch46, loss is 2.160272401188383, net loss is 1.469655483114419
```



In [6]:
```
result = []
for i in range(10000):
    predict = clsfir.predict(x_test[i])[1]
    result.append(predict)

num_test= 10000
num_correct = np.sum(result == y_test) #计算准确率
accuracy = float(num_correct) / num_test
print("acc: %f" % accuracy)
```

```
acc: 0.473100
```

实验总结

- 简单的三层网络对于Cifar10分类无法达到很理想效果
- 增加网络深度和引入激活函数对于实验效果具有较高的提升
- 数据预处理降低模型效果，推测是处理消除了大量网络能够利用的信息。

Question & Answer