机器学习第二次作业

1、BP 公式推导

BP1: $a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$ $\qquad C = -\sum_i y_i \ln a_i^L$

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_i^L}$$

$$\frac{\partial C}{\partial a_j^L} = \frac{\partial(-\sum_i y_i \ln a_i^L)}{\partial a_j^L} = -\sum_j y_j \frac{1}{a_j}$$

当i=j时

$$\frac{\partial a_j^L}{\partial z_i^L} = \frac{\partial\left(\frac{e^{z_j^L}}{\sum_k e^{z_k^L}}\right)}{\partial z_i^L} = e^{z_i^L}\frac{1}{\sum_k e^{z_k^L}} - \frac{(e^{z_i^L})^2}{(\sum_k e^{z_k^L})^2}$$

$$= \frac{e^{z_i^L}}{\sum_k e^{z_k^L}}\left(1-\frac{e^{z_i^L}}{\sum_k e^{z_k^L}}\right)$$

$$= a_i(1-a_i)$$

当i≠j时

$$\frac{\partial a_j^L}{\partial z_j^L} = -e^{z_j^L}\left(\frac{1}{\sum_k e^{z_k^L}}\right)^2 e^{\frac{1}{z_i^L}} = -a_i a_j$$

$$\therefore \frac{\partial C}{\partial z_i^L} = \left(-\sum_j y_j \frac{1}{a_j}\right)\frac{\partial a_j^L}{\partial z_i^L}$$

$$= -\frac{y_i}{a_i^L}a_i^L(1-a_i^L) + \sum_{i\neq j}\frac{y_j}{a_j^L}a_i^L a_j^L$$

$$= -y_i + y_i a_i^L + \sum_{i\neq j} y_j a_i^L$$

$$= -y_i(1-a_i^L) + a_j^L \sum_{i\neq j} y_i$$

$$= a_j^L - y_i$$

$\therefore$ BP1为 $\delta^L = a^L - y$

BP2: $\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_{i=1}^{n^{l+1}} \frac{\partial C}{\partial z_i^{l+1}} \cdot \frac{\partial z_i^{l+1}}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l}$

$$z_k^{l+1} = \sum_{t=1}^{n^l} w_{kt}^l a_t^l + b_k^l$$
$$= \sum_{t=1}^{n^l} w_{kt}^l \, ReLU(z_t^l) + b_k^l$$

$\cancel{\therefore \delta_j^l = \sum_{j=1}^{n^{l+1}} \delta_i^{l+1}}$

$$\therefore \delta_j^l = \sum_{j=1}^{n^{l+1}} \delta_i^{l+1} W^{l+1}_{[i;:]} \, ReLU(z_j^l)'$$

$$\therefore \delta^l = (W^{l+1})^T \delta^{l+1} \odot ReLU(z^l)'$$

其中 $ReLU(z_i^l)' = \begin{cases} 1 & z_i^l \geq 0 \\ 0 & z_i^l < 0 \end{cases}$

BP3：先推导倒数第2层的代价函数对偏置的导数

$$\frac{\partial C}{\partial b_i^{L-1}} = \frac{\partial C}{\partial z_i^{L}} \frac{\partial z_i^{L}}{\partial b_i^{L-1}}$$

$$z_i^L = \sum_t w_{it}^{L-1} a_t^{L-1} + b_i^{L-1}$$

$$\therefore \frac{\partial C}{\partial b_i^{L-1}} = \delta_i^L * 1 = \delta_i^L$$

$$\therefore \frac{\partial C}{\partial b^{L-1}} = \delta^L$$

再推导一般形式

$$\frac{\partial C}{\partial b_i^{l-1}} = \frac{\partial C}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial b_i^{l-1}} = \delta_i^l * 1 = \delta_i^l$$

$$\therefore \frac{\partial C}{\partial b^{l-1}} = \delta^l$$

因此 $\frac{\partial C}{\partial b^{l-1}} = \delta^l$

BP4：先推导倒数第2层的代价函数对权重的导数

$$\frac{\partial C}{\partial w_{jk}^{L-1}} = \frac{\partial C}{\partial z_j^L} \cdot \frac{\partial z_j^L}{w_{jk}^{L-1}}$$

$$z_j^L = \sum_k w_{jk}^{L-1} a_k^{L-1} + b_k^{L-1}$$

$$\therefore \frac{\partial C}{\partial w_{jk}^{L-1}} = \delta_j^L a_k^{L-1}$$

$$\therefore \frac{\partial C}{\partial w_j^{L-1}} = \left[\frac{\partial C}{\partial w_{j1}^{L-1}}, \frac{\partial C}{\partial w_{j2}^{L-1}}, \cdots \frac{\partial C}{\partial w_{jk}^{L-1}}, \cdots\right]^T$$

$$= [a_1^{L-1}\delta_j^L, a_2^{L-1}\delta_j^L, \cdots a_k^{L-1}\delta_j^L]^T$$

$$= [a_1^{L-1}, a_2^{L-1}, \cdots a_k^{L-1}, \cdots]^T \cdot \delta_j^L = a^{L-1} \cdot \delta_j^L$$

$$\therefore \frac{\partial C}{\partial w^{L-1}} = [a^{L-1}\delta_1^L, a^{L-1}\delta_2^L \cdots a^{L-1}, \delta_k^L \cdots]$$

$$= a^{L-1}[\delta_1^L, \delta_2^L \cdots \delta_k^L]$$

$$= a^{L-1}(\delta^L)^T$$

再推导一般形式

$$\frac{\partial C}{\partial w_{jk}^{l-1}} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^{l-1}}$$

$$z_j^l = \sum_{t=1} w_{jt}^{l-1} a_t^{l-1} + b_t^{l-1}$$

$$\therefore \frac{\partial z_j^l}{\partial w_{jk}^{l-1}} = a_t^{l-1}$$

$$\therefore \frac{\partial C}{\partial w_{jk}^{l-1}} = \delta_j^l a_k^{l-1}$$

由上面同理一步步变为向量得 $\frac{\partial C}{\partial w^{l-1}} = a^{l-1}(\delta^l)^T$

2、编程题

（1）将 sigmoid 函数改为 ReLU 函数

```
def ReLU(z):
    """ReLU 函数"""
    return np.where(z < 0, 0, z)


def ReLU_prime(z):
    """ReLU 函数的导数"""
    return np.where(z < 0, 0, 1)
```

（2）更改损失为交叉熵损失，最后一层用 softmax 函数表示

```
def fn(a, y):
    """交叉熵损失函数"""
    sums = 0
    for zi in a:
        sums += np.exp(zi)
    sums = np.log(sums)

    return np.sum(-y * (a - sums))

def feedforward(self, a):
    """前向传播"""
    for b, w in zip(self.biases[:-1],
self.weights[:-1]):
        a = ReLU(np.dot(w, a) + b)
    a = np.dot(self.weights[-1], a) + self.biases[-
1]
    return a
```

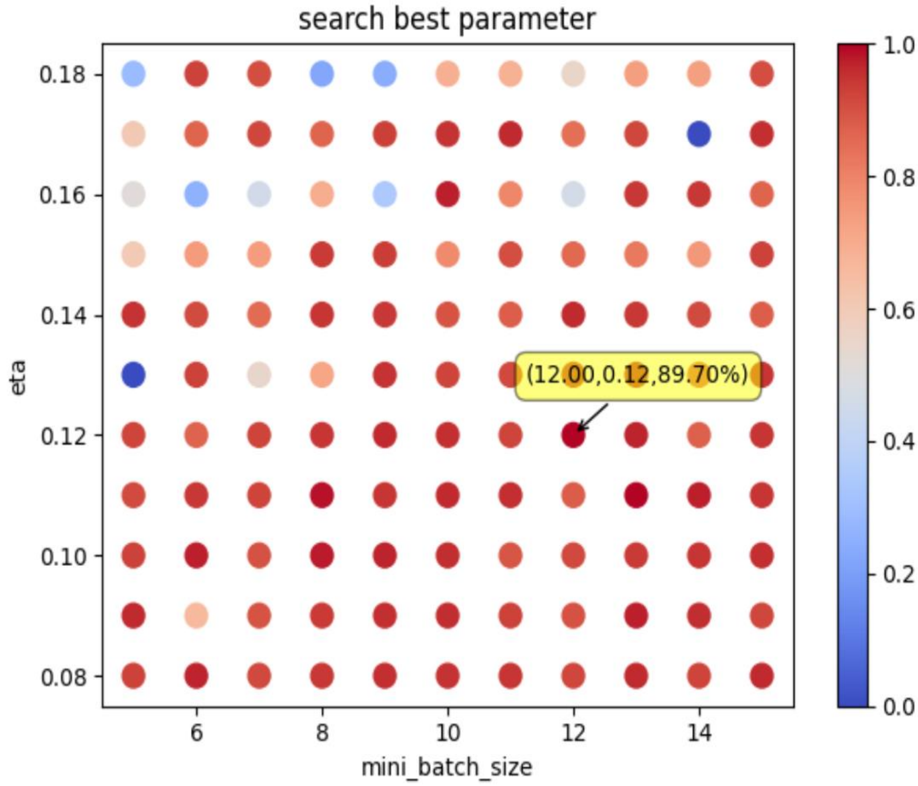
（3）采用网格搜索寻找比较好的超参数

```
# 网格搜索大致好的超参数
x_scatter = np.arange(5, 16, 1)
y_scatter = np.linspace(0.08, 0.18, 11)
results = {}
# x_scatter = np.arange(5, 7, 1)
# y_scatter = np.linspace(0.08, 0.10, 3)
# results = {}
```

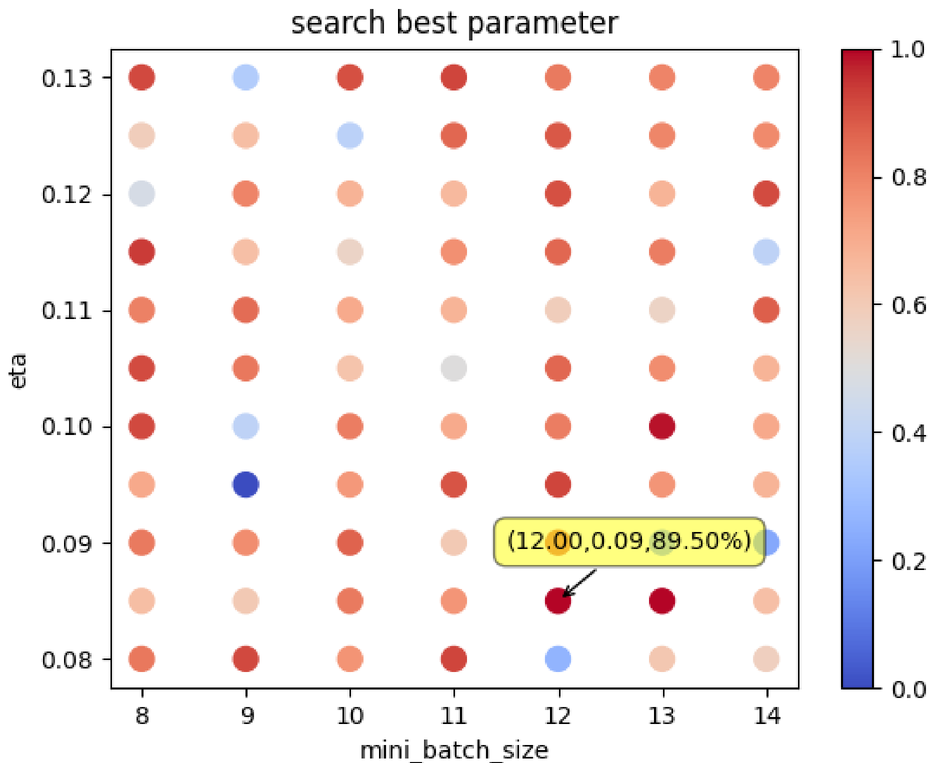
```
for x in x_scatter:
    for y in y_scatter:
        training_data, validation_data, test_data =
load_data.load_data_wrapper()
        training_data = training_data[0:5000]
        test_data = test_data[0:1000]
        net = network.Network([784, 15, 10],
cost=network.CrossEntropyCost)
        res = net.SGD(training_data, 10, x, y,
                      evaluation_data=test_data,
                      monitor_training_accuracy=True,

monitor_evaluation_accuracy=True)
        accuracy = max([i / 1000 for i in res[1]])
        results[(x, y)] = accuracy
marker_size = 100  # default: 20
best_point = max(results, key=results.get)
best_acc = max(results.values())
worst_acc = min(results.values())
colors = [results[x] for x in results.keys()]
colors = np.array(colors)
colors = 1-((best_acc - colors)/(best_acc -
worst_acc))
[Y, X] = np.meshgrid(y_scatter, x_scatter)

X = X.reshape(len(X.reshape(-1, 1)))
Y = Y.reshape(len(Y.reshape(-1, 1)))
plt.figure()
plt.scatter(X, Y, marker_size, c=colors,
cmap=plt.cm.coolwarm)
plt.annotate('(%.2f,%.2f,%.2f%%)' % (best_point[0],
best_point[1], best_acc*100),
             xy=best_point, xytext=(-30, 30),
textcoords='offset pixels',
             bbox=dict(boxstyle='round,pad=0.5',
fc='yellow', alpha=0.5),
             arrowprops=dict(arrowstyle='->',
connectionstyle='arc3,rad=0'))
plt.colorbar()
plt.xlabel('mini_batch_size')
plt.ylabel('eta')
plt.title('search best parameter')
plt.show()
```



继续缩小超参数区间找到最好的大致区间

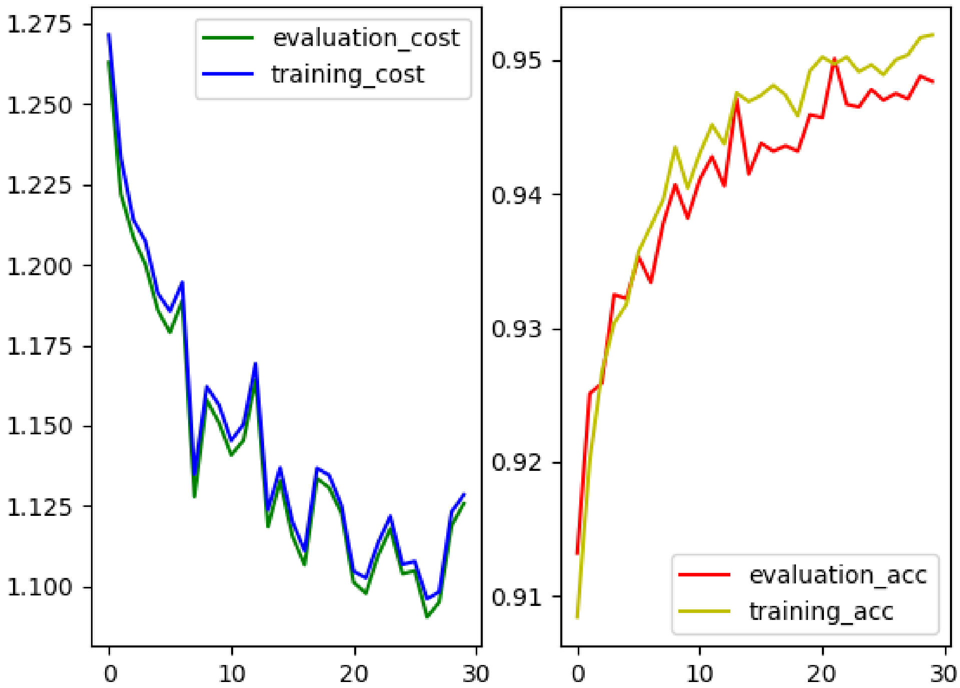


（4）采用隐含层有 15 个神经元的网络，在大致区间进行试验，找到

最好的结果。

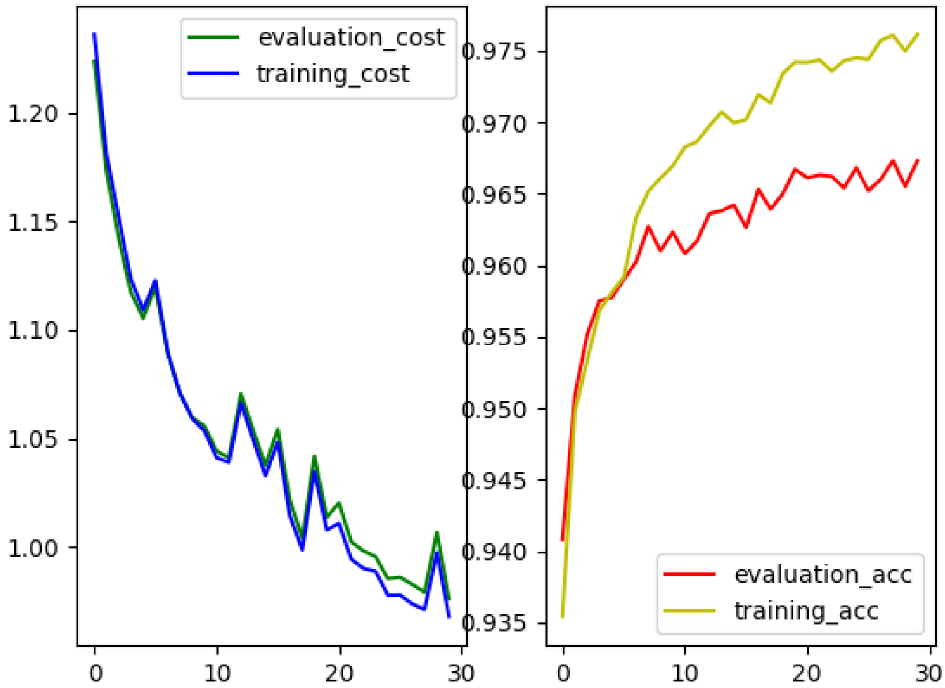
```
training_data, validation_data, test_data =
load_data.load_data_wrapper()
net = network.Network([784, 15, 10],
cost=network.CrossEntropyCost)
y = net.SGD(training_data, 30, 9, 0.1, lambda1=0.3,
          evaluation_data=validation_data,
          monitor_training_cost=True,
          monitor_evaluation_cost=True,
          monitor_training_accuracy=True,
          monitor_evaluation_accuracy=True)
x = np.arange(30)
y0 = [i for i in y[0]]
y1 = [i/10000 for i in y[1]]
y2 = [i for i in y[2]]
y3 = [i/50000 for i in y[3]]
plt.subplot(1, 2, 1)
plt.plot(x, y0, 'g', label='evaluation_cost')
plt.plot(x, y2, 'b', label='training_cost')
plt.title("The change of cost")
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(x, y1, 'r', label='evaluation_acc')
plt.plot(x, y3, 'y', label='training_acc')
plt.title("The change of accuracy")
plt.legend()
```

```
plt.show()
```



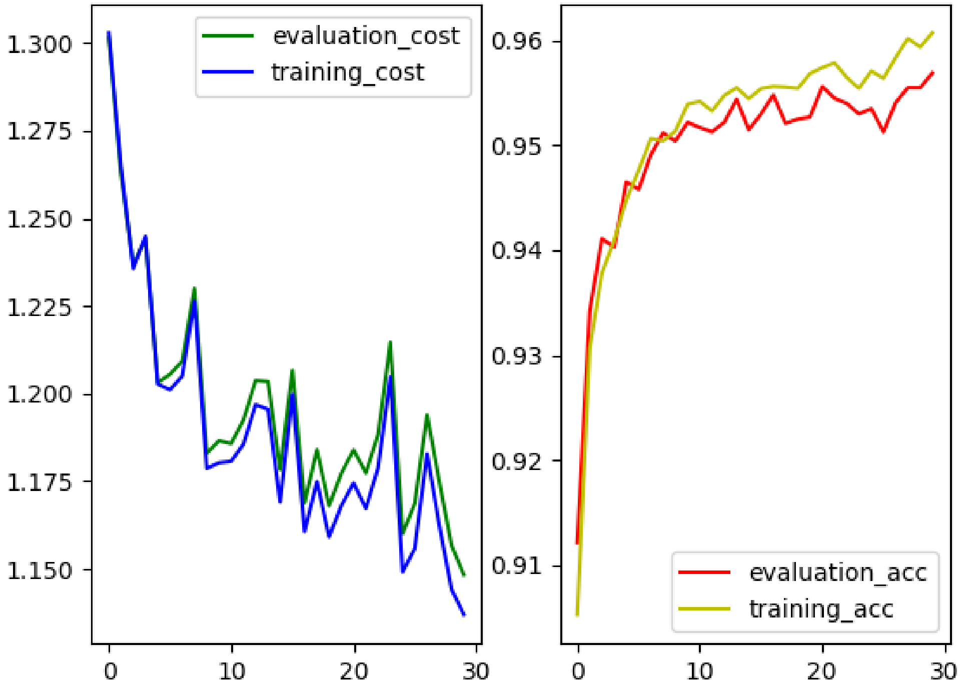
由图可知，损失函数在验证集和训练集上都在下降。并且精度在验证集和训练集上精度都接近 95%，结果不错。

（5）将隐含层改为 30 个神经元



由图可以看出，损失函数得到了进一步的下降，训练集的精度提升到97.5%，但验证集的精度只是提升到了 96.5%，出现了过拟合现象。此时可以通过正则化或者减小中间层的神经元数量来解决

（6）正则化



正则化加上降低隐藏层神经元，过拟合现象降低了。但感觉也不是很好。可能没找到更好的超参数