第四次作业

姓名：樊宇

学号：2021E8018782022

Part 1

Question 1

(a)

输出层：

准则函数为：

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_j (t_j - z_j)^2 \\
&= \frac{1}{2}\sum_j (t_j - f(net_j))^2
\end{aligned}
$$

采用梯度下降，令其对w_{hj}求导：

$$
\frac{\partial J(\boldsymbol{w})}{\partial w_{hj}} = (t_j - z_j) f'(net_j) y_h
$$

因为输出层的激活函数是softmax函数，

$$
\begin{aligned}
f'(net_j) &= \frac{\partial}{\partial net_j}\left(\frac{e^{net_j}}{\sum_m e^{net_m}}\right) \\
&= \frac{e^{net_j}\sum_m e^{net_m} - e^{net_j}e^{net_j}}{(\sum_m e^{net_m})^2} \\
&= f(net_j)(1 - f(net_j))
\end{aligned}
$$

因此，输出层的更新规则为：

$$
w_{hj} = w_{hj} - \eta\delta_j y_h
$$

其中η为迭代步长，$\delta_j = (t_j - z_j) f'(net_j)$

隐含层：

准则函数为：

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_j (t_j - z_j)^2 \\
&= \frac{1}{2}\sum_j (t_j - f(\sum_h w_{hj} g(net_h))^2 \\
&= \frac{1}{2}\sum_j (t_j - f(\sum_h w_{hj} g(\sum_i w_{ih} x_i))^2
\end{aligned}
$$

梯度下降，令其对w_{ih}求导：

$$\begin{aligned}\frac{\partial J(\boldsymbol{w})}{\partial w_{ih}} &= \sum_j (t_j - z_j) f'(net_j) \frac{\partial net_j}{\partial w_{ih}} \\ &= \sum_j (t_j - z_j) f'(net_j) w_{hj} g'(net_h) x_i\end{aligned}$$

隐含层的激活函数为sigmoid函数：

$$\begin{aligned}\frac{\partial g(net_h)}{\partial net_h} &= \frac{\partial}{\partial net_h}\left(\frac{e^{-net_h}}{1+e^{-net_h}}\right) \\ &= \frac{1+e^{-net_h}-1}{(1+e^{-net_h})^2} \\ &= g(net_h)(1-g(net_h))\end{aligned}$$

因此，隐含层的更新规则为：

$$w_{hj} = w_{hj} - \eta \delta_h x_i$$

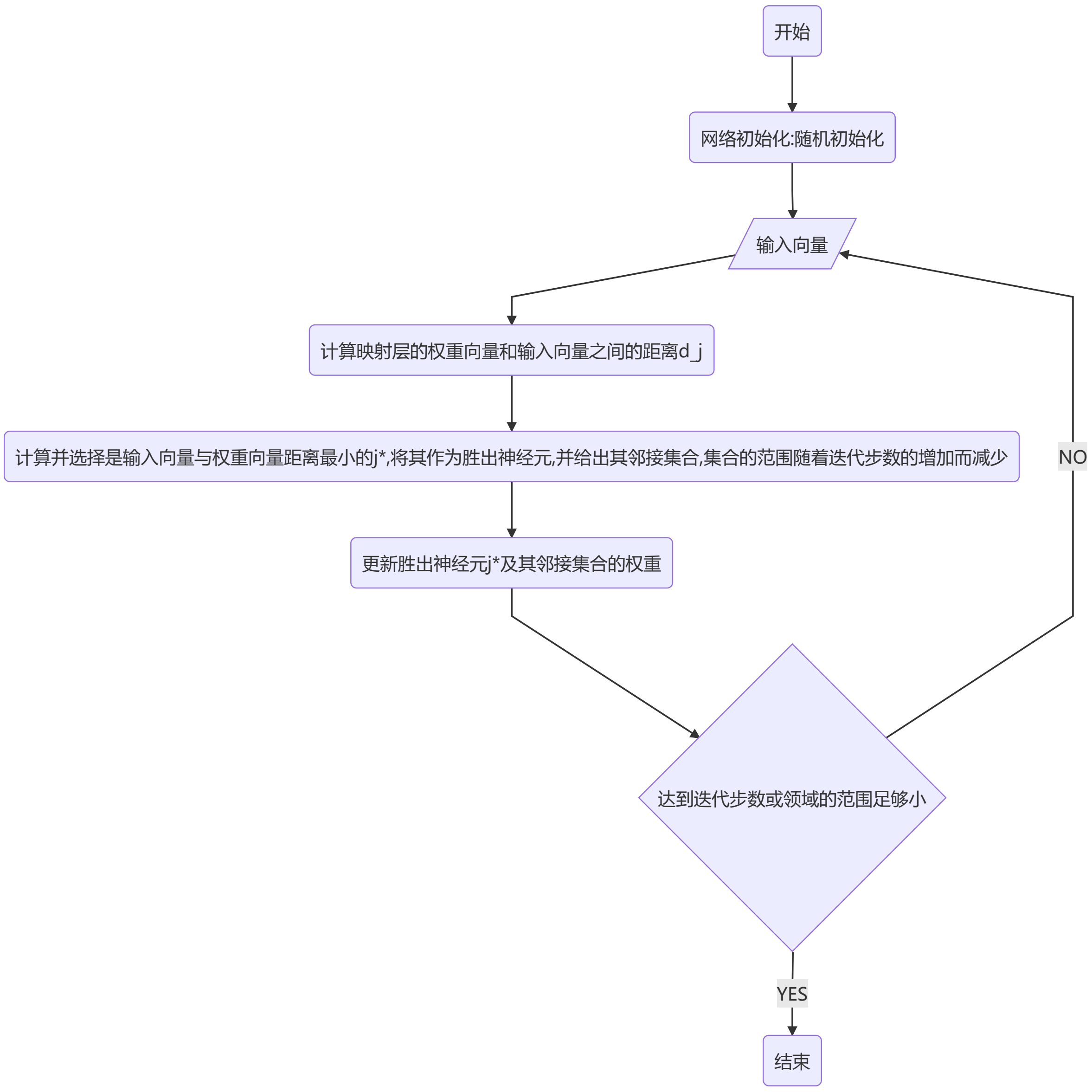
其中η为迭代步长，$\delta_h = g'(net_h) \sum_j w_{hj} \delta_j$

(b)

反向传播算法的核心思想是将输出的误差反向传播给上一层，以此类推，不断向前传播，通过误差计算权重的梯度。

- 首先通过计算损失函数对输出层的权重的导数，然后定义输出层的误差为输出的结果与真实结果之间的差值，并且乘上激活函数导数（因为该导数通常为小于等于1，所以可将其看作放缩因子），于是梯度可定于为误差乘以上一层的输出
- 然后计算损失函数对隐含层的权重的导数，在推导的过程中发现，在该层的误差可以定义为输出层的误差乘以输出层的权重（即隐含层的结点将数据传给谁，就收集谁回传给该结点的误差），再乘以隐含层的激活函数的导数（同样可视为放缩因子），同样梯度可定义为误差乘以上一层的输出

Question 2


开始
网络初始化:随机初始化
输入向量
计算映射层的权重向量和输入向量之间的距离d_j
计算并选择是输入向量与权重向量距离最小的j*,将其作为胜出神经元,并给出其邻接集合,集合的范围随着迭代步数的增加而减少
更新胜出神经元j*及其邻接集合的权重
达到迭代步数或领域的范围足够小
NO
YES
结束

计算步骤:

- S1：网络初始化，通常采用随机初始化
- S2：输入向量
- S3：计算映射层的权重向量与输入向量的距离：

$$d_j = \sqrt{\sum_i (x_i - w_{ij})^2}$$

- S4：选出与权重向量的距离最小的神经元，将其作为胜出神经元$j*$，并给出其临近神经元集合$h(.,j*)$

$$h(j, j*) = exp(-||j - j*||^2/\sigma^2)$$

- S5：调整权重

$$\Delta w_{ij} = \eta h(j, j*)(x_i - w_{ij})$$
$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

- S6：检查是否达到预先设定的要求，否则返回S2

Question 3

(1)

假设输入的图片为灰度图，即只有一个通道

Layer	Input Size	Filter	Output Size	Num of Params
Conv1	400x400x1	20x1x5x5(padding = 2)	400x400x20	500+20(bias unit)
Pool1	400x400x20	2x2	200x200x20	0
Conv2	200x200x20	30x20x3x3(padding = 1)	200x200x30	5400+30(bias unit)
Pool2	200x200x30	2x2	100x100x30	0
Conv3	100x100x30	20x30x3x3(padding = 1)	100x100x20	5400+20(bias unit)
Pool3	100x100x20	2x2	50x50x20	0
Conv4	50x50x20	10x20x3x3(padding = 1)	50x50x10	1800+10(bias unit)
Pool4	50x50x10	2x2	25x25x10	0
FC	6250	\	10	62510

其中滤波器的大小为（滤波器个数x滤波器通道数x滤波器大小）

因此，权值共享和局部连接总参数个数为：

$$520 + 5430 + 5420 + 1810 + 62510 = 75690$$

Layer	Input Size	Output Size	Num of Params
Conv1	400x400x1	400x400x20	5.12e11
Pool1	400x400x20	200x200x20	0
Conv2	200x200x20	200x200x30	9.6e11
Pool2	200x200x30	100x100x30	0
Conv3	100x100x30	100x100x20	0.6e11
Pool3	100x100x20	50x50x20	0
Conv4	50x50x20	50x50x10	6.25e8
Pool4	50x50x10	25x25x10	0
FC	6250	10	62510

因此，全连接和非权值共享的总参数个数约为：$1.53e12$

因此，减少的权重数量约为：$1.53e12$

(2)

最大池化要满足梯度之和不变的原则

最大池化在前向传播时仅将patch中最大的元素传播给下一层，因此在反向传播时也仅将梯度传播给前一层的最大的元素（因为池化层没有参数，所以将梯度直接传播给前一层），其他的像素不接受梯度，为0。

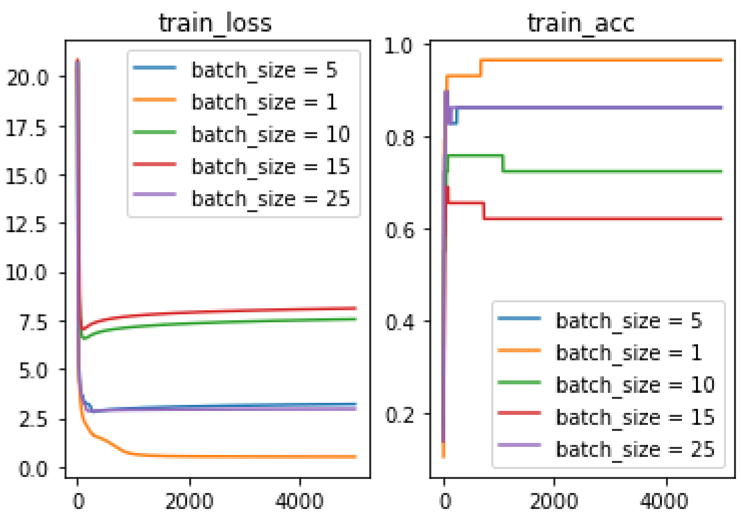
(3)

- 改变滤波器的大小
- 改变滤波器的个数
- 改变滤波器的步长
- 改变池化窗口的大小
- 改变池化窗口的步长
- 增加卷积层的数目

Part 2

1

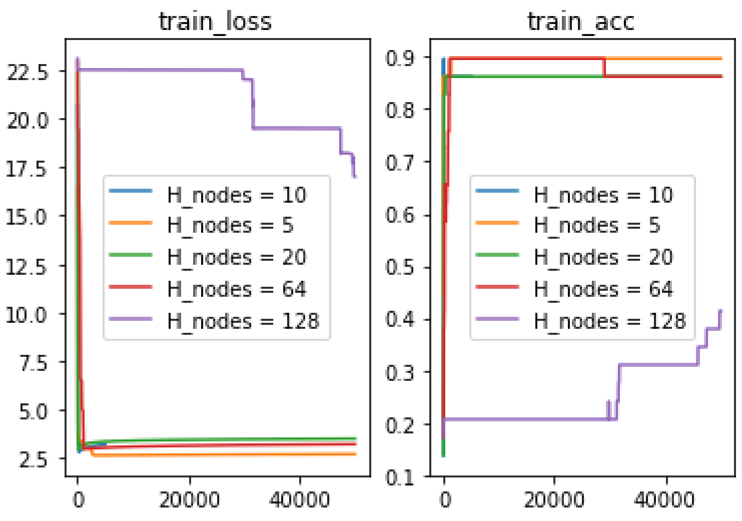
batch_size = 1 时即为单样本方式更新权重



由上图可知采用单样本方式更新权重时，在训练样本上取得的效果最好，但也很容易造成过拟合现象，batch_size过大时（接近训练样本总数时），总体上类似于单样本更新，但由于不等于训练样本，会出现浪费数据资源的现象

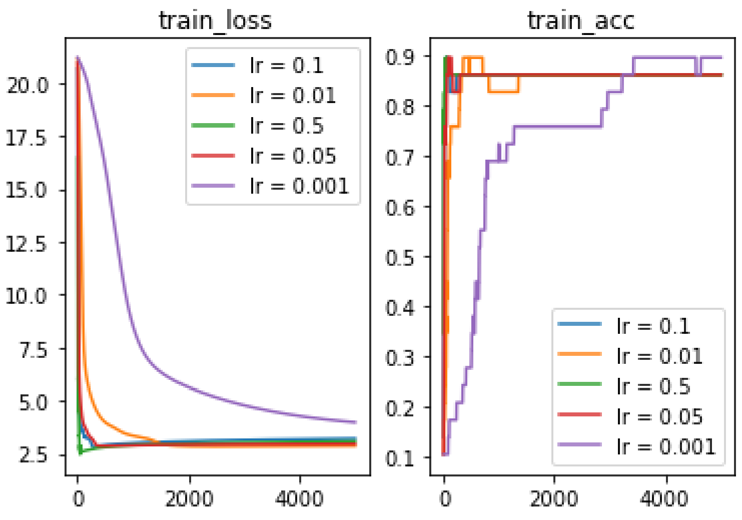
2

(a)



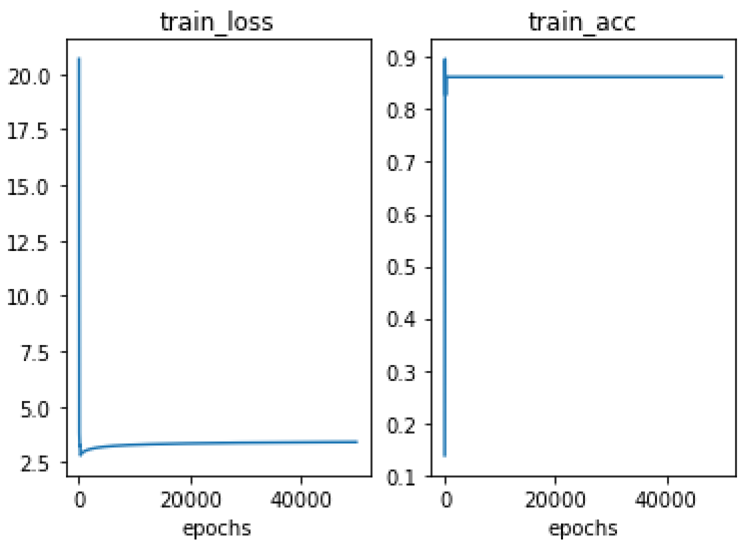
隐藏结点相对于输入结点过多时，收敛得很慢

(b)



由上图可知，步长越小，收敛速度越慢，最终在训练集上的效果越好，但也有可能存在过拟合的情况

(c)



由上图可知，随着训练轮数的增加，在训练集上的表现越好，达到一定轮数后，训练效果保持不变

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv('./data.csv')
```

```
data = data.values
```

```
train_data = data[:, :-1]
```

```
label_1 = np.array([1, 0, 0])
label_2 = np.array([0, 1, 0])
label_3 = np.array([0, 0, 1])
train_labels = np.zeros(shape = (data.shape[0], 3))
for i in range(data.shape[0]):
    train_labels[i, :] = eval('label_' + str(int(data[i, -1])))
```

```
# 线性层
class Linear:
    def __init__(self, in_shape, out_shape):
        np.random.seed(7)
        self.W = np.random.rand(in_shape, out_shape)
        self.b = np.random.rand(1, out_shape)
        self.W_grad = np.zeros((in_shape, out_shape))
        self.b_grad = np.zeros((1, out_shape))

    def forward(self, X):
        return np.matmul(X, self.W) + self.b

    def backward(self, X, grad):
        self.W_grad = np.matmul(X.T, grad) # k个样本
        self.b_grad = np.matmul(grad.T, np.ones(X.shape[0]))
        return np.matmul(grad, self.W.T)

    def update(self, lr):
        self.W = self.W + self.W_grad * lr
        self.b = self.b + self.b_grad * lr
```

```
class Sigmoid:
    def __init__(self):
        pass

    def forward(self, X):
        return 1 / (1 + np.exp(-X))

    def backward(self, X, grad):
        return self.forward(X) * ( 1 - self.forward(X)) * grad
```

```
class Tanh:
    def __init__(self):
        pass

    def forward(self, X):
        return (np.exp(X) - np.exp(-X)) / (np.exp(X) + np.exp(-X))

    def backward(self, X, grad):
        return ( 1 - np.power(self.forward(X),2)) * grad
```

```
class BP:
    def __init__(self, size1, size2, lr):
        self.size1 = size1
        self.size2 = size2
        self.lr = lr
        pass

    def model_construct(self):
        self.Linear1 = Linear(self.size1[0], self.size1[1])
        self.Tanh1 = Tanh()
        self.Linear2 = Linear(self.size2[0], self.size2[1])
        self.Sigmoid1 = Sigmoid()

    def MSEloss(self, X, Y):
        return np.sum(np.power(self.predict(X) - Y, 2) / 2)

    def acc(self, X, Y):
        count = (np.sum(np.argmax(Y, axis = 1) == np.argmax(self.predict(X),
axis = 1)))
        return count / X.shape[0]

    def predict(self, X):
        o0 = X
        a1 = self.Linear1.forward(o0)
        o1 = self.Tanh1.forward(a1)
        a2 = self.Linear2.forward(o1)
        o2  = self.Sigmoid1.forward(a2)
        return o2

    def update(self, X, Y):

        o0 = X
        a1 = self.Linear1.forward(o0)
        o1 = self.Tanh1.forward(a1)
        a2 = self.Linear2.forward(o1)
        o2  = self.Sigmoid1.forward(a2)

        # 逐层反向传播梯度
        grad = Y - o2
        grad = self.Sigmoid1.backward(a2, grad)
        grad = self.Linear2.backward(o1, grad)
        grad = self.Tanh1.backward(a1, grad)
        grad = self.Linear1.backward(o0, grad)

        # 更新参数
        self.Linear1.update(self.lr)
        self.Linear2.update(self.lr)

    def train(self, X_train, Y_train, epoch, batch_size):
        train_loss = []
        train_acc = []

        for i in range(epoch):
            for j in range(X_train.shape[0] // batch_size):
                self.update(X_train[j * batch_size : (j + 1) * batch_size, :],
Y_train[j * batch_size : (j + 1) * batch_size, :])

            loss = self.MSEloss(X_train, Y_train)
            acc = self.acc(X_train, Y_train)

            #print('epoch = ', i)
            #print('loss = {}, acc = {}'.format(loss, acc))
            train_loss.append(loss)
            train_acc.append(acc)

            if loss <= 0.01:
                break

        return train_loss, train_acc
```

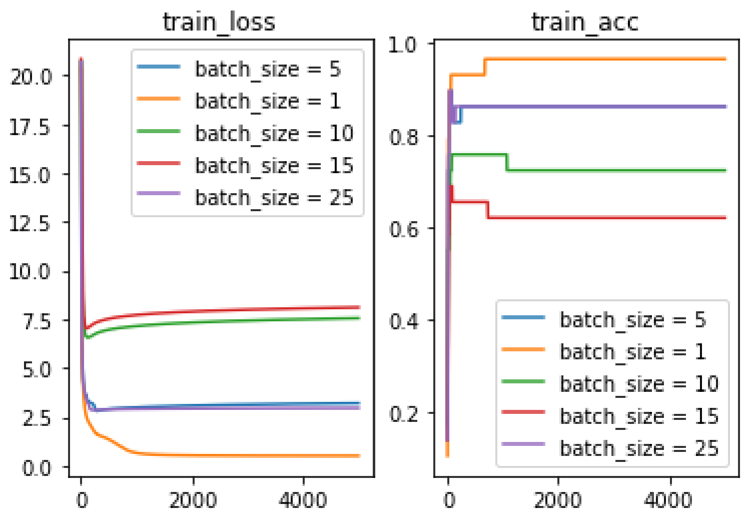
```
num_output = 3
num_hidden_nodes = 10
lr = 0.1
epoch = 5000
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_1, train_acc_1] = model.train(train_data, train_labels, epoch,
batch_size)
```

```
# 批处理大小对训练数据的影响， batch_size = 1 时即为单样本更新权重
num_output = 3
num_hidden_nodes = 10
lr = 0.1
batch_size = 1

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_2, train_acc_2] = model.train(train_data, train_labels, epoch,
batch_size)

## ====================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.1
batch_size = 10

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
```

```python
model.model_construct()
[train_loss_3, train_acc_3] = model.train(train_data, train_labels, epoch,
batch_size)

## ==================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.1
batch_size = 15

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_4, train_acc_4] = model.train(train_data, train_labels, epoch,
batch_size)

## ==================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.1
batch_size = 25

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_5, train_acc_5] = model.train(train_data, train_labels, epoch,
batch_size)
```

```python
plt.figure()

plt.subplot(1, 2, 1)
plt.plot(range(len(train_loss_1)), train_loss_1, label = 'batch_size = 5')
plt.plot(range(len(train_loss_2)), train_loss_2, label = 'batch_size = 1')
plt.plot(range(len(train_loss_3)), train_loss_3, label = 'batch_size = 10')
plt.plot(range(len(train_loss_4)), train_loss_4, label = 'batch_size = 15')
plt.plot(range(len(train_loss_5)), train_loss_5, label = 'batch_size = 25')
plt.legend()
plt.title('train_loss')

plt.subplot(1, 2, 2)
plt.plot(range(len(train_acc_1)), train_acc_1, label = 'batch_size = 5')
plt.plot(range(len(train_acc_2)), train_acc_2, label = 'batch_size = 1')
plt.plot(range(len(train_acc_3)), train_acc_3, label = 'batch_size = 10')
plt.plot(range(len(train_acc_4)), train_acc_4, label = 'batch_size = 15')
plt.plot(range(len(train_acc_5)), train_acc_5, label = 'batch_size = 25')
plt.legend()
plt.title('train_acc')
```

```
Text(0.5, 1.0, 'train_acc')
```

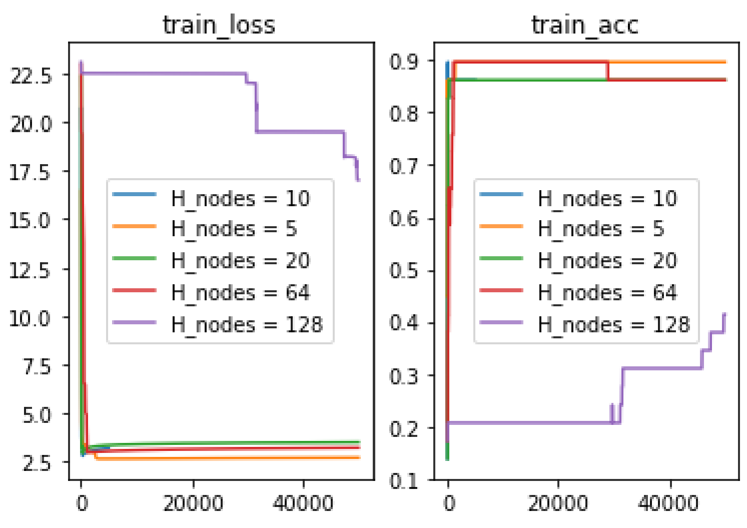


```
# 隐含层不同节点数目对训练精度的影响
num_output = 3
num_hidden_nodes = 5
lr = 0.1
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_6, train_acc_6] = model.train(train_data, train_labels, epoch,
batch_size)

## ==================================================================
num_output = 3
num_hidden_nodes = 20
lr = 0.1
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_7, train_acc_7] = model.train(train_data, train_labels, epoch,
batch_size)

## ==================================================================
num_output = 3
num_hidden_nodes = 64
lr = 0.1
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_8, train_acc_8] = model.train(train_data, train_labels, epoch,
batch_size)

## ==================================================================
num_output = 3
```

```
num_hidden_nodes = 128
lr = 0.1
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_9, train_acc_9] = model.train(train_data, train_labels, epoch,
batch_size)
```

```
plt.figure()

plt.subplot(1, 2, 1)
plt.plot(range(len(train_loss_1)), train_loss_1, label = 'H_nodes = 10')
plt.plot(range(len(train_loss_6)), train_loss_6, label = 'H_nodes = 5')
plt.plot(range(len(train_loss_7)), train_loss_7, label = 'H_nodes = 20')
plt.plot(range(len(train_loss_8)), train_loss_8, label = 'H_nodes = 64')
plt.plot(range(len(train_loss_9)), train_loss_9, label = 'H_nodes = 128')
plt.legend()
plt.title('train_loss')

plt.subplot(1, 2, 2)
plt.plot(range(len(train_acc_1)), train_acc_1, label = 'H_nodes = 10')
plt.plot(range(len(train_acc_6)), train_acc_6, label = 'H_nodes = 5')
plt.plot(range(len(train_acc_7)), train_acc_7, label = 'H_nodes = 20')
plt.plot(range(len(train_acc_8)), train_acc_8, label = 'H_nodes = 64')
plt.plot(range(len(train_acc_9)), train_acc_9, label = 'H_nodes = 128')
plt.legend()
plt.title('train_acc')
```

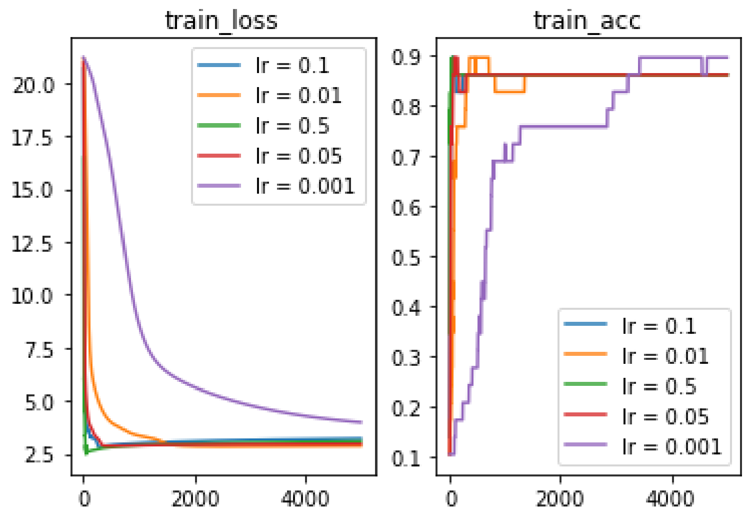
```
Text(0.5, 1.0, 'train_acc')
```



```
# 批处理大小对训练数据的影响， batch_size = 1 时即为单样本更新权重
num_output = 3
num_hidden_nodes = 10
lr = 0.01
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_10, train_acc_10] = model.train(train_data, train_labels, epoch,
batch_size)

## ===================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.5
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_11, train_acc_11] = model.train(train_data, train_labels, epoch,
batch_size)

## ===================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.05
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_12, train_acc_12] = model.train(train_data, train_labels, epoch,
batch_size)

## ===================================================================
num_output = 3
num_hidden_nodes = 10
lr = 0.001
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_13, train_acc_13] = model.train(train_data, train_labels, epoch,
batch_size)
```

```
plt.figure()

plt.subplot(1, 2, 1)
plt.plot(range(len(train_loss_1)), train_loss_1, label = 'lr = 0.1')
plt.plot(range(len(train_loss_10)), train_loss_10, label = 'lr = 0.01')
plt.plot(range(len(train_loss_11)), train_loss_11, label = 'lr = 0.5')
plt.plot(range(len(train_loss_12)), train_loss_12, label = 'lr = 0.05')
plt.plot(range(len(train_loss_13)), train_loss_13, label = 'lr = 0.001')
plt.legend()
plt.title('train_loss')

plt.subplot(1, 2, 2)
plt.plot(range(len(train_acc_1)), train_acc_1, label = 'lr = 0.1')
plt.plot(range(len(train_acc_10)), train_acc_10, label = 'lr = 0.01')
plt.plot(range(len(train_acc_11)), train_acc_11, label = 'lr = 0.5')
plt.plot(range(len(train_acc_12)), train_acc_12, label = 'lr = 0.05')
plt.plot(range(len(train_acc_13)), train_acc_13, label = 'lr = 0.001')
plt.legend()
plt.title('train_acc')
```

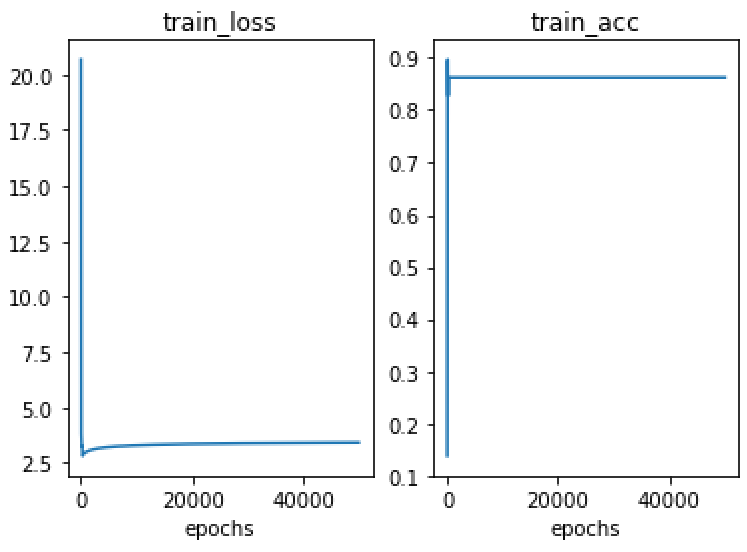
```
Text(0.5, 1.0, 'train_acc')
```



```
num_output = 3
num_hidden_nodes = 10
lr = 0.1
epoch = 50000
batch_size = 5

model = BP(size1 = [train_data.shape[-1], num_hidden_nodes], size2 =
[num_hidden_nodes, num_output], lr = lr)
model.model_construct()
[train_loss_14, train_acc_14] = model.train(train_data, train_labels, epoch,
batch_size)
```

```
plt.figure()

plt.subplot(1, 2, 1)
plt.plot(range(len(train_loss_14)), train_loss_14)
plt.title('train_loss')
plt.xlabel('epochs')

plt.subplot(1, 2, 2)
plt.plot(range(len(train_acc_14)), train_acc_14)
plt.title('train_acc')
plt.xlabel('epochs')
```

```
Text(0.5, 0, 'epochs')
```



```

```