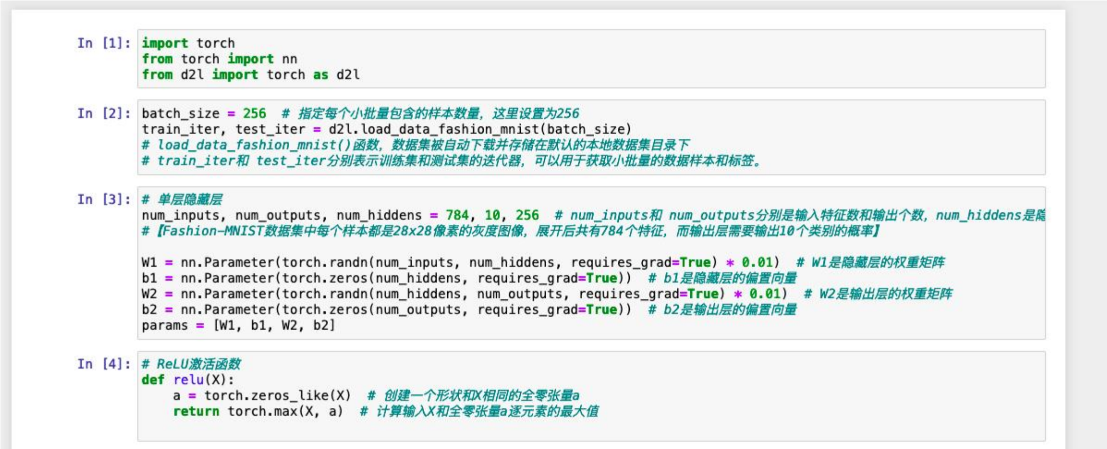


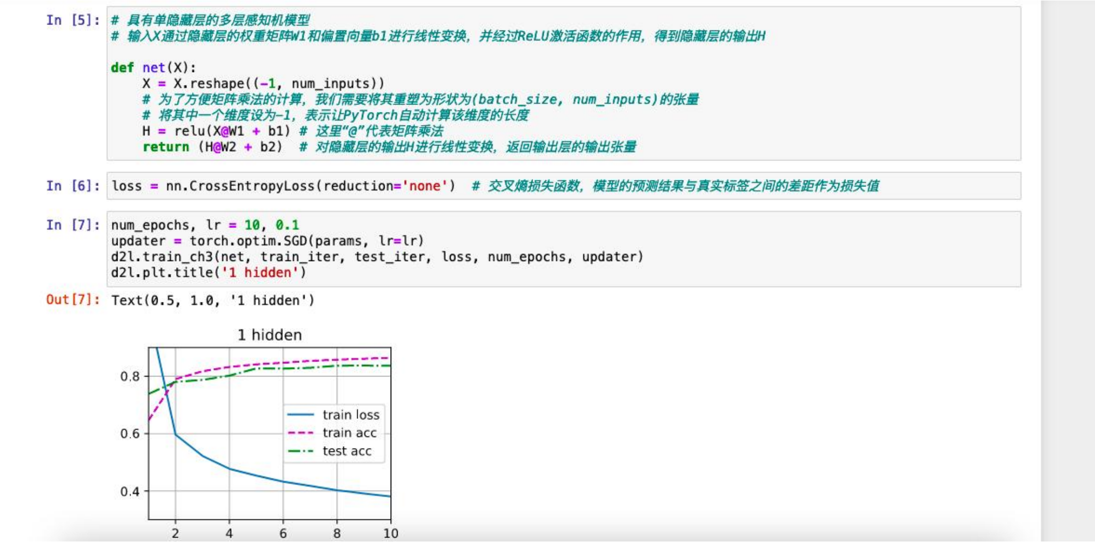
```
In [1]: import torch
        from torch import nn
        from d2l import torch as d2l

In [2]: batch_size = 256  # 指定每个小批量包含的样本数量，这里设置为256
        train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
        # load_data_fashion_mnist()函数，数据集被自动下载并存储在默认的本地数据集目录下
        # train_iter和 test_iter分别表示训练集和测试集的迭代器，可以用于获取小批量的数据样本和标签。

In [3]: # 单层隐藏层
        num_inputs, num_outputs, num_hiddens = 784, 10, 256  # num_inputs和 num_outputs分别是输入特征数和输出个数，num_hiddens是隐
        #【Fashion-MNIST数据集中每个样本都是28x28像素的灰度图像，展开后共有784个特征，而输出层需要输出10个类别的概率】

        W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)  # W1是隐藏层的权重矩阵
        b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))  # b1是隐藏层的偏置向量
        W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)  # W2是输出层的权重矩阵
        b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))  # b2是输出层的偏置向量
        params = [W1, b1, W2, b2]

In [4]: # ReLU激活函数
        def relu(X):
            a = torch.zeros_like(X)  # 创建一个形状和X相同的全零张量a
            return torch.max(X, a)  # 计算输入X和全零张量a逐元素的最大值
```



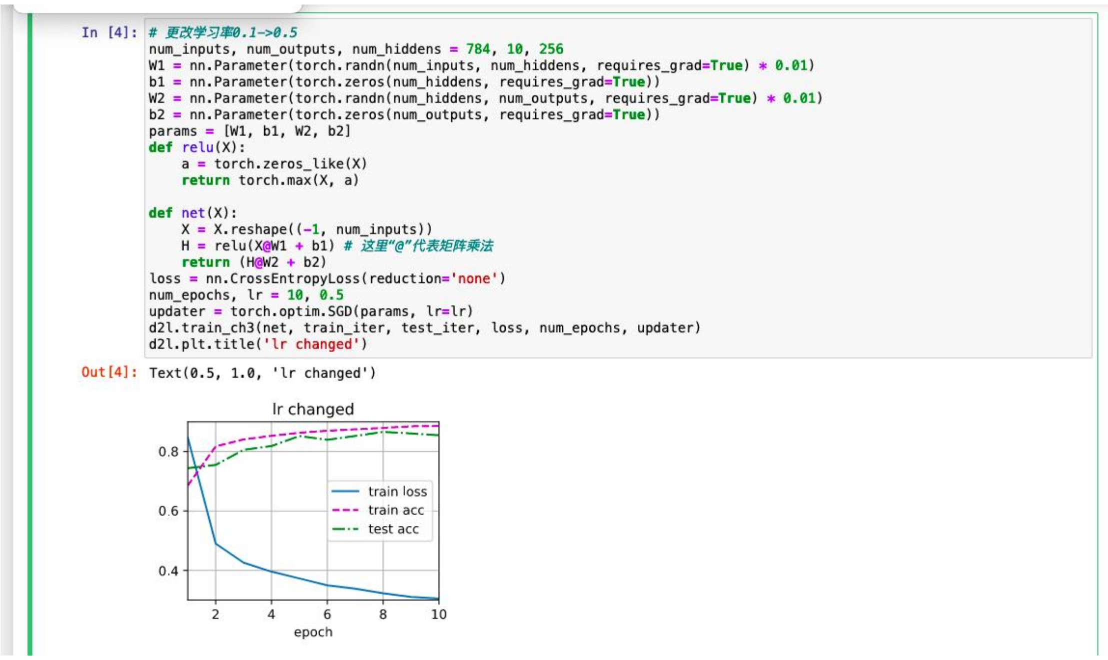
```
In [5]: # 具有单隐藏层的多层感知机模型
        # 输入X通过隐藏层的权重矩阵W1和偏置向量b1进行线性变换，并经过ReLU激活函数的作用，得到隐藏层的输出H

        def net(X):
            X = X.reshape((-1, num_inputs))
            # 为了方便矩阵乘法的计算，我们需要将其重塑为形状为(batch_size, num_inputs)的张量
            # 将其中一个维度设为-1，表示让PyTorch自动计算该维度的长度
            H = relu(X@W1 + b1) # 这里“@”代表矩阵乘法
            return (H@W2 + b2)  # 对隐藏层的输出H进行线性变换，返回输出层的输出张量

In [6]: loss = nn.CrossEntropyLoss(reduction='none')  # 交叉熵损失函数，模型的预测结果与真实标签之间的差距作为损失值

In [7]: num_epochs, lr = 10, 0.1
        updater = torch.optim.SGD(params, lr=lr)
        d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
        d2l.plt.title('1 hidden')

Out[7]: Text(0.5, 1.0, '1 hidden')
```



```
In [4]: # 更改学习率0.1->0.5
        num_inputs, num_outputs, num_hiddens = 784, 10, 256
        W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)
        b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))
        W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)
        b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
        params = [W1, b1, W2, b2]
        def relu(X):
            a = torch.zeros_like(X)
            return torch.max(X, a)

        def net(X):
            X = X.reshape((-1, num_inputs))
            H = relu(X@W1 + b1) # 这里“@”代表矩阵乘法
            return (H@W2 + b2)
        loss = nn.CrossEntropyLoss(reduction='none')
        num_epochs, lr = 10, 0.5
        updater = torch.optim.SGD(params, lr=lr)
        d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
        d2l.plt.title('lr changed')

Out[4]: Text(0.5, 1.0, 'lr changed')
```

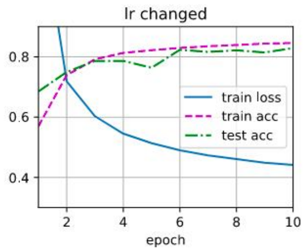
In [5]:

```
# 更改学习率0.1->0.05
num_inputs, num_outputs, num_hiddens = 784, 10, 256
W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)
b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))
W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)
b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
params = [W1, b1, W2, b2]
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)

def net(X):
    X = X.reshape((-1, num_inputs))
    H = relu(X@W1 + b1) # 这里“@”代表矩阵乘法
    return (H@W2 + b2)
loss = nn.CrossEntropyLoss(reduction='none')
num_epochs, lr = 10, 0.05
updater = torch.optim.SGD(params, lr=lr)
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
d2l.plt.title('lr changed')
```

Out[5]:

```
Text(0.5, 1.0, 'lr changed')
```



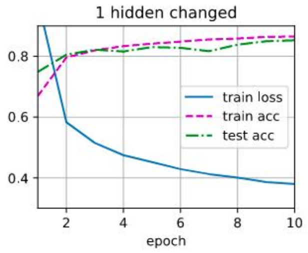
In [7]:

```
# 更改单层隐藏层中的隐藏单元数256->256*2
num_inputs, num_outputs, num_hiddens = 784, 10, 256*2
W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)
b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))
W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)
b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
params = [W1, b1, W2, b2]
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)

def net(X):
    X = X.reshape((-1, num_inputs))
    H = relu(X@W1 + b1) # 这里“@”代表矩阵乘法
    return (H@W2 + b2)
loss = nn.CrossEntropyLoss(reduction='none')
num_epochs, lr = 10, 0.1
updater = torch.optim.SGD(params, lr=lr)
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
d2l.plt.title('1 hidden changed')
```

Out[7]:

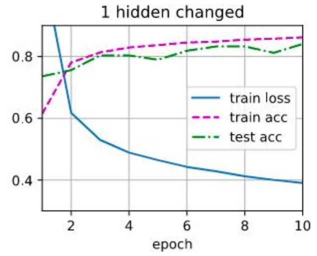
```
Text(0.5, 1.0, '1 hidden changed')
```



In [8]:
```
# 更改单层隐藏层中的隐藏单元数256->128
num_inputs, num_outputs, num_hiddens = 784, 10, 128
W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens, requires_grad=True) * 0.01)
b1 = nn.Parameter(torch.zeros(num_hiddens, requires_grad=True))
W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs, requires_grad=True) * 0.01)
b2 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
params = [W1, b1, W2, b2]
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)

def net(X):
    X = X.reshape((-1, num_inputs))
    H = relu(X@W1 + b1) # 这里“@”代表矩阵乘法
    return (H@W2 + b2)
loss = nn.CrossEntropyLoss(reduction='none')
num_epochs, lr = 10, 0.1
updater = torch.optim.SGD(params, lr=lr)
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
d2l.plt.title('1 hidden changed')
```

Out[8]:
```
Text(0.5, 1.0, '1 hidden changed')
```

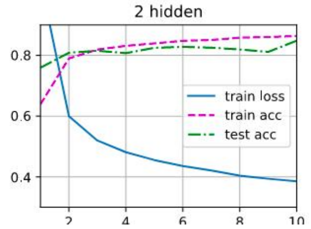


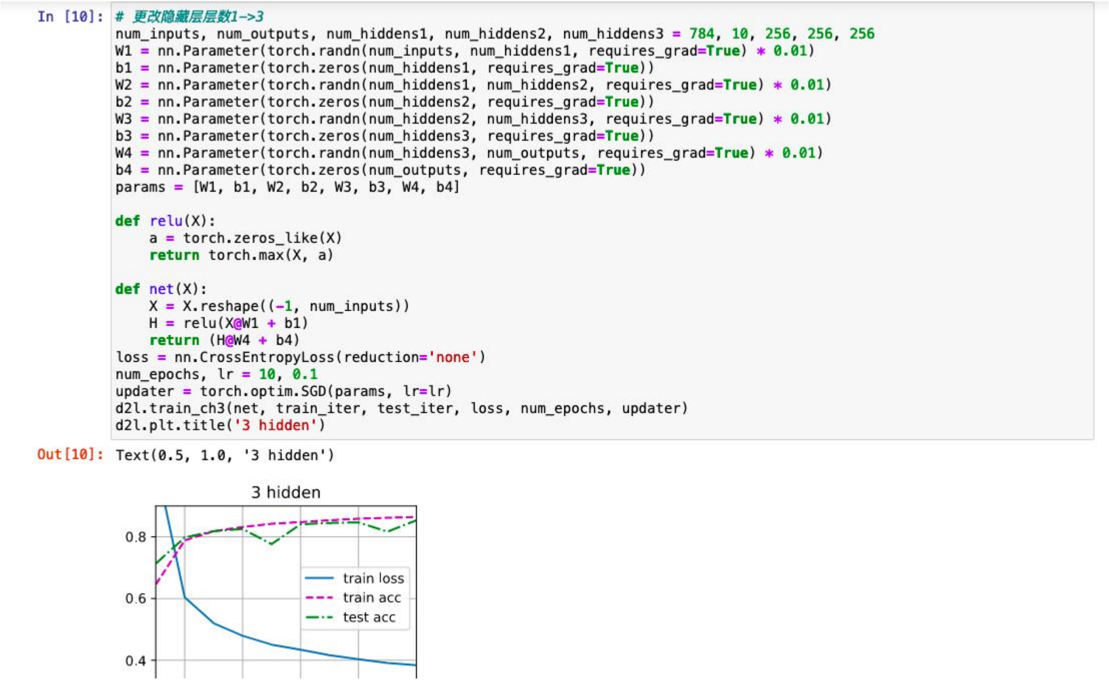
In [9]:
```
# 更改隐藏层层数1->2
num_inputs, num_outputs, num_hiddens1, num_hiddens2 = 784, 10, 256, 256
W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens1, requires_grad=True) * 0.01)
b1 = nn.Parameter(torch.zeros(num_hiddens1, requires_grad=True))
W2 = nn.Parameter(torch.randn(num_hiddens1, num_hiddens2, requires_grad=True) * 0.01)
b2 = nn.Parameter(torch.zeros(num_hiddens2, requires_grad=True))
W3 = nn.Parameter(torch.randn(num_hiddens2, num_outputs, requires_grad=True) * 0.01)
b3 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
params = [W1, b1, W2, b2, W3, b3]

def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)

def net(X):
    X = X.reshape((-1, num_inputs))
    H = relu(X@W1 + b1)
    return (H@W3 + b3)
loss = nn.CrossEntropyLoss(reduction='none')
num_epochs, lr = 10, 0.1
updater = torch.optim.SGD(params, lr=lr)
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
d2l.plt.title('2 hidden')
```

Out[9]:
```
Text(0.5, 1.0, '2 hidden')
```



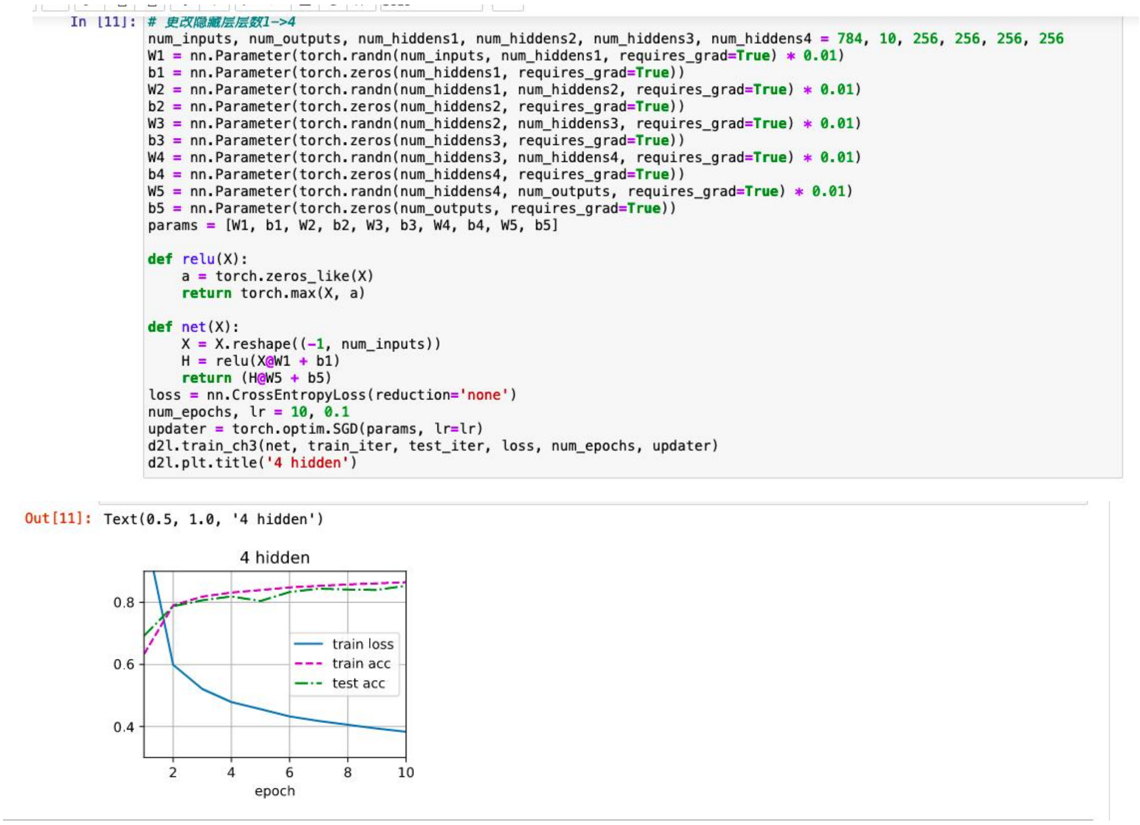


```
In [10]: # 更改隐藏层层数1->3
         num_inputs, num_outputs, num_hiddens1, num_hiddens2, num_hiddens3 = 784, 10, 256, 256, 256
         W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens1, requires_grad=True) * 0.01)
         b1 = nn.Parameter(torch.zeros(num_hiddens1, requires_grad=True))
         W2 = nn.Parameter(torch.randn(num_hiddens1, num_hiddens2, requires_grad=True) * 0.01)
         b2 = nn.Parameter(torch.zeros(num_hiddens2, requires_grad=True))
         W3 = nn.Parameter(torch.randn(num_hiddens2, num_hiddens3, requires_grad=True) * 0.01)
         b3 = nn.Parameter(torch.zeros(num_hiddens3, requires_grad=True))
         W4 = nn.Parameter(torch.randn(num_hiddens3, num_outputs, requires_grad=True) * 0.01)
         b4 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
         params = [W1, b1, W2, b2, W3, b3, W4, b4]

         def relu(X):
             a = torch.zeros_like(X)
             return torch.max(X, a)

         def net(X):
             X = X.reshape((-1, num_inputs))
             H = relu(X@W1 + b1)
             return (H@W4 + b4)
         loss = nn.CrossEntropyLoss(reduction='none')
         num_epochs, lr = 10, 0.1
         updater = torch.optim.SGD(params, lr=lr)
         d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
         d2l.plt.title('3 hidden')

Out[10]: Text(0.5, 1.0, '3 hidden')
```



```
In [11]: # 更改隐藏层层数1->4
         num_inputs, num_outputs, num_hiddens1, num_hiddens2, num_hiddens3, num_hiddens4 = 784, 10, 256, 256, 256, 256
         W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens1, requires_grad=True) * 0.01)
         b1 = nn.Parameter(torch.zeros(num_hiddens1, requires_grad=True))
         W2 = nn.Parameter(torch.randn(num_hiddens1, num_hiddens2, requires_grad=True) * 0.01)
         b2 = nn.Parameter(torch.zeros(num_hiddens2, requires_grad=True))
         W3 = nn.Parameter(torch.randn(num_hiddens2, num_hiddens3, requires_grad=True) * 0.01)
         b3 = nn.Parameter(torch.zeros(num_hiddens3, requires_grad=True))
         W4 = nn.Parameter(torch.randn(num_hiddens3, num_hiddens4, requires_grad=True) * 0.01)
         b4 = nn.Parameter(torch.zeros(num_hiddens4, requires_grad=True))
         W5 = nn.Parameter(torch.randn(num_hiddens4, num_outputs, requires_grad=True) * 0.01)
         b5 = nn.Parameter(torch.zeros(num_outputs, requires_grad=True))
         params = [W1, b1, W2, b2, W3, b3, W4, b4, W5, b5]

         def relu(X):
             a = torch.zeros_like(X)
             return torch.max(X, a)

         def net(X):
             X = X.reshape((-1, num_inputs))
             H = relu(X@W1 + b1)
             return (H@W5 + b5)
         loss = nn.CrossEntropyLoss(reduction='none')
         num_epochs, lr = 10, 0.1
         updater = torch.optim.SGD(params, lr=lr)
         d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, updater)
         d2l.plt.title('4 hidden')

Out[11]: Text(0.5, 1.0, '4 hidden')
```