专业英语：神经网络实验报告

课程名称：专业英语　年级：2018　指导老师：周烜
姓名：孙秋实　项目名称：手动搭建神经网络　学号：10185501402
项目地址：https://github.com/QiushiSun/Professional-English-Neural-Network-by-Hand

This is a bilingual Lab report for course: Professional English, formatted by LaTeX

Part 1
实验目标 (Targets)

- **不使用已有的集成** Machine Learning **库，使用** Python **手动搭建一个神经网络**

  Construct a neural network by hand without existing Machine Learning tools

- **详细解释参数调优过程**（注：本实验对一些调试过程进行了可视化处理）

  Show the procedure of optimization(Remark: information visualization is involved)

  (1) 对比交叉熵代价函数和二次代价函数在此问题中的优劣

  Cross-Entropy function vs Quadratic cost function

  (2) Mini-batch size 神经网络的对准确率的影响

  Mini-batch size's influence on accuracy

  (3) 正则化参数值对神经网络的准确率的影响

  Regularization's influence on accuracy

  (4) 隐藏层设置对神经网络的准确率影响

  Hidden-layers' influence on accuracy

  (5) 学习率对神经网络的准确率影响

  Eta's influence on accuracy

- **找出最优神经网络**

  Find the optimized neural network for this problem

Part 2
实验内容与设计思想 (Lab Content and Design)

- 参考手写数字识别样例代码 neural-networks-and-deep-learning

  Take book:*neural networks and deep learning* as a reference

- 手动搭建一个神经网络，对 mushroom 数据集进行分类，判断蘑菇是否有毒，不断优化神经网络将分类准确度提升到极限

  Construct this neural net work by hand for classifying mushrooms according to its poisonousness(or not). Optimizing it to improve the accuracy of classification to its limit

Part 3
使用环境 (Environment)

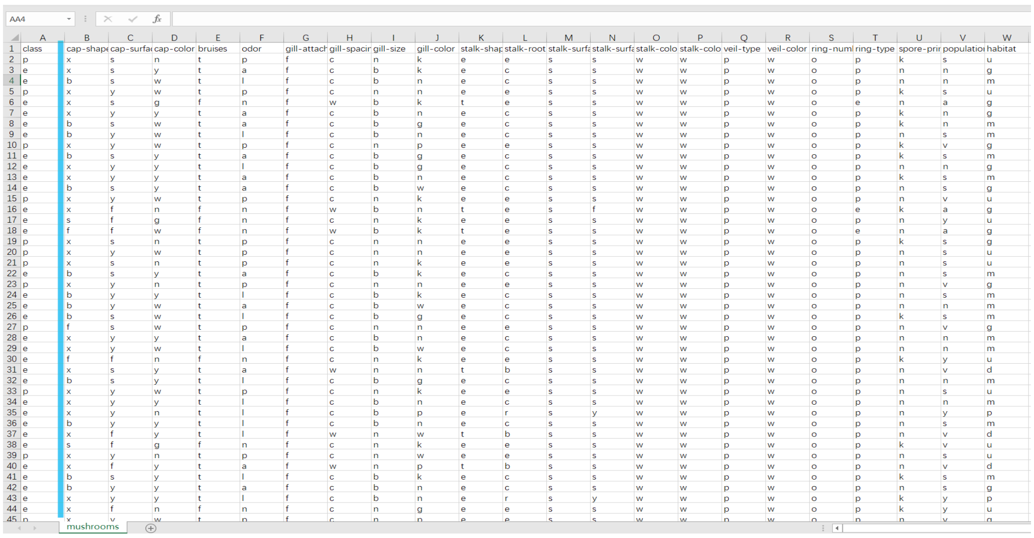
- Python 3.7
- Jupyter Lab

Part 4
实验过程 (Procedure)

Step 1
数据预处理 (Data Preprocessing)

首先看一下 mushroom.csv 数据集的数据格式，如图，class(p-e) 用于区分毒性

First, take a look at the data format of the mushroom.csv data set. As is shown below, class(P-E) indicates the poisonousness of the mushrooms



导入 csv 格式的数据，对数据进行预处理

Import the mushroom.csv and make preprocessing

```python
import pandas as pd
import numpy as np
data = pd.read_csv('mushrooms.csv')
```

为了对蘑菇进行分类，我将蘑菇的毒性标记为：

In order to classify them, we mark the poisonousness as:

(1) 有毒 (toxic)$\mapsto$ 1

(2) 无毒 (non-toxic)$\mapsto$ 0

使用 map 函数进行映射，把毒性转化为数字

Use function *map* to covert the “poisonousness” into number

```
poisonousness_map={'p':0,'e':1}
data['class']=data['class'].map(poisonousness_map)
```

接下来要将各个参数全部转化为数字，并且压缩到区间 [0,1] 之间

What we should do now is to convert the remaining parameters into numbers and squeeze them into $[0,1]$

第一种方法比较简单，直接调用预处理包

One simple method is to call python library for preprocessing

```
encoder = preprocessing.LabelEncoder()
for col in data.columns:
    data[col] = encoder.fit_transform(data[col])
data = np.array(data)
```

但因为 preprocessing 这个库附属于现成的机器学习库，也可以手动进行转换

While *preprocessing* is affiliated to a Machine Learning library, we can do it by hand instead

```
row_num=data.shape[0] #获得行
col_num=data.shape[1] #获得列
for i in range(row_num):
    for k in range(1,col_num): #一定要从1开始，不要动POISONOUSNESS
        data.iloc[i,k]=(ord(data.iloc[i,k])-ord('a'))/26 #纯粹只为把字母转化为数字
```

上述两种方法均可达成目的，处理后的数据集如图所示

Two methods stated above can both solve our problem, and the data set after preprocessing is shown below

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	0	0.884615	0.692308	0.5	0.730769	0.576923	0.192308	0.0769231	0.5	0.384615	...	0.692308	0.846154	0.846154	0.576923	0.846154	0.538462	0.576923	0.384615	0.692308	0.769231
1	1	0.884615	0.692308	0.923077	0.730769	0	0.192308	0.0769231	0.0384615	0.384615	...	0.692308	0.846154	0.846154	0.576923	0.846154	0.538462	0.576923	0.5	0.5	0.230769
2	1	0.0384615	0.692308	0.846154	0.730769	0.423077	0.192308	0.0769231	0.0384615	0.5	...	0.692308	0.846154	0.846154	0.576923	0.846154	0.538462	0.576923	0.5	0.5	0.461538
3	0	0.884615	0.923077	0.846154	0.730769	0.576923	0.192308	0.0769231	0.5	0.5	...	0.692308	0.846154	0.846154	0.576923	0.846154	0.538462	0.576923	0.384615	0.692308	0.769231
4	1	0.884615	0.692308	0.230769	0.192308	0.5	0.192308	0.846154	0.0384615	0.384615	...	0.692308	0.846154	0.846154	0.576923	0.846154	0.538462	0.153846	0.5	0	0.230769
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	1	0.384615	0.692308	0.5	0.192308	0.5	0	0.0769231	0.0384615	0.923077	...	0.692308	0.538462	0.538462	0.576923	0.538462	0.538462	0.576923	0.0384615	0.0769231	0.423077
8120	1	0.884615	0.692308	0.5	0.192308	0.5	0	0.0769231	0.0384615	0.923077	...	0.692308	0.538462	0.538462	0.576923	0.5	0.538462	0.576923	0.0384615	0.807692	0.423077
8121	1	0.192308	0.692308	0.5	0.192308	0.5	0	0.0769231	0.0384615	0.5	...	0.692308	0.538462	0.538462	0.576923	0.538462	0.538462	0.576923	0.0384615	0.0769231	0.423077
8122	0	0.384615	0.923077	0.5	0.192308	0.923077	0.192308	0.0769231	0.5	0.0384615	...	0.384615	0.846154	0.846154	0.576923	0.846154	0.538462	0.153846	0.846154	0.807692	0.423077
8123	1	0.884615	0.692308	0.5	0.192308	0.5	0	0.0769231	0.0384615	0.923077	...	0.692308	0.538462	0.538462	0.576923	0.538462	0.538462	0.576923	0.538462	0.0769231	0.423077

在将字母特征转化为数字后，再用 train_test_split 方法分离测试集和训练集，rate 为测试集和训练集的比例

After converting these characteristics into numbers, we use *train test split*to split train data and test data by ratio *rate* (*rate* is set as 0.25 in my optimized neural network)

```
train, test = train_test_split(data, test_size = rate) #RATE=0.25
```

最后，调整数据格式（LabelEncoder 是用来对分类特征值进行编码，即对不连续的数值或文本进行编码。

Finally, adjusting the data format (LabelEncoder is used to number the characteristics, it can be used to number the discontinuous text or data).

fit_transform(data)：相当于先进行 fit 再进行 transform，即把 data 装载入到字典中去以后再进行变换以得到索引值）。

fit_transform(data)：put data into a python dictionary and transform it to get the index.

```python
def data_loader():
# PREPROCESSING
data = pd.read_csv('mushrooms.csv')
encoder = preprocessing.LabelEncoder()
for Colum in data.columns:
    data[Colum] = encoder.fit_transform(data[Colum])
data = np.array(data)

train_dataset, test_dataset = train_test_split(data, test_size = 0.25)#TEST_SIZE IS CHANGEABLE
#SPLIT TEST DATA AND TRAIN DATA

#PROCESS INPUT AND OUTPUT AND LABEL
train_output = [x[0] for x in train_dataset]
train_in = np.array([x[1:] for x in train_dataset]).astype('float')
test_out = [x[0] for x in test_dataset]
test_in = np.array([x[1:] for x in test_dataset]).astype('float')


# PRACTICE VECTORIZATION
train_out_vec = [vectorized(y) for y in train_output]
train_in_vec = [np.reshape(x, (22,1)) for x in train_in]

test_out_vec = [vectorized(y) for y in test_out]
test_in_vec = [np.reshape(x, (22,1)) for x in test_in]

train_data = list(zip(train_in_vec, train_out_vec))
test_data = list(zip(test_in_vec, test_out_vec))

#DIVIDED DATASETS
return train_data, test_data
```

到此为止我们完成了对数据集的处理，接下来开始搭建神经网络

Now, we have finished the preprocessing of our dataset, and we launch the construction of the neural network

Step 2

神经网络的构建 (The Construction Of Neural Network)

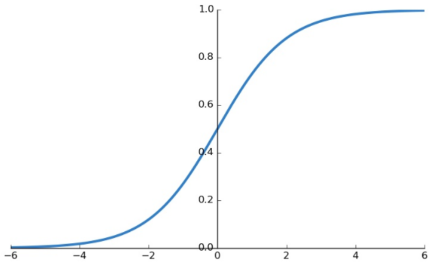
Section 1

激活函数 (Activation Function)

激活函数为 Sigmoid 函数 $\sigma(x) = \frac{1}{e^{-x}+1}, \sigma(x)' = \sigma(x)(1-\sigma(x))$

```python
def sigmoid(x):
    return 1.0/(1.0+np.exp(-x))
```

```python
def sigmoid_prime(x): #Sigmoid函数的导数用其自身可以表示
    return sigmoid(x)*(1-sigmoid(x))
```



Section 2

损失函数 (Loss Functions)

本次手动搭建一共测试了两种损失函数，首先是二次损失函数

In this Lab, two loss functions are trialed, the first of which is the Quadratic loss function

$$C_{Quadratic} = \frac{1}{2n} \sum_x \left\| y(x) - a^L(x) \right\|^2$$

实现方法为:

It's implementation is shown below:

```python
    class QuadraticCost(object):
    @staticmethod
    def fn(a, y):
        return 0.5*np.linalg.norm(a-y)**2
    @staticmethod
    def delta(z, a, y):
        return (a-y) * sigmoid_prime(z)
```

其次是交叉熵损失函数

And the other one is the Cross-Entropy loss function

$$C_{Cross-Entropy} = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)]$$

实现方法为:

And it's implementation is shown below:

```python
    class CrossEntropyCost(object):
    @staticmethod
    def fn(a, y):
        return np.sum(np.nan_to_num(-y*np.log(a)-(1-y)*np.log(1-a)))
    @staticmethod
    def delta(z, a, y):
        return (a-y)
```

Section 3
神经网络结构 (The Structure of Neural Network)

仿造 *neural networks and deep learning* 中识别 MISNT 数据集构建 *Network* 类，实现以下函数

首先传入 *sizes* 参数构建神经网络的全连接层，默认使用交叉熵代价函数和默认权重初始化

First, the parameter *sizes* are passed to construct the full connection layer of the neural network. By default, the network is initialized by the cross-entropy cost function and the default weight initializer.

```
def __init__(self, sizes, cost=CrossEntropyCost):
    """
    The list ``sizes`` contains the number of neurons in the respective layers of the network. For
        example, if the list was [2, 3, 1] then it would be a three-layer network, with the first
        layer containing 2 neurons, the second layer 3 neurons, and the third layer 1 neuron. The
        biases and weights for the network are initialized randomly, using``self.
        default_weight_initializer`` (see docstring for that method)
    """
    self.num_layers = len(sizes)
    self.sizes = sizes
    self.default_weight_initializer()
    self.cost = cost
```

需要实现对权重的初始化

We should initialize the weight first

这里使用两种权重初始化方式，默认权重初始化为均值为 0，标准差为 1 的高斯分布随机分布。第二种权重初始化后均值为 0，标准差为 $(1/n)^{1/2}$，避免隐藏神经元饱和

Two methods of initialization are implemented here, the default one use a Gaussian distribution of parameter $\mu = 0, \sigma = 1$. The second one set $\mu = 0, \sigma = (1/n)^{1/2}$ to avoid the saturation of the hidden neurons

```

def default_weight_initializer(self):
    """
    Initialize each weight using a Gaussian distribution with mean 0 and standard deviation 1 over the
         square root of the number of weights connecting to the same neuron. Initialize the biases
        using a Gaussian distribution with mean 0 and standard deviation 1. Note that the first layer
         is assumed to be an input layer, and by convention we won't set any biases for those neurons
        , since biases are only ever used in computing the outputs from later layers.
    """
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
    self.weights = [np.random.randn(y, x) for x, y in zip(self.sizes[:-1], self.sizes[1:])]
def large_weight_initializer(self):
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
    self.weights = [np.random.randn(y, x)/np.sqrt(x) for x, y in zip(self.sizes[:-1], self.sizes[1:])]
```

实现前馈函数 $\Rightarrow$ 用训练好的权重和偏置来计算网络输出（Test Dataset）

Feedforward function used trained weights and biases to calculate the output

```
def Feed_forward(self, a):
"""Return the output of the network if ``a`` is input."""
```

```
    for b, w in zip(self.biases, self.weights):
        a = sigmoid(np.dot(w, a)+b)
    return a
```

接下来是随机梯度下降函数，也就是神经网络的核心部分

Now is the function of stochastic gradient descend, which is the core of neural networks

```
def SGD(self, training_data, epochs, mini_batch_size, eta,
    Lambda = 0.0,
    evaluation_data=None,
    monitor_evaluation_accuracy=False):

    if evaluation_data: n_data = len(evaluation_data)
    n = len(training_data)

    evaluation_cost, evaluation_accuracy = [], []
    training_cost, training_accuracy = [], []

    #每个迭代期将训练集随机打乱，将数据集分成多个mini-batch
    for j in range(epochs):
    random.shuffle(training_data)
    mini_batches = [
        training_data[k:k+mini_batch_size]
        for k in range(0, n, mini_batch_size)]

    for mini_batch in mini_batches:
        self.update_mini_batch(
            mini_batch, eta, Lambda, len(training_data))
    print("Epoch %s training complete" % j)

    if monitor_evaluation_accuracy:
        accuracy = self.accuracy(evaluation_data)
        evaluation_accuracy.append(accuracy)
        print("Accuracy on evaluation data: {} / {}".format(self.accuracy(evaluation_data), n_data))

    return evaluation_accuracy
```

Section 4

其他函数 (Miscellaneous/Auxiliary Functions)

实现数据的小批次更新

Implement Mini-batch Update

```
def update_mini_batch(self, mini_batch, eta):
    """ Update the network's weights and biases by applying gradient descent using backpropagation to a
        single mini batch. The ``mini_batch`` is a list of tuples ``(x, y)``, and ``eta`` is the learning
         rate.
    """
    nabla_b = [np.zeros(b.shape) for b in self.biases]
```

```
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

反向传播算法的实现，可以参考章节 *How the backpropagation algorithm works*

The implementation of back propagation that take *How the backpropagation algorithm works* as a reference

```
def backprop(self, x, y):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # FEEDFORWARD
    activation = x
    activations = [x]
    zs = []
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    # BACKWARD PASS
    delta = (self.cost).delta(zs[-1], activations[-1], y)
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
    for l in range(2, self.num_layers):
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

还需要函数来判断网络输出和数据集相同的个数，衡量训练效果好坏

We need to evaluate the training outcome through a function

```
def accuracy(self, data):
    results = [(np.argmax(self.Feed_forward(x)), np.argmax(y))
               for (x, y) in data]
    return sum(int(x == y) for (x, y) in results)
```

总代价函数，顾名思义，用于计算一个神经网络中训练产生的代价

I use “Total cost function” to calculate the total cost of the network

```
def total_cost(self, data, Lambda):
```

```
cost = 0.0
for x, y in data:
    a = self.Feed_forward(x)
    cost += self.cost.fn(a, y) / len(data)
cost += 0.5*(Lambda/len(data))*sum(np.linalg.norm(w)**2 for w in self.weights)
return cost
```

最后，找到了最优神经网络后需要把模型保存下来 (格式为 json)

Finally, I can save the best network in a .json file

```
def save(self, filename):
data = {"sizes": self.sizes,
        "weights": [w.tolist() for w in self.weights],
        "biases": [b.tolist() for b in self.biases],
        "cost": str(self.cost.__name__)}
Json_file = open(filename, "w")
json.dump(data, Json_file) #把结果写入JSON格式文件
Json_file.close()
```

Step 3

初步分类尝试 (Preliminary Classification)

接下来就可以运行这个神经网络，尝试分类一些蘑菇了（先凭直觉选了一些参数）

And now we can launch this neural network, try to classify some mushrooms(I selected some parameters intuitively)

```
[22]: test_of_accuracy_one = best_fit_paras.SGD(train_datas, epochs, mini_batch, eta, evaluation_data = test_datas, \
                                          monitor_evaluation_accuracy = True)
```

```
Epoch 0 training complete
Accuracy on evaluation data: 1546 / 2031
Epoch 1 training complete
Accuracy on evaluation data: 1749 / 2031
Epoch 2 training complete
Accuracy on evaluation data: 1811 / 2031
Epoch 3 training complete
Accuracy on evaluation data: 1882 / 2031
Epoch 4 training complete
Accuracy on evaluation data: 1905 / 2031
Epoch 5 training complete
Accuracy on evaluation data: 1910 / 2031
Epoch 6 training complete
Accuracy on evaluation data: 1919 / 2031
Epoch 7 training complete
Accuracy on evaluation data: 1906 / 2031
Epoch 8 training complete
Accuracy on evaluation data: 1911 / 2031
Epoch 9 training complete
Accuracy on evaluation data: 1941 / 2031
Epoch 10 training complete
Accuracy on evaluation data: 1964 / 2031
Epoch 11 training complete
Accuracy on evaluation data: 1984 / 2031
Epoch 12 training complete
Accuracy on evaluation data: 2001 / 2031
Epoch 13 training complete
Accuracy on evaluation data: 1873 / 2031
Epoch 14 training complete
Accuracy on evaluation data: 2001 / 2031
```

网络运行了起来，但效果还不是最好，接下来进行参数调优，争取找到最佳网络

It works, but the outcome isn't satisfactory, I'll try to find the best network through parameter tuning

Part 5

控制变量调参 (Adjusting Parameter：Variable-Controlling Approach)

调整过程比较冗长，这里只挑选了几组比较有代表性的对照试验进行展示，更详细的数据已单独整理。

The process of adjusting is rather lengthy. Only a few groups of representative controlled trials have been selected for demonstration, and more detailed data have been sorted separately.

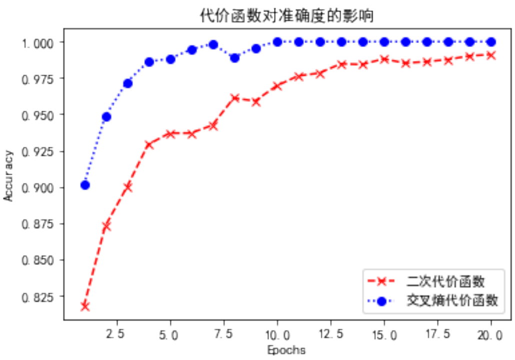
Section 1

调参：代价函数 (Loss Function)

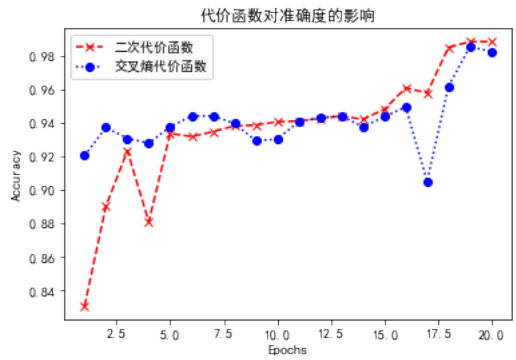
首先选定一组参数，对两种代价函数在不同 Eta 参数下的性能进行对比

Firstly, we select a set of parameters and compare the performance of two loss functions under different Etas

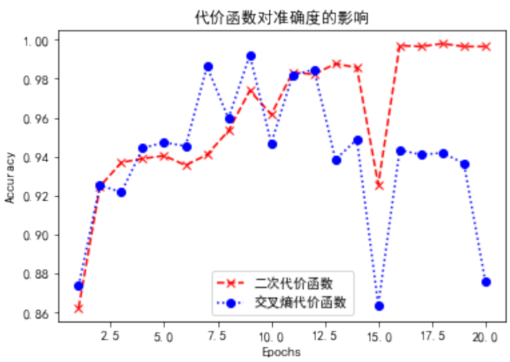
$$layers = [22, 10, 10, 2] \; epochs = 20 \; mini\_batch = 10 \; lambda = 0.05$$



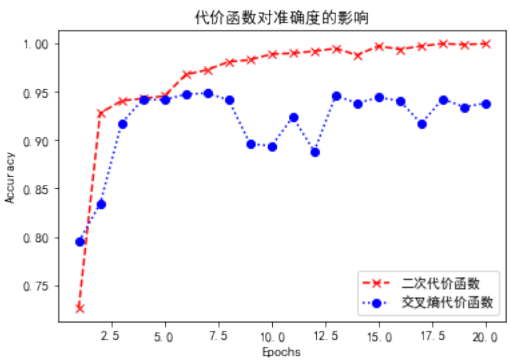
(a) $eta = 0.1$



(b) $eta = 0.2$

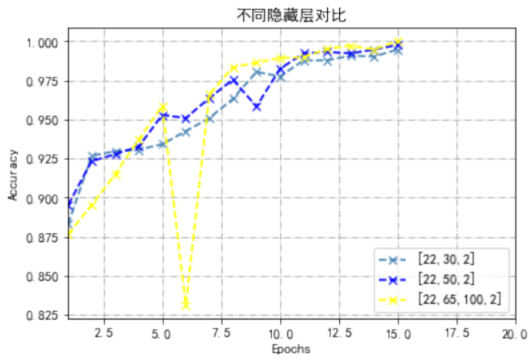


(c) $eta = 0.3$

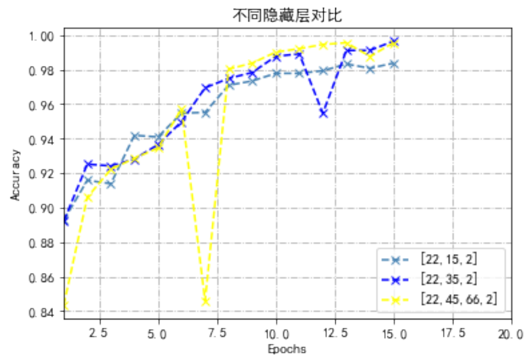


(d) $eta = 0.5$

Section 2
调参：隐藏层 (Hidden Layer)



(e) *Hidden Layer analysis I*

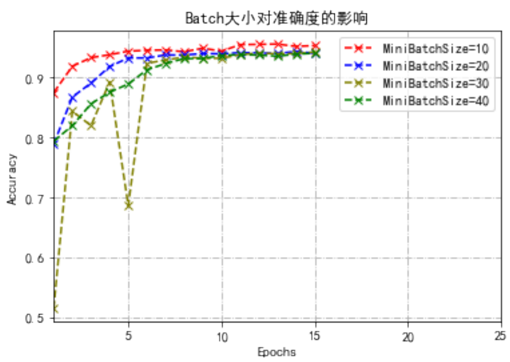


(f) *Hidden Layer analysis II*

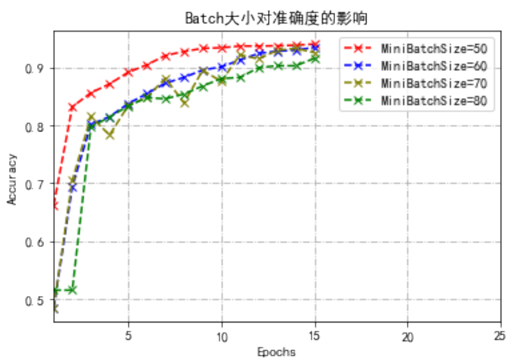
在这个分类问题中，隐藏层对网络最终的准确性影响并不大，原因应该是训练次数已经足够，网络的准确率在不同的隐藏层影响下都能到顶。

In this classification problem, the hidden layer doesn't affect the accuracy to a large extent. The reason might be that the network has already "saturated" through enough training and its accuracy can reach the limit under different hidden layers.

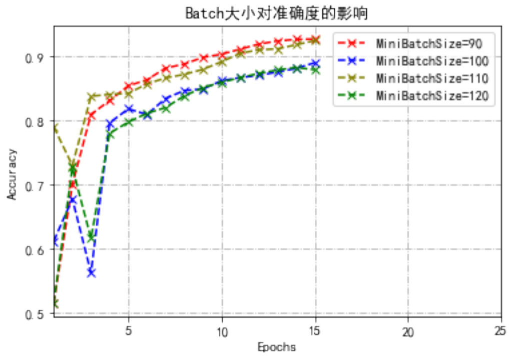
Section 3
调参：Mini-batch



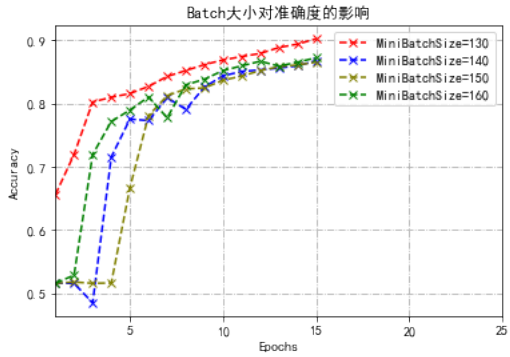
(g) $BatchSize = 10/20/30/40$



(h) $BatchSize = 50/60/70/80$



(i) $BatchSize = 90/100/110/120$

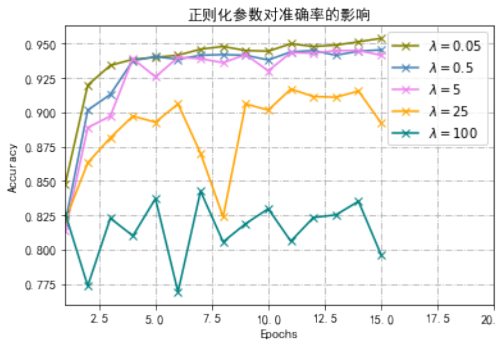


(j) $BatchSize = 130/140/150/160$

从图上可以看出，相对较小的 batch 对网络的准确度效果相差不大，较大的 batch 会明显降低网络的准确度 ($accuracy \leq 0.9$)，故最终选择 batch 大小为 10。

We can see that smaller mini-batch size doesn't affect the accuracy much, while bigger mini-batch size negatively affect the accuracy. At last, I select the batch size to be 10.

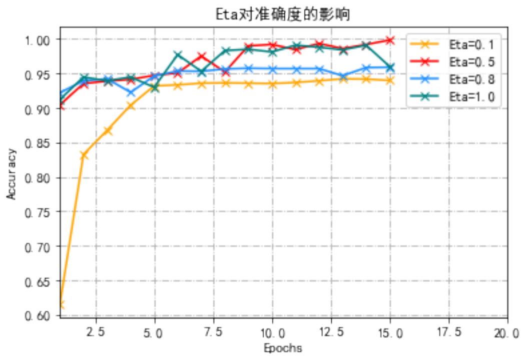
Section 4
调参：正则化参数 (Regularization)



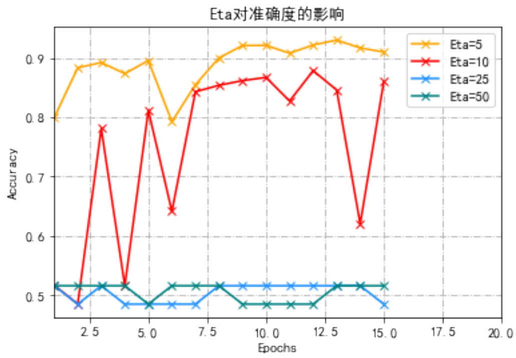
如图所示，正则化参数 λ 的大小与网络的准确度呈负相关，最优网络中 $\lambda = 0.05$。

As is shown in the graph, λ is inversely related to the accuracy of the network, we have $\lambda = 0.05$ in the best network.

Section 5
调参：学习率 (Eta)



(k) $Eta = 0.1/0.5/0.8/1.0$



(l) $Eta = 5/10/25/50$

经过上述的调参过程，找到了最优神经网络。

After the tunning process stated above, the best network is here.

$$layers = [22, 65, 100, 2]\ Epochs = 30\ Mini\_batch = 10\ Eta = 0.5\ \lambda = 0.05\ Loss\ Function = Quadratic$$

```
Epoch 9
Accuracy on evaluation data: 3236 / 3250
Epoch 10
Accuracy on evaluation data: 3239 / 3250
Epoch 11
Accuracy on evaluation data: 3245 / 3250
Epoch 12
Accuracy on evaluation data: 3240 / 3250
Epoch 13
Accuracy on evaluation data: 3246 / 3250
Epoch 14
Accuracy on evaluation data: 3250 / 3250
Epoch 15
Accuracy on evaluation data: 3245 / 3250
Epoch 16
Accuracy on evaluation data: 3248 / 3250
Epoch 17
Accuracy on evaluation data: 3250 / 3250
Epoch 18
Accuracy on evaluation data: 3250 / 3250
Epoch 19
Accuracy on evaluation data: 3249 / 3250
Epoch 20
Accuracy on evaluation data: 3250 / 3250
Epoch 21
Accuracy on evaluation data: 3250 / 3250
Epoch 22
Accuracy on evaluation data: 3248 / 3250
Epoch 23
Accuracy on evaluation data: 3250 / 3250
Epoch 24
Accuracy on evaluation data: 3250 / 3250
Epoch 25
Accuracy on evaluation data: 3250 / 3250
Epoch 26
Accuracy on evaluation data: 3247 / 3250
Epoch 27
Accuracy on evaluation data: 3250 / 3250
Epoch 28
Accuracy on evaluation data: 3250 / 3250
Epoch 29
Accuracy on evaluation data: 3250 / 3250
Epoch 30
Accuracy on evaluation data: 3250 / 3250
```

网络的效果应该是达到极限了，准确率可以稳定保持在 99.5% 以上。

It seems that the capability of this network has reached its limit, the accuracy remains $>$ 99.5% steadily.

Part 6
参考 (Reference)

(1) Code:https://github.com/mnielsen/neural-networks-and-deep-learning

(2) Instruction:http://neuralnetworksanddeeplearning.com/