Introduction to Machine Learning
Homework 1

刘潇远

April 17, 2019

1 [15pts] Decision Tree I

(1) [5pts] Assume there is a space contains three binary features X, Y, Z and the objective function is $f(x, y, z) = \neg(x \text{ XOR } y)$. Let H denotes the decision tree constructed by these three features. Please answer the following question: Is function f realizable? If the answer is yes, please draw the decision tree H otherwise please give the reason.

解:

由

$$\begin{aligned} f(x,y,z) &= \neg(x \text{ XOR } y) \\ &= \neg((\neg x \wedge y) \vee (x \wedge \neg y)) \\ &= \neg(\neg x \wedge y) \wedge \neg(x \wedge \neg y) \\ &= (x \vee \neg y) \wedge (\neg x \vee y) \\ &= (x \wedge \neg x) \vee (x \wedge y) \vee (\neg y \wedge x) \vee ((\neg y \wedge y) \\ &= (x \wedge y) \vee (\neg x \wedge \neg y) \end{aligned}$$

可知题设条件决策树可实现(realizable)，决策树如下

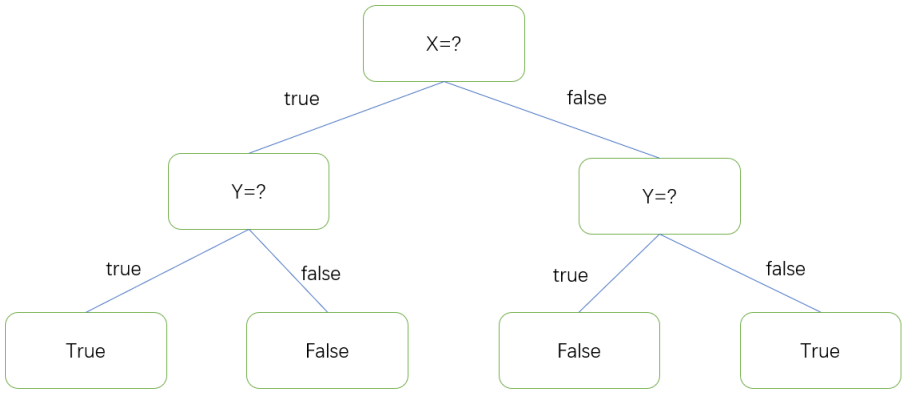

Figure 1: Decision Tree

(2) [10pts] Now we have a dataset show by Table.1:

Table 1:example dataset

X	Y	Z	f
1	0	1	1
1	1	0	0
0	0	0	0
0	1	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	0

Please use Gini value as partition criterion to draw the decision tree from the dataset. When Gini value is same for two features, please follow the alphabetical order.

解：

首先计算用X划分之后所获得的两个分支结点的Gini value为

$$Gini(D_{X=1}) = 1 - \frac{2}{4}^2 - \frac{2}{4}^2 = 0.5$$
$$Gini(D_{X=0}) = 1 - \frac{2}{4}^2 - \frac{2}{4}^2 = 0.5$$

同样可得用Y和Z划分后所得节点的Gini value

$$Gini(D_{Y=1}) = 0.5$$
$$Gini(D_{Y=0}) = 0.5$$
$$Gini(D_{Z=1}) = \frac{4}{9} = 0.444$$
$$Gini(D_{Z=0}) = 0$$

于是可计算出属性为X的Gini index为

$$\begin{aligned} Gini_index(D, X) &= \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v) \\ &= \frac{4}{8} \times 0.5 + frac48 \times 0.5 \\ &= 0.5 \end{aligned}$$

同样可得Y和Z的Gini index为

$$Gini_index(D, Y) = 0.5$$
$$Gini_index(D, Z) = 0.333$$

则$a_* = \text{argmin}_{a \in A}\, Gini_index(D, a) = Z$，因此按照属性Z划分后的dataset为

$D_{Z=1}$

X	Y	Z	f
1	0	1	1
0	1	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	0

$D_{Z=0}$

X	Y	Z	f
1	1	0	0
0	0	0	0

其中，对于$D_{Z=0}$，所有样本都属于同一类，即$f = 0$，因此不必再划分
对于$D_{Z=1}$，计算X和Y的Gini index为

$$Gini_index(D_{Z=1}, X) = 0.444 Gini_index(D_{Z=1}, Y) = 0.444$$

按照字母序，选取X为划分属性，$D_{Z=1}$划分如下：

$D_{X=1,Z=1}$

X	Y	Z	f
1	0	1	1
1	0	1	1
1	1	1	0

$D_{X=0,Z=1}$

X	Y	Z	f
0	1	1	1
0	0	1	0
0	1	1	1

此时只剩下属性Y，使用Y对其进行划分：

$D_{X=1,Y=1,Z=1}$

X	Y	Z	f
1	1	1	0

$D_{X=1,Y=0,Z=1}$

X	Y	Z	f
1	0	1	1
1	0	1	1

$D_{X=0,Y=1,Z=1}$

X	Y	Z	f
0	1	1	1
0	1	1	1

$D_{X=0,Y=0,Z=1}$

X	Y	Z	f
0	0	1	0

显然上面四个数据集都不能继续划分，所以数据集D被划分为了$D_{Z=0}$、$D_{X=1,Y=1,Z=1}$、$D_{X=1,Y=0,Z=1}$、$D_{X=}$和$D_{X=0,Y=0,Z=1}$，决策树如下

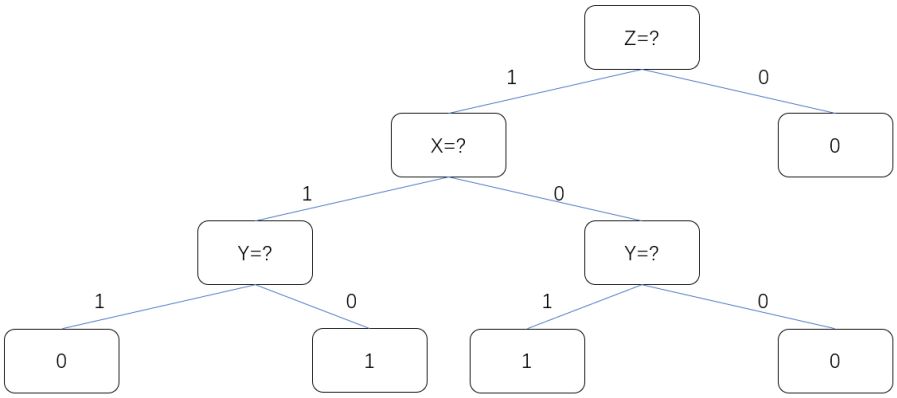

Figure 2: Decision Tree

2 [25pts] Decision Tree

Consider the following matrix:

$$\begin{bmatrix} 24 & 53 & 23 & 25 & 32 & 52 & 22 & 43 & 52 & 48 \\ 40 & 52 & 25 & 77 & 48 & 110 & 38 & 44 & 27 & 65 \end{bmatrix}$$

which contains 10 examples and each example contains two features x_1 and x_2. The corresponding label of these 10 examples as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

In this problem, we want to build a decision tree to do the classification task.
(1) [5pts] Calculate the entropy of the root node.
解：
信息熵定义为

$$Ent(D) = -\sum_{k=1}^{|y|} p_k log_2 p_k$$

显然$|y| = 2$。在决策树学习开始时，根节点包含所有样例，其中正例占$\frac{6}{10}$反例占$\frac{4}{10}$，因此根结点信息熵为

$$\begin{aligned} Ent(D) &= -\sum_{k=1}^{2} p_k log_2 p_k \\ &= -(\frac{6}{10} log_2 \frac{6}{10} + \frac{4}{10} log_2 \frac{4}{10}) \\ &= 0.9710 \end{aligned}$$

(2) [10pts] Building your decision tree. What is your split rule and the classification error?
解：
数据集包含两个连续属性，对于属性1，其候选划分点集合包含9个候选值:$T_1 = 22.5, 23.5, 24.5, 28.5, 37.5, 45.5, 50, 52.5$，根据

$$\begin{aligned} Gain(D, a) &= max_{t \in T_a} Gain(D, a, t) \\ &= max_{t \in T_a} Ent(D) - \sum_{\lambda \in -,+} \frac{|D_t^\lambda|}{|D|} Ent(D_t^\lambda) \end{aligned}$$

计算出信息增益为0.1445，对应划分点52.5

同理属性2的信息增益为0.3220，对应划分点26，因此选取属性2进行根节点划分，划分后的数据集D1包含第$\{1, 2, 4, 5, 6, 7, 8, 10\}$列样本，D0包含第$\{3, 9\}$列样本，由于D0中样本属于同一类，因此无需继续划分，下面对D1进行划分

D1信息熵为0.8113，计算属性1和属性2的信息增益及其对应划分点分别为，0.3113,37.5和0.2045,58.5，因此是选取属性1对D1进行划分，划分后的数据集D1'包含第$\{1, 4, 5, 7\}$列样本，D0'包含第$\{2, 6, 8, 10\}$列样本，其中D1' 属于同一类，无需继续划分，下面对D0'进行划分

D0'的信息熵为1，计算属性1和属性2的信息增益及其对应划分点为0.3113,45.5和1,58.5，因此选取属性2对D0'进行划分，划分后的D0'的两个子集中数据都分别属于同一类，划分结束，决策树如下

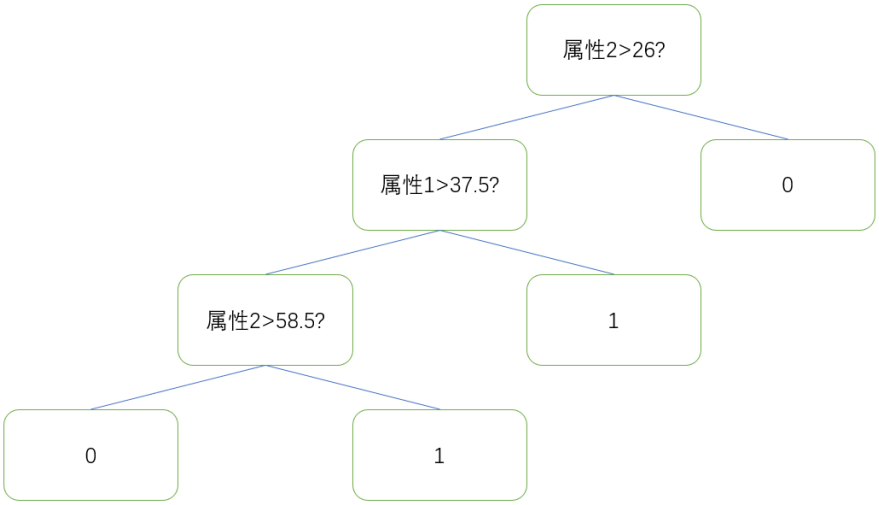

Figure 3: Decision Tree

(3) [10pts] A multivariate decision tree is a generalization of univariate decision trees, where more than one attribute can be used in the decision for each split. That is, the split need not be orthogonal to a feature's axis.

Building a multivariate decision tree where each decision rule is a linear classifier that makes decisions based on the sign of $\alpha x_1 + \beta x_2 - 1$. What is the depth of your tree, as well as α and β?

解：

经计算，数据集线性可分，a=-0.8203，b=0.6810，决策树如下

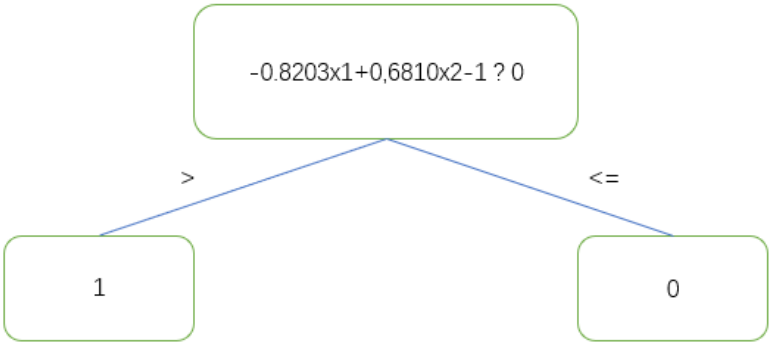

Figure 4: Decision Tree

3 [25pts] Convolutional Neural Networks

Using Fig. **??** as an example. Assuming that the loss function of the convolutional neural network is cross-entropy:

(1) [5 pts] Briefly describe the forward propagation process;
解：
输入层28*28的输入。卷积层C1的输出是深度为6，每层24*24的特征，因此卷积层采用了6个滤波器，提取了输入的6个特征，输入单元的大小是5*5，对5*5的数据进行卷积，步幅为1，由此就得到了图示6个24*24的map。之后采样层S2 保持深度不变，在每个深度上对进行规模为2*2 的采样，采样得到6个12*12的map。卷积层C3采用了2个滤波器，因此将采样层S2的输出由深度6变为深度12，之后对S2输出的每个深度切片进行5*5的卷积得到4*4的map，采样层S4同样采用2*2的采样方式得到12个4*4的map。至此，原始输入被映射成了12维特征向量，最后通过12个神经元构成的全连接输出层进行输出。

(2) [5 pts] What is the difference between Relu and Sigmoid activation functions; 解：
Relu激活函数为

$$f_R(x) = max(0, x)$$

sigmoid函数为

$$f_s(x) = \frac{1}{1 + e^x}$$

二者导数分别为

$$f'_R(x) = \begin{cases} 0 & max\{0, x\} = 0 \\ 1 & max\{0, x\} = x \end{cases}$$

$$f'_s(x) = f(x)(1 - f(x))$$

显然，对于sigmoid激活函数，$f_s(x)$两端饱和，$f'_s(x)$是二次型，在梯度反向传播的过程中容易出现梯度消失问题，使得训练网络时收敛过慢。而对于ReLU激活函数，$f_R(x)$ 分段线性，$f'_R(x)$也是分段线性，避免了梯度消失的问题，训练过程中收敛速度很快

(3) [5 pts] Derivation of the fully connected layer;

(4) [5 pts] Derivation of the pooling layer with average pooling;

(5) [5 pts] Derivation of the convolutional layer with Relu;
以上三问合并在一起解答
解：
损失函数为

$$E = \frac{1}{m} \sum_{k=1}^{m} E_k$$

$$E_k = -\sum_{j=1}^{l} y_j^k log \hat{y}_j^k$$

对于全连接层，只需将教材上的证明步骤进行简单修改，将损失函数和激活函数进行更正即可得到

$$\begin{aligned}
&\Delta w_{hj} = \eta g_j b_h \\
&\Delta \theta_j = -\eta g_j \\
&g_j = -\frac{\partial E_k}{\partial y_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} = \frac{y_j^k}{\hat{y}_j^k} \cdot f'(\beta_j - \theta_j) \\
&f'(x) = f'_R(x)
\end{aligned}$$

卷积层：
前向传播公式为

$$\begin{aligned}
\beta_{i,j}^l &= \sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-1} a_{i-i',j-j'}^{l-1} \times w_{i',j'} \\
a^l &= f(\beta^l - \theta_l)
\end{aligned}$$

反向传播公式为

$$\begin{aligned}
\frac{\partial E}{\partial w_{i,j}^l} &= \sum_{i'} \sum_{j'} \frac{\partial E}{\partial z_{i'j'}^l} \frac{\partial z_{i'j'}}{\partial w_{i,j}^l} \\
\frac{\partial z_{i'j'}}{\partial w_{i,j}^l} &= \frac{\sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-1} a_{i-i',j-j'}^{l-1} \times w_{i',j'}}{\partial w_{i,j}^l}
\end{aligned}$$

$\frac{\partial E}{\partial z_{i'j'}^l}$的求解是一个迭代的过程

$$\begin{aligned}
\frac{\partial E}{\partial z_{i'j'}^{l-1}} &= \sum_{i'=0} \sum j' = 0 \frac{\partial E}{\partial z_{i-i',j-j'}^l} \frac{\partial z_{i-i',j-j'}^l}{\partial z_{i,j}^{l-1}} \\
\frac{\partial z_{i-i',j-j'}^l}{\partial z_{i,j}^{l-1}} &= \frac{\partial \sum_s \sum_t z_{i-i'+s,j-j'+t}^{l-1} w_{s,t}^l}{\partial z_{i,j}^{l-1}} = w_{i',j'}^l
\end{aligned}$$

$$\frac{\partial E}{\partial z_{i'j'}^{l-1}} = \sum_{i'=0} \sum j' = 0 \frac{\partial E}{\partial z_{i-i',j-j'}^l} w_{i',j'}^l$$

池化层：
前向传播公式，采用平均策略

$$\beta_{i,j}^l = \frac{1}{k_1 \times k_2} \sum_{i'=0}^{k_1-1} \sum_{j'=0}^{k_2-1} a_{i*k_1+i',j*k_2+j'}^{l-1}$$

反向传播公式为

$$\frac{\partial E}{\partial a_{i,j}} = \frac{1}{k_1 \times k_2}$$

4 [35 pts] Neural Network in Practice

In this task, you are asked to build a Convolutional Neural Networks (CNNs) from scratch and examine performance of the network you just build on **MNIST** dataset. Fortunately, there are some out-of-the-box deep learning tools that can help you get started very quickly. For this task, we would like to ask you to work with the **Pytorch** deep learning framework. Additionally, Pytorch comes with a built-in dataset class for MNIST digit classification task in the **torchvision** package, including a training set and a validation set. You may find a pytorch introduction at here. Note that, you can use CPU or GPU for training at your choice.

Please find the detailed requirements below.

(1) [5 pts] You are encouraged to implement the code using *Python3*, implementations in any other programming language will not be judged. Please name the source file (which contains the main function) as *CNN_main.py*. Finally, your code needs to print the performance on the provided validation set once executed.

(2) [10 pts] Use any type of CNNs as you want and draw graphs to show your network architecture in the submitted report. You are encouraged to try more architectures.

(3) [15 pts] During training, you may want to try some different optimization algorithms, such as SGD, Adam. Also, you need to study the effect of learning rate and the number of epoch, on the performance (accuracy).

(4) [5 pts] Plot graphs (learning curves) to demonstrate the change of training loss as well as the validation loss during training.

解：
MINST是一个手写数字识别的数据集，可以pytorch的datasets包直接进行下载
组建网络时，需要考虑的有，网络结构、梯度下降优化算法和步长。在实验中我首先尝试了0.1的步长，然后考虑到我是用CPU进行训练，将训练轮数定为20.优化算法及CNN 网络结构如下

4.1 梯度下降优化

梯度下降法作为机器学习中较常使用的优化算法，其有着三种不同的形式：批量梯度下降（Batch Gradient Descent）、随机梯度下降（Stochastic Gradient Descent）以及小批量梯度下降（Mini-Batch Gradient Descent）。其中小批量梯度下降法也常用在深度学习中进行模型的训练。
考虑采用均方误差作为代价函数、只含一个特征的线性回归，其假设函数和代价函数如下

$$h_\theta(x_i) = \theta_1 x_i$$

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

4.1.1 批量梯度下降

批量梯度下降是最原始的形式，它是指在每次迭代时使用所有样本进行梯度更新,对损失函数求偏导得

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i^j$$

由此得参数更新

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i^j$$

批量梯度下降得好处时一次迭代能够对所有样本进行计算，可以利用矩阵进行并行计算，而且由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，批量梯度下降一定能够得到全局最优。但是批量梯度下降也有缺点，主要在于当样本数目m很大时，每次迭代需要对所有样本进行计算，训练过程回很慢，而且不同样本的训练有可能相互抵消

4.1.2 随机梯度下降

随机梯度下降法不同于批量梯度下降，随机梯度下降是每次迭代使用一个样本来对参数进行更新，加快训练速度，对于第i个样本，其目标函数为

$$J_i(\theta) = \frac{1}{2}(h_\theta(x_i) - y_i)^2$$

对目标函数求偏导得

$$\frac{\partial J_i(\theta)}{\partial \theta_j} = (h_\theta(x_i) - y_i)x_i^j$$

因此随机梯度下降的参数更新如下

$$\theta_j := \theta_j - \alpha(h_\theta(x_i)y_i)x_i^j$$

随机梯度下降法的优点在于其损失函数不是全部数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。但是他也有缺点，随机梯度下降的准确度会有所下降，即使在目标函数为强凸函数的情况下，随随机梯度下降仍然无法做到线性收敛，而且由于单个样本并不能代表全体样本的趋势，随机梯度下降有可能收敛到局部最优。而且批量梯度下降并行程度高的优点也消失了。

假设有30万个样本，对于批量梯度下降而言，每轮迭代需要计算30万个样本才能对参数进行一次更新，而随机梯度下降每次只需要一个样本，就算是使用了所有训练样本，也是计算了30万样本，参数被更新了30万次，这可以保证收敛到一个合适的最小值。而批量梯度下降一次未必能收敛到最小值，所以需要计算超过30 万个样本，因此随机梯度下降的速度快于批量梯度下降

4.1.3 小批量梯度下降

小批量梯度下降是对批量梯度下降以及随机梯度下降的折中，也就是说，每次迭代使用$batch_size$（DataLoader函数的参数）个样本进行参数更新，在我的程序中，训练集取值64，测试集取值128。小批量梯度下降的参数更新为

$$\theta_j := \theta_j - \alpha \frac{1}{batch\_size} \sum_{i=k}^{k+batch\_size-1} (h_\theta(x_i) y_i) x_i^j$$

小批量梯度下降与批量梯度下降一样能有运用矩阵并行运算，因此每个$batch_size$上与随机随机梯度下降单个数据进行参数更新速度差不多，而且能够大大减少收敛所需要的迭代次数，同时使得收敛结果媲美梯度下降的效果。但是$batch_size$ 的选择也需要遵循一定的原则，选择不当可能会带来一些问题，我选择64和128的原因是看网上资料，大家都是这么选的。。。

4.2 CNN网络结构

4.2.1

首先尝试了如下结构的CNN网络

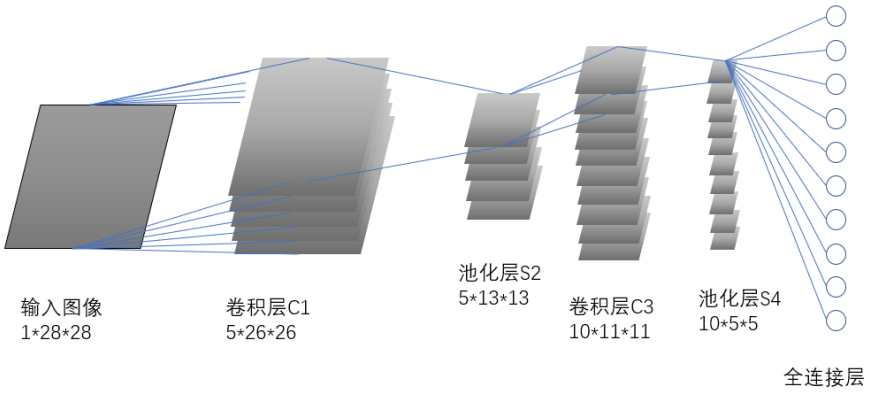

Figure 5: CNN Structure

迭代20轮的结果如下所示，测试集准确率在98%附近震荡

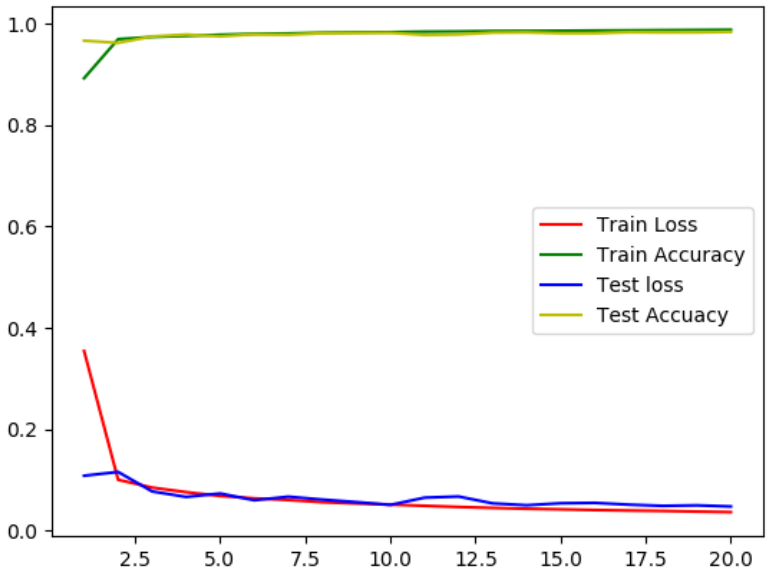

Figure 6: Performance

4.2.2

考虑到第三题的图。。。又尝试了如下结构的CNN网络

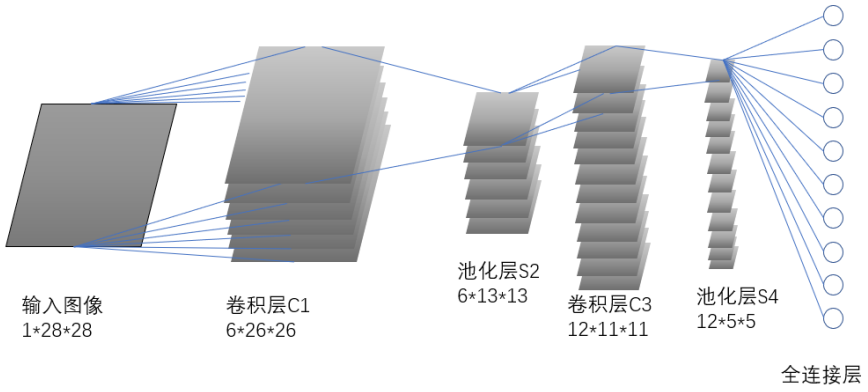

Figure 7: CNN Structure

迭代20轮的结果如下所示，测试集准确率在99%附近震荡

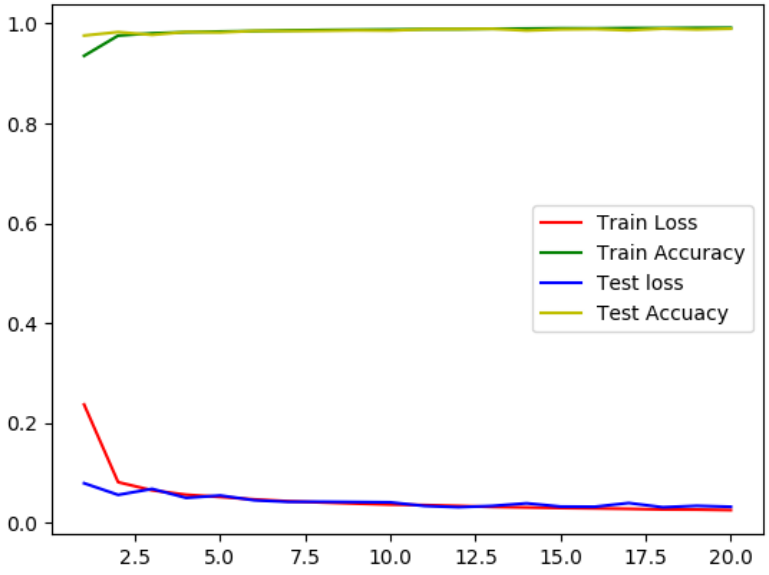

Figure 8: Performance

4.2.3

考虑到要求（2），又尝试了如下结构的CNN网络

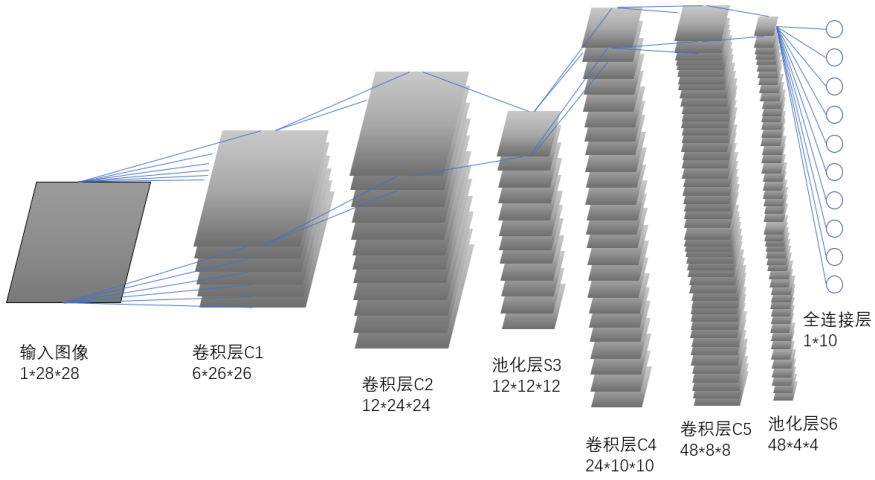

Figure 9: CNN Structure

迭代20轮的结果如下所示，测试集准确率在99%附近震荡

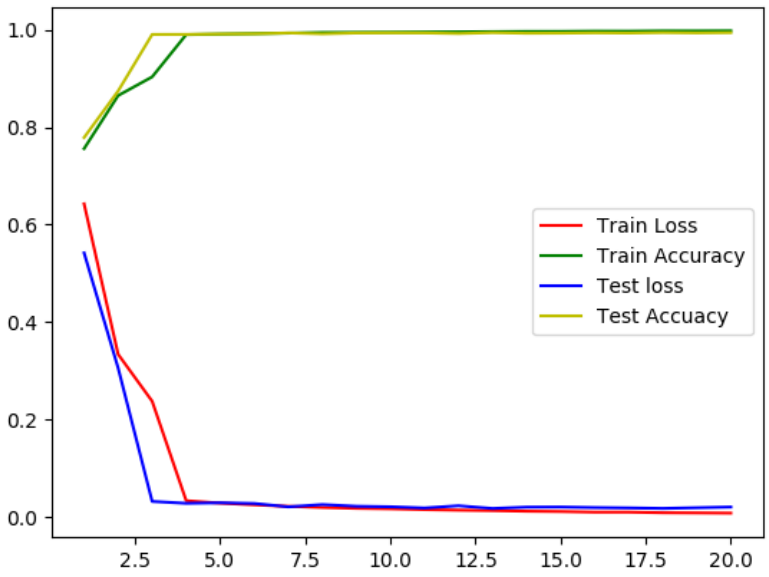

Figure 10: Performance

上述三种CNN结构，分别对用$CNN1_main.py$、$CCN2_main.py$和$CNN_main.py$ 。发现这三个在运行过程中，测试集的准确率在震荡，原因可能是更新步长没有动态调整。
在实验中我还发现一个问题，就是每次运行同一个程序其结果都会有所不同，比如第一个CNN结构，最差的一次陷入局部最优，正确率在58%附近震荡。