

# 神经网络作业

助教：杨雪松

指导教师：高 阳

[yangxuesong@smail.nju.edu.cn](mailto:yangxuesong@smail.nju.edu.cn)

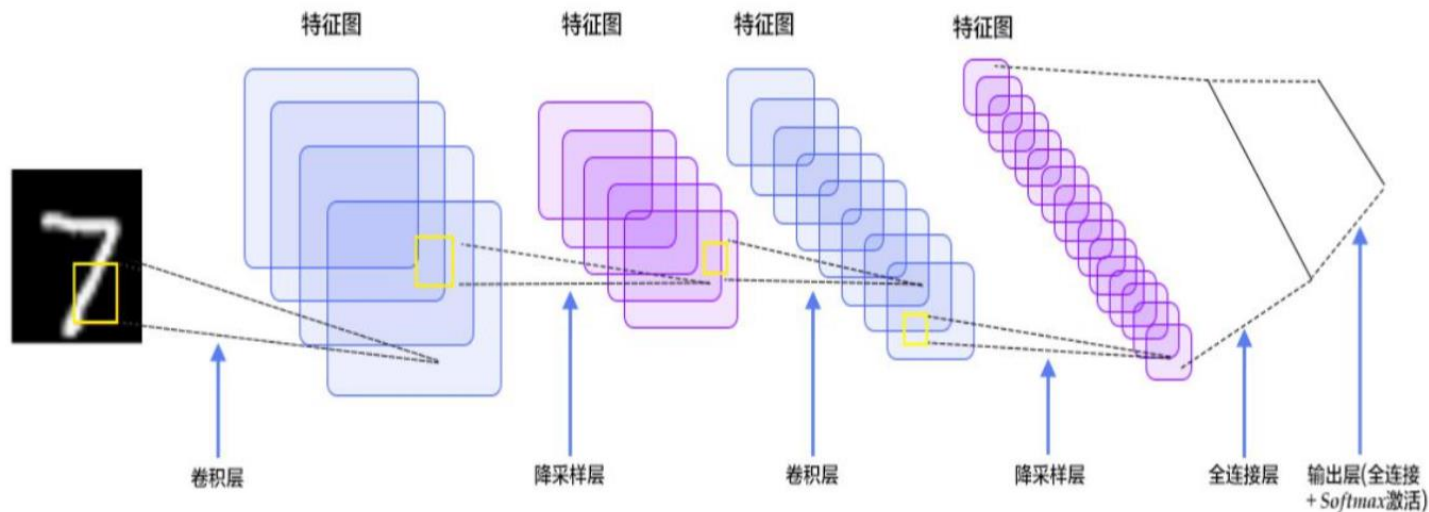
✓ 整体架构:

 输入层

 卷积层

 池化层

 全连接层




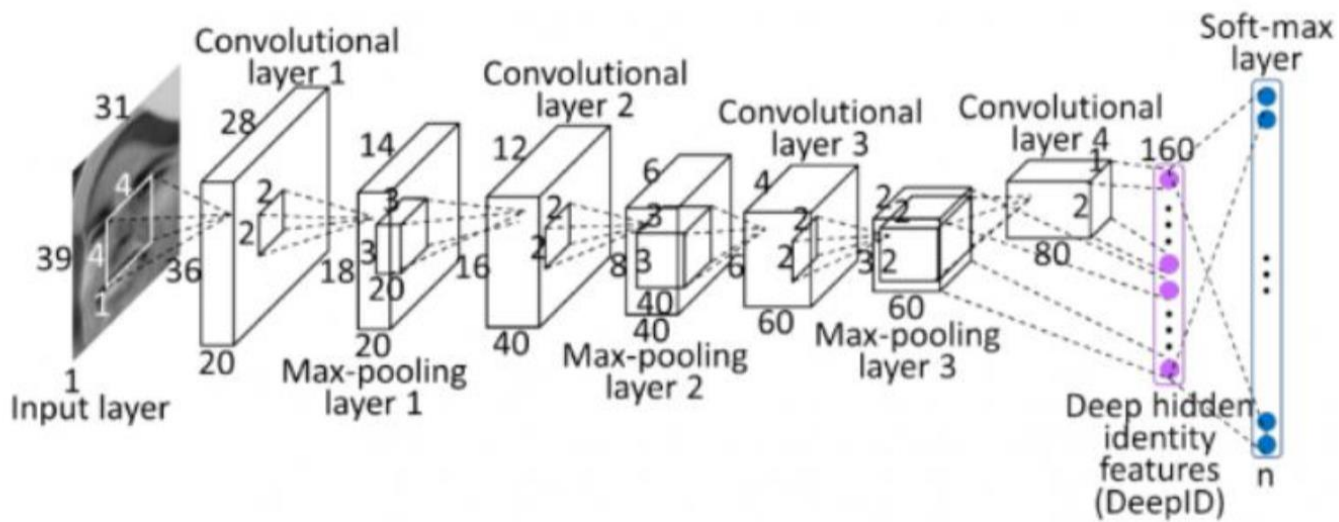
✓ 卷积层涉及参数:

### 滑动窗口步长

## 卷积核尺寸

## 边缘填充

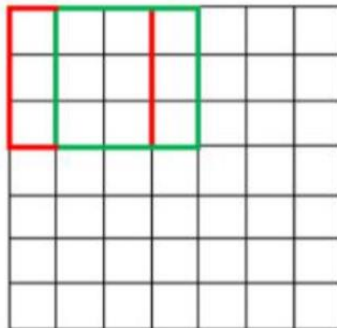
 卷积核个数



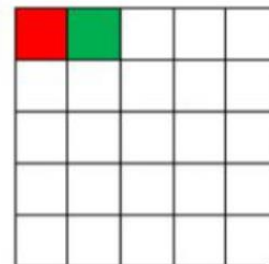
✓ 步长:

✎ 步长为1的卷积:

7 x 7 Input Volume

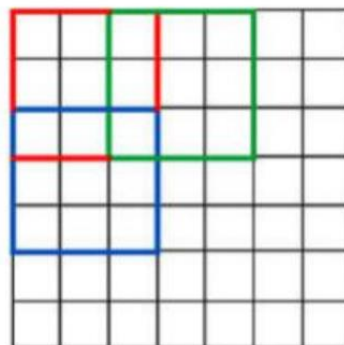


5 x 5 Output Volume

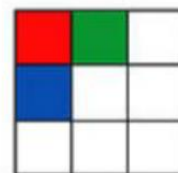


✎ 步长为2的卷积:

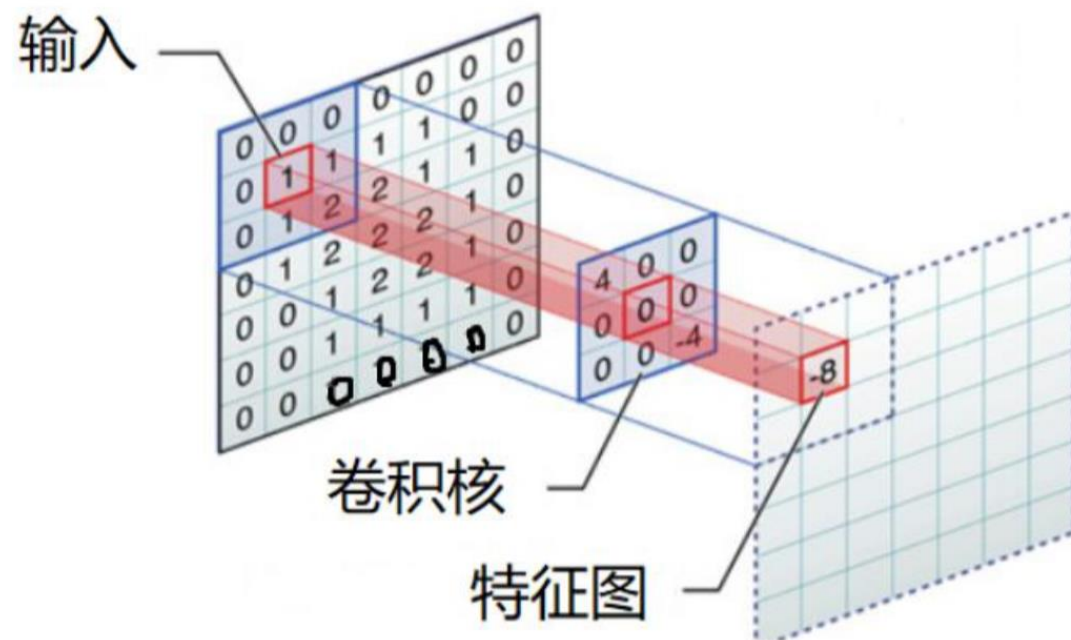
7 x 7 Input Volume



3 x 3 Output Volume



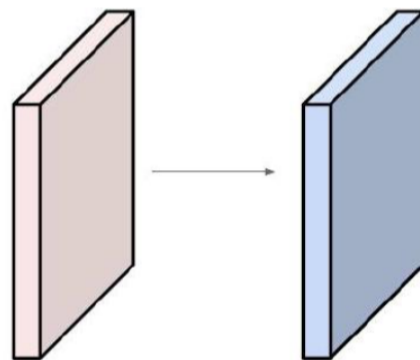
✓ 边界填充:



✓ 卷积结果计算公式：

✎ 长度：  $H_2 = \frac{H_1 - F_H + 2P}{S} + 1$

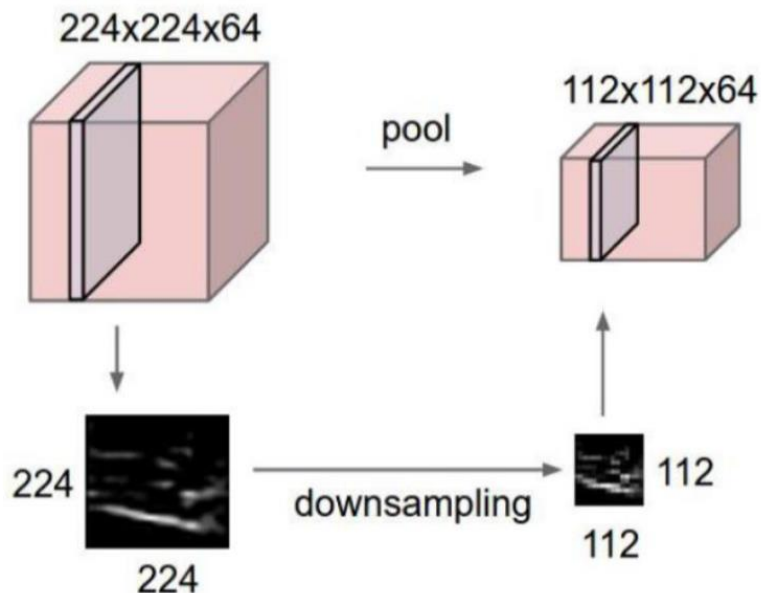
✎ 宽度：  $W_2 = \frac{W_1 - F_W + 2P}{S} + 1$



✎ 如果输入数据是 $32 \times 32 \times 3$ 的图像，用10个 $5 \times 5 \times 3$ 的filter来进行卷积操作，指定步长为1，边界填充为2，最终输出的规模为？

✎  $(32 - 5 + 2 \times 2) / 1 + 1 = 32$ ，所以输出规模为 $32 \times 32 \times 10$ ，经过卷积操作后也可以保持特征图长度、宽度不变。

✓ 池化层:

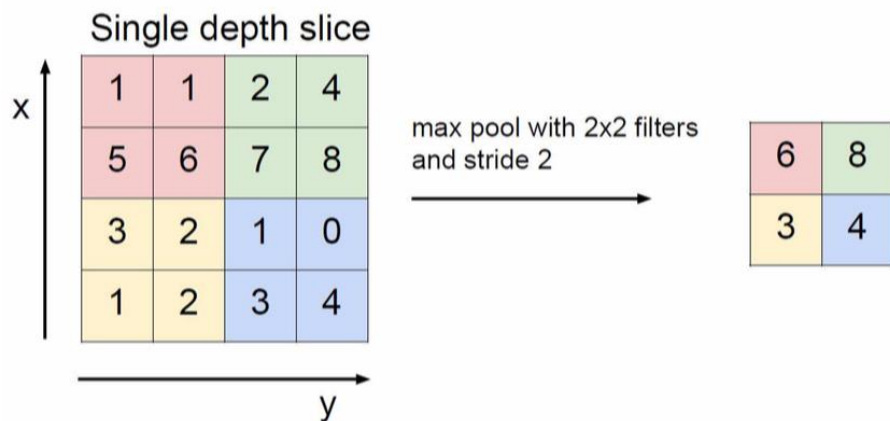


1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

7	9
8	

✓ 最大池化:

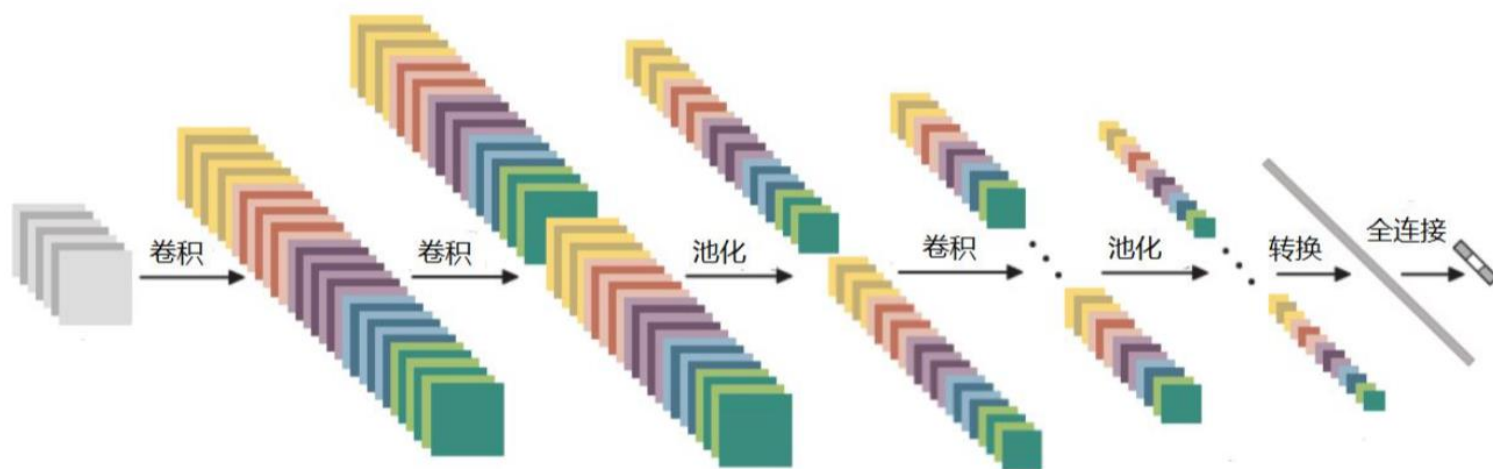
MAX POOLING



1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

7	9
8	

✓ 特征图变化:



# 作业要求

用神经网络实现“手写体识别”。可以尝试使用一些图像预处理技术（去噪，归一化，分割等），再使用CNN进行特征提取。

可以采用各种框架：Pytorch(推荐)，Keras，Tensorflow。

- 需要提交的内容包括但不限于：
  - 代码，关键部分代码需要注释；
  - PDF文档，文档中要有明确的过程说明、样例截图等。
- 数据集说明：
  - Mnist数据集：60000个训练样本和10000个测试样本组成，每个样本都是一张 $28 * 28$ 像素的灰度手写数字图片；
  - 数据集下载：[官方网站](#)。一共4个文件，训练集、训练集标签、测试集、测试集标签。



# 处理过程（样例）

- 搭配实验环境，导入包
- 给定输入输出分别构建训练集和测试集（验证集）
- **DataLoader**来迭代取数据

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```



- 卷积网络模块构建

一般卷积层，relu层，池化层可以写成一个套餐

注意卷积最后结果还是一个特征图，需要把图转换成向量才能做分类或者回归任务。

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        self.out = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output
```

# 输入大小 (1, 28, 28)

# 灰度图

# 要得到多少个特征图

# 卷积核大小

# 步长

# 如果希望卷积后大小跟原来一样，需要设置padding

# 输出的特征图为 (16, 28, 28)

# relu层

# 进行池化操作 (2x2 区域)，输出结果为: (16, 14, 14)

# 下一个套餐的输入 (16, 14, 14)

# 输出 (32, 14, 14)

# relu层

# 输出 (32, 7, 7)

# 全连接层得到的结果

# flatten操作，结果为: (batch\_size, 32 \* 7 \* 7)

- 准确率作为评估标准

```
def accuracy(predictions, labels):  
    pred = torch.max(predictions.data, 1)[1]  
    rights = pred.eq(labels.data.view_as(pred)).sum()  
    return rights, len(labels)
```

- 训练网络模型

```
# 实例化  
net = CNN()  
#损失函数  
criterion = nn.CrossEntropyLoss()  
#优化器  
optimizer = optim.Adam(net.parameters(), lr=0.001) #定义优化器, 普通的随机梯度下降算法
```

#开始训练循环

```
for epoch in range(num_epochs):
```

#当前epoch的结果保存下来

```
train_rights = []
```

```
for batch_idx, (data, target) in enumerate(train_loader): #针对容器中的每一个批进行循环
```

```
    net.train()
```

```
    output = net(data)
```

```
    loss = criterion(output, target)
```

```
    optimizer.zero_grad()
```

```
    loss.backward()
```

```
    optimizer.step()
```

```
    right = accuracy(output, target)
```

```
    train_rights.append(right)
```

```
if batch_idx % 100 == 0:
```

```
    net.eval()
```

```
    val_rights = []
```

```
for (data, target) in test_loader:
```

```
    output = net(data)
```

```
    right = accuracy(output, target)
```

```
    val_rights.append(right)
```

#准确率计算

```
train_r = (sum([tup[0] for tup in train_rights]), sum([tup[1] for tup in train_rights]))
```

```
val_r = (sum([tup[0] for tup in val_rights]), sum([tup[1] for tup in val_rights]))
```

```
print('当前epoch: {} [{} / {}] ( {:.0f}%) \t 损失: {:.6f} \t 训练集准确率: {:.2f}% \t 测试集正确率: {:.2f}%')
```

当前epoch: 0	[0/60000 (0%)]	损失: 2.298275	训练集准确率: 18.75%	测试集正确率: 16.69%
当前epoch: 0	[6400/60000 (11%)]	损失: 0.366936	训练集准确率: 77.09%	测试集正确率: 91.76%
当前epoch: 0	[12800/60000 (21%)]	损失: 0.197412	训练集准确率: 85.04%	测试集正确率: 95.32%
当前epoch: 0	[19200/60000 (32%)]	损失: 0.065437	训练集准确率: 88.57%	测试集正确率: 95.96%
当前epoch: 0	[25600/60000 (43%)]	损失: 0.245751	训练集准确率: 90.43%	测试集正确率: 97.05%
当前epoch: 0	[32000/60000 (53%)]	损失: 0.116508	训练集准确率: 91.65%	测试集正确率: 97.33%
当前epoch: 0	[38400/60000 (64%)]	损失: 0.106026	训练集准确率: 92.51%	测试集正确率: 97.47%
当前epoch: 0	[44800/60000 (75%)]	损失: 0.024781	训练集准确率: 93.20%	测试集正确率: 97.79%
当前epoch: 0	[51200/60000 (85%)]	损失: 0.040254	训练集准确率: 93.77%	测试集正确率: 97.44%
当前epoch: 0	[57600/60000 (96%)]	损失: 0.013604	训练集准确率: 94.19%	测试集正确率: 97.57%
当前epoch: 1	[0/60000 (0%)]	损失: 0.038379	训练集准确率: 100.00%	测试集正确率: 97.90%
当前epoch: 1	[6400/60000 (11%)]	损失: 0.091921	训练集准确率: 97.94%	测试集正确率: 98.26%
当前epoch: 1	[12800/60000 (21%)]	损失: 0.082685	训练集准确率: 97.88%	测试集正确率: 98.12%
当前epoch: 1	[19200/60000 (32%)]	损失: 0.030613	训练集准确率: 97.95%	测试集正确率: 98.53%
当前epoch: 1	[25600/60000 (43%)]	损失: 0.098491	训练集准确率: 97.96%	测试集正确率: 98.30%
当前epoch: 1	[32000/60000 (53%)]	损失: 0.078065	训练集准确率: 97.97%	测试集正确率: 98.50%
当前epoch: 1	[38400/60000 (64%)]	损失: 0.013370	训练集准确率: 98.02%	测试集正确率: 98.55%
当前epoch: 1	[44800/60000 (75%)]	损失: 0.065581	训练集准确率: 98.09%	测试集正确率: 98.65%
当前epoch: 1	[51200/60000 (85%)]	损失: 0.077535	训练集准确率: 98.12%	测试集正确率: 98.23%
当前epoch: 1	[57600/60000 (96%)]	损失: 0.007826	训练集准确率: 98.16%	测试集正确率: 98.65%
当前epoch: 2	[0/60000 (0%)]	损失: 0.170131	训练集准确率: 98.44%	测试集正确率: 98.57%
当前epoch: 2	[6400/60000 (11%)]	损失: 0.046841	训练集准确率: 98.64%	测试集正确率: 98.40%
当前epoch: 2	[12800/60000 (21%)]	损失: 0.095354	训练集准确率: 98.50%	测试集正确率: 98.58%
当前epoch: 2	[19200/60000 (32%)]	损失: 0.009594	训练集准确率: 98.58%	测试集正确率: 98.68%
当前epoch: 2	[25600/60000 (43%)]	损失: 0.017973	训练集准确率: 98.62%	测试集正确率: 98.82%
当前epoch: 2	[32000/60000 (53%)]	损失: 0.045781	训练集准确率: 98.66%	测试集正确率: 98.63%
当前epoch: 2	[38400/60000 (64%)]	损失: 0.056535	训练集准确率: 98.65%	测试集正确率: 98.94%
当前epoch: 2	[44800/60000 (75%)]	损失: 0.014779	训练集准确率: 98.66%	测试集正确率: 98.97%
当前epoch: 2	[51200/60000 (85%)]	损失: 0.010532	训练集准确率: 98.71%	测试集正确率: 98.50%
当前epoch: 2	[57600/60000 (96%)]	损失: 0.076463	训练集准确率: 98.68%	测试集正确率: 98.79%

# 评分标准

准确率	基础分	加分项	加分
99.0%以上	6分	不使用框架	1分
96.0%-98.9%	5.5分	做了对比实验	0.3分
90.0%-95.9%	5分	增加其他指标	0.1分
80-89.9%	4分	其他	0.1分
70-79.9%	3分		
60-69.9%	2分		

备注：

- 准确率低于60%，基础分为0；
- 所有分数总和若超过6分，一律按照6分处理；
- 判断标准主要采用准确率，即精度。在精度指标以外可扩展其他指标，作为加分项。

# SVM小实验

助教：陈嘉言

指导教师：高 阳

[mf20330006@smail.nju.edu.cn](mailto:mf20330006@smail.nju.edu.cn)



# 作业要求

在给定的数据集上用SVM实现一个垃圾邮件分类器。[数据集下载](#)。SVM算法可采用一些工具库（如scikit-learn等）完成。

- 需要提交的内容包括但不限于：
  - 代码，关键部分代码需要注释；
  - PDF文档，阐述SVM的基本原理，包含详细的步骤说明、运行结果和截图，给出精确率和召回率。
- 数据集说明：
  - 邮件来自Enron-Spam[语料库](#)
  - 上面的链接里包含了原始数据集和预处理后的数据集，本次实验可以直接使用预处理后的数据集。其中含有33716封邮件，分为6个文件夹，每个文件夹中有ham和spam子文件夹分别存放正常邮件与垃圾邮件。
  - 自己切分训练集和测试集设计完成实验。
  - 有兴趣的同学也可以尝试一下文本预处理的过程。
- 推荐阅读：[这里](#)提供了一个比较基础的实验参考教程。

# 处理过程（样例）

准备实验环境。推荐大家按照以下步骤（已经装好环境的同学可以跳过这一页）进行：

- [安装Anaconda](#)。

conda是一个Python环境管理工具，可以为你的项目建立一个纯净的Python环境，不受电脑里可能有多个Python版本的影响，[链接](#)。

- 在建立好的 conda 环境里安装 Jupyter Notebook（这里我的环境名字是PY3）。

```
$(PY3) pip install jupyter
```

同样也安装需要的其他库numpy、scikit-learn、pandas、matplotlib。

- 在项目目录运行jupyter。

```
$(PY3) jupyter notebook
```

# 处理过程（样例）

- Step 1: 数据预处理

去除无用字符，词干提取和词形还原（这些数据集里都已经完成）。

- Step2: 读取文本内容

拆分训练和测试数据集后，从文件夹中读取邮件的正文部分。

```
def extract_text(mail_dir):
    files = []
    for f in os.listdir(mail_dir):
        d = os.path.join(mail_dir, f)
        if os.path.isdir(d):
            files += [os.path.join(d, "ham", fi) for fi in os.listdir(os.path.join(d, "ham")) if fi.endswith('.txt')]

    for f in os.listdir(mail_dir):
        d = os.path.join(mail_dir, f)
        if os.path.isdir(d):
            files += [os.path.join(d, "spam", fi) for fi in os.listdir(os.path.join(d, "spam")) if fi.endswith('.txt')]

    text_matrix = np.ndarray((len(files)), dtype=object)
    docID = 0
    for fil in files:
        with open(fil, 'r', errors="ignore") as fi:
            next(fi) # skip subject line
            data = fi.read().replace('\n', ' ')
            text_matrix[docID] = data
            docID += 1

    return text_matrix
```

# 处理过程（样例）

- Step 3: 特征提取

将文本内容转化成特征，这里我采用TF-IDF方法来计算词的权重，大家也可以尝试不同的方法。

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.svm import SVC, NuSVC, LinearSVC
import numpy as np
import os

train_dir = '/Users/dora/workspace/svm/train/'
train_matrix = extract_text(train_dir)

count_v1 = CountVectorizer(stop_words = 'english', max_df = 0.5, decode_error = 'ignore', binary = True)
counts_train = count_v1.fit_transform(train_matrix)
tfidftransformer = TfidfTransformer()
tfidf_train = tfidftransformer.fit(counts_train).transform(counts_train)
```

- Step4: 训练分类器

使用scikit-learn可以很简单地构建一个SVM模型，这里使用了默认参数。

```
#训练
model = LinearSVC()
model.fit(tfidf_train, train_labels)
```

# 处理过程（样例）

- Step5: 测试和评估

在测试集上验证我们的模型，测试数据提取特征时共用了训练集的词汇表，这里评估时计算了混淆矩阵、精确率和召回率。

```
1 from sklearn.metrics import confusion_matrix, precision_score, recall_score
2 import pandas as pd
3
4
5 test_dir = '/Users/dora/workspace/svm/test/'
6 test_matrix = extract_text(test_dir)
7 print(test_matrix.shape)
8
9 count_v2 = CountVectorizer(vocabulary=count_v1.vocabulary_, stop_words = 'english', max_df = 0.5, decode_error = 'ignore')
10 counts_test = count_v2.fit_transform(test_matrix)
11 tfidf_test = tfidftransformer.fit(counts_test).transform(counts_test)
12
13 test_labels = np.zeros(12718)
14 test_labels[6315:12718] = 1
15 result = model.predict(tfidf_test)
16
17 cm = pd.DataFrame(
18     confusion_matrix(test_labels, result),
19     index=['non-spam', 'spam'],
20     columns=['non-spam', 'spam']
21 )
22 print(cm)
23
24 print("precision score:", precision_score(test_labels, result))
25 print("recall score:", recall_score(test_labels, result))
```

```
(12718,)
      non-spam  spam
non-spam     6010   305
spam           57  6346
precision score: 0.9541422342504886
recall score: 0.9910979228486647
```

# 评分标准

- 本次小实验设计希望同学们学习并掌握SVM原理的同时进行代码实现，因此评分分数设计为：
  - 完成作业要求，提交代码（3`）。
  - 按要求完成报告
    - 阐述SVM算法原理，对实验步骤进行阐述（1`）；
    - 展示实验结果，需要给出精确率和召回率，其他评估手段也可以加入到实验报告中。（2`）；
  - 注：所有对结果有帮助的步骤或尝试都可以作为加分项。

# RL小作业

助教：王健琦

指导教师：高 阳

[wangjianqi@smail.nju.edu.cn](mailto:wangjianqi@smail.nju.edu.cn)

# Demo with Tabular Q-Learning

## Q-learning 解决寻宝游戏

强化学习 (Reinforcement Learning, 简称RL) 是机器学习中的一个领域, 强调如何基于环境而行动, 以取得最大化的预期利益。

Q-learning 算法是强化学习领域著名算法之一。在有限的动作和状态空间内, 他通过感知状态(state)的“奖励和惩罚” (reward), 学习如何选择下一步“动作” (action)。

Tabular Q-learning 算法是利用动作值函数(action-value function), 来学习一个Q-table, 指导智能体(agent)选择动作。Q 函数会根据输入的状态和动作, 返回在该状态下执行该动作的未来奖励期望。在我们探索环境之前, Q-table 会给出相同的任意的设定值。随着对环境的持续探索, 这个 Q-table 会通过迭代地使用 Bellman 方程更新 Q值来给出越来越好的近似。

在一些需要执行动作, 进行观察, 积累经验, 形成模型的现实任务场景中, 强化学习算法展现出决策与博弈上的强大效果。



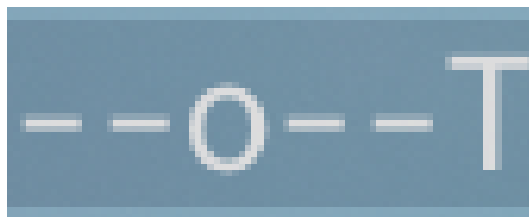
# Demo with Tabular Q-Learning

## Q-learning 解决寻宝游戏

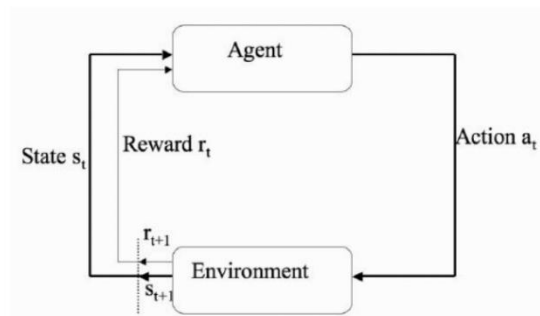
我们考虑这样一个游戏场景。假设在一维空间中，我们使用一个探索者左右移动寻找宝藏。假定探索者每次只能水平移动一步，同时移动到规定位置即找到宝藏，游戏结束。要求利用Q-learning 算法完成智能体寻宝的过程。

(注：建议对于实验场景和RL相关概念不清楚，先阅读本节PPT附录的参考资料)

游戏场景设计如下：



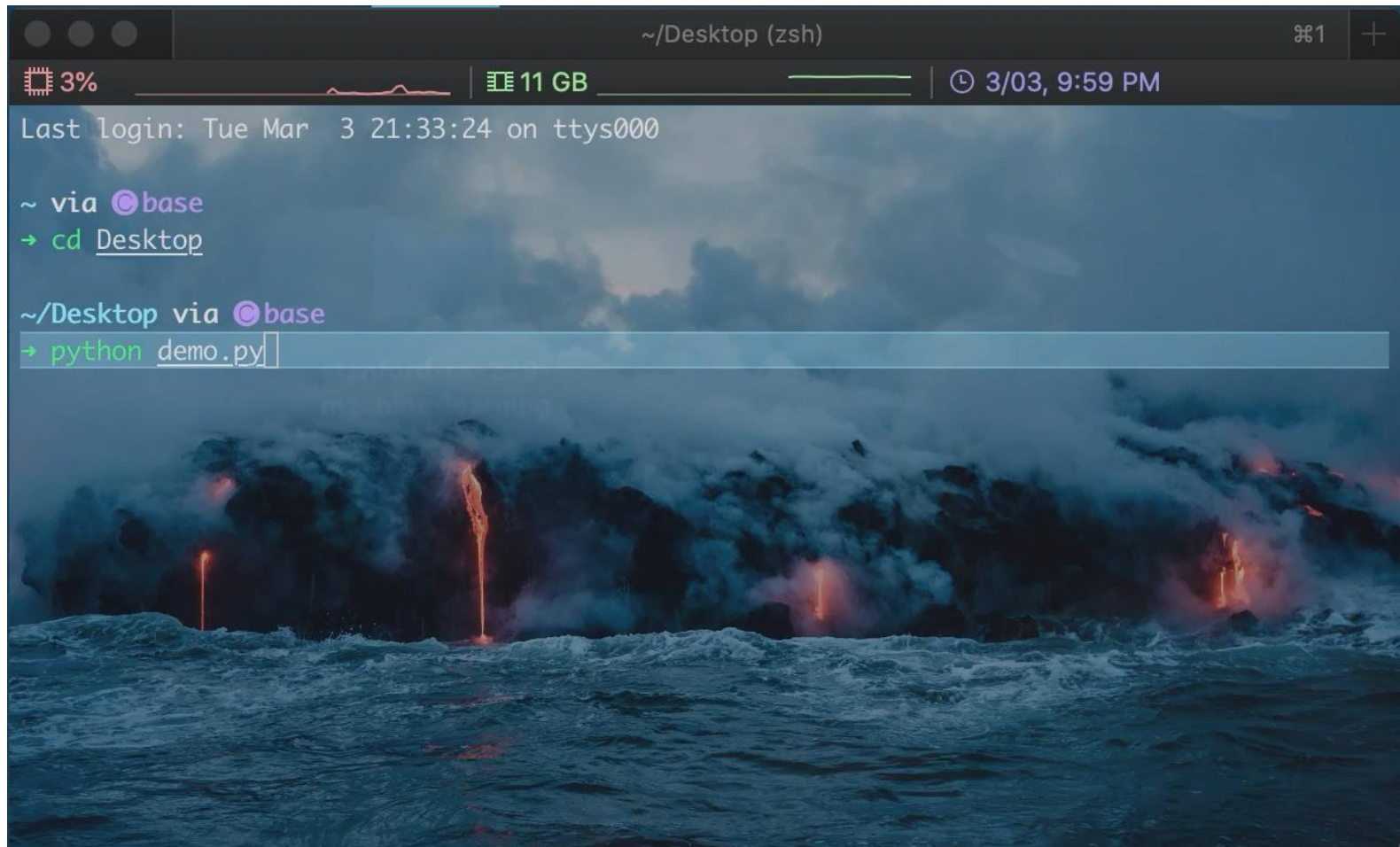
- O 表示当前智能体的位置；
- T表示宝藏位置；
- 起点与场景长度不做要求。



# Demo with Tabular Q-Learning

Q-learning 解决寻宝游戏

- 游戏过程:



# Demo with Tabular Q-Learning

Q-learning 解决寻宝游戏

- 实验要求:

提交完成上述算法设计和代码，报告内容包括但不限于：

- 伪代码描述Q-learning 的过程；
- 结合代码描述：
  - 参数，表格设计；
  - 描述  $\epsilon$ -greedy 策略选择过程；
  - 描述 Q值更新过程；
- 提交实验结果，输出学到的Q表。

# Demo with Tabular Q-Learning

Q-learning 解决寻宝游戏

- 参考资料:

- 《机器学习算法视角第二版》

- 11.2 状态和行动空间，奖赏函数，折扣，策略；

- 11.4 Q-learning 推导；

- [莫烦RL](#)

- Q-learning 思维；

- Demo；

- Sutton 《Reinforcement Learning: An Introduction》

- [本书中文版](#)；

- 对于本次实验有兴趣的同学可以关注Q-learning 实现的Tic-Tac-Toe例子，编写代码完成训练与博弈的过程，实现人机对抗；

- 本书配套代码推荐: [代码](#)。

# Demo with Tabular Q-Learning

Q-learning 解决寻宝游戏

- 评分标准:

本次RL小实验设计主要为了解Q-learning过程，所以侧重于通过完成报告阐述对该算法了解，因此各分数设计为：

- 利用Q-learning实现小游戏，提交代码(1`);
- 报告中描述Q-learning伪代码(1`);
- 报告中包含：Q表，策略选择，Q值更新(3`);
- 报告中包含分析Q表(1`)。

# 决策树作业

助教：陈嘉言

指导教师：高 阳

[mf20330006@smail.nju.edu.cn](mailto:mf20330006@smail.nju.edu.cn)

# 作业要求

给定隐形眼镜小量数据集，构造决策树预测患者佩戴隐形眼镜类型，并通过Matplotlib绘制树图形。

- 提交完成上述算法设计和代码，报告内容包括但不限于：
  - 阐述决策树算法的原理；
  - 给出决策树部分的核心代码；
  - 提交实验结果，展示树图形。
- 数据集说明：
  - 隐形眼镜数据集中包含患者眼部状况的观察条件以及医生推荐的隐形眼镜的类型。隐形眼镜标签包括：hard（硬材质）、soft（软材质）、不适合佩戴隐形眼镜（no lenses）；
  - 从第一列开始，特征分别为：age、prescript、astigmatic、tearRate。

1	young	myope	no	reduced	no lenses
2	young	myope	no	normal	soft
3	young	myope	yes	reduced	no lenses
4	young	myope	yes	normal	hard
5	young	hyper	no	reduced	no lenses
6	young	hyper	no	normal	soft
7	young	hyper	yes	reduced	no lenses
8	young	hyper	yes	normal	hard
9	pre myope	no	reduced	no lenses	
10	pre myope	no	normal	soft	
11	pre myope	yes	reduced	no lenses	
12	pre myope	yes	normal	hard	
13	pre hyper	no	reduced	no lenses	
14	pre hyper	no	normal	soft	
15	pre hyper	yes	reduced	no lenses	
16	pre hyper	yes	normal	no lenses	
17	presbyopic	myope	no	reduced	no lenses
18	presbyopic	myope	no	normal	no lenses
19	presbyopic	myope	yes	reduced	no lenses
20	presbyopic	myope	yes	normal	hard
21	presbyopic	hyper	no	reduced	no lenses
22	presbyopic	hyper	no	normal	soft
23	presbyopic	hyper	yes	reduced	no lenses
24	presbyopic	hyper	yes	normal	no lenses

# 处理过程

- 数据预处理，得到原始数据集；
- 构建划分数数据集函数；
- 构建计算信息增益(香农熵)函数；
- 选择最优的数据集划分方式；
- 创建决策树；
- 绘制树图形。



# 处理过程（样例）

1. 原始数据集，判断是不是属于鱼类：

	不浮出水面 是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

2. 处理之后的数据集为：

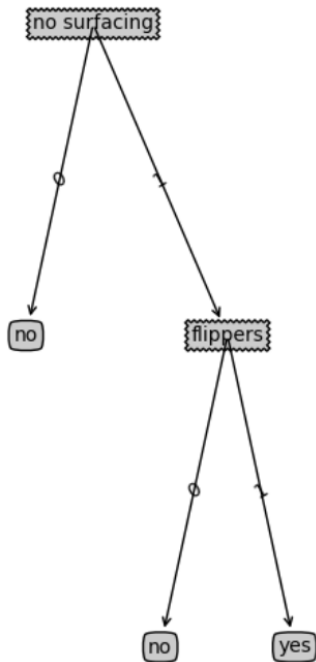
```
dataSet = [  
    [1, 1, 'yes'],  
    [1, 1, 'yes'],  
    [1, 0, 'no'],  
    [0, 1, 'no'],  
    [0, 1, 'no'],  
]  
labels = ['no surfacing', 'flippers']
```

# 处理过程（样例）

3. 生成决策树：

```
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
```

4. 绘制树图形：



# 参考资料

- 教材《机器学习 算法视角 第二版》
  - 第十二章，决策树学习
- [机器学习之-常见决策树算法\(ID3、C4.5、CART\)](#)
- [绘制树图形代码](#) (参考)

# 评分标准

- 本次小实验设计希望同学们学习并掌握基本决策树原理的同时进行代码实现，因此评分分数设计为：
  - 完成作业要求，提交代码（3`）。
  - 按要求完成报告
    - 阐述决策树算法原理（1`）；
    - 决策树部分核心代码（1`）；
    - 展示实验结果（1`）。

# Kmeans作业

助教: 杨雪松

指导教师: 高 阳

[yangxuesong@smail.nju.edu.cn](mailto:yangxuesong@smail.nju.edu.cn)

# 作业要求

实现图形界面演示Kmeans聚类过程。每次随机生成100个坐标点，随机选取k个中心点作为聚类中心，进行迭代，能够在界面中显示出聚类的变化（比如：不同簇坐标点颜色不同等）直到聚类中心位置不再变化或达到设定的迭代次数，得出各聚类中心点的坐标以及聚类结果。

- 提交完成上述算法设计和代码，报告内容包括但不限于：
  - 阐述Kmeans算法的原理；
  - 给出实现Kmeans算法部分的核心代码；
  - 提交实验结果，展示部分聚类状态并讲解。
- 参考资料：
  - [KMEANS算法以及代码讲解](#)；
  - [机器学习算法之聚类算法](#)。

# 评分标准

- 掌握Kmeans聚类原理，完成作业要求，提交代码和报告（3`）；
- 聚类变化过程中展示中心坐标变化（1`）；
- 报告中对核心代码进行讲解（1`）；
- 展示实验结果（1`）。