1. Neural Networks: MNIST image classfication

(a)(b)(c)前向传播部分很简单，反向传播部分的推导请参考CS231作业1，其中如下代码

```
#计算第二层梯度
p3 = np.exp(scores) / p2.reshape(-1, 1)
p3[np.arange(N), y] -= 1
dW2 = X1.T.dot(p3) / N + 2 * reg * W2
db2 = np.sum(p3, axis=0) / N
grads["W2"] = dW2
grads["b2"] = db2
```

对应于这里的

```
N2, D2 = y.shape
t2 = y - labels
db2 = np.sum(t2, axis=0) / N2
dW2 = h.T.dot(t2) / N2
if Lambda != 0:
    dW2 += 2 * Lambda * W2
dX2 = t2.dot(W2.T)
```

注意CS231的习题和这里最大的不同为前者的y是数值，即$0, 1, \ldots, 9$的形式，而后者的则是one-hot的形式，所以以下两句代码作用相同：

```
#CS231
p3[np.arange(N), y] -= 1

#CS229
t2 = y - labels
```

在第一层中，只要知道sigmoid函数的导数为

$$\sigma'(s) = \sigma(s)(1 - \sigma(s))$$

即可，这部分的代码为：

```
#第一层梯度
N2, D2 = data.shape
t1 = h * (1 - h)
dW1 = data.T.dot(t1 * dX2) / N2
if Lambda != 0:
    dW1 += 2 * Lambda * W1
db1 = np.sum(t1, axis=0) / N2
```

完整的代码见my_nn_starter.py文件。

2. EM Convergence

首先回顾定义以及不等式:

$$
\begin{aligned}
Q_i(z^{(i)}) &= p(z^{(i)}|x^{(i)};\theta) \\
\theta &:= \arg\max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \\
l(\theta^{(t+1)}) &\geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \\
&\geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \\
&= l(\theta^{(t)})
\end{aligned}
$$

如果θ最终收敛到θ'，那么

$$
\begin{aligned}
Q_i(z^{(i)}) &= p(z^{(i)}|x^{(i)};\theta') \\
\theta' &:= \arg\max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}
\end{aligned}
$$

所以

$$
\begin{aligned}
0 =&\nabla_\theta\Big(\sum_i\sum_{z^{(i)}}Q_i(z^{(i)})\log\frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\Big)\Big|_{\theta=\theta'}\\
&=\sum_i\sum_{z^{(i)}}Q_i(z^{(i)})\nabla_\theta(\log p(x^{(i)},z^{(i)};\theta))|_{\theta=\theta'}\\
&=\sum_i\sum_{z^{(i)}}p(z^{(i)}|x^{(i)};\theta')\frac{\nabla_\theta p(x^{(i)},z^{(i)};\theta))|_{\theta=\theta'}}{p(x^{(i)},z^{(i)};\theta')}\\
&=\sum_i\sum_{z^{(i)}}\frac{p(x^{(i)},z^{(i)};\theta')}{p(x^{(i)};\theta')}\frac{\nabla_\theta p(x^{(i)},z^{(i)};\theta))|_{\theta=\theta'}}{p(x^{(i)},z^{(i)};\theta')}\\
&=\sum_i\sum_{z^{(i)}}\frac{\nabla_\theta p(x^{(i)},z^{(i)};\theta))|_{\theta=\theta'}}{p(x^{(i)};\theta')}\\
&=\sum_i\frac{\nabla_\theta\sum_{z^{(i)}}p(x^{(i)},z^{(i)};\theta))|_{\theta=\theta'}}{p(x^{(i)};\theta')}\\
&=\sum_i\frac{\nabla_\theta p(x^{(i)};\theta))|_{\theta=\theta'}}{p(x^{(i)};\theta')}\\
&=\sum_i\nabla_\theta(\log p(x^{(i)};\theta))|_{\theta=\theta'}\\
&=\nabla_\theta\Big(\sum_i\log p(x^{(i)};\theta)\Big)|_{\theta=\theta'}\\
&=\nabla_\theta l(\theta)|_{\theta=\theta'}
\end{aligned}
$$

3. PCA

首先计算$f_u(x)$，设

$$
v=\alpha u
$$

带入原式可得

$$
\begin{aligned}
||x-\alpha u||^2&=(x-\alpha u)^T(x-\alpha u)\\
&=x^Tx-2\alpha u^Tx+\alpha^2u^Tu
\end{aligned}
$$

关于α求导，并令导数为0可得

$$
\begin{aligned}
2u^Tu\alpha&=2u^Tx\\
\alpha&=\frac{u^Tx}{u^Tu}=u^Tx
\end{aligned}
$$

所以

$$
\begin{aligned}
f_u(x) &= (u^Tx)u \\
||x-(u^Tx)u||^2 &= (x^Tx - 2\alpha u^Tx + \alpha^2 u^Tu)|_{\alpha=u^Tx} \\
&= x^Tx - 2(u^Tx)^2 + (u^Tx)^2 \\
&= x^Tx - (u^Tx)^2
\end{aligned}
$$

所以我们的目标为求解如下问题

$$
\begin{aligned}
&\min_u \sum_{i=1}^m \Big((x^{(i)})^Tx^{(i)} - (u^Tx^{(i)})^2\Big) \\
&\text{subject to } u^Tu = 1
\end{aligned}
$$

记

$$
X = \begin{bmatrix} —(x^{(1)})^T— \\ —(x^{(2)})^T— \\ \vdots \\ —(x^{(m)})^T— \end{bmatrix}
$$

那么

$$
Xu = \begin{bmatrix} —(x^{(1)})^Tu— \\ —(x^{(2)})^Tu— \\ \vdots \\ —(x^{(m)})^Tu— \end{bmatrix}
$$

所以

$$
\begin{aligned}
\sum_{i=1}^m \Big((x^{(i)})^Tx^{(i)} - (u^Tx^{(i)})^2\Big) &= X^TX - (Xu)^T(Xu) \\
&= X^TX - u^TX^TXu
\end{aligned}
$$

注意前一项是常数，所以原问题可以化为

$$
\begin{aligned}
&\max_u u^TX^TXu \\
&\text{subject to } u^Tu = 1
\end{aligned}
$$

因此可以构造拉格朗日乘子：

$$
L(\lambda,\alpha) = u^TX^TXu - \lambda(u^Tu-1)
$$

求梯度可得

$$
\nabla_u L(\lambda,\alpha) = 2X^TXu - 2\lambda u
$$

令上式为0，那么

$$X^TXu=\lambda u \tag{1}$$

这说明u是X^TX的特征值，并且

$$u^TX^TXu=\lambda u^Tu=\lambda$$

所以u是X^TX最大特征值对应的特征向量，即第一主成分。

4. Independent components analysis

报错参考：安装sounddevice报错的解决方案。

这里更新公式为：

$$W:=W+\alpha\left(\begin{bmatrix}1-2g(w_1^Tx^{(i)})\\1-2g(w_2^Tx^{(i)})\\ \ldots\\1-2g(w_n^Tx^{(i)})\end{bmatrix}x^{(i)T}+(W^T)^{-1}\right)$$

其中

$$W=\begin{bmatrix}-w_1^T-\\ \ldots\\-w_n^T-\end{bmatrix}$$

注意我们有

$$s_j^{(i)}=w_j^Tx^{(i)}$$

所以恢复数据的公式为

$$s^{(i)}=Wx^{(i)},S=\begin{bmatrix}-(s^{(1)})^T-\\ \ldots\\-(s^{(n)})^T-\end{bmatrix}=\begin{bmatrix}-(x^{(1)})^TW^T-\\ \ldots\\-(x^{(n)})^TW^T-\end{bmatrix}=XW^T$$

代码如下：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 29 13:49:57 2019

@author: qinzhen
"""

### Independent Components Analysis
###
### This program requires a working installation of:
###
### On Mac:
```

```python
###     1. portaudio: On Mac: brew install portaudio
###     2. sounddevice: pip install sounddevice
###
### On windows:
###      pip install pyaudio sounddevice
###

import sounddevice as sd
import numpy as np

Fs = 11025

def normalize(dat):
    return 0.99 * dat / np.max(np.abs(dat))

def load_data():
    mix = np.loadtxt('mix.dat')
    return mix

def play(vec):
    sd.play(vec, Fs, blocking=True)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def unmixer(X):
    M, N = X.shape
    W = np.eye(N)

    anneal = [0.1, 0.1, 0.1, 0.05, 0.05, 0.05, 0.02, 0.02, 0.01, 0.01,
              0.005, 0.005, 0.002, 0.002, 0.001, 0.001]
    print('Separating tracks ...')
    ######## Your code here ##########

    for alpha in anneal:
        #打乱数据
        np.random.permutation(X)
        for i in range(M):
            x = X[i, :].reshape(-1, 1)
            WX = W.dot(x)
            grad = (1 - 2 * sigmoid(WX)).dot(x.T) + np.linalg.inv(W.T)
            W += alpha * grad
    ###################################
    return W

def unmix(X, W):
    S = np.zeros(X.shape)

    ######### Your code here ##########
    S = X.dot(W.T)
    ##################################
    return S
```

```python
X = normalize(load_data())

for i in range(X.shape[1]):
    print('Playing mixed track %d' % i)
    play(X[:, i])

W = unmixer(X)
S = normalize(unmix(X, W))

for i in range(S.shape[1]):
    print('Playing separated track %d' % i)
    play(S[:, i])
```

5. Markov decision processes

(a)$\forall \pi$，定义

$$B^\pi(V)(s) = R(s) + \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V(s')$$

那么

$$\begin{aligned}B^\pi(V_1)(s) &= R(s) + \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V_1(s')\\ B^\pi(V_2)(s) &= R(s) + \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V_2(s')\end{aligned}$$

所以

$$\begin{aligned}|B^\pi(V_1)(s) - B^\pi(V_2)(s)| &= |\gamma\sum_{s'\in S} P_{s\pi(s)}(s')V_1(s') - \gamma\sum_{s'\in S} P_{s\pi(s)}(s')V_2(s')|\\ &= \gamma|\sum_{s'\in S} P_{s\pi(s)}(s')(V_1(s') - V_2(s'))|\\ &\le \gamma\sum_{s'\in S} P_{s\pi(s)}(s')|V_1(s') - V_2(s')|\\ &\le \gamma\sum_{s'\in S} P_{s\pi(s)}(s')\|V_1 - V_2\|_\infty\\ &= \gamma\|V_1 - V_2\|_\infty\end{aligned}$$

因此

$$\|B^\pi(V_1) - B^\pi(V_2)\|_\infty = \max_{s\in\mathcal{S}} |B^\pi(V_1)(s) - B^\pi(V_2)(s)| \le \gamma\|V_1 - V_2\|_\infty$$

注意由定义我们有

$$
\begin{aligned}
B(V_1)(s) &= R(s) + \gamma \max_{a\in A} \sum_{s'\in S} P_{sa}(s')V_1(s') \\
B(V_2)(s) &= R(s) + \gamma \max_{a\in A} \sum_{s'\in S} P_{sa}(s')V_2(s')
\end{aligned}
$$

由对称性，不妨设

$$
B(V_1)(s) \ge B(V_2)(s)
$$

如果记

$$
\pi_1 = \arg\max_a \sum_{s'\in S} P_{sa}(s')V_1(s')
$$

从而

$$
\begin{aligned}
B^{\pi_1}(V_1)(s) = B(V_1)(s) \\
B^{\pi_1}(V_2)(s) \le B(V_2)(s)
\end{aligned}
$$

因此

$$
\begin{gathered}
0 \le B(V_1)(s) - B(V_2)(s) \le B^{\pi_1}(V_1)(s) - B^{\pi_1}(V_2)(s) \\
|B(V_1)(s) - B(V_2)(s)| \le |B^{\pi_1}(V_1)(s) - B^{\pi_1}(V_2)(s)| \le \|B^{\pi}(V_1) - B^{\pi}(V_2)\|_\infty
\end{gathered}
$$

类似的，如果

$$
B(V_1)(s) \ge B(V_2)(s)
$$

那么记

$$
\pi_2 = \arg\max_a \sum_{s'\in S} P_{sa}(s')V_1(s')
$$

依然可得

$$
|B(V_1)(s) - B(V_2)(s)| \le |B^{\pi_2}(V_1)(s) - B^{\pi_2}(V_2)(s)| \le \|B^{\pi}(V_1) - B^{\pi}(V_2)\|_\infty
$$

所以无论那种情形，我们都有

$$
|B(V_1)(s) - B(V_2)(s)| \le \|B^{\pi}(V_1) - B^{\pi}(V_2)\|_\infty
$$

对左边关于s取最大值可得

$$
\begin{aligned}
\|B(V_1) - B(V_2)\|_\infty &= \max_s |B(V_1)(s) - B(V_2)(s)| \\
&\le \|B^{\pi}(V_1) - B^{\pi}(V_2)\|_\infty \\
&\le \gamma \|V_1 - V_2\|_\infty
\end{aligned}
$$

(b)如果存在V使得

$$
V = B(V)
$$

那么任取同样满足上述条件的V_0，即

$$V_0 = B(V_0)$$

那么

$$\begin{aligned} \|V_0 - V\|_\infty &= \|B(V_0) - B(V)\|_\infty \\ &\le \gamma \|V_0 - V\|_\infty \\ &\ldots \\ \|V_0 - V\|_\infty &\le \gamma^n \|V_0 - V\|_\infty \end{aligned}$$

因此

$$\|V_0 - V\|_\infty \le \lim_{n\to\infty} \gamma^n \|V_0 - V\|_\infty = 0$$

所以

$$V_0 = V$$

6.Reinforcement Learning: The inverted pendulum

首先是初始化：

```
#价值函数
value_function = np.random.uniform(0, 0.1, NUM_STATES)
#计数
tran_cnt = np.zeros((NUM_STATES, NUM_STATES, NUM_ACTIONS))
#概率
tran_prob = np.ones((NUM_STATES, NUM_STATES, NUM_ACTIONS)) / NUM_STATES
#记录奖励
reward_value = np.zeros(NUM_STATES)
#记录奖励总数
reward_cnt = np.zeros(NUM_STATES)
#奖励
state_reward = np.zeros(NUM_STATES)
```

第一步初始化价值函数为均匀分布，第二步记录用行动a从状态b到状态c的次数，第三步初始化状态转移概率P_{sa}，第四步记录累计奖励，第五步统计奖励次数，第六步是初始化奖励函数$R(s)$。

然后是对行动作出选择：

```
#期望收益，比较其大小
s0 = np.sum(tran_prob[state, :, 0] * value_function)
s1 = np.sum(tran_prob[state, :, 1] * value_function)
if s0 > s1:
    action = 0
elif s0 < s1:
    action = 1
else:
    action = np.random.randint(0, 2)
```

上述代码第一步是计算

$$\sum_{s' \in S} P_{s\pi(s)}(s')V^{\pi}(s')$$

根据这一项选择采取哪种行动。

其次是每次行动后对统计量进行更新:

```
tran_cnt[state, new_state, action] += 1
reward_value[new_state] += R
reward_cnt[new_state] += 1
```

接着在到达最终状态后，更新价值函数，状态转移概率:

```
#统计在某个状态采取某个动作的次数
for i in range(NUM_ACTIONS):
    for j in range(NUM_STATES):
        #在状态j采取行动i的总数
        num = np.sum(tran_cnt[j, :, i])
        if num > 0:
             tran_prob[j, :, i] = tran_cnt[j, :, i] / num
#更新奖励
state_reward[reward_cnt>0] = reward_value[reward_cnt>0] / reward_cnt[reward_cnt>0]
```

最后一步利用值迭代更新:

- 对每个状态s，初始化$V(s) := 0$
- 重复直到收敛{
    - 对每个状态，更新$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V(s')$

  }

代码如下

```
iterations = 0
while True:
    #值迭代更新后的值
    v0 = GAMMA * tran_prob[:, :, 0].dot(value_function) + state_reward
    v1 = GAMMA * tran_prob[:, :, 1].dot(value_function) + state_reward
    v = np.c_[v0, v1]
    #取最大值
    value_function_new = np.max(v, axis=1)
    #计算更新幅度
    delta = np.linalg.norm(value_function_new - value_function)
    #更新价值函数
    value_function = np.copy(value_function_new)
    iterations += 1
    #更新幅度变小，则停止循环
    if delta < TOLERANCE:
        break

#更新一次收敛的计数
if iterations == 1:
```

```
    consecutive_no_learning_trials += 1
else:
    consecutive_no_learning_trials = 0
```