Problem 1: Markov Decision Processes

**1.a**

Each of 4 states have 4 actions, so there are totally 4\*4\*4\*4 = **256** deterministic policies.

**1.b** Please take source code 1.b.py and 1.b.2.py for reference.

Optimal Value function for four initial state: [4.111111111111112, 3.8888888888888893, 3.6111111111111116, 3.6111111111111116]

Optimal Policy for four initial state: **[a2, a3, a2, a2]**

There is a unique optimal deterministic policy, that always select action **[a2, a3, a2, a2]**

It is unique because there are specific one action that maximize the Bellman Equations.

**1.c** Please take source code 1.c.py for reference.

Run program 10 times, it output different results. The optimal stochastic policy is not unique.

It is not unique because, when chose action from some random distribution, the action will never concentrate to unique. So, the policy is not unique.

**1.d** Please take source code 1.d.py for reference.

When change λ from 0.8 to 0.01, the optimal policy’s result will not change. But the training progress will converge much quicker.

Problem 2: Poisson Maximum Likelihood Estimation

1. The likelihood function:

L(λ; x1, …, xm) =

For convenience, get the log-likelihood function:

ln L(λ; x1, …, xm) = =

= -mλ - +

Take derivation of

ln L(λ; x1, …, xm) = -m +

Let derivation equals to 0, we get

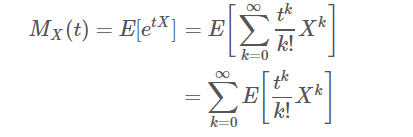
1. To gurantee that MLE <= + ε with probalility at least 1-exp(-5), it also means MLE >= + ε with probalility at most exp(-5), this is:

Pr(MLE - >= ε) <= exp(-5)

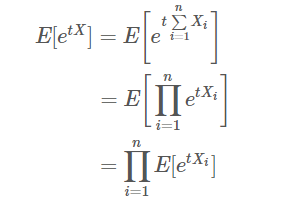
Let Y = MLE – and a = ε, then

Pr(MLE >= ε) <=

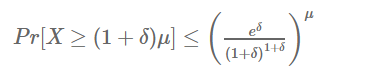
Use Taylor series, X = = 1/m, we have



When X = , we have



Use Chenoff Bound formula, X = = 1/m , μ = E(X), we have



When probability = exp(-5), let

2 exp(−2ε2m) = exp(-5)

**Then m = 5 / (2 ln2 ε2)**

So, the lower bound of the mumber of samples is **5 / (2 ln2 ε2)**

1. P()-/10 + 1 when < 10, P() = 0 when > 10, impossible for > 10

When < 10, The likelihood function:

L(λ; x1, …, xm) = P()

For convenience, get the log-likelihood function:

ln L(λ; x1, …, xm) = = + ln P()

= -mλ - + + ln(-/10 + 1)

Take derivation of

ln L(λ; x1, …, xm) = -m + +

Let derivation equals to 0, we get

/ 2m, b = 10m+1+,

Problem 3: Neural Networks

The network structure for perceptrons can be formed as three layers:

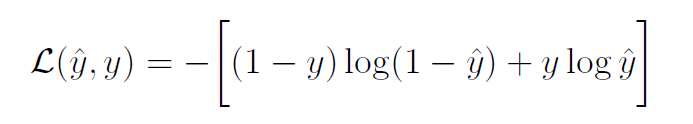
The **1 layer(input layer)** consist of 10 neurons consisting of input boolean value.

The **2 layer(hidden layer)** consist of 3 neurons. Each accept all inputs as first layer’s outputs. Setting w = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], b = 0. Neuron: when the sum of four inputs equals 0, output 1, otherwise 0. Neuron: when the sum of four inputs equals 4, output 1, otherwise 0. Neuron: when the sum of four inputs equals 8, output 1, otherwise 0.

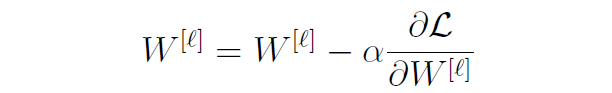
The **3 layer(output layer)** consist of 1 neuron accepting all input boolean value from second layer. Setting w = [1, 1, 1], b = 0, when the sum of three inputs equals 1, output 1 representing divided by 4, otherwise 0 representing not divided by 4.

For relu neural network, please reference source code 3.py,

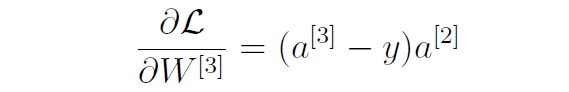
Define loss maximized loss function:

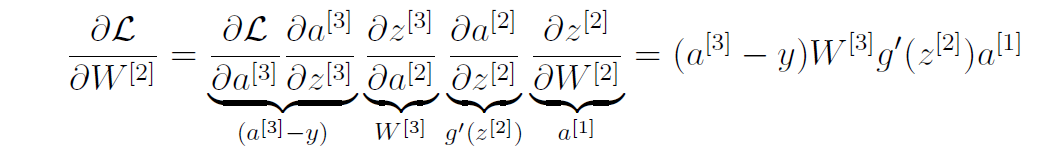


W update rule,



There are two layers need to update, differentiate seperately for both layer3 and layer2,





When I increase the size of training set, the learned weights converge more accurately to my perceptron solution as vary the size of the raning set from 100 to 10000.