Homework -10

Collaboration - Aakarsh

Resources - Class Notes

Problem 1:

Part a:

- If we look at polynomial weights algorithm the weight update equation is written as $W^{t+1}_i = W^t_i (1 - \varepsilon l^t_i)$. ε is the amount by which we change the weights in every iteration.
- It determines how slow or fast we reach convergence. If ε is too small we take longer to converge and if too large then we actually might miss the converging point and keep oscillating around it.
- Coming to the question if we set $\varepsilon = \ln(N)/T$ and substitute it in the average regret equation from notes viz avg_regret $\leq \varepsilon + \frac{ln(N)}{\varepsilon * T}$ we get avg_regret $\leq \frac{ln(N)}{T} + 1$
- If N is too large and T is small we might end up with a very high average regret which is not desirable.

Part b

- Again according the problem if we choose ε = 1 then the average regret equation changes to
  avg_regret $\leq \frac{ln(N)}{T} + 1$ if we substitute ε = 1.
  (Is it me or is this again same as part a. I want the points so here you go..)
- We can argue on the same lines as part 1. If there is a case where N is too large and T is small then the average regret is too high.
- We take bigger steps towards convergence and change weights by a big factor. Sometimes missing convergence and leading to oscillations around convergence point.

Part c

Interesting.. Looks like we know what action to play everytime to maximize utility

- If there is an action which has zero loss $W^{t+1}_i = W_i^t (1 - \varepsilon l_i^t)$ then it's weight doesn't get updated as l_i^t is always zero.
- We want to reduce weights of actions having higher loss towards zero.
- So the idea would be to set a high $\boldsymbol{\varepsilon = 0.99}$ very large and close to zero.
- In every iteration, actions with low loss don't get their weights changed much but actions with high losses get their weights reduced by a much bigger factor and eventually making them zero.
- After some iterations actions having high losses will have their weights close to 0 and action with zero loss would have highest weight making it the best candidate to be picked in every iteration to make average regret less and maximize utility.

Problem 2

Part a

```
import numpy as np
import math
u = np.array([[0,-1,1,-0.5],[1,0,-1,0.5],[-1,1,0,-0.5],[0.5,-0.5,0.5,0]])
```

```
print(u)
```

```
[[ 0.  -1.   1.  -0.5]
 [ 1.   0.  -1.   0.5]
 [-1.   1.   0.  -0.5]
 [ 0.5 -0.5  0.5  0. ]]
```

```
def probs_from_weights(weights):
    return weights/np.sum(weights)
```

Part b

```
def best_response(strategy,flag=0):
    temp = []
    temp.append(np.dot(strategy,-u[:,0]))
    temp.append(np.dot(strategy,-u[:,1]))
    temp.append(np.dot(strategy,-u[:,2]))
    temp.append(np.dot(strategy,-u[:,3]))
    if flag:
        return max(temp)
    return temp.index(max(temp))
```

Part c

```
def util_of_each_action(index):
    return u[:,index]
```

Part d

```
def convert_utils_to_losses(utilities):
    temp = []
    for i in utilities:
        if i == -1:
            temp.append(1)
        elif i == 1:
            temp.append(0)
        elif i<0:
            temp.append(0.5 + abs(i)/2)
        elif i>0:
            temp.append(0 + i/2)
        elif i==0:
            temp.append(0.5)
    return np.array(temp)
```

Part e

```python
def update_weights(weights,losses,epsilon):
    updated_weights = weights*(1-epsilon*losses)
    return updated_weights
```

Part f

We use the equation from notes ε = sqrt(ln(N)/T). Part of the reason for using this epsilon is we get to convergence in a smoother way as graph from part h explains. we also reason why setting epsilon to any other value won't be a good idea in Problem 1.

```
rock = []
paper = []
scissor = []
mithril = []


def compute_equlibrium(T,epsilon):
    weights = np.array([1,1,1,1])
    list_probs = []
    for i in range(0,T):
        probs = probs_from_weights(weights)
        if T == 50000:
            rock.append(probs[0])
            paper.append(probs[1])
            scissor.append(probs[2])
            mithril.append(probs[3])
        list_probs.append(probs)
        response = best_response(probs)
        utilities = util_of_each_action(response)
        losses = convert_utils_to_losses(utilities)
        weights = update_weights(weights,losses,epsilon)
    list_probs =  np.array(list_probs)
    return np.mean(list_probs,axis=0)
T = [10,100,1000,10000,100000,50000]

stratigies = []
for i in T:
    epsilon = math.sqrt(math.log(4)/i)
    stratigies.append(compute_equlibrium(i,epsilon))
```

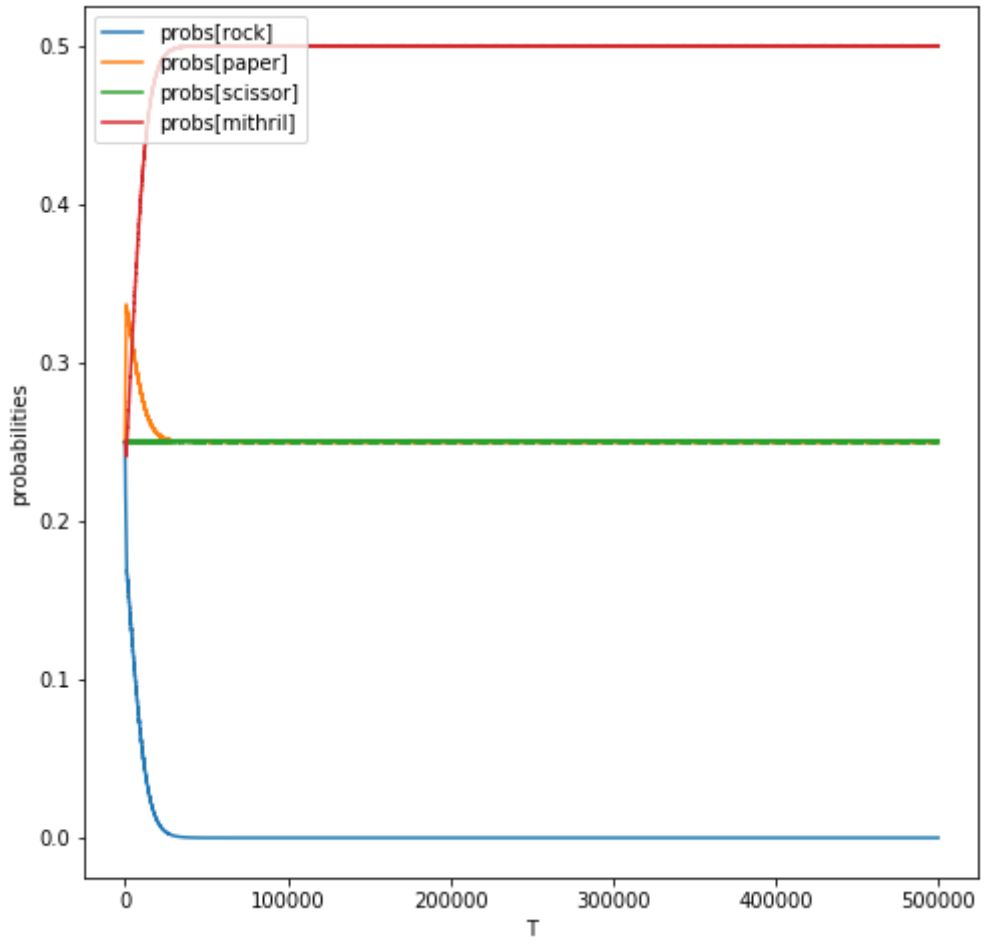
Part g

```
T = [10,100,1000,10000,100000,50000]

stratigies = []
for i in T:
    epsilon = math.sqrt(math.log(4)/i)
    stratigies.append(compute_equlibrium(i,epsilon))

utilities = []
for i in stratigies:
    utilities.append(best_response(i,1))

print(utilities)
```

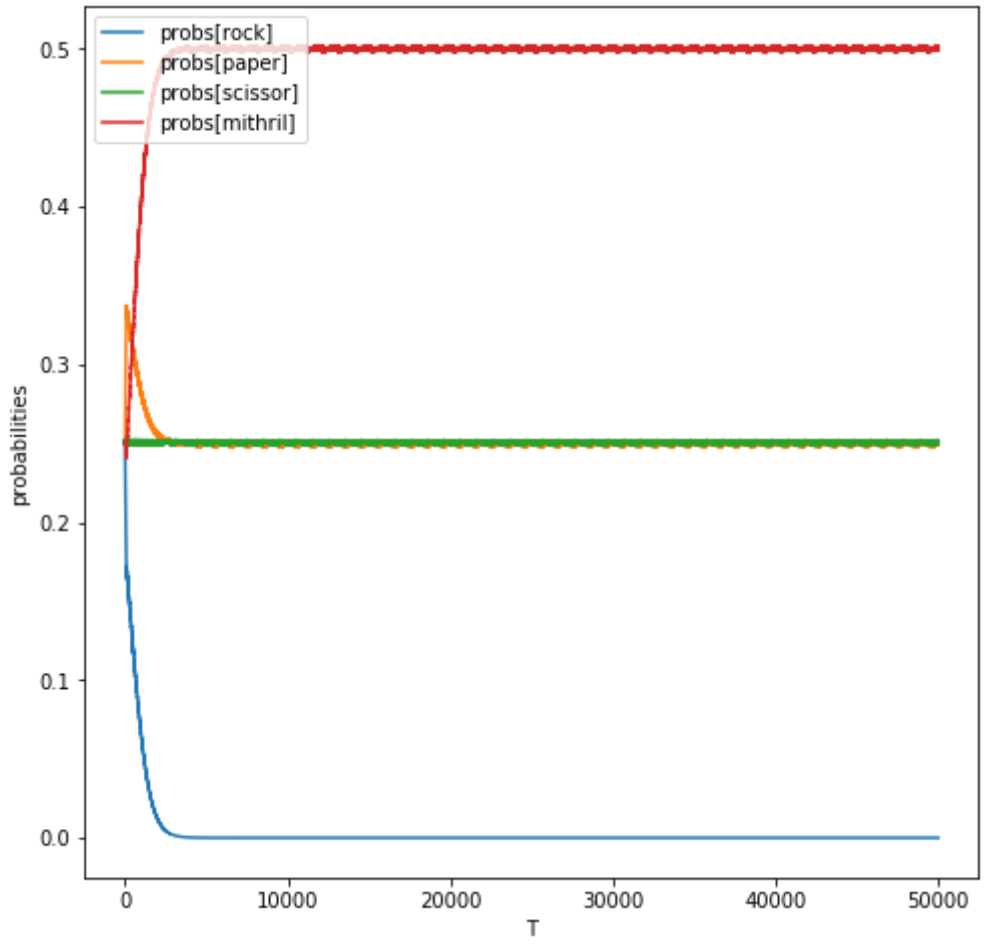
```
[0.07746730608387142, 0.04217632069583864, 0.019041629018897402, 0.0062153654030444455, 0.001974757645995376, 0.002790692944464
5456]
```

Part h

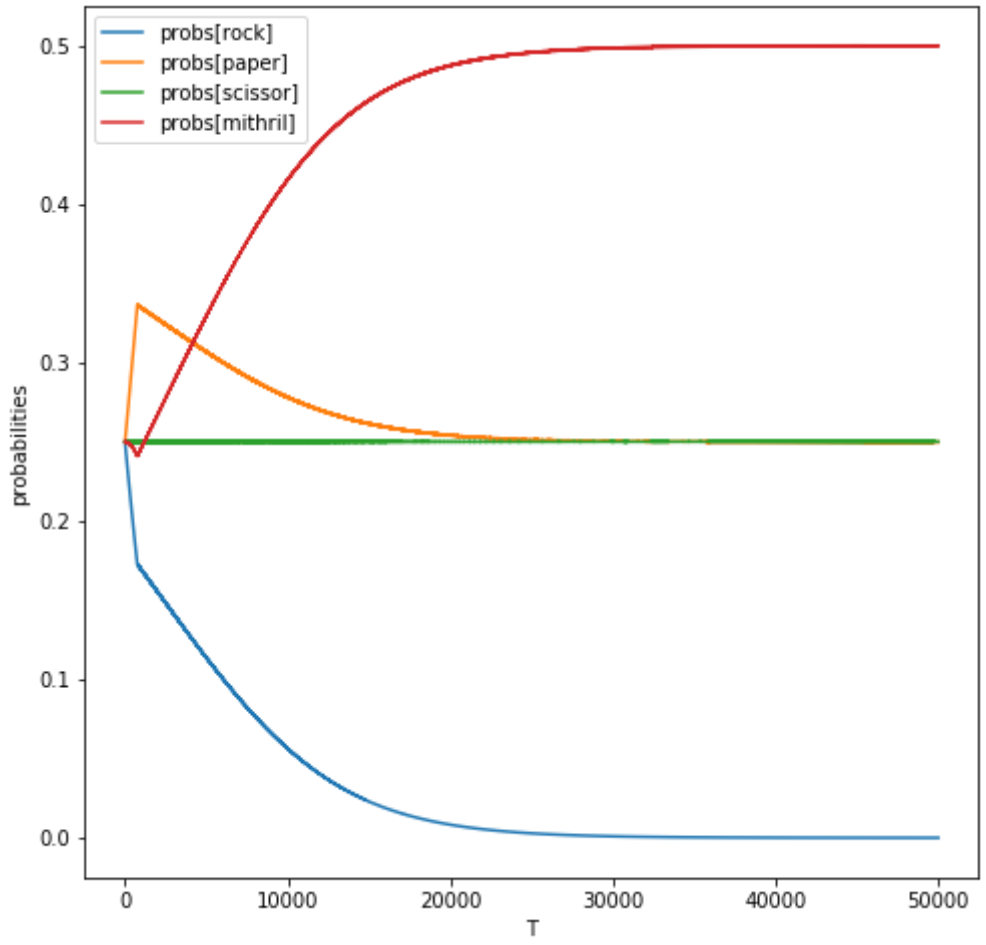
Normal epsilon



3*epsilon


probs[rock]
probs[paper]
probs[scissor]
probs[mithril]
probabilities
T
0.0
0.1
0.2
0.3
0.4
0.5
0
10000
20000
30000
40000
50000

epsilon/3.0



From the graphs we notice that over T iterations action Mithril converges and if we play it for every round we can expect to minimize loss and get better utility over choosing any other action if we played for all T rounds.

When we increase epsilon by a factor of 3 we see that convergence is not as smooth as graph 1 because we are taking bigger steps to convergence. We increase weights which gives less loss and more utility and decrease the

weights which gives more loss and less utility by a bigger stride. Hence the unevenness.

When Epsilon is divided by 3 the convergence is too slow hence we don’t see that sharp increase or dip in the graph instead the lines are smoother than graph 1. Epsilon can be thought of as a learning rate. Normal epsilon is optimal, rest one converges faster but graph is not smooth and the other is slow to converge and takes smaller stride.