RemoveNodeMAB-Different_bandit

March 22, 2017

Compute the performance of MAB methods

```
In [9]: import numpy as np
        import time
        import sys
        from numpy import *
        import matplotlib.pyplot as plt
        from sklearn import metrics
        %matplotlib inline
        #plt.rcParams['figure.figsize'] = (15, 6)
```

0.1 Load BOKEH lib.

```
In [10]: from bokeh.layouts import row, gridplot
         from bokeh.plotting import figure, output_notebook, show
         from bokeh.models import Legend

         ##########################################################################################
         TOOLS = 'box_zoom,box_select,crosshair,resize,reset,lasso_select,pan,save,poly_select,t
         output_notebook()
         ##########################################################################################
```

1 Load the data

```
In [11]: X_train = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/X
         y_train = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/y
         X_test = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/X_
         y_test = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/y_
         X_deploy = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/
         y_deploy = np.load('/Users/salemameen/Desktop/banditsbook/python_WineQuilty/winequilty/

         print('Number of training examples',len(X_train))
         print('Number of validation examples',len(X_test))
         print('Number of testing examples',len(X_deploy))
```

```
Number of training examples 4157
Number of validation examples 1040
```

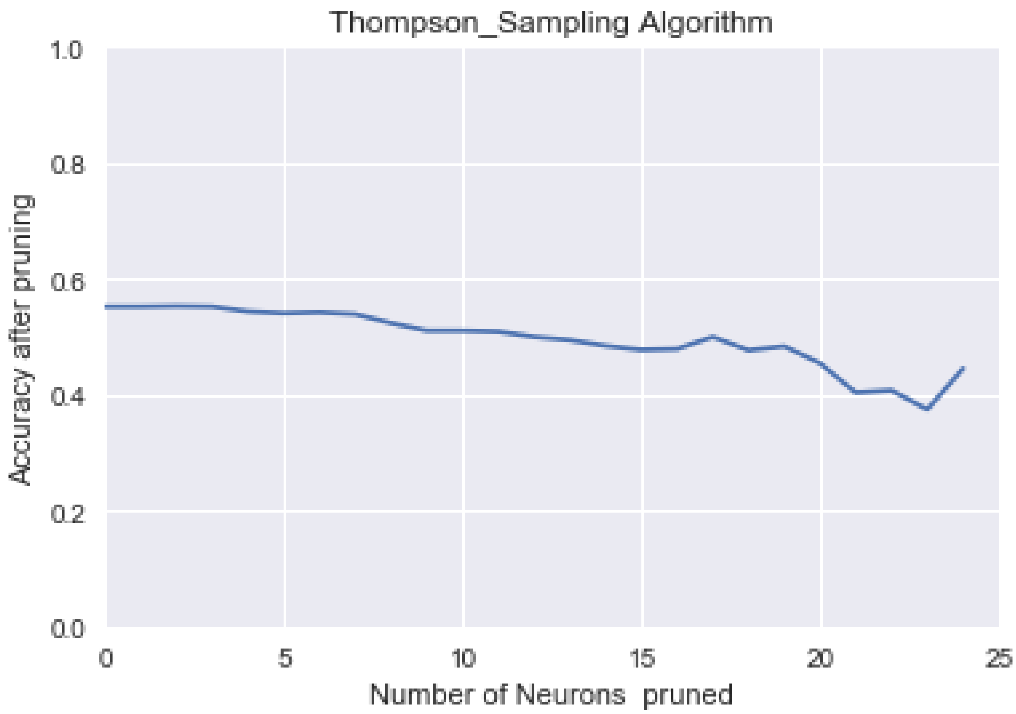
```
Number of testing examples 1300
```

```
In [12]: exec(open("core.py").read())  # pyhton 3x
         #exec(compile(open('core.py', "rb").read(), 'core.py', 'exec'))

         #execfile("./core.py") # python 2.7
```

1.1 Run Thompson Sampling pruning Algorithm

```
In [13]: algo = Thompson_Sampling([], [])
         Alg_name = 'Thompson_Sampling Algorithm'
         path = './Thompson_Sampling/'
         sys.path.append("./Thompson_Sampling")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.2061460018157959 seconds
Finsh playing start pruining:
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.50
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.45
Test after pruning= 0.40
Test after pruning= 0.41
Test after pruning= 0.37
Test after pruning= 0.45
```

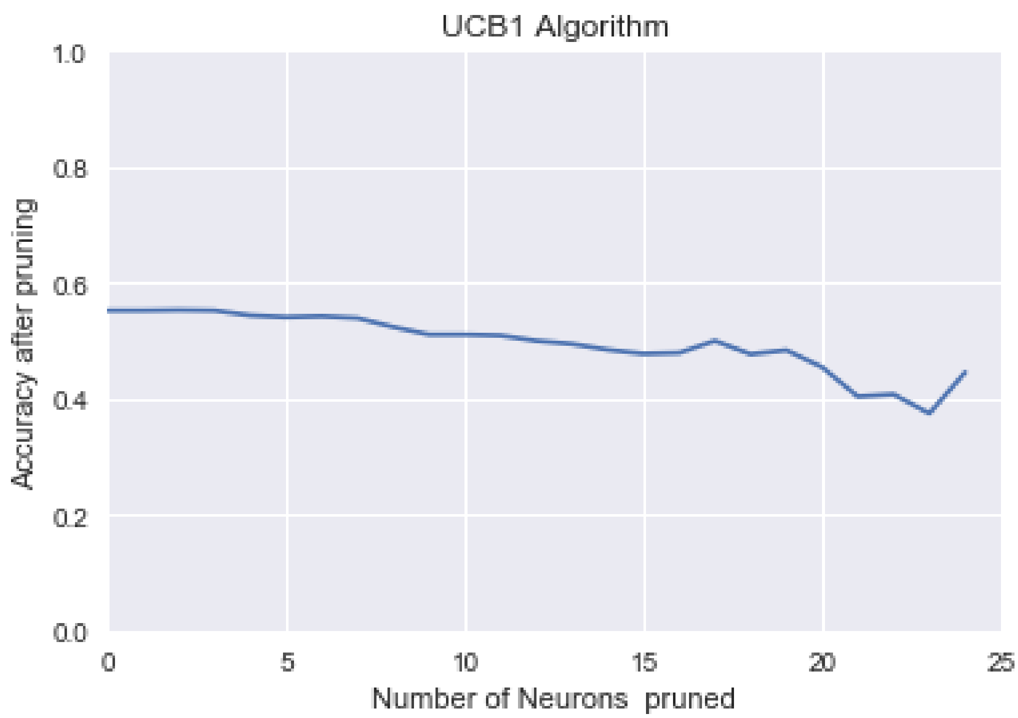


1.2 Run UCB1 pruning Algorithm

```
In [14]: algo = UCB1([], [])
         Alg_name = 'UCB1 Algorithm'
         path = './UCB1/'
         sys.path.append("./UCB1")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.1848280429840088 seconds
Finsh playing start pruining:
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.51
```

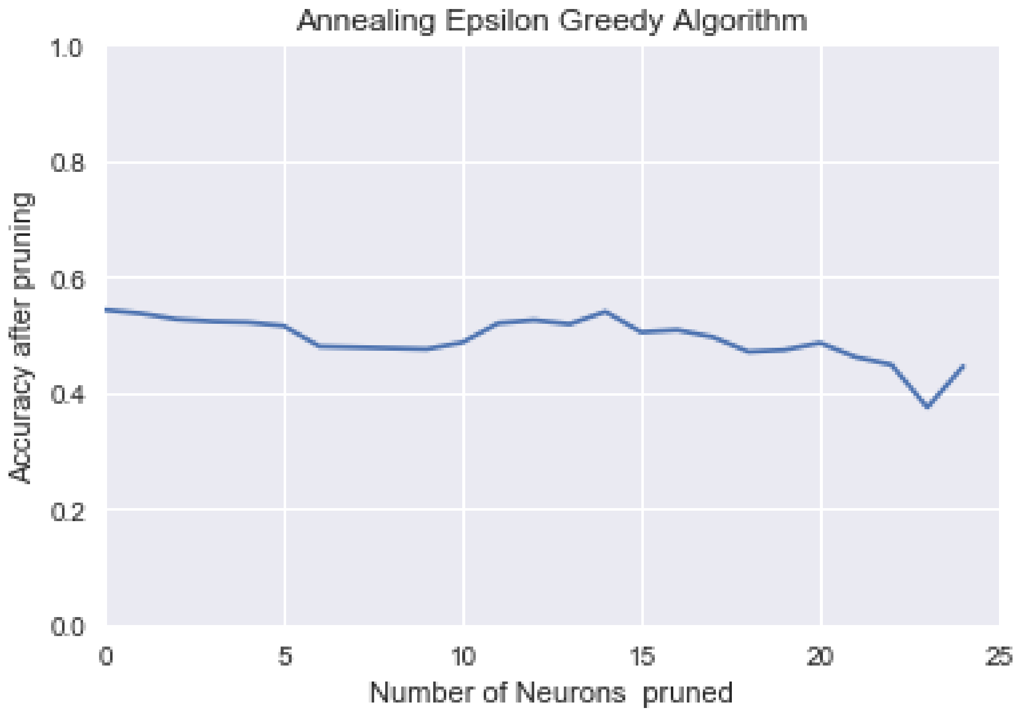
```
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.50
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.45
Test after pruning= 0.40
Test after pruning= 0.41
Test after pruning= 0.37
Test after pruning= 0.45
```



2 Run Annealing Epsilon Greedy pruning Algorithm

```
In [15]: algo = AnnealingEpsilonGreedy([], [])
         Alg_name = 'Annealing Epsilon Greedy Algorithm'
         path = './AnnealingEpsilonGreedy/'
         sys.path.append("./AnnealingEpsilonGreedy")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.18368005752563477 seconds
Finsh playing start pruining:
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.49
Test after pruning= 0.52
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.54
Test after pruning= 0.50
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.47
Test after pruning= 0.47
Test after pruning= 0.49
Test after pruning= 0.46
Test after pruning= 0.45
Test after pruning= 0.37
Test after pruning= 0.45
```

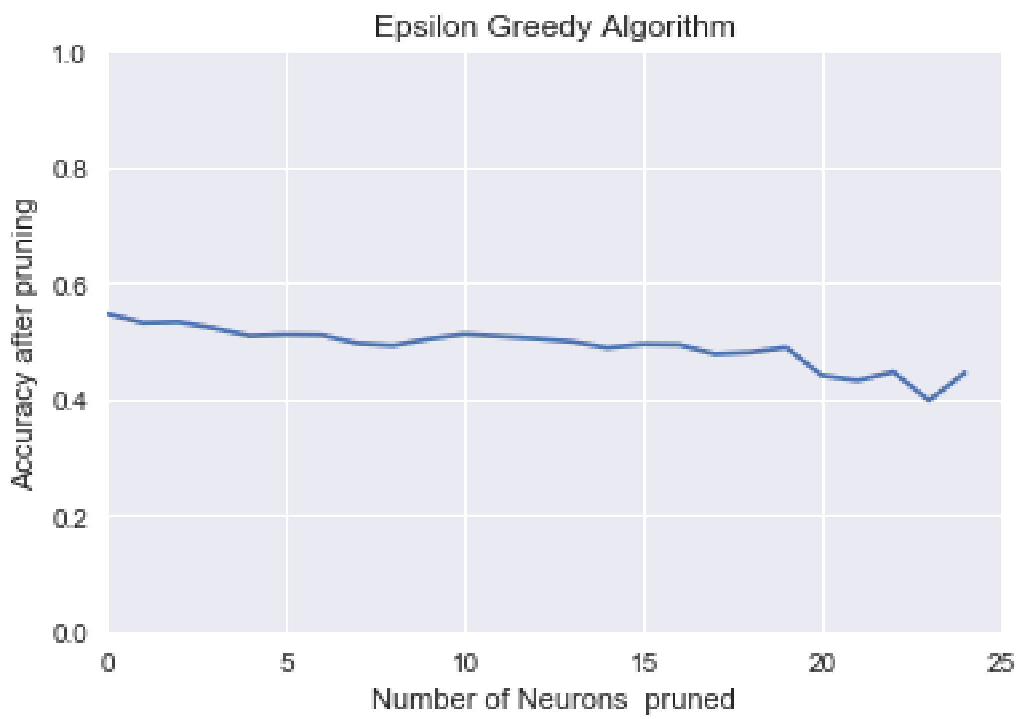


3 Run Epsilon Greedy pruning Algorithm

```
In [16]: epsilon = 0.9 # epsilon = (0,1)
         algo = EpsilonGreedy(epsilon, [], [])
         Alg_name = 'Epsilon Greedy Algorithm'
         path = './EpsilonGreedy/'
         sys.path.append("./AnnealingEpsilonGreedy")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.19059395790100098 seconds
Finsh playing start pruining:
Test after pruning= 0.55
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.50
```

```
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.49
Test after pruning= 0.49
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.49
Test after pruning= 0.44
Test after pruning= 0.43
Test after pruning= 0.45
Test after pruning= 0.40
Test after pruning= 0.45
```

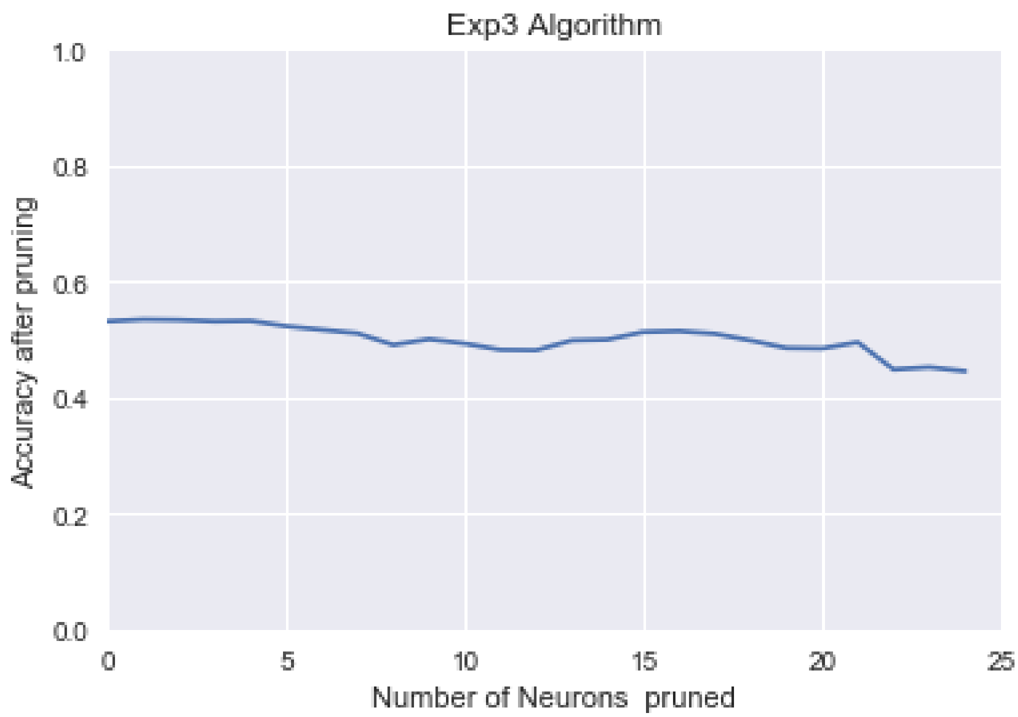


4 Run Exp3 pruning Algorithm

```
In [17]: exp3_gamma = 0.2 #exp3_gamma in [0.1, 0.2, 0.3, 0.4, 0.5]
         algo = Exp3(exp3_gamma, [])
         Alg_name = 'Exp3 Algorithm'
         path = './Exp3/'
```

```
        sys.path.append("./EpsilonGreedy")
        exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.1860790252685547 seconds
Finsh playing start pruining:
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.49
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.50
Test after pruning= 0.50
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.48
Test after pruning= 0.50
Test after pruning= 0.45
Test after pruning= 0.45
Test after pruning= 0.45
```

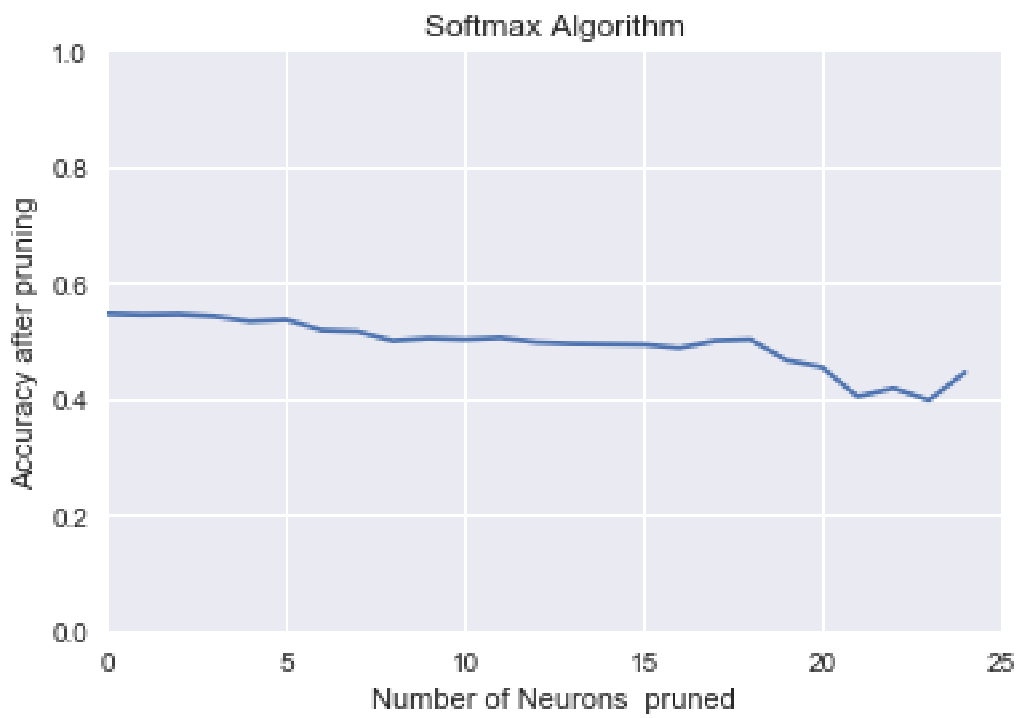


5 Run Softmax pruning Algorithm

```
In [18]: temperature = 0.9
         algo = Softmax(temperature, [], [])
         Alg_name = 'Softmax Algorithm'
         path = './Softmax/'
         sys.path.append("./Softmax")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.1844310760498047 seconds
Finsh playing start pruining:
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.54
Test after pruning= 0.53
Test after pruning= 0.54
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.50
Test after pruning= 0.50
```

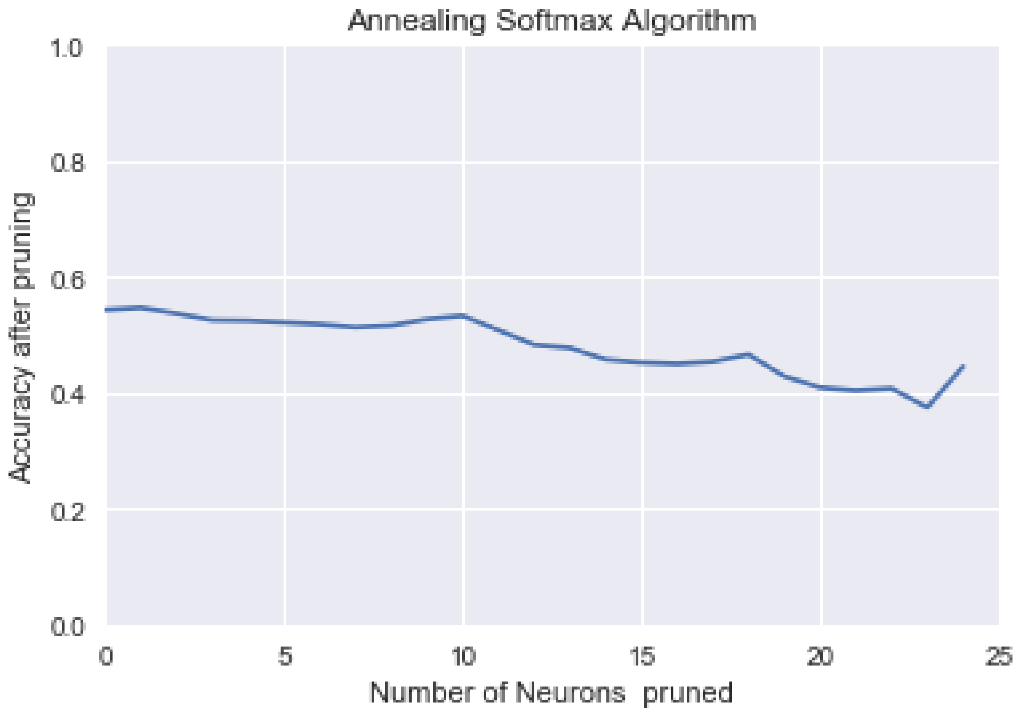
```
Test after pruning= 0.50
Test after pruning= 0.51
Test after pruning= 0.50
Test after pruning= 0.50
Test after pruning= 0.49
Test after pruning= 0.49
Test after pruning= 0.49
Test after pruning= 0.50
Test after pruning= 0.50
Test after pruning= 0.47
Test after pruning= 0.45
Test after pruning= 0.40
Test after pruning= 0.42
Test after pruning= 0.40
Test after pruning= 0.45
```



6 Run Annealing Softmax pruning Algorithm

In [19]:
```
algo = AnnealingSoftmax([], [])
Alg_name = 'Annealing Softmax Algorithm'
path = './AnnealingSoftmax/'
sys.path.append("./AnnealingSoftmax")
exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.20129728317260742 seconds
Finsh playing start pruining:
Test after pruning= 0.54
Test after pruning= 0.55
Test after pruning= 0.54
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.52
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.51
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.46
Test after pruning= 0.45
Test after pruning= 0.45
Test after pruning= 0.45
Test after pruning= 0.47
Test after pruning= 0.43
Test after pruning= 0.41
Test after pruning= 0.40
Test after pruning= 0.41
Test after pruning= 0.37
Test after pruning= 0.45
```

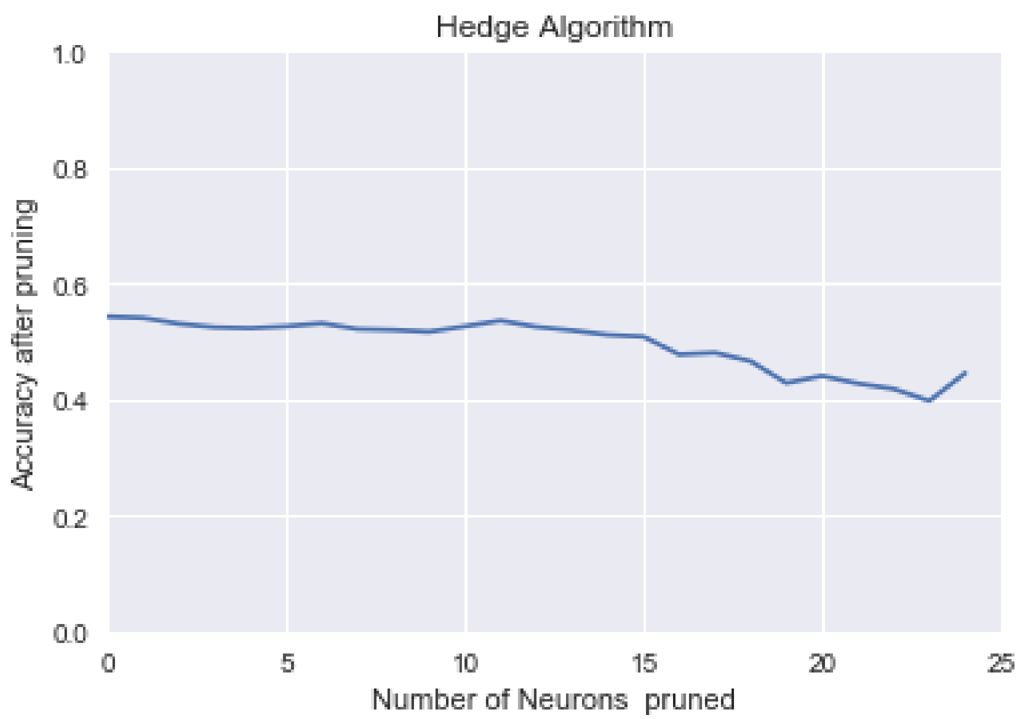


7 Run Hedge pruning Algorithm

```
In [20]: eta = 0.9  # eta in [.5, .8, .9, 1, 2]
         algo = Hedge(eta, [], [])
         Alg_name = 'Hedge Algorithm'
         path = './Hedge/'
         sys.path.append("./Hedge")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 1.06
Test fraction correct (NN-Accuracy) = 0.55
The time for running this method is 0.19228482246398926 seconds
Finsh playing start pruining:
Test after pruning= 0.54
Test after pruning= 0.54
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.53
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.52
Test after pruning= 0.52
```

```
Test after pruning= 0.53
Test after pruning= 0.54
Test after pruning= 0.53
Test after pruning= 0.52
Test after pruning= 0.51
Test after pruning= 0.51
Test after pruning= 0.48
Test after pruning= 0.48
Test after pruning= 0.47
Test after pruning= 0.43
Test after pruning= 0.44
Test after pruning= 0.43
Test after pruning= 0.42
Test after pruning= 0.40
Test after pruning= 0.45
```

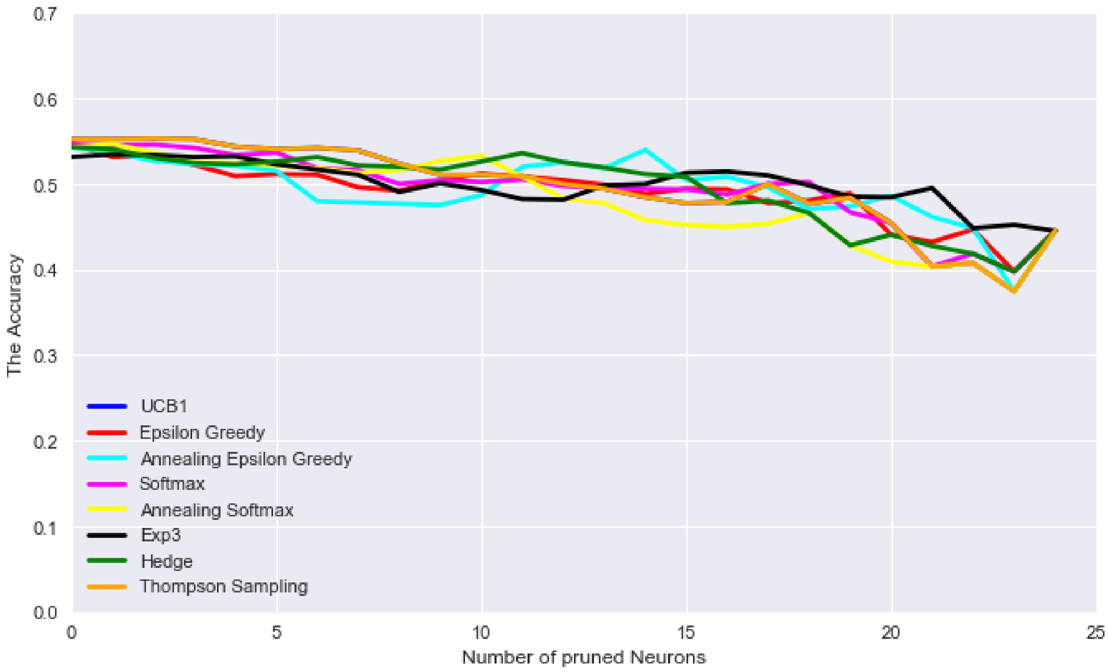


8 Compare the accuracy of the models

In [21]:
```
ucb1 = np.load('./UCB1/AccuracyAftrerPrune.npy')
EpsilonGreedy = np.load('./EpsilonGreedy/AccuracyAftrerPrune.npy')
AnnealingEpsilonGreedy = np.load('./AnnealingEpsilonGreedy/AccuracyAftrerPrune.npy')
Softmax = np.load('./Softmax/AccuracyAftrerPrune.npy')
```

```
AnnealingSoftmax = np.load('./AnnealingSoftmax/AccuracyAftrerPrune.npy')
Exp3 = np.load('./Exp3/AccuracyAftrerPrune.npy')
Hedge = np.load('./Hedge/AccuracyAftrerPrune.npy')
ThompsonSampling = np.load('./Thompson_Sampling/AccuracyAftrerPrune.npy')
Accuracy = np.load('AccuracyBeforePruning.npy')
```

In [22]:
```
fig = plt.figure(figsize=(10, 6), dpi=80)
ax = fig.add_subplot(111)
N = len(ucb1)
## necessary variables
ind = np.arange(N)                  # the x locations for the groups
plt.plot(ind , ucb1 , color="blue", linewidth=2.5, linestyle="-", label="UCB1")
plt.plot(ind , EpsilonGreedy, color="red", linewidth=2.5, linestyle="-", label="Epsilon
plt.plot(ind , AnnealingEpsilonGreedy, color="cyan", linewidth=2.5, linestyle="-", labe
plt.plot(ind , Softmax, color="magenta", linewidth=2.5, linestyle="-", label="Softmax")
plt.plot(ind , AnnealingSoftmax, color="yellow", linewidth=2.5, linestyle="-", label="A
plt.plot(ind , Exp3, color="black", linewidth=2.5, linestyle="-", label="Exp3")
plt.plot(ind , Hedge, color="green", linewidth=2.5, linestyle="-", label="Hedge")
plt.plot(ind , ThompsonSampling, color="orange", linewidth=2.5, linestyle="-", label="T
plt.legend(loc = 3)
plt.axis([0, 25, 0, 0.7])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

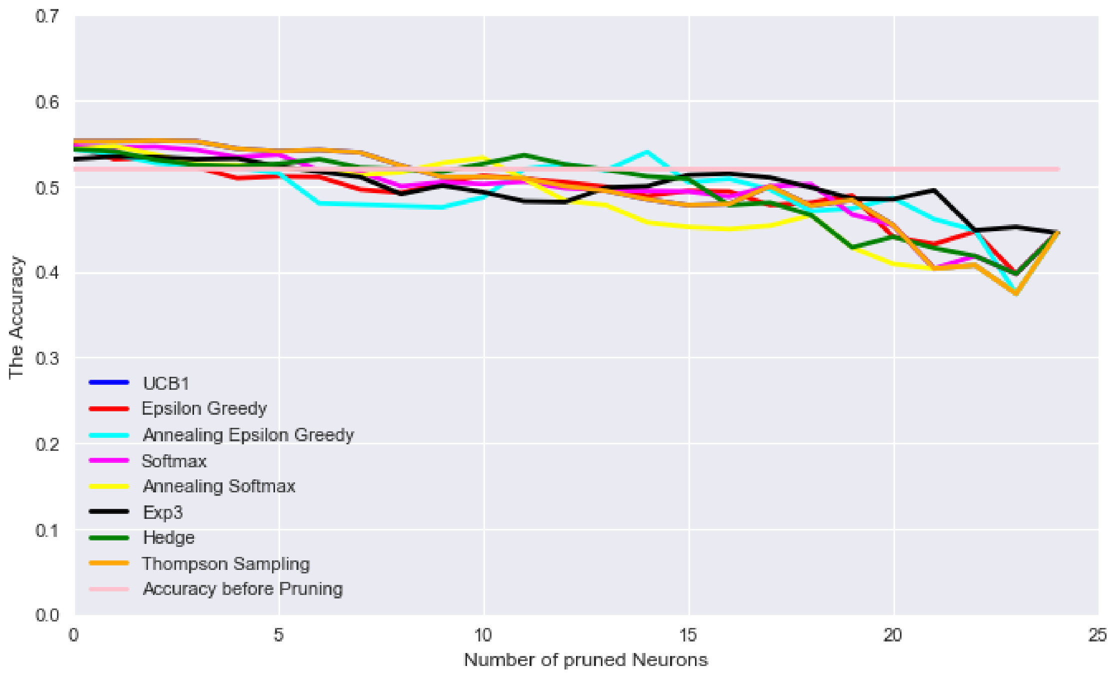


```
In [23]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #p1.circle(ind, ucb1, legend="ucb1", color="orange")
         p1.line(ind, ucb1, legend="ucb1", line_color="orange", line_width=2)

         #p1.square(ind, EpsilonGreedy, legend="Epsilon Greedy", fill_color=None, line_color="re
         p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

         #p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color=
         p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blu

         #p1.diamond(ind, Softmax, legend="Softmax", line_color="green")
         p1.line(ind, Softmax, legend="Softmax", line_color="green", line_width=2)

         #p1.arc(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", end_angle
         p1.line(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", line_widt

         #p1.oval(ind, Exp3, legend="Exp3", line_color="black", height=0.01, width=0.01)
         p1.line(ind, Exp3, legend="Exp3", line_color="black", line_width=2)

         #p1.arc(ind, Hedge, legend="Hedge", line_color="yellow")
         #p1.triangle(ind, Hedge, legend="Hedge", line_color="yellow")
         p1.line(ind, Hedge, legend="Hedge", line_color="yellow", line_width=2)


         #p1.square_cross(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink")
         p1.line(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink", line_widt


         #p1.line(ind, Exp3, legend="2*sin(x)", line_dash=(4, 4), line_color="orange", line_widt
         #p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
         p1.title.align = "center"


         show(p1)
         #show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```

8.1 Comparing All algorithms with the model before pruning

```
In [24]: fig = plt.figure(figsize=(10, 6), dpi=80)
         ax = fig.add_subplot(111)
         N = len(ucb1)
         Acc = [Accuracy for col in range(N)]
         ## necessary variables
         ind = np.arange(N)                # the x locations for the groups
         plt.plot(ind , ucb1 , color="blue", linewidth=2.5, linestyle="-", label="UCB1")
         plt.plot(ind , EpsilonGreedy, color="red", linewidth=2.5, linestyle="-", label="Epsilon
```

```
plt.plot(ind , AnnealingEpsilonGreedy, color="cyan", linewidth=2.5, linestyle="-", labe
plt.plot(ind , Softmax, color="magenta", linewidth=2.5, linestyle="-", label="Softmax")
plt.plot(ind , AnnealingSoftmax, color="yellow", linewidth=2.5, linestyle="-", label="A
plt.plot(ind , Exp3, color="black", linewidth=2.5, linestyle="-", label="Exp3")
plt.plot(ind , Hedge, color="green", linewidth=2.5, linestyle="-", label="Hedge")
plt.plot(ind , ThompsonSampling, color="orange", linewidth=2.5, linestyle="-", label="T
plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
plt.legend(loc = 3)
plt.axis([0, 25, 0, 0.7])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

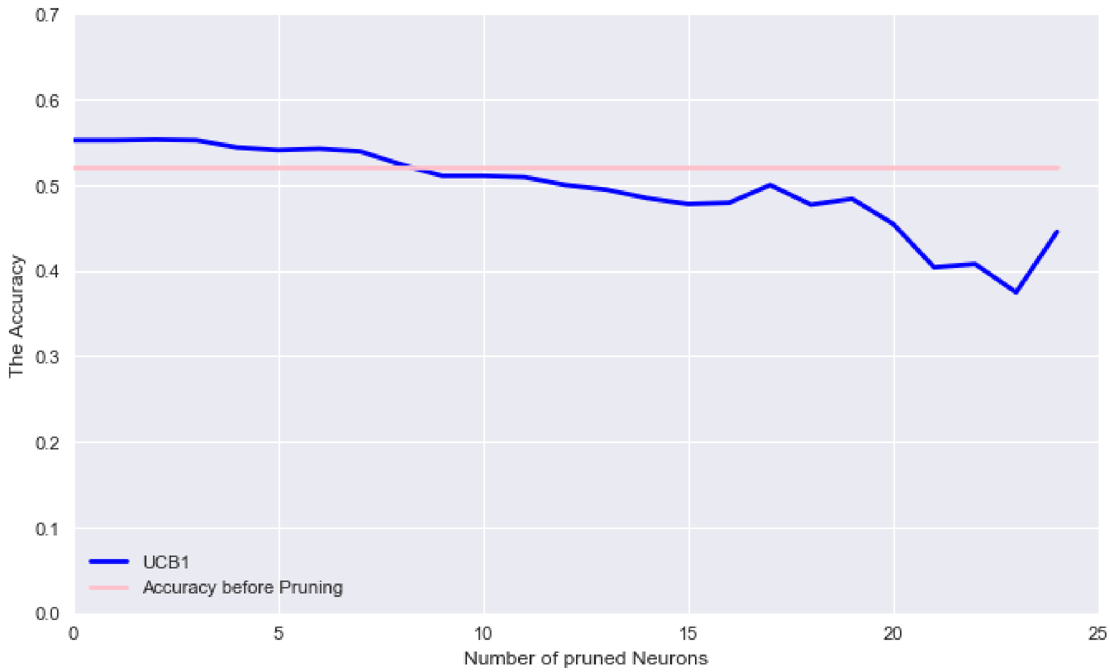



```
In [25]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #p1.circle(ind, ucb1, legend="ucb1", color="orange")
         p1.line(ind, ucb1, legend="ucb1", line_color="orange", line_width=2)

         #p1.square(ind, EpsilonGreedy, legend="Epsilon Greedy", fill_color=None, line_color="re
         p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

         #p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color=
         p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blu
```

```
#p1.diamond(ind, Softmax, legend="Softmax", line_color="green")
p1.line(ind, Softmax, legend="Softmax", line_color="green", line_width=2)

#p1.arc(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", end_angle
p1.line(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", line_widt

#p1.oval(ind, Exp3, legend="Exp3", line_color="black", height=0.01, width=0.01)
p1.line(ind, Exp3, legend="Exp3", line_color="black", line_width=2)

#p1.arc(ind, Hedge, legend="Hedge", line_color="yellow")
#p1.triangle(ind, Hedge, legend="Hedge", line_color="yellow")
p1.line(ind, Hedge, legend="Hedge", line_color="yellow", line_width=2)


#p1.square_cross(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink")
p1.line(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink", line_widt


p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=
#p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
p1.title.align = "center"


show(p1)
#show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```

8.2 UCB1

In [26]:
```
fig = plt.figure(figsize=(10, 6), dpi=80)
ax = fig.add_subplot(111)
N = len(ucb1)
Acc = [Accuracy for col in range(N)]
## necessary variables
ind = np.arange(N)                # the x locations for the groups
plt.plot(ind , ucb1 , color="blue", linewidth=2.5, linestyle="-", label="UCB1")
plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
plt.legend(loc = 3)
plt.axis([0, 25, 0, 0.7])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

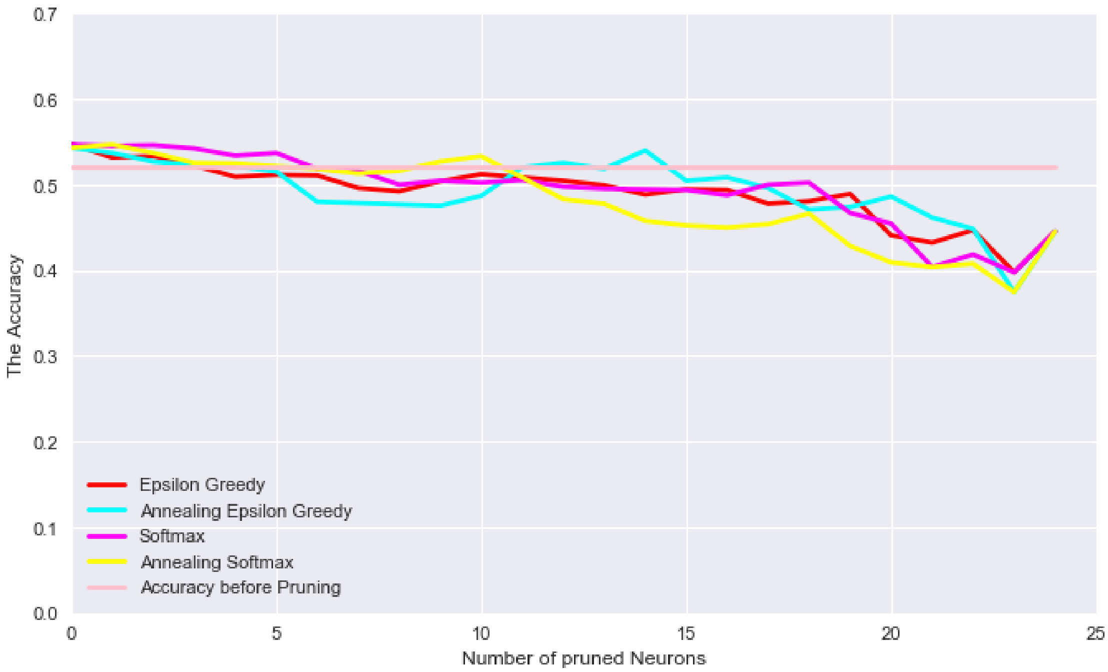



```
In [27]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #p1.circle(ind, ucb1, legend="ucb1", color="orange")
         p1.line(ind, ucb1, legend="ucb1", line_color="orange", line_width=2)
         p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=
         #p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
         p1.title.align = "center"


         show(p1)
         #show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```

8.3 Epsilon greedy and Softmax

```
In [28]: fig = plt.figure(figsize=(10, 6), dpi=80)
         ax = fig.add_subplot(111)
         N = len(EpsilonGreedy)
         Acc = [Accuracy for col in range(N)]
         ## necessary variables
         ind = np.arange(N)                    # the x locations for the groups
         plt.plot(ind , EpsilonGreedy, color="red", linewidth=2.5, linestyle="-", label="Epsilon
         plt.plot(ind , AnnealingEpsilonGreedy, color="cyan", linewidth=2.5, linestyle="-", labe
         plt.plot(ind , Softmax, color="magenta", linewidth=2.5, linestyle="-", label="Softmax")
         plt.plot(ind , AnnealingSoftmax, color="yellow", linewidth=2.5, linestyle="-", label="A
```

```
plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
plt.legend(loc = 3)
plt.axis([0, 25, 0, 0.7])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

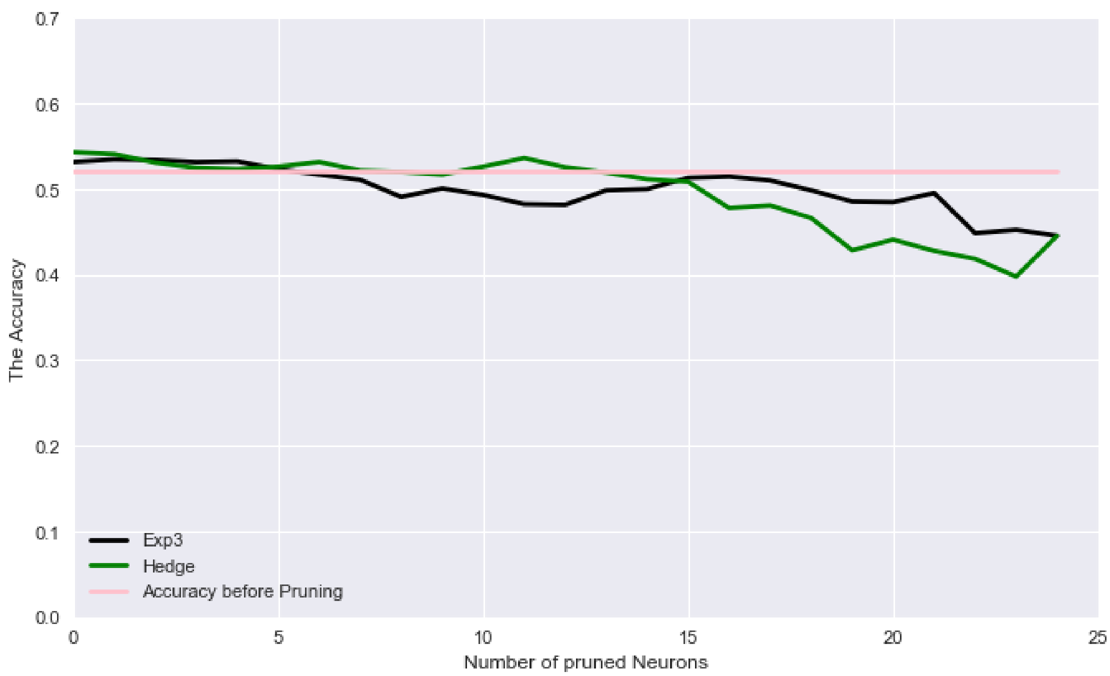



```
In [29]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #p1.square(ind, EpsilonGreedy, legend="Epsilon Greedy", fill_color=None, line_color="re
         p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

         #p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color=
         p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blu

         #p1.diamond(ind, Softmax, legend="Softmax", line_color="green")
         p1.line(ind, Softmax, legend="Softmax", line_color="green", line_width=2)

         #p1.arc(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", end_angle
         p1.line(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", line_widt

         p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=
         #p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
```

```
p1.title.align = "center"


show(p1)
#show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```

8.4 Adversial Bandits Hedge and EXP3

```
In [30]: fig = plt.figure(figsize=(10, 6), dpi=80)
         ax = fig.add_subplot(111)
         N = len(Exp3)
         Acc = [Accuracy for col in range(N)]
         ## necessary variables
         ind = np.arange(N)                # the x locations for the groups
         plt.plot(ind , Exp3, color="black", linewidth=2.5, linestyle="-", label="Exp3")
         plt.plot(ind , Hedge, color="green", linewidth=2.5, linestyle="-", label="Hedge")
         plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
         plt.legend(loc = 3)
         plt.axis([0, 25, 0, 0.7])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```

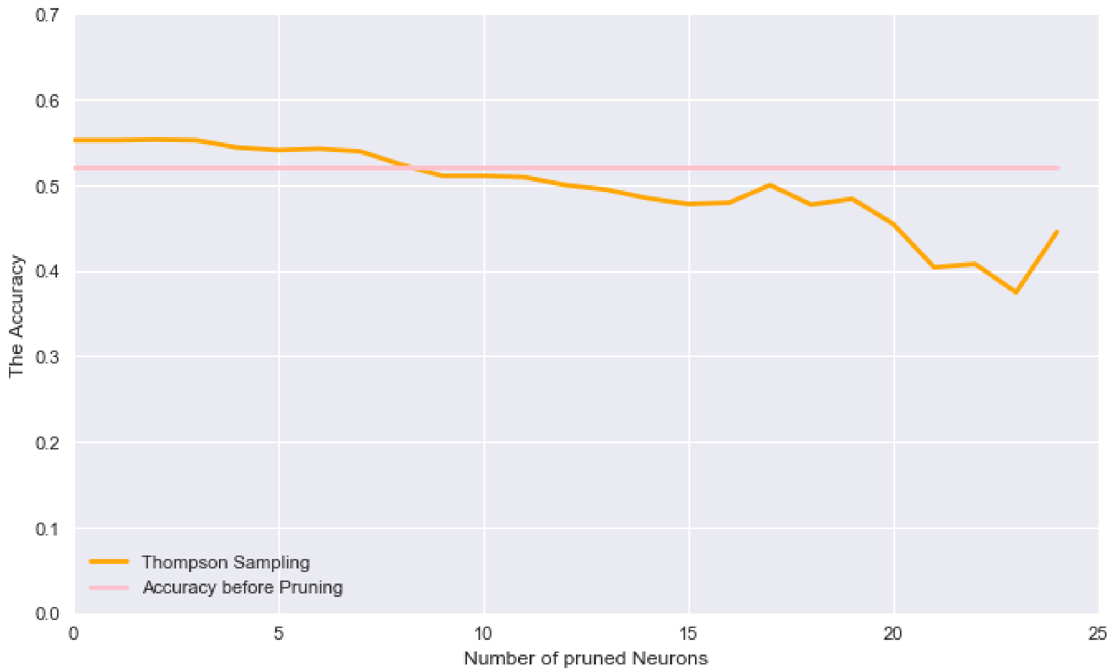



```
In [31]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)
```

```
#p1.oval(ind, Exp3, legend="Exp3", line_color="black", height=0.01, width=0.01)
p1.line(ind, Exp3, legend="Exp3", line_color="black", line_width=2)

#p1.arc(ind, Hedge, legend="Hedge", line_color="yellow")
#p1.triangle(ind, Hedge, legend="Hedge", line_color="yellow")
p1.line(ind, Hedge, legend="Hedge", line_color="yellow", line_width=2)

p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=
#p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
p1.title.align = "center"


show(p1)
#show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```

9 Thompson Sampling

```
In [32]: fig = plt.figure(figsize=(10, 6), dpi=80)
         ax = fig.add_subplot(111)
         N = len(ThompsonSampling)
         Acc = [Accuracy for col in range(N)]
         ## necessary variables
         ind = np.arange(N)                # the x locations for the groups
         plt.plot(ind , ThompsonSampling, color="orange", linewidth=2.5, linestyle="-", label="T
         plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
         plt.legend(loc = 3)
         plt.axis([0, 25, 0, 0.7])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```



In [33]:
```
p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)

#p1.square_cross(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink")
p1.line(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink", line_widt


p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=
#p1.square(ind, Hedge, legend="3*sin(x)", fill_color=None, line_color="brown")
p1.title.align = "center"


show(p1)
#show(gridplot(p1, p2, ncols=2, plot_width=400, plot_height=400))  # open a browser
```