RemoveNodeMAB-Different_bandit

March 22, 2017

Compute the performance of MAB methods

In [1]:
```
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)
```

In [2]:
```
## Load BOKEH Lib.
```

In [3]:
```
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,ta
output_notebook()
####################################################################################
```

1 Load the data

In [4]:
```
X_train = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/X_train.npy')
y_train = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/y_train.npy')
X_test = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/X_test.npy')
y_test = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/y_test.npy')
X_deploy = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/X_deploy.npy')
y_deploy = np.load('/Users/salemameen/Desktop/banditsbook/python_car/car/y_deploy.npy')



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 1105
Number of validation examples 277
Number of testing examples 346
```

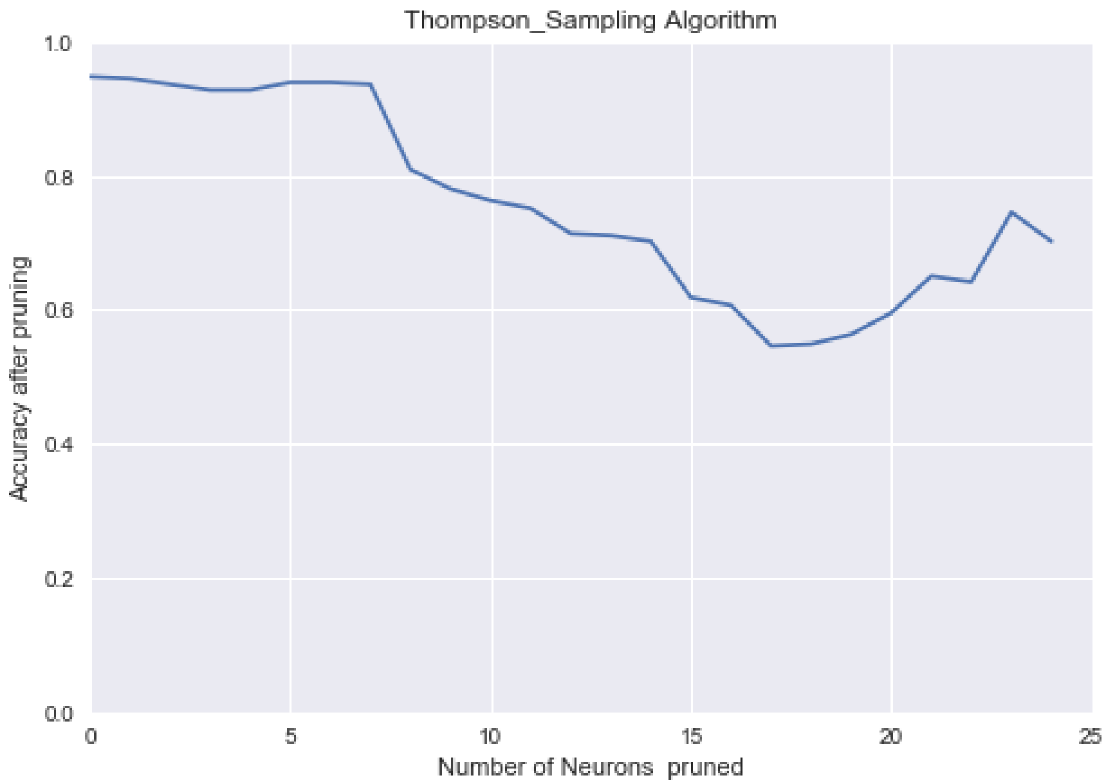
```
In [5]: exec(open("core.py").read())  # pyhton 3x
        #exec(compile(open('core.py', "rb").read(), 'core.py', 'exec'))

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

1.1 Run Thompson Sampling pruning Algorithm

```
In [6]: algo = Thompson_Sampling([], [])
        Alg_name = 'Thompson_Sampling Algorithm'
        path = './Thompson_Sampling/'
        sys.path.append("./Thompson_Sampling")
        exec(open("mnist_cnnFORTESTING.py").read())
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklearn/cross_vali
  "This module will be removed in 0.20.", DeprecationWarning)
Using Theano backend.


Test fraction correct (NN-Score) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.3398900032043457 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.95
Test after pruning= 0.94
Test after pruning= 0.93
Test after pruning= 0.93
Test after pruning= 0.94
Test after pruning= 0.94
Test after pruning= 0.94
Test after pruning= 0.81
Test after pruning= 0.78
Test after pruning= 0.76
Test after pruning= 0.75
Test after pruning= 0.71
Test after pruning= 0.71
Test after pruning= 0.70
Test after pruning= 0.62
Test after pruning= 0.61
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.56
Test after pruning= 0.60
Test after pruning= 0.65
```

```
Test after pruning= 0.64
Test after pruning= 0.75
Test after pruning= 0.70
```

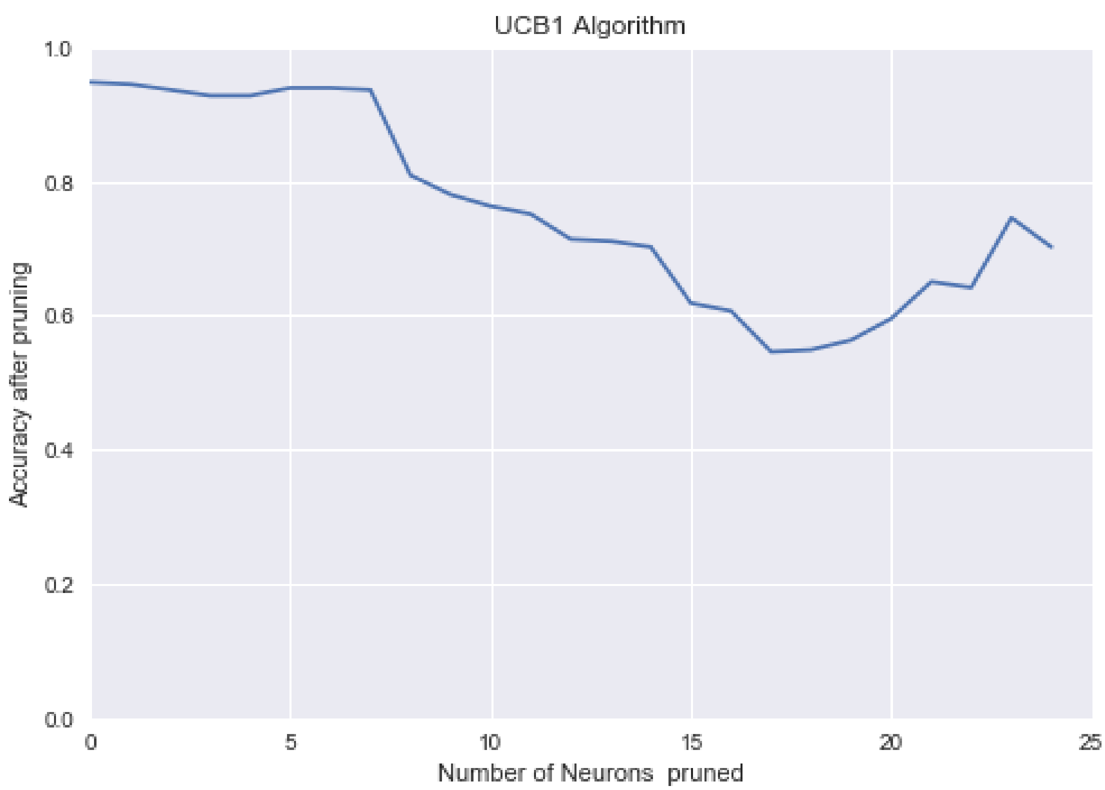


1.2 Run UCB1 pruning Algorithm

```
In [7]: algo = UCB1([], [])
        Alg_name = 'UCB1 Algorithm'
        path = './UCB1/'
        sys.path.append("./UCB1")
        exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.32602810859680176 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.95
Test after pruning= 0.94
Test after pruning= 0.93
Test after pruning= 0.93
```

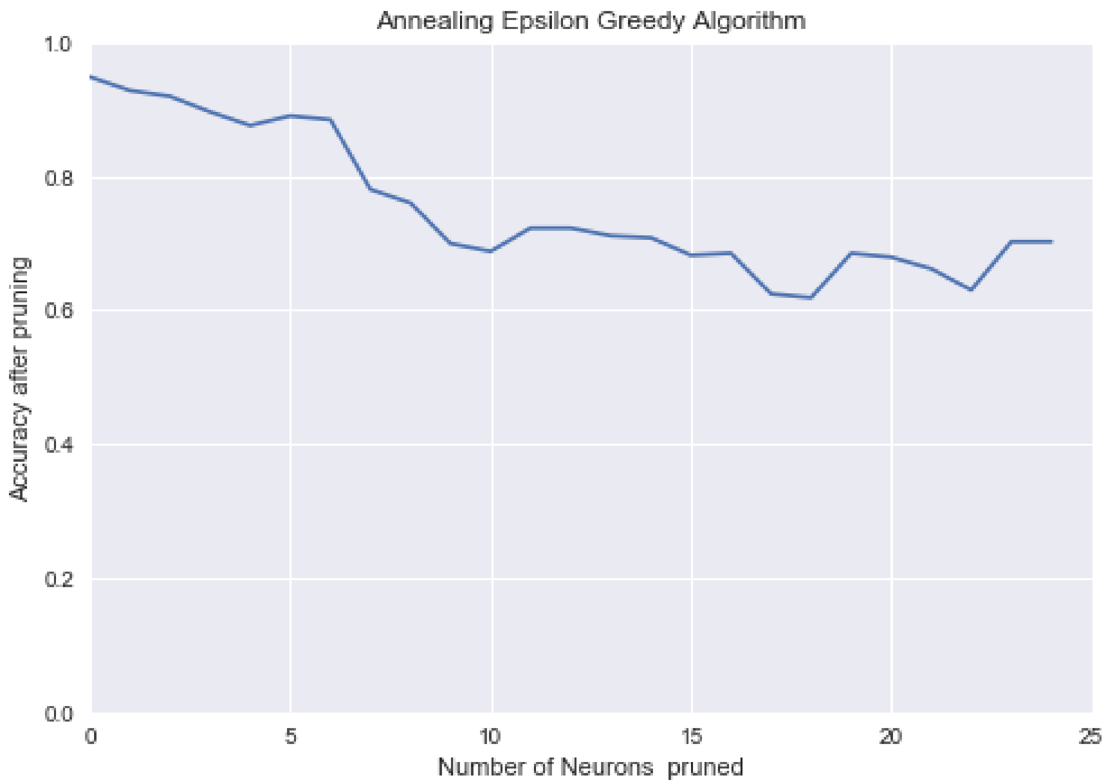
```
Test after pruning= 0.94
Test after pruning= 0.94
Test after pruning= 0.94
Test after pruning= 0.81
Test after pruning= 0.78
Test after pruning= 0.76
Test after pruning= 0.75
Test after pruning= 0.71
Test after pruning= 0.71
Test after pruning= 0.70
Test after pruning= 0.62
Test after pruning= 0.61
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.56
Test after pruning= 0.60
Test after pruning= 0.65
Test after pruning= 0.64
Test after pruning= 0.75
Test after pruning= 0.70
```



2 Run Annealing Epsilon Greedy pruning Algorithm

```
In [8]: 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) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.32065510749816895 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.93
Test after pruning= 0.92
Test after pruning= 0.90
Test after pruning= 0.88
Test after pruning= 0.89
Test after pruning= 0.88
Test after pruning= 0.78
Test after pruning= 0.76
Test after pruning= 0.70
Test after pruning= 0.69
Test after pruning= 0.72
Test after pruning= 0.72
Test after pruning= 0.71
Test after pruning= 0.71
Test after pruning= 0.68
Test after pruning= 0.68
Test after pruning= 0.62
Test after pruning= 0.62
Test after pruning= 0.68
Test after pruning= 0.68
Test after pruning= 0.66
Test after pruning= 0.63
Test after pruning= 0.70
Test after pruning= 0.70
```

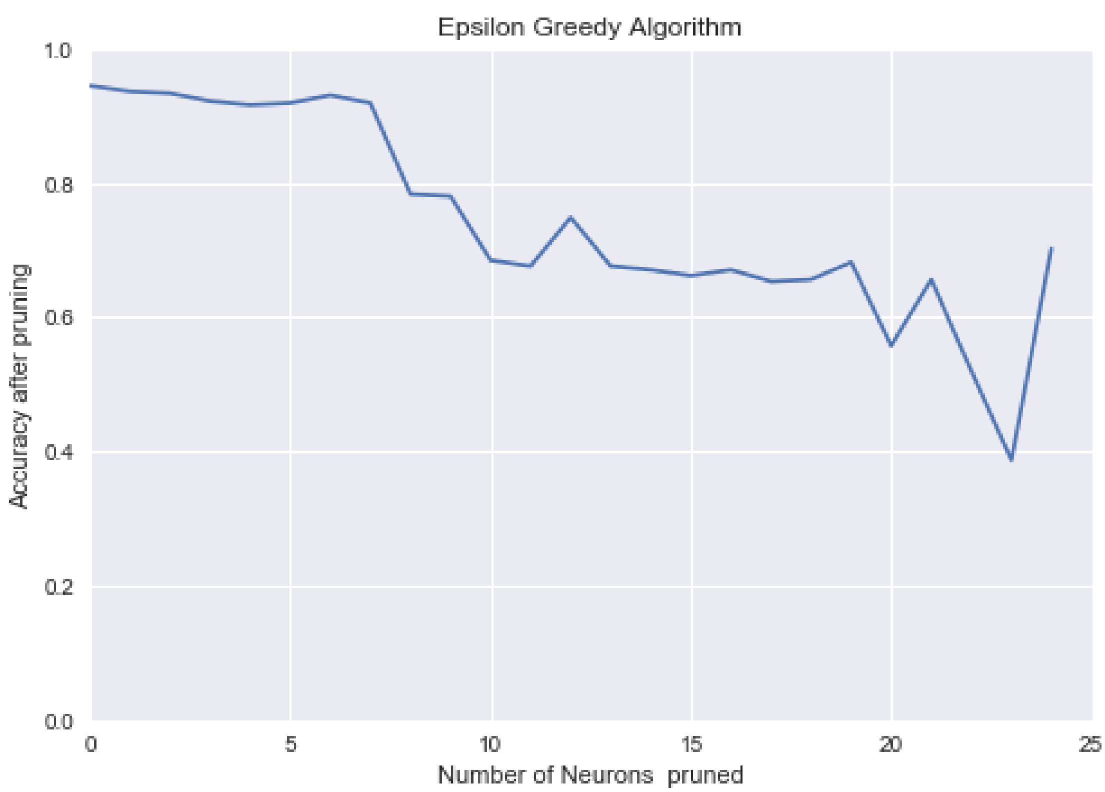


3 Run Epsilon Greedy pruning Algorithm

```
In [9]: 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) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.33334994316101074 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.94
Test after pruning= 0.93
Test after pruning= 0.92
Test after pruning= 0.92
Test after pruning= 0.92
Test after pruning= 0.93
Test after pruning= 0.92
```

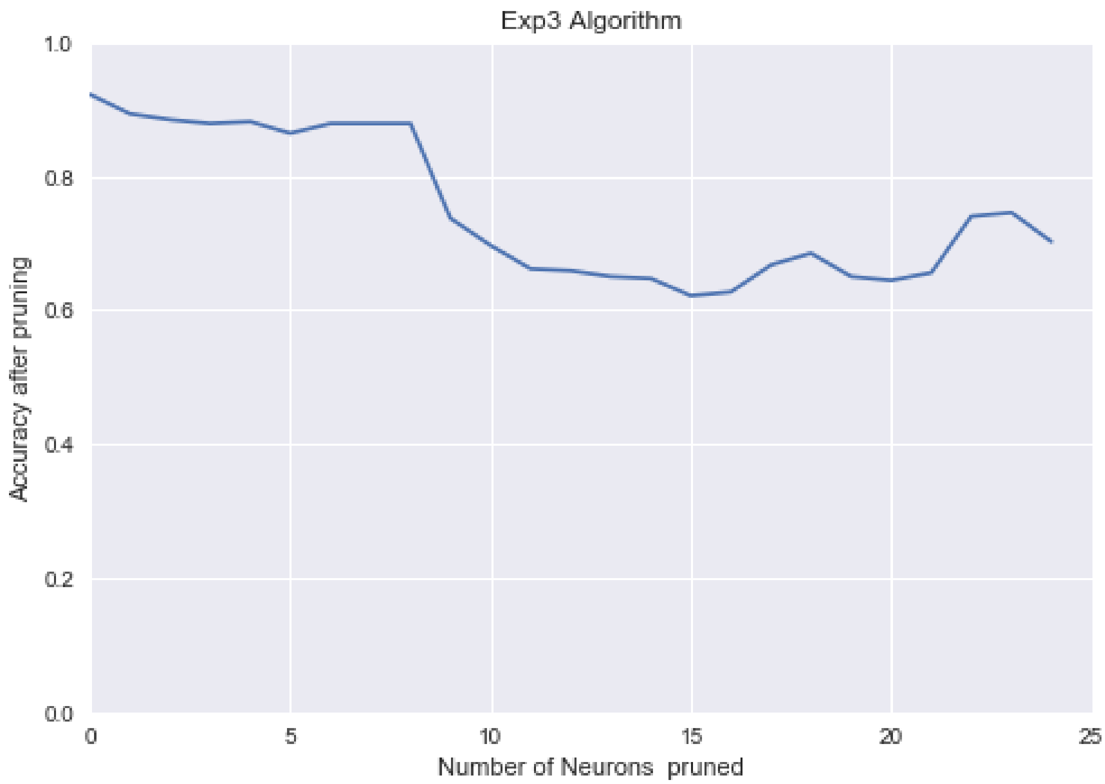
```
Test after pruning= 0.78
Test after pruning= 0.78
Test after pruning= 0.68
Test after pruning= 0.68
Test after pruning= 0.75
Test after pruning= 0.68
Test after pruning= 0.67
Test after pruning= 0.66
Test after pruning= 0.67
Test after pruning= 0.65
Test after pruning= 0.66
Test after pruning= 0.68
Test after pruning= 0.56
Test after pruning= 0.66
Test after pruning= 0.52
Test after pruning= 0.39
Test after pruning= 0.70
```



4 Run Exp3 pruning Algorithm

```
In [10]: 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) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.3298780918121338 seconds
Finsh playing start pruining:
Test after pruning= 0.92
Test after pruning= 0.89
Test after pruning= 0.88
Test after pruning= 0.88
Test after pruning= 0.88
Test after pruning= 0.86
Test after pruning= 0.88
Test after pruning= 0.88
Test after pruning= 0.88
Test after pruning= 0.74
Test after pruning= 0.70
Test after pruning= 0.66
Test after pruning= 0.66
Test after pruning= 0.65
Test after pruning= 0.65
Test after pruning= 0.62
Test after pruning= 0.63
Test after pruning= 0.67
Test after pruning= 0.68
Test after pruning= 0.65
Test after pruning= 0.64
Test after pruning= 0.66
Test after pruning= 0.74
Test after pruning= 0.75
Test after pruning= 0.70
```

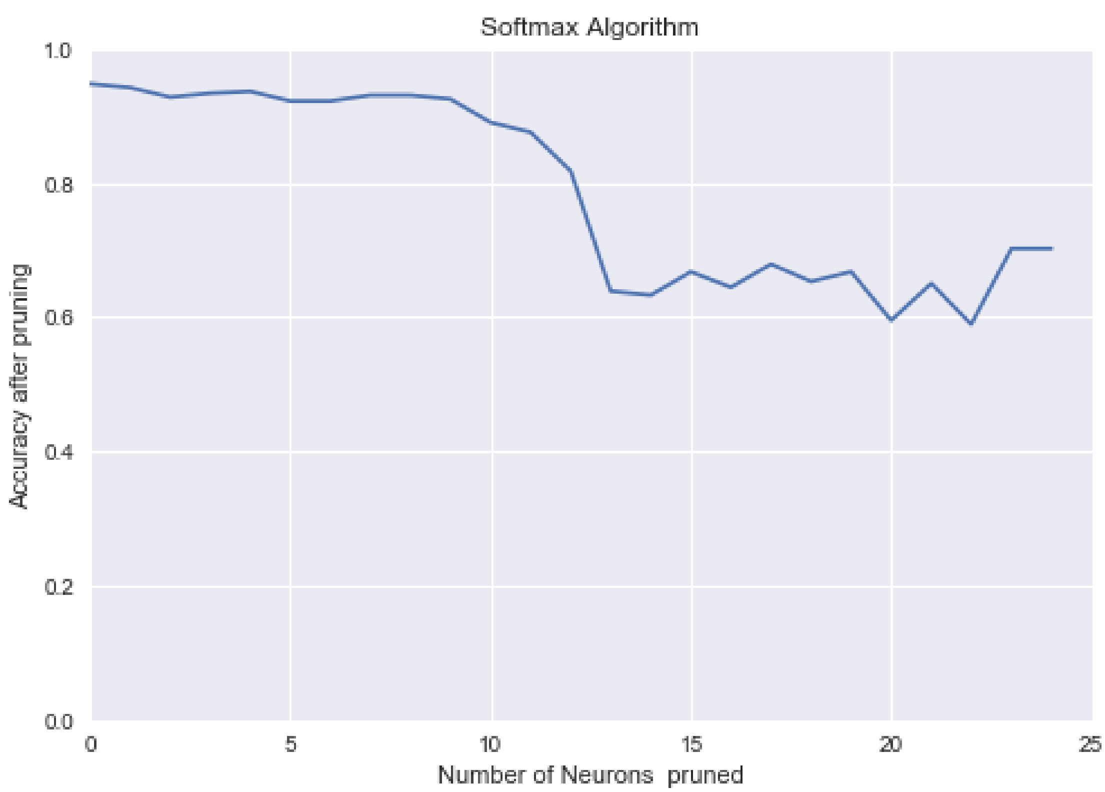


5 Run Softmax pruning Algorithm

```
In [11]: 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) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.3278310298919678 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.94
Test after pruning= 0.93
Test after pruning= 0.93
Test after pruning= 0.94
Test after pruning= 0.92
Test after pruning= 0.92
Test after pruning= 0.93
```

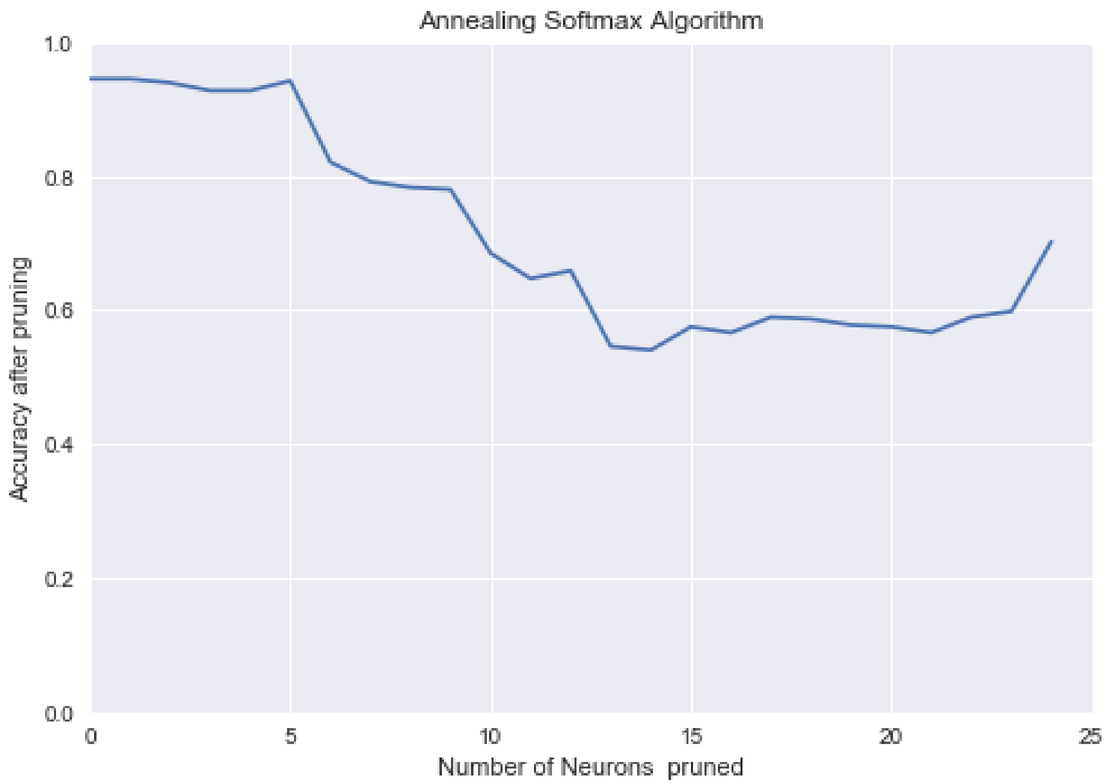
```
Test after pruning= 0.93
Test after pruning= 0.92
Test after pruning= 0.89
Test after pruning= 0.88
Test after pruning= 0.82
Test after pruning= 0.64
Test after pruning= 0.63
Test after pruning= 0.67
Test after pruning= 0.64
Test after pruning= 0.68
Test after pruning= 0.65
Test after pruning= 0.67
Test after pruning= 0.60
Test after pruning= 0.65
Test after pruning= 0.59
Test after pruning= 0.70
Test after pruning= 0.70
```



6 Run Annealing Softmax pruning Algorithm

```
In [12]: algo = AnnealingSoftmax([], [])
         Alg_name = 'Annealing Softmax Algorithm'
         path = './AnnealingSoftmax/'
         sys.path.append("./AnnealingSoftmax")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
Test fraction correct (NN-Score) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.32720184326171875 seconds
Finsh playing start pruining:
Test after pruning= 0.95
Test after pruning= 0.95
Test after pruning= 0.94
Test after pruning= 0.93
Test after pruning= 0.93
Test after pruning= 0.94
Test after pruning= 0.82
Test after pruning= 0.79
Test after pruning= 0.78
Test after pruning= 0.78
Test after pruning= 0.68
Test after pruning= 0.65
Test after pruning= 0.66
Test after pruning= 0.55
Test after pruning= 0.54
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.70
```

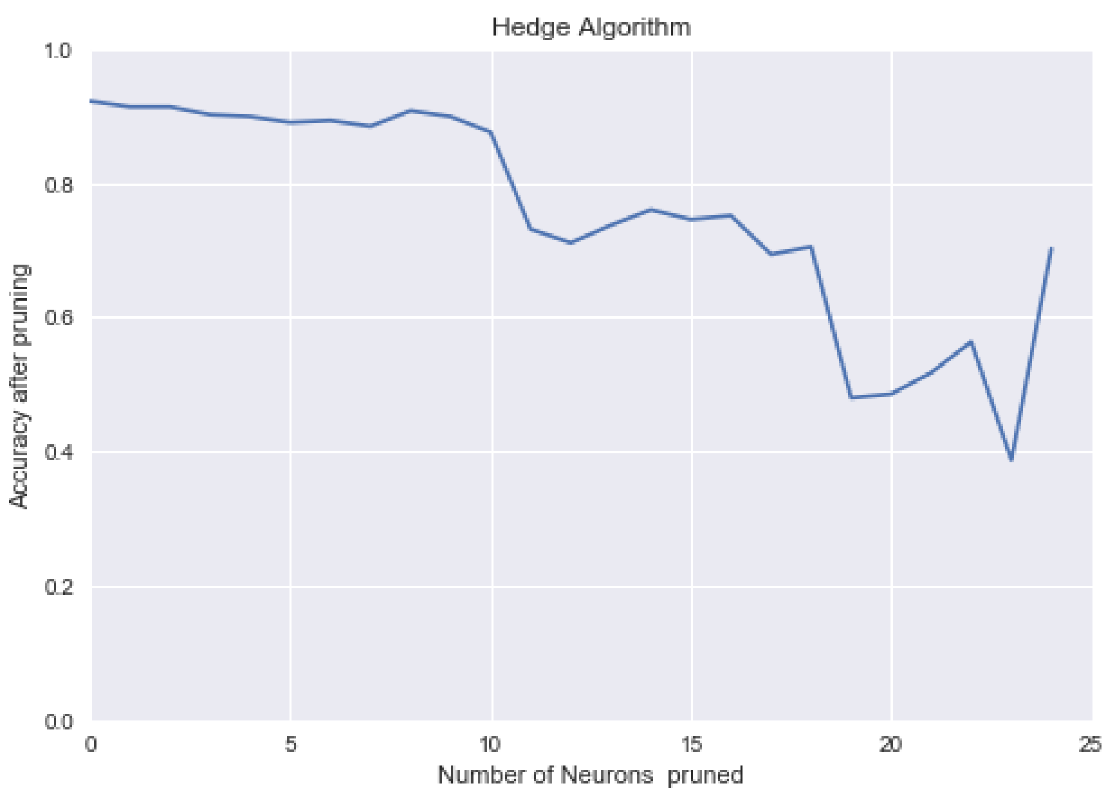


7 Run Hedge pruning Algorithm

```
In [13]: 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) = 0.14
Test fraction correct (NN-Accuracy) = 0.95
The time for running this method is 0.33802318572998047 seconds
Finsh playing start pruining:
Test after pruning= 0.92
Test after pruning= 0.91
Test after pruning= 0.91
Test after pruning= 0.90
Test after pruning= 0.90
Test after pruning= 0.89
Test after pruning= 0.89
Test after pruning= 0.88
```

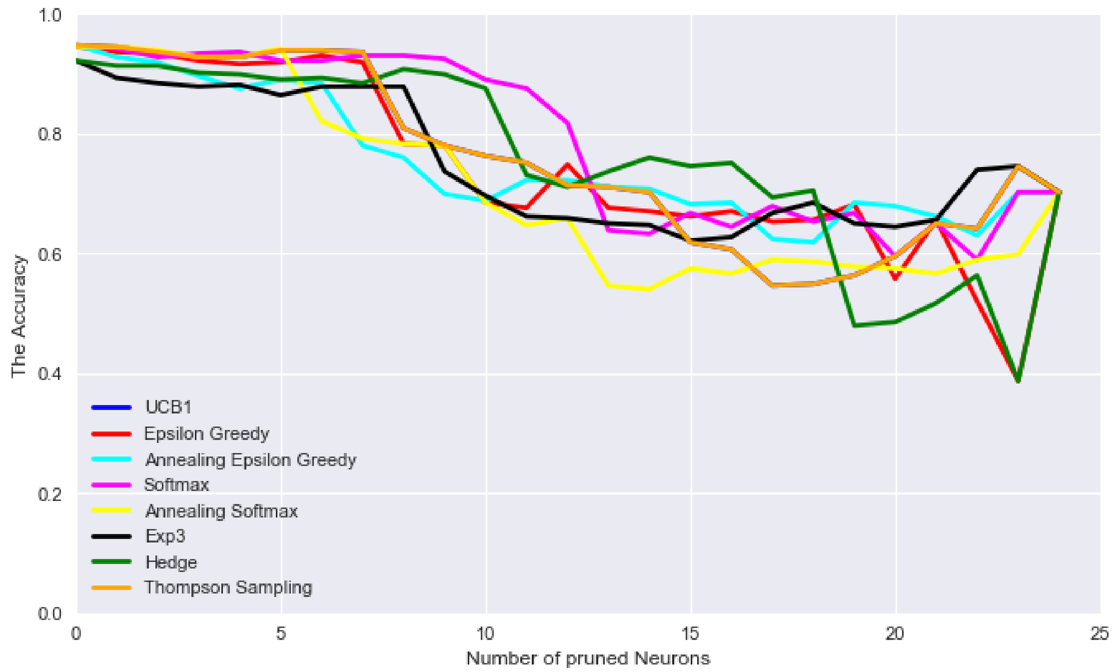
```
Test after pruning= 0.91
Test after pruning= 0.90
Test after pruning= 0.88
Test after pruning= 0.73
Test after pruning= 0.71
Test after pruning= 0.74
Test after pruning= 0.76
Test after pruning= 0.75
Test after pruning= 0.75
Test after pruning= 0.69
Test after pruning= 0.71
Test after pruning= 0.48
Test after pruning= 0.49
Test after pruning= 0.52
Test after pruning= 0.56
Test after pruning= 0.39
Test after pruning= 0.70
```



8 Compare the accuracy of the models

```
In [14]: 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 [15]: 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, 1])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```

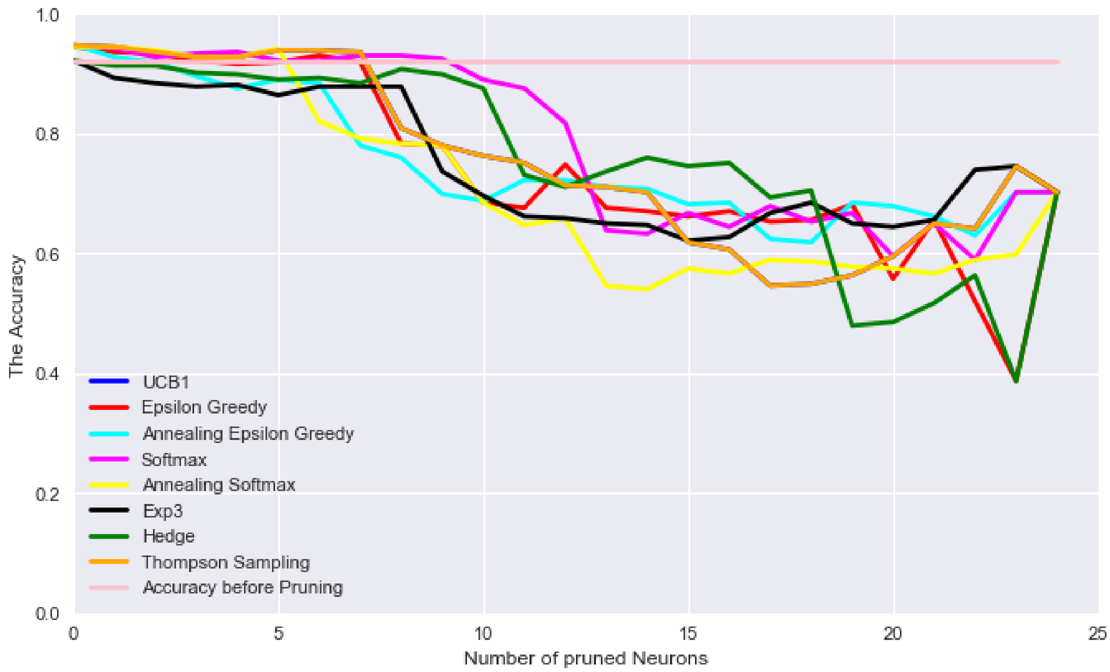



```
In [16]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #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 [17]:
```
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, 1])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

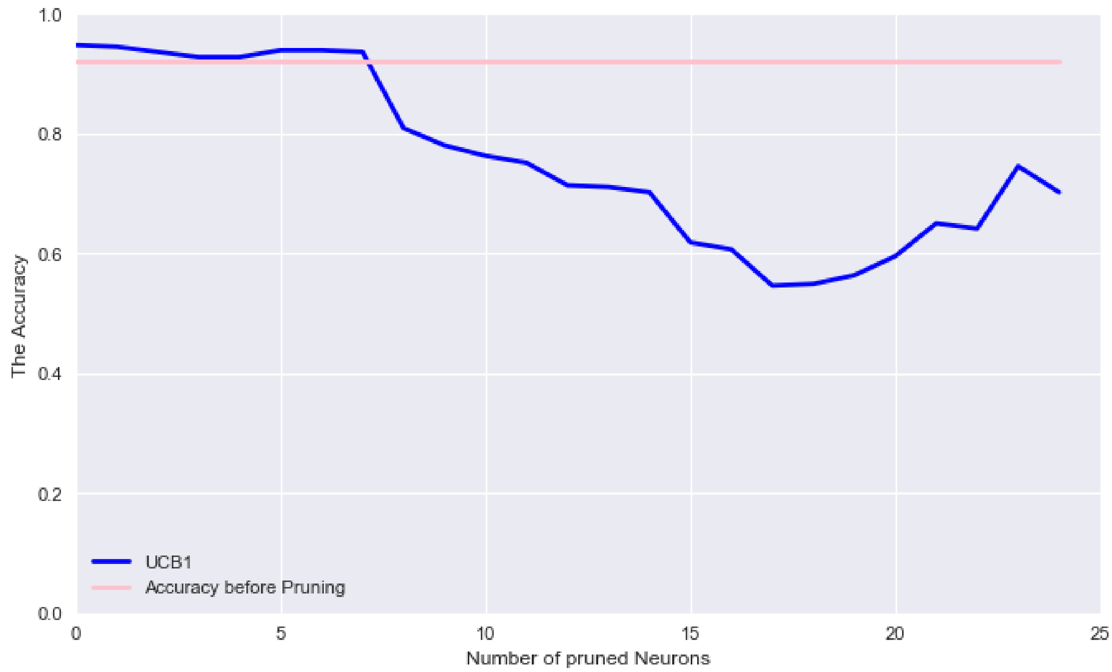



```
In [18]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #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 [19]: 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, 1])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```

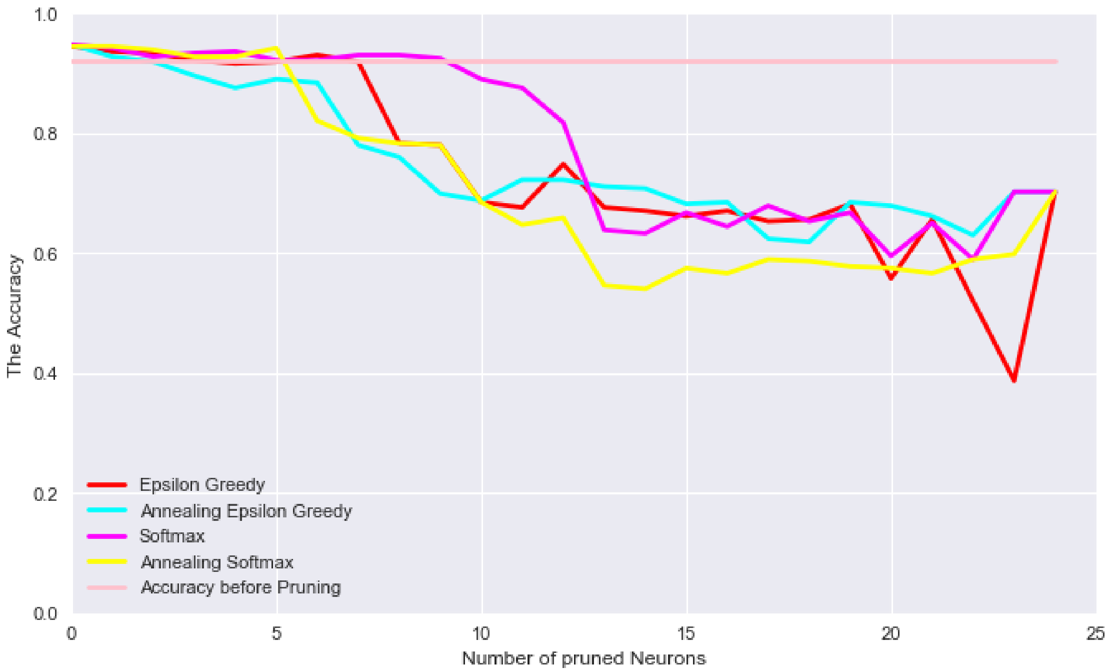



```
In [20]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


         #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 [21]: 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, 1])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

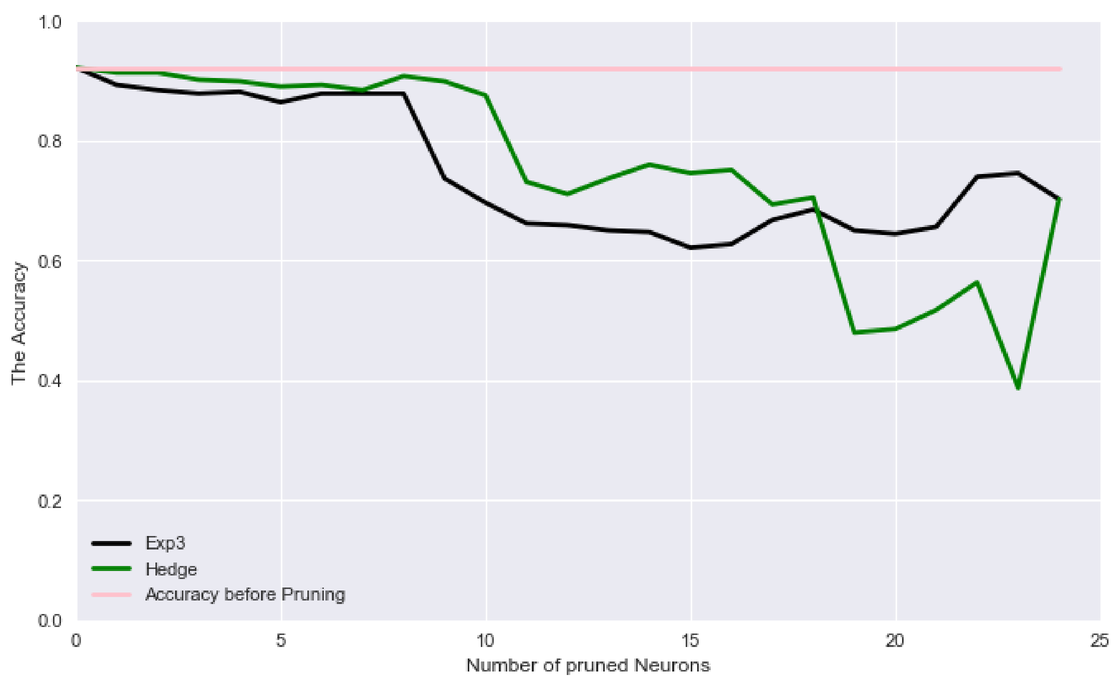


In [22]:
```
p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


#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 [23]: 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, 1])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```

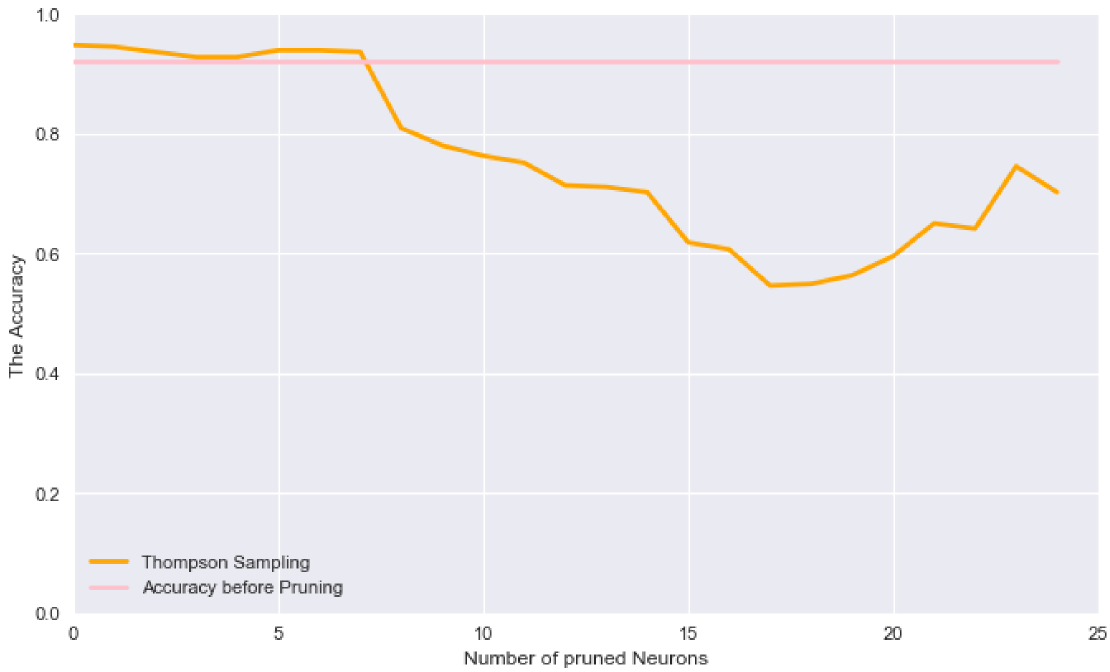



```
In [24]: p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)
```

```
#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 [25]: 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, 1])
         plt.xlabel('Number of pruned Neurons')
         plt.ylabel('The Accuracy')
         plt.grid(True)
         plt.show()
```



In [26]:
```
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
```