

RemoveNodeMAB-Different_bandit

March 23, 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)
```

0.1 Load BOKEH Lib.

```
In [2]: 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 [3]: X_train = np.load('./valley/X_train.npy')
y_train = np.load('./valley/y_train.npy')
X_test = np.load('./valley/X_test.npy')
y_test = np.load('./valley/y_test.npy')
X_deploy = np.load('./valley/X_deploy.npy')
y_deploy = np.load('./valley/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 484
Number of validation examples 122
Number of testing examples 606
```

```
In [4]: exec(open("core.py").read()) # python 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 [5]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.22579097747802734 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

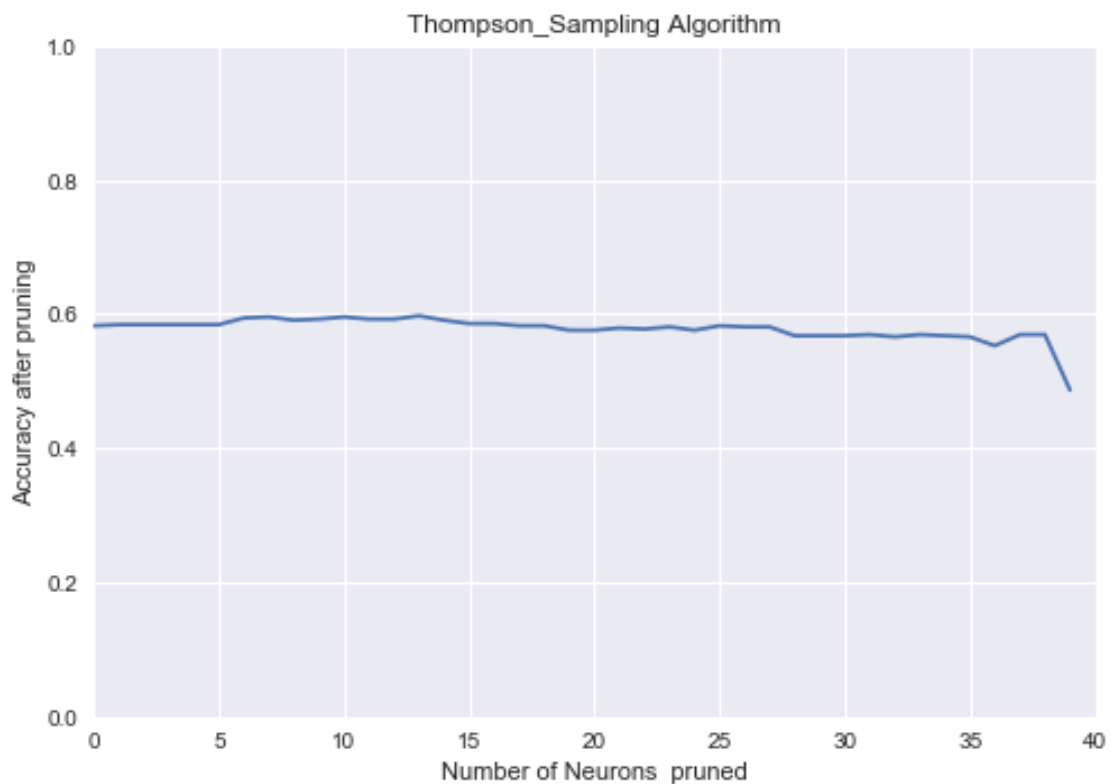
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58
 Test after pruning= 0.58
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.55
 Test after pruning= 0.57
 Test after pruning= 0.57
 Test after pruning= 0.49

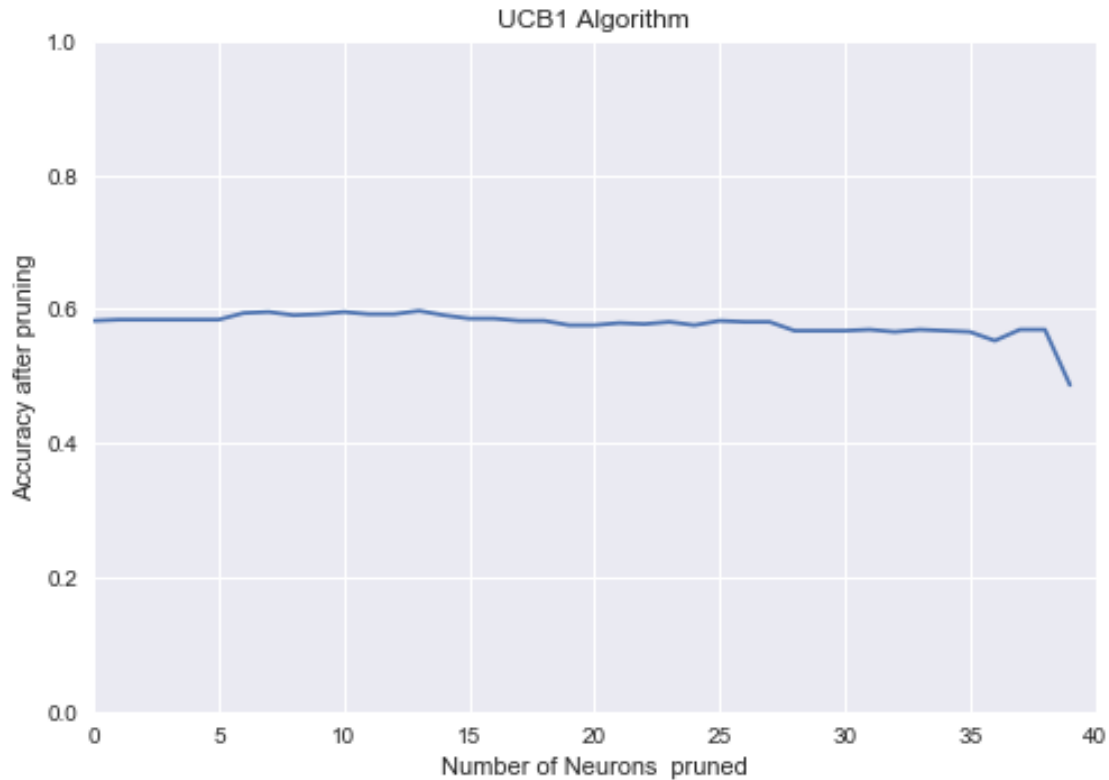


1.2 Run UCB1 pruning Algorithm

```

In [6]: algo = UCB1([], [])
        Alg_name = 'UCB1 Algorithm'
        path = './UCB1/'
        sys.path.append("./UCB1")
        exec(open("mnist_cnnFORTESTING.py").read())
  
```

Test fraction correct (NN-Score) = 0.70
Test fraction correct (NN-Accuracy) = 0.58
The time for running this method is 0.230607271194458 seconds
Finsh playing start pruning:
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.55
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.49



2 Run Annealing Epsilon Greedy pruning Algorithm

```
In [7]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.21454095840454102 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.59

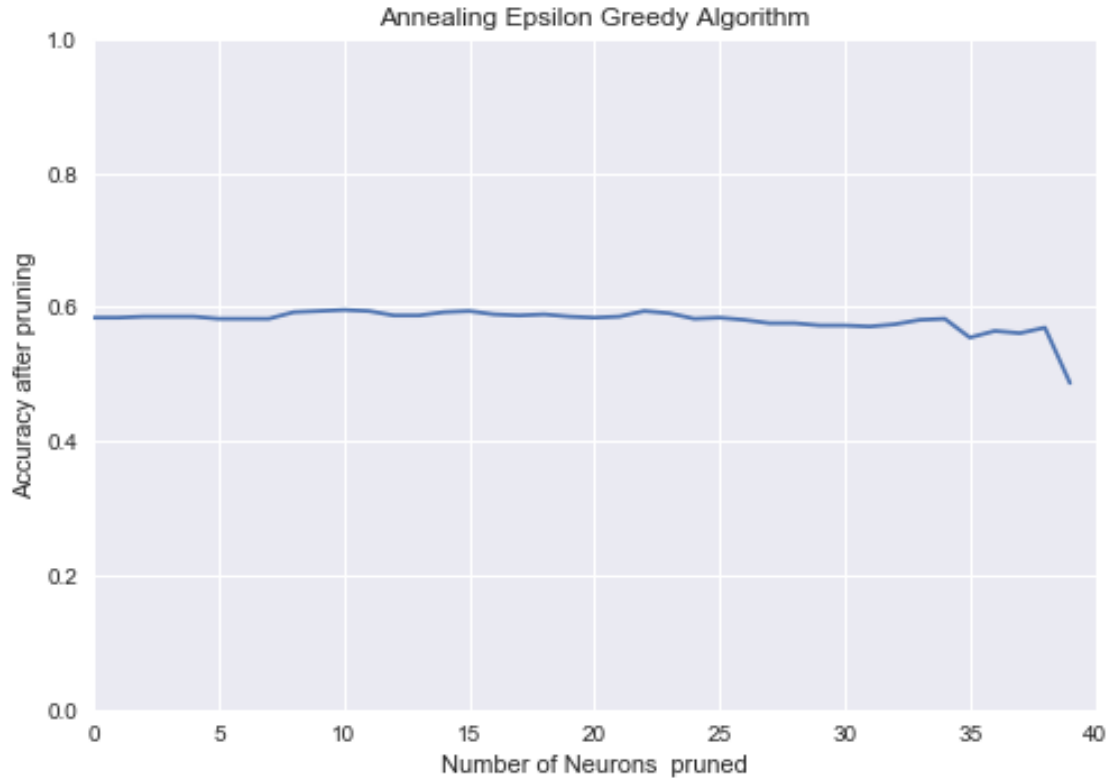
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.55
Test after pruning= 0.56
Test after pruning= 0.56
Test after pruning= 0.57
Test after pruning= 0.49



3 Run Epsilon Greedy pruning Algorithm

```
In [8]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.22977185249328613 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

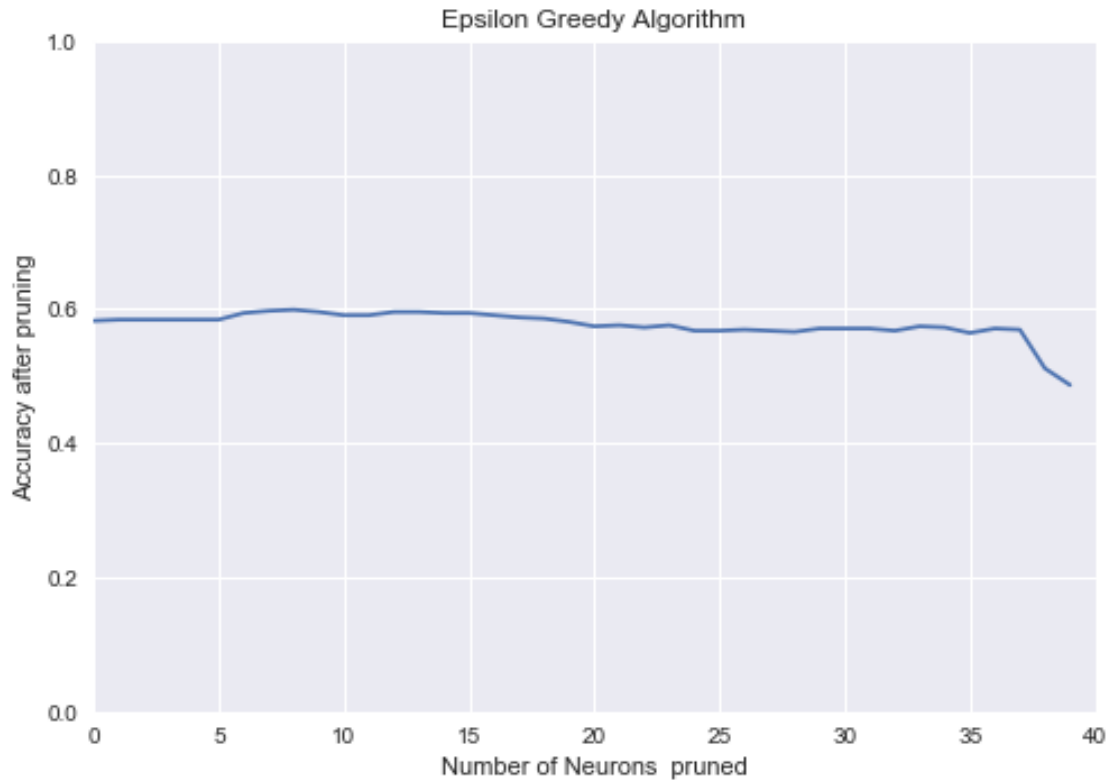
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.60
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.56
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.51
Test after pruning= 0.49



4 Run Exp3 pruning Algorithm

```
In [9]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.21202373504638672 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

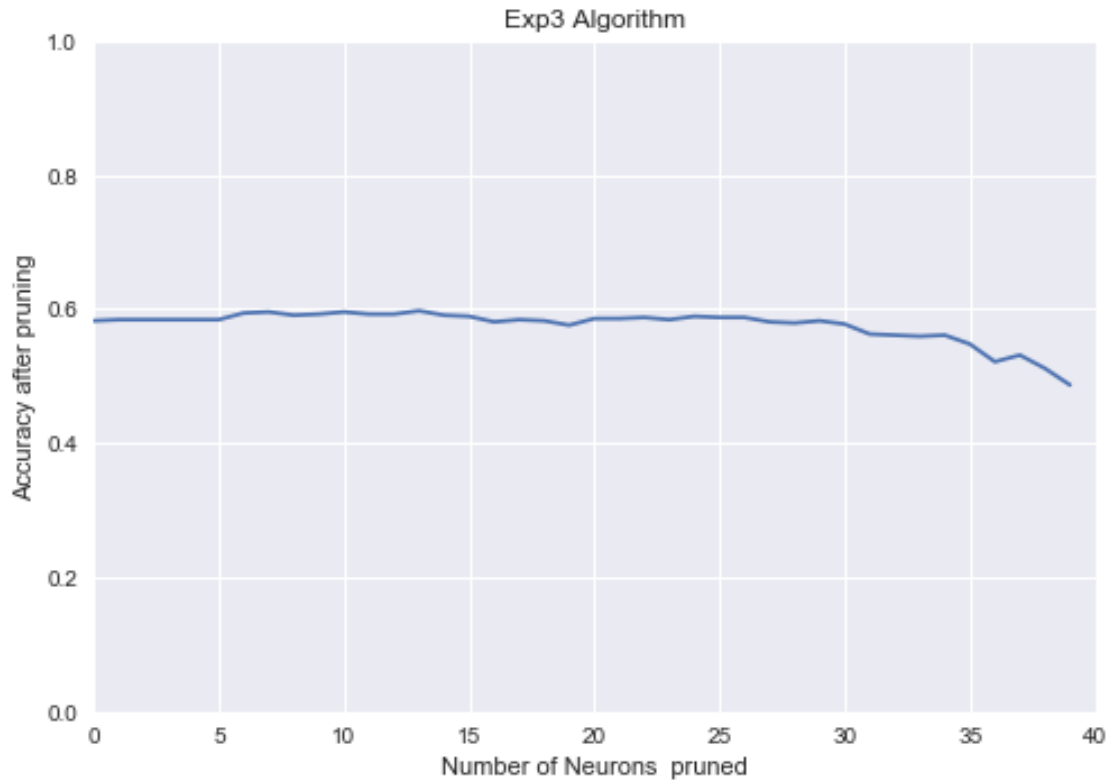
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.56
Test after pruning= 0.56
Test after pruning= 0.56
Test after pruning= 0.56
Test after pruning= 0.55
Test after pruning= 0.52
Test after pruning= 0.53
Test after pruning= 0.51
Test after pruning= 0.49



5 Run Softmax pruning Algorithm

```
In [10]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.23189401626586914 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

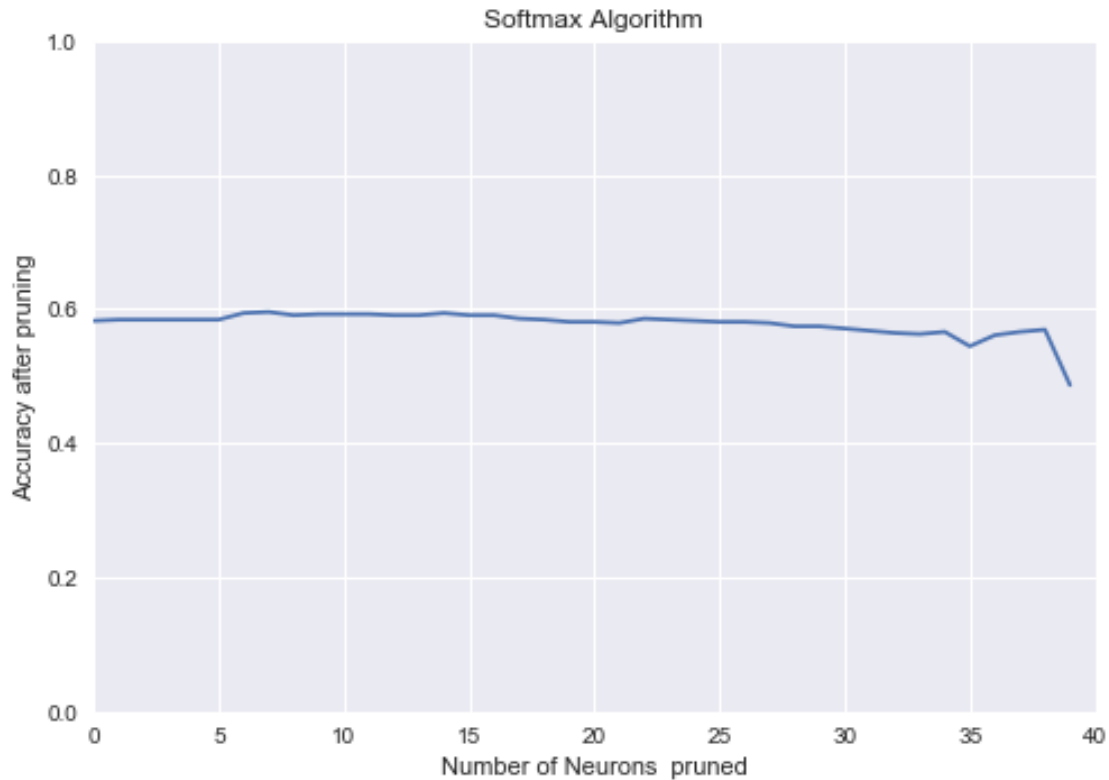
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.56
Test after pruning= 0.56
Test after pruning= 0.57
Test after pruning= 0.54
Test after pruning= 0.56
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.49



6 Run Annealing Softmax pruning Algorithm

```
In [11]: algo = AnnealingSoftmax([], [])
         Alg_name = 'Annealing Softmax Algorithm'
         path = './AnnealingSoftmax/'
         sys.path.append("./AnnealingSoftmax")
         exec(open("mnist_cnnFORTESTING.py").read())
```

Test fraction correct (NN-Score) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.21093082427978516 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.59

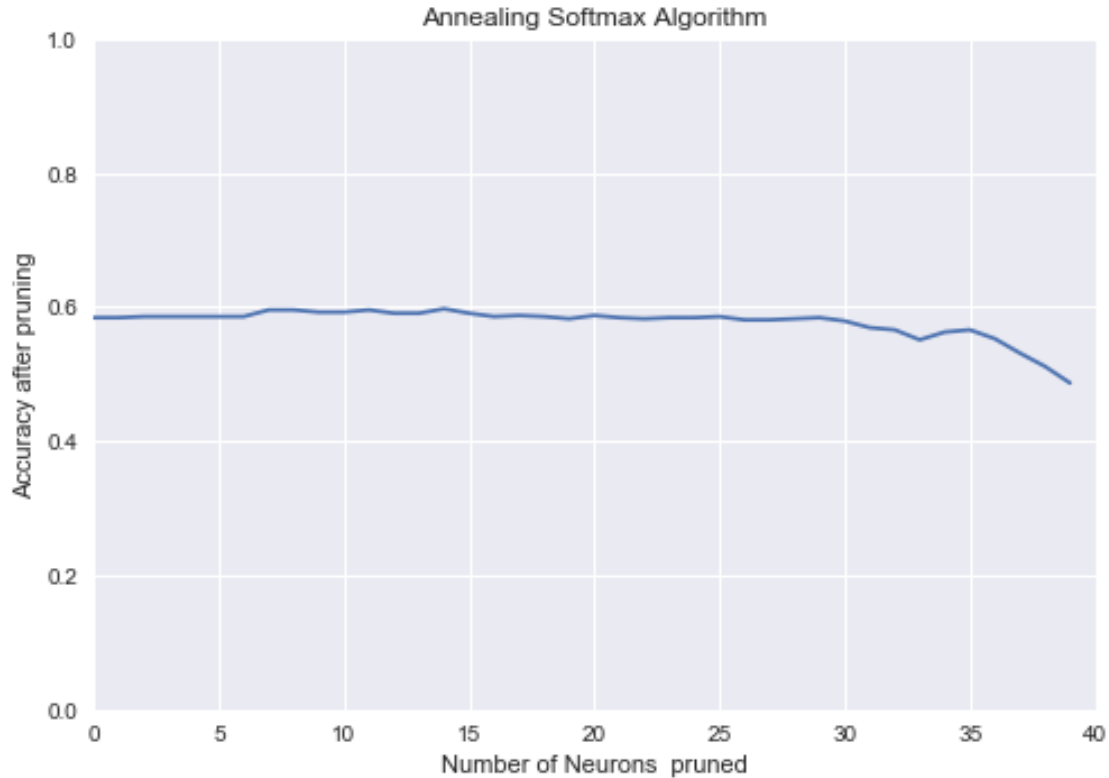
Test after pruning= 0.59

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.60

Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.55
Test after pruning= 0.56
Test after pruning= 0.57
Test after pruning= 0.55
Test after pruning= 0.53
Test after pruning= 0.51
Test after pruning= 0.49



7 Run Hedge pruning Algorithm

```
In [12]: 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) = 0.70

Test fraction correct (NN-Accuracy) = 0.58

The time for running this method is 0.22306513786315918 seconds

Finsh playing start pruning:

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.58

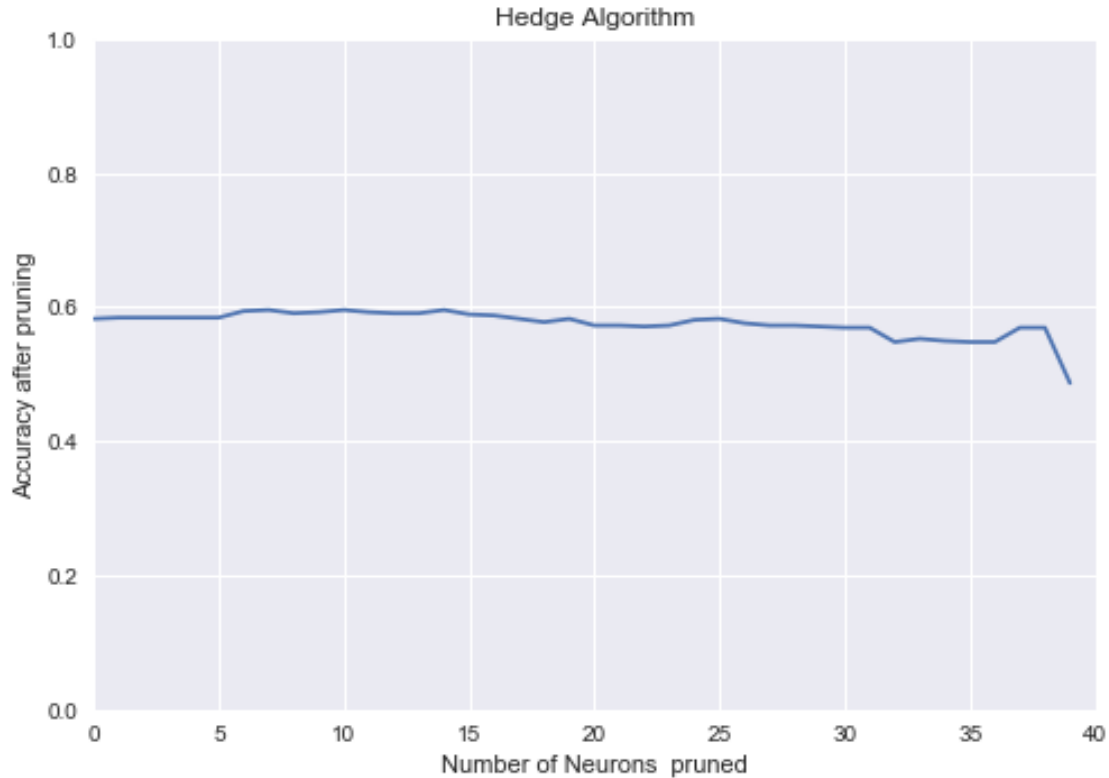
Test after pruning= 0.58

Test after pruning= 0.58

Test after pruning= 0.59

Test after pruning= 0.60

Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.60
Test after pruning= 0.59
Test after pruning= 0.59
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.58
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.55
Test after pruning= 0.57
Test after pruning= 0.57
Test after pruning= 0.49



8 Compare the accuracy of the models

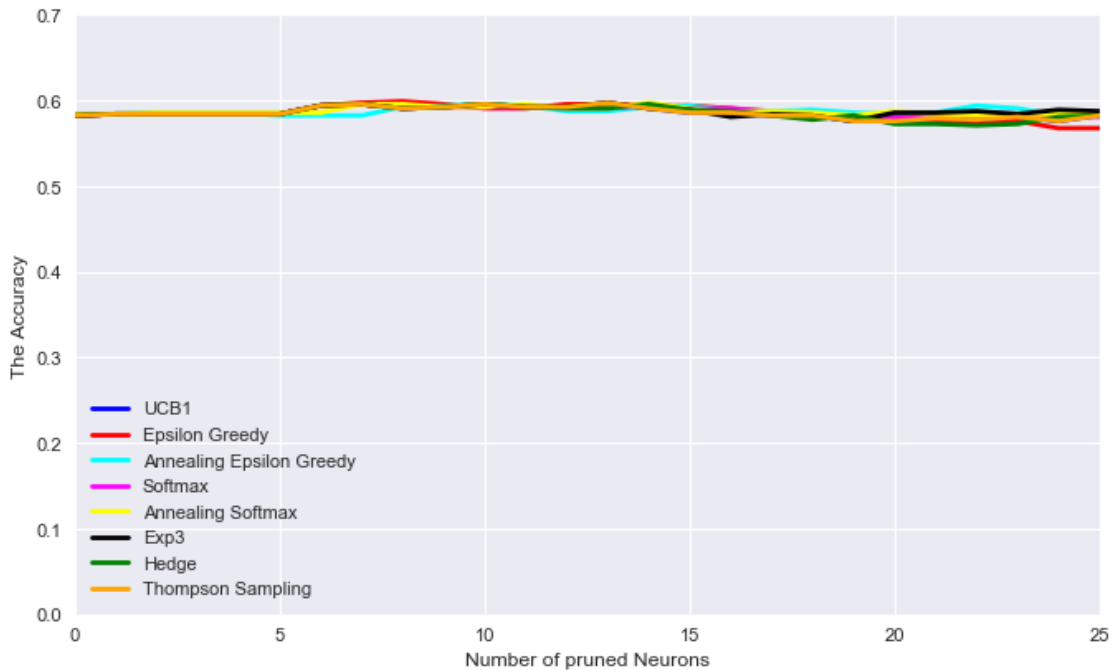
```
In [13]: ucb1 = np.load('./UCB1/AccuracyAftrePrune.npy')
        EpsilonGreedy = np.load('./EpsilonGreedy/AccuracyAftrePrune.npy')
        AnnealingEpsilonGreedy = np.load('./AnnealingEpsilonGreedy/AccuracyAftrePrune.npy')
        Softmax = np.load('./Softmax/AccuracyAftrePrune.npy')
        AnnealingSoftmax = np.load('./AnnealingSoftmax/AccuracyAftrePrune.npy')
        Exp3 = np.load('./Exp3/AccuracyAftrePrune.npy')
        Hedge = np.load('./Hedge/AccuracyAftrePrune.npy')
        ThompsonSampling = np.load('./Thompson_Sampling/AccuracyAftrePrune.npy')
        Accuracy = np.load('AccuracyBeforePruning.npy')

In [14]: 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="-", label="Annealing")
        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 [15]: 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="red")
p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

#p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue")
p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue", line_width=2)

#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

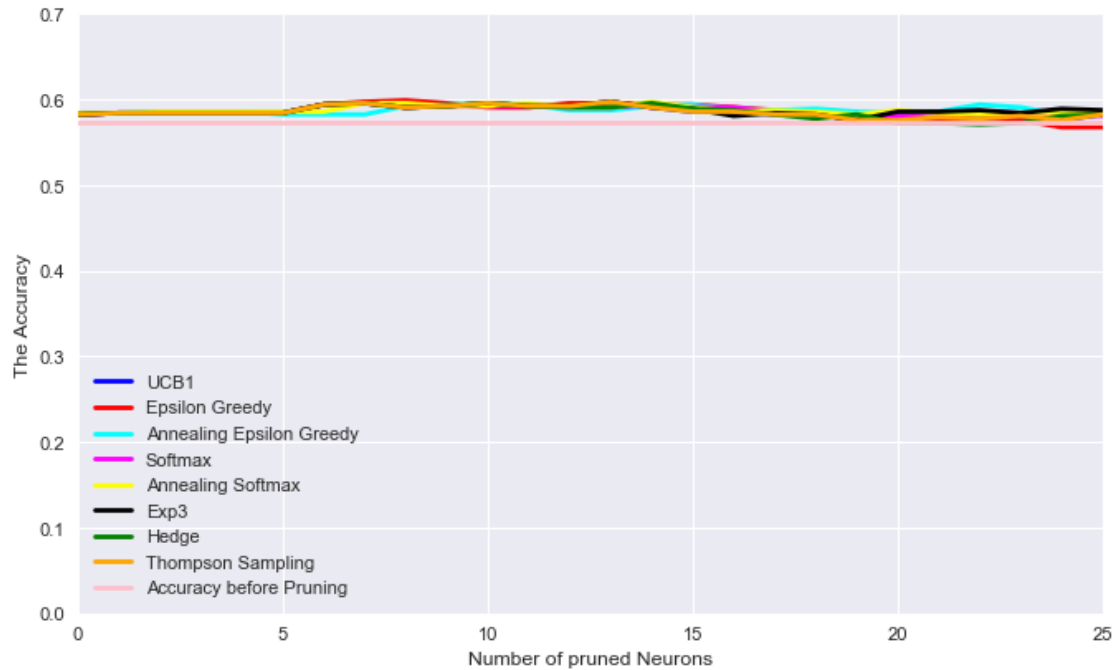
```

8.1 Comparing All algorithms with the model before pruning

```

In [16]: 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="-", label="Annealing Epsilon")
plt.plot(ind, Softmax, color="magenta", linewidth=2.5, linestyle="-", label="Softmax")
plt.plot(ind, AnnealingSoftmax, color="yellow", linewidth=2.5, linestyle="-", label="Annealing Softmax")
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="Thompson Sampling")
plt.plot(ind, Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before pruning")
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 [17]: 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="red")
p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

#p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue")
p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue", line_width=2)

#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=0.5)
p1.line(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", line_width=2)

#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

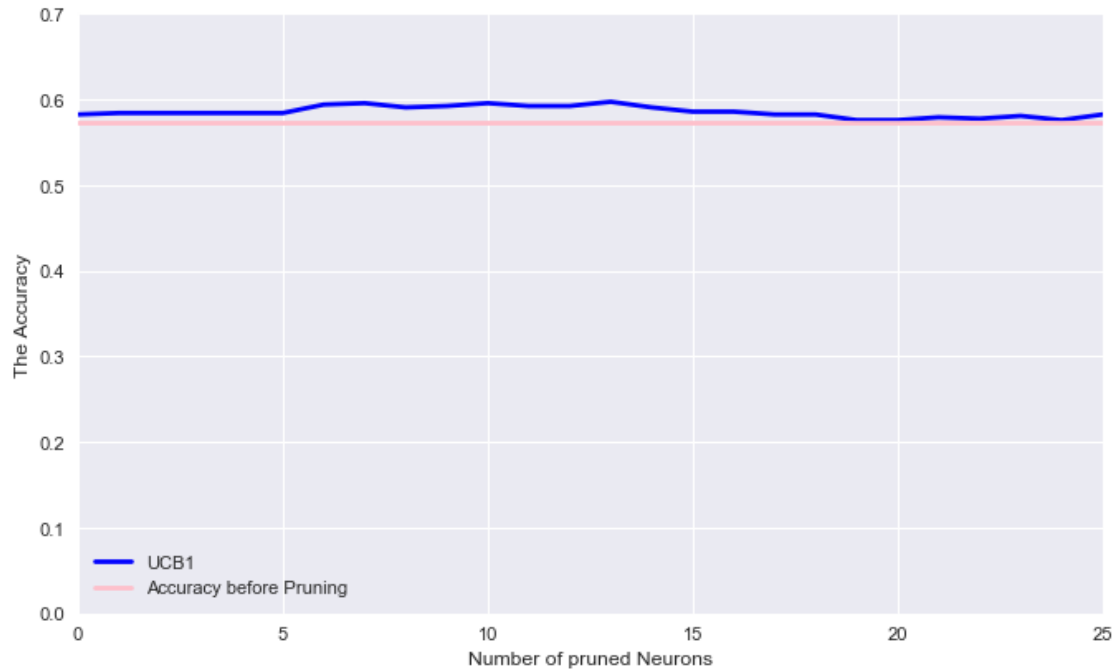
```

8.2 UCB1

```

In [18]: 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 [19]: 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=2)
#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
```

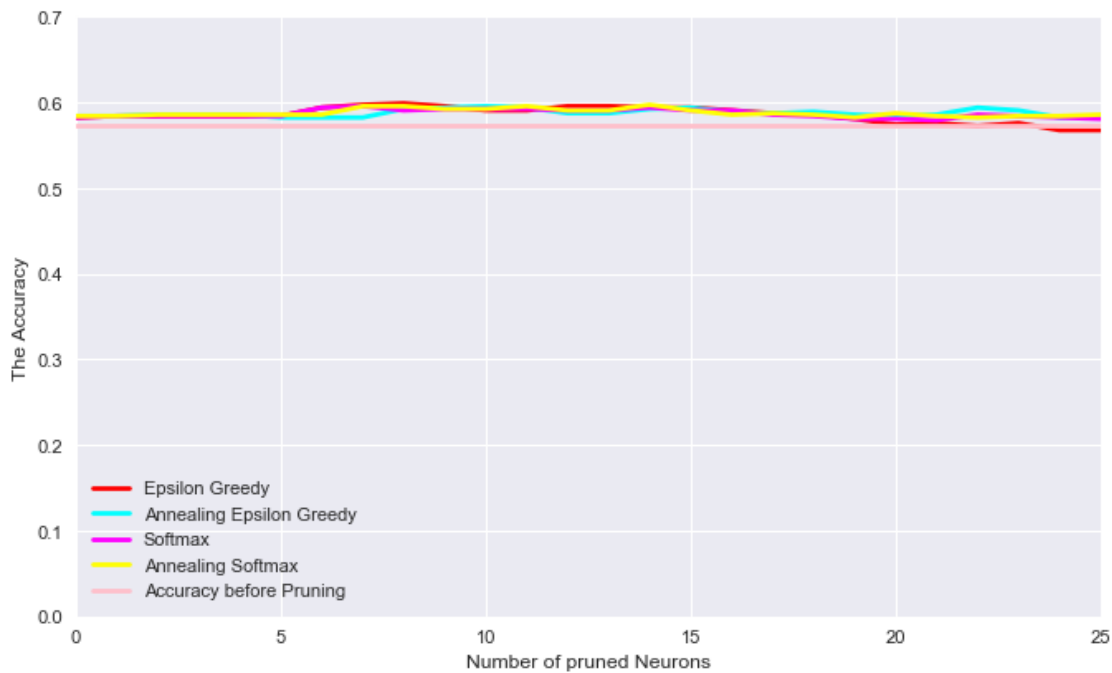
8.3 Epsilon greedy and Softmax

```
In [20]: 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="-", label="AnnealingEpsilon")
plt.plot(ind, Softmax, color="magenta", linewidth=2.5, linestyle="-", label="Softmax")
plt.plot(ind, AnnealingSoftmax, color="yellow", linewidth=2.5, linestyle="-", label="AnnealingSoftmax")
```

```

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 [21]: 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="red", line_width=2)
p1.line(ind, EpsilonGreedy, legend="Epsilon Greedy", line_color="red", line_width=2)

#p1.ellipse(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue", line_width=2)
p1.line(ind, AnnealingEpsilonGreedy, legend="Annealing Epsilon Greedy", line_color="blue", line_width=2)

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

#p1.arc(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", end_angle=90, line_width=2)
p1.line(ind, AnnealingSoftmax, legend="Annealing Softmax", line_color="grey", line_width=2)

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

```

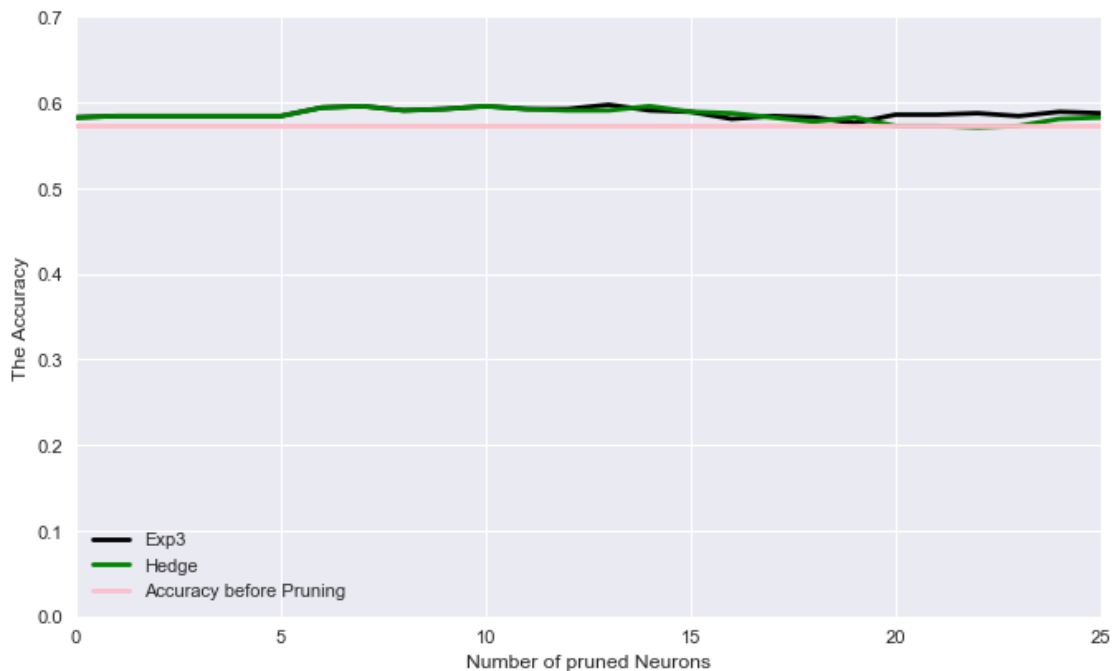
```
p1.title.align = "center"
```

```
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 [22]: 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 [23]: 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=2)
#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

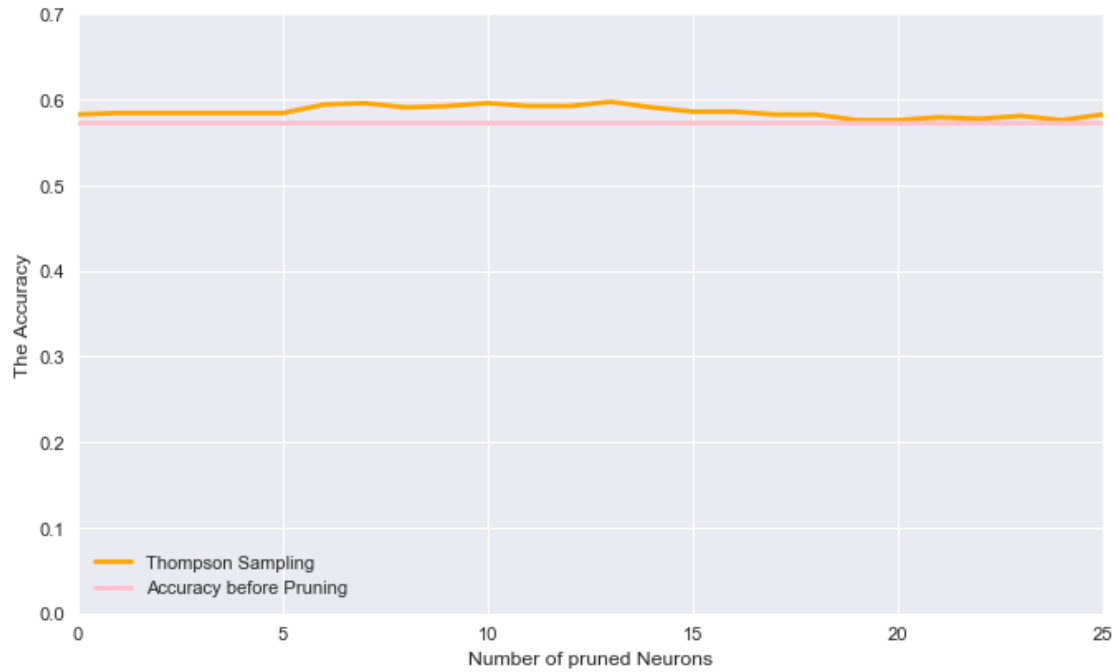
```

9 Thompson Sampling

```

In [24]: 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="Thompson Sampling")
         plt.plot(ind, Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before pruning")
         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.square_cross(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink")
p1.line(ind, ThompsonSampling, legend="Thompson Sampling", line_color="pink", line_width=2)

p1.line(ind, Acc, legend="Accuracy", line_dash=(4, 4), line_color="orange", line_width=2)
#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
```