RemoveNodeMAB-Different_bandit

March 22, 2017

Compute the performance of MAB methods

```
In [1]: import numpy as np
        import time
        import sys
        import matplotlib.pyplot as plt
        from sklearn import metrics
        %matplotlib inline
        #plt.rcParams['figure.figsize'] = (15, 6)
```

0.1 Load BOKEH libariry

```
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('X_train.npy')
        y_train = np.load('y_train.npy')
        X_test = np.load('X_test.npy')
        y_test = np.load('y_test.npy')
        X_deploy = np.load('X_deploy.npy')
        y_deploy = np.load('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 2944
Number of validation examples 736
Number of testing examples 921
```

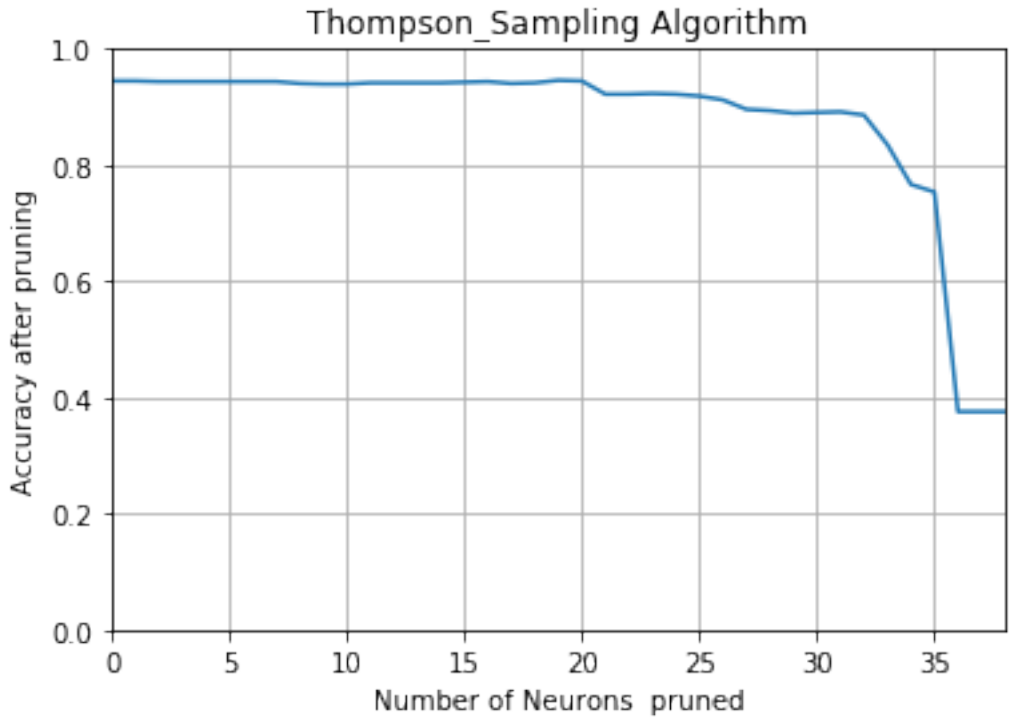
```
In [4]: exec(open("core.py").read())  # pyhton 3x
        #exec(compile(open('core.py', "rb").read(), 'core.py', 'exec'))

        #execfile("/Users/salemameen/Desktop/banditsbook/python/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())
```

```
Using Theano backend.


736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.7069087028503418 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.942453853924
Test accuracy after pruning: 0.943539630254
Test accuracy after pruning: 0.940282302299
Test accuracy after pruning: 0.941368078629
Test accuracy after pruning: 0.945711183949
Test accuracy after pruning: 0.944625407619
Test accuracy after pruning: 0.921824104688
Test accuracy after pruning: 0.921824104688
Test accuracy after pruning: 0.922909880435
Test accuracy after pruning: 0.921824104105
Test accuracy after pruning: 0.918566775115
Test accuracy after pruning: 0.912052117134
```

```
Test accuracy after pruning: 0.895765472636
Test accuracy after pruning: 0.893593919394
Test accuracy after pruning: 0.889250814073
Test accuracy after pruning: 0.890336590403
Test accuracy after pruning: 0.891422366734
Test accuracy after pruning: 0.885993486119
Test accuracy after pruning: 0.834961998605
Test accuracy after pruning: 0.766558089228
Test accuracy after pruning: 0.753528772684
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

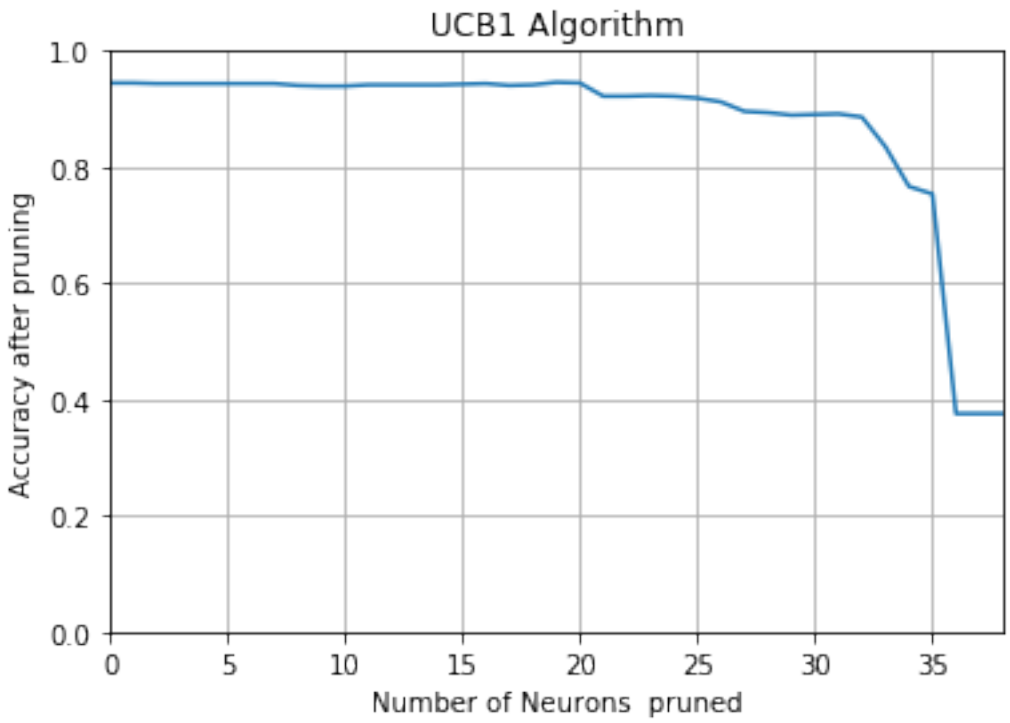


1.2 Run UCB1 pruning Algorithm

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

```
736 test samples
Test score: 0.338247084349
```

```
Test accuracy: 0.944625407166
The time for running this method is 1.6735870838165283 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.941368077593
Test accuracy after pruning: 0.942453853924
Test accuracy after pruning: 0.943539630254
Test accuracy after pruning: 0.940282302299
Test accuracy after pruning: 0.941368078629
Test accuracy after pruning: 0.945711183949
Test accuracy after pruning: 0.944625407619
Test accuracy after pruning: 0.921824104688
Test accuracy after pruning: 0.921824104688
Test accuracy after pruning: 0.922909880435
Test accuracy after pruning: 0.921824104105
Test accuracy after pruning: 0.918566775115
Test accuracy after pruning: 0.912052117134
Test accuracy after pruning: 0.895765472636
Test accuracy after pruning: 0.893593919394
Test accuracy after pruning: 0.889250814073
Test accuracy after pruning: 0.890336590403
Test accuracy after pruning: 0.891422366734
Test accuracy after pruning: 0.885993486119
Test accuracy after pruning: 0.834961998605
Test accuracy after pruning: 0.766558089228
Test accuracy after pruning: 0.753528772684
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

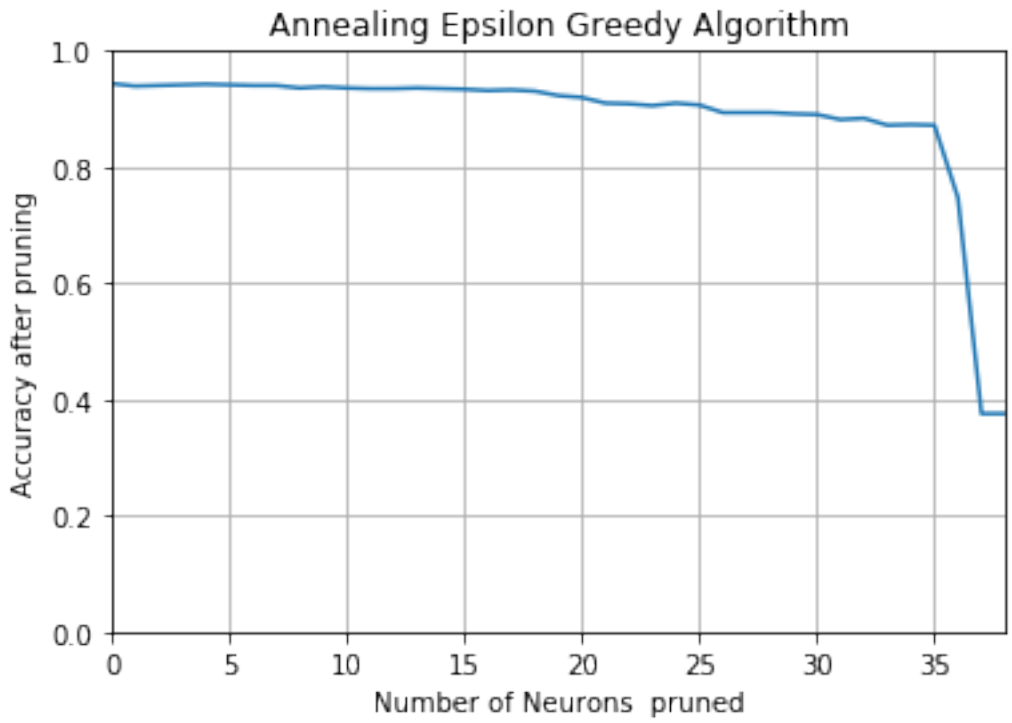


2 Run Annealing Epsilon Greedy pruning Algorithm

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

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.6822741031646729 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.939196524933
Test accuracy after pruning: 0.940282301263
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.942453854506
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.935939196526
Test accuracy after pruning: 0.938110749186
```

```
Test accuracy after pruning: 0.935939196526
Test accuracy after pruning: 0.934853420195
Test accuracy after pruning: 0.934853420195
Test accuracy after pruning: 0.935939196526
Test accuracy after pruning: 0.934853419613
Test accuracy after pruning: 0.933767643283
Test accuracy after pruning: 0.931596091658
Test accuracy after pruning: 0.932681867988
Test accuracy after pruning: 0.930510314293
Test accuracy after pruning: 0.922909879853
Test accuracy after pruning: 0.919652550862
Test accuracy after pruning: 0.909880564927
Test accuracy after pruning: 0.908794788015
Test accuracy after pruning: 0.905537459607
Test accuracy after pruning: 0.909880563892
Test accuracy after pruning: 0.906623234902
Test accuracy after pruning: 0.893593918941
Test accuracy after pruning: 0.893593919976
Test accuracy after pruning: 0.893593919976
Test accuracy after pruning: 0.891422367316
Test accuracy after pruning: 0.890336590403
Test accuracy after pruning: 0.881650379763
Test accuracy after pruning: 0.883821932423
Test accuracy after pruning: 0.871878393828
Test accuracy after pruning: 0.872964170158
Test accuracy after pruning: 0.871878392792
Test accuracy after pruning: 0.747014114704
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

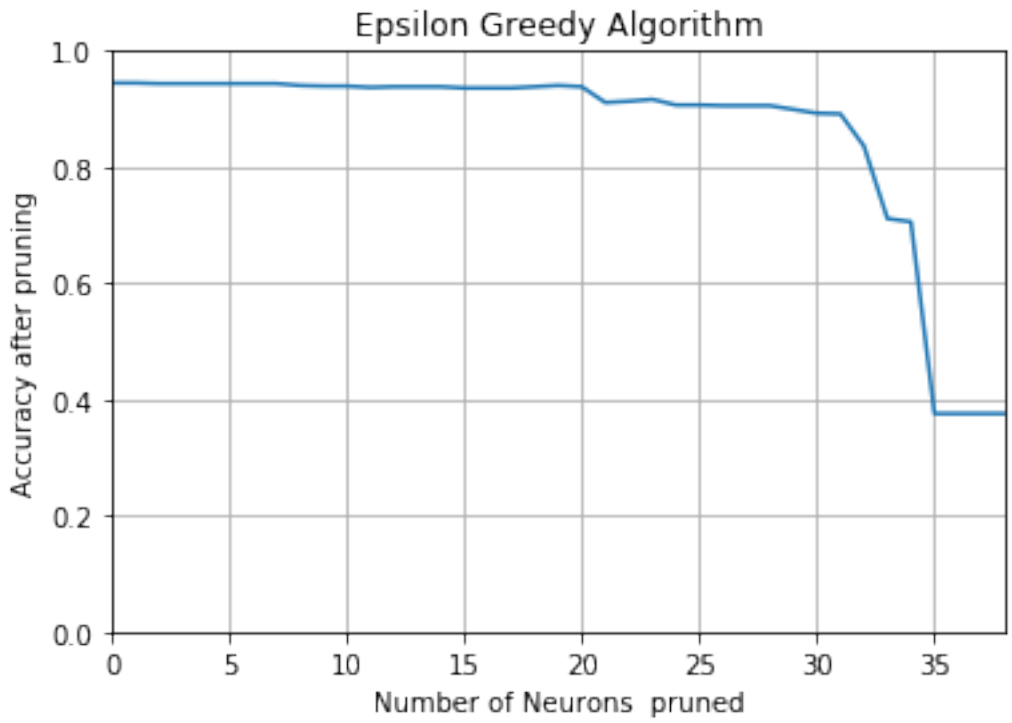


3 Run Epsilon Greedy pruning Algorithm

```
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())
```

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.5948808193206787 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
```

```
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.937024972273
Test accuracy after pruning: 0.938110748603
Test accuracy after pruning: 0.938110748603
Test accuracy after pruning: 0.938110748603
Test accuracy after pruning: 0.935939196979
Test accuracy after pruning: 0.935939196396
Test accuracy after pruning: 0.935939196396
Test accuracy after pruning: 0.938110748474
Test accuracy after pruning: 0.940282301716
Test accuracy after pruning: 0.938110749056
Test accuracy after pruning: 0.910966340675
Test accuracy after pruning: 0.913137893335
Test accuracy after pruning: 0.916395222908
Test accuracy after pruning: 0.906623235355
Test accuracy after pruning: 0.906623235355
Test accuracy after pruning: 0.905537459025
Test accuracy after pruning: 0.905537459025
Test accuracy after pruning: 0.905537459025
Test accuracy after pruning: 0.899022801044
Test accuracy after pruning: 0.892508143064
Test accuracy after pruning: 0.891422366734
Test accuracy after pruning: 0.836047774935
Test accuracy after pruning: 0.711183496265
Test accuracy after pruning: 0.705754614614
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

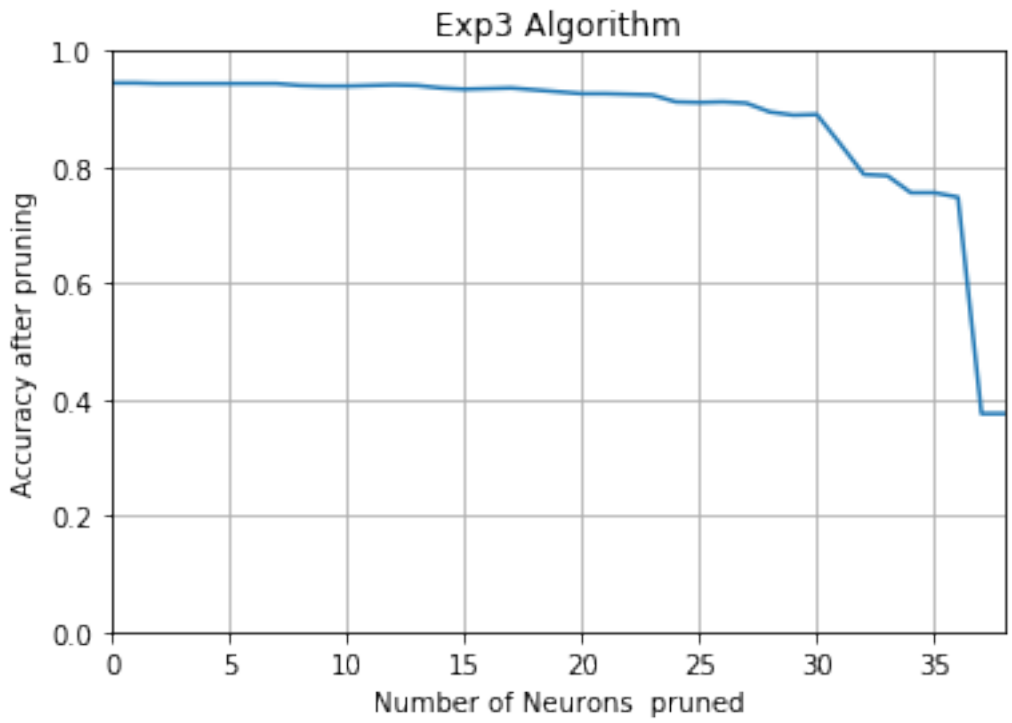


4 Run Exp3 pruning Algorithm

```
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())
```

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.6179580688476562 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
```

```
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.935939196526
Test accuracy after pruning: 0.933767643865
Test accuracy after pruning: 0.934853420195
Test accuracy after pruning: 0.935939196526
Test accuracy after pruning: 0.932681867535
Test accuracy after pruning: 0.929424537963
Test accuracy after pruning: 0.926167208972
Test accuracy after pruning: 0.926167208972
Test accuracy after pruning: 0.925081433678
Test accuracy after pruning: 0.923995657348
Test accuracy after pruning: 0.912052117134
Test accuracy after pruning: 0.910966340804
Test accuracy after pruning: 0.912052116552
Test accuracy after pruning: 0.909880563892
Test accuracy after pruning: 0.894679695724
Test accuracy after pruning: 0.889250814073
Test accuracy after pruning: 0.890336590403
Test accuracy after pruning: 0.83930510289
Test accuracy after pruning: 0.787187840082
Test accuracy after pruning: 0.785016287422
Test accuracy after pruning: 0.755700325927
Test accuracy after pruning: 0.755700325345
Test accuracy after pruning: 0.748099891617
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

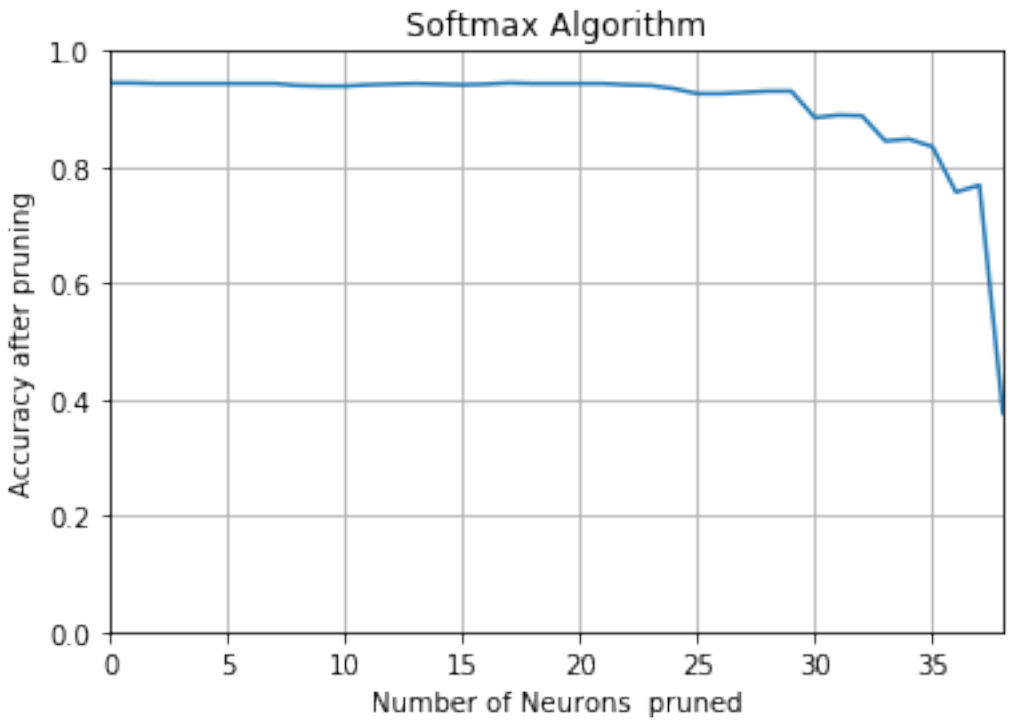


5 Run Softmax pruning Algorithm

```
In [10]: temperature = 0.9
         algo = Softmax(temperature, [], [])
         Alg_name = 'Softmax Algorithm'
         path = './Softmax/'
         sys.path.append("./Softmax")
         exec(open("mnist_cnnFORTESTING.py").read())
```

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.5992538928985596 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
```

```
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.942453854506
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.942453854506
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.942453854506
Test accuracy after pruning: 0.944625406584
Test accuracy after pruning: 0.943539630254
Test accuracy after pruning: 0.943539630254
Test accuracy after pruning: 0.943539631289
Test accuracy after pruning: 0.943539631289
Test accuracy after pruning: 0.941368078629
Test accuracy after pruning: 0.940282302299
Test accuracy after pruning: 0.934853420648
Test accuracy after pruning: 0.926167209425
Test accuracy after pruning: 0.926167209425
Test accuracy after pruning: 0.928338762086
Test accuracy after pruning: 0.930510314746
Test accuracy after pruning: 0.930510314746
Test accuracy after pruning: 0.884907708753
Test accuracy after pruning: 0.889250814073
Test accuracy after pruning: 0.888165038779
Test accuracy after pruning: 0.844733985576
Test accuracy after pruning: 0.847991314566
Test accuracy after pruning: 0.834961998605
Test accuracy after pruning: 0.756786102257
Test accuracy after pruning: 0.768729641888
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

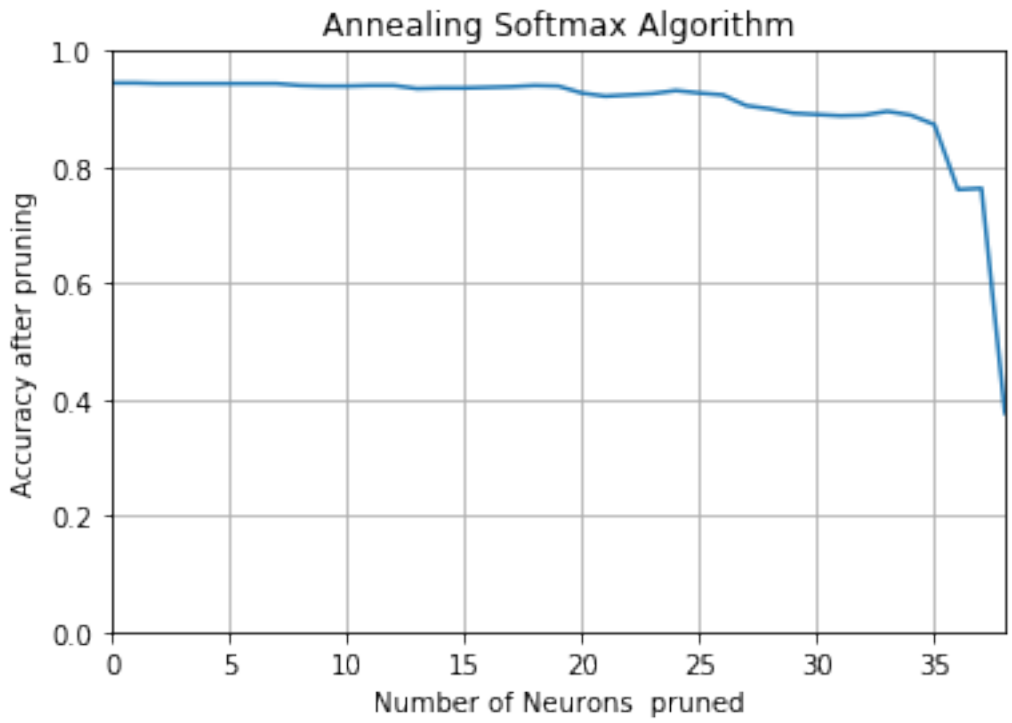


6 Run Annealing Softmax pruning Algorithm

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

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.604109764099121 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.939196525516
```

```
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.940282301263
Test accuracy after pruning: 0.934853420648
Test accuracy after pruning: 0.935939196979
Test accuracy after pruning: 0.935939196979
Test accuracy after pruning: 0.937024973309
Test accuracy after pruning: 0.938110749639
Test accuracy after pruning: 0.940282301263
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.927252985173
Test accuracy after pruning: 0.921824104558
Test accuracy after pruning: 0.923995657218
Test accuracy after pruning: 0.926167209878
Test accuracy after pruning: 0.931596091076
Test accuracy after pruning: 0.927252985173
Test accuracy after pruning: 0.923995657218
Test accuracy after pruning: 0.905537459025
Test accuracy after pruning: 0.900108577374
Test accuracy after pruning: 0.892508143064
Test accuracy after pruning: 0.890336590403
Test accuracy after pruning: 0.888165038779
Test accuracy after pruning: 0.889250815109
Test accuracy after pruning: 0.895765473089
Test accuracy after pruning: 0.889250815109
Test accuracy after pruning: 0.872964169122
Test accuracy after pruning: 0.761129207577
Test accuracy after pruning: 0.763300760238
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```

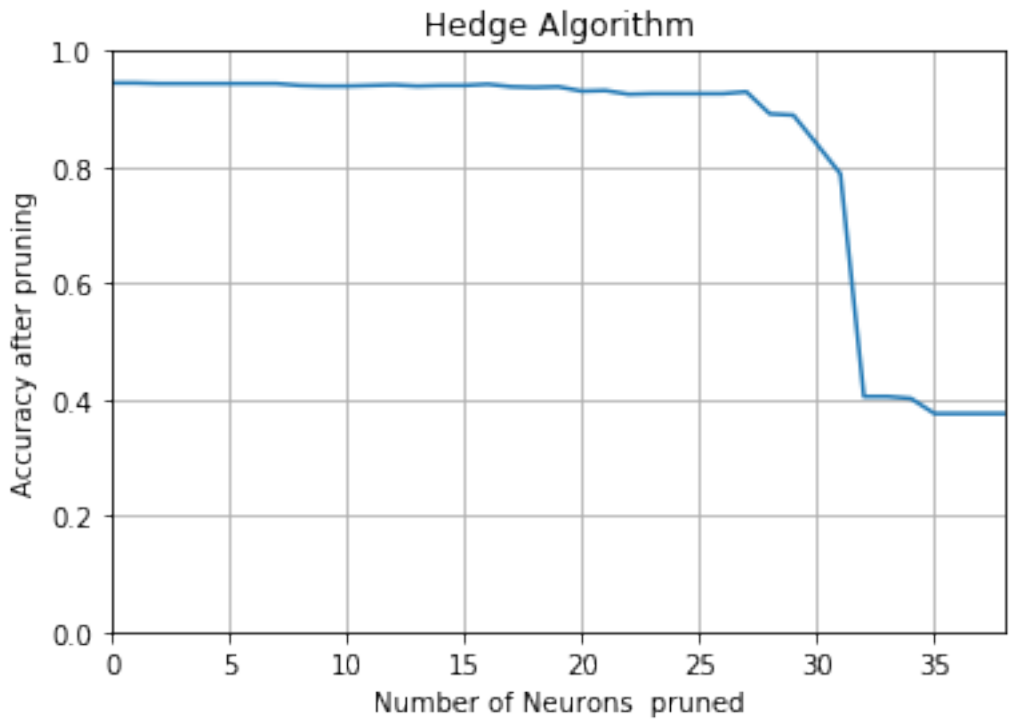


7 Run Hedge pruning Algorithm

```
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())
```

```
736 test samples
Test score: 0.338247084349
Test accuracy: 0.944625407166
The time for running this method is 1.6032681465148926 seconds
Finsh playing start pruining:
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.944625407166
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.943539630836
Test accuracy after pruning: 0.940282301846
```

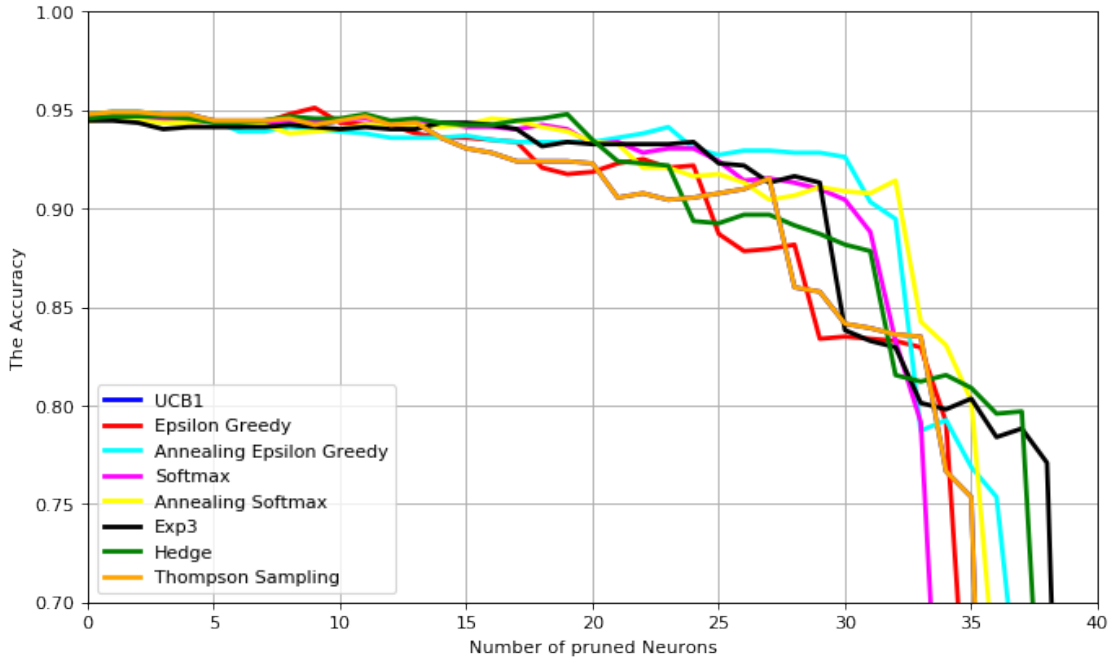
```
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.939196525516
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.941368078176
Test accuracy after pruning: 0.939196524933
Test accuracy after pruning: 0.940282301263
Test accuracy after pruning: 0.940282301846
Test accuracy after pruning: 0.942453854506
Test accuracy after pruning: 0.938110749186
Test accuracy after pruning: 0.937024972856
Test accuracy after pruning: 0.938110748603
Test accuracy after pruning: 0.930510315328
Test accuracy after pruning: 0.931596091658
Test accuracy after pruning: 0.925081433095
Test accuracy after pruning: 0.926167209425
Test accuracy after pruning: 0.926167209425
Test accuracy after pruning: 0.926167208843
Test accuracy after pruning: 0.926167208843
Test accuracy after pruning: 0.929424537833
Test accuracy after pruning: 0.891422366734
Test accuracy after pruning: 0.889250814073
Test accuracy after pruning: 0.83930510289
Test accuracy after pruning: 0.788273616412
Test accuracy after pruning: 0.404994571151
Test accuracy after pruning: 0.404994571151
Test accuracy after pruning: 0.401737241983
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
Test accuracy after pruning: 0.375678610061
```



8 Compare the accuracy of the models

```
In [13]: ucb1 = np.load('/Users/salemameen/Desktop/banditsbook/python/UCB1/AccuracyAftrerPrune.n
         EpsilonGreedy = np.load('/Users/salemameen/Desktop/banditsbook/python/EpsilonGreedy/Acc
         AnnealingEpsilonGreedy = np.load('/Users/salemameen/Desktop/banditsbook/python/Annealin
         Softmax = np.load('/Users/salemameen/Desktop/banditsbook/python/Softmax/AccuracyAftrerP
         AnnealingSoftmax = np.load('/Users/salemameen/Desktop/banditsbook/python/AnnealingSoftm
         Exp3 = np.load('/Users/salemameen/Desktop/banditsbook/python/Exp3/AccuracyAftrerPrune.n
         Hedge = np.load('/Users/salemameen/Desktop/banditsbook/python/Hedge/AccuracyAftrerPrune
         ThompsonSampling = np.load('/Users/salemameen/Desktop/banditsbook/python/Thompson_Sampl
         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="-", 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, 40, 0.7, 1])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

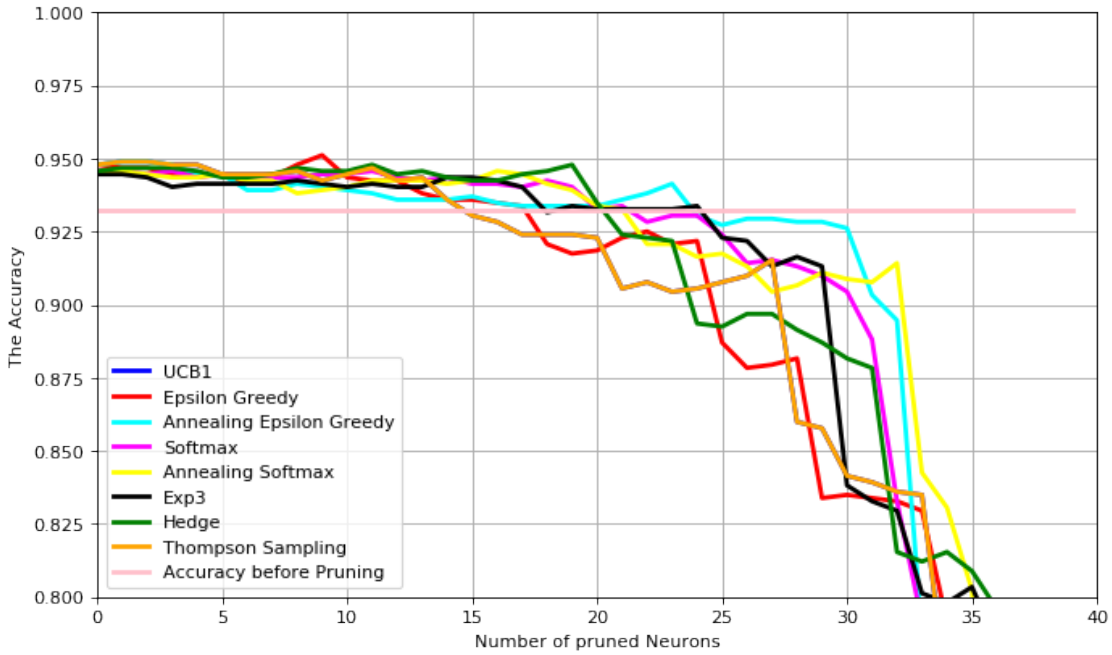


In [15]:
```
p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


#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="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 [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="-", 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, 40, 0.8, 1])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

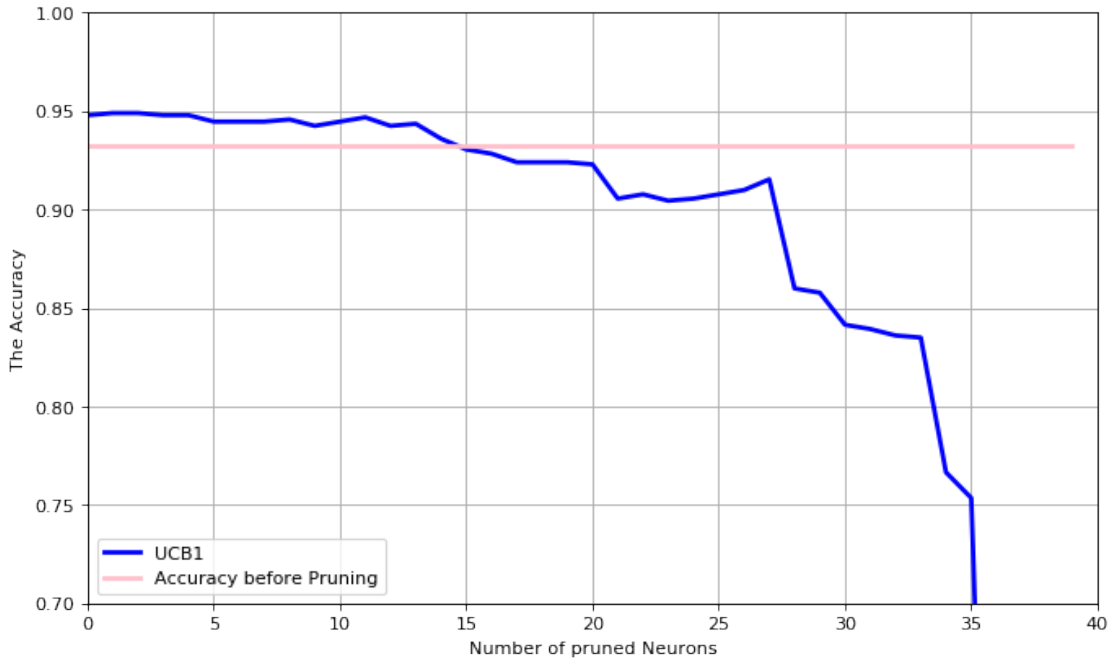


In [17]:
```
p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)


    #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="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

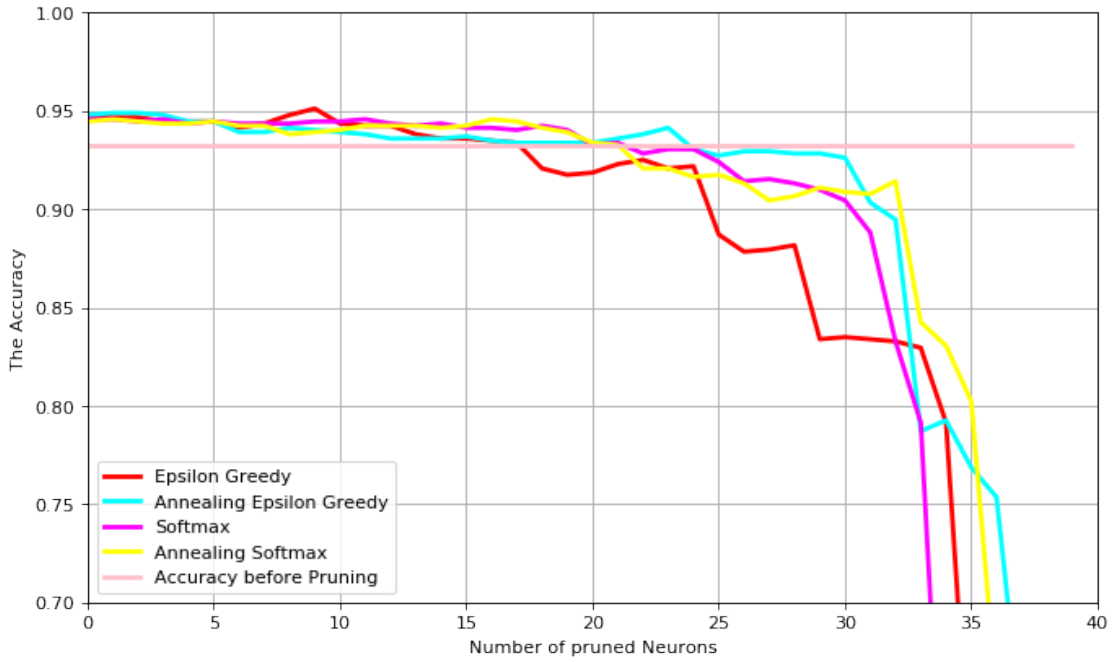
In [18]:
```
fig = plt.figure(figsize=(10, 6), dpi=80)
ax = fig.add_subplot(111)
N = len(ucb1)
Acc = [Accuracy for col in range(N)]
## necessary variables
ind = np.arange(N)                # the x locations for the groups
plt.plot(ind , ucb1 , color="blue", linewidth=2.5, linestyle="-", label="UCB1")
plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
plt.legend(loc = 3)
plt.axis([0, 40, 0.7, 1])
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=
         #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

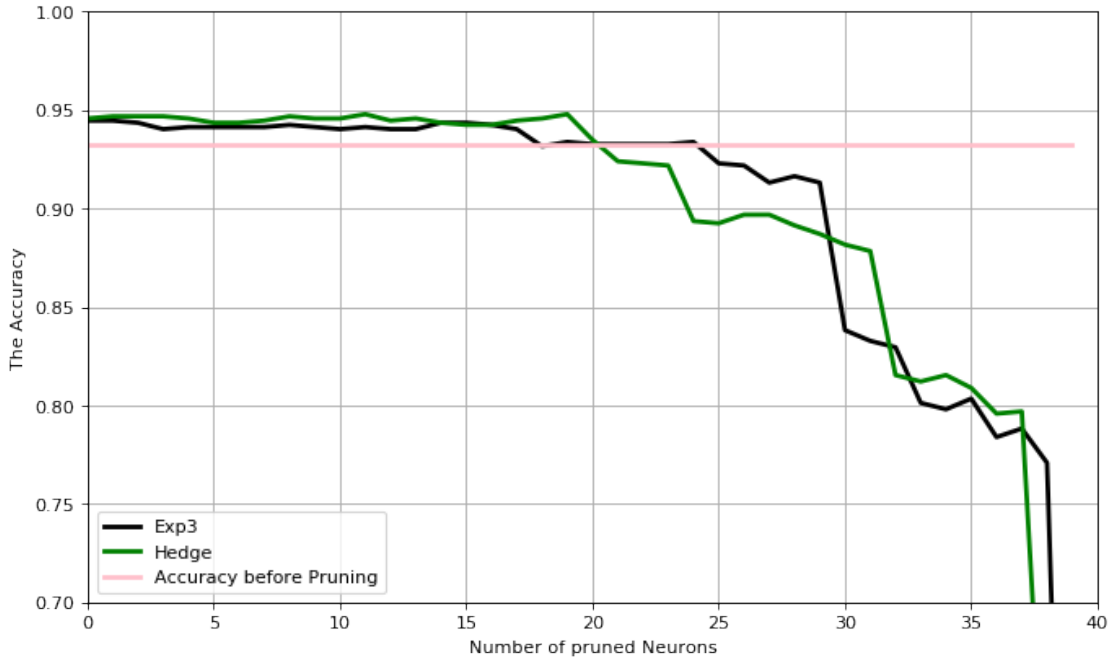
```
In [20]: fig = plt.figure(figsize=(10, 6), dpi=80)
         ax = fig.add_subplot(111)
         N = len(EpsilonGreedy)
         Acc = [Accuracy for col in range(N)]
         ## necessary variables
         ind = np.arange(N)                # the x locations for the groups
         plt.plot(ind , EpsilonGreedy, color="red", linewidth=2.5, linestyle="-", label="Epsilon
         plt.plot(ind , AnnealingEpsilonGreedy, color="cyan", linewidth=2.5, linestyle="-", 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, 40, 0.7, 1])
         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="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 [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, 40, 0.7, 1])
plt.xlabel('Number of pruned Neurons')
plt.ylabel('The Accuracy')
plt.grid(True)
plt.show()
```

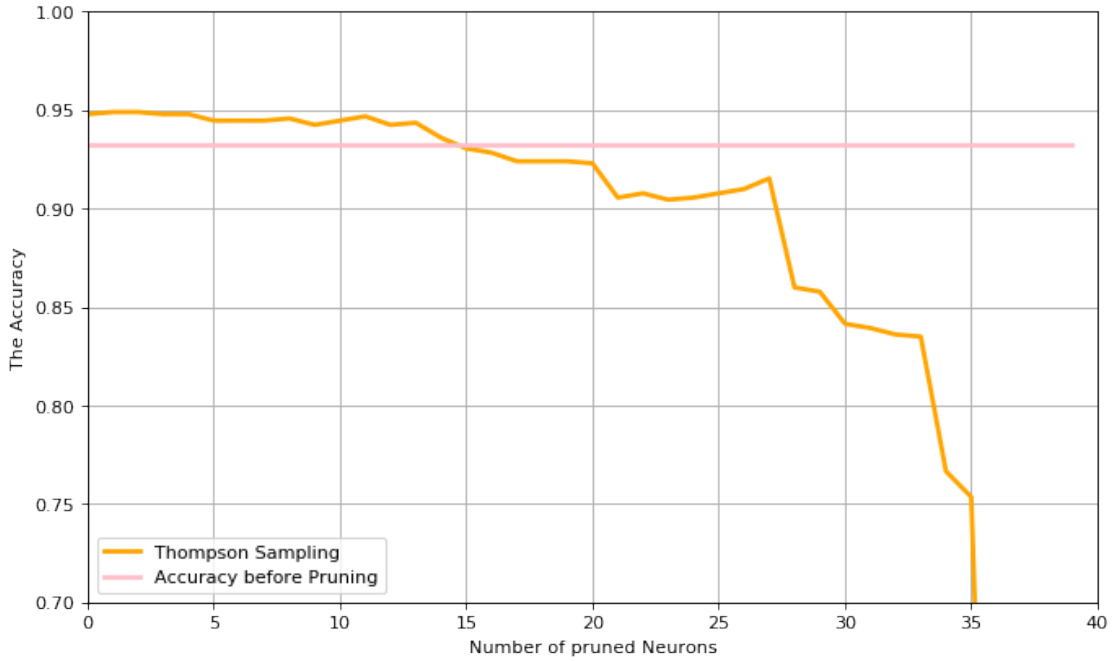


In [23]:
```
p1 = figure(title="The Performance over the number of neurons' pruned", tools=TOOLS)

#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=
#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="T
plt.plot(ind , Acc, color="pink", linewidth=2.5, linestyle="-", label="Accuracy before
plt.legend(loc = 3)
plt.axis([0, 40, 0.7, 1])
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_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
```