Shlukování pomocí ANN

```
In [1]: import numpy as np
        from neural_net.neuralnet import CompetitiveNeuralNet, NeuralNet
        import pandas as pd
        from matplotlib import pyplot as plt
        import itertools
        import json
        import csv
        from pylab import rcParams
        import random
        rcParams['figure.figsize'] = 16, 6 # resize figures
        plt.style.use('bmh')
```

Úvod

Tento projekt aplikuje shlukování pomocí neuronové sítě na následující proměnné časové řady:

- Precipitation
- Insolation
- Evaporation
- AvgHumidity
- Windspeed
- AvgTemp

Preprocessing dat

```
In [2]: with open('resources/cruzeirodosul2010daily.csv', 'r') as f:
            reader = csv.DictReader(f, delimiter=';')
            dates = np.array([l['Date'] for l in reader], dtype='int64') / 100_000 # squish the range of dates
        with open('resources/cruzeirodosul2010daily.csv', 'r') as f:
            reader = csv.DictReader(f, delimiter=';')
            inputs = np.array([[l['Precipitation'], l['Insolation'], l['Evaporation'], l['AvgHumidity'], l['WindSpee
            inputs = np.where(inputs == '', np.nan, inputs)
            inputs = np.where(inputs == '#N/A', np.nan, inputs)
            inputs = inputs.astype('float64')

        dates[0:5], inputs
```

```
Out[2]: (array([0.40179, 0.4018 , 0.40181, 0.40182, 0.40183]),
         array([[ 0.      ,  9.5     ,  1.7     , 79.25    ,  1.666667, 27.14    ],
                [ 0.      ,  8.7     ,  3.5     , 75.25    ,  1.      , 27.7     ],
                [ 0.      ,  8.3     ,  4.1     , 73.      ,  0.      , 28.68    ],
                ...,
                [ 5.8     ,  8.8     ,       nan, 92.      ,  1.02888 , 27.84    ],
                [ 0.      ,  0.2     ,       nan, 93.      ,  0.      , 26.6     ],
                [ 0.      ,  7.1     ,       nan, 97.      ,  0.      , 27.78    ]]))
```

```
In [11]: inputs_interp = NeuralNet.linear_interpolation(inputs)
         norm_inputs = NeuralNet.normalize(inputs_interp)

         vec_len = len(norm_inputs[0])
         norm_inputs[0:2], vec_len
```

```
Out[11]: (array([[0.        , 0.087236  , 0.01561065, 0.72773186, 0.01530456,
                  0.24921947],
                 [0.        , 0.07988981, 0.03213958, 0.69100092, 0.00918274,
                  0.2543618 ]]),
          6)
```

Vypracování

```
In [22]: neurons = [2, 3, 4, 5]
         learning_rates = [0.5, 0.1, 0.05, 0.005]
         params = itertools.product(neurons, learning_rates)
         no_epochs = 10
         stats = []
         for p in params:
             print(p)
             indices = []
             cnn = CompetitiveNeuralNet(vec_len, p[0])
             for e in cnn.cluster(input_dataset=diff_norm_inputs, no_epochs=no_epochs, learning_rate=p[1]):
                 print(e['dunn index'])
                 indices.append(e['dunn index'])
             stats.append({
                 'indices': indices,
                 'neurons': p[0],
                 'learning rate': p[1],
                 'epochs': no_epochs,
                 'labeled data': {k: [val.tolist() for val in vals] for k, vals in e['labeled dataset'].items()}
             })
         stats_json = json.dumps(stats)
         with open('resources/stats_clustering.json', 'w') as f:
             f.write(stats_json)
```

```
(2, 0.5)
0.049037973612090224
0.00683713252664844
0.016542382488818383
0.003419546911876512
0.009949081907811795
0.002279915677438479
0.007113733645561438
0.0017100183828272612
0.00553603639082704
0.0013680538793868597
(2, 0.1)
0.0788477779890632
0.0007571686605265163
0.026667684448244956
0.0003786031347269763
0.01604724441369293
0.00025240628184885306
0.011476615485010971
0.0001893062843997218
0.008932438689943081
0.0001514457828224585
(2, 0.05)
0.22418239352815222
0.22444789281461525
0.22453762323891313
0.22458271920549086
0.2246098506272087
0.22462796900914292
0.22464092578031022
0.22465065159997635
0.22465822101011163
0.22466427961496915
(2, 0.005)
0.3999474125467287
0.001079680792799447
0.1353377713686805
0.0005400036611138325
0.081450089262973
0.0003600387520442777
0.058251472813716516
0.00027004268502646363
0.04533741400603778
0.00021604068718730432
(3, 0.5)
0.028775732627319618
0.029191014886148392
0.029420133560699693
0.02955123582893162
0.029635116561530395
0.029693193274979636
0.029735726703800897
0.029768198660013304
0.029793792465026156
0.02981447982896077
(3, 0.1)
0.06629099744686096
0.0018161982554336202
0.03474990255457579
0.0009091320400222315
0.02093435538243271
0.0006063178283684204
0.014978776084390726
0.00045482458425794096
0.011661212757480166
0.00036390105919850127
(3, 0.05)
0.14669259319933803
0.06838646577306354
0.0005514147872209469
0.03629385540867214
0.0275668806677963
0.0002758920423606483
0.02072250151340555
0.017266376986381893
0.0001839690839357199
0.014501625341222272
(3, 0.005)
0.026874255860854457
0.013678866110791598
0.0003533175569298899
0.006774931891280862
0.005522201530108729
0.0001767367488230483
0.003879452611886896
0.0034603924293445994
0.00011784183172045299
0.0027180443244014404
(4, 0.5)
0.058116393163204945
0.03417270033530892
0.003022496821070615
0.01809727702218827
0.013581460688969944
0.001511769004503502
0.010415312702523329
0.008476218129714708
0.0010079614043483325
0.007311630514788401
(4, 0.1)
0.09496527148013886
0.058869820399568513
0.03328308824336406
0.000886130482112097
0.01992433960322585
0.020140647323939623
0.014395532605714509
0.000442700966573585
0.011137895155711562
0.012147499949331135
(4, 0.05)
0.17007325580741992
0.003452390284137042
0.03584780223299912
0.001729663633117882
0.040510607031016284
0.0011538819176475817
0.035565888956402406
0.0008657015380613089
0.02769720412415826
0.0006927005523739934
(4, 0.005)
0.08434825813134868
0.06538818445571984
0.028091632331568993
0.0007474134974948586
0.017492450104168285
0.022324564884407434
0.01201701868604568
0.00037416446944737135
0.009764928333875395
0.013461670385662837
(5, 0.5)
0.06446169014637727
0.02542659169156412
0.010735152480131785
0.012751294981936313
0.017299899753349248
0.0053734381802011105
0.00997083880788186
0.006407889505245475
0.003583601651686893
0.005122675768299839
(5, 0.1)
0.046738066370258416
0.021888996265458406
0.013182758674664933
0.00998038030005186
0.0006330814586265304
0.007800048853978501
0.0064869408443017145
0.004926280530089793
0.004461566938715828
0.00031710812445461293
(5, 0.05)
0.15662009453806428
0.091478939345583
0.04957656771945778
0.0012822637950920899
0.03584289837005591
0.031207712195517538
0.02137223802704113
0.0006420580540381952
0.020039257593648125
0.018812158452317148
(5, 0.005)
0.06533004578354012
0.028448448190142504
0.023026895351946474
0.016677702567563207
0.000813191179574978
0.011231758572053606
0.009314555995835168
0.008526150915724345
0.007472613105873297
0.0004068481523023849
```

Citlivostní analýza

```
In [23]: df = pd.read_json('resources/stats_clustering.json')
         df[['neurons', 'epochs', 'learning rate']]
```

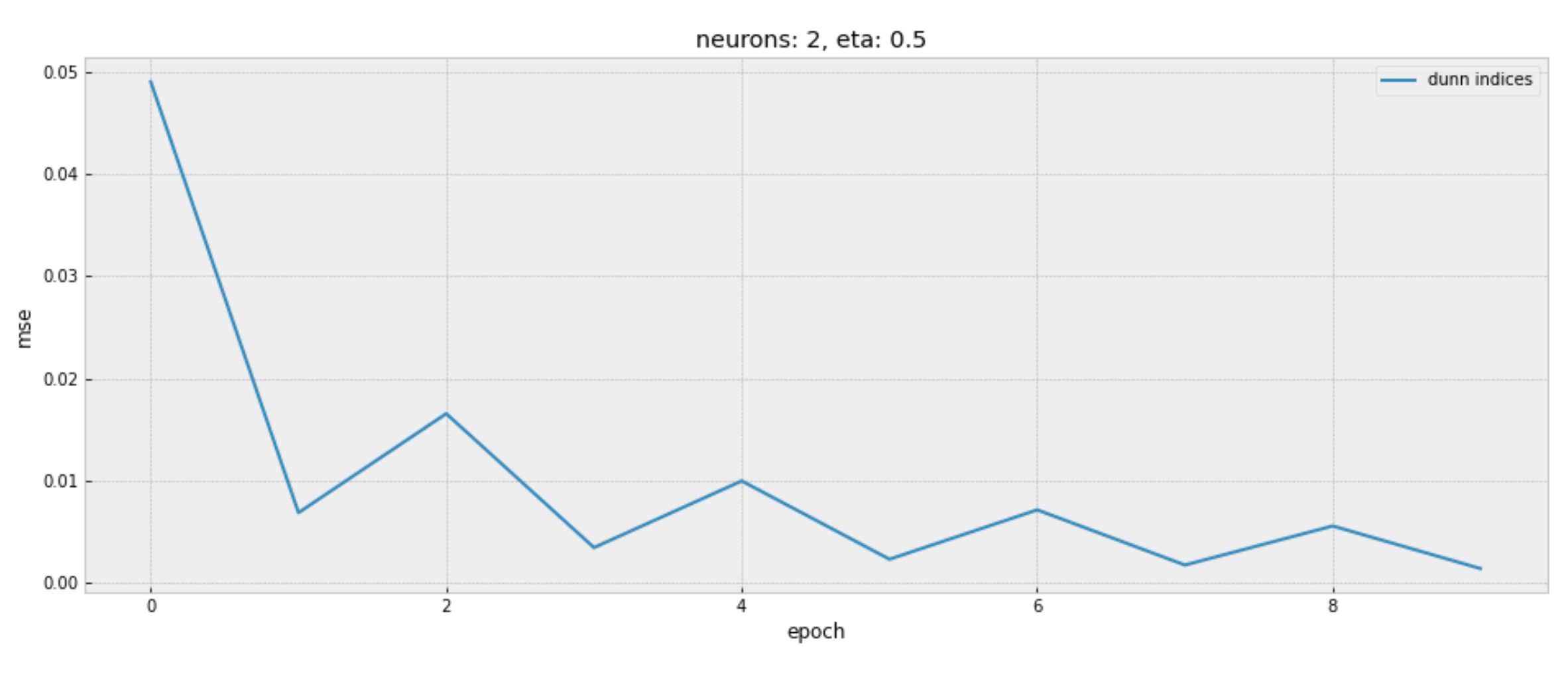
Out[23]:

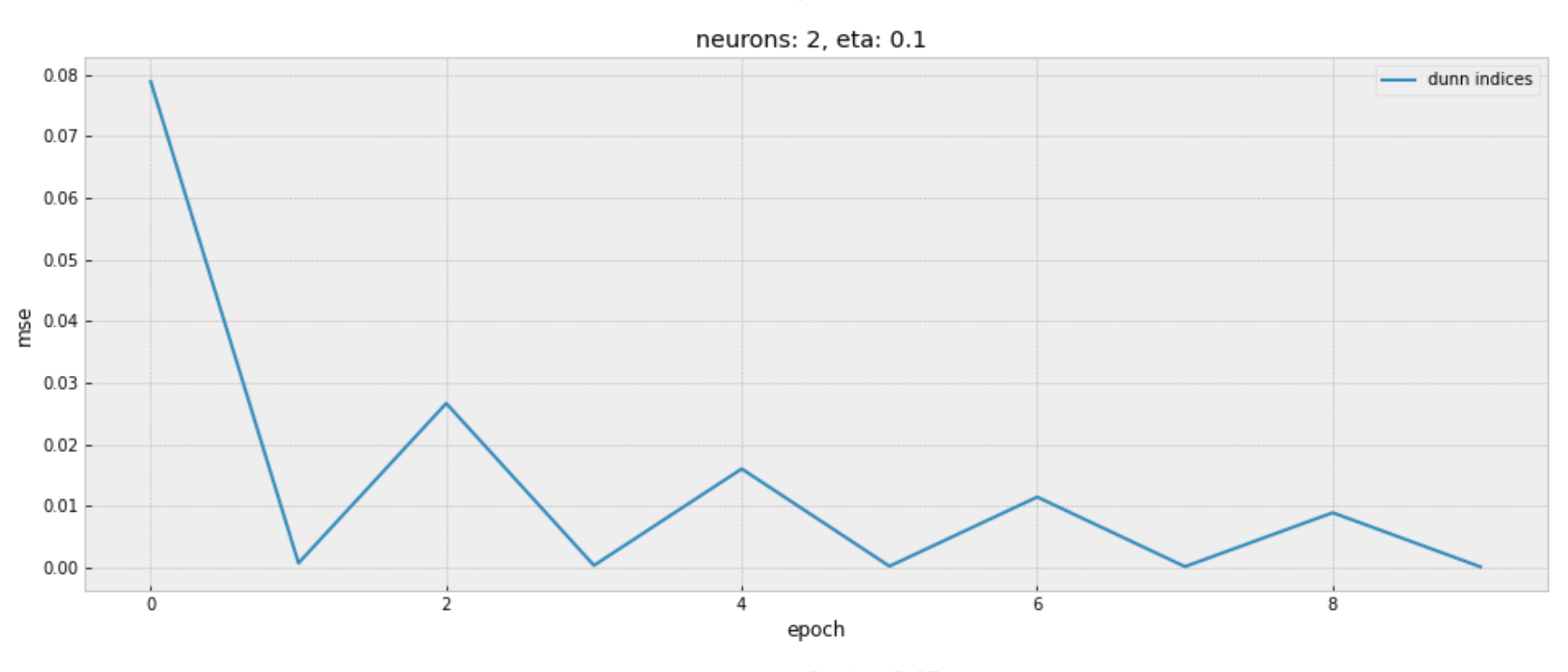
	neurons	epochs	learning rate
0	2	10	0.500
1	2	10	0.100
2	2	10	0.050
3	2	10	0.005
4	3	10	0.500
5	3	10	0.100
6	3	10	0.050
7	3	10	0.005
8	4	10	0.500
9	4	10	0.100
10	4	10	0.050
11	4	10	0.005
12	5	10	0.500
13	5	10	0.100
14	5	10	0.050
15	5	10	0.005

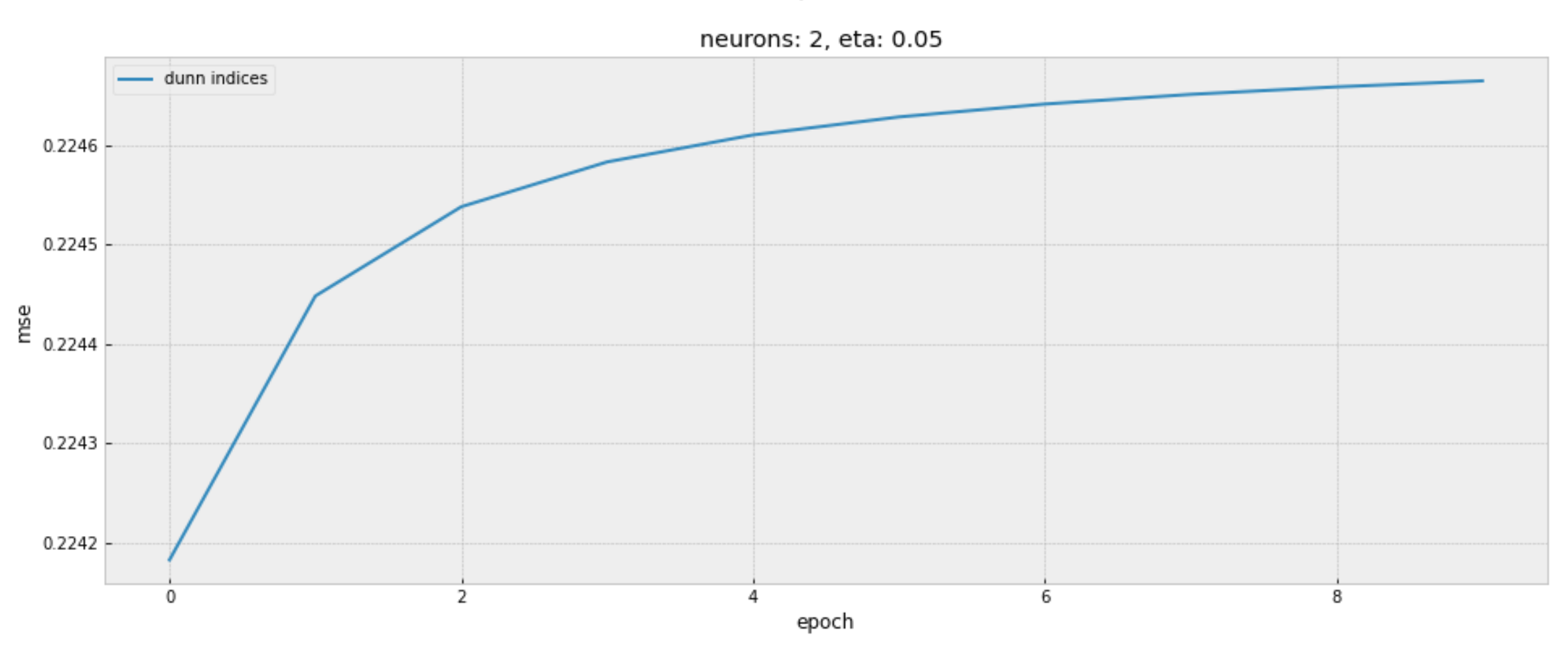
Grafické znázornění vývoje Dunnových indexů pro jednotlivé kombinace hyperparametrů

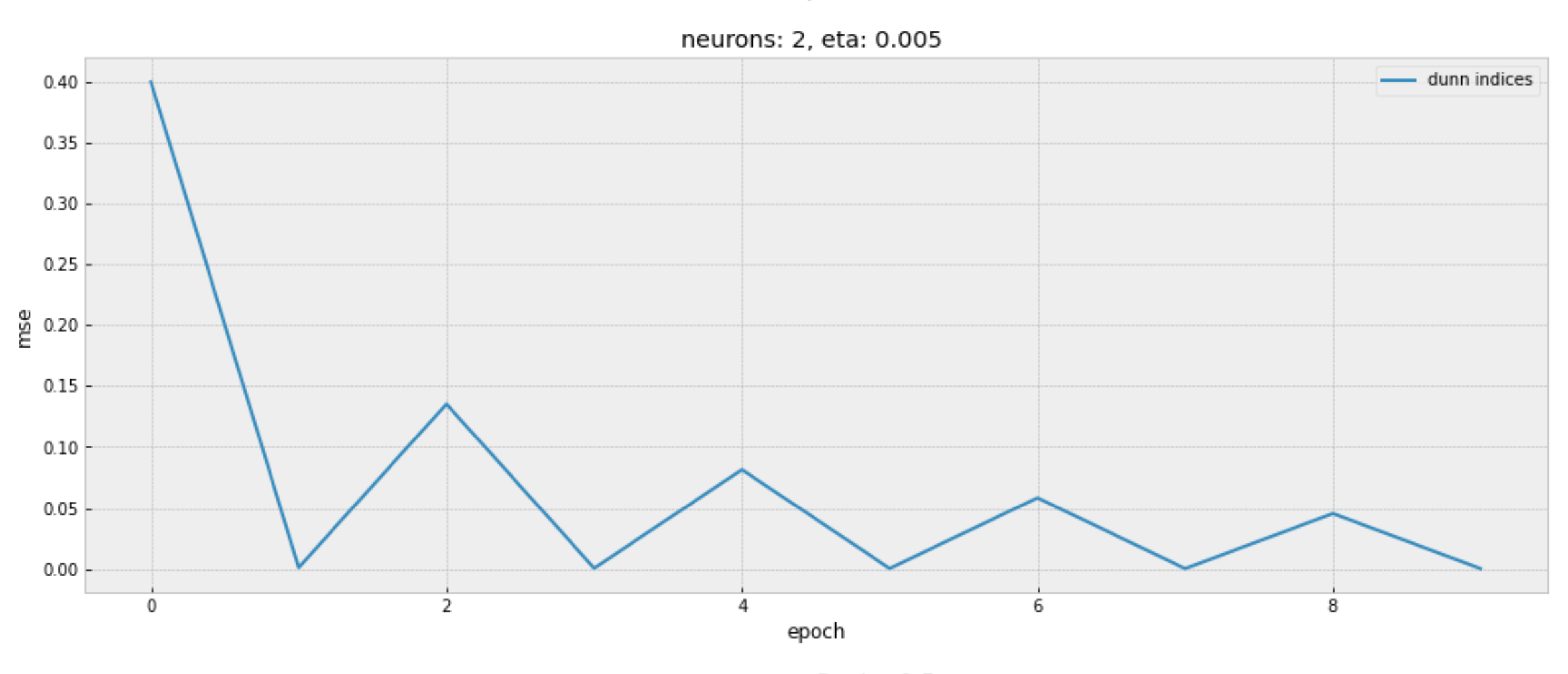
```
In [24]: # plt.plot(df['mse list'], label='mse')
         for i in range(15):
             plt.plot(df['indices'][i], label='dunn indices')
             plt.title('neurons: {}, eta: {} '.format(df['neurons'][i], df['learning rate'][i]))
             plt.xlabel('epoch')
             plt.ylabel('mse')
             plt.legend()
             ax, fig = plt.subplots()
         plt.plot(df['indices'][15], label='dunn indices')
         plt.title('neurons: {}, eta: {} '.format(df['neurons'][15], df['learning rate'][15]))
         plt.xlabel('epoch')
         plt.ylabel('mse')
         plt.legend()
```

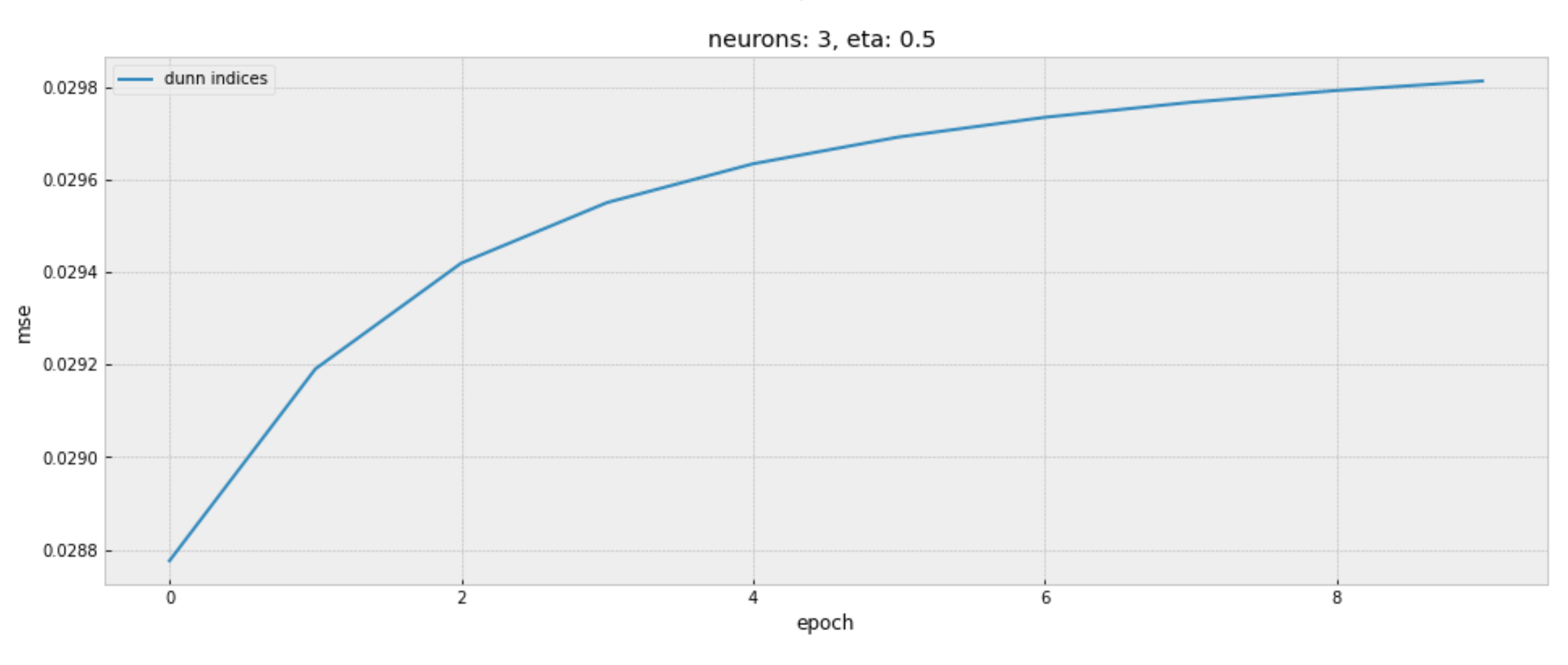
```
Out[24]: <matplotlib.legend.Legend at 0x7f877e1a34c0>
```

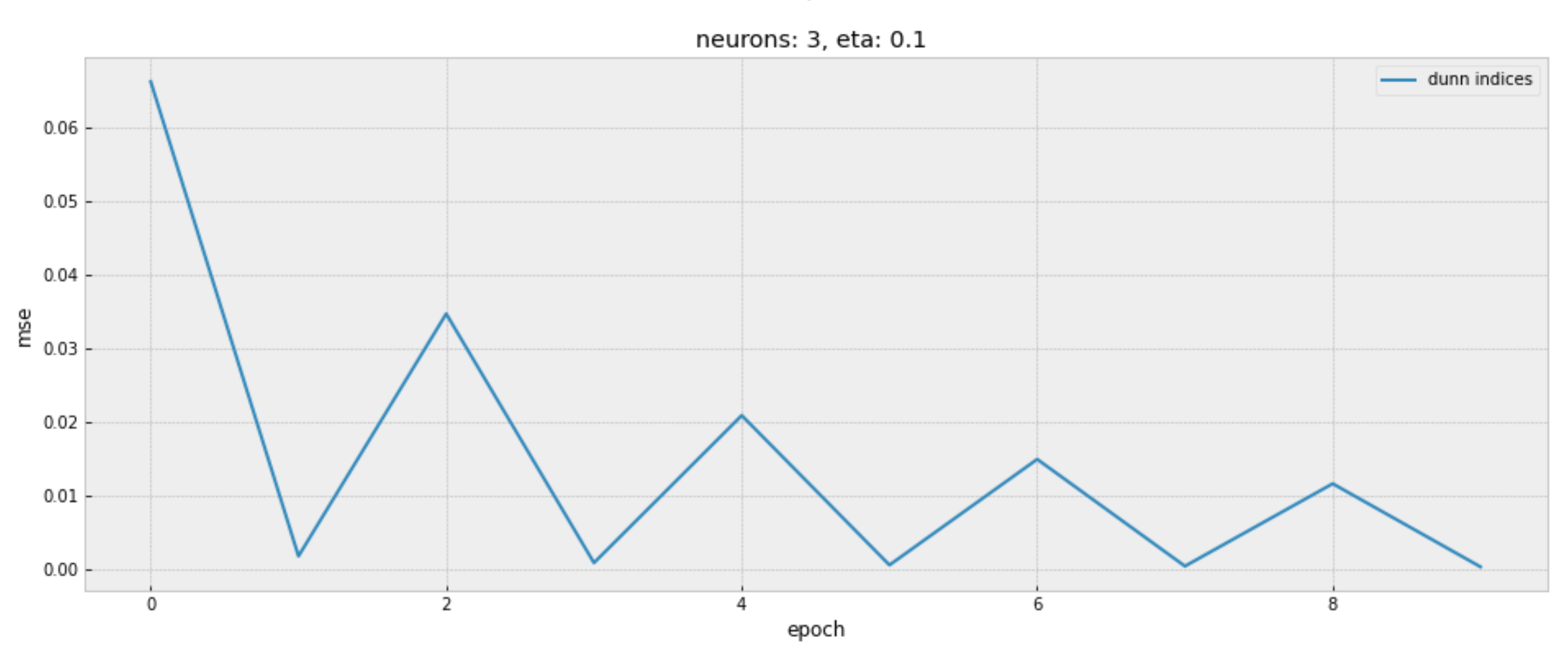


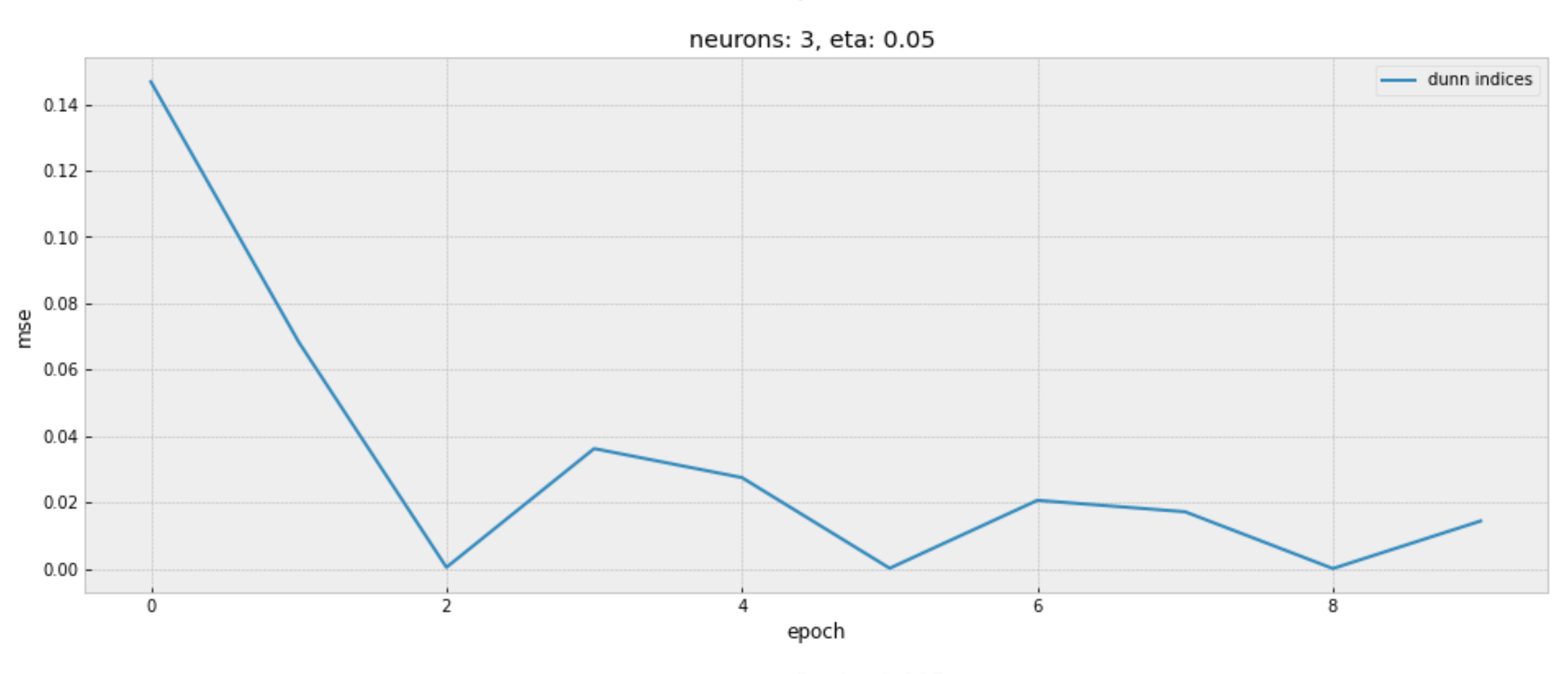


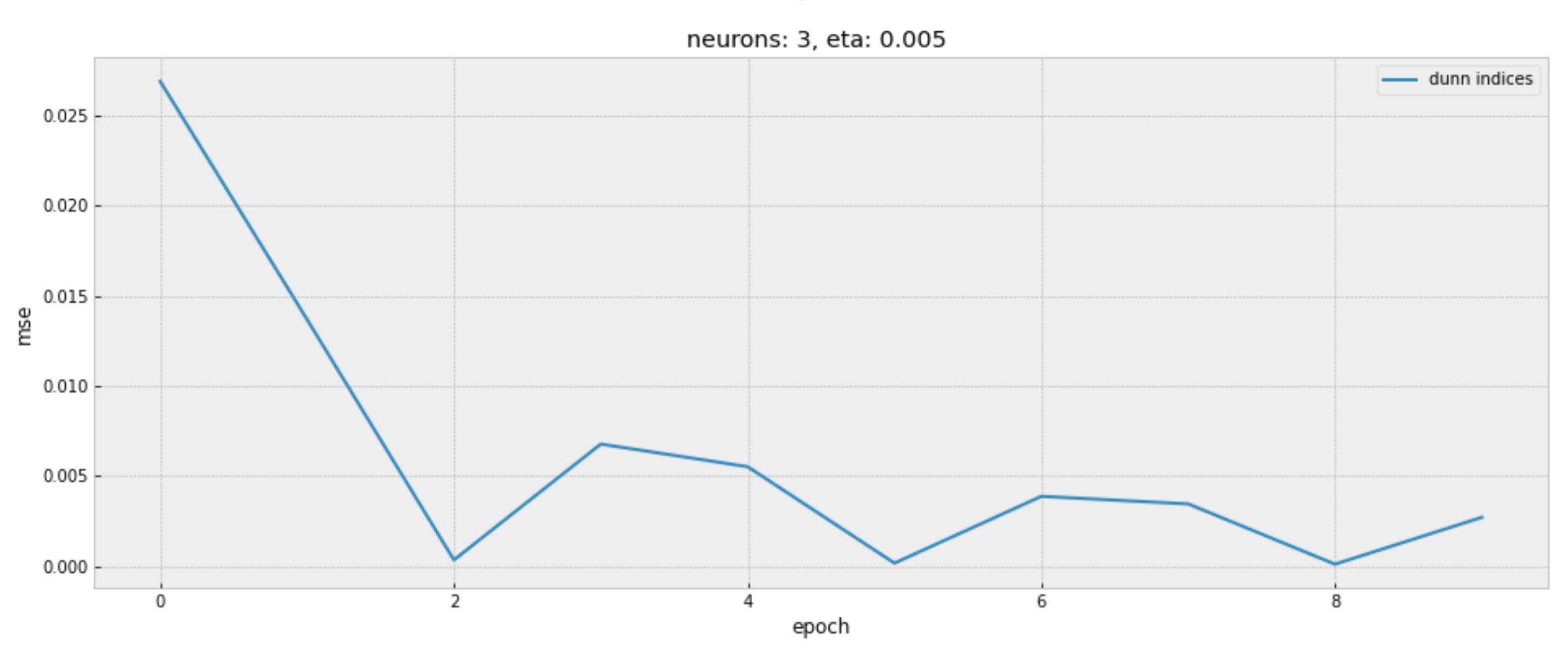


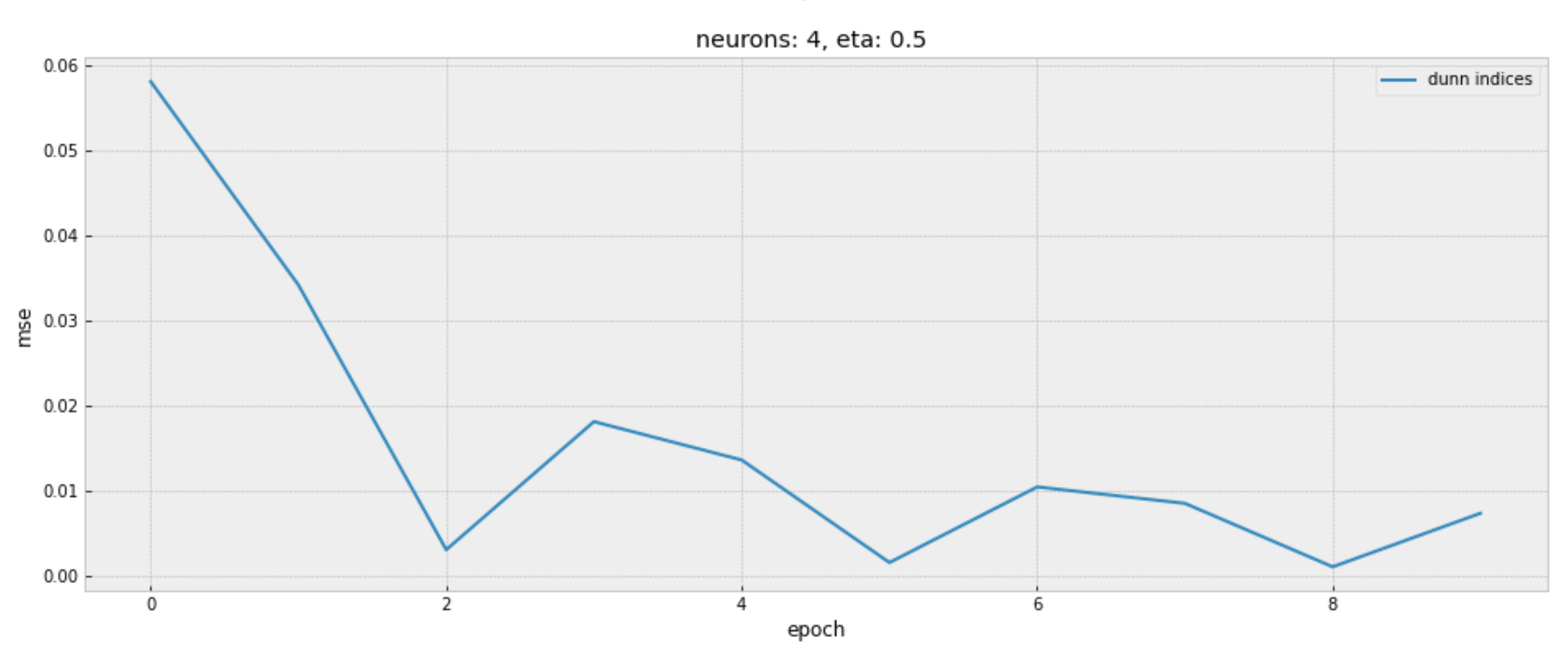


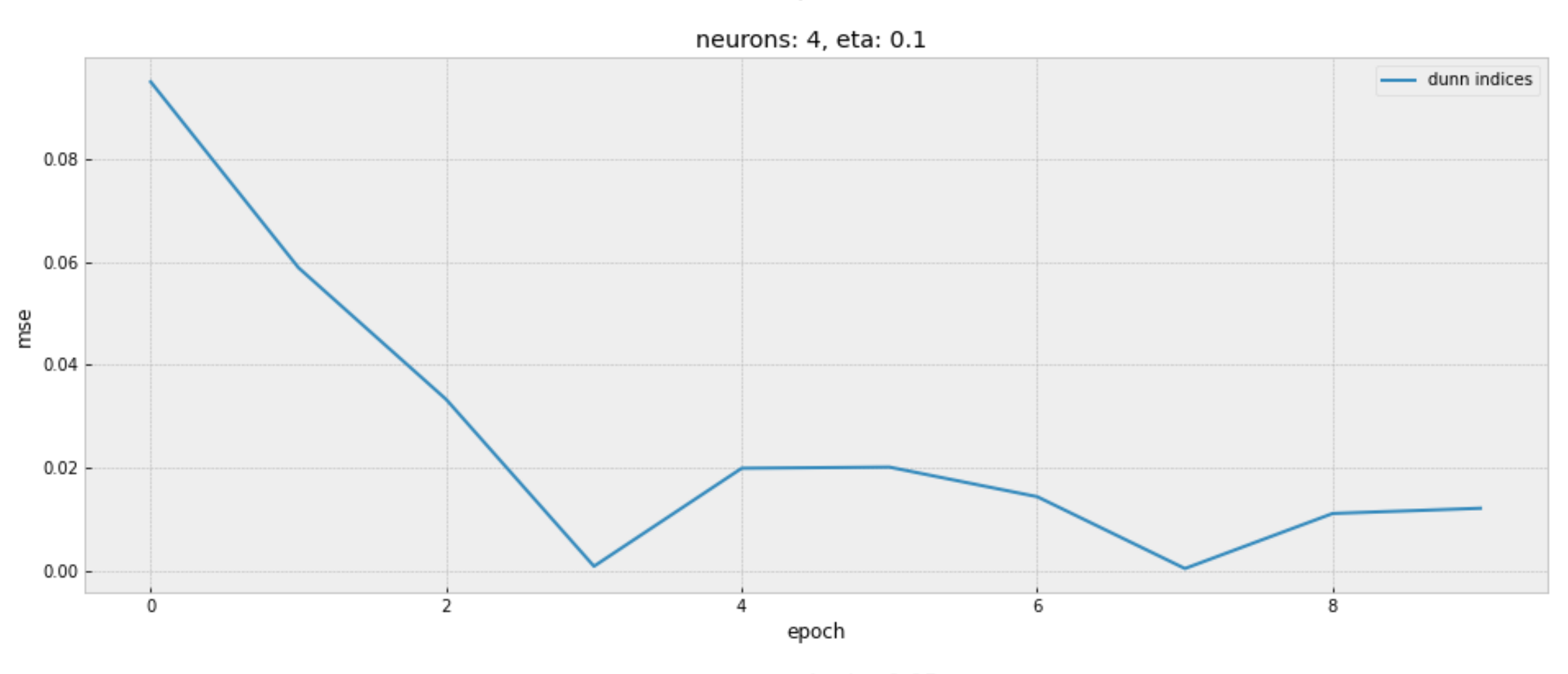


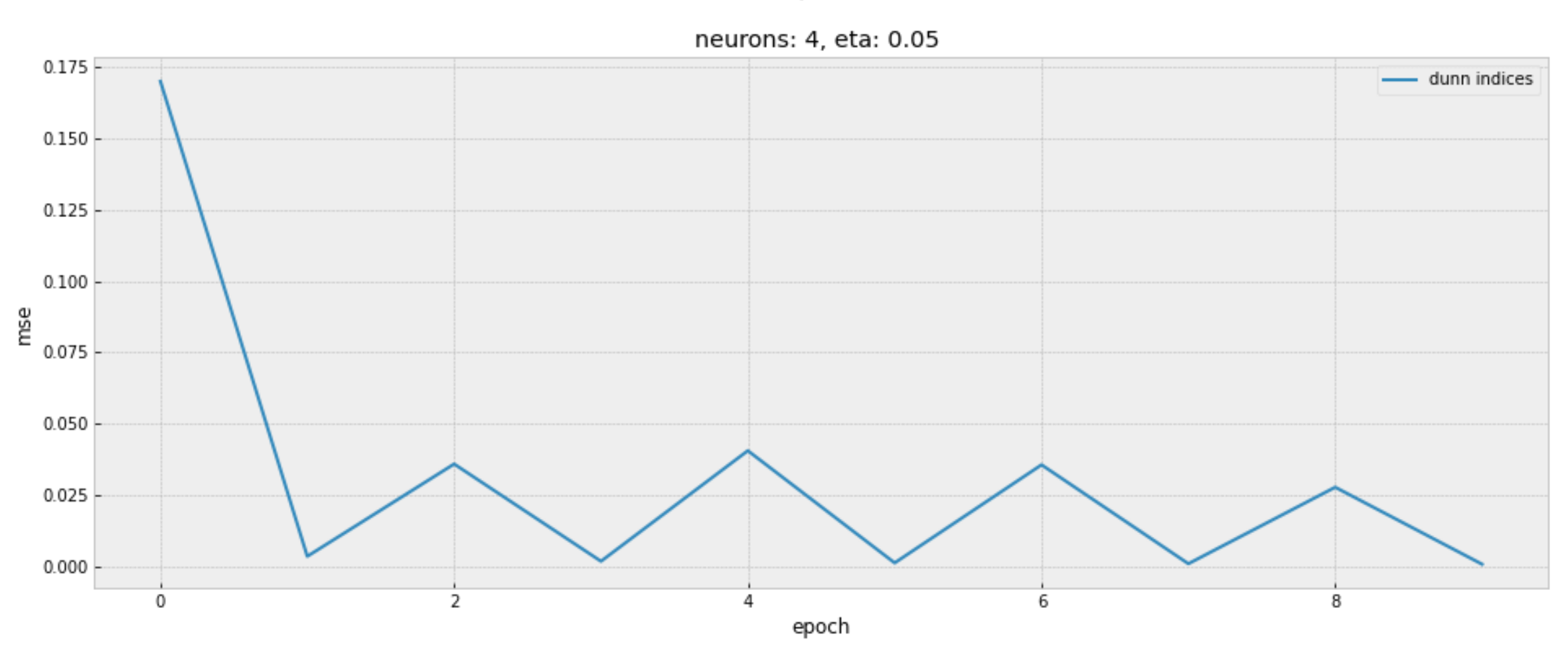


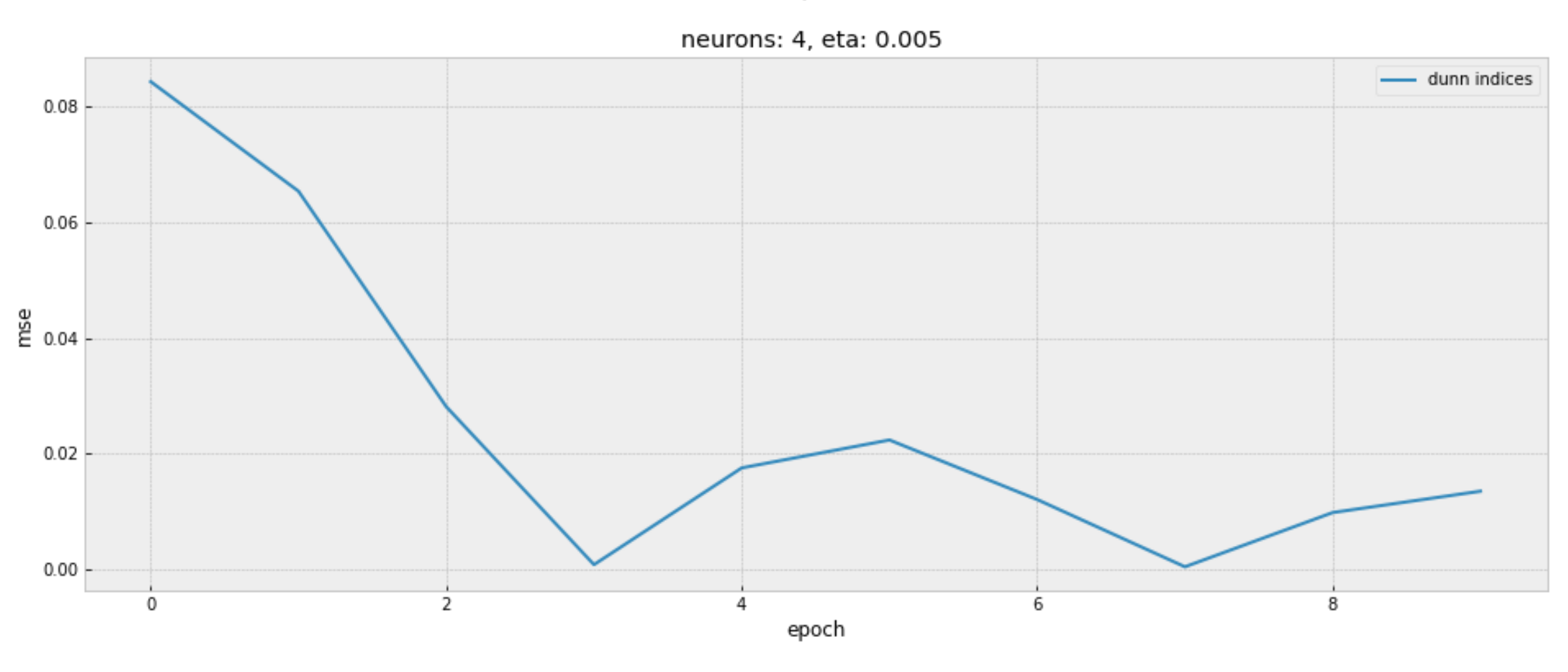


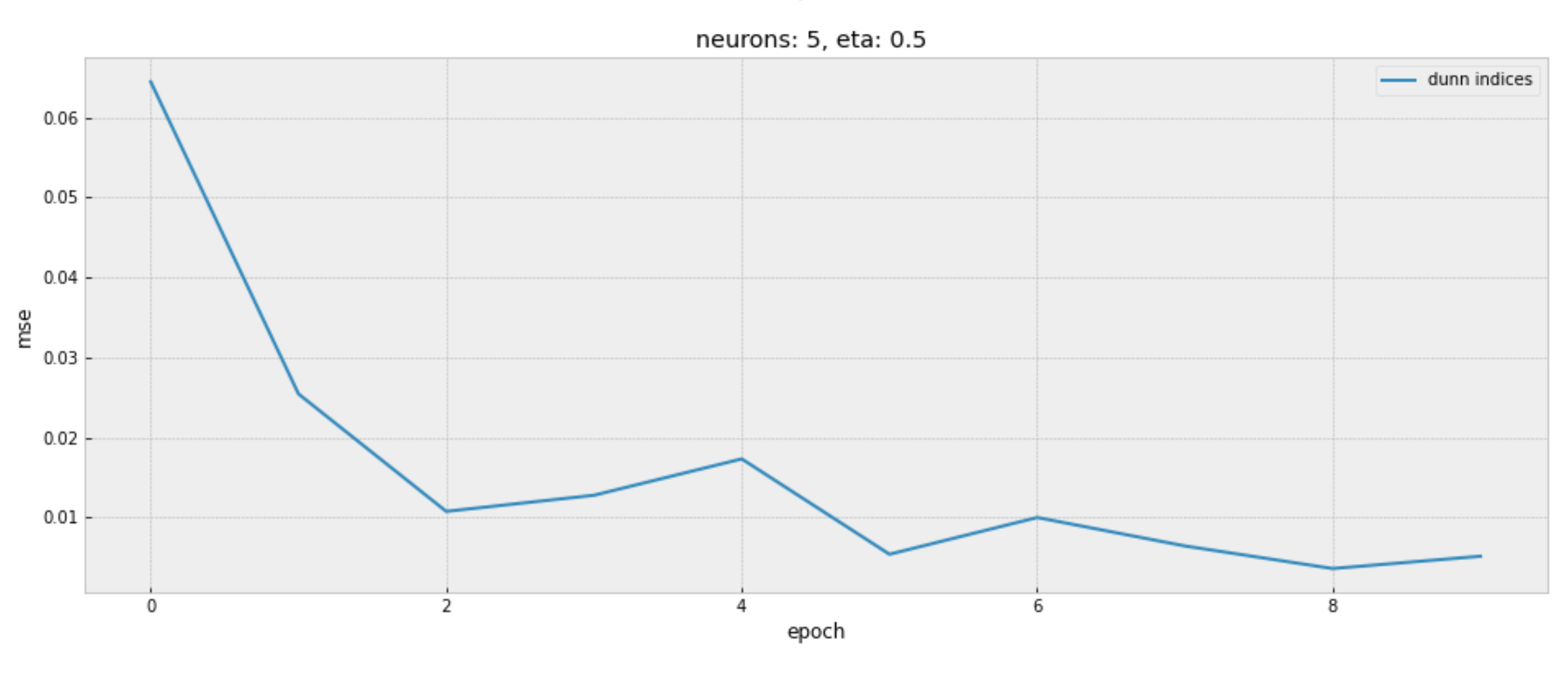


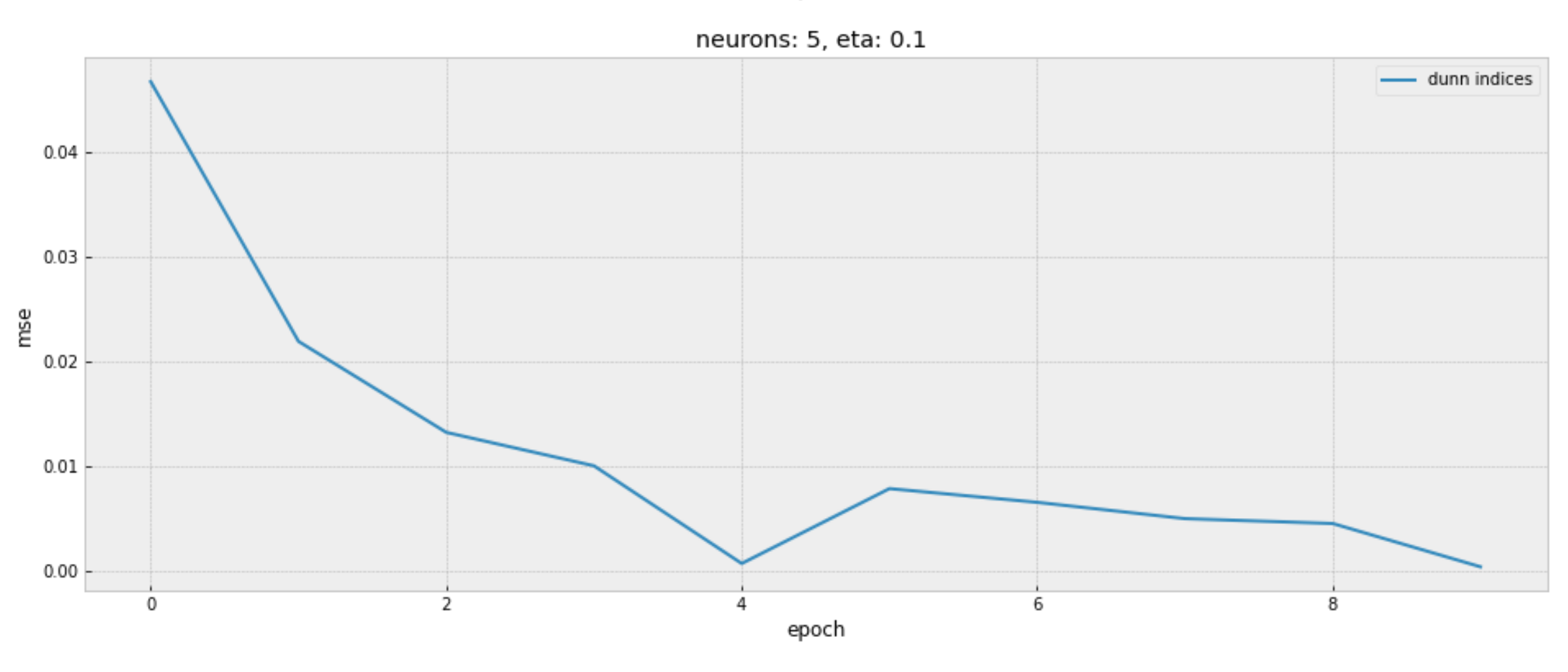


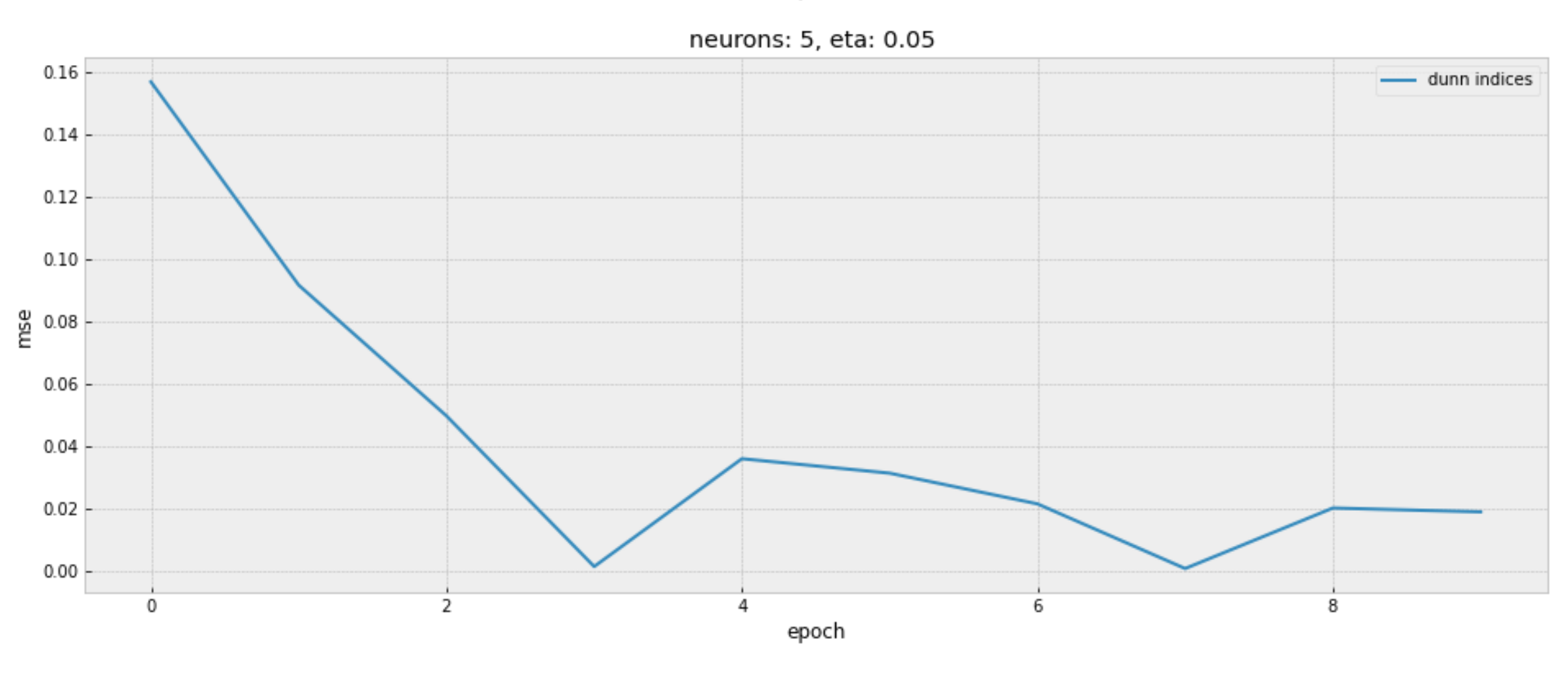


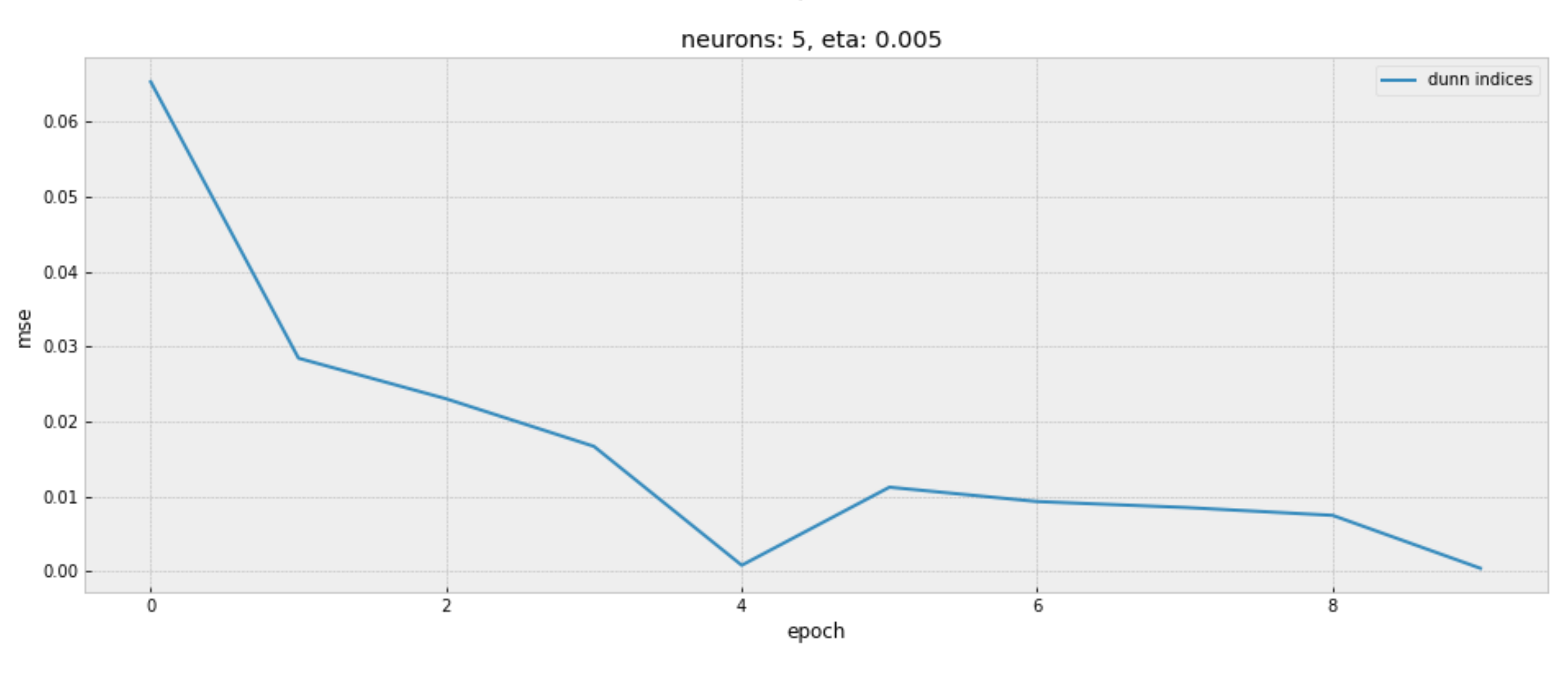












Závěr

Dunnův index vykázal rostoucí tendenci jen ve dvou případech. Možné vysvětlení může být algoritmus učení, který má rostoucí sklon k jedinému vítěznému neuronu.