Model_I_3parameters

March 17, 2021

[5]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import os
from functions.poll_data import party_in_region, region_in_party
import pickle
import matplotlib.pyplot as plt
#https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

from tqdm.auto import tqdm, trange
```

0.1 Percent voting people

[6]:
```
voter_w = pd.read_csv('dane_years/voters/percent_voters.csv',header=None)
```

0.2 Stat data

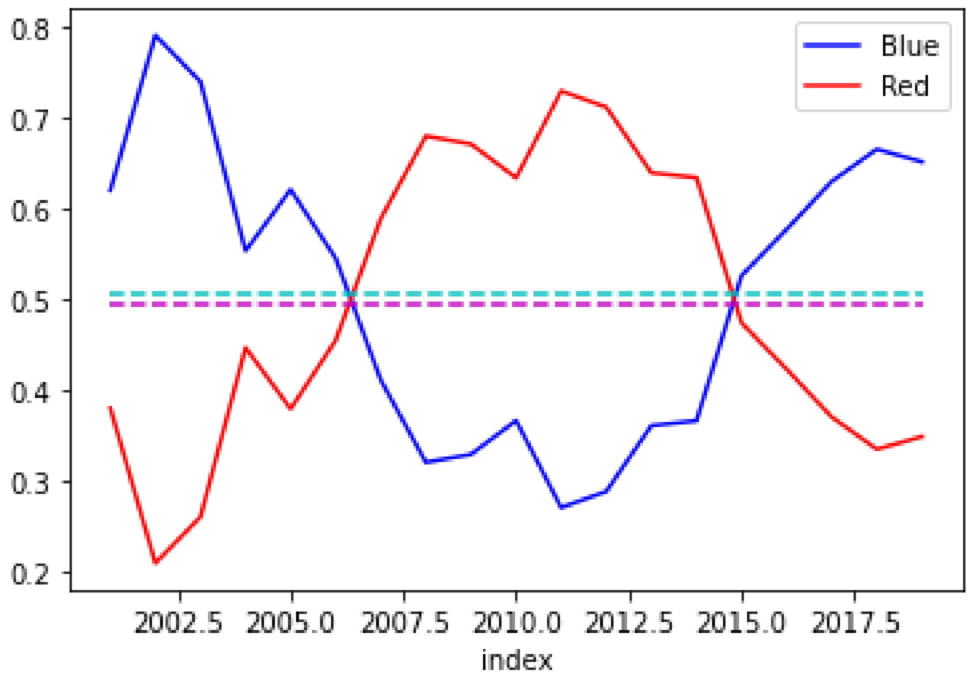
[7]:
```
path = 'dane_years/'
files = list(filter(lambda x: os.path.isfile(path+x), os.listdir(path)))
files.sort()
```

[8]:
```
stat_list = [(lambda x: pd.read_csv(path+x,index_col=0, header=0))(f) for f in␣
↪files[:-1]]
```

[9]:
```
for yi in range(len(stat_list)):
    y = files[yi].split('.')[0]
    c = stat_list[yi].columns
    c = [y+'-'+ci for ci in c]
    # c = [y[2:]+'-'+str(ci) for ci in range(len(c))]
    stat_list[yi].columns = c
```

0.3 Poll data

[10]:
```
#pool_data = pd.read_csv('dane_years/pools_edited.csv', index_col=0)
pool_data_middle = pd.read_csv('dane_years/pools_data/percent_votes.csv',␣
↪index_col=0).iloc[:,:-1]
pool_data_middle = pool_data_middle.divide(pool_data_middle.sum(1),0)
pool_data_middle.plot(color=['b','r'])
means = pool_data_middle.mean(0)
plt.plot([2001,2019],[means[0],means[0]],'c--')
plt.plot([2001,2019],[means[1],means[1]],'m--')
plt.show()
```



0.4 Voting data

[11]:
```
path = 'wyniki_wyborow/Simple/'
files = list(filter(lambda x: os.path.isfile(path+x), os.listdir(path)))
files.sort()
files
```

[11]:
```
['2001_WS.csv',
 '2005_WS.csv',
 '2007_WS.csv',
 '2011_WS.csv',
 '2015_WS.csv',
 '2019_WS.csv']
```

```
[12]: vote_list = [(lambda x: pd.read_csv(path+x,index_col=0, header=0))(f) for f in␣
      ↪files[:]]
      vote_list[0] = vote_list[0].iloc[1:,:]
```

```
[13]: vote_list[0]['jednostka'] = [j.upper() for j in vote_list[0]['jednostka']]
      vote_list[0] = vote_list[0].sort_values(['jednostka'])
      vote_list[0].columns = ['województwo'] + vote_list[0].columns.values.tolist()[1:
      ↪]
```

```
[14]: vote_list[0] = vote_list[0].set_index('województwo')
```

```
[15]: if(False):
          for yi in range(len(vote_list)):
              y = files[yi].split('_WS')[0]
              c = vote_list[yi].columns
              #c = [y+'-'+ci for ci in c]
              # c = [y[2:]+'-'+str(ci) for ci in range(len(c))]
              vote_list[yi].columns = c

              #vote_list[yi] = vote_list[yi].div(vote_list[yi].sum(axis=1), axis=0).
      ↪fillna(0)
```

0.5 Neighbours

```
[16]: with open('wojew_neighbours.pkl', 'rb') as f:
          neighbours = pickle.load(f)
```

0.6 Use 2 approaches to estimate date from years without elections

```
[17]: #party_in_region(df_vote, df_poll)
      par_in_reg_list = [vote_list[0].iloc[:,:-1]]
      #region_in_party(df_vote, df_poll)
      reg_in_par_list = [vote_list[0].iloc[:,:-1]]
      for pool in pool_data_middle[1:].iterrows():
          if int(pool[0]) < 2005: df_vote = vote_list[0]
          elif int(pool[0]) < 2007: df_vote = vote_list[1]
          elif int(pool[0]) < 2011: df_vote = vote_list[2]
          elif int(pool[0]) < 2015: df_vote = vote_list[3]
          elif int(pool[0]) < 2019: df_vote = vote_list[4]
          else: df_vote = vote_list[5]

          par_in_reg_list.append(party_in_region(df_vote.iloc[:,:-1], pool[1]))
          reg_in_par_list.append(region_in_party(df_vote.iloc[:,:-1], pool[1]))
```

```
[18]: for vl, i in zip(vote_list.copy(),[0,4,6,10,14,18]):
```

```
    par_in_reg_list[i] = vl.iloc[:,:-1].div(vl.iloc[:,:-1].sum(1),0) #vl.iloc[:
↪,:-1]
    reg_in_par_list[i] = vl.iloc[:,:-1].div(vl.iloc[:,:-1].sum(1),0) #vl.iloc[:
↪,:-1]
```

0.7 Prepare input (X) (parameters which we will multiply)

Input - wagi, na outpucie mnożenie wag razy wartości i na tej podstawie ocena. - poprzednie wybory (par in reg/ reg in par) - wpływ sąsiadów (avg over nighbours/ weighted avg) - pole zewnętrzne

[19]:
```
pool_d = par_in_reg_list if (False) else reg_in_par_list
```

[20]:
```
X = []
# iterate over years [from 2002 - 2019]
for y in range(pool_data_middle.shape[0]-1):
    # iterate over districts
    tmp_x = []
    for d in range(vote_list[0].shape[0]):
        # 1. last election: Blue, Red, Gray
        #    Blue/All
        # 2. neighbours
        # 3. one (1)
        lo = pool_d[y].iloc[d,:]
        neigh = neighbours[lo.name.lower()]
        avg_n = [pool_d[y].loc[n.upper()][0]/pool_d[y].loc[n.upper()].sum() for␣
↪n in neigh]
        avg_n = sum(avg_n)/len(neigh)
        tmp_x.append([lo[0]/lo.sum(), avg_n, 1])
    X.append(tmp_x)
```

[21]:
```
X = np.array(X)
X.shape
```

[21]:
```
(18, 16, 3)
```

0.8 Prepare Y

[22]:
```
Y = []
for y in range(1,pool_data_middle.shape[0]):
    # iterate over districts
    tmp_y = []
    for d in range(vote_list[0].shape[0]):
        # 1. last election: Blue, Red, Gray
        #    Blue/All
        # 2. neighbours
        # 3. one (1)
        lo = pool_d[y].iloc[d,:]
```

```
        tmp_y.append([lo[0]/lo.sum()])
    Y.append(tmp_y)
```

[23]:
```
Y = np.array(Y)
Y.shape
```

[23]: (18, 16, 1)

0.9 Parameters to be estimated

- waga poprzednie wybory
- waga wpływu sąsiadów (avg over nighbours/ weighted avg)
- pole zewnętrzne

0.10 Process

- input,
- warstwy,
- output (parametry lub wagi),
- output*parameters (the real output - wynik wyborów)

OR - input (parameters), - simple network to get the weight = parameters to multiply, - output (next election)

0.11 Training phase (looking for parameters)

Functions for models

[24]:
```
X.shape
```

[24]: (18, 16, 3)

0.11.1 Models with percentage of Blue support per district

[25]:
```
def model_percent(a,x):
    '''
    INPUT:
    a - vector of weights 16x3
    x - vector of input data 18x16x3
    OUTPUT:
    y - predicted value in (0,1)
    '''
    d0 = x.shape[0] if (len(x.shape) == 3) else 1

    a = np.repeat(a, d0, 0)
    x = x.reshape(-1, 3)
    #return 1 / (1+np.exp(-np.sum(x.dot(a.T))))
    y = 1 / (1+np.exp(-np.sum(x*a, 1, keepdims=True) ))
```

```
    return y

def grad_percent(a,x,y):
    '''
    INPUT:
    a - vector of weights 16x3
    x - vector of input data 18x16x3
    '''
    #return a * np.exp(-x.T.dot(a)) / (1+np.exp(-x.T.dot(a)))**2
    #return a*np.exp(-np.sum(x*a,1,keepdims=True)) / (1+np.exp(-np.
↪sum(x*a,1,keepdims=True)))**2
    d0 = x.shape[0] if (len(x.shape) == 3) else 1

    a = np.repeat(a, d0, 0)
    x = x.reshape(-1, 3)
    y = y.reshape(-1, 1)
    y1 = -(2 *
          ( y - 1/(1+np.exp(-np.sum(x.dot(a.T),1,keepdims=True))) ) *
          1/(1+np.exp(-np.sum(x.dot(a.T),1,keepdims=True)))**2 *
          np.exp(-np.sum(x.dot(a.T),1,keepdims=True)) *
          x)

    y1 = -(2 *
          ( y - 1/(1+np.exp(-np.sum(x*a,1,keepdims=True))) ) *
          1/(1+np.exp(-np.sum(x*a,1,keepdims=True)))**2 *
          np.exp(-np.sum(x*a,1,keepdims=True)) *
          x)

    return y1
```

0.11.2 Setup for testing model

[26]:
```
neigh_ndx = []
for d in range(X.shape[1]):
    # 1. last election: Blue, Red, Gray
    #    Blue/All
    # 2. neighbours
    # 3. one (1)
    lo = par_in_reg_list[0].iloc[d,:]
    neigh = neighbours[lo.name.lower()]
    indexs = par_in_reg_list[0].index.values
    neigh_ndx.append(np.searchsorted(indexs, np.char.upper(neigh)))
```

[27]:
```
def prepare_input(y):
    tmp_x = np.zeros((y.shape[0],3))
    for d in range(y.shape[0]):
```

```
        neigh = neigh_ndx[d]
        avg_n = [y[n,0]/np.sum(y[neigh,0]) for n in neigh]
        avg_n = sum(avg_n)/len(neigh)
        tmp_x[d] = np.array([y[d,0], avg_n, 1])
    return(tmp_x)
```

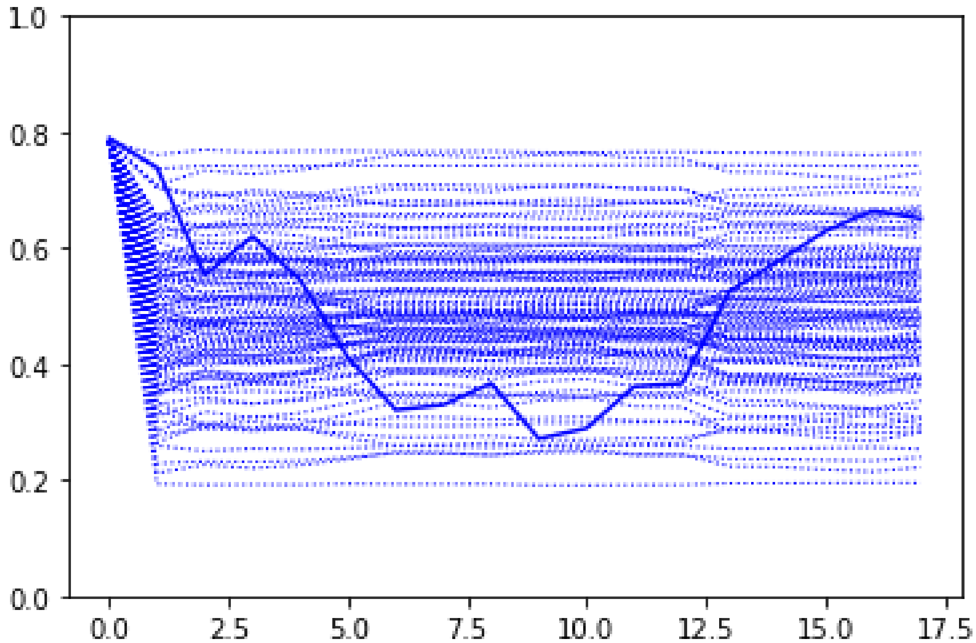
[28]:
```
def model(a,x,Y):
    y = Y[0]
    loss = []
    out = np.zeros(Y.shape)
    out[0] = y
    for year in range(1,X.shape[0]):
        xi = prepare_input(y)
        y = model_percent(a,xi)
        loss.append(np.sum((y - Y[year])**2))
        #print(y.shape,'loss:', np.sum((y - Y[year])**2))
        out[year] = y
    return loss, out
```

[29]:
```
def model_prev(a,x,Y):
    y = Y[0]
    loss = []
    out = np.zeros(Y.shape)
    out[0] = y
    for year in range(1,X.shape[0]):
        xi = prepare_input(Y[year])
        y = model_percent(a,xi)
        loss.append(np.sum((y - Y[year])**2))
        #print(y.shape,'loss:', np.sum((y - Y[year])**2))
        out[year] = y
    return loss, out
```

0.12 Setup random a

[30]:
```
loss_p = np.inf
loss_v = np.inf

a_avg = np.random.rand(X.shape[1],X.shape[2])
a_all = a_avg
a_nxt = a_avg
a_wgth = a_avg
a_tmp = a_avg

step = 1
beta = 0.01
```

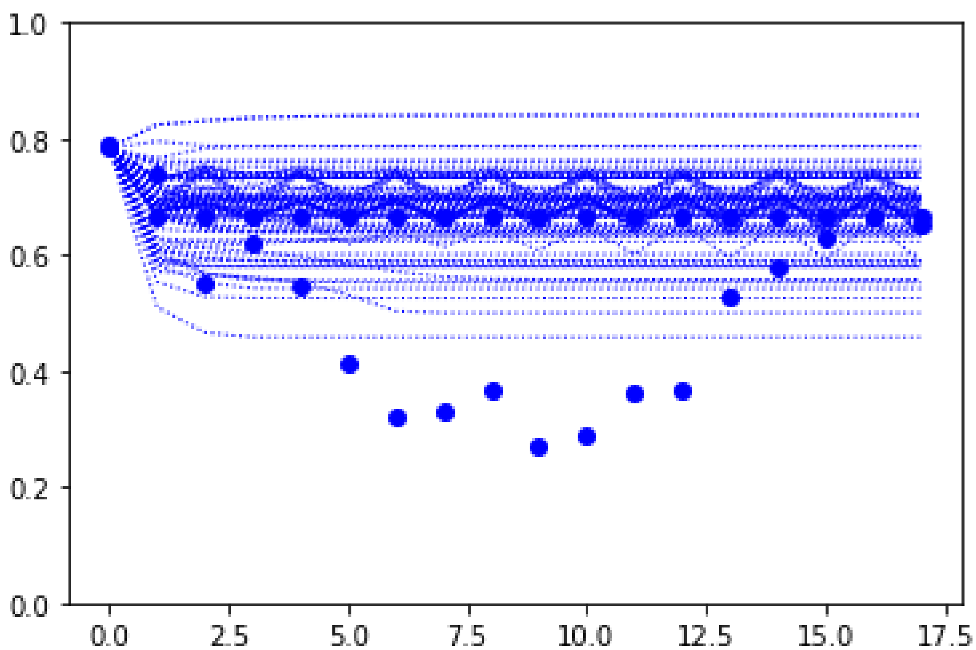
```
[31]: for i in range(100):
          a_avg = (np.random.rand(X.shape[1],X.shape[2])-0.5)*10
          l, o = model_prev(a_avg,X,Y)

          plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)

      plt.plot(pool_data_middle['Blue'].values[1:],'b')
      plt.ylim(0,1)
      plt.show()
```



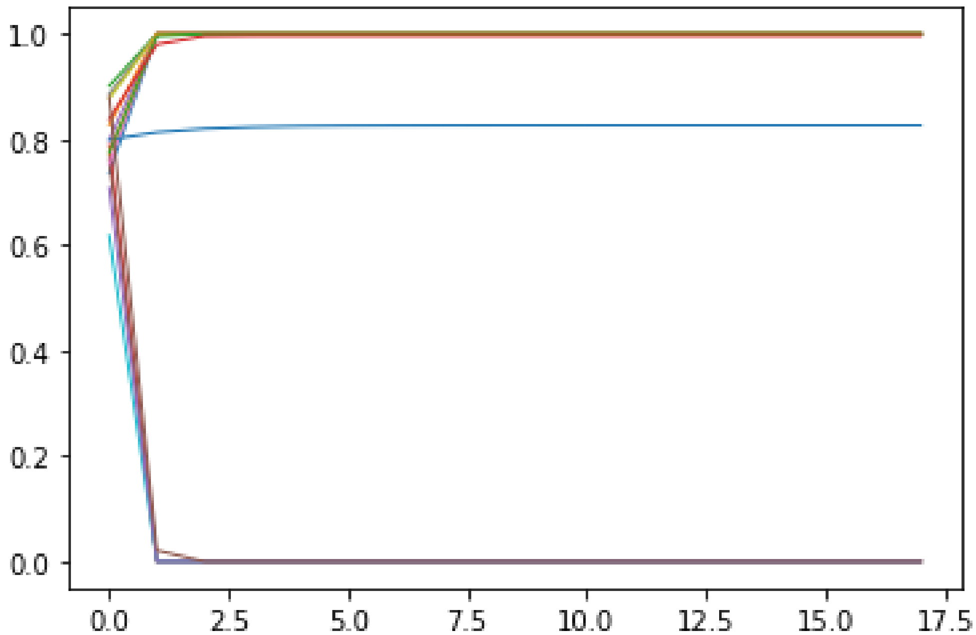
0.13 All at once

```
[42]: a_avg = (np.random.rand(X.shape[1],X.shape[2])-0.5)
      for epoch in range(10**4):
          grad = grad_percent(a_avg,X,Y).reshape(18,16,3)

          #if epoch==0: print('first grad max/min:', np.max(grad),'/',np.min(grad))
          grad = np.sum(grad, axis=0)

          #if epoch==0: print('first grad max/min:', np.max(grad),'/',np.min(grad))
          a_avg = a_avg - step*grad

          #if epoch%50==0:
          #    if np.sum((model_percent(ap,X) - Y.reshape(-1,1))**2) < loss_p: step␣
       ↪*= (1+beta)
```

```
#    else: step /= (1-beta)

    loss_p = np.sum((model_percent(a_avg,X) - Y.reshape(-1,1))**2)

    if loss_p == np.nan:
        break

    if epoch%100==0:
        #print('loss sum:',loss_p)
        l, o = model(a_avg,X,Y)
        plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)
#l1, o1 = model_prev(a_avg,X,Y)

#plt.plot(np.average(o1,1, voter_w[1]),'bs', linewidth=1)

l, o = model(a_avg,X,Y)
plt.plot(np.average(o,1, voter_w[1]),'b-o', linewidth=1)
plt.plot(pool_data_middle['Blue'].values[1:],'bo')
plt.ylim(0,1)
plt.show()
```



[44]:
```
plt.plot(o.reshape(18,16), linewidth=1)
plt.show()
```

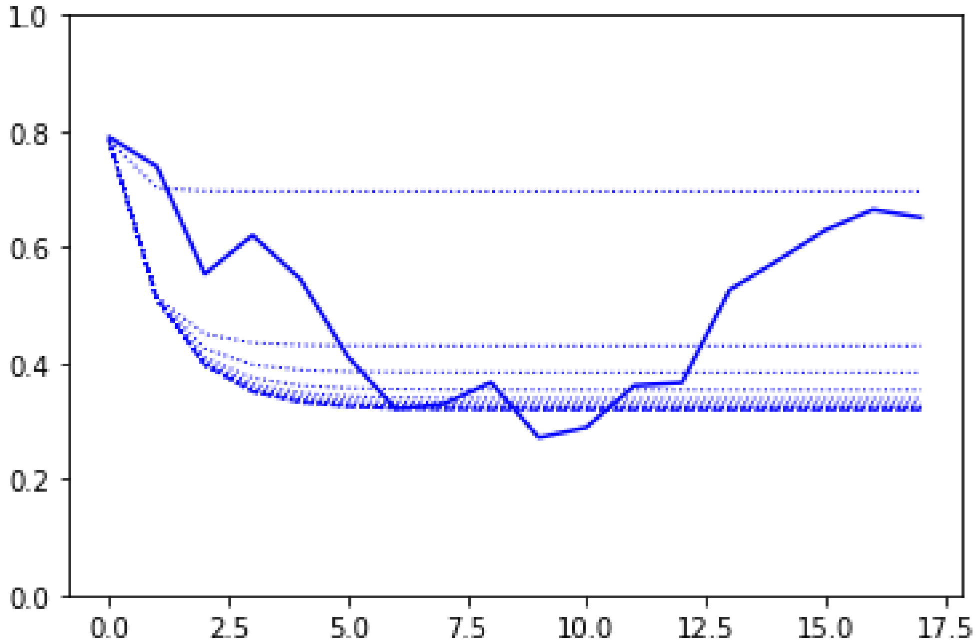


0.14 Shuffle years and on at once

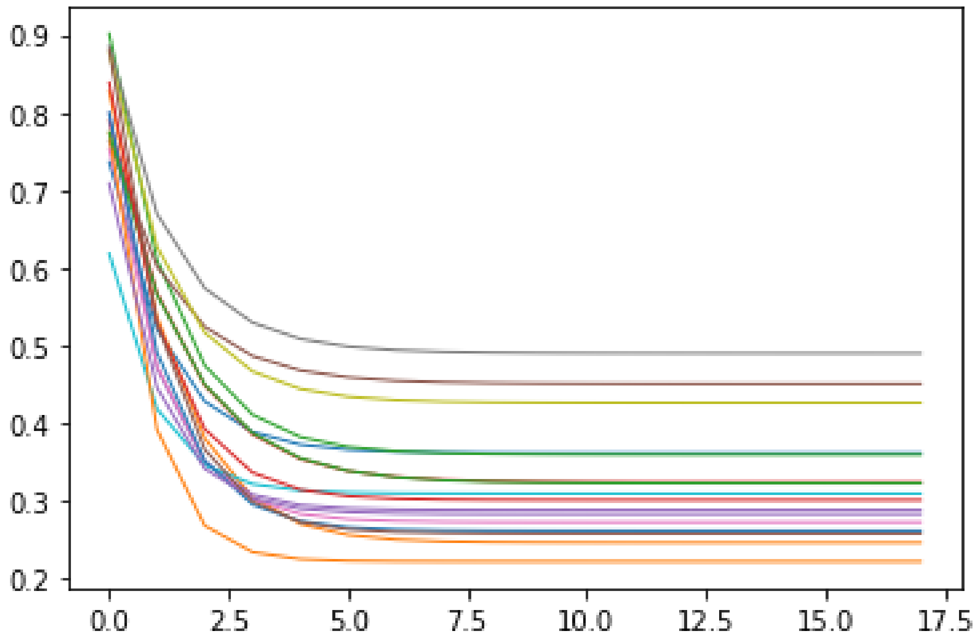
[45]:
```
a_all = np.random.rand(X.shape[1],X.shape[2])
loss_l = np.inf
step = 0.1

for epoch in range(10**3):
    shuffle_i = np.arange(X.shape[0])
    np.random.shuffle(shuffle_i)
    loss_p = 0
    for i in shuffle_i:
        grad = grad_percent(a_all,X[i],Y[i])#.reshape(18,16,3)
        #grad = np.sum(grad, axis=0)
        a_all = a_all - step*grad

        #if epoch%50==0:
        #    if np.sum((model_percent(ap,X) - Y.reshape(-1,1))**2) < loss_p:␣
↪step *= (1+beta)
        #    else: step /= (1-beta)

        loss_p += np.sum((model_percent(a_all,X[i]) - Y[i].reshape(-1,1))**2)

    loss_l = loss_p

    if epoch%100==0:
```

```
        print('loss sum:',loss_p)
        l, o = model(a_all,X,Y)
        plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)

l, o = model(a_all,X,Y)
plt.plot(np.average(o,1, voter_w[1]),'b--', linewidth=1)
plt.plot(pool_data_middle['Blue'].values[1:],'b')
plt.ylim(0,1)
plt.show()
```

```
loss sum: 20.243371556199705
loss sum: 2.8444056144915084
loss sum: 2.1010098691918593
loss sum: 1.9135139964124652
loss sum: 1.8583986926405451
loss sum: 1.8411791534702378
loss sum: 1.8342114055670744
loss sum: 1.8311296097680807
loss sum: 1.8282755683359018
loss sum: 1.8262078797311656
```



[46]:
```
plt.plot(o.reshape(18,16), linewidth=1)
plt.show()
```



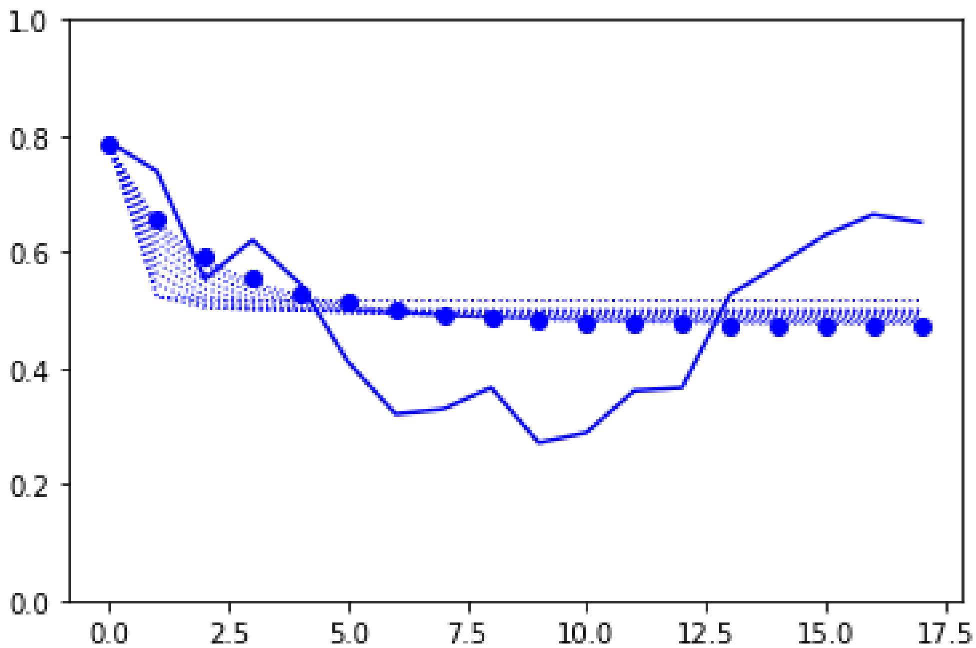
0.15 Use output -> next input (grad each step)

[44]:
```
a_step = np.random.rand(X.shape[1],X.shape[2]) - 0.5
```

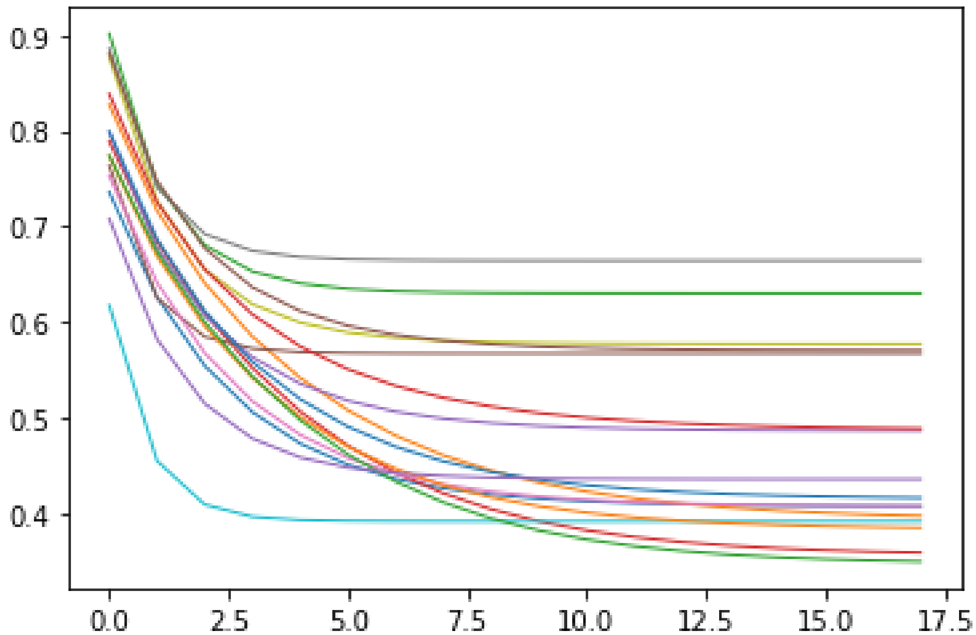
[45]:
```
for epoch in range(10**3):
    print(epoch,end='\r')
    loss_p = 0
    y = Y[0]
    for i in range(X.shape[0]):
        xi = prepare_input(y)
        y = model_percent(a_step,xi)

        grad = grad_percent(a_step,xi,Y[i])#.reshape(18,16,3)
        #grad = np.sum(grad, axis=0)
        a_step = a_step - step*grad

        loss_p += np.sum((model_percent(a_step,xi) - Y[i].reshape(-1,1))**2)

    if epoch%100==0:
        l, o = model(a_step,X,Y)
        plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)

l, o = model(a_step,X,Y)
plt.plot(np.average(o,1, voter_w[1]),'bo', linewidth=1)
```

```
plt.plot(pool_data_middle['Blue'].values[1:],'b')
plt.ylim(0,1)
plt.show()
```

999



[46]:
```
l, o = model(a_step,X,Y)
plt.plot(o.reshape(18,16), linewidth=1)
plt.show()
```



0.16 Use output -> next input (grad each epoch)

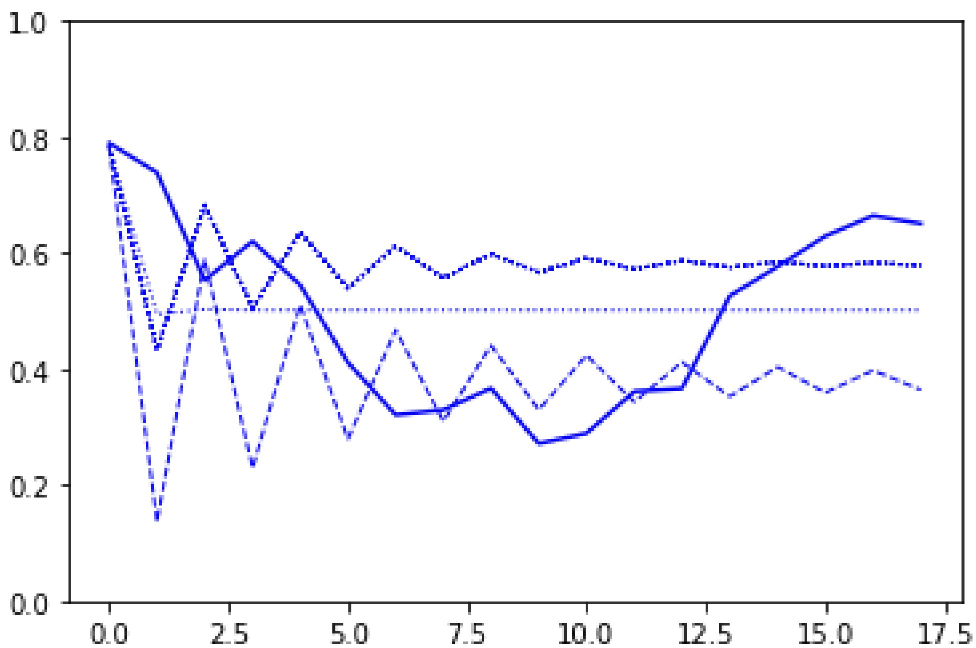
[47]:
```
loss_l = np.inf
a_nxt = np.random.rand(X.shape[1],X.shape[2]) - 0.5
```

[48]:
```
# https://towardsdatascience.com/introduction-to-evolution-strategy-1b78b9d48385

for epoch in range(10**3):
    print(epoch,end='\r')
    loss_p = 0
    y = Y[0]
    grad = 0

    for i in range(1,X.shape[0]):
        xi = prepare_input(y)
        y = model_percent(a_nxt,xi)
        grad += grad_percent(a_nxt,xi,Y[i])
        loss_p += np.sum((model_percent(a_nxt,xi) - Y[i].reshape(-1,1))**2)

    grad = np.sum(grad, axis=0)
    a_nxt = a_nxt - step*grad

    if epoch%100==0:
        l, o = model(a_nxt,X,Y)
        plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)
```

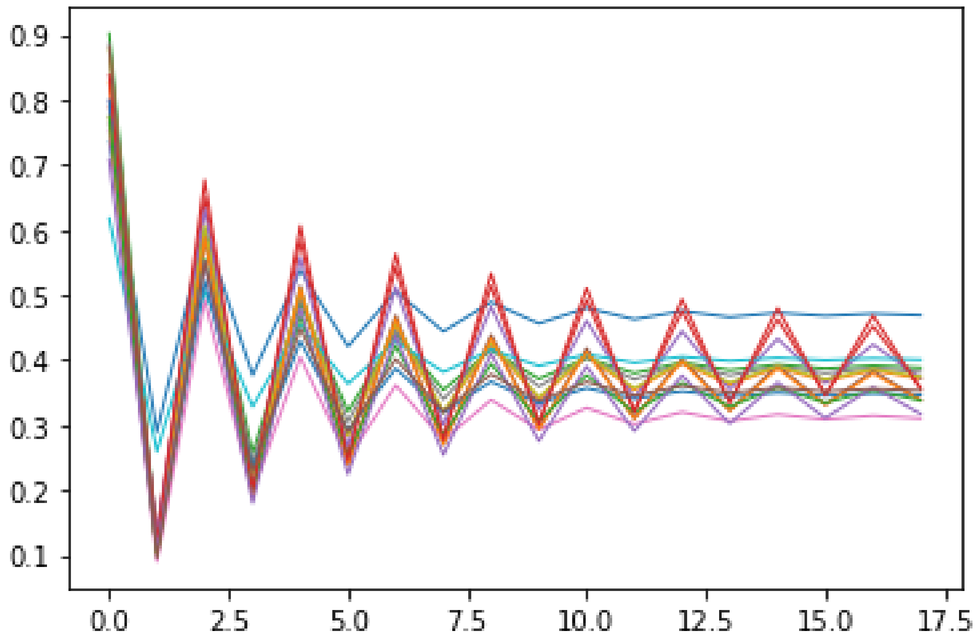
```
l, o = model(a_nxt,X,Y)

plt.plot(np.average(o,1, voter_w[1]),'b--', linewidth=1)
plt.plot(pool_data_middle['Blue'].values[1:],'b')
plt.ylim(0,1)
```

999

[48]: (0.0, 1.0)



[49]:
```
plt.plot(o.reshape(18,16), linewidth=1)
plt.show()
```



0.17 Use output -> next input (grad each step) + weights(linear weight)

[50]:
```
a_step_wgth = np.random.rand(X.shape[1],X.shape[2]) - 0.5
```

[ ]:
```
for epoch in range(10**3):
    loss_p = 0
    y = Y[0]
    for i in range(X.shape[0]):
        xi = prepare_input(y)
        y = model_percent(a_step_wgth,xi)

        grad = grad_percent(a_step_wgth,xi,Y[i])#.reshape(18,16,3)
        #grad = np.sum(grad, axis=0)
        a_step_wgth = a_step_wgth - a_step_wgth*grad*(i+1)/X.shape[0]

        loss_p += np.sum((model_percent(a_step_wgth,xi) - Y[i].
 ↪reshape(-1,1))**2)

    if epoch%100==0:
        print('loss sum:',loss_p)
        l, o = model(a_step_wgth,X,Y)
        plt.plot(np.average(o,1, voter_w[1]),'b:', linewidth=1)

l, o = model(a_step_wgth,X,Y)
plt.plot(np.average(o,1, voter_w[1]),'b--', linewidth=1)
```

```
plt.plot(np.mean(o,1),'b--')
plt.plot(pool_data_middle['Blue'].values[1:],'b')
plt.ylim(0,1)
plt.show()
```

```
loss sum: 9.793974992718836
loss sum: 43.31084037793109
loss sum: 45.87577845963534
loss sum: 45.870040148890276
loss sum: 45.85237966615744
loss sum: 45.821132971437365
loss sum: 45.77152759622534
```

[ ]:
```
plt.plot(o.reshape(18,16), linewidth=1)
plt.show()
```

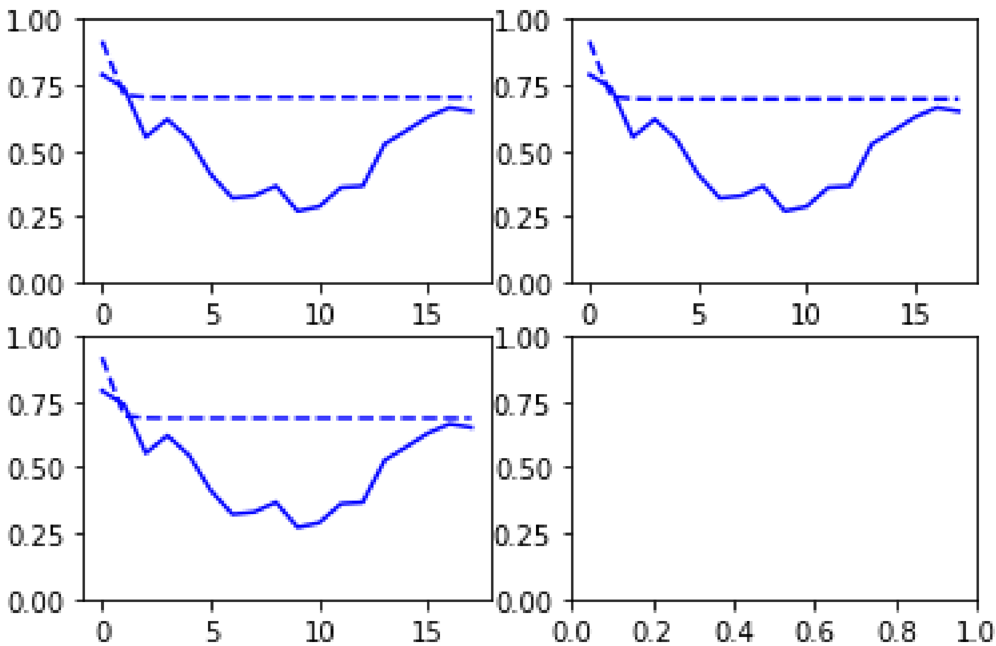
0.18 Use output -> next input (grad each epoch) + weights(linear weight)

[100]:
```
# https://towardsdatascience.com/introduction-to-evolution-strategy-1b78b9d48385
loss_l = np.inf
a_wgth = a_tmp
```

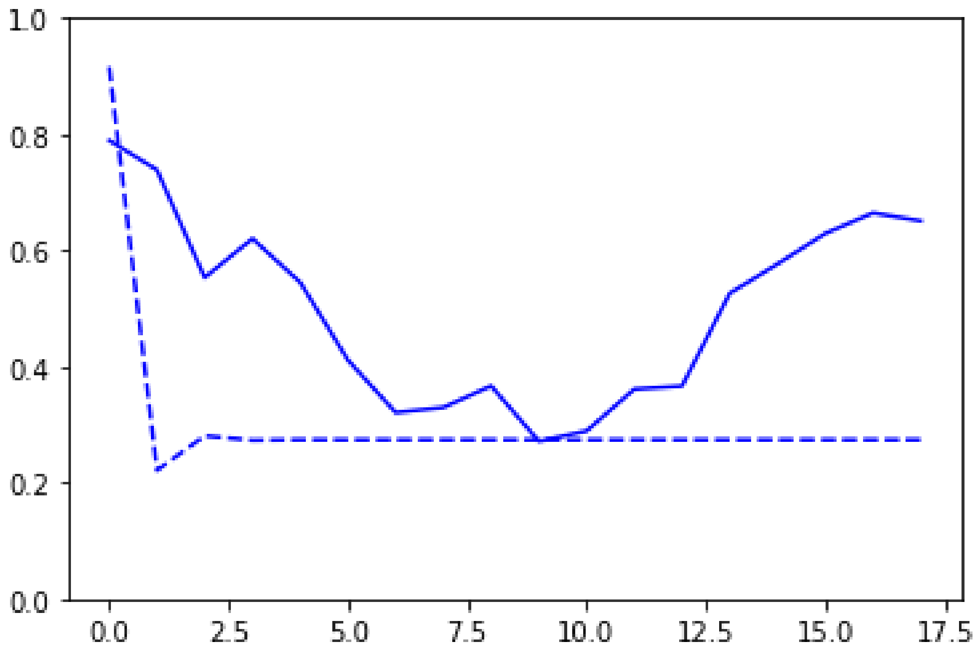
[101]:
```
fig, ax = plt.subplots(nrows=2, ncols=2)

for epoch in range(10**3):
    loss_p = 0
    y = Y[0]
    grad = np.zeros(X[0].shape)

    for i in range(1,X.shape[0]):
        xi = prepare_input(y)
        y = model_percent(a_wgth,xi)
        grad += grad_percent(a_wgth,xi,Y[i])*(i+1)/X.shape[0]
        loss_p += np.sum((model_percent(a_wgth,xi) - Y[i].reshape(-1,1))**2)

    grad = np.sum(grad, axis=0)

    if loss_p > loss_l:
        print('loss sum:',loss_p)
        break
    a_wgth = a_wgth - step*grad
    loss_l = loss_p

    if epoch%100==0:
        print('loss sum:',loss_p)
        n = epoch//(100)
```

```
        if(n<9):
            l, o = model(a_wgth,X,Y)
            ax[n//2,n%2].plot(np.mean(o,1),'b--')
            ax[n//2,n%2].plot(pool_data_middle['Blue'].values[1:],'b')
            ax[n//2,n%2].set_ylim(0,1)

l, o = model(a_wgth,X,Y)

plt.plot(np.mean(o,1),'b--')
plt.plot(pool_data_middle['Blue'].values[1:],'b')
plt.ylim(0,1)
```

```
loss sum: 29.71564767715839
loss sum: 29.088269778970982
loss sum: 27.957788865802776
loss sum: 29.845451877176824
```

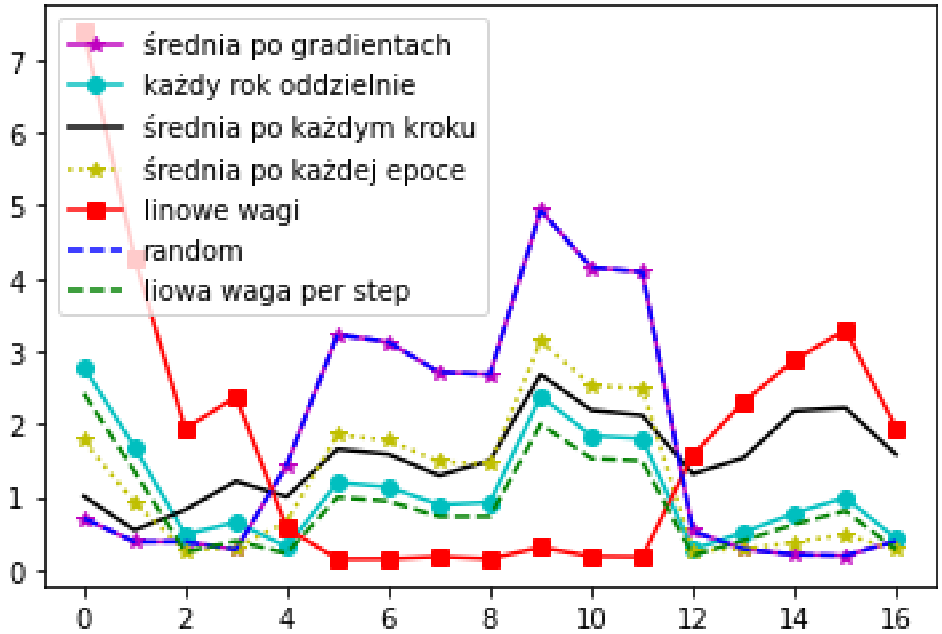


[101]: (0.0, 1.0)



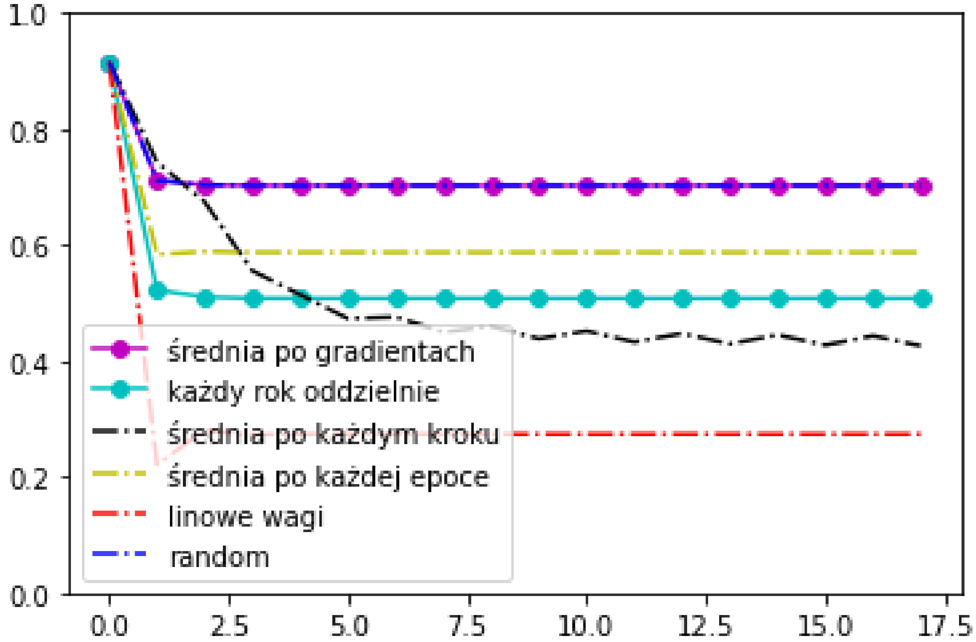
[102]:
```
l_avg, o_avg = model(a_avg,X,Y)
l_all, o_all = model(a_all,X,Y)
l_step, o_step = model(a_step,X,Y)
l_nxt, o_nxt = model(a_nxt,X,Y)
l_wgth, o_wgth = model(a_wgth,X,Y)
l_rnd, o_rnd = model(a_tmp,X,Y)
l_tp_w, o_stp_w = model(a_step_wgth,X,Y)

plt.plot(l_avg,'m-*')
plt.plot(l_all,'c-o')
plt.plot(l_step,'k-')
plt.plot(l_nxt,'y:*')
plt.plot(l_wgth,'r-s')
plt.plot(l_rnd,'b--')
plt.plot(l_tp_w,'g--')

#plt.plot([0,16],[np.mean(l_avg),np.mean(l_avg)],'m--')
#plt.plot([0,16],[np.mean(l_all),np.mean(l_all)],'c--')
#plt.plot([0,16],[np.mean(o_step),np.mean(o_step)],'k--')
#plt.plot([0,16],[np.mean(l_nxt),np.mean(l_nxt)],'y--')
#plt.plot([0,16],[np.mean(l_wgth),np.mean(l_wgth)],'r--')

plt.legend(['średnia po gradientach',
            'każdy rok oddzielnie',
            'średnia po każdym kroku',
```

```
             'średnia po każdej epoce',
             'linowe wagi',
             'random',
             'liowa waga per step'
            ])

plt.savefig('model/compare_gradient_outputs.pdf', bbox_inches='tight')#,␣
 ↪format='eps')
```



[103]:
```
plt.plot(o_avg.mean(1), 'm-o')
plt.plot(o_all.mean(1), 'c-o')
plt.plot(o_step.mean(1), 'k-.')
plt.plot(o_nxt.mean(1), 'y-.')
plt.plot(o_wgth.mean(1), 'r-.')
plt.plot(o_rnd.mean(1), 'b-.')

plt.legend(['średnia po gradientach',
            'każdy rok oddzielnie',
            'średnia po każdym kroku',
            'średnia po każdej epoce',
            'linowe wagi',
            'random'
           ])
plt.ylim(0,1)
```

[103]: (0.0, 1.0)



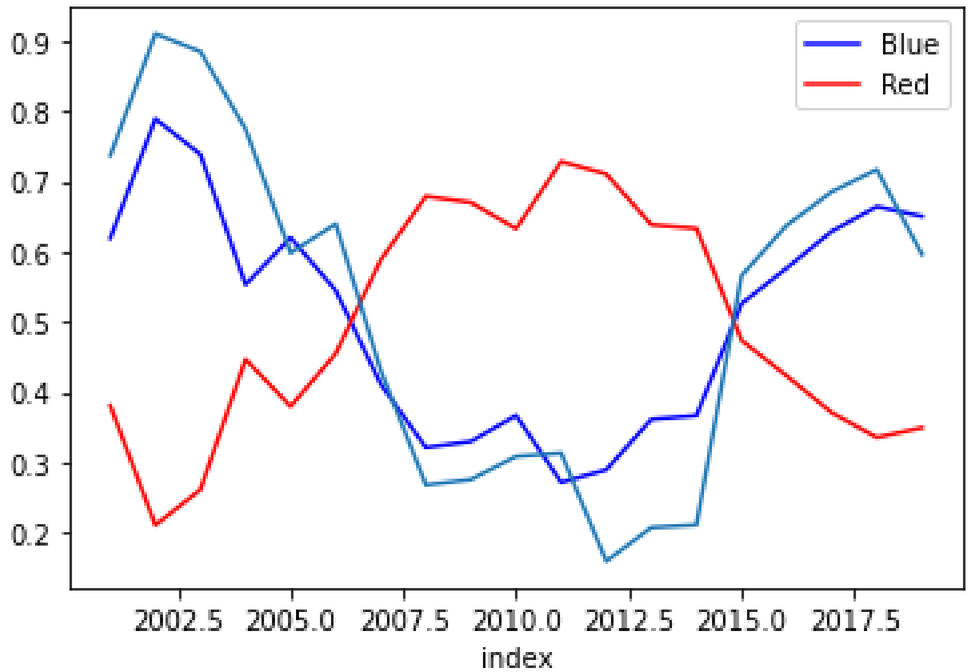
0.19 Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)

https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/ http://proceedings.mlr.press/v57/zhao16.pdf https://towardsdatascience.com/time-series-forecasting-with-recurrent-neural-networks-74674e289816 https://developer.apple.com/documentation/coreml/cor https://www.youtube.com/watch?v=WCUNPb-5EYI

[106]:
```
pool_d_real = [pool.divide(pool.sum(1),0) for pool in pool_d]
```

[109]:
```
pool_d_real2  = [(pool['Blue'] * voters[1].values).sum() for pool in␣
 ↪pool_d_real]
```

[113]:
```
pool_data_middle = pd.read_csv('dane_years/pools_data/percent_votes.csv',␣
 ↪index_col=0).iloc[:,:-1]
pool_data_middle = pool_data_middle.divide(pool_data_middle.sum(1),0)
pool_data_middle.plot(color=['b','r'])
plt.plot(pool_data_middle.index, pool_d_real2)
plt.show()
```


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0
2017.5
index
Blue
Red

```
[ ]:
```