chess_data (4)

April 29, 2020

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
%matplotlib inline
import seaborn as sns
```

In [3]:
```
df = pd.read_csv('chess.csv')
```

In [5]:
```
df[['moves', 'opening_name']]
```

Out[5]:
```
                                                   moves  \
0      d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3 Ba5...
1      d4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe5 Nxe5 Qd4 Nc6...
2      e4 e5 d3 d6 Be3 c6 Be2 b5 Nd2 a5 a4 c5 axb5 Nc...
3      d4 d5 Nf3 Bf5 Nc3 Nf6 Bf4 Ng4 e3 Nc6 Be2 Qd7 O...
4      e4 e5 Nf3 d6 d4 Nc6 d5 Nb4 a3 Na6 Nc3 Be7 b4 N...
...                                                  ...
20053  d4 f5 e3 e6 Nf3 Nf6 Nc3 b6 Be2 Bb7 O-O Be7 Ne5...
20054  d4 d6 Bf4 e5 Bg3 Nf6 e3 exd4 exd4 d5 c3 Bd6 Bd...
20055  d4 d5 Bf4 Nc6 e3 Nf6 c3 e6 Nf3 Be7 Bd3 O-O Nbd...
20056  e4 d6 d4 Nf6 e5 dxe5 dxe5 Qxd1+ Kxd1 Nd5 c4 Nb...
20057  d4 d5 Bf4 Na6 e3 e6 c3 Nf6 Nf3 Bd7 Nbd2 b5 Bd3...

                                 opening_name
0            Slav Defense: Exchange Variation
1      Nimzowitsch Defense: Kennedy Variation
2       King's Pawn Game: Leonardis Variation
3      Queen's Pawn Game: Zukertort Variation
4                            Philidor Defense
...                                       ...
20053                           Dutch Defense
20054                            Queen's Pawn
20055         Queen's Pawn Game: Mason Attack
20056                           Pirc Defense
20057         Queen's Pawn Game: Mason Attack

[20058 rows x 2 columns]
```

```
In [168]: df['moves'] = [' '.join(st.split(' ')[:10]) for st in df['moves']]

In [169]: df[['moves','opening_name']]

Out[169]:                                           moves  \
          0        d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+
          1        d4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe5 Nxe5
          2              e4 e5 d3 d6 Be3 c6 Be2 b5 Nd2 a5
          3          d4 d5 Nf3 Bf5 Nc3 Nf6 Bf4 Ng4 e3 Nc6
          4             e4 e5 Nf3 d6 d4 Nc6 d5 Nb4 a3 Na6
          ...                                         ...
          20053        d4 f5 e3 e6 Nf3 Nf6 Nc3 b6 Be2 Bb7
          20054      d4 d6 Bf4 e5 Bg3 Nf6 e3 exd4 exd4 d5
          20055        d4 d5 Bf4 Nc6 e3 Nf6 c3 e6 Nf3 Be7
          20056  e4 d6 d4 Nf6 e5 dxe5 dxe5 Qxd1+ Kxd1 Nd5
          20057        d4 d5 Bf4 Na6 e3 e6 c3 Nf6 Nf3 Bd7

                                          opening_name
          0            Slav Defense: Exchange Variation
          1      Nimzowitsch Defense: Kennedy Variation
          2       King's Pawn Game: Leonardis Variation
          3      Queen's Pawn Game: Zukertort Variation
          4                             Philidor Defense
          ...                                        ...
          20053                            Dutch Defense
          20054                             Queen's Pawn
          20055         Queen's Pawn Game: Mason Attack
          20056                            Pirc Defense
          20057         Queen's Pawn Game: Mason Attack

          [20058 rows x 2 columns]

In [170]: p = set(df['opening_name'])
          with open("jason.txt", "w") as f:
              for k in p:
                  f.write(k + "\n")

In [138]: finalList = []
          from collections import Counter
          for game in df['opening_name']:
                  if ':' in game:
                          game = game[:game.find(':')]
                  if '|' in game:
                          game = game[:game.find('|')]
                  game = game.replace('\n', '')

                  finalList.append(game)
          df['opening'] = finalList
```

```
print(list(map(lambda a:a[0], Counter(finalList).most_common(30))))
```

```
['Sicilian Defense', 'French Defense', "Queen's Pawn Game", 'Italian Game', "King's Pawn Game"
```

```
In [139]: ops = ['Sicilian Defense', 'French Defense', 'Ruy Lopez', 'Italian Game', 'English Op
          mdf = df[['moves','opening']]

In [140]: filtered_mdf = mdf.query("opening in list(['Sicilian Defense', 'French Defense', 'Ruy

In [418]: kval = 12
          fact=1000

In [419]: m = np.concatenate((np.ones((1,8)),np.zeros((4,8))), axis=0)

In [420]: s = np.array([np.array([5,2,3,9,kval,3,2,5])])
          B = fact*np.concatenate((s,m,-np.ones((1,8)),-s))

In [421]: board = pd.DataFrame(B, columns=list('abcdefgh'))

In [422]: board.rename(lambda i: 8-i, axis=0)
```

Out[422]:

	a	b	c	d	e	f	g	h
8	5000.0	2000.0	3000.0	9000.0	12000.0	3000.0	2000.0	5000.0
7	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-1000.0	-1000.0	-1000.0	-1000.0	-1000.0	-1000.0	-1000.0	-1000.0
1	-5000.0	-2000.0	-3000.0	-9000.0	-12000.0	-3000.0	-2000.0	-5000.0

```
In [423]: chr((ord('a')+2))

Out[423]: 'c'

In [424]: move = lambda cp,i,j: (chr((ord(cp[0])+j)),int(cp[1])+i)

In [425]: move('d4',1,0)

Out[425]: ('d', 5)

In [426]: trans = {5:'R', 2:"N",3:"B", 9:"Q", kval:"K", 1:"P"}
          cur = list(trans.keys())
          for key in cur:
              trans[fact*key] = "b"+trans[key]
              trans[-fact*key] = "w"+trans[fact*key][1]
          transback = {'bR': 5, "bB":3, "bN":2, "bQ":9, "bK":kval, "bP":1}
          cur = list(transback.keys())
          for key in cur:
              transback[key] *= fact
              transback['w'+key[1]] = -transback[key]
          trans
```

```
Out[426]: {5: 'R',
 2: 'N',
 3: 'B',
 9: 'Q',
 12: 'K',
 1: 'P',
 5000: 'bR',
 -5000: 'wR',
 2000: 'bN',
 -2000: 'wN',
 3000: 'bB',
 -3000: 'wB',
 9000: 'bQ',
 -9000: 'wQ',
 12000: 'bK',
 -12000: 'wK',
 1000: 'bP',
 -1000: 'wP'}
```

```
In [427]: boardLetter = board.apply(lambda r: [trans.get(int(n), "") for n in r], axis=0).renar
          boardBack = boardLetter.apply(lambda r: [transback.get(n, 0) for n in r], axis=0).rer
          numForm = lambda b: b.apply(lambda r: [transback.get(n, 0) for n in r], axis=0)
```

```
In [428]: boardLetter
```

Out[428]:

	a	b	c	d	e	f	g	h
8	bR	bN	bB	bQ	bK	bB	bN	bR
7	bP	bP	bP	bP	bP	bP	bP	bP
6								
5								
4								
3								
2	wP	wP	wP	wP	wP	wP	wP	wP
1	wR	wN	wB	wQ	wK	wB	wN	wR

```
In [429]: boardBack
```

Out[429]:

	a	b	c	d	e	f	g	h
0	5000	2000	3000	9000	12000	3000	2000	5000
1	1000	1000	1000	1000	1000	1000	1000	1000
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	-1000	-1000	-1000	-1000	-1000	-1000	-1000	-1000
7	-5000	-2000	-3000	-9000	-12000	-3000	-2000	-5000

```
In [430]: move("a2", 2, 0)
```

```
Out[430]: ('a', 4)
```

```
In [431]: game1 = 'd4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3'
          game2 = 'd4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe5 Nxe5 Qd4 Nc6'
          game3 = 'd4 f5 e3 e6 Nf3 Nf6 Nc3 b6 Be2 Bb7'
          game4 = 'd4 d5 Bf4 Nc6 e3 Nf6 c3 e6 Nf3 Be7 Bd3 O-O Nbd'
          game5 = 'e4 d6 d4 Nf6 e5 dxe5 dxe5 Qxd1+ Kxd1 Nd5 c4'
```

```
In [432]: def inc(letter, times = 1):
                  if len(letter) == 2: return [inc(letter[0]),inc(letter[1])]
                  return(chr(ord(letter) + times))

          def dec(letter, times = 1):
                  if len(letter) == 2: return [dec(letter[0]),dec(letter[1])]
                  return(chr(ord(letter) - times))

          def inBounds(char):
                  if len(char) == 2: return inBounds(char[0]) and inBounds(char[1])
                  return char in (['a','b','c','d','e','f','g','h'] + list(map(lambda a: str(a

          def prevMoves(move, turn):

                  #pawn
                  if move[0] == move[0].lower():
                          if turn == 'b':
                                  if move[1] == '5':
                                          return [f'{move[0]}6', f'{move[0]}7'], 'P'
                                  else:
                                          return [f'{move[0]}{inc(move[1])}'], 'P'
                          else:
                                  if move[1] == '4':
                                          return [f'{move[0]}3', f'{move[0]}2'], 'P'
                                  else:
                                          return [f'{move[0]}{dec(move[1])}'], 'P'

                  #bishop
                  possibleMoves = set()
                  dir1 = [move[1], move[2]][:]
                  dir2 = [move[1], move[2]][:]
                  dir3 = [move[1], move[2]][:]
                  dir4 = [move[1], move[2]][:]

                  if move[0] == 'B':
                          while inBounds(dir1):
                                  possibleMoves.add(''.join(dir1))
                                  dir1 = inc(dir1)
                          while inBounds(dir2):
                                  possibleMoves.add(''.join(dir2))
                                  dir2 = dec(dir2)
                          while inBounds(dir3):
```

```
                    possibleMoves.add(''.join(dir3))
                    dir3[0] = inc(dir3[0])
                    dir3[1] = dec(dir3[1])
            while inBounds(dir4):
                    possibleMoves.add(''.join(dir4))
                    dir4[0] = dec(dir4[0]); dir4[1] = inc(dir4[1])
            return possibleMoves, 'B'

#rook
    if move[0] == 'R':
            while inBounds(dir1):
                    possibleMoves.add(''.join(dir1))
                    dir1[0] = inc(dir1[0])
            while inBounds(dir2):
                    possibleMoves.add(''.join(dir2))
                    dir2[0] = dec(dir2[0])
            while inBounds(dir3):
                    possibleMoves.add(''.join(dir3))
                    dir3[1] = inc(dir3[1])
            while inBounds(dir4):
                    possibleMoves.add(''.join(dir4))
                    dir4[1] = dec(dir4[1])
            return possibleMoves, 'R'

    dir5 = [move[1], move[2]][:]
    dir6 = [move[1], move[2]][:]
    dir7 = [move[1], move[2]][:]
    dir8 = [move[1], move[2]][:]

#knight
    if move[0] == 'N':
            dir1[0] = inc(dir1[0], 2); dir1[1] = inc(dir1[1])
            dir2[0] = inc(dir2[0], 2); dir2[1] = dec(dir2[1])
            dir3[0] = dec(dir3[0], 2); dir3[1] = inc(dir3[1])
            dir4[0] = dec(dir4[0], 2); dir4[1] = dec(dir4[1])
            dir5[0] = inc(dir5[0], 1); dir5[1] = inc(dir5[1], 2)
            dir6[0] = inc(dir6[0], 1); dir6[1] = dec(dir6[1], 2)
            dir7[0] = dec(dir7[0], 1); dir7[1] = inc(dir7[1], 2)
            dir8[0] = dec(dir8[0], 1); dir8[1] = dec(dir8[1], 2)
            directionList = [dir1, dir2, dir3, dir4, dir5, dir6, dir7, dir8]
            [possibleMoves.add(''.join(a)) for a in filter(inBounds, directionLis
            return possibleMoves, 'N'

#queen
    if move[0] == 'Q':
            while inBounds(dir1):
                    possibleMoves.add(''.join(dir1))
                    dir1 = inc(dir1)
```

```
                while inBounds(dir2):
                        possibleMoves.add(''.join(dir2))
                        dir2 = dec(dir2)
                while inBounds(dir3):
                        possibleMoves.add(''.join(dir3))
                        dir3[0] = inc(dir3[0])
                        dir3[1] = dec(dir3[1])
                while inBounds(dir4):
                        possibleMoves.add(''.join(dir4))
                        dir4[0] = dec(dir4[0])
                        dir4[1] = inc(dir4[1])
                while inBounds(dir5):
                        possibleMoves.add(''.join(dir5))
                        dir5[0] = inc(dir5[0])
                while inBounds(dir6):
                        possibleMoves.add(''.join(dir6))
                        dir6[0] = dec(dir6[0])
                while inBounds(dir7):
                        possibleMoves.add(''.join(dir7))
                        dir7[1] = inc(dir7[1])
                while inBounds(dir8):
                        possibleMoves.add(''.join(dir8))
                        dir8[1] = dec(dir8[1])
                return possibleMoves, 'Q'

    #king
        if move[0] == 'K':
                dir1 = inc(dir1)
                dir2 = dec(dir2)
                dir3[0] = inc(dir3[0]); dir3[1] = dec(dir3[1])
                dir4[0] = dec(dir4[0]); dir4[1] = inc(dir4[1])
                dir5[0] = inc(dir5[0])
                dir6[0] = dec(dir6[0])
                dir7[1] = inc(dir7[1])
                dir8[1] = dec(dir8[1])
                directionList = [dir1, dir2, dir3, dir4, dir5, dir6, dir7, dir8]
                [possibleMoves.add(''.join(a)) for a in filter(inBounds, directionLis
                return possibleMoves, 'K'

        else:
                print('error')

def constructNewBoard(previousMove, move, pieceType, currentBoard):
    newBoard = currentBoard.copy()
    if len(move) == 3: move = move[1:]
    newBoard[previousMove[0]][int(previousMove[1])] = ''
    newBoard[move[0]][int(move[1])] = pieceType
    return newBoard
```

```python
    def castle(currentBoard, turn, style):
        newBoard = currentBoard.copy()
        val = 1 if turn == 'w' else 8
        if style == 'K':
            newBoard['e'][val] = ''
            newBoard['h'][val] = 'K'
            newBoard['f'][val] = 'R'
        else:
            newBoard['a'][val] = ''
            newBoard['d'][val] = 'K'
            newBoard['d'][val] = 'R'
        return newBoard
```

```python
In [433]: '''
    Will not work for entire game. Will always work for first five moves, likely even te
    game don't play like absolute animals.
    Will not work for ambiguous moves (meaning two of the same piece have overlapping po
    for nonpawn pieces, which requires a minimum of 5 moves in a row by a single player.
    '''
    def prevMove(turn, move, currentBoard):
        if 'x' in move and move[0] == (move[0].lower()):
            possibleMove = move[0]+str(int(move[3])-1) if turn == 'w' else move[0]+str(i
            move = move[1:]
            return constructNewBoard(possibleMove, move, turn + 'P', currentBoard)



        elif move == 'O-O':
            return castle(currentBoard, turn, 'K')
        elif move == 'O-O-O':
            return castle(currentBoard, turn, 'Q')

        if 'x' in move:
            move = ''.join([char for char in move if char != 'x'])

        move = ''.join([char for char in move if char != 'x'])

        if len(move) == 4:
            requiredCol = move[1]
            move = move[0] + move[2:]
            prevMoveSet, pieceType = prevMoves(move, turn)
            pieceType = turn + pieceType
            for possibleMove in prevMoveSet:
                if ((currentBoard[possibleMove[0]][int(possibleMove[1])]) == pieceType) a
                    return constructNewBoard(possibleMove, move, pieceType, currentBoard
```

```python
        else:
            prevMoveSet, pieceType = prevMoves(move, turn)
            pieceType = turn + pieceType
            for possibleMove in prevMoveSet:
                if (currentBoard[possibleMove[0]][int(possibleMove[1])]) == pieceType:
                    return constructNewBoard(possibleMove, move, pieceType, currentBoard)



def run(moveList: str, currentBoard):
    moveList = (''.join([char for char in moveList if char != '+'])).split()
    BOARDS = []
    for i in range(len(moveList)):
        turn = 'b' if i%2 else 'w'
        currentBoard = prevMove(turn, moveList[i], currentBoard)
        if i>0 and i%2:
            BOARDS.append(numForm(currentBoard))
    return sum(BOARDS)/len(BOARDS)

# def run(moveList: str, currentBoard):
#     moveList = (''.join([char for char in moveList if char != '+'])).split()
#     BOARDS = []
#     for i in range(len(moveList)):
#         turn = 'b' if i%2 else 'w'
#         print(moveList[i])
#         currentBoard = prevMove(turn, moveList[i], currentBoard)
#         if i>0 and i%2:
#             print(currentBoard, '\n\n\n')
#             BOARDS.append(numForm(currentBoard))
#     return BOARDS
```

In [434]:
```python
BOARDS = run(game3, boardLetter.copy())
abs(BOARDS)
```

Out[434]:

	a	b	c	d	e	f	g	h
8	5000.0	2000.0	2400.0	9000.0	12000.0	3000.0	800.0	5000.0
7	1000.0	1200.0	1000.0	1000.0	200.0	0.0	1000.0	1000.0
6	0.0	400.0	0.0	0.0	800.0	1200.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	1000.0	0.0	0.0
4	0.0	0.0	0.0	1000.0	0.0	0.0	0.0	0.0
3	0.0	0.0	800.0	0.0	800.0	1200.0	0.0	0.0
2	1000.0	1000.0	1000.0	0.0	800.0	1000.0	1000.0	1000.0
1	5000.0	1200.0	3000.0	9000.0	12000.0	2400.0	800.0	5000.0

In [435]:
```python
'd4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3'
```

Out[435]: 'd4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf3 Bb4+ Nc3'

1 PCA BEGIN

In [436]:
```
training = filtered_mdf[:int(len(filtered_mdf)*.8)]
testing = filtered_mdf[int(len(filtered_mdf)*.8):]
filtered_mdf.head()
```

Out[436]:

	moves	opening
3	d4 d5 Nf3 Bf5 Nc3 Nf6 Bf4 Ng4 e3 Nc6	Queen's Pawn Game
5	e4 c5 Nf3 Qa5 a3	Sicilian Defense
8	e4 e5 Bc4 Nc6 Nf3 Nd4 d3 Nxf3+ Qxf3 Nf6	Italian Game
11	e4 e6 d4 d5 e5 c5 c3 Nc6 Nf3 Qb6	French Defense
12	e4 e6 Nf3 d5 exd5 exd5 Qe2+ Be7 Nc3 Nf6	French Defense

In [437]:
```
ops
```

Out[437]:
```
['Sicilian Defense',
 'French Defense',
 'Ruy Lopez',
 'Italian Game',
 'English Opening',
 "Queen's Gambit Declined",
 'Caro-Kann Defense',
 "King's Indian Defense",
 "Queen's Pawn Game",
 'Nimzo-Indian Defense']
```

In [1123]:
```
import time
s = time.perf_counter()
# Mset = {ops[i]: [run(moveseq, boardLetter.copy()) for moveseq in testing[testing.

Mset = {}
for i in range(len(ops)):
    Mset[ops[i]] = []
    for moveseq in training[training.opening == ops[i]].moves:
        try:
            Mset[ops[i]].append(run(moveseq, boardLetter.copy()))
        except:
            print(moveseq)
            Mset[ops[i]].append(run(prev, boardLetter.copy()))
        prev = moveseq


print(f"time: {time.perf_counter()-s:.3f}")
```

```
e4 e6 d4 Qh4 Nc3 Nf6 e5 Ne4 Nf3 Qxf2#
e4 e6 Nc3 Qh4 d3 Bc5 Nf3 Qxf2#
e4 e6 d4 Qh4 Nc3 Nf6 e5 Ne4 Nf3 Qxf2#
e4 e6 d4 Qh4 Nc3 Nf6 e5 Ne4 Nf3 Qxf2#
e4 e5 Nf3 Nc6 Bc4 h6 O-O Bc5 Kh1 d6
```

```
e4 e5 c4 Qh4 d3 Bc5 Nf3 Qxf2#
c4 e5 Nc3 Bc5 Na4 d6 h3 Qf6 b3 Qxf2#
c4 e5 Nc3 Bc5 g3 Qf6 Bg2 Qxf2#
e4 c6 d4 Nf6 e5 Nd5 c4 Nb4 Bd2 N4a6
time: 130.547
```

```
In [1125]: len(Mset[ops[0]])

Out[1125]: 2064

In [440]: pca_res = {opening: PCA(Mset[opening]) for opening in ops}

In [441]: pca_res[ops[1]]

Out[441]: (array([ 5.0000e+03,  1.7840e+03,  2.8680e+03,  8.4780e+03,  1.2000e+04,
                   2.5200e+03,  1.6000e+03,  5.0000e+03,  9.9200e+02,  9.9600e+02,
                   7.6400e+02,  3.8400e+02,  1.8400e+02,  9.9200e+02,  1.0040e+03,
                   9.9600e+02,  0.0000e+00,  2.6800e+02,  2.4400e+02,  7.2000e+01,
                   7.6800e+02,  3.6800e+02,  8.0000e+01,  1.2000e+01,  3.2000e+01,
                  -1.2000e+01,  2.5200e+02,  6.1200e+02, -1.6000e+02,  0.0000e+00,
                  -4.8000e+01,  0.0000e+00, -4.0000e+00,  1.3600e+02, -1.6000e+01,
                  -4.9200e+02, -5.3800e+02, -5.2000e+01,  4.0000e+01,  1.2600e+02,
                  -2.4000e+01,  0.0000e+00, -4.6800e+02, -1.0000e+02,  0.0000e+00,
                  -7.7200e+02, -2.4000e+01, -4.0000e+00, -9.7200e+02, -9.8800e+02,
                  -9.2400e+02, -5.2400e+02, -4.0600e+02, -9.4800e+02, -9.9600e+02,
                  -9.9600e+02, -5.0000e+03, -1.4480e+03, -2.9280e+03, -8.6760e+03,
                  -1.1952e+04, -2.7720e+03, -1.2080e+03, -4.9800e+03]),
           array([[ 0.00000000e+00+0.j        ,  2.83926954e+02+0.j        ,
                    2.55326208e+02-5.20539301j, ..., -3.21726450e+02+5.20539301j,
                   -7.99264574e+02+6.94052402j, -8.59721572e-01+0.j        ],
                  [ 0.00000000e+00+0.j        ,  2.23726720e+02+0.j        ,
                   -2.29701784e+02+3.52811663j, ..., -1.91811723e+02-3.52811663j,
                    2.51591613e+02-4.70415551j, -2.06589459e+02+0.j        ],
                  [ 0.00000000e+00+0.j        , -1.11061770e+02+0.j        ,
                    3.14934812e+01+3.52811663j, ..., -7.90474167e+02-3.52811663j,
                   -2.62046556e+02-4.70415551j, -5.96562685e+01+0.j        ],
                  ...,
                  [ 0.00000000e+00+0.j        , -1.03678594e+02+0.j        ,
                   -5.73173144e+02+3.52811663j, ..., -8.83613414e+01-3.52811663j,
                   -9.32377005e+01-4.70415551j,  1.12419526e+02+0.j        ],
                  [ 0.00000000e+00+0.j        ,  2.97583182e+02+0.j        ,
                   -1.44643803e+02+3.52811663j, ..., -2.48401715e+02-3.52811663j,
                    1.40241266e+03-4.70415551j,  1.23921920e+02+0.j        ],
                  [ 0.00000000e+00+0.j        , -2.72219347e+02+0.j        ,
                    7.79044026e+01+3.52811663j, ..., -9.91893651e+01-3.52811663j,
                   -1.69544755e+02-4.70415551j,  9.61399403e+00+0.j        ]]),
           array([[ -888709.72148051 +4317.00593758j,
                   -3533014.73844715 -2925.98472859j,
```

```
          4132967.9907687  -2925.98472861j, ...,
         -4787293.5740473  -2925.98472861j,
          3191912.52922749 -2925.98472865j,
          3953910.95968038 -2925.98472861j],
        [-2426969.3509418 -28303.45693479j,
         -2339345.40996196+19183.54617874j,
         -3295705.55747886+19183.54617889j, ...,
          6571560.27378419+19183.54617888j,
         -4013683.97921545+19183.54617913j,
          2126249.94667592+19183.54617892j],
        [ 1171014.26534102 -4879.18838282j,
         -9313380.40250069 +3307.02132507j,
          7678818.45644386 +3307.0213251j , ...,
         -1680318.29726842 +3307.0213251j ,
          7856547.0433158  +3307.02132514j,
         11012000.03742841 +3307.02132511j],
        ...,
        [ 3193325.2976692 -39061.26915865j,
           279851.65371332+26474.98722264j,
         -3211395.22127809+26474.98722285j, ...,
          1724200.58776876+26474.98722284j,
         -4298233.2341375 +26474.98722318j,
         -1204929.50418755+26474.9872229j ],
        [-2731674.55922526 -9911.06829399j,
          -509386.01630206 +6717.53407142j,
          -837681.75253488 +6717.53407147j, ...,
          3424229.14155802 +6717.53407147j,
         -1564689.98668101 +6717.53407155j,
           452416.54836911 +6717.53407148j],
        [ -483795.41194799 +5184.5714395j ,
          1028797.66659473 -3514.00416761j,
          1771600.79377597 -3514.00416764j, ...,
           145017.23099777 -3514.00416764j,
         -2093876.88719653 -3514.00416768j,
          -745559.53730469 -3514.00416764j]]))
```

In [442]:
```
filtered_mdf[filtered_mdf.opening == ops[0]]
```

Out[442]:

	moves	opening
5	e4 c5 Nf3 Qa5 a3	Sicilian Defense
22	e4 c5 Bc4 Nf6 Nc3 d6 Nf3 g6 Ng5 e6	Sicilian Defense
24	e4 c5 d4 cxd4 Qxd4 Nc6 Qa4 e5 Be3 Nf6	Sicilian Defense
30	e4 c5 d4 cxd4 Qxd4 Nc6 Qa4 Nf6 Nc3 g6	Sicilian Defense
31	e4 c5 Nf3 d6 Bb5+ Bd7 Bxd7+ Nxd7 O-O Ngf6	Sicilian Defense
...	...	...
20024	e4 c5 Nh3 Nc6 Bc4 Nf6 Ng5 e6 Qf3 Ne5	Sicilian Defense
20026	e4 c5 Nf3 d6 Bc4 e6 d4 cxd4 Nxd4 Nf6	Sicilian Defense
20027	e4 c5 d4 cxd4 c3 dxc3 Bc4 cxb2 Bxb2 e6	Sicilian Defense

```
20030      e4 c5 Nf3 g6 d4 cxd4 Nxd4 Bg7 Be3 Nf6   Sicilian Defense
20045          e4 c5 c3 g6 d4 d6 Nf3 Nf6 e5 dxe5   Sicilian Defense

[2573 rows x 2 columns]
```

In [443]:
```python
y = list(testing.moves)
sample = y[1]
testing
```

Out[443]:
```
                                        moves           opening
16148                         e4 c5 Nf3 Nc6   Sicilian Defense
16151     Nf3 Nc6 d4 d5 e3 e6 c4 Nf6 Nc3 Bb4  Queen's Pawn Game
16152      e4 e6 Nf3 d5 e5 c5 d4 Nc6 Bb5 Qb6     French Defense
16154   e4 e5 Nf3 Nc6 Bc4 Nf6 Ng5 d5 exd5 Na5       Italian Game
16155   e4 e5 Nf3 Nc6 Bc4 d6 d4 exd4 Nxd4 Bd7       Italian Game
...                                       ...                ...
20049   e4 e6 Nf3 d5 Nc3 Bb4 exd5 exd5 d4 Bg4     French Defense
20050      c4 e5 d4 exd4 Qxd4 Nf6 Bg5 Be7 e4    English Opening
20051    e4 e6 Nf3 d5 Bb5+ Bd7 c4 c6 Ba4 Qa5     French Defense
20055     d4 d5 Bf4 Nc6 e3 Nf6 c3 e6 Nf3 Be7  Queen's Pawn Game
20057     d4 d5 Bf4 Na6 e3 e6 c3 Nf6 Nf3 Bd7  Queen's Pawn Game

[1765 rows x 2 columns]
```

In [444]:
```python
def PCA(BOARDS):

    N = 8
    M = len(BOARDS)
    mew = [0 for _ in range(N**2)]
    GAMMA = []
    for board in BOARDS:
        boardvec = np.concatenate(np.array(board))
        GAMMA.append(boardvec)
        mew = np.array([boardvec[i] + mew[i] for i in range(N**2)])
    mean_board = mew/M
    mean_boardB = mean_board.reshape((N,N))
    A = np.array([gamma - mean_board for gamma in GAMMA]).T #array of PHIs
    C = (A.T @ A)
    w1,v1 = np.linalg.eig(C)
    U = np.array([np.array(sum(v1[l][k]*A.T[k] for k in range(1,M))) for l in range(N
    ref = [bd.reshape((N, N)) for bd in U]
    OMEGA = np.array([U @ A.T[i] for i in range(len(A.T))])
    return mean_board, U, OMEGA
```

In [501]:
```python
total_mean = E(np.array([pca_res[opening][0] for opening in ops]))
A_in=np.array([np.concatenate(np.array(run(game, boardLetter.copy())))for game in y[
S = [set() for _ in range(len(ops))]
correct = np.array(list(testing.opening)[::6])
pred = []
```

```
    total_U = [pca_res[opening][1] for opening in ops]
    for i in range(len(A_in)):
        global_e = float('inf')
        ans = ''
        for opening in ops:
            U = pca_res[opening][1]
            omean = pca_res[opening][0]
            omg = pca_res[opening][2]

            omg_in = np.array((A_in[i] - omean))
            e = np.linalg.norm(omg_in)
            if e < global_e:
                ans = opening
                global_e = e
        pred += [ans]

    res = np.array(pred)

    E(res == correct)
```

Out[501]: 0.7661016949152543

```
In [514]: def confusion(res, actual):
              df = pd.DataFrame(data={nm: np.zeros(len(set(res))) for nm in sorted(set(res))})
              for p, a in zip(res, actual):
                  df.loc[p , a] += 1
              return df
```

```
In [524]: confusion(res, correct)
```

Out[524]:

```
                        Caro-Kann Defense  English Opening  French Defense  \
Caro-Kann Defense                    22.0              2.0             1.0
English Opening                       0.0             14.0             0.0
French Defense                        0.0              2.0            39.0
Italian Game                          0.0              0.0             0.0
King's Indian Defense                 0.0              2.0             0.0
Nimzo-Indian Defense                  0.0              0.0             0.0
Queen's Gambit Declined               0.0              3.0             1.0
Queen's Pawn Game                     1.0              0.0             0.0
Ruy Lopez                             0.0              0.0             1.0
Sicilian Defense                      1.0              1.0             1.0

                        Italian Game  King's Indian Defense  \
Caro-Kann Defense                0.0                    0.0
English Opening                  0.0                    0.0
French Defense                   0.0                    0.0
Italian Game                    30.0                    0.0
King's Indian Defense            0.0                    5.0
Nimzo-Indian Defense             0.0                    0.0
```

```
Queen's Gambit Declined          0.0                    1.0
Queen's Pawn Game                0.0                    0.0
Ruy Lopez                        8.0                    0.0
Sicilian Defense                 0.0                    0.0

                         Nimzo-Indian Defense  Queen's Gambit Declined  \
Caro-Kann Defense                         0.0                      0.0
English Opening                           0.0                      0.0
French Defense                            0.0                      1.0
Italian Game                              0.0                      0.0
King's Indian Defense                     1.0                      0.0
Nimzo-Indian Defense                      5.0                      1.0
Queen's Gambit Declined                   2.0                      8.0
Queen's Pawn Game                         0.0                      1.0
Ruy Lopez                                 0.0                      0.0
Sicilian Defense                          0.0                      0.0

                         Queen's Pawn Game  Ruy Lopez  Sicilian Defense
Caro-Kann Defense                      4.0        0.0               7.0
English Opening                        0.0        0.0               2.0
French Defense                         1.0        0.0               7.0
Italian Game                           0.0        2.0               7.0
King's Indian Defense                  1.0        0.0               0.0
Nimzo-Indian Defense                   1.0        0.0               0.0
Queen's Gambit Declined                3.0        0.0               0.0
Queen's Pawn Game                     21.0        0.0               0.0
Ruy Lopez                              0.0       33.0               3.0
Sicilian Defense                       0.0        0.0              49.0
```

In [557]:
```
def metrics(C):
    total_TP = sum(np.array(C)[i][i] for i in range(len(C)))
    L = ['precision', 'recall', 'f1', 'accuracy']
    d = {label: [] for label in L}
    prec = {}
    recall = {}
    f1 = {}
    acc = {}
    for label in C:
        TP = C.loc[label, label]
        TN = total_TP - TP
        FN = sum(C.loc[label])-TP
        FP = sum(C[label])-TP
        d['precision'] += [round(TP/(TP+FP),2)]
        d['recall'] += [round(TP/(TP+FN),2)]
        d['f1'] += [round(2*TP/(2*TP+FP+FN),2)]
        d['accuracy'] += [round((TP+TN)/sum(sum(np.array(C))),2)]
    for lbl in L:
        d[lbl] += [E(d[lbl])]
```

```
    R = pd.DataFrame(data=d).rename(dict(zip(range(11), list(C.columns)+ ["Avg."])))
    return R
```

In [558]:
```
eboards = [pca_res[opening][0] for opening in ops]
```

In [559]:
```
metrics(confusion(res,correct))
```

Out[559]:

	precision	recall	f1	accuracy
Caro-Kann Defense	0.920	0.610	0.730	0.77
English Opening	0.580	0.880	0.700	0.77
French Defense	0.910	0.780	0.840	0.77
Italian Game	0.790	0.770	0.780	0.77
King's Indian Defense	0.830	0.560	0.670	0.77
Nimzo-Indian Defense	0.620	0.710	0.670	0.77
Queen's Gambit Declined	0.730	0.440	0.550	0.77
Queen's Pawn Game	0.680	0.910	0.780	0.77
Ruy Lopez	0.940	0.730	0.820	0.77
Sicilian Defense	0.650	0.940	0.770	0.77
Avg.	0.765	0.733	0.731	0.77

In [496]:
```
E([np.linalg.norm(eboards[i] - eboards[j]) for i in range(len(eboards)) for j in rang
```

Out[496]:
```
3576.761103989388
```

In [497]:
```
eboards
```

Out[497]:
```
[array([ 5.00000000e+03,  1.25600000e+03,  2.89200000e+03,  8.40600000e+03,
         1.20000000e+04,  2.85600000e+03,  1.47733333e+03,  5.00000000e+03,
         8.60000000e+02,  9.84000000e+02,  2.28000000e+02,  6.44000000e+02,
         9.00000000e+02,  1.00000000e+03,  9.76000000e+02,  9.96000000e+02,
         1.36000000e+02,  1.44000000e+02,  6.80000000e+02,  4.00000000e+02,
         1.00000000e+02,  4.69333333e+02,  1.20000000e+02,  4.00000000e+00,
         1.30000000e+02, -2.40000000e+02,  7.76000000e+02,  3.20000000e+01,
         2.93333333e+01,  0.00000000e+00, -3.20000000e+01,  0.00000000e+00,
        -2.36000000e+02,  0.00000000e+00, -3.84000000e+02, -2.72000000e+02,
        -9.28000000e+02,  0.00000000e+00,  3.60000000e+01,  0.00000000e+00,
        -1.20000000e+01, -1.20000000e+01, -4.20000000e+02, -8.00000000e+01,
        -8.40000000e+01, -1.08400000e+03, -3.60000000e+01,  0.00000000e+00,
        -9.80000000e+02, -1.00000000e+03, -9.24000000e+02, -6.40000000e+02,
        -1.32000000e+02, -1.00000000e+03, -1.02400000e+03, -1.00000000e+03,
        -4.98000000e+03, -1.64000000e+03, -2.98800000e+03, -8.24400000e+03,
        -1.17800000e+04, -2.16800000e+03, -8.28000000e+02, -4.90000000e+03]),
 array([ 5.0000e+03,  1.7840e+03,  2.8680e+03,  8.4780e+03,  1.2000e+04,
         2.5200e+03,  1.6000e+03,  5.0000e+03,  9.9200e+02,  9.9600e+02,
         7.6400e+02,  3.8400e+02,  1.8400e+02,  9.9200e+02,  1.0040e+03,
         9.9600e+02,  0.0000e+00,  2.6800e+02,  2.4400e+02,  7.2000e+01,
         7.6800e+02,  3.6800e+02,  8.0000e+01,  1.2000e+01,  3.2000e+01,
        -1.2000e+01,  2.5200e+02,  6.1200e+02, -1.6000e+02,  0.0000e+00,
        -4.8000e+01,  0.0000e+00, -4.0000e+00,  1.3600e+02, -1.6000e+01,
```

```
       -4.9200e+02, -5.3800e+02, -5.2000e+01,  4.0000e+01,  1.2600e+02,
       -2.4000e+01,  0.0000e+00, -4.6800e+02, -1.0000e+02,  0.0000e+00,
       -7.7200e+02, -2.4000e+01, -4.0000e+00, -9.7200e+02, -9.8800e+02,
       -9.2400e+02, -5.2400e+02, -4.0600e+02, -9.4800e+02, -9.9600e+02,
       -9.9600e+02, -5.0000e+03, -1.4480e+03, -2.9280e+03, -8.6760e+03,
       -1.1952e+04, -2.7720e+03, -1.2080e+03, -4.9800e+03]),
array([ 5.0000e+03,  4.0800e+02,  2.9160e+03,  8.7480e+03,  1.1904e+04,
        2.5440e+03,  1.4720e+03,  4.9600e+03,  7.2400e+02,  9.1600e+02,
        9.8000e+02,  8.3600e+02,  1.4800e+02,  9.9200e+02,  9.8400e+02,
        9.7600e+02,  2.7600e+02,  0.0000e+00,  1.4600e+03,  1.5600e+02,
        0.0000e+00,  5.8000e+02,  1.6000e+01,  2.4000e+01,  0.0000e+00,
       -1.1360e+03,  3.7200e+02,  8.0000e+00,  9.3200e+02,  0.0000e+00,
       -2.8000e+01,  1.2000e+01, -2.8800e+02,  3.6000e+01, -3.6000e+01,
        1.1200e+02, -9.3600e+02,  0.0000e+00,  4.8000e+01,  0.0000e+00,
       -4.0000e+00, -8.4000e+01, -1.3600e+02, -3.6000e+01,  0.0000e+00,
       -1.5280e+03,  0.0000e+00, -8.0000e+00, -9.9600e+02, -1.0000e+03,
       -9.7600e+02, -9.1200e+02, -3.6000e+01, -1.0000e+03, -1.0000e+03,
       -9.9200e+02, -4.9600e+03, -1.8880e+03, -2.9880e+03, -8.9640e+03,
       -1.0168e+04, -1.2120e+03, -4.0800e+02, -4.2200e+03]),
array([ 4.98000000e+03,  4.05333333e+02,  2.96400000e+03,  8.56800000e+03,
        1.18080000e+04,  2.25600000e+03,  1.37200000e+03,  4.94000000e+03,
        9.96000000e+02,  9.96000000e+02,  9.96000000e+02,  8.28000000e+02,
        5.12000000e+02,  1.01600000e+03,  1.00000000e+03,  8.32000000e+02,
        4.00000000e+00,  3.60000000e+01,  1.44533333e+03,  9.20000000e+01,
        1.20000000e+01,  6.24000000e+02,  0.00000000e+00,  1.68000000e+02,
        6.13333333e+01,  0.00000000e+00,  5.88000000e+02,  2.80000000e+01,
        9.60000000e+02,  0.00000000e+00, -1.64000000e+02,  0.00000000e+00,
        0.00000000e+00,  3.20000000e+01, -1.73600000e+03,  5.20000000e+01,
       -9.48000000e+02,  0.00000000e+00,  2.40000000e+01,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00, -2.08000000e+02, -1.20000000e+02,
        0.00000000e+00, -1.39866667e+03,  0.00000000e+00, -4.00000000e+00,
       -1.00000000e+03, -9.96000000e+02, -9.04000000e+02, -8.28000000e+02,
       -3.60000000e+01, -1.00000000e+03, -1.00000000e+03, -9.96000000e+02,
       -4.98000000e+03, -1.88800000e+03, -2.98800000e+03, -8.92800000e+03,
       -1.10120000e+04, -1.18000000e+03, -4.37333333e+02, -4.58000000e+03]),
array([ 5.0000e+03,  1.3920e+03,  2.8560e+03,  8.5320e+03,  1.1712e+04,
        2.2080e+03,  1.0640e+03,  4.8800e+03,  9.4800e+02,  1.0360e+03,
        7.3200e+02,  7.4000e+02,  3.7200e+02,  9.7200e+02,  1.1280e+03,
        9.9200e+02,  4.0000e+01,  9.2000e+01,  7.3200e+02,  9.6000e+01,
        2.2400e+02,  8.9200e+02,  1.1600e+02,  8.0000e+00,  1.2000e+01,
        0.0000e+00,  4.8400e+02,  3.2000e+02,  4.1600e+02,  1.5600e+02,
        8.0000e+00,  0.0000e+00,  0.0000e+00,  1.0800e+02, -9.2000e+02,
       -1.4800e+02, -7.2000e+01,  0.0000e+00,  2.0000e+01,  0.0000e+00,
       -4.8000e+01, -5.6000e+01, -8.7600e+02, -1.4400e+02, -1.4000e+02,
       -5.3600e+02, -2.6400e+02, -4.0000e+00, -9.5200e+02, -9.9600e+02,
       -2.6800e+02, -8.2400e+02, -8.2400e+02, -9.8400e+02, -1.1800e+03,
       -9.9600e+02, -5.0000e+03, -1.0640e+03, -2.9040e+03, -8.4600e+03,
       -1.1952e+04, -2.4960e+03, -1.4960e+03, -4.9800e+03]),
```

```
array([ 5.0000e+03,  1.8800e+03,  2.9640e+03,  8.9280e+03,  1.1856e+04,
        2.3520e+03,  1.0640e+03,  4.9400e+03,  9.6800e+02,  9.6400e+02,
        8.8000e+02,  1.0800e+02,  4.3200e+02,  9.8800e+02,  9.9600e+02,
        9.9200e+02,  3.2000e+01,  5.6000e+01,  1.2000e+02,  6.0000e+01,
        7.0800e+02,  9.0400e+02,  4.0000e+00,  8.0000e+00,  0.0000e+00,
        8.0000e+00,  6.8000e+01,  9.1600e+02,  4.0000e+00,  0.0000e+00,
       -2.8800e+02,  0.0000e+00, -3.6000e+01,  2.5200e+02, -5.6400e+02,
       -9.6400e+02,  4.0000e+00, -6.0000e+01,  0.0000e+00, -1.2000e+01,
       -4.4000e+01,  0.0000e+00, -7.3600e+02,  0.0000e+00, -1.6000e+02,
       -4.9600e+02,  0.0000e+00,  0.0000e+00, -9.5600e+02, -9.9200e+02,
       -3.5200e+02, -1.6000e+01, -8.3600e+02, -1.0000e+03, -1.0000e+03,
       -1.0000e+03, -5.0000e+03, -1.1760e+03, -2.5680e+03, -8.8200e+03,
       -1.2000e+04, -2.9520e+03, -1.5040e+03, -5.0000e+03]),
array([ 5.000e+03,  1.816e+03,  2.364e+03,  8.532e+03,  1.200e+04,
        2.964e+03,  1.672e+03,  5.000e+03,  9.720e+02,  9.840e+02,
        4.800e+01,  3.320e+02,  8.440e+02,  1.000e+03,  9.920e+02,
        9.640e+02,  2.000e+01,  8.000e+00,  8.720e+02,  2.400e+01,
        1.920e+02,  2.800e+02,  3.200e+01,  3.600e+01,  7.600e+01,
       -2.400e+01,  2.800e+01,  8.200e+02, -1.640e+02,  3.440e+02,
        1.200e+01,  3.600e+01, -4.000e+00,  1.200e+01, -1.440e+02,
       -4.960e+02, -5.080e+02, -1.040e+02,  1.440e+02,  0.000e+00,
        0.000e+00, -2.400e+01, -3.560e+02, -1.560e+02, -2.000e+01,
       -5.200e+02, -1.600e+01, -1.600e+01, -9.920e+02, -1.000e+03,
       -8.760e+02, -4.480e+02, -1.160e+02, -9.440e+02, -1.000e+03,
       -9.840e+02, -5.000e+03, -1.536e+03, -2.856e+03, -8.928e+03,
       -1.200e+04, -2.616e+03, -1.440e+03, -5.000e+03]),
array([ 4.980e+03,  1.928e+03,  2.976e+03,  9.000e+03,  9.332e+03,
        1.296e+03,  1.280e+02,  3.880e+03,  1.000e+03,  1.000e+03,
        9.840e+02,  6.600e+02,  9.960e+02,  1.000e+03,  1.884e+03,
        9.840e+02,  0.000e+00,  1.200e+01,  6.000e+01,  3.640e+02,
        0.000e+00,  1.876e+03,  8.080e+02,  1.600e+01,  0.000e+00,
        0.000e+00,  4.000e+00,  0.000e+00,  4.000e+00,  0.000e+00,
       -1.440e+02,  0.000e+00,  0.000e+00,  0.000e+00, -7.920e+02,
       -9.200e+02, -2.600e+02, -8.800e+01,  1.200e+01, -4.000e+00,
        0.000e+00,  0.000e+00, -1.056e+03, -4.800e+01, -3.600e+01,
       -5.240e+02, -3.200e+01, -8.000e+00, -1.000e+03, -1.000e+03,
       -2.080e+02, -1.520e+02, -7.600e+02, -9.800e+02, -1.016e+03,
       -9.880e+02, -5.000e+03, -9.440e+02, -2.772e+03, -8.928e+03,
       -1.200e+04, -2.856e+03, -1.472e+03, -5.000e+03]),
array([ 5.0000e+03,  1.3760e+03,  2.3280e+03,  8.8920e+03,  1.1952e+04,
        2.6400e+03,  9.7600e+02,  4.9800e+03,  9.8000e+02,  1.0000e+03,
        8.0000e+02,  6.8000e+01,  7.3200e+02,  9.7600e+02,  1.0240e+03,
        9.6400e+02,  1.2000e+01,  3.6000e+01,  6.3600e+02,  1.6800e+02,
        3.0800e+02,  9.7200e+02,  4.0000e+01,  3.6000e+01,  2.0000e+01,
       -1.6000e+01,  7.2000e+01,  9.6400e+02, -2.0000e+01,  4.2000e+02,
       -8.8000e+01,  5.2000e+01, -4.0000e+00,  9.2000e+01, -1.2000e+01,
       -9.3600e+02,  2.4000e+01, -1.3480e+03,  1.4400e+02, -8.0000e+00,
       -2.0000e+01,  0.0000e+00, -4.1200e+02, -6.0000e+01, -2.6400e+02,
```

```
       -1.0480e+03, -1.6000e+01, -4.4000e+01, -9.7600e+02, -9.8800e+02,
       -8.8800e+02, -1.6800e+02, -7.0800e+02, -9.8400e+02, -9.9200e+02,
       -9.4400e+02, -5.0000e+03, -1.6080e+03, -1.3200e+03, -8.9640e+03,
       -1.1952e+04, -2.9400e+03, -8.9600e+02, -4.9480e+03]),
array([  5000.,   1936.,   2976.,   9000.,  10416.,   1404.,    184.,
         4340.,   1000.,   1000.,    920.,    676.,    272.,   1000.,
         1000.,    984.,      0.,     24.,     48.,     16.,    800.,
         1784.,      0.,     16.,     12.,      0.,     52.,    320.,
            0.,      0.,   -144.,      0.,   -108.,   1308.,   -788.,
         -968.,     20.,    -24.,      0.,    -12.,    -56.,    -36.,
         -876.,    -96.,    -92.,   -400.,      0.,      0.,   -944.,
         -984.,   -860.,   -216.,   -900.,  -1000.,  -1000.,  -1000.,
        -5000.,   -872.,  -2604.,  -8136., -12000.,  -2976.,  -1584.,
        -5000.])]
```

2 Begin neural network

In [1056]:
```python
def sigmoid(x, p=0):
    if p:
        return sigmoid(x)*(1-sigmoid(x))
    return 1/(1+np.exp(-x))

def ReLU(x, p=0):
    if p:
        return (x>0) * 1
    return (x>0)*x

def tanh(x, p=0):
    if p:
        return 4*sigmoid(2*x, p=1)
    return 2*sigmoid(2*x)-1

def softmax(X):


    exps = np.exp(X-np.max(X))

    res = exps / np.sum(exps)
    i = 0
    ans = float("-inf")
    for j,r in enumerate(res):
        if r > ans:
            i = j
            ans = r
```

```
    return i

import random

class NeuralNetwork():
    def __init__(self, neurons, f=sigmoid):

        self.biases = [np.random.randn(d, 1) for d in neurons[1:]]
        self.weights = [np.random.randn(d1, d2) for d2,d1 in zip(neurons[:-1], neuro

        self.layers = len(neurons)
        self.neurons = neurons

        # random initialization
        self.f = f

    def SGD(self, training, epochs, batchSize, eta, testing=None):
        if testing:
            M = len(testing)
        EPS = []
        n = len(training)
        for i in range(epochs):
            random.shuffle(training)
            mini_batches = [training[j:j+batchSize] for j in range(0,n, batchSize)]

            for batchj in mini_batches:
                gradB = [np.zeros(b.shape) for b in self.biases]
                gradW = [np.zeros(w.shape) for w in self.weights]

                for x,y in batchj:
                    dgradB, dgradW = self.back_prop(x,y)
                    gradB = [gB + dgB for gB, dgB in zip(gradB, dgradB)]
                    gradW = [gW + dgW for gW, dgW in zip(gradW, dgradW)]

                self.biases = [B - (eta/len(batchj))*gB for B, gB in zip(self.biases
                self.weights = [w - (eta/len(batchj))*gw for w, gw in zip(self.weig
            EPS.append(self.test(testing)/M)
        return np.array(EPS)


    def back_prop(self,x,y):
        gradB = [np.zeros(b.shape) for b in self.biases]
        gradW = [np.zeros(w.shape) for w in self.weights]

        # FF
        a, A, F= x, [x], []

        for b, w in zip(self.biases, self.weights):
```

```
                r = (w @ a)+b
                F.append(r)
                a = self.f(r)
                A.append(a)

            # BP
            D = (A[-1] - y) * self.f(F[-1], p=1)
            gradB[-1] = D
            gradW[-1] = (D @ A[-2].T)


            # derivative activation func.
            sp = self.f(F[-2], p=1)

            D = (self.weights[-1].T @ D) * sp
            gradB[-2] = D

            gradW[-2] = (D @ A[-3].T)

            return gradB, gradW

        def feed_forward(self, x):
            for b, w in zip(self.biases, self.weights):
                x = self.f((w@x)+b)
            return x

        def test(self, testing):
    #          for(x,y) in testing:
    #              print(x)
    #              print("-"*30)
    #              print(y)
    #              print("*"*30)
    #              break
            res = [(softmax(self.feed_forward(x)), y) for (x, y) in testing]
            return sum(int(x == y) for (x, y) in res)

        def classify(self, game: str):
            B=np.concatenate(np.array(run(game, boardLetter.copy()))/10000).reshape(64,
            res = softmax(self.feed_forward(B))
            return sorted(ops)[res]
```

```
In [1129]: DAT = []

           for opening in Mset.keys():
               for data in Mset[opening]:
                   VEC = np.concatenate(np.array(data, dtype=np.float64)/10000)
                   DAT.append((VEC.reshape(len(VEC), 1), opening))
```

```
random.shuffle(DAT)
```

In [1131]:
```
train, test = DAT[:len(DAT)//2], DAT[len(DAT)//2:]
train = [(tr, vectorize_result(res)) for tr, res in train]
test = [(t, numerize_result(res)) for t, res in test]
```

In [922]:
```
train, test = DAT[:len(DAT)//2], DAT[len(DAT)//2:]
train = [(tr, vectorize_result(res)) for tr, res in train]
test = [(t, numerize_result(res)) for t, res in test]



layers = [64,44,10]
sims = 18
results = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 1.0, testing =
results2 = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 2.0, testing
results3 = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 3.0, testing
results4 = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 0.1, testing
results5 = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 5.0, testing
results6 = np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, 1.7, testing

print(f'Time taken: {time.perf_counter()-s:.2f}s')
```

```
Time taken: 710.88s
```

In [1132]:
```
len(train)
```

Out[1132]:
```
3529
```

In [908]:
```
def vectorize_result(opening):
    r = np.zeros((10,1))
    r[sorted(ops).index(opening)]=1.0
    return r

def numerize_result(opening):
    return sorted(ops).index(opening)
```

In [909]:
```
E(results)
```

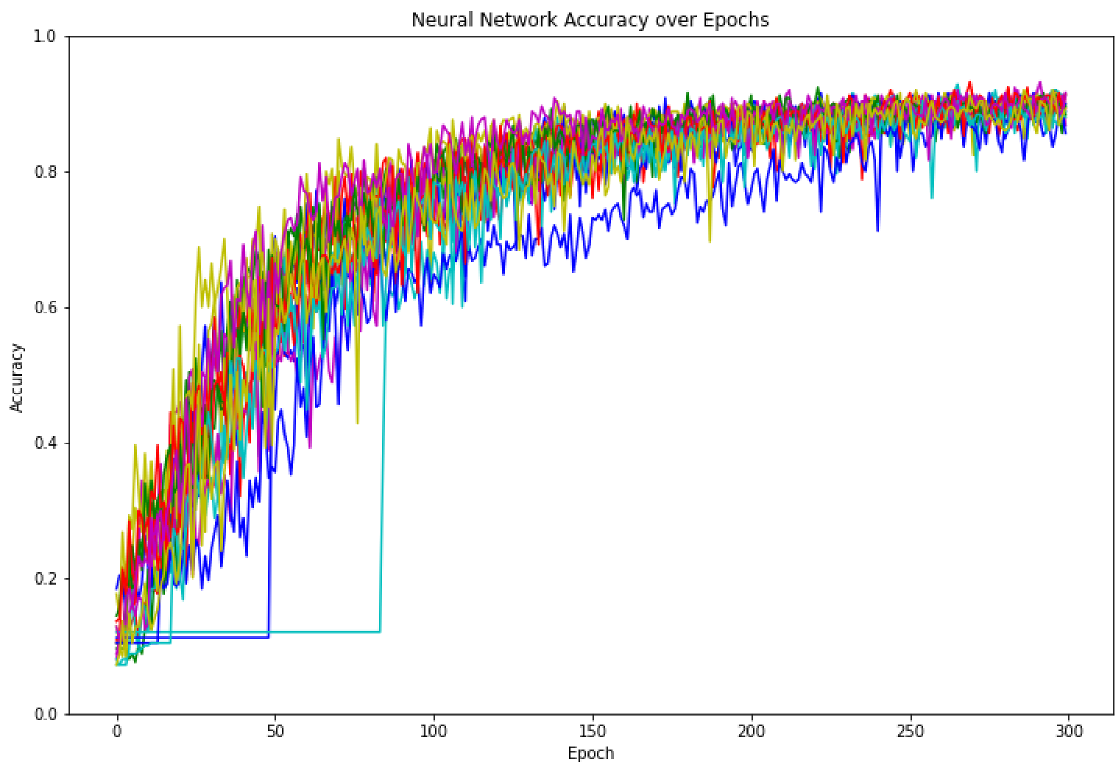
Out[909]:
```
array([0.12064, 0.13088, 0.14448, 0.15472, 0.16848, 0.17344, 0.17952,
       0.2008 , 0.20864, 0.21808, 0.22848, 0.23456, 0.2504 , 0.24944,
       0.27664, 0.28704, 0.28976, 0.31088, 0.32384, 0.3336 , 0.32   ,
       0.34256, 0.35584, 0.36304, 0.37696, 0.39552, 0.39824, 0.4072 ,
       0.38816, 0.412  , 0.41952, 0.42128, 0.43184, 0.44992, 0.46032,
       0.448  , 0.47616, 0.49248, 0.47616, 0.4984 , 0.50768, 0.49504,
       0.51152, 0.5176 , 0.53712, 0.53376, 0.52064, 0.53984, 0.54288,
       0.55504, 0.54768, 0.55328, 0.53744, 0.5648 , 0.57936, 0.56032,
       0.58208, 0.58208, 0.58672, 0.58896, 0.60352, 0.60272, 0.58048,
```

```
       0.60064, 0.5904 , 0.6112 , 0.59632, 0.6128 , 0.61216, 0.62496,
       0.61456, 0.61552, 0.63584, 0.63024, 0.64192, 0.64176, 0.66336,
       0.63968, 0.64656, 0.64576, 0.65008, 0.66368, 0.65856, 0.65424,
       0.64736, 0.65536, 0.66256, 0.66224, 0.6704 , 0.66688, 0.66848,
       0.65696, 0.65904, 0.68128, 0.67232, 0.67744, 0.68976, 0.67248,
       0.67584, 0.70224, 0.71184, 0.70128, 0.7144 , 0.72832, 0.7176 ,
       0.70848, 0.71872, 0.71712, 0.7144 , 0.7064 , 0.73216, 0.71408,
       0.73136, 0.72688, 0.72416, 0.73328, 0.72416, 0.73664, 0.73088,
       0.73936, 0.74   , 0.74032, 0.75104, 0.7328 , 0.74992, 0.74448,
       0.74848, 0.748  , 0.75072, 0.75776, 0.74576, 0.7424 , 0.75296,
       0.74768, 0.74336, 0.76048, 0.75936, 0.74576, 0.7528 , 0.75216,
       0.76304, 0.76592, 0.74784, 0.76448, 0.75024, 0.76208, 0.77216,
       0.76576, 0.75984, 0.76384, 0.76928, 0.76048, 0.76032, 0.77136,
       0.77232, 0.77232, 0.76192, 0.77472, 0.7736 , 0.77728, 0.77344,
       0.77776, 0.7808 , 0.77328, 0.77472, 0.77824, 0.77744, 0.77856,
       0.77408, 0.78464, 0.77952, 0.78192, 0.77488, 0.7848 , 0.78576,
       0.78256, 0.7912 , 0.7904 , 0.78048, 0.78128, 0.79072, 0.78448,
       0.792  , 0.77968, 0.78768, 0.79104, 0.7888 , 0.78352, 0.78768,
       0.7872 , 0.78704, 0.79728, 0.79488, 0.79488, 0.79632, 0.788  ,
       0.79792, 0.79968, 0.7984 , 0.78832])
```

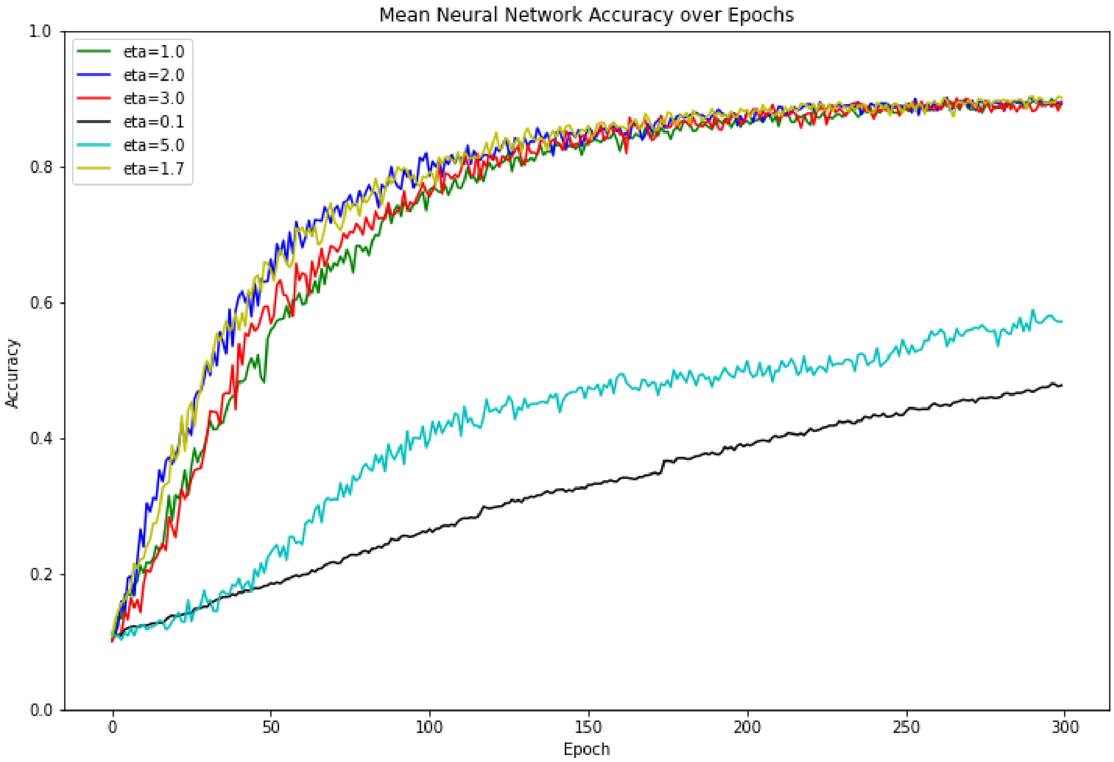
In [1058]:
```
softmax(np.array([1.5, 0, 1]))
```

Out[1058]:
```
0
```

In [1078]:
```
fig_dims=(12,8)
fig, ax = plt.subplots(figsize=fig_dims)
plt.ylim((0,1))
plt.xlabel('Epoch')
i=0
c = 'bgrcmy'
for res in results:
    sns.lineplot(data=pd.Series(res), color = c[i%len(c)])
    i+=1
plt.ylabel("Accuracy")
plt.title("Neural Network Accuracy over Epochs")
plt.show()
```



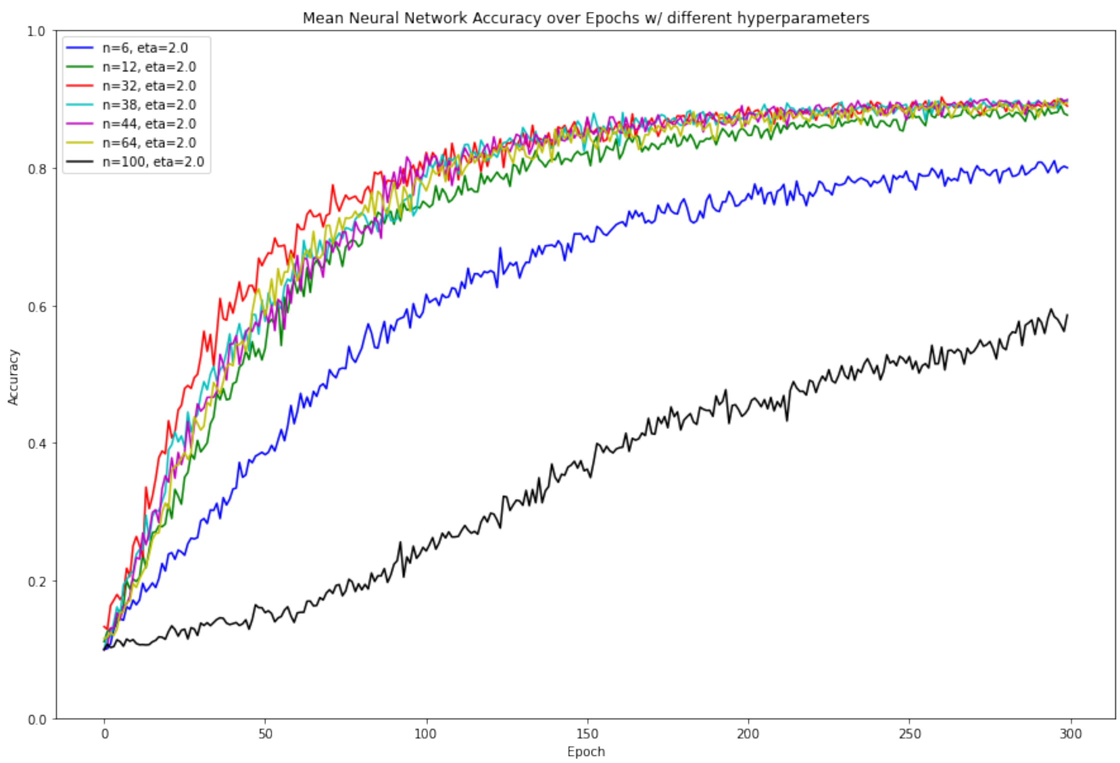

```
In [1079]: fig_dims=(12,8)
           fig, ax = plt.subplots(figsize=fig_dims)
           plt.ylim((0,1))
           plt.xlabel('Epoch')
           sns.lineplot(data=pd.Series(E(results)), color = 'g')
           sns.lineplot(data=pd.Series(E(results2)), color = 'b')
           sns.lineplot(data=pd.Series(E(results3)), color = 'r')
           sns.lineplot(data=pd.Series(E(results4)), color = 'k')
           sns.lineplot(data=pd.Series(E(results5)), color = 'c')
           sns.lineplot(data=pd.Series(E(results6)), color = 'y')
           plt.legend(['eta=1.0', 'eta=2.0', 'eta=3.0', 'eta=0.1', 'eta=5.0', 'eta=1.7'])
           plt.ylabel("Accuracy")
           plt.title("Mean Neural Network Accuracy over Epochs")
           plt.show()
```




```
In [931]: ETA = [2.0]
          NEURONS = [6,12,32,38,44,64,100]
          sims = 15

          fig_dims=(15,10)
          fig, ax = plt.subplots(figsize=fig_dims)
          plt.ylim((0,1))
          plt.xlabel('Epoch')
          plt.ylabel("Accuracy")
          plt.title("Mean Neural Network Accuracy over Epochs w/ different hyperparameters")
          labels = []
          j = 0

          import time

          s = time.perf_counter()
          c = 'bgrcmyk'
          for neur in NEURONS:
              layers = [64, neur, 10]
              i = 0
              for eta in ETA:
                  res = E(np.array([NeuralNetwork(layers, sigmoid).SGD(train, 300, 10, eta, tes
                  sns.lineplot(data=pd.Series(res), color=c[j%len(c)])
```

```
#          if j > 2:
#              axe.lines[0].set_linestyle("--")
            labels.append(f"n={neur}, eta={eta}")
            print(f"n={neur}, eta={eta}"+" done")
            i+=1
        j+=1
    print(f'Time taken: {int((time.perf_counter()-s)//60)}m {(time.perf_counter()-s)%60:
    plt.legend(labels)
    plt.show()
```

```
n=6, eta=2.0 done
n=12, eta=2.0 done
n=32, eta=2.0 done
n=38, eta=2.0 done
n=44, eta=2.0 done
n=64, eta=2.0 done
n=100, eta=2.0 done
Time taken: 11m 28.02s
```




```
In [1138]: s = time.perf_counter()
           net = NeuralNetwork([64,44,10], sigmoid)
           r=net.SGD(train, 300, 10, 2.0, testing = test)
           print(f'Time taken: {int((time.perf_counter()-s)//60)}m {(time.perf_counter()-s)%60
```

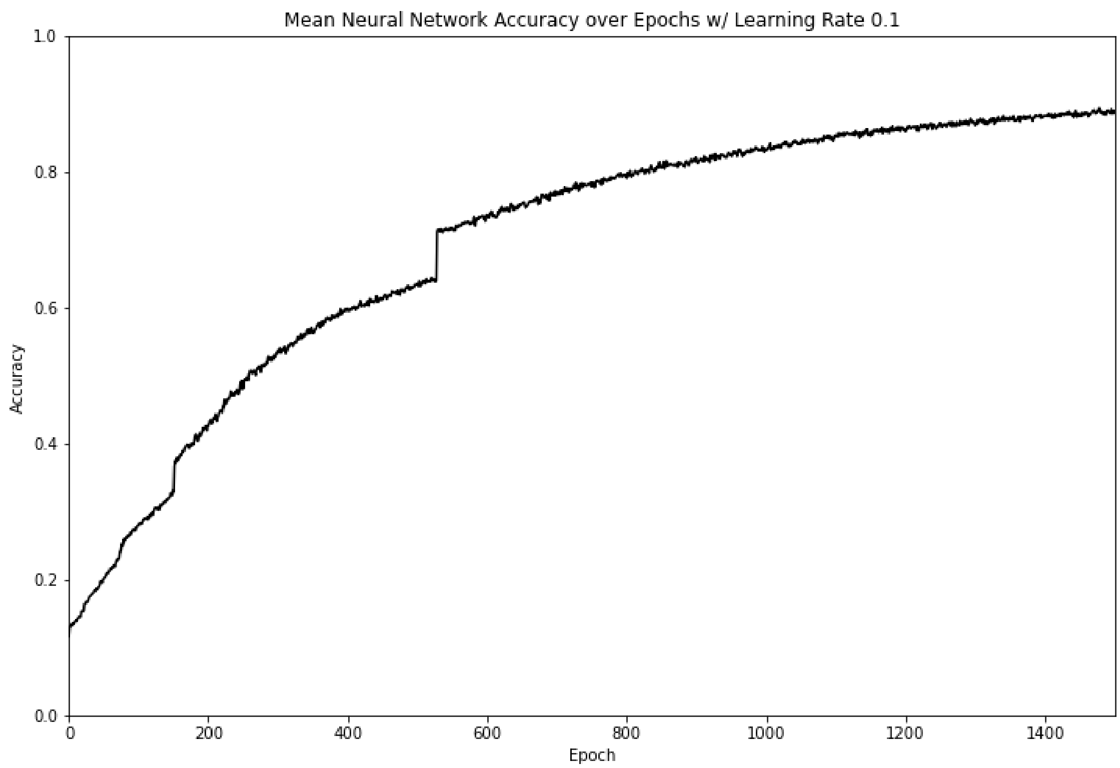
```
Time taken: 2m 9.04s
```

```
In [1139]: r
```

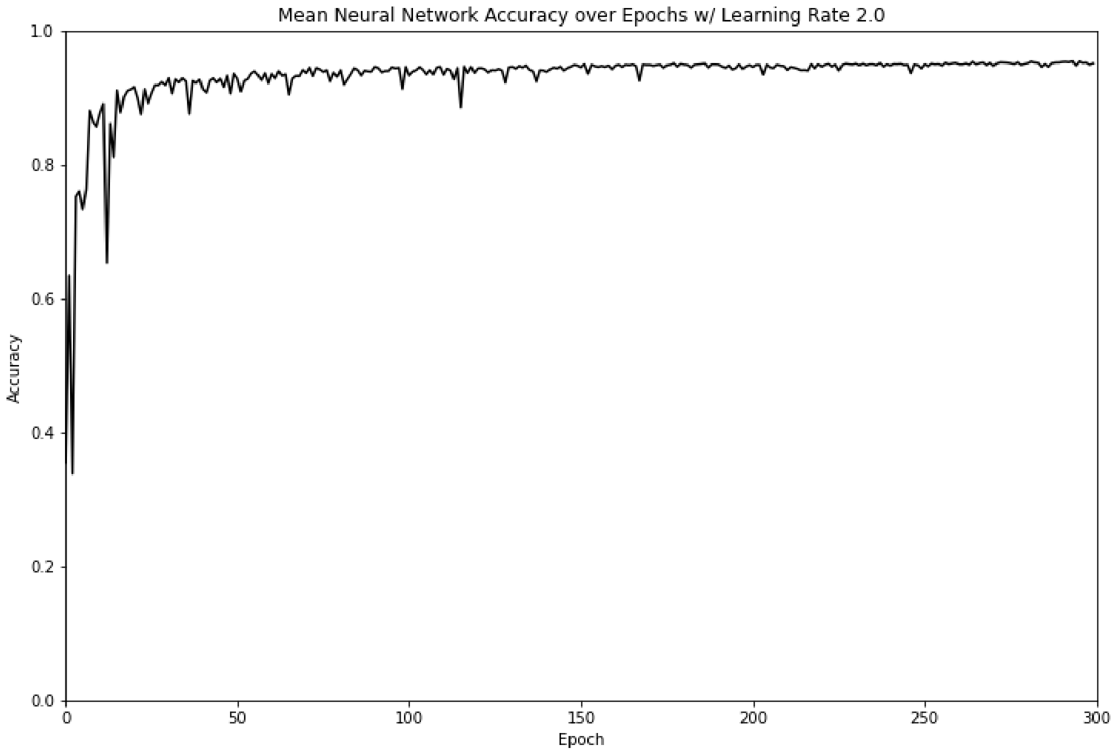
```
Out[1139]: array([0.35477472, 0.63417399, 0.33833947, 0.75233777, 0.76027203,
              0.73335222, 0.7631057 , 0.88070275, 0.86398413, 0.85689997,
              0.87673562, 0.89090394, 0.6534429 , 0.86143383, 0.81099462,
              0.91102295, 0.87786908, 0.9013885 , 0.91045622, 0.91243978,
              0.91612355, 0.89940493, 0.87531879, 0.91272315, 0.89175404,
              0.90677246, 0.91867385, 0.91839048, 0.92349107, 0.91867385,
              0.92944177, 0.90620572, 0.92774157, 0.92292434, 0.9291584 ,
              0.92462454, 0.87588552, 0.92547464, 0.92207424, 0.92717484,
              0.91243978, 0.90762256, 0.92575801, 0.9291584 , 0.92292434,
              0.9283083 , 0.91584018, 0.93312553, 0.90620572, 0.9359592 ,
              0.92859167, 0.90932275, 0.92547464, 0.92802494, 0.93624256,
              0.93964296, 0.93369226, 0.92632474, 0.93680929, 0.92150751,
              0.93567583, 0.92887504, 0.93964296, 0.93284216, 0.9351091 ,
              0.90450553, 0.9283083 , 0.9325588 , 0.93227543, 0.94162652,
              0.93680929, 0.94502692, 0.93227543, 0.94446019, 0.94219326,
              0.93850949, 0.94134316, 0.92434117, 0.93765939, 0.93142533,
              0.94134316, 0.91952394, 0.92717484, 0.9351091 , 0.94417682,
              0.94105979, 0.93312553, 0.94049306, 0.93935959, 0.93879286,
              0.94616039, 0.94417682, 0.93794276, 0.93992632, 0.93992632,
              0.94559365, 0.94389345, 0.94474355, 0.91300652, 0.94616039,
              0.93312553, 0.93879286, 0.94134316, 0.94587702, 0.94162652,
              0.93482573, 0.94134316, 0.93482573, 0.94531029, 0.94616039,
              0.934259  , 0.94332672, 0.94105979, 0.9274582 , 0.94446019,
              0.88551998, 0.94672712, 0.93680929, 0.94616039, 0.93709266,
              0.94389345, 0.94417682, 0.94247662, 0.93765939, 0.94134316,
              0.94077642, 0.94275999, 0.94077642, 0.92235761, 0.94531029,
              0.94616039, 0.94332672, 0.94729385, 0.94502692, 0.94814395,
              0.94190989, 0.93935959, 0.92405781, 0.94105979, 0.94077642,
              0.93850949, 0.94247662, 0.94502692, 0.94361009, 0.94644375,
              0.94049306, 0.94389345, 0.94644375, 0.94899405, 0.94786058,
              0.94531029, 0.95041088, 0.9359592 , 0.94927742, 0.94701048,
              0.94587702, 0.94729385, 0.94587702, 0.94786058, 0.94247662,
              0.94786058, 0.94927742, 0.94587702, 0.94927742, 0.94842732,
              0.94984415, 0.94729385, 0.92519127, 0.94956078, 0.94899405,
              0.94899405, 0.94757722, 0.94871068, 0.94956078, 0.94474355,
              0.94842732, 0.94927742, 0.95154435, 0.94644375, 0.95069425,
              0.94956078, 0.94871068, 0.94502692, 0.94927742, 0.94984415,
              0.95041088, 0.95126098, 0.94502692, 0.94984415, 0.94984415,
              0.94956078, 0.94672712, 0.94446019, 0.94814395, 0.94219326,
              0.94389345, 0.94984415, 0.94304336, 0.94672712, 0.94871068,
              0.94587702, 0.94899405, 0.94899405, 0.93454236, 0.94956078,
              0.94531029, 0.94417682, 0.94927742, 0.94786058, 0.94701048,
              0.94134316, 0.94587702, 0.94474355, 0.94332672, 0.94105979,
```

```
                0.94134316, 0.94049306, 0.95041088, 0.94389345, 0.95012751,
                0.94644375, 0.94927742, 0.95069425, 0.94644375, 0.94956078,
                0.94049306, 0.94899405, 0.95097761, 0.95012751, 0.94956078,
                0.95097761, 0.94842732, 0.95097761, 0.94842732, 0.95012751,
                0.95012751, 0.94842732, 0.95182771, 0.94644375, 0.95012751,
                0.94786058, 0.95041088, 0.95012751, 0.95069425, 0.94871068,
                0.94927742, 0.93652593, 0.94984415, 0.94842732, 0.94361009,
                0.95012751, 0.94701048, 0.95097761, 0.94984415, 0.95041088,
                0.94786058, 0.95239445, 0.95012751, 0.95126098, 0.95211108,
                0.94984415, 0.94984415, 0.95126098, 0.94927742, 0.95381128,
                0.94984415, 0.95182771, 0.95267781, 0.94871068, 0.95211108,
                0.94757722, 0.95126098, 0.95296118, 0.95267781, 0.95239445,
                0.95154435, 0.95012751, 0.95296118, 0.94899405, 0.95041088,
                0.95154435, 0.95409464, 0.95267781, 0.95211108, 0.94587702,
                0.95041088, 0.94616039, 0.95154435, 0.95267781, 0.95267781,
                0.95352791, 0.95381128, 0.95324455, 0.95466138, 0.94786058,
                0.95437801, 0.95211108, 0.95267781, 0.94927742, 0.95126098])
```

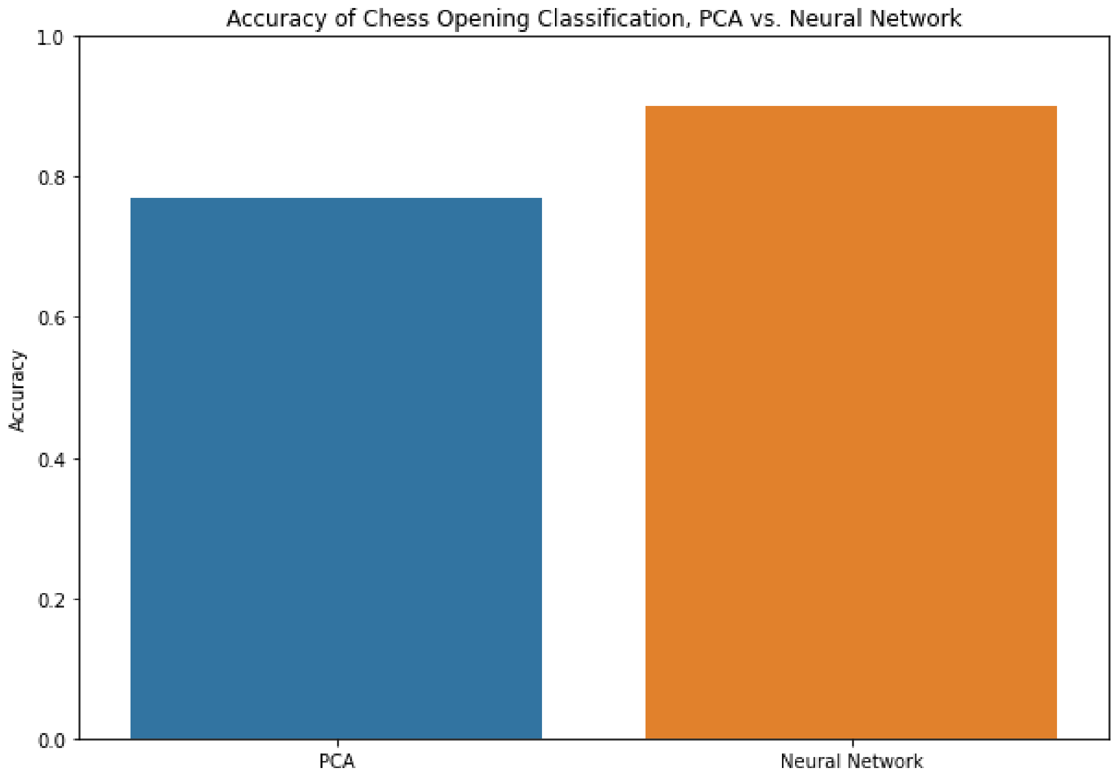
```
In [1060]: net.classify("e4 c5 d4 cxd4 Qxd4 Nc6 Qa4 e5 Be3 Nf6")

Out[1060]: 'Sicilian Defense'

In [1061]: net.classify("Nf3 Nc6 d4 d5 e3 e6 c4 Nf6 Nc3 Bb4")

Out[1061]: "Queen's Pawn Game"

In [1009]: sims = 8
           R = E(np.array([Network([64,44,10], sigmoid).SGD(train, 1500, 10, 0.1, testing = tes

In [1063]: fig_dims=(12,8)
           fig, ax = plt.subplots(figsize=fig_dims)
           plt.ylim((0,1))
           plt.xlim((0,1500))
           plt.xlabel('Epoch')
           plt.ylabel("Accuracy")
           plt.title("Mean Neural Network Accuracy over Epochs w/ Learning Rate 0.1")
           sns.lineplot(data=pd.Series(R), color="k")
           plt.show()
```




```
In [1141]: fig_dims=(12,8)
           fig, ax = plt.subplots(figsize=fig_dims)
           plt.ylim((0,1))
           plt.xlim((0,300))
           plt.xlabel('Epoch')
           plt.ylabel("Accuracy")
           plt.title("Mean Neural Network Accuracy over Epochs w/ Learning Rate 2.0")
           sns.lineplot(data=pd.Series(r), color="k")
           plt.show()
```



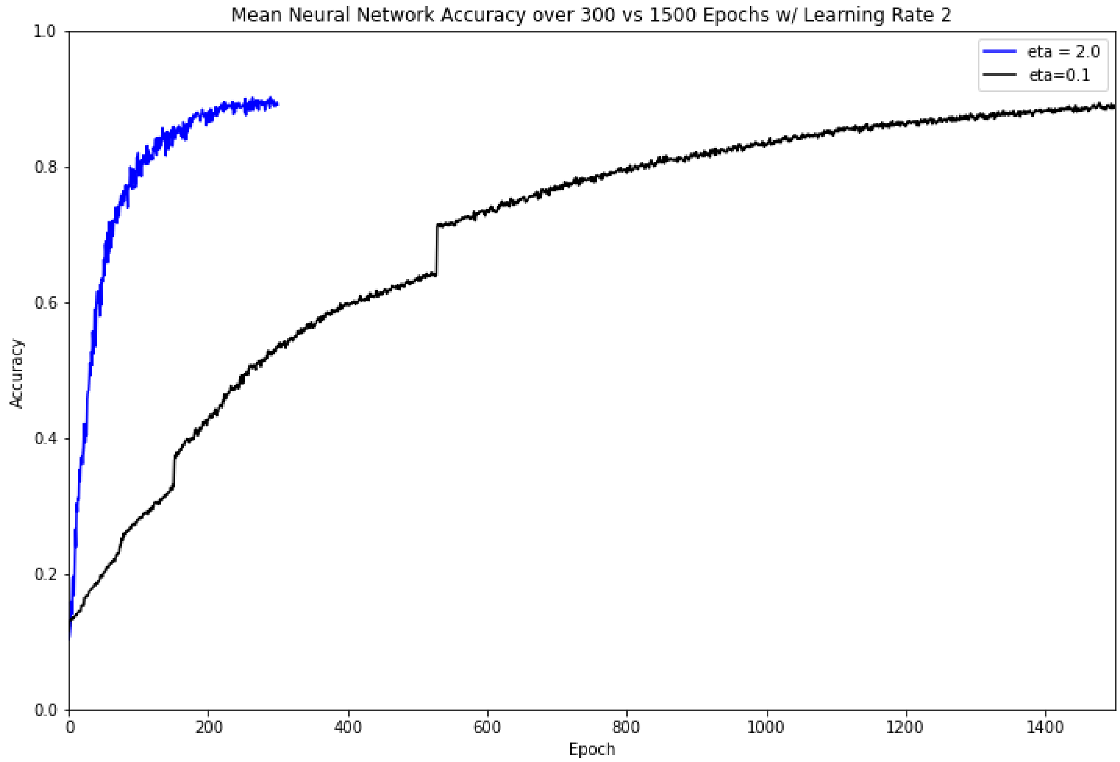

```
In [1008]: R

Out[1008]: array([0.879 , 0.8785, 0.8785, ..., 0.8775, 0.8775, 0.877 ])

In [1062]: len(DAT)

Out[1062]: 500

In [1076]: d={"PCA":[.77], "Neural Network": [.90]}

           fig_dims=(10,7)
           fig, ax = plt.subplots(figsize=fig_dims)
           plt.ylim((0,1))
           sns.barplot(data=pd.DataFrame(data=d))
           plt.ylabel("Accuracy")
           plt.title("Accuracy of Chess Opening Classification, PCA vs. Neural Network")
           plt.show()
```



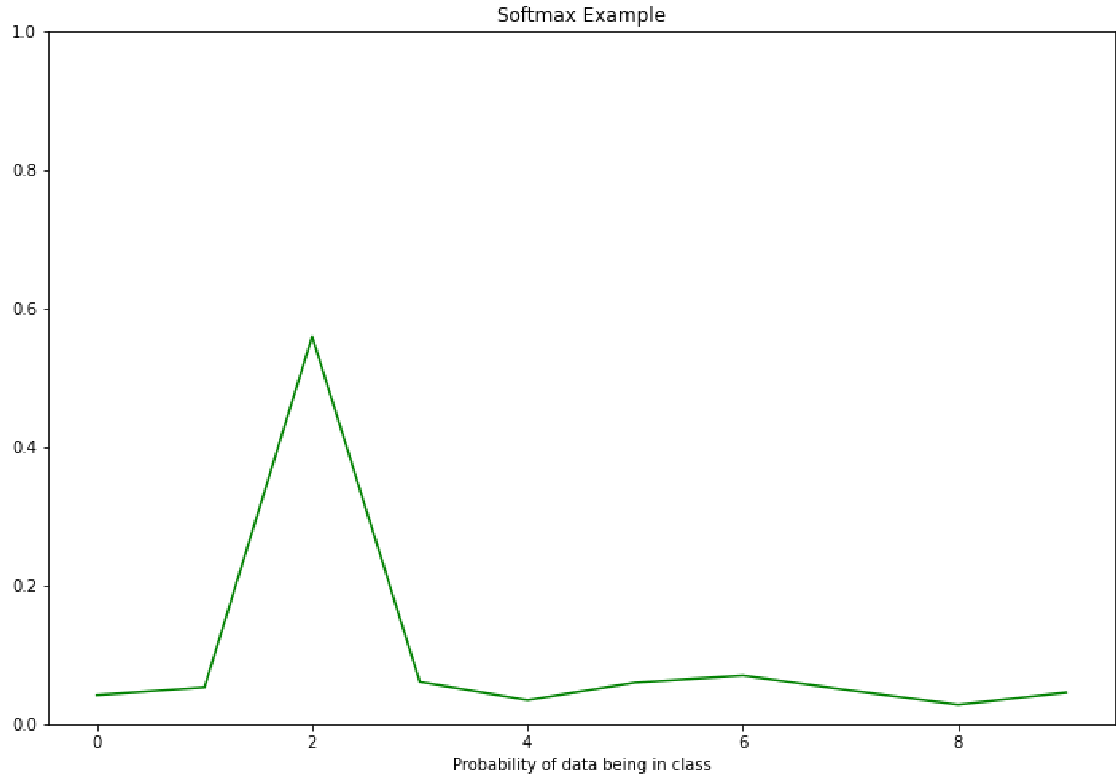
In [1066]:
```
pd.Series(data=d)
```

Out[1066]:
```
PCA               0.77
Neural Network    0.90
dtype: float64
```

In [1122]:
```
fig_dims=(12,8)
fig, ax = plt.subplots(figsize=fig_dims)
plt.ylim((0,1))
plt.xlim((0,1500))
plt.xlabel('Epoch')
plt.ylabel("Accuracy")
plt.title("Mean Neural Network Accuracy over 300 vs 1500 Epochs w/ Learning Rate 2")
sns.lineplot(data=pd.Series(E(results2)), color="b")
sns.lineplot(data=pd.Series(R), color="k")
plt.legend(["eta = 2.0", "eta=0.1"])
plt.show()
```




```
In [1090]: net.classify('e4 e6 Nf3 d5 e5 c5 d4 Nc6 Bb5 Qb6')

Out[1090]: 'French Defense'

In [1091]: def softmaxpt(X):
               exps = np.exp(X-np.max(X))
               return exps / np.sum(exps)

In [1098]: a = abs(np.random.randn(10,1))
           a[2] = 3

In [1105]: S = softmaxpt(a).reshape((10,))

In [1115]: fig_dims=(12,8)
           fig, ax = plt.subplots(figsize=fig_dims)
           plt.ylim((0,1))
           sns.lineplot(data = pd.Series(data=S), color="g")

           plt.xlabel("Probability of data being in class")
           plt.title("Softmax Example")
           plt.show()
```




```
In [1118]: [[i for i in range(1, j)] for j in range(2,7)]

Out[1118]: [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5]]

In [1119]: [list(range(1,j)) for j in range(2,7)]

Out[1119]: [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5]]

In [1120]: net.classify('e4 e6 Nf3 d5 e5 c5 d4 Nc6 Bb5 Qb6')

Out[1120]: 'French Defense'

In [1121]: net.classify('e4 e5 Nf3 Nc6 Bc4 Nf6 Ng5 d5 exd5 Na5         ')

Out[1121]: 'Italian Game'

In [ ]:
```