In [51]:
```
!pip install time
!pip install tqdm
```

```
ERROR: Could not find a version that satisfies the requirement time
(from versions: none)
ERROR: No matching distribution found for time
Collecting tqdm
  Downloading tqdm-4.64.0-py2.py3-none-any.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.4/78.4 kB 795.0 kB/
s eta 0:00:0031m1.5 MB/s eta 0:00:01m
Installing collected packages: tqdm
Successfully installed tqdm-4.64.0
```

In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
2022-07-13 08:09:56.547867: W tensorflow/stream_executor/platform/d
efault/dso_loader.cc:64] Could not load dynamic library 'libcudart.
so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object fil
e: No such file or directory
2022-07-13 08:09:56.547890: I tensorflow/stream_executor/cuda/cudar
t_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU
set up on your machine.

TensorFlow version: 2.9.1
```

In [54]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from tqdm import tqdm
```

In [392]:
```
data = pd.read_csv('data/handtyped_nn/train.csv')
label = data['label']
```

In [393]:
```
data /= 254
data['label'] = label
```

In [394]:
```
data.head()
```

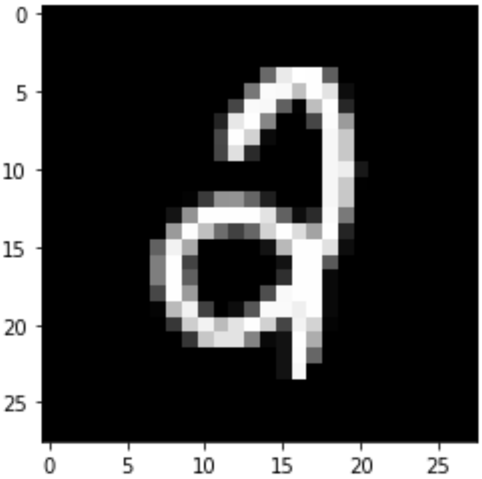
Out[394]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixe
0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
2	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
3	4	0.0	0.0	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	0.0	0.0	...	0.0	

5 rows × 785 columns

In [395]:
```
plt.imshow(data.values[10000][1:].reshape(28, 28), cmap='gray')
```

Out[395]:
```
<matplotlib.image.AxesImage at 0x7fb71f7987c0>
```

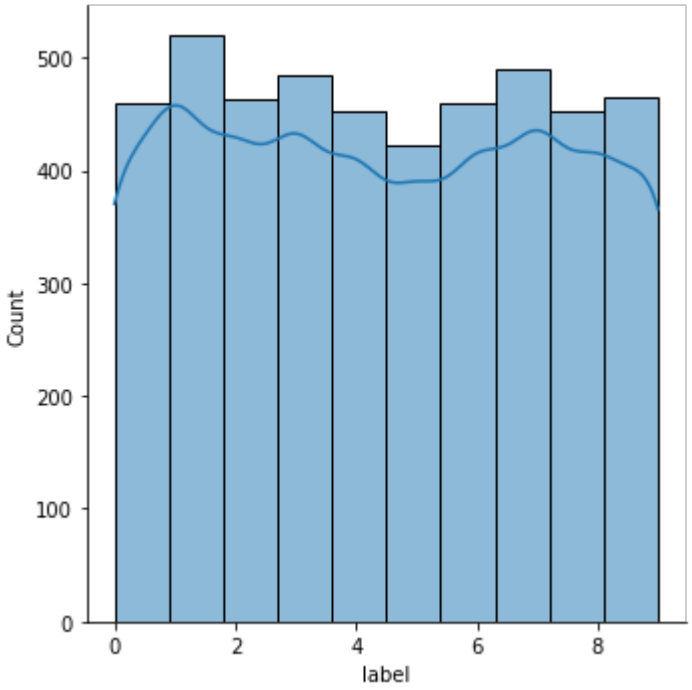


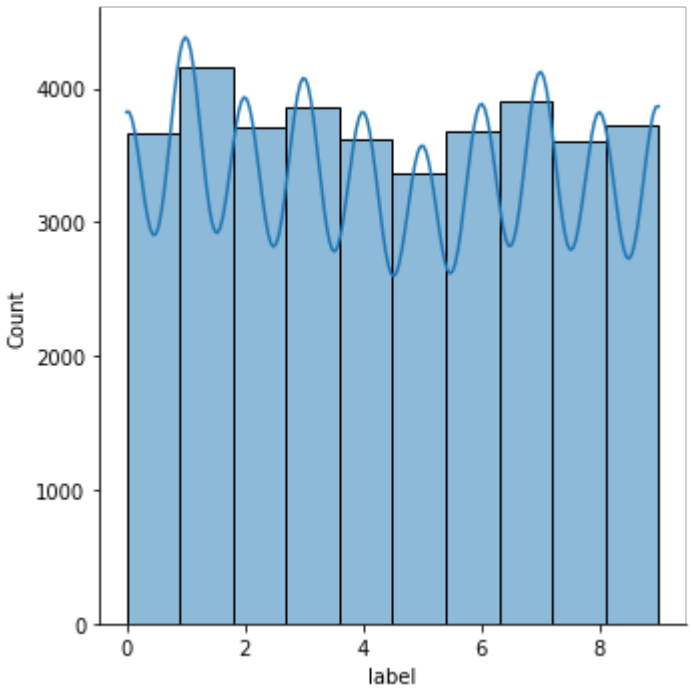
In [396]:
```
data.values[10000][1:].reshape(28, 28)
```

Out[396]:
```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
```

In [397]:
```
def split_data_1to10(data):
    test = data.sort_values(by=['label'])[::9]
    train = pd.concat([data, test]).drop_duplicates(keep=False)
    sns.displot(test['label'], bins=10, kde=True);
    sns.displot(train['label'], bins=10, kde=True);
    return test, train
```

In [398]:
```
1 test, train = split_data_1to10(data)
```





In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [399]:
```
1 class MyNN:
2     def __init__(self):
3         self.learn_rate = 0.01
4 
5         self._weights0 = np.random.rand(10, 16)-0.5
6         self._weights1 = np.random.rand(16, 49)-0.5
7         self._weights2 = np.random.rand(49, 784)-0.5
```

```
 8
 9         self._input  = np.zeros([10])
10         self._layer1 = np.zeros([16])
11         self._layer2 = np.zeros([49])
12         self._output = np.zeros([784])
13
14         self._correction_0 = np.zeros([10, 16])
15         self._correction_1 = np.zeros([16, 49])
16         self._correction_2 = np.zeros([49, 784])
17
18         self._activation0 = self.act_relu
19         self._activation1 = self.act_relu
20         self._activation2 = self.act_relu
21
22         self._der_act_0 = self.der_act_relu
23         self._der_act_1 = self.der_act_relu
24         self._der_act_2 = self.der_act_relu
25
26     def zero(self):
27         self._input  = np.zeros([10])
28         self._layer1 = np.zeros([16])
29         self._layer2 = np.zeros([49])
30         self._output = np.zeros([784])
31
32     def act_sigmoid(self, layer):
33         return np.array([1/(1+np.exp(-x)) for x in layer])
34
35     def act_relu(self, layer):
36         return np.array([x if x > 0 else 0 for x in layer])
37
38     def act_softplus(self, layer):
39         return np.array([np.ln(1+np.exp(x)) for x in layer])
40
41     def der_act_sigmoid(self, l):
42         _exp = np.exp(-l)
43         return _exp / (1+_exp)**2
44
45     def der_act_relu(self, l):
46         if l > 0:
47             return 1
48         else:
49             return 0
50
51     def def_act_softplus(self, l):
52         return 1 / (1+np.exp(-l))
53
54     def predict(self, num):
55         self.zero()
56         self._input = np.array([100 if k == num[0] else 0 for k i
57         #print(num, self._input)
58
59         self._layer1 = np.dot(self._input, self._weights0)
60         self._layer1 = self._activation0(self._layer1)
61         #print("   ", self._layer1)
62
63
64         self._layer2 = np.dot(self._layer1, self._weights1)
65         self._layer2 = self._activation1(self._layer2)
66
67         self._output = np.dot(self._layer2, self._weights2)
```

```
68          self._output = self._activation2(self._output)
69
70          return self._output
71
72      def learn(self, labels):
73
74          main_delta   = self._output - labels
75          delta_output = main_delta * [self._der_act_2(x) for x in
76          delta_w_2    = np.dot(delta_output.reshape(len(delta_outp
77                                  self._layer2.reshape(1, len(self._l
78
79          delta_layer2 = np.dot(self._weights2, delta_output) * [se
80          delta_w_1    = np.dot(delta_layer2.reshape(len(delta_laye
81                                  self._layer1.reshape(1, len(self._l
82
83          delta_layer1 = np.dot(self._weights1, delta_layer2) * [se
84          delta_w_0    = np.dot(delta_layer1.reshape(len(delta_laye
85                                  self._input.reshape(1, len(self._in
86
87          self._correction_0 += np.transpose(delta_w_0)
88          self._correction_1 += np.transpose(delta_w_1)
89          self._correction_2 += np.transpose(delta_w_2)
90
91      def update(self):
92          self._weights0 = self._weights0 - self.learn_rate * self.
93          self._weights1 = self._weights1 - self.learn_rate * self.
94          self._weights2 = self._weights2 - self.learn_rate * self.
95
96          self._correction_0 = np.zeros([10, 16])
97          self._correction_1 = np.zeros([16, 49])
98          self._correction_2 = np.zeros([49, 784])
99
```

In [400]:

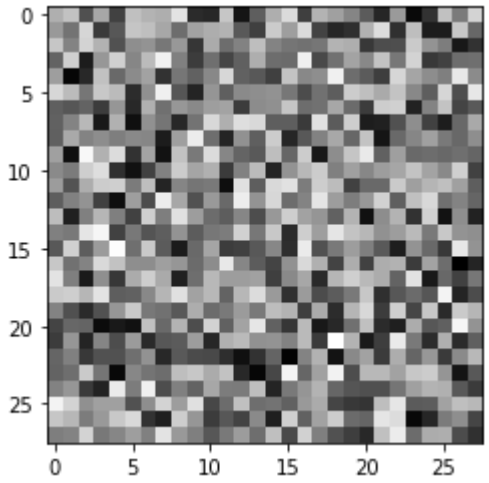
```
1  NN = MyNN()
2
3  NN._activation0 = NN.act_sigmoid
4  NN._activation1 = NN.act_sigmoid
5  NN._activation2 = NN.act_sigmoid
6
7  NN._der_act_0 = NN.der_act_sigmoid
8  NN._der_act_1 = NN.der_act_sigmoid
9  NN._der_act_2 = NN.der_act_sigmoid
10
```

In [ ]:

```
1
```

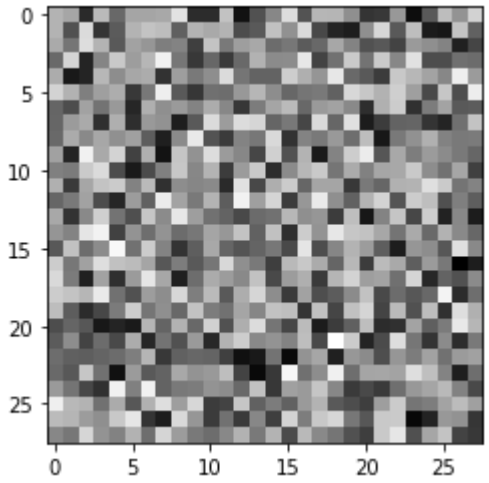
In [401]:
```
res = NN.predict([0])
plt.imshow(res.reshape(28, 28), cmap='gray')
```

Out[401]: <matplotlib.image.AxesImage at 0x7fb7201760d0>



In [402]:
```
res = NN.predict([1])
plt.imshow(res.reshape(28, 28), cmap='gray')
```

Out[402]: <matplotlib.image.AxesImage at 0x7fb71fe8e250>



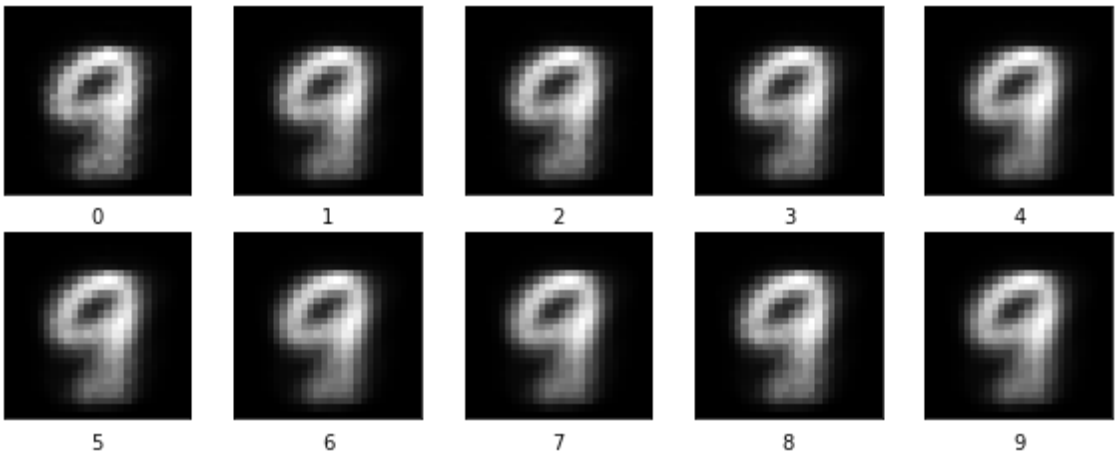
In [403]:
```
def print_predictions(nn):
    plt.figure(figsize=(10,10))
    for i in range(10):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        res = nn.predict([i])
        plt.imshow(res.reshape(28, 28), cmap='gray')
        plt.xlabel(i)
    plt.show()

```

In [404]:
```
#data = data.sort_values(by=['label'])

```

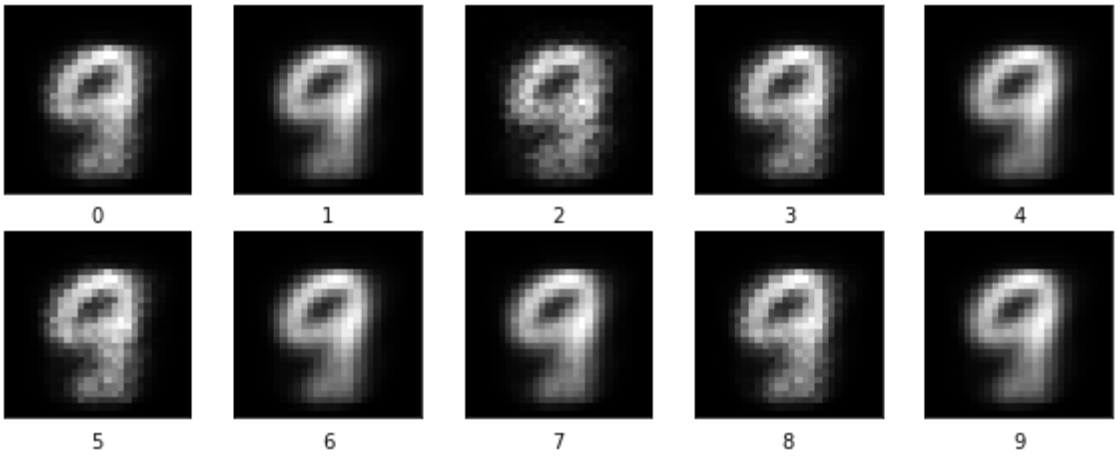
In [405]:
```
epohs = 30
uprate = 5
```

```
print_flag = 5000
_data = test
for e in range(epohs):
    print('E:', e)
    for i in tqdm(range(len(_data))):
        NN.predict( _data.values[i][:1] )
        expect = _data.values[i][1:]
        NN.learn(expect)
        if not i % uprate:
            NN.update()
        #if not i % print_flag:
        #    print_predictions(NN)
    print_predictions(NN)
```

```
E: 0

  1%|
| 61/4667 [00:02<03:03, 25.07it/s]/tmp/ipykernel_5437/3794245865.p
y:33: RuntimeWarning: overflow encountered in exp
  return np.array([1/(1+np.exp(-x)) for x in layer])
100%|
| 4667/4667 [02:32<00:00, 30.6
1it/s]
```

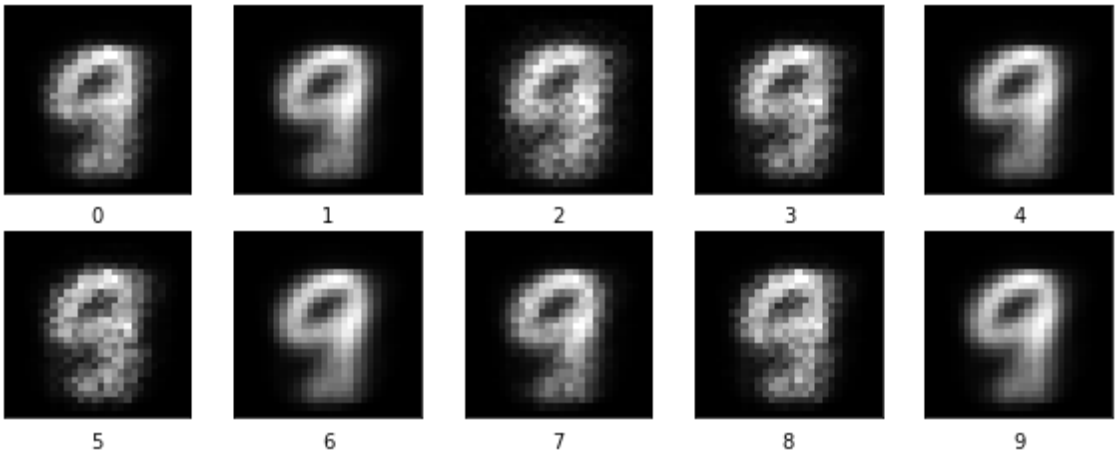



```
E: 1

100%|
| 4667/4667 [02:32<00:00, 30.6
9it/s]
```



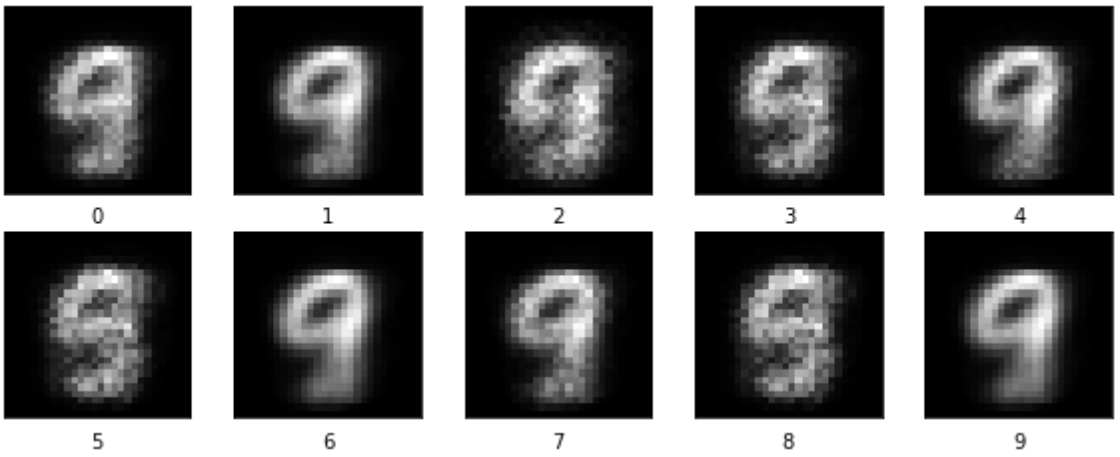

```
E: 2

100%|
```

| 4667/4667 [02:26<00:00, 31.8
8it/s]



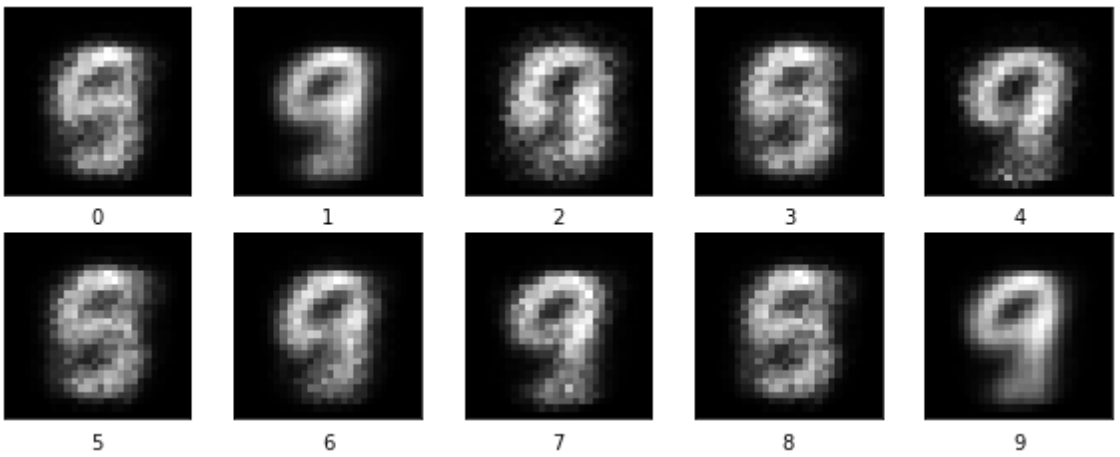
E: 3

100%| | 4667/4667 [02:25<00:00, 32.1
6it/s]



E: 4

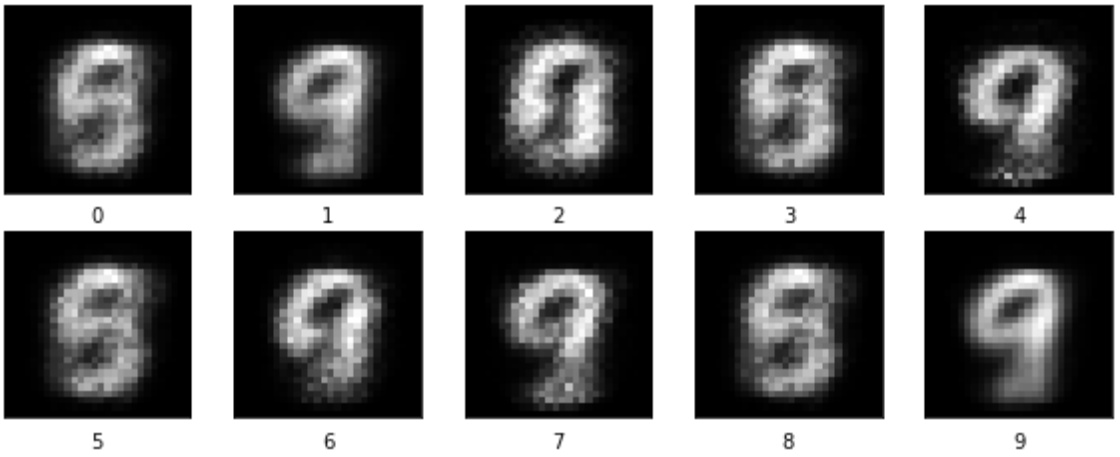
100%| | 4667/4667 [02:29<00:00, 31.2
1it/s]



E: 5

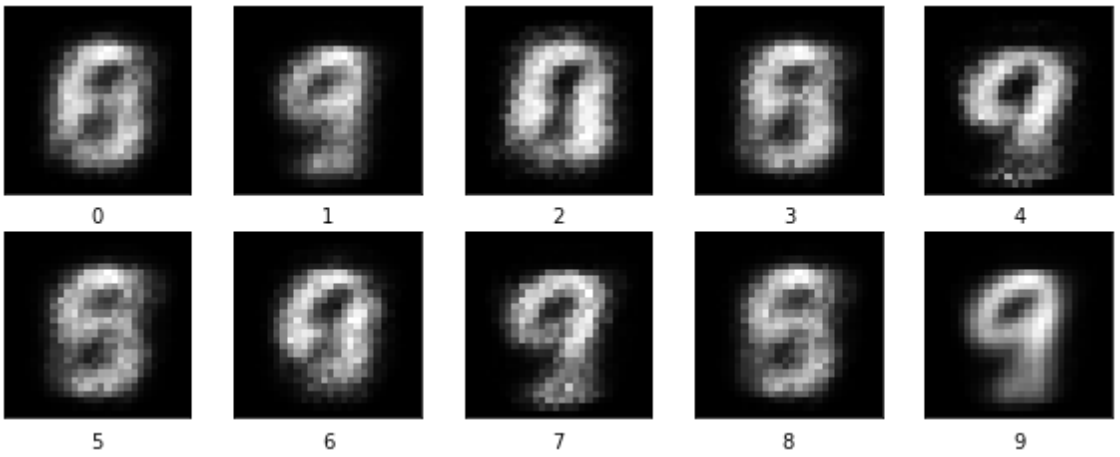
100%|

| 4667/4667 [02:26<00:00, 31.90it/s]



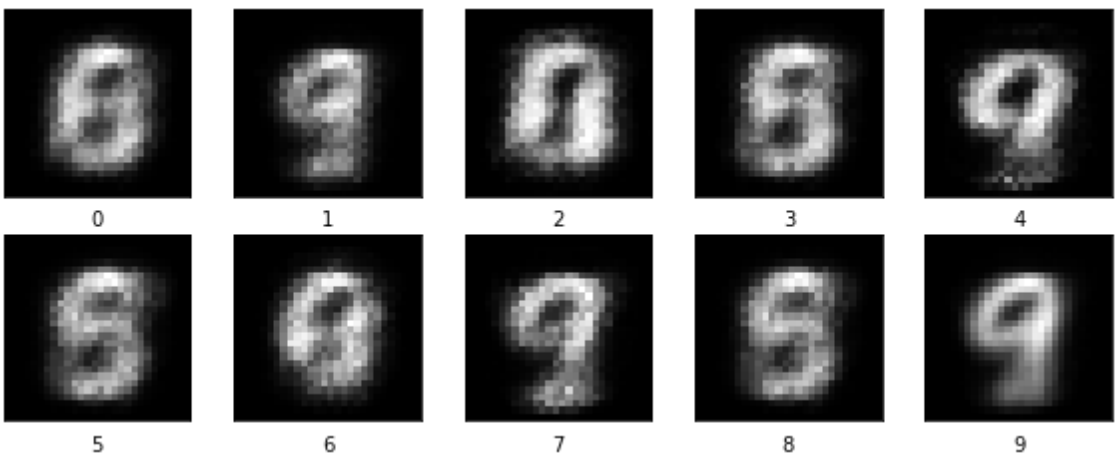
E: 6

100%| | 4667/4667 [02:25<00:00, 32.04it/s]



E: 7

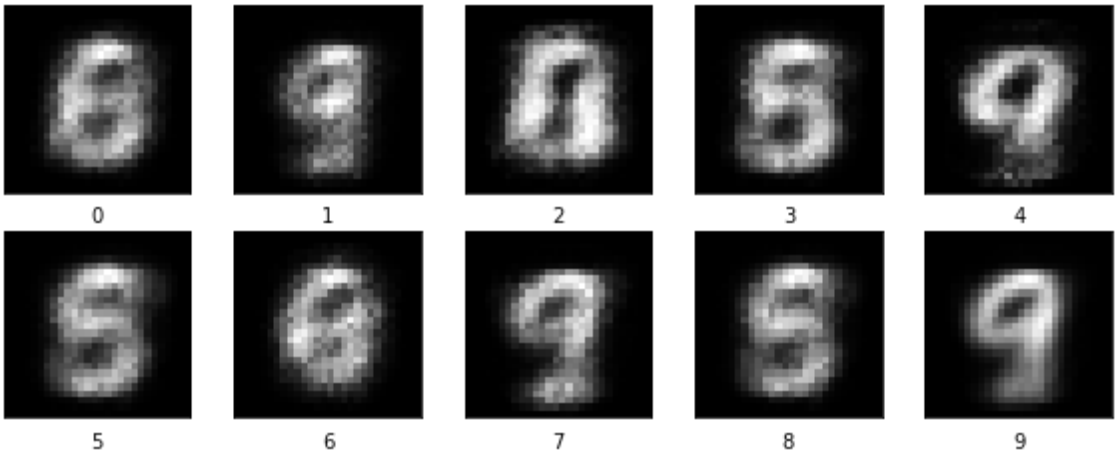
100%| | 4667/4667 [02:25<00:00, 32.13it/s]



E: 8

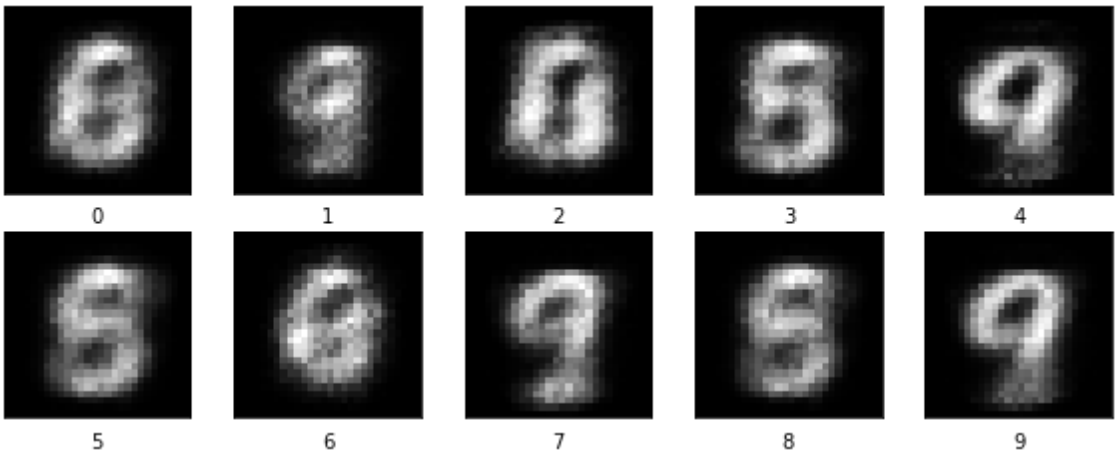
100%|

100%|| 4667/4667 [02:33<00:00, 30.32it/s]



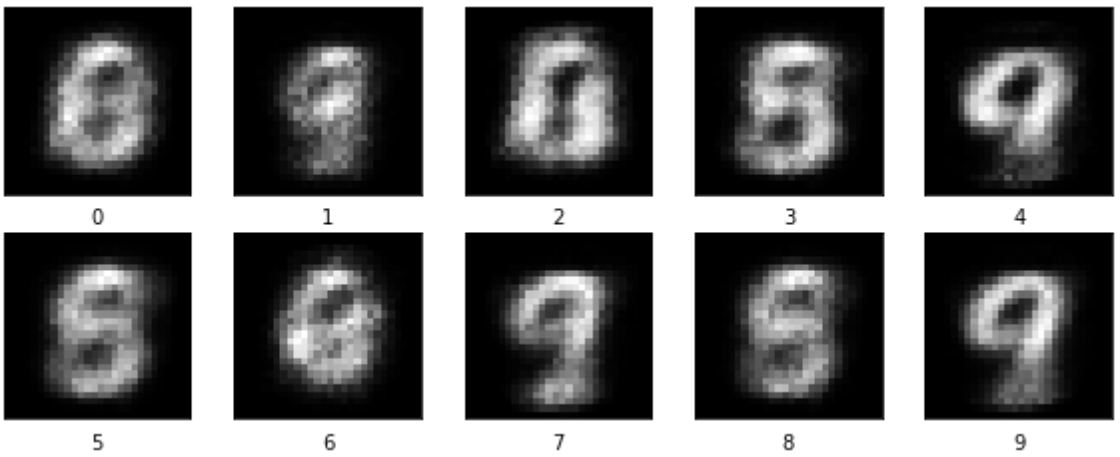
E: 9

100%|| 4667/4667 [02:27<00:00, 31.68it/s]



E: 10

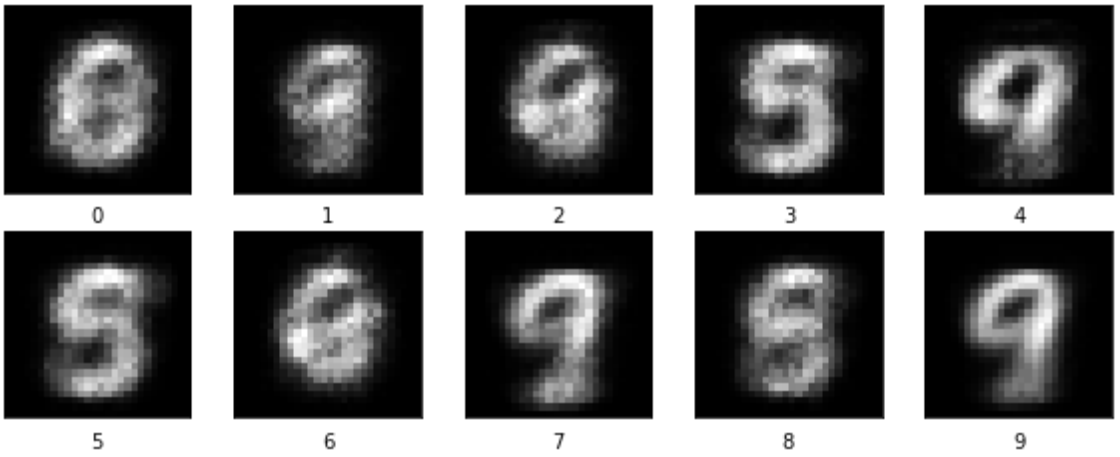
100%|| 4667/4667 [02:25<00:00, 32.07it/s]



E: 11

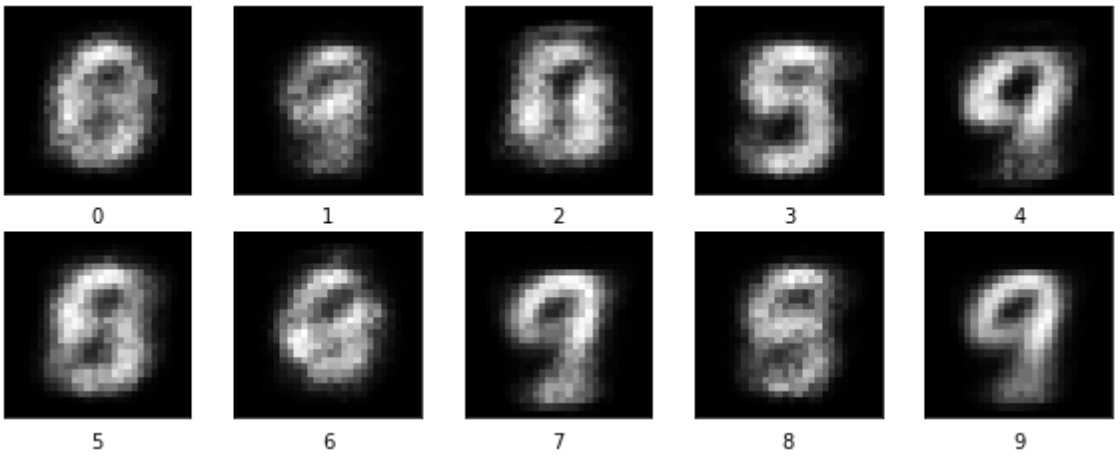
100%|

100%|| 4667/4667 [02:30<00:00, 31.08it/s]



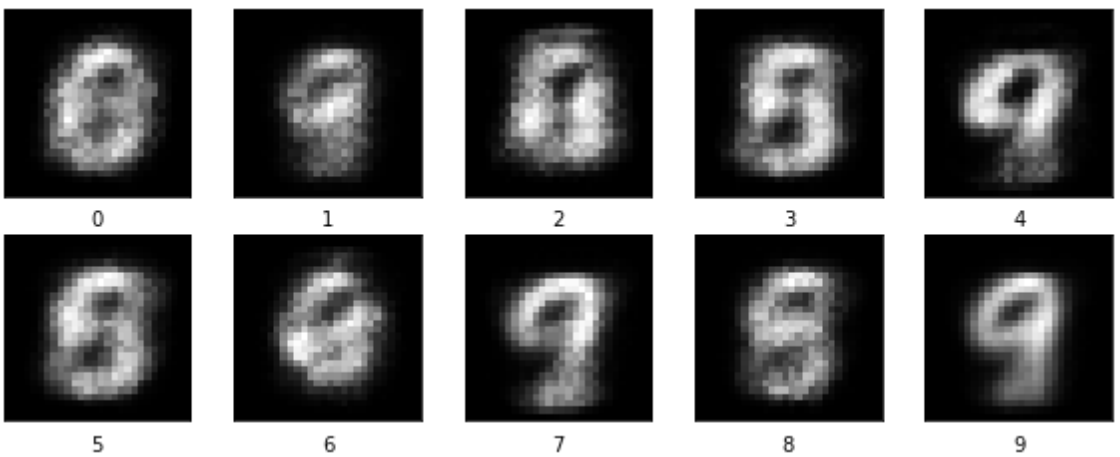
E: 12

100%|| 4667/4667 [02:24<00:00, 32.26it/s]



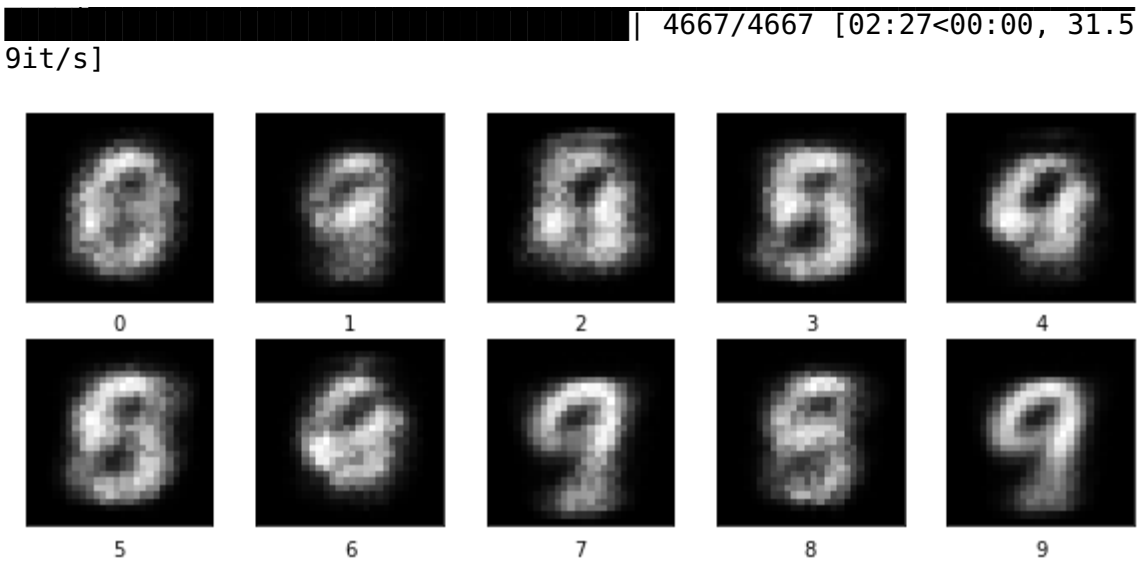
E: 13

100%|| 4667/4667 [02:28<00:00, 31.32it/s]

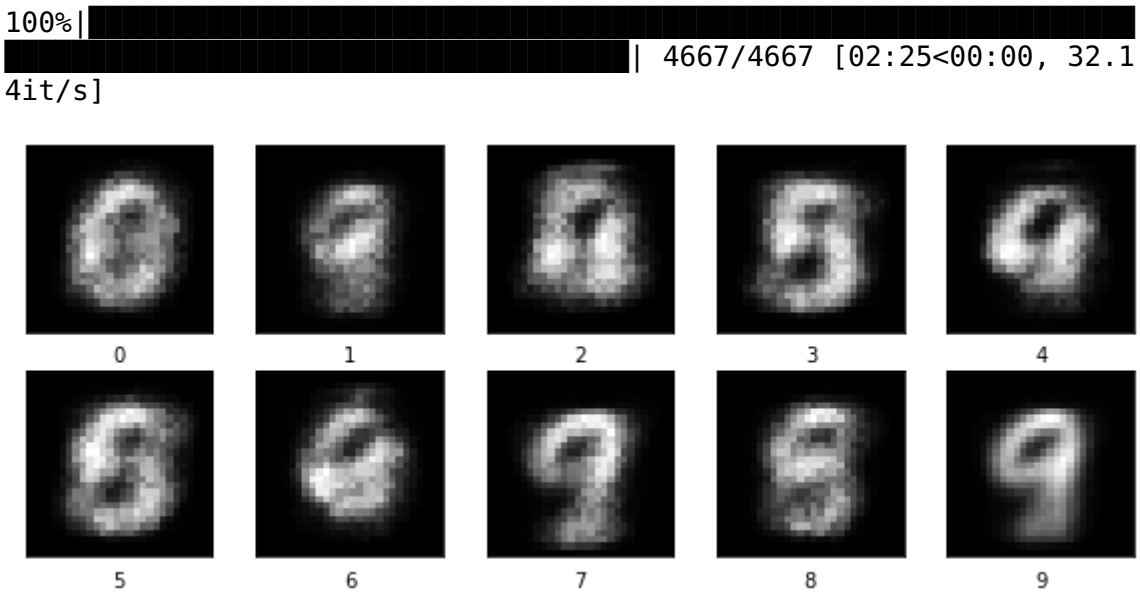


E: 14

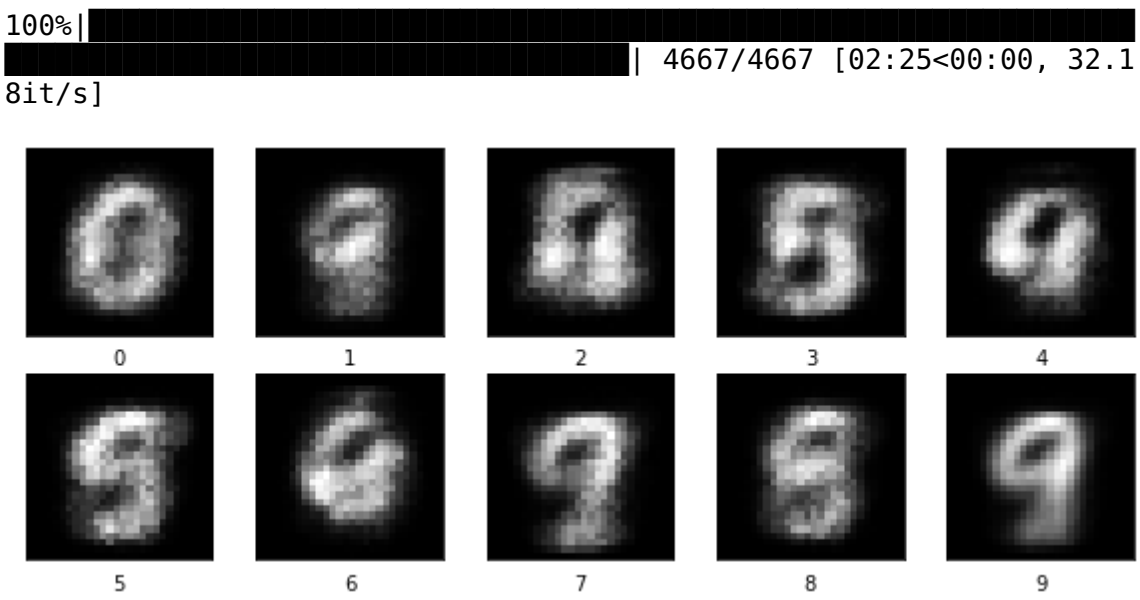
100%|


| 4667/4667 [02:27<00:00, 31.59it/s]
0
1
2
3
4
5
6
7
8
9

E: 15


100%|
| 4667/4667 [02:25<00:00, 32.14it/s]
0
1
2
3
4
5
6
7
8
9

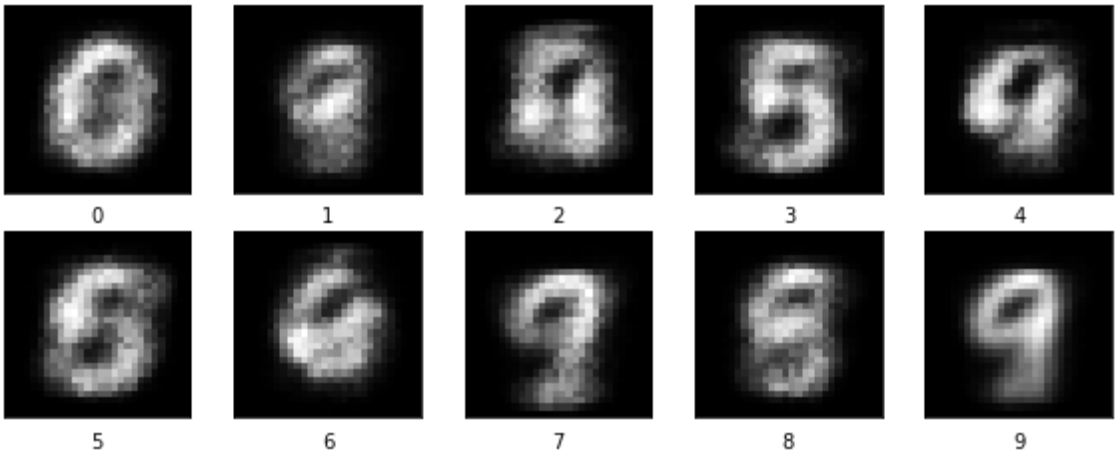
E: 16


100%|
| 4667/4667 [02:25<00:00, 32.18it/s]
0
1
2
3
4
5
6
7
8
9

E: 17

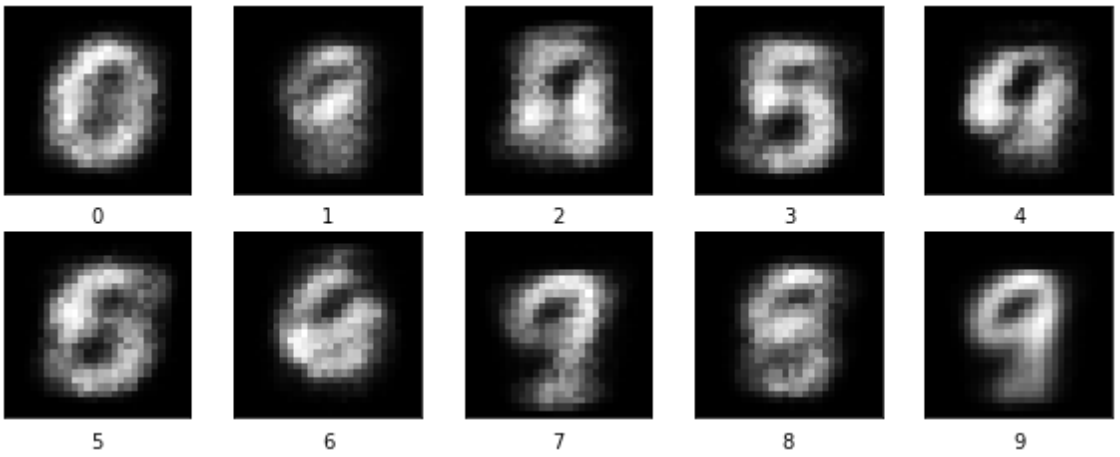

100%|

100%|| 4667/4667 [02:27<00:00, 31.65it/s]



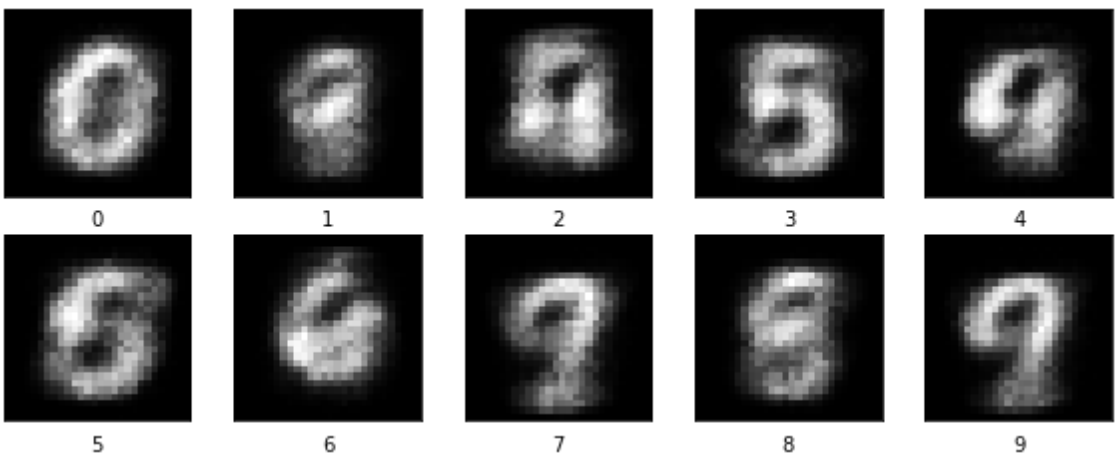
E: 18

100%|| 4667/4667 [02:25<00:00, 32.05it/s]



E: 19

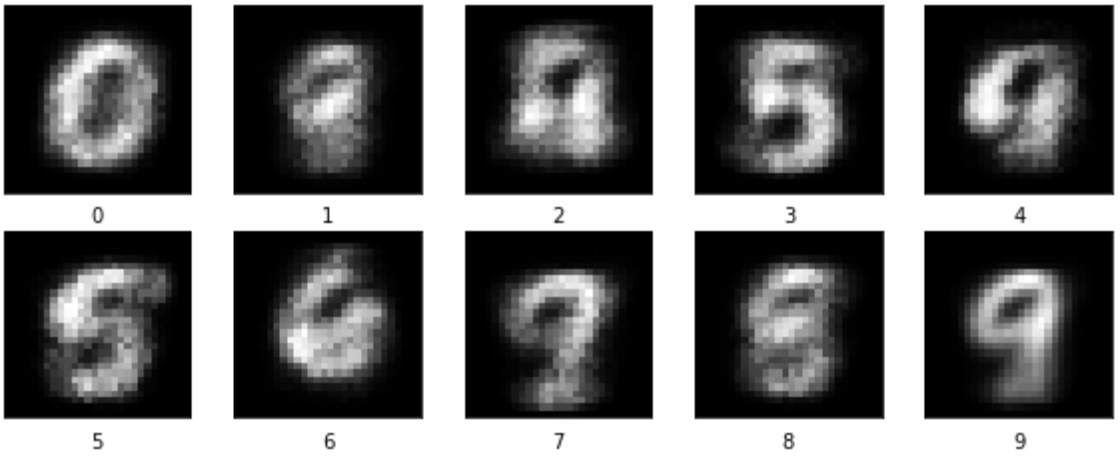
100%|| 4667/4667 [02:27<00:00, 31.72it/s]



E: 20

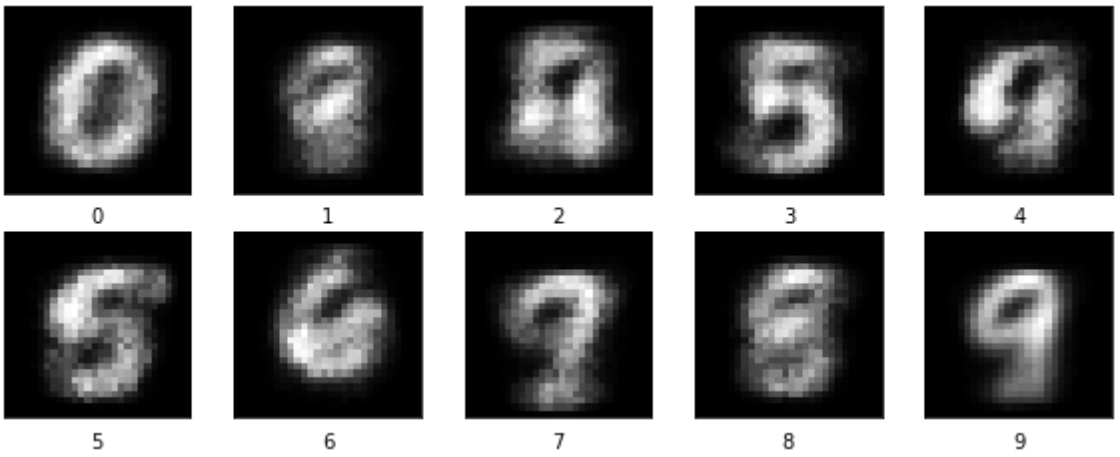
100%|

| 4667/4667 [02:25<00:00, 32.02it/s]



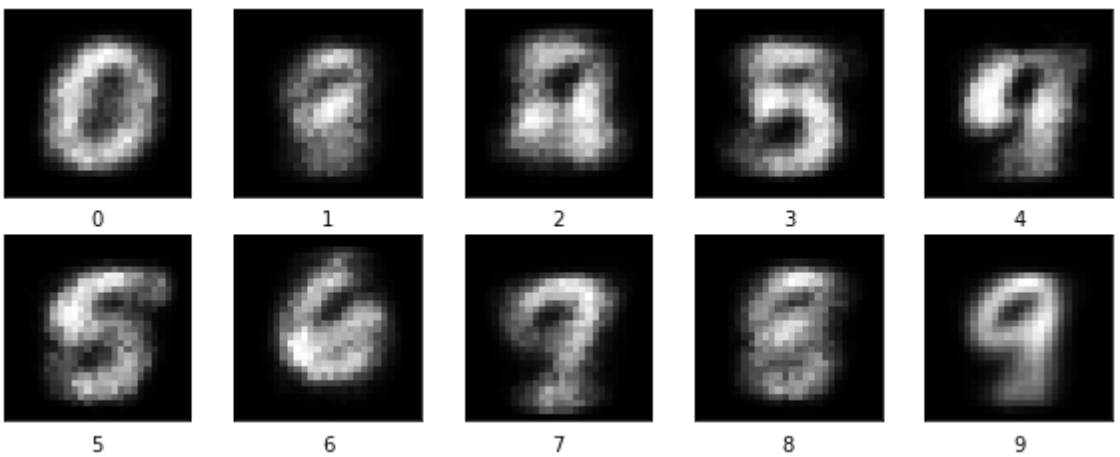
E: 21

100%| | 4667/4667 [02:26<00:00, 31.93it/s]



E: 22

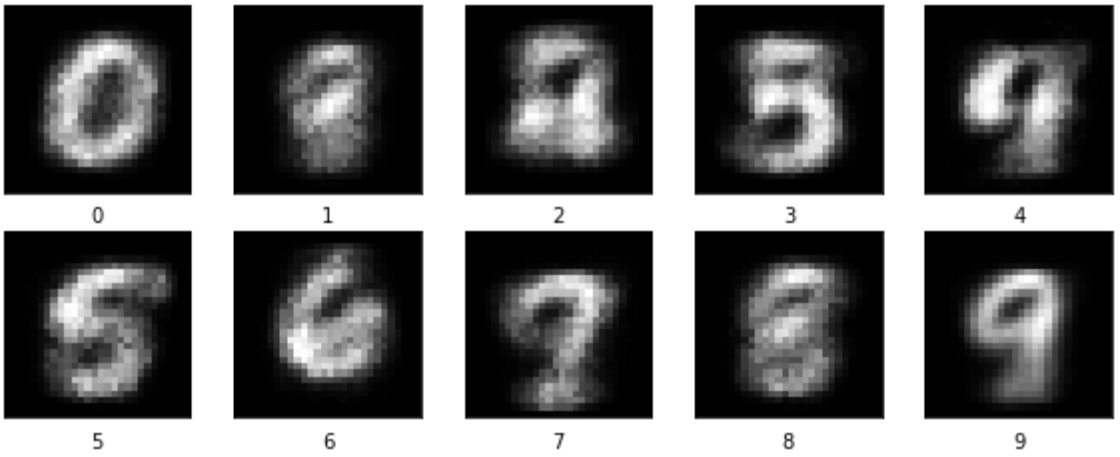
100%| | 4667/4667 [02:25<00:00, 32.17it/s]



E: 23

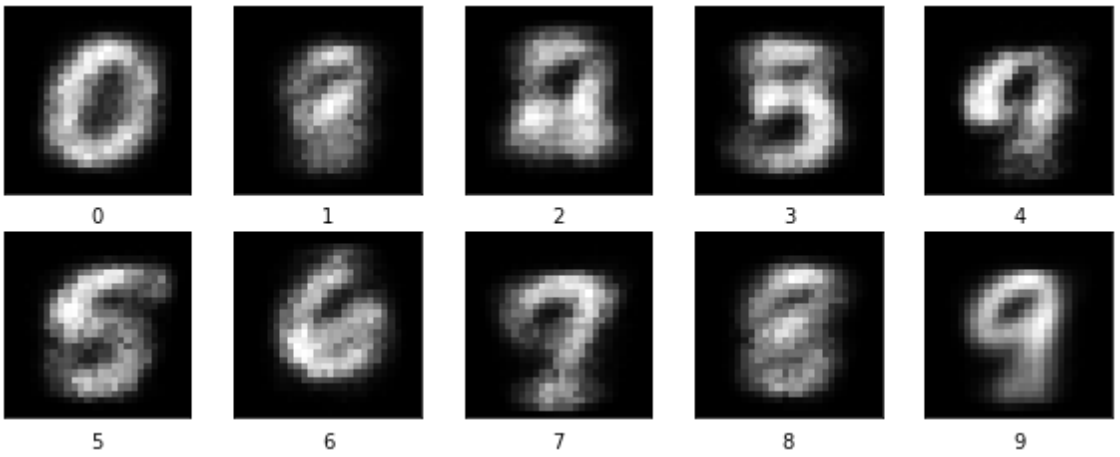
100%|

100%|| 4667/4667 [02:25<00:00, 32.04it/s]



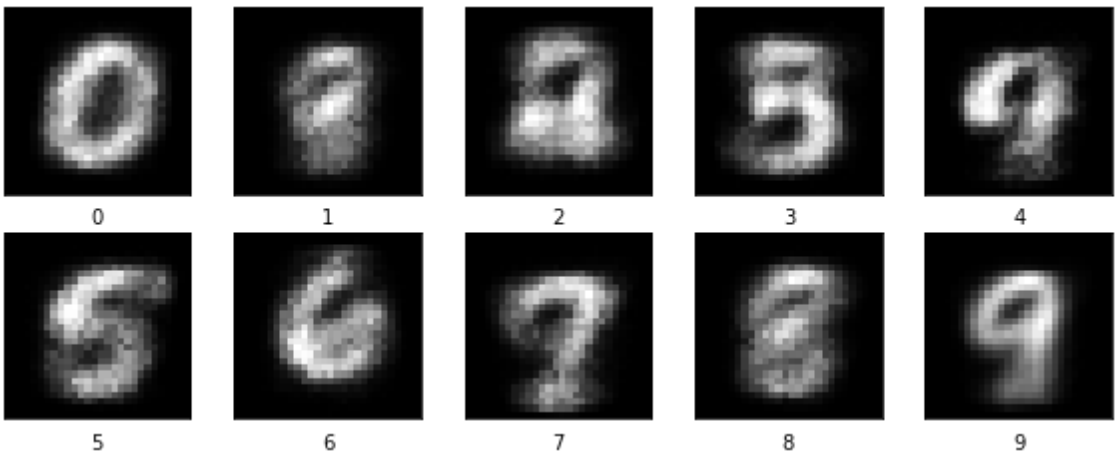
E: 24

100%|| 4667/4667 [02:24<00:00, 32.27it/s]



E: 25

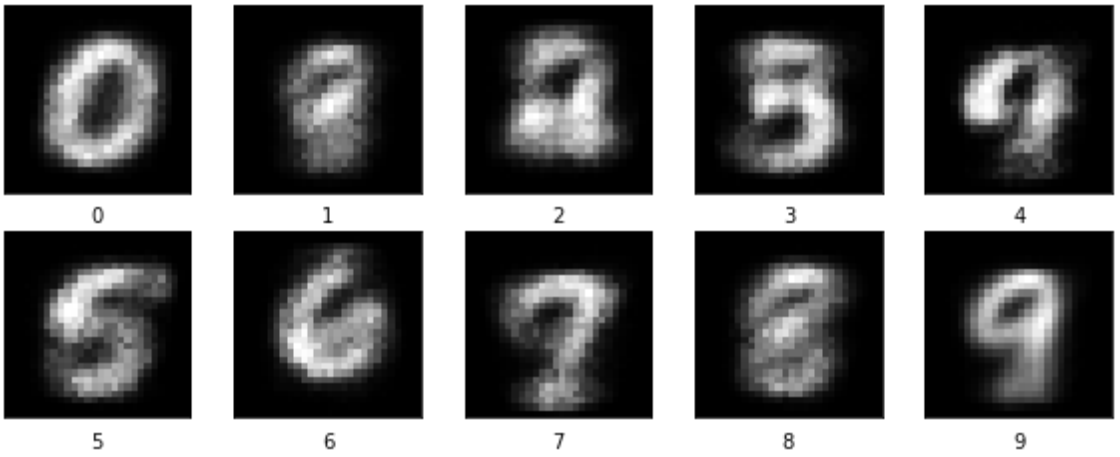
100%|| 4667/4667 [02:29<00:00, 31.27it/s]



E: 26

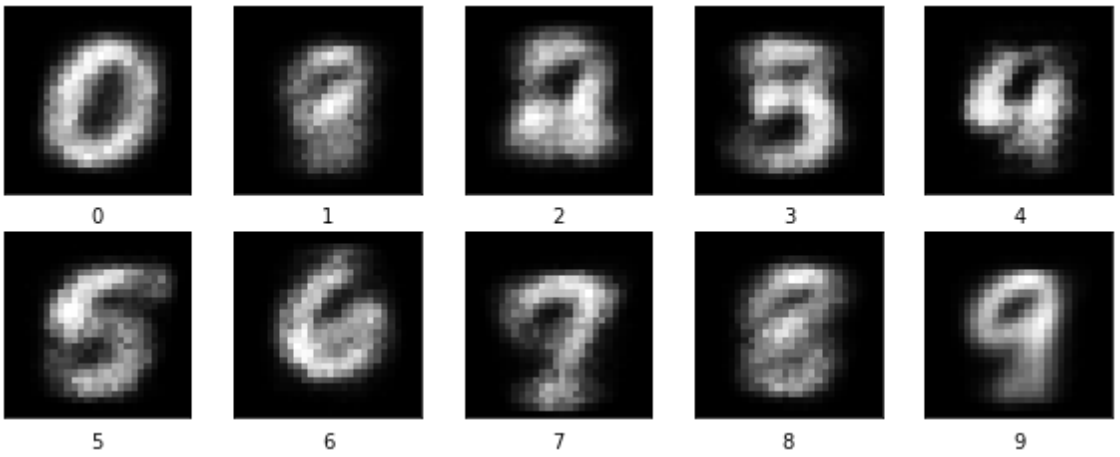
100%|

| 4667/4667 [02:25<00:00, 32.18it/s]



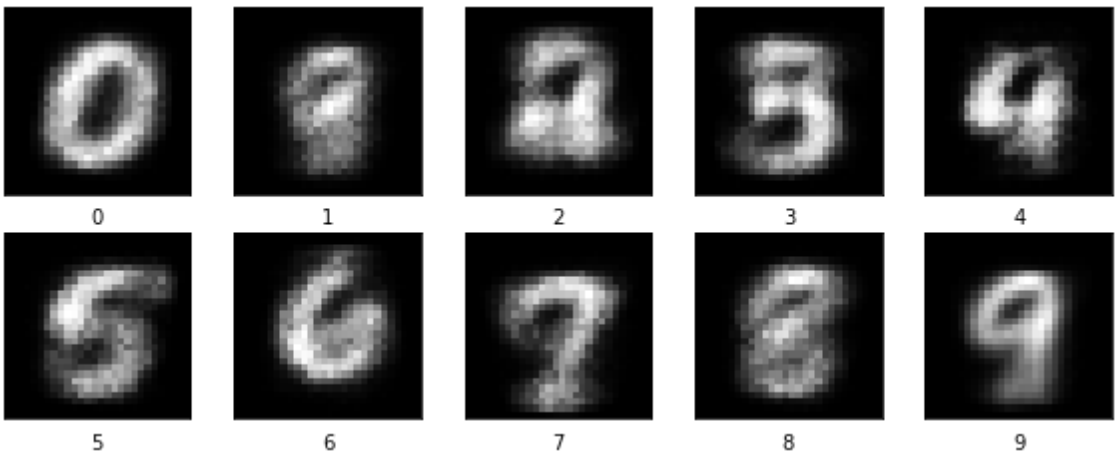
E: 27

100%| | 4667/4667 [02:25<00:00, 32.06it/s]



E: 28

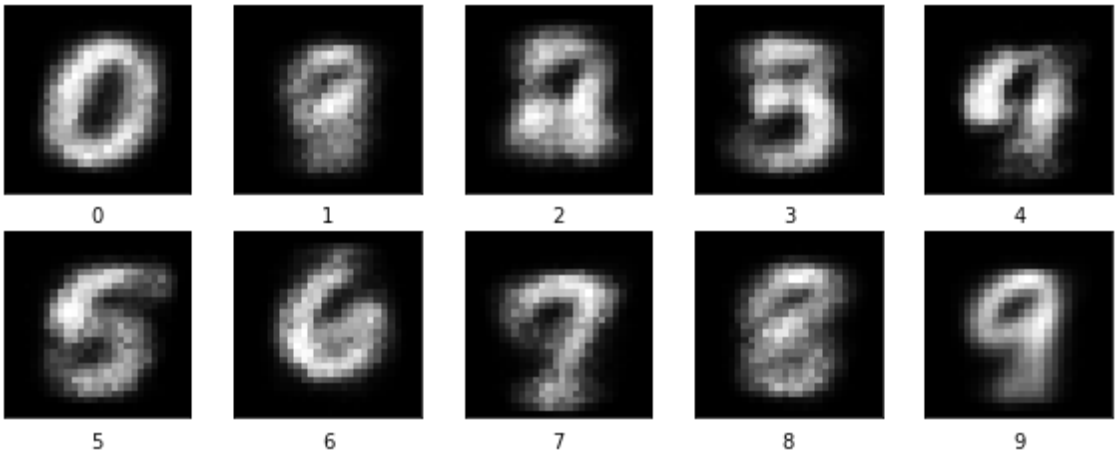
100%| | 4667/4667 [02:24<00:00, 32.22it/s]



E: 29

100%|

```
| 4667/4667 [02:35<00:00, 29.96it/s]
```



In [ ]:
```
1 res = NN.predict([1])
2 plt.imshow(res.reshape(28, 28), cmap='gray')
```

In [329]:
```
1 np.exp(-1)
```

Out[329]:
```
0.36787944117144233
```

In [325]:
```
1 np.log(-1)
```

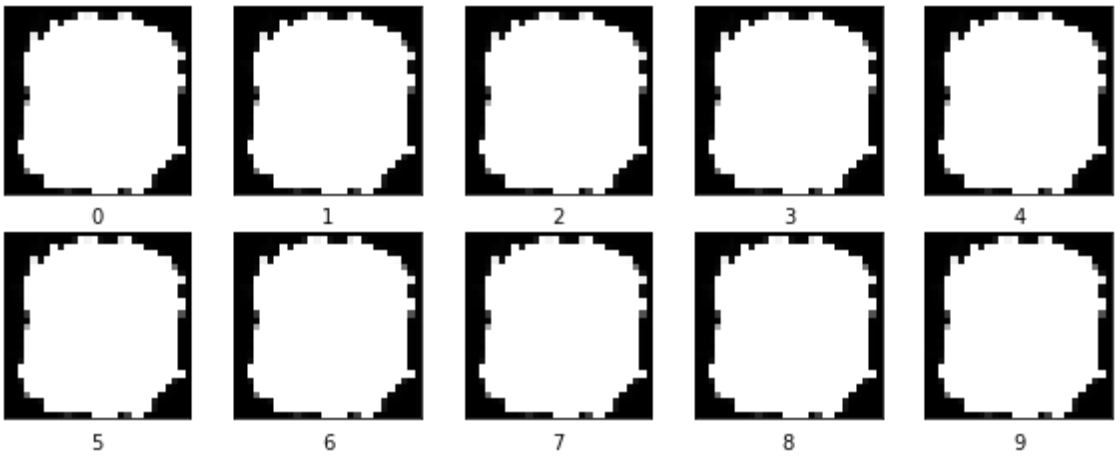
```
/tmp/ipykernel_5437/153577755.py:1: RuntimeWarning: invalid value encountered in log
  np.log(-1)
```

Out[325]:
```
nan
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [57]:
```
plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    res = NN.predict([i])
    plt.imshow(res.reshape(28, 28), cmap='gray')
    plt.xlabel(i)
plt.show()

```



In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
class MyNN:
    def __init__(self):
        self.learn_rate = 0.01

        self._weights0 = np.random.rand(784, 49)*0.1 - 0.05
        self._weights1 = np.random.rand(49, 16)*0.1 - 0.05
        self._weights2 = np.random.rand(16, 10)*0.1 - 0.05

        self._input  = np.zeros([784])
        self._layer1 = np.zeros([49])
        self._layer2 = np.zeros([16])
        self._output = np.zeros([10])

        self._correction_0 = np.zeros([784, 49])
        self._correction_1 = np.zeros([49, 16])
        self._correction_2 = np.zeros([16, 10])

        self._activation0 = self.act_relu
        self._activation1 = self.act_relu
        self._activation2 = self.act_relu

        self._der_act_0 = self.der_act_relu
        self._der_act_1 = self.der_act_relu
```

```
        self._der_act_2 = self.der_act_relu

    def zero(self):
        self._input  = np.zeros([784])
        self._layer1 = np.zeros([49])
        self._layer2 = np.zeros([16])
        self._output = np.zeros([10])

    def act_sigmoid(self, layer):
        return np.array([1/(1+np.exp(-x)) for x in layer])

    def act_relu(self, layer):
        return np.array([x if x > 0 else 0 for x in layer])

    def act_softplus(self, layer):
        return np.array([np.ln(1+np.exp(x)) for x in layer])

    def der_act_sigmoid(self, l):
        _exp = np.exp(-l)
        return _exp / (1+_exp)**2

    def der_act_relu(self, l):
        if l > 0:
            return 1
        else:
            return 0

    def def_act_softplus(self, l):
        return 1 / (1+np.exp(-l))

    def predict(self, img):
        self.zero()
        self._input = img

        self._layer1 = np.dot(self._input, self._weights0)
        self._layer1 = self._activation0(self._layer1)

        self._layer2 = np.dot(self._layer1, self._weights1)
        self._layer2 = self._activation1(self._layer2)

        self._output = np.dot(self._layer2, self._weights2)
        self._output = self._activation2(self._output)

        return self._output

    def learn(self, labels):
        main_delta   = self._output - labels
        delta_output = main_delta * [self._der_act_2(x) for x in
        delta_w_2    = np.dot(delta_output.reshape(len(delta_out
                               self._layer2.reshape(1, len(self._

        delta_layer2 = np.dot(self._weights2, delta_output) * [s
        delta_w_1    = np.dot(delta_layer2.reshape(len(delta_lay
                               self._layer1.reshape(1, len(self._

        delta_layer1 = np.dot(self._weights1, delta_layer2) * [s
        delta_w_0    = np.dot(delta_layer1.reshape(len(delta_lay
                               self._input.reshape(1, len(self._i

        self._correction_0 += np.transpose(delta_w_0)
```

```
 84          self._correction_1 += np.transpose(delta_w_1)
 85          self._correction_2 += np.transpose(delta_w_2)
 86
 87      def update(self):
 88          self._weights0 = self._weights0 - self.learn_rate * self
 89          self._weights1 = self._weights1 - self.learn_rate * self
 90          self._weights2 = self._weights2 - self.learn_rate * self
 91
 92          self._correction_0 = np.zeros([784, 49])
 93          self._correction_1 = np.zeros([49, 16])
 94          self._correction_2 = np.zeros([16, 10])
 95
 96      def check_correct(self, test):
 97          correct = 0
 98          for i in range(len(test)):
 99              res = np.argmax(self.predict(test.values[i][1:]))
100              if res == test.values[i][:1]:
101                  correct+=1
102          return correct/len(test)
103
104      def fit(self, test, train, epochs, learn_rate, batch):
105          self.learn_rate = learn_rate
106          for e in range(epochs):
107              for i in range(len(train)):
108                  NN.predict( train.values[i][1:] )
109                  expect = [1 if k == train.values[i][:1] else 0 f
110                  NN.learn(expect)
111                  if not i %batch:
112                      NN.update()
113              print("e{}, corr = {}".format(e, NN.check_correct(te
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```