

```
In [0]: # if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
In [0]: #%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):

    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")

    if (len(x)>1):

        plt.legend()
        plt.grid()

    fig.canvas.draw()
```

```
In [0]: # the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
In [0]: print("Number of training examples : ", X_train.shape[0], "and each image is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples : ", X_test.shape[0], "and each image is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

Number of training examples : 60000 and each image is of shape (28, 28)

Number of training examples : 10000 and each image is of shape (28, 28)

```
In [0]: # if you observe the input shape its 2 dimensional vector  
# for each image we have a (28*28) vector  
# we will convert the (28*28) vector into single dimensional vector of  
1 * 784  
  
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])  
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

```
In [0]: # after converting the input images from 3d to 2d vectors  
  
print("Number of training examples : ", X_train.shape[0], "and each image  
      is of shape (%d)"%(X_train.shape[1]))  
print("Number of training examples : ", X_test.shape[0], "and each image  
      is of shape (%d)"%(X_test.shape[1]))
```

Number of training examples : 60000 and each image is of shape (784)  
Number of training examples : 10000 and each image is of shape (784)

```
In [0]: # An example data point  
print(X_train[0])
```

0	0	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	3	18	18	18	126	136	175	26	166	25	
4	247	127	0	0	0	0	0	0	0	0	0	0	0	0	30	36	94	15	
0	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0	0	0	
2	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	8	
3	82	56	39	0	0	0	0	0	0	0	0	0	0	0	0	18	219	25	
0	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	15	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	14	1	154	253	90	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	139	253	190	2	
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	11	190	253	70	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	35	24
0	225	160	108	1	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	81	240	253	253	119	25	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	45	186	253	253	150	27	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
In [0]: # if we observe the above matrix each cell is having a value between 0-255  
# before we move to apply machine learning algorithms lets try to normalize the data
```

```
# X => (X - Xmin)/(Xmax-Xmin) = X/255
```

```
X_train = X_train/255
```

$$\bar{X}_{\text{test}} = X_{\text{test}} / 255$$

```
In [0]: # example data point after normalizing  
print(X_train[0])
```

0.88235294	0.6745098	0.99215686	0.94901961	0.76470588	0.25098039
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.19215686
0.93333333	0.99215686	0.99215686	0.99215686	0.99215686	0.99215686
0.99215686	0.99215686	0.99215686	0.98431373	0.36470588	0.32156863
0.32156863	0.21960784	0.15294118	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.07058824	0.85882353	0.99215686
0.99215686	0.99215686	0.99215686	0.99215686	0.77647059	0.71372549
0.96862745	0.94509804	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.31372549	0.61176471	0.41960784	0.99215686
0.99215686	0.80392157	0.04313725	0.	0.16862745	0.60392157
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.05490196	0.00392157	0.60392157	0.99215686	0.35294118
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.	0.54509804	0.99215686	0.74509804	0.00784314	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.04313725
0.74509804	0.99215686	0.2745098	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.	0.1372549	0.94509804
0.88235294	0.62745098	0.42352941	0.00392157	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.31764706	0.94117647	0.99215686
0.99215686	0.46666667	0.09803922	0.	0.	0.
0.	0.	0.	0.	0.	0.

0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.17647059	0.72941176	0.99215686	0.99215686
0.58823529	0.10588235	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.0627451	0.36470588	0.98823529	0.99215686	0.73333333
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.	0.97647059	0.99215686	0.97647059	0.25098039	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.18039216	0.50980392	0.71764706	0.99215686
0.99215686	0.81176471	0.00784314	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.15294118	0.58039216
0.89803922	0.99215686	0.99215686	0.99215686	0.98039216	0.71372549
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.09411765	0.44705882	0.86666667	0.99215686	0.99215686	0.99215686
0.99215686	0.78823529	0.30588235	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.09019608	0.25882353	0.83529412	0.99215686
0.99215686	0.99215686	0.99215686	0.77647059	0.31764706	0.00784314
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.07058824	0.67058824
0.85882353	0.99215686	0.99215686	0.99215686	0.99215686	0.76470588
0.31372549	0.03529412	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.
0.21568627	0.6745098	0.88627451	0.99215686	0.99215686	0.99215686

```
In [0]: # here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 0, 1, 0,
# 0, 0, 0]
# this conversion needed for MLPs

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

print("After converting the output into a vector : ", Y_train[0])
```

---

Class label of first image : 5  
After converting the output into a vector : [0. 0. 0. 0. 0. 0. 1. 0. 0.  
0.]

```
In [0]: from keras.models import Sequential  
from keras.layers import Dense, Activation
```

```
In [0]: # some model parameters  
  
output_dim = 10  
input_dim = X_train.shape[1]  
  
batch_size = 128  
nb_epoch = 20
```

## Architecture 1: 784-501-101-10

### — Model 1: sigmoid activation + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization  
  
model_1 = Sequential()  
  
model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),  
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)) )  
  
model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN  
ormal(mean=0.0, stddev=0.058, seed=None)) )  
  
model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()
```

Model: "sequential\_15"

Layer (type)	Output Shape	Param #
dense_39 (Dense)	(None, 501)	393285

```
dense_40 (Dense)           (None, 101)      50702
dense_41 (Dense)           (None, 10)       1020
=====
Total params: 445,007
Trainable params: 445,007
Non-trainable params: 0
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 7s 110us/step - loss: 0.
5731 - acc: 0.8503 - val_loss: 0.2581 - val_acc: 0.9268
Epoch 2/20
60000/60000 [=====] - 6s 94us/step - loss: 0.2
244 - acc: 0.9342 - val_loss: 0.1858 - val_acc: 0.9430
Epoch 3/20
60000/60000 [=====] - 6s 95us/step - loss: 0.1
649 - acc: 0.9518 - val_loss: 0.1501 - val_acc: 0.9557
Epoch 4/20
60000/60000 [=====] - 6s 96us/step - loss: 0.1
280 - acc: 0.9623 - val_loss: 0.1185 - val_acc: 0.9635
Epoch 5/20
60000/60000 [=====] - 6s 96us/step - loss: 0.1
001 - acc: 0.9701 - val_loss: 0.1010 - val_acc: 0.9683
Epoch 6/20
60000/60000 [=====] - 6s 96us/step - loss: 0.0
792 - acc: 0.9767 - val_loss: 0.0888 - val_acc: 0.9726
Epoch 7/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
640 - acc: 0.9817 - val_loss: 0.0837 - val_acc: 0.9726
Epoch 8/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
516 - acc: 0.9848 - val_loss: 0.0772 - val_acc: 0.9761
```

```
- -  
Epoch 9/20  
60000/60000 [=====] - 6s 94us/step - loss: 0.0  
423 - acc: 0.9874 - val_loss: 0.0812 - val_acc: 0.9738  
Epoch 10/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.0  
343 - acc: 0.9902 - val_loss: 0.0735 - val_acc: 0.9770  
Epoch 11/20  
60000/60000 [=====] - 6s 102us/step - loss: 0.  
0285 - acc: 0.9922 - val_loss: 0.0707 - val_acc: 0.9780  
Epoch 12/20  
60000/60000 [=====] - 6s 100us/step - loss: 0.  
0215 - acc: 0.9945 - val_loss: 0.0612 - val_acc: 0.9807  
Epoch 13/20  
60000/60000 [=====] - 6s 95us/step - loss: 0.0  
180 - acc: 0.9955 - val_loss: 0.0614 - val_acc: 0.9809  
Epoch 14/20  
60000/60000 [=====] - 6s 95us/step - loss: 0.0  
136 - acc: 0.9970 - val_loss: 0.0677 - val_acc: 0.9790  
Epoch 15/20  
60000/60000 [=====] - 6s 95us/step - loss: 0.0  
117 - acc: 0.9974 - val_loss: 0.0651 - val_acc: 0.9808  
Epoch 16/20  
60000/60000 [=====] - 6s 96us/step - loss: 0.0  
086 - acc: 0.9985 - val_loss: 0.0776 - val_acc: 0.9772  
Epoch 17/20  
60000/60000 [=====] - 6s 96us/step - loss: 0.0  
069 - acc: 0.9987 - val_loss: 0.0659 - val_acc: 0.9813  
Epoch 18/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
058 - acc: 0.9987 - val_loss: 0.0733 - val_acc: 0.9782  
Epoch 19/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
043 - acc: 0.9992 - val_loss: 0.0693 - val_acc: 0.9808  
Epoch 20/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
034 - acc: 0.9994 - val_loss: 0.0687 - val_acc: 0.9819
```

In [0]: `score = model_1.evaluate(X_test, Y_test, verbose=0)`

```
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

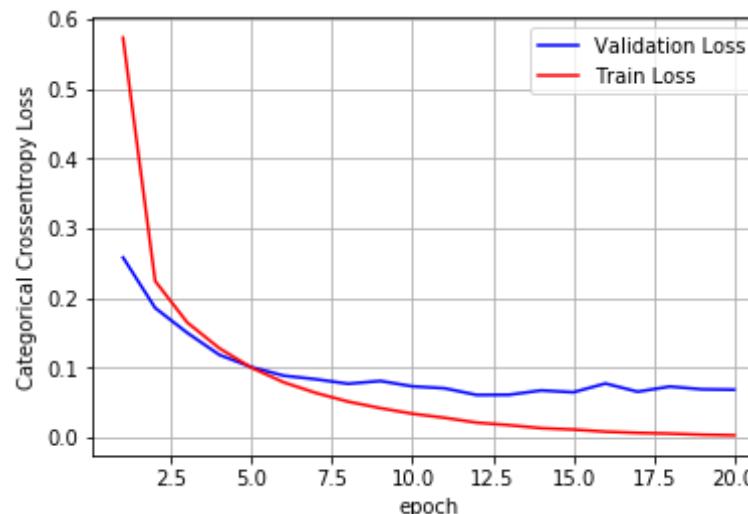
```
Test score: 0.06872767313871009
Test accuracy: 0.9819
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)
```

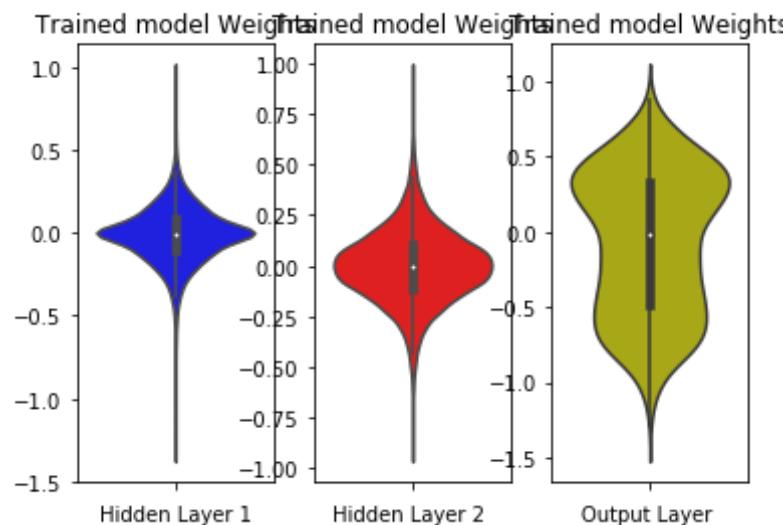
```

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 1 : sigmoid activation + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.058, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_16"

Layer (type)	Output Shape	Param #
dense_42 (Dense)	(None, 501)	393285
dense_43 (Dense)	(None, 101)	50702
dense_44 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metri
cs=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 6s 97us/step - loss: 2.2  
664 - acc: 0.2363 - val\_loss: 2.2134 - val\_acc: 0.4085  
Epoch 2/20  
60000/60000 [=====] - 5s 79us/step - loss: 2.1  
629 - acc: 0.4781 - val\_loss: 2.0989 - val\_acc: 0.5886

```
 520  acc: 0.7701 - val_loss: 2.0000 - val_acc: 0.5000
Epoch 3/20
60000/60000 [=====] - 5s 79us/step - loss: 2.0
294 - acc: 0.5880 - val_loss: 1.9394 - val_acc: 0.6175
Epoch 4/20
60000/60000 [=====] - 5s 78us/step - loss: 1.8
462 - acc: 0.6396 - val_loss: 1.7281 - val_acc: 0.6675
Epoch 5/20
60000/60000 [=====] - 5s 77us/step - loss: 1.6
229 - acc: 0.6761 - val_loss: 1.4950 - val_acc: 0.6982
Epoch 6/20
60000/60000 [=====] - 5s 78us/step - loss: 1.3
977 - acc: 0.7100 - val_loss: 1.2798 - val_acc: 0.7339
Epoch 7/20
60000/60000 [=====] - 5s 78us/step - loss: 1.2
064 - acc: 0.7401 - val_loss: 1.1102 - val_acc: 0.7695
Epoch 8/20
60000/60000 [=====] - 5s 79us/step - loss: 1.0
581 - acc: 0.7642 - val_loss: 0.9811 - val_acc: 0.7822
Epoch 9/20
60000/60000 [=====] - 5s 79us/step - loss: 0.9
452 - acc: 0.7844 - val_loss: 0.8823 - val_acc: 0.7976
Epoch 10/20
60000/60000 [=====] - 5s 79us/step - loss: 0.8
574 - acc: 0.8000 - val_loss: 0.8037 - val_acc: 0.8113
Epoch 11/20
60000/60000 [=====] - 5s 78us/step - loss: 0.7
868 - acc: 0.8125 - val_loss: 0.7404 - val_acc: 0.8227
Epoch 12/20
60000/60000 [=====] - 5s 78us/step - loss: 0.7
290 - acc: 0.8228 - val_loss: 0.6872 - val_acc: 0.8324
Epoch 13/20
60000/60000 [=====] - 5s 79us/step - loss: 0.6
809 - acc: 0.8317 - val_loss: 0.6437 - val_acc: 0.8412
Epoch 14/20
60000/60000 [=====] - 5s 78us/step - loss: 0.6
403 - acc: 0.8401 - val_loss: 0.6064 - val_acc: 0.8482
Epoch 15/20
60000/60000 [=====] - 5s 78us/step - loss: 0.6
058 - acc: 0.8470 - val_loss: 0.5745 - val_acc: 0.8531
```

```
Epoch 16/20
60000/60000 [=====] - 5s 78us/step - loss: 0.5
764 - acc: 0.8537 - val_loss: 0.5472 - val_acc: 0.8607
Epoch 17/20
60000/60000 [=====] - 5s 79us/step - loss: 0.5
509 - acc: 0.8587 - val_loss: 0.5232 - val_acc: 0.8661
Epoch 18/20
60000/60000 [=====] - 5s 77us/step - loss: 0.5
291 - acc: 0.8639 - val_loss: 0.5026 - val_acc: 0.8699
Epoch 19/20
60000/60000 [=====] - 5s 77us/step - loss: 0.5
098 - acc: 0.8677 - val_loss: 0.4850 - val_acc: 0.8738
Epoch 20/20
60000/60000 [=====] - 5s 77us/step - loss: 0.4
930 - acc: 0.8711 - val_loss: 0.4693 - val_acc: 0.8768
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

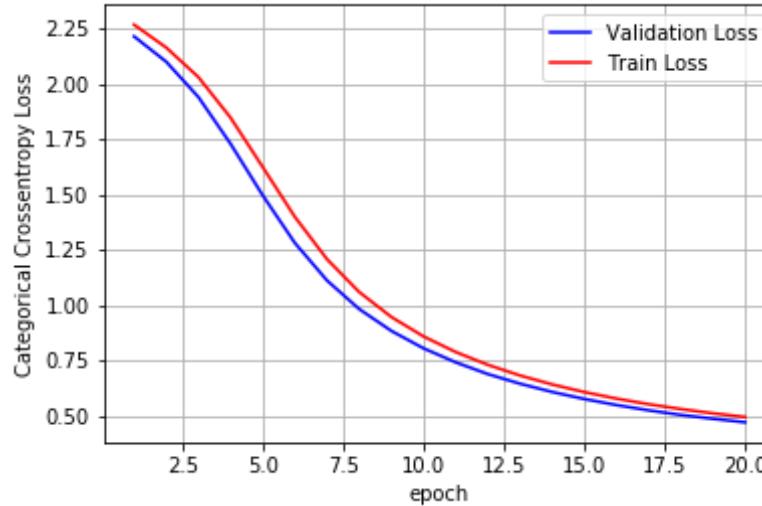
```
Test score: 0.46927674760818483
Test accuracy: 0.8768
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

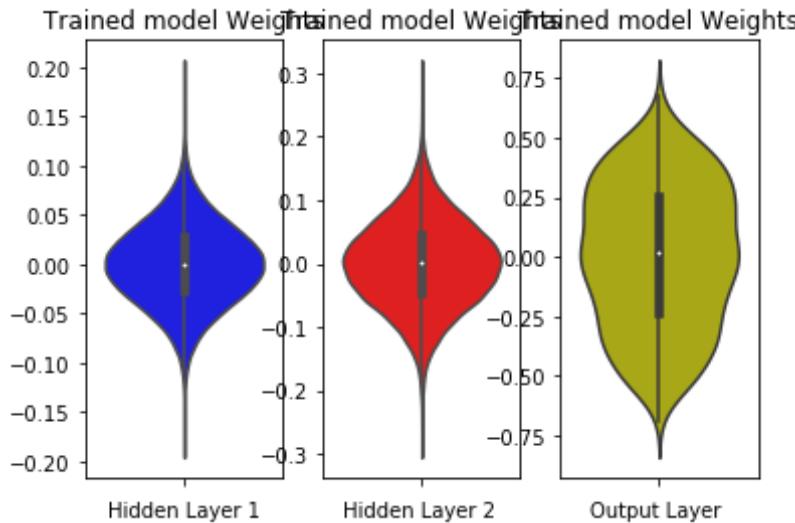
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



## Model 1 : sigmoid activation + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.140, seed=None)) )

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_17"

Layer (type)	Output Shape	Param #
=====		

dense_45 (Dense)	(None, 501)	393285
dense_46 (Dense)	(None, 101)	50702
dense_47 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 7s 113us/step - loss: 0.  
5234 - acc: 0.8697 - val_loss: 0.2584 - val_acc: 0.9266
Epoch 2/20
60000/60000 [=====] - 6s 95us/step - loss: 0.2  
225 - acc: 0.9363 - val_loss: 0.1880 - val_acc: 0.9440
Epoch 3/20
60000/60000 [=====] - 6s 95us/step - loss: 0.1  
650 - acc: 0.9521 - val_loss: 0.1452 - val_acc: 0.9572
Epoch 4/20
60000/60000 [=====] - 6s 95us/step - loss: 0.1  
272 - acc: 0.9630 - val_loss: 0.1232 - val_acc: 0.9635
Epoch 5/20
60000/60000 [=====] - 6s 95us/step - loss: 0.1  
011 - acc: 0.9708 - val_loss: 0.1056 - val_acc: 0.9670
Epoch 6/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0  
813 - acc: 0.9763 - val_loss: 0.0934 - val_acc: 0.9710
Epoch 7/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0  
666 - acc: 0.9807 - val_loss: 0.0847 - val_acc: 0.9723
Epoch 8/20
```

```
60000/60000 [=====] - 6s 95us/step - loss: 0.0
541 - acc: 0.9845 - val_loss: 0.0760 - val_acc: 0.9764
Epoch 9/20
60000/60000 [=====] - 6s 96us/step - loss: 0.0
444 - acc: 0.9869 - val_loss: 0.0715 - val_acc: 0.9774
Epoch 10/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
350 - acc: 0.9902 - val_loss: 0.0712 - val_acc: 0.9774
Epoch 11/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
284 - acc: 0.9923 - val_loss: 0.0671 - val_acc: 0.9786
Epoch 12/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
236 - acc: 0.9938 - val_loss: 0.0625 - val_acc: 0.9809
Epoch 13/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
188 - acc: 0.9954 - val_loss: 0.0623 - val_acc: 0.9797
Epoch 14/20
60000/60000 [=====] - 6s 98us/step - loss: 0.0
149 - acc: 0.9966 - val_loss: 0.0655 - val_acc: 0.9797
Epoch 15/20
60000/60000 [=====] - 6s 98us/step - loss: 0.0
121 - acc: 0.9976 - val_loss: 0.0616 - val_acc: 0.9819
Epoch 16/20
60000/60000 [=====] - 6s 99us/step - loss: 0.0
095 - acc: 0.9981 - val_loss: 0.0614 - val_acc: 0.9815
Epoch 17/20
60000/60000 [=====] - 6s 99us/step - loss: 0.0
079 - acc: 0.9986 - val_loss: 0.0690 - val_acc: 0.9796
Epoch 18/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
060 - acc: 0.9990 - val_loss: 0.0686 - val_acc: 0.9801
Epoch 19/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
055 - acc: 0.9989 - val_loss: 0.0637 - val_acc: 0.9827
Epoch 20/20
60000/60000 [=====] - 6s 98us/step - loss: 0.0
044 - acc: 0.9992 - val_loss: 0.0731 - val_acc: 0.9789
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

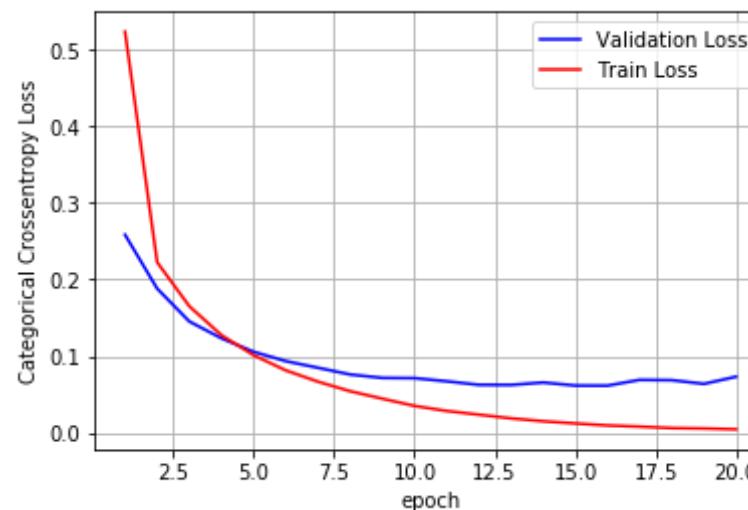
Test score: 0.07310124088092998  
Test accuracy: 0.9789

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
```

```

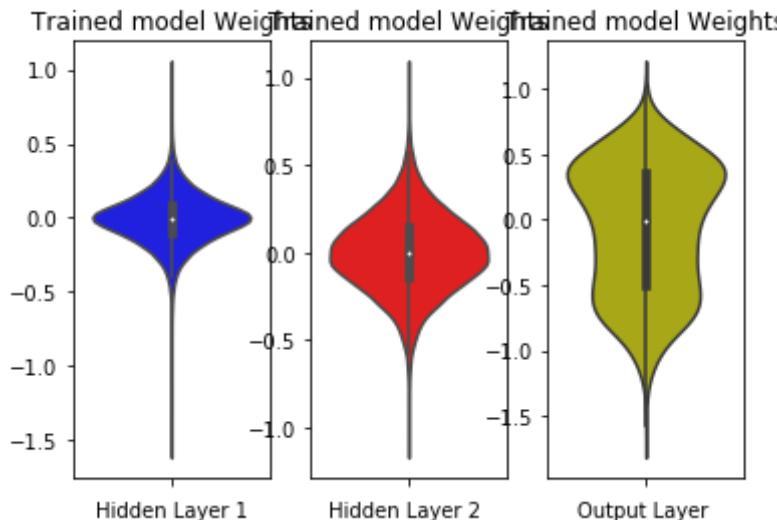
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : sigmoid activation + GradientDescentOptimizer with Relu

## Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)) )

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_18"

Layer (type)	Output Shape	Param #
<hr/>		
dense_48 (Dense)	(None, 501)	393285
dense_49 (Dense)	(None, 101)	50702
dense_50 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nbr_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/20

60000/60000 [=====] - 6s 94us/step - loss: 2.1

894 - acc: 0.3638 - val\_loss: 2.0313 - val\_acc: 0.5796

Epoch 2/20

```
Epoch 2/20
60000/60000 [=====] - 5s 76us/step - loss: 1.8
673 - acc: 0.6337 - val_loss: 1.6789 - val_acc: 0.6953
Epoch 3/20
60000/60000 [=====] - 5s 76us/step - loss: 1.5
238 - acc: 0.7151 - val_loss: 1.3500 - val_acc: 0.7514
Epoch 4/20
60000/60000 [=====] - 5s 76us/step - loss: 1.2
344 - acc: 0.7643 - val_loss: 1.0986 - val_acc: 0.7842
Epoch 5/20
60000/60000 [=====] - 5s 76us/step - loss: 1.0
214 - acc: 0.7951 - val_loss: 0.9202 - val_acc: 0.8119
Epoch 6/20
60000/60000 [=====] - 5s 77us/step - loss: 0.8
719 - acc: 0.8154 - val_loss: 0.7958 - val_acc: 0.8317
Epoch 7/20
60000/60000 [=====] - 5s 77us/step - loss: 0.7
667 - acc: 0.8307 - val_loss: 0.7073 - val_acc: 0.8434
Epoch 8/20
60000/60000 [=====] - 5s 79us/step - loss: 0.6
906 - acc: 0.8420 - val_loss: 0.6416 - val_acc: 0.8551
Epoch 9/20
60000/60000 [=====] - 5s 78us/step - loss: 0.6
336 - acc: 0.8506 - val_loss: 0.5924 - val_acc: 0.8611
Epoch 10/20
60000/60000 [=====] - 5s 76us/step - loss: 0.5
893 - acc: 0.8580 - val_loss: 0.5528 - val_acc: 0.8698
Epoch 11/20
60000/60000 [=====] - 5s 75us/step - loss: 0.5
540 - acc: 0.8649 - val_loss: 0.5210 - val_acc: 0.8731
Epoch 12/20
60000/60000 [=====] - 5s 75us/step - loss: 0.5
252 - acc: 0.8700 - val_loss: 0.4947 - val_acc: 0.8783
Epoch 13/20
60000/60000 [=====] - 5s 75us/step - loss: 0.5
011 - acc: 0.8737 - val_loss: 0.4734 - val_acc: 0.8814
Epoch 14/20
60000/60000 [=====] - 5s 76us/step - loss: 0.4
806 - acc: 0.8773 - val_loss: 0.4538 - val_acc: 0.8855
Epoch 15/20
```

```
60000/60000 [=====] - 5s 76us/step - loss: 0.4
632 - acc: 0.8806 - val_loss: 0.4383 - val_acc: 0.8882
Epoch 16/20
60000/60000 [=====] - 5s 75us/step - loss: 0.4
481 - acc: 0.8832 - val_loss: 0.4243 - val_acc: 0.8923
Epoch 17/20
60000/60000 [=====] - 4s 73us/step - loss: 0.4
349 - acc: 0.8857 - val_loss: 0.4120 - val_acc: 0.8935
Epoch 18/20
60000/60000 [=====] - 4s 74us/step - loss: 0.4
232 - acc: 0.8878 - val_loss: 0.4016 - val_acc: 0.8941
Epoch 19/20
60000/60000 [=====] - 4s 74us/step - loss: 0.4
129 - acc: 0.8894 - val_loss: 0.3918 - val_acc: 0.8969
Epoch 20/20
60000/60000 [=====] - 5s 76us/step - loss: 0.4
038 - acc: 0.8915 - val_loss: 0.3834 - val_acc: 0.8999
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

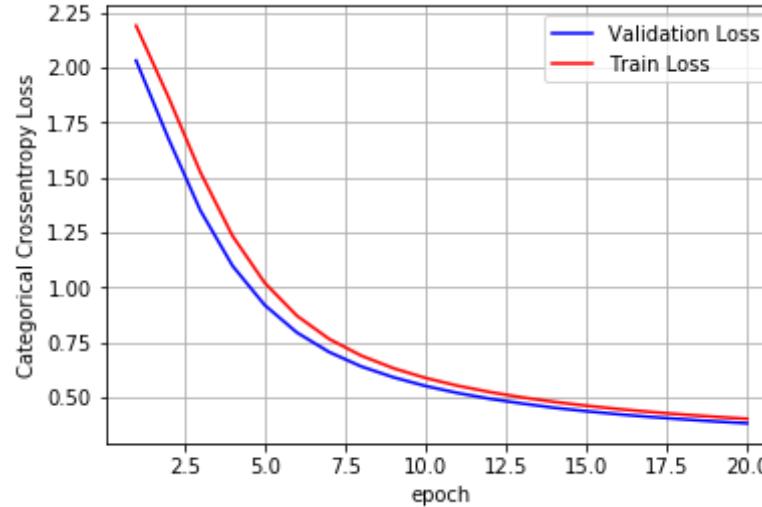
```
Test score: 0.3833876079082489
Test accuracy: 0.8999
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

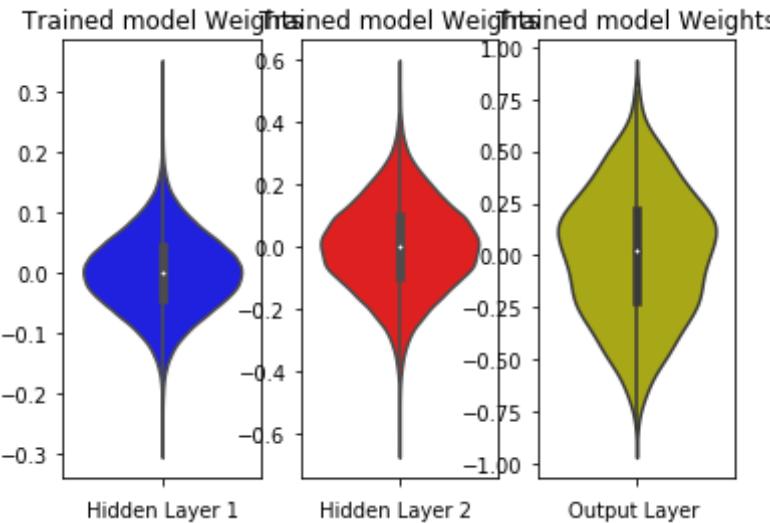
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



## Model 2: Relu activation + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)) )

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_19"

Layer (type)	Output Shape	Param #
dense_51 (Dense)	(None, 501)	393285
dense_52 (Dense)	(None, 101)	50702
dense_53 (Dense)	(None, 10)	1020
Total params:	445,007	
Trainable params:	445,007	
Non-trainable params:	0	

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
2441 - acc: 0.9291 - val_loss: 0.1240 - val_acc: 0.9609
Epoch 2/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
924 - acc: 0.9717 - val_loss: 0.0817 - val_acc: 0.9731
Epoch 3/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
573 - acc: 0.9825 - val_loss: 0.0696 - val_acc: 0.9773
Epoch 4/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
404 - acc: 0.9875 - val_loss: 0.0767 - val_acc: 0.9774
Epoch 5/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
288 - acc: 0.9911 - val_loss: 0.0624 - val_acc: 0.9805
Epoch 6/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
224 - acc: 0.9931 - val_loss: 0.0759 - val_acc: 0.9774
Epoch 7/20
60000/60000 [=====] - 6s 96us/step - loss: 0.0
```

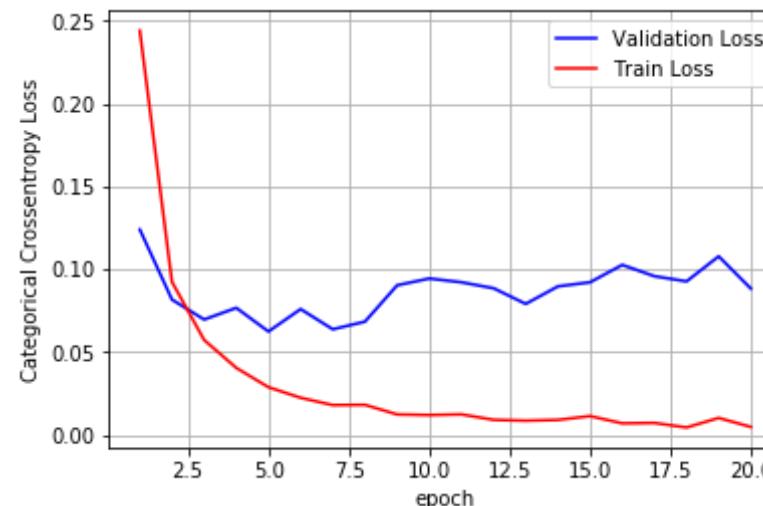
```
180 - acc: 0.9943 - val_loss: 0.0637 - val_acc: 0.9819
Epoch 8/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
181 - acc: 0.9940 - val_loss: 0.0683 - val_acc: 0.9807
Epoch 9/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
123 - acc: 0.9959 - val_loss: 0.0903 - val_acc: 0.9786
Epoch 10/20
60000/60000 [=====] - 6s 94us/step - loss: 0.0
120 - acc: 0.9962 - val_loss: 0.0944 - val_acc: 0.9760
Epoch 11/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
123 - acc: 0.9961 - val_loss: 0.0921 - val_acc: 0.9775
Epoch 12/20
60000/60000 [=====] - 6s 96us/step - loss: 0.0
091 - acc: 0.9968 - val_loss: 0.0885 - val_acc: 0.9800
Epoch 13/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
086 - acc: 0.9972 - val_loss: 0.0791 - val_acc: 0.9823
Epoch 14/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
090 - acc: 0.9968 - val_loss: 0.0896 - val_acc: 0.9798
Epoch 15/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
113 - acc: 0.9962 - val_loss: 0.0921 - val_acc: 0.9792
Epoch 16/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
069 - acc: 0.9975 - val_loss: 0.1026 - val_acc: 0.9788
Epoch 17/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
072 - acc: 0.9977 - val_loss: 0.0958 - val_acc: 0.9798
Epoch 18/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
045 - acc: 0.9986 - val_loss: 0.0927 - val_acc: 0.9817
Epoch 19/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
102 - acc: 0.9969 - val_loss: 0.1078 - val_acc: 0.9779
Epoch 20/20
```

```
60000/60000 [=====] - 6s 92us/step - loss: 0.0  
048 - acc: 0.9987 - val_loss: 0.0883 - val_acc: 0.9809
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.08829486077914112  
Test accuracy: 0.9809
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



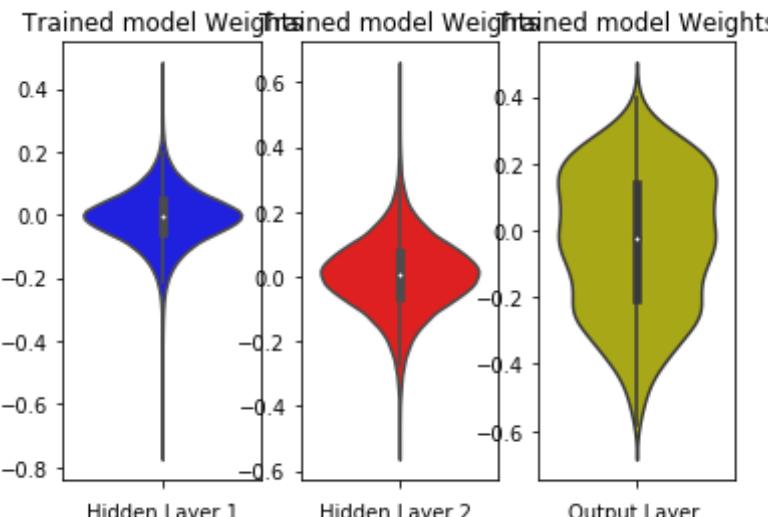
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 2 : relu activation + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_20"

Layer (type)	Output Shape	Param #
<hr/>		
dense_54 (Dense)	(None, 501)	393285
dense_55 (Dense)	(None, 101)	50702
dense_56 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples. validate on 10000 samples

```
train on 60000 samples, validate on 10000 samples

Epoch 1/20
60000/60000 [=====] - 6s 93us/step - loss: 1.0
661 - acc: 0.7526 - val_loss: 0.5005 - val_acc: 0.8796
Epoch 2/20
60000/60000 [=====] - 4s 73us/step - loss: 0.4
386 - acc: 0.8856 - val_loss: 0.3621 - val_acc: 0.9055
Epoch 3/20
60000/60000 [=====] - 4s 74us/step - loss: 0.3
539 - acc: 0.9021 - val_loss: 0.3150 - val_acc: 0.9128
Epoch 4/20
60000/60000 [=====] - 4s 74us/step - loss: 0.3
152 - acc: 0.9108 - val_loss: 0.2869 - val_acc: 0.9207
Epoch 5/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
892 - acc: 0.9177 - val_loss: 0.2665 - val_acc: 0.9251
Epoch 6/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
693 - acc: 0.9234 - val_loss: 0.2515 - val_acc: 0.9314
Epoch 7/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
530 - acc: 0.9290 - val_loss: 0.2383 - val_acc: 0.9347
Epoch 8/20
60000/60000 [=====] - 5s 75us/step - loss: 0.2
388 - acc: 0.9327 - val_loss: 0.2262 - val_acc: 0.9366
Epoch 9/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
263 - acc: 0.9358 - val_loss: 0.2178 - val_acc: 0.9388
Epoch 10/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
155 - acc: 0.9391 - val_loss: 0.2059 - val_acc: 0.9414
Epoch 11/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
053 - acc: 0.9424 - val_loss: 0.1990 - val_acc: 0.9431
Epoch 12/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
961 - acc: 0.9449 - val_loss: 0.1906 - val_acc: 0.9458
Epoch 13/20
60000/60000 [=====] - 5s 75us/step - loss: 0.1
879 - acc: 0.9470 - val_loss: 0.1829 - val_acc: 0.9481
```

```
Epoch 14/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
803 - acc: 0.9490 - val_loss: 0.1767 - val_acc: 0.9487
Epoch 15/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
730 - acc: 0.9507 - val_loss: 0.1710 - val_acc: 0.9509
Epoch 16/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
666 - acc: 0.9527 - val_loss: 0.1643 - val_acc: 0.9525
Epoch 17/20
60000/60000 [=====] - 4s 73us/step - loss: 0.1
605 - acc: 0.9551 - val_loss: 0.1603 - val_acc: 0.9537
Epoch 18/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
549 - acc: 0.9562 - val_loss: 0.1545 - val_acc: 0.9550
Epoch 19/20
60000/60000 [=====] - 4s 73us/step - loss: 0.1
495 - acc: 0.9580 - val_loss: 0.1501 - val_acc: 0.9563
Epoch 20/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
441 - acc: 0.9598 - val_loss: 0.1468 - val_acc: 0.9582
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

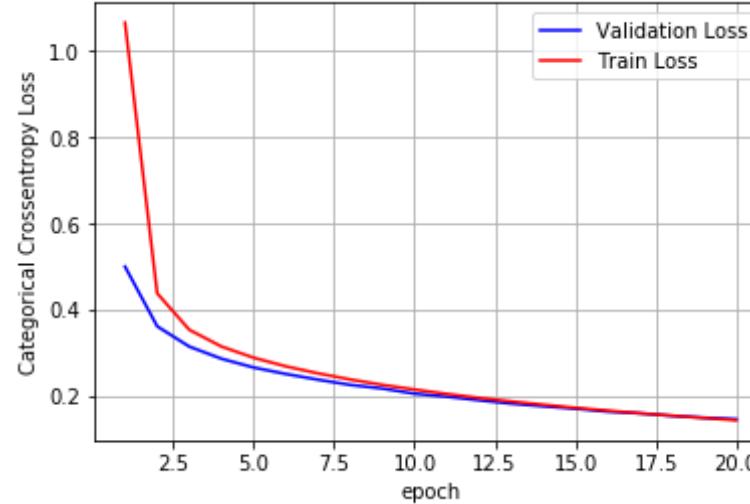
```
Test score: 0.1468325086183846
Test accuracy: 0.9582
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

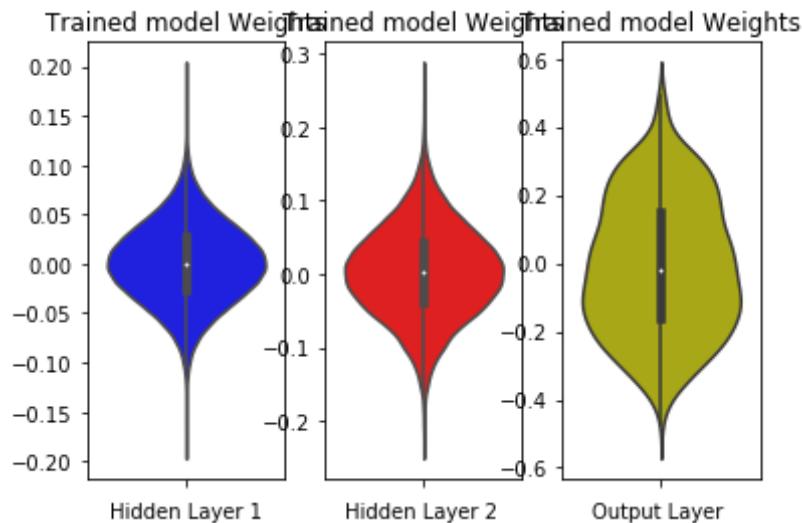
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
```

```
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 1 : relu activation + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)) )

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_21"

Layer (type)	Output Shape	Param #
dense_57 (Dense)	(None, 501)	393285
dense_58 (Dense)	(None, 101)	50702
dense_59 (Dense)	(None, 10)	1020
Total params:	445,007	
Trainable params:	445,007	
Non-trainable params:	0	

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
2340 - acc: 0.9307 - val_loss: 0.1266 - val_acc: 0.9616
Epoch 2/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
861 - acc: 0.9731 - val_loss: 0.0919 - val_acc: 0.9714
Epoch 3/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
558 - acc: 0.9829 - val_loss: 0.0902 - val_acc: 0.9738
Epoch 4/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
381 - acc: 0.9879 - val_loss: 0.0781 - val_acc: 0.9775
Epoch 5/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
285 - acc: 0.9910 - val_loss: 0.0801 - val_acc: 0.9761
Epoch 6/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
202 - acc: 0.9939 - val_loss: 0.0715 - val_acc: 0.9794
Epoch 7/20
60000/60000 [=====] - 6s 94us/step - loss: 0.0
```

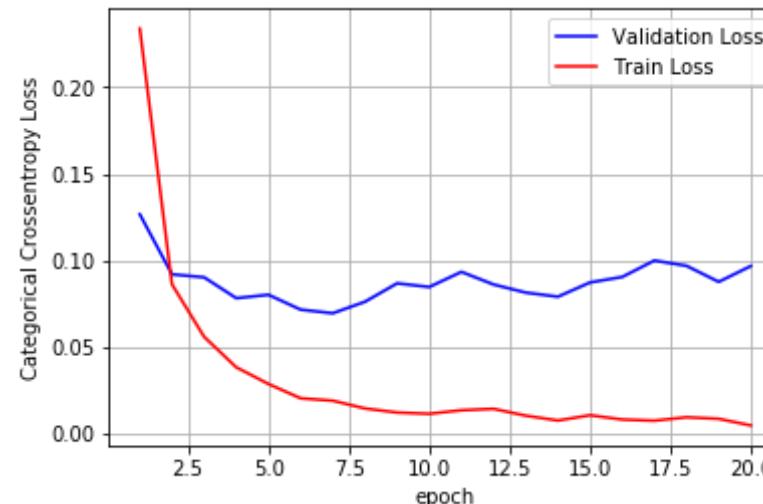
```
188 - acc: 0.9941 - val_loss: 0.0694 - val_acc: 0.9812
Epoch 8/20
60000/60000 [=====] - 6s 95us/step - loss: 0.0
143 - acc: 0.9955 - val_loss: 0.0760 - val_acc: 0.9805
Epoch 9/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
120 - acc: 0.9962 - val_loss: 0.0867 - val_acc: 0.9781
Epoch 10/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
112 - acc: 0.9962 - val_loss: 0.0846 - val_acc: 0.9804
Epoch 11/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
133 - acc: 0.9956 - val_loss: 0.0933 - val_acc: 0.9758
Epoch 12/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
141 - acc: 0.9956 - val_loss: 0.0860 - val_acc: 0.9784
Epoch 13/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
102 - acc: 0.9966 - val_loss: 0.0813 - val_acc: 0.9820
Epoch 14/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
074 - acc: 0.9977 - val_loss: 0.0789 - val_acc: 0.9836
Epoch 15/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
104 - acc: 0.9964 - val_loss: 0.0872 - val_acc: 0.9810
Epoch 16/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
079 - acc: 0.9973 - val_loss: 0.0904 - val_acc: 0.9799
Epoch 17/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
073 - acc: 0.9978 - val_loss: 0.0999 - val_acc: 0.9806
Epoch 18/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
092 - acc: 0.9970 - val_loss: 0.0968 - val_acc: 0.9793
Epoch 19/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
084 - acc: 0.9974 - val_loss: 0.0875 - val_acc: 0.9831
Epoch 20/20
```

```
60000/60000 [=====] - 5s 91us/step - loss: 0.0  
045 - acc: 0.9987 - val_loss: 0.0967 - val_acc: 0.9806
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.09668173986358906  
Test accuracy: 0.9806
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



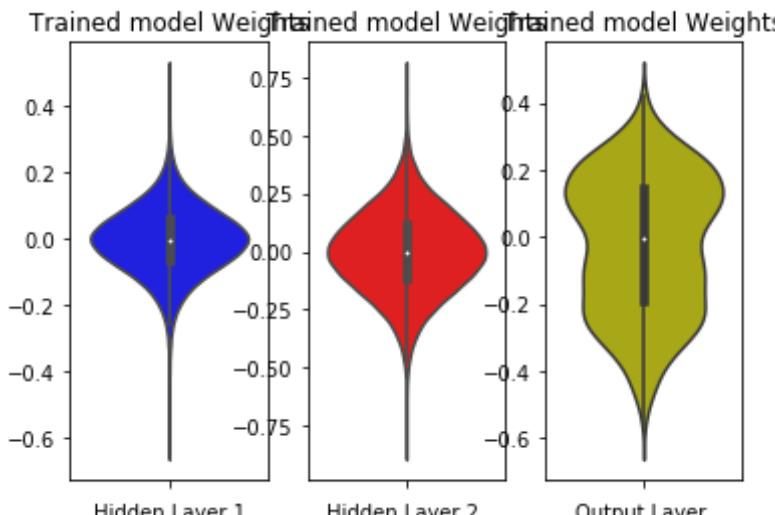
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 2 : relu activation + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)) )

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_22"

Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 501)	393285
dense_61 (Dense)	(None, 101)	50702
dense_62 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20

```
--|-- --|--  
60000/60000 [=====] - 6s 95us/step - loss: 0.7  
519 - acc: 0.7855 - val_loss: 0.3920 - val_acc: 0.8908  
Epoch 2/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.3  
539 - acc: 0.9000 - val_loss: 0.3049 - val_acc: 0.9140  
Epoch 3/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.2  
910 - acc: 0.9184 - val_loss: 0.2666 - val_acc: 0.9249  
Epoch 4/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.2  
567 - acc: 0.9271 - val_loss: 0.2401 - val_acc: 0.9311  
Epoch 5/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.2  
324 - acc: 0.9335 - val_loss: 0.2226 - val_acc: 0.9358  
Epoch 6/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.2  
142 - acc: 0.9392 - val_loss: 0.2076 - val_acc: 0.9399  
Epoch 7/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
994 - acc: 0.9431 - val_loss: 0.1955 - val_acc: 0.9452  
Epoch 8/20  
60000/60000 [=====] - 4s 74us/step - loss: 0.1  
868 - acc: 0.9467 - val_loss: 0.1846 - val_acc: 0.9466  
Epoch 9/20  
60000/60000 [=====] - 4s 75us/step - loss: 0.1  
761 - acc: 0.9501 - val_loss: 0.1776 - val_acc: 0.9494  
Epoch 10/20  
60000/60000 [=====] - 4s 75us/step - loss: 0.1  
665 - acc: 0.9524 - val_loss: 0.1707 - val_acc: 0.9510  
Epoch 11/20  
60000/60000 [=====] - 5s 75us/step - loss: 0.1  
583 - acc: 0.9555 - val_loss: 0.1646 - val_acc: 0.9527  
Epoch 12/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
509 - acc: 0.9575 - val_loss: 0.1573 - val_acc: 0.9544  
Epoch 13/20  
60000/60000 [=====] - 5s 75us/step - loss: 0.1  
440 - acc: 0.9590 - val_loss: 0.1519 - val_acc: 0.9549  
Epoch 14/20
```

```
60000/60000 [=====] - 5s 76us/step - loss: 0.1
380 - acc: 0.9606 - val_loss: 0.1473 - val_acc: 0.9565
Epoch 15/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
321 - acc: 0.9629 - val_loss: 0.1428 - val_acc: 0.9575
Epoch 16/20
60000/60000 [=====] - 5s 75us/step - loss: 0.1
268 - acc: 0.9642 - val_loss: 0.1382 - val_acc: 0.9589
Epoch 17/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
221 - acc: 0.9657 - val_loss: 0.1352 - val_acc: 0.9595
Epoch 18/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
174 - acc: 0.9672 - val_loss: 0.1320 - val_acc: 0.9609
Epoch 19/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
133 - acc: 0.9683 - val_loss: 0.1285 - val_acc: 0.9624
Epoch 20/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
093 - acc: 0.9695 - val_loss: 0.1246 - val_acc: 0.9635
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

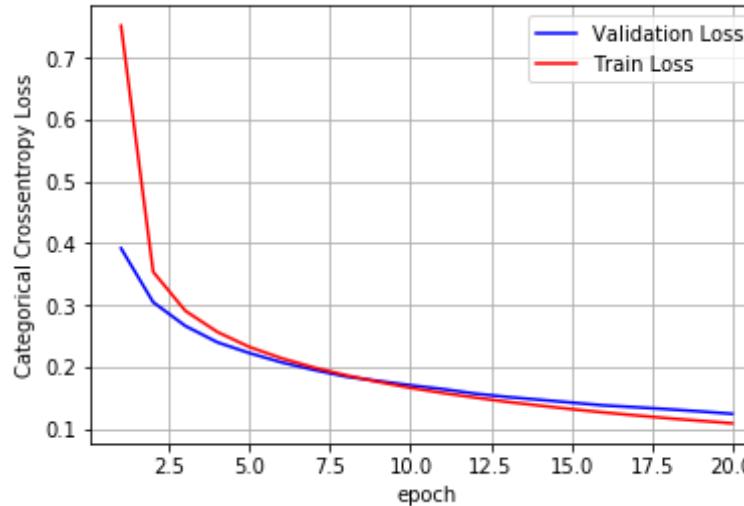
```
Test score: 0.12461129240207375
Test accuracy: 0.9635
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

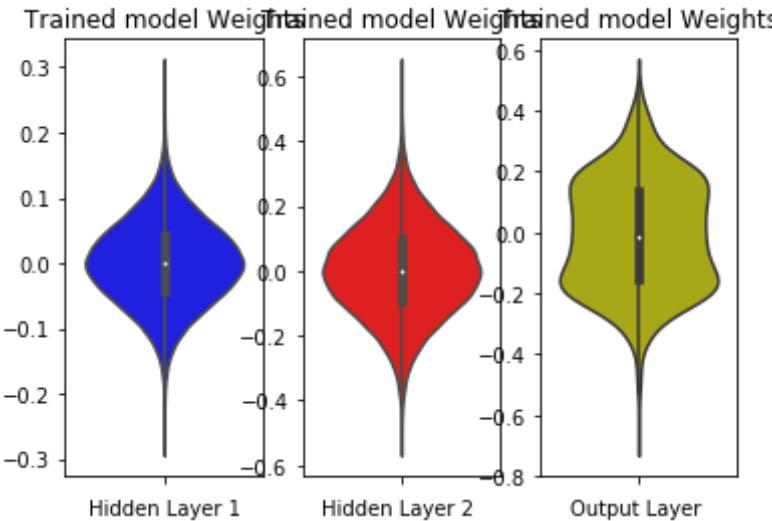
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



### Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.058, seed=None)) )
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_23"

Layer (type)	Output Shape	Param #
dense_63 (Dense)	(None, 501)	393285
batch_normalization_9 (Batch Normalization)	(None, 501)	2004
dense_64 (Dense)	(None, 101)	50702
batch_normalization_10 (Batch Normalization)	(None, 101)	404
dense_65 (Dense)	(None, 10)	1020
<hr/>		
Total params: 447,415		
Trainable params: 446,211		
Non-trainable params: 1,204		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
2758 - acc: 0.9195 - val\_loss: 0.1722 - val\_acc: 0.9514  
Epoch 2/20  
60000/60000 [=====] - 7s 109us/step - loss: 0.  
1511 - acc: 0.9564 - val\_loss: 0.1353 - val\_acc: 0.9599  
Epoch 3/20  
60000/60000 [=====] - 7s 110us/step - loss: 0.  
1095 - acc: 0.9676 - val\_loss: 0.1094 - val\_acc: 0.9668  
Epoch 4/20  
60000/60000 [=====] - 7s 111us/step - loss: 0.  
0829 - acc: 0.9767 - val\_loss: 0.1022 - val\_acc: 0.9676  
Epoch 5/20  
60000/60000 [=====] - 7s 112us/step - loss: 0.

```
0654 - acc: 0.9807 - val_loss: 0.0973 - val_acc: 0.9709
Epoch 6/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
0524 - acc: 0.9845 - val_loss: 0.0808 - val_acc: 0.9754
Epoch 7/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0402 - acc: 0.9876 - val_loss: 0.0815 - val_acc: 0.9748
Epoch 8/20
60000/60000 [=====] - 7s 110us/step - loss: 0.
0330 - acc: 0.9898 - val_loss: 0.0829 - val_acc: 0.9749
Epoch 9/20
60000/60000 [=====] - 7s 111us/step - loss: 0.
0261 - acc: 0.9923 - val_loss: 0.0777 - val_acc: 0.9754
Epoch 10/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
0222 - acc: 0.9934 - val_loss: 0.0728 - val_acc: 0.9789
Epoch 11/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
0190 - acc: 0.9940 - val_loss: 0.0721 - val_acc: 0.9770
Epoch 12/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0172 - acc: 0.9948 - val_loss: 0.0762 - val_acc: 0.9782
Epoch 13/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0144 - acc: 0.9955 - val_loss: 0.0787 - val_acc: 0.9790
Epoch 14/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0129 - acc: 0.9959 - val_loss: 0.0859 - val_acc: 0.9783
Epoch 15/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0118 - acc: 0.9962 - val_loss: 0.0735 - val_acc: 0.9803
Epoch 16/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0115 - acc: 0.9964 - val_loss: 0.0842 - val_acc: 0.9759
Epoch 17/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0110 - acc: 0.9965 - val_loss: 0.0878 - val_acc: 0.9771
Epoch 18/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
```

```
0085 - acc: 0.9974 - val_loss: 0.0829 - val_acc: 0.9779
Epoch 19/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0091 - acc: 0.9972 - val_loss: 0.0827 - val_acc: 0.9776
Epoch 20/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0099 - acc: 0.9968 - val_loss: 0.0829 - val_acc: 0.9782
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

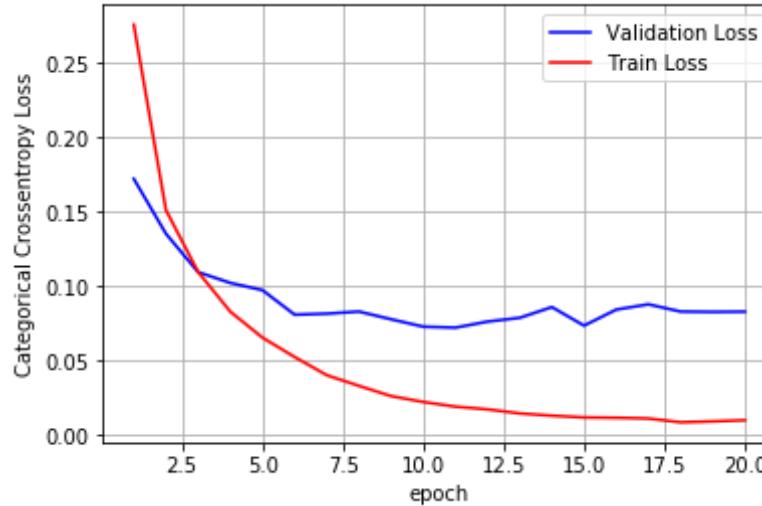
```
Test score: 0.08285859919666254
Test accuracy: 0.9782
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

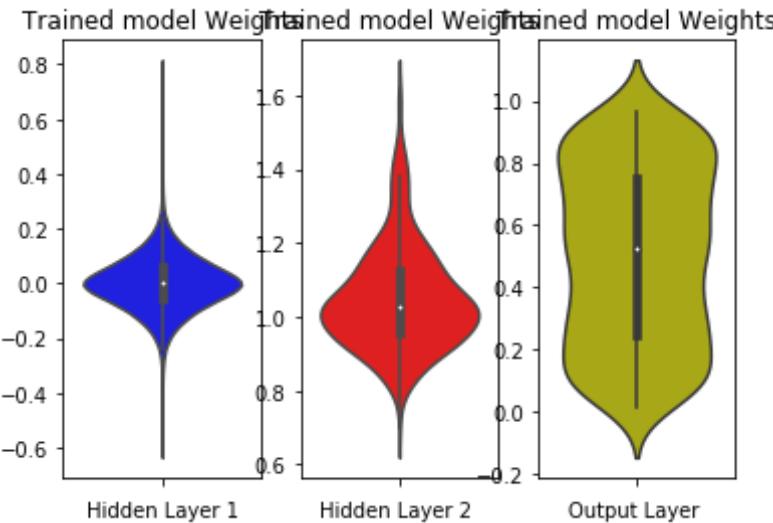
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_24"

Layer (type)	Output Shape	Param #
dense_66 (Dense)	(None, 501)	393285
batch_normalization_11 (Batch Normalization)	(None, 501)	2004
dense_67 (Dense)	(None, 101)	50702
batch_normalization_12 (Batch Normalization)	(None, 101)	404
dense_68 (Dense)	(None, 10)	1020

---

Total params: 447,415  
Trainable params: 446,211  
Non-trainable params: 1,204

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 8s 130us/step - loss: 0.
4947 - acc: 0.8546 - val_loss: 0.3299 - val_acc: 0.9073
Epoch 2/20
60000/60000 [=====] - 6s 93us/step - loss: 0.3
332 - acc: 0.9040 - val_loss: 0.2928 - val_acc: 0.9158
Epoch 3/20
60000/60000 [=====] - 6s 92us/step - loss: 0.3
027 - acc: 0.9129 - val_loss: 0.2747 - val_acc: 0.9202
Epoch 4/20
60000/60000 [=====] - 6s 92us/step - loss: 0.2
816 - acc: 0.9200 - val_loss: 0.2604 - val_acc: 0.9264
Epoch 5/20
60000/60000 [=====] - 6s 92us/step - loss: 0.2
657 - acc: 0.9250 - val_loss: 0.2476 - val_acc: 0.9289
```

```
- -  
Epoch 6/20  
60000/60000 [=====] - 5s 91us/step - loss: 0.2  
517 - acc: 0.9290 - val_loss: 0.2372 - val_acc: 0.9318  
Epoch 7/20  
60000/60000 [=====] - 5s 91us/step - loss: 0.2  
395 - acc: 0.9325 - val_loss: 0.2282 - val_acc: 0.9355  
Epoch 8/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.2  
292 - acc: 0.9356 - val_loss: 0.2203 - val_acc: 0.9380  
Epoch 9/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.2  
199 - acc: 0.9385 - val_loss: 0.2129 - val_acc: 0.9401  
Epoch 10/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.2  
099 - acc: 0.9421 - val_loss: 0.2067 - val_acc: 0.9414  
Epoch 11/20  
60000/60000 [=====] - 5s 92us/step - loss: 0.2  
021 - acc: 0.9440 - val_loss: 0.1989 - val_acc: 0.9445  
Epoch 12/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.1  
950 - acc: 0.9458 - val_loss: 0.1928 - val_acc: 0.9451  
Epoch 13/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.1  
878 - acc: 0.9475 - val_loss: 0.1868 - val_acc: 0.9480  
Epoch 14/20  
60000/60000 [=====] - 6s 95us/step - loss: 0.1  
805 - acc: 0.9504 - val_loss: 0.1840 - val_acc: 0.9476  
Epoch 15/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.1  
756 - acc: 0.9517 - val_loss: 0.1785 - val_acc: 0.9502  
Epoch 16/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.1  
698 - acc: 0.9527 - val_loss: 0.1734 - val_acc: 0.9503  
Epoch 17/20  
60000/60000 [=====] - 6s 94us/step - loss: 0.1  
643 - acc: 0.9548 - val_loss: 0.1697 - val_acc: 0.9507  
Epoch 18/20  
60000/60000 [=====] - 6s 93us/step - loss: 0.1  
583 - acc: 0.9566 - val_loss: 0.1652 - val_acc: 0.9536
```

```
Epoch 19/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
550 - acc: 0.9570 - val_loss: 0.1614 - val_acc: 0.9538
Epoch 20/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
505 - acc: 0.9590 - val_loss: 0.1565 - val_acc: 0.9555
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

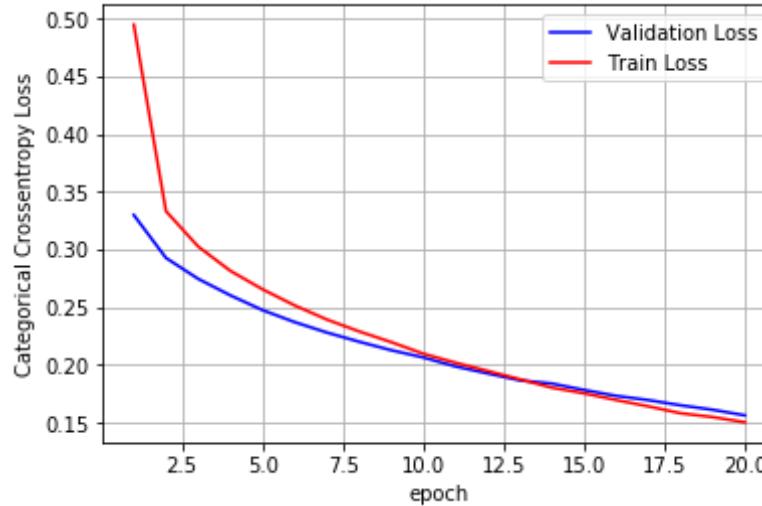
```
Test score: 0.15652678999304773
Test accuracy: 0.9555
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

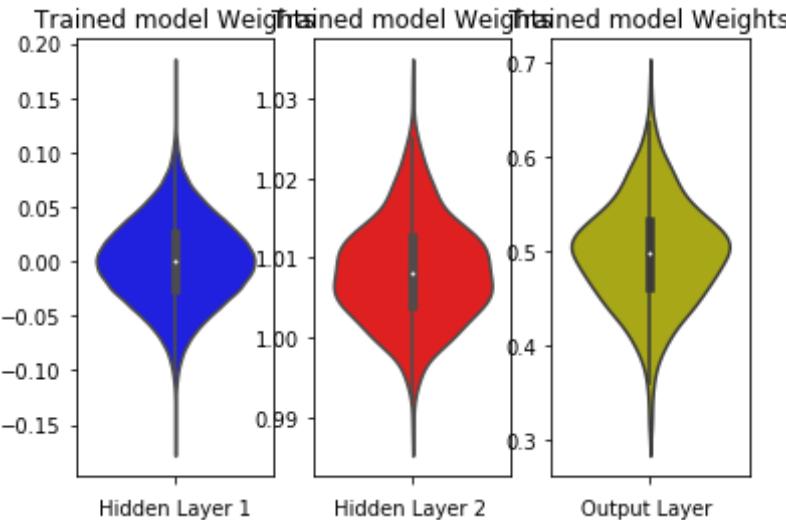
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.140, seed=None)) )
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_25"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```
=====
dense_69 (Dense)           (None, 501)          393285
batch_normalization_13 (Batch Normalization) (None, 501)      2004
dense_70 (Dense)           (None, 101)          50702
batch_normalization_14 (Batch Normalization) (None, 101)      404
dense_71 (Dense)           (None, 10)           1020
=====
Total params: 447,415
Trainable params: 446,211
Non-trainable params: 1,204
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 150us/step - loss: 0.
2760 - acc: 0.9192 - val_loss: 0.1669 - val_acc: 0.9509
Epoch 2/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
1379 - acc: 0.9608 - val_loss: 0.1314 - val_acc: 0.9622
Epoch 3/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0949 - acc: 0.9726 - val_loss: 0.1107 - val_acc: 0.9681
Epoch 4/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0700 - acc: 0.9797 - val_loss: 0.0947 - val_acc: 0.9712
Epoch 5/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0520 - acc: 0.9850 - val_loss: 0.0927 - val_acc: 0.9708
Epoch 6/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
```

```
0386 - acc: 0.9893 - val_loss: 0.0802 - val_acc: 0.9750
Epoch 7/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0310 - acc: 0.9908 - val_loss: 0.0809 - val_acc: 0.9763
Epoch 8/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0241 - acc: 0.9934 - val_loss: 0.0801 - val_acc: 0.9760
Epoch 9/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0189 - acc: 0.9946 - val_loss: 0.0841 - val_acc: 0.9740
Epoch 10/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
0172 - acc: 0.9948 - val_loss: 0.0831 - val_acc: 0.9763
Epoch 11/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0142 - acc: 0.9958 - val_loss: 0.0900 - val_acc: 0.9754
Epoch 12/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0127 - acc: 0.9963 - val_loss: 0.0868 - val_acc: 0.9761
Epoch 13/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
0095 - acc: 0.9974 - val_loss: 0.0794 - val_acc: 0.9771
Epoch 14/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
0106 - acc: 0.9969 - val_loss: 0.0885 - val_acc: 0.9762
Epoch 15/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0120 - acc: 0.9960 - val_loss: 0.0816 - val_acc: 0.9781
Epoch 16/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0096 - acc: 0.9970 - val_loss: 0.0903 - val_acc: 0.9759
Epoch 17/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0085 - acc: 0.9973 - val_loss: 0.0854 - val_acc: 0.9772
Epoch 18/20
60000/60000 [=====] - 7s 119us/step - loss: 0.
0068 - acc: 0.9980 - val_loss: 0.0725 - val_acc: 0.9816
Epoch 19/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
```

```
0087 - acc: 0.9971 - val_loss: 0.0828 - val_acc: 0.9789
Epoch 20/20
60000/60000 [=====] - 7s 119us/step - loss: 0.
0101 - acc: 0.9965 - val_loss: 0.0888 - val_acc: 0.9780
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

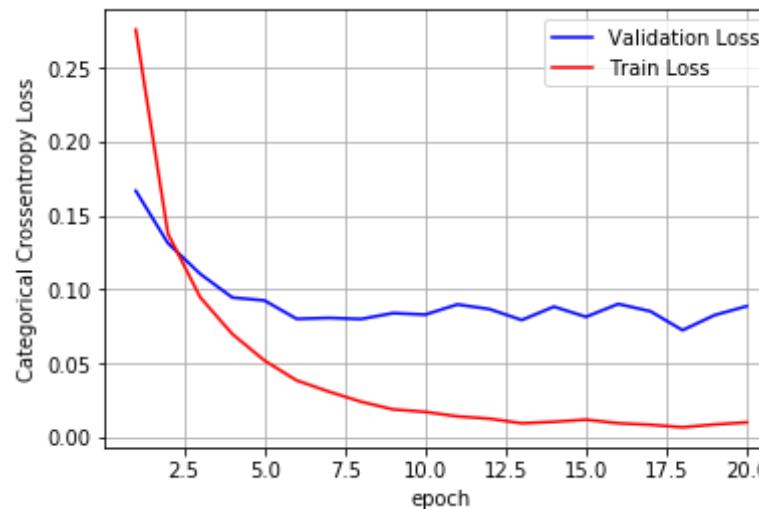
```
Test score: 0.08880745201615282
Test accuracy: 0.978
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



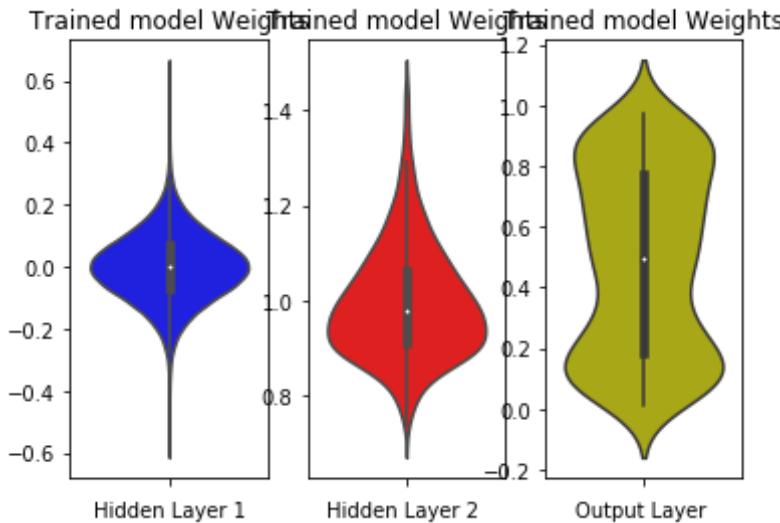
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.140, seed=None) ))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_26"

Layer (type)	Output Shape	Param #
<hr/>		

dense_72 (Dense)	(None, 501)	393285
batch_normalization_15 (Batch Normalization)	(None, 501)	2004
dense_73 (Dense)	(None, 101)	50702
batch_normalization_16 (Batch Normalization)	(None, 101)	404
dense_74 (Dense)	(None, 10)	1020
<hr/>		
Total params: 447,415		
Trainable params: 446,211		
Non-trainable params: 1,204		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 8s 130us/step - loss: 0.
6637 - acc: 0.8006 - val_loss: 0.3908 - val_acc: 0.8845
Epoch 2/20
60000/60000 [=====] - 6s 95us/step - loss: 0.3
756 - acc: 0.8913 - val_loss: 0.3243 - val_acc: 0.9038
Epoch 3/20
60000/60000 [=====] - 6s 95us/step - loss: 0.3
242 - acc: 0.9058 - val_loss: 0.2932 - val_acc: 0.9136
Epoch 4/20
60000/60000 [=====] - 6s 96us/step - loss: 0.2
953 - acc: 0.9141 - val_loss: 0.2705 - val_acc: 0.9213
Epoch 5/20
60000/60000 [=====] - 6s 94us/step - loss: 0.2
731 - acc: 0.9211 - val_loss: 0.2548 - val_acc: 0.9253
Epoch 6/20
60000/60000 [=====] - 6s 95us/step - loss: 0.2
```

```
566 - acc: 0.9257 - val_loss: 0.2433 - val_acc: 0.9294
Epoch 7/20
60000/60000 [=====] - 6s 95us/step - loss: 0.2
419 - acc: 0.9309 - val_loss: 0.2336 - val_acc: 0.9333
Epoch 8/20
60000/60000 [=====] - 6s 96us/step - loss: 0.2
296 - acc: 0.9345 - val_loss: 0.2243 - val_acc: 0.9356
Epoch 9/20
60000/60000 [=====] - 6s 98us/step - loss: 0.2
186 - acc: 0.9370 - val_loss: 0.2163 - val_acc: 0.9385
Epoch 10/20
60000/60000 [=====] - 6s 93us/step - loss: 0.2
091 - acc: 0.9404 - val_loss: 0.2086 - val_acc: 0.9394
Epoch 11/20
60000/60000 [=====] - 6s 92us/step - loss: 0.2
012 - acc: 0.9425 - val_loss: 0.2021 - val_acc: 0.9423
Epoch 12/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
936 - acc: 0.9448 - val_loss: 0.1969 - val_acc: 0.9436
Epoch 13/20
60000/60000 [=====] - 6s 94us/step - loss: 0.1
856 - acc: 0.9470 - val_loss: 0.1914 - val_acc: 0.9446
Epoch 14/20
60000/60000 [=====] - 6s 94us/step - loss: 0.1
790 - acc: 0.9500 - val_loss: 0.1868 - val_acc: 0.9467
Epoch 15/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
730 - acc: 0.9515 - val_loss: 0.1818 - val_acc: 0.9482
Epoch 16/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
662 - acc: 0.9537 - val_loss: 0.1781 - val_acc: 0.9477
Epoch 17/20
60000/60000 [=====] - 6s 94us/step - loss: 0.1
617 - acc: 0.9550 - val_loss: 0.1741 - val_acc: 0.9486
Epoch 18/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
563 - acc: 0.9567 - val_loss: 0.1696 - val_acc: 0.9505
Epoch 19/20
60000/60000 [=====] - 6s 92us/step - loss: 0.1
```

```
520 - acc: 0.9572 - val_loss: 0.1666 - val_acc: 0.9520
Epoch 20/20
60000/60000 [=====] - 6s 93us/step - loss: 0.1
475 - acc: 0.9583 - val_loss: 0.1634 - val_acc: 0.9531
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

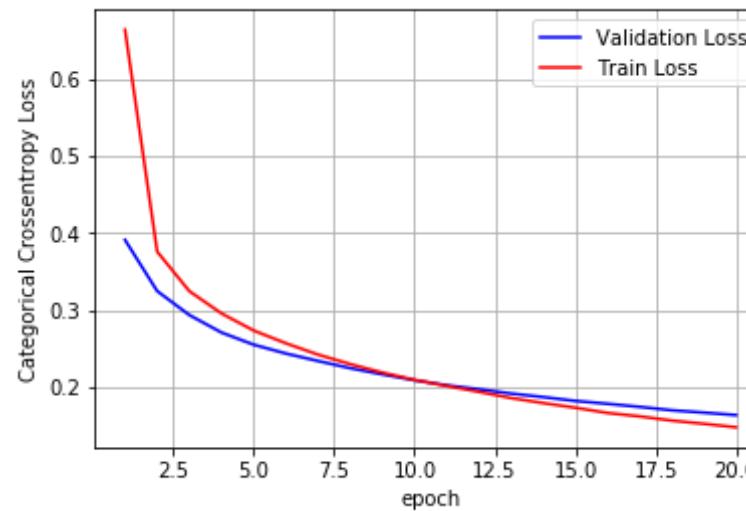
```
Test score: 0.1633817608937621
Test accuracy: 0.9531
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



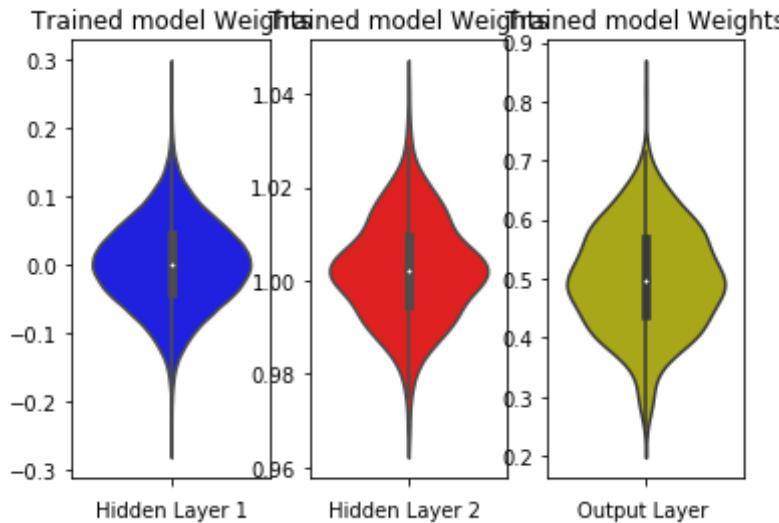
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 4: relu activation with Dropout + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_97"

Layer (type)	Output Shape	Param #
dense_381 (Dense)	(None, 501)	393285
dropout_92 (Dropout)	(None, 501)	0
dense_382 (Dense)	(None, 101)	50702
dropout_93 (Dropout)	(None, 101)	0
dense_383 (Dense)	(None, 10)	1020

---

Total params: 445,007  
Trainable params: 445,007  
Non-trainable params: 0

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 19s 319us/step - loss: 0.4638 - acc: 0.8602 - val_loss: 0.1439 - val_acc: 0.9571
Epoch 2/20
60000/60000 [=====] - 8s 141us/step - loss: 0.2088 - acc: 0.9400 - val_loss: 0.1112 - val_acc: 0.9649
Epoch 3/20
60000/60000 [=====] - 8s 131us/step - loss: 0.1623 - acc: 0.9535 - val_loss: 0.0915 - val_acc: 0.9717
Epoch 4/20
60000/60000 [=====] - 8s 133us/step - loss: 0.1363 - acc: 0.9607 - val_loss: 0.0827 - val_acc: 0.9747
Epoch 5/20
60000/60000 [=====] - 8s 130us/step - loss: 0.1213 - acc: 0.9652 - val_loss: 0.0799 - val_acc: 0.9758
Epoch 6/20
```

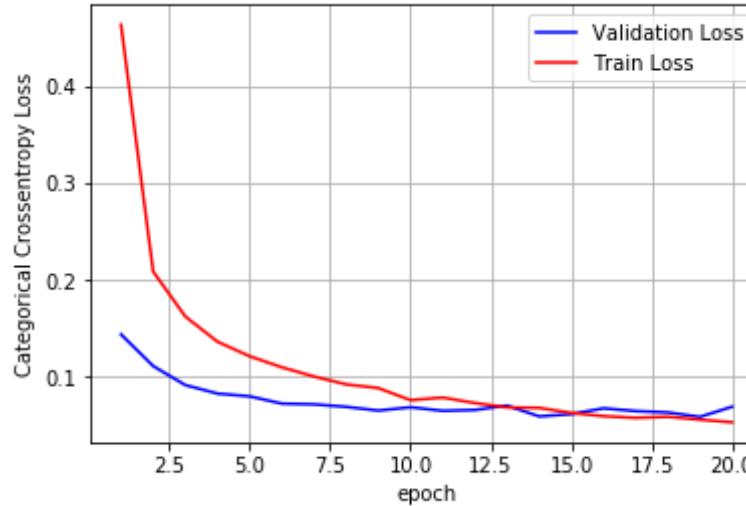
```
60000/60000 [=====] - 8s 132us/step - loss: 0.  
1099 - acc: 0.9674 - val_loss: 0.0724 - val_acc: 0.9788  
Epoch 7/20  
60000/60000 [=====] - 8s 133us/step - loss: 0.  
1004 - acc: 0.9708 - val_loss: 0.0716 - val_acc: 0.9789  
Epoch 8/20  
60000/60000 [=====] - 8s 132us/step - loss: 0.  
0922 - acc: 0.9731 - val_loss: 0.0690 - val_acc: 0.9807  
Epoch 9/20  
60000/60000 [=====] - 8s 132us/step - loss: 0.  
0884 - acc: 0.9742 - val_loss: 0.0652 - val_acc: 0.9808  
Epoch 10/20  
60000/60000 [=====] - 8s 133us/step - loss: 0.  
0759 - acc: 0.9767 - val_loss: 0.0686 - val_acc: 0.9818  
Epoch 11/20  
60000/60000 [=====] - 8s 130us/step - loss: 0.  
0786 - acc: 0.9767 - val_loss: 0.0649 - val_acc: 0.9819  
Epoch 12/20  
60000/60000 [=====] - 8s 130us/step - loss: 0.  
0730 - acc: 0.9778 - val_loss: 0.0658 - val_acc: 0.9816  
Epoch 13/20  
60000/60000 [=====] - 8s 134us/step - loss: 0.  
0684 - acc: 0.9795 - val_loss: 0.0704 - val_acc: 0.9815  
Epoch 14/20  
60000/60000 [=====] - 8s 136us/step - loss: 0.  
0680 - acc: 0.9797 - val_loss: 0.0592 - val_acc: 0.9836  
Epoch 15/20  
60000/60000 [=====] - 8s 134us/step - loss: 0.  
0628 - acc: 0.9812 - val_loss: 0.0615 - val_acc: 0.9833  
Epoch 16/20  
60000/60000 [=====] - 8s 134us/step - loss: 0.  
0594 - acc: 0.9816 - val_loss: 0.0675 - val_acc: 0.9813  
Epoch 17/20  
60000/60000 [=====] - 8s 132us/step - loss: 0.  
0576 - acc: 0.9822 - val_loss: 0.0647 - val_acc: 0.9829  
Epoch 18/20  
60000/60000 [=====] - 8s 133us/step - loss: 0.  
0588 - acc: 0.9818 - val_loss: 0.0633 - val_acc: 0.9822  
Epoch 19/20
```

```
60000/60000 [=====] - 8s 131us/step - loss: 0.  
0557 - acc: 0.9824 - val_loss: 0.0586 - val_acc: 0.9835  
Epoch 20/20  
60000/60000 [=====] - 8s 132us/step - loss: 0.  
0531 - acc: 0.9839 - val_loss: 0.0691 - val_acc: 0.9815
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.06914280058416516  
Test accuracy: 0.9815
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

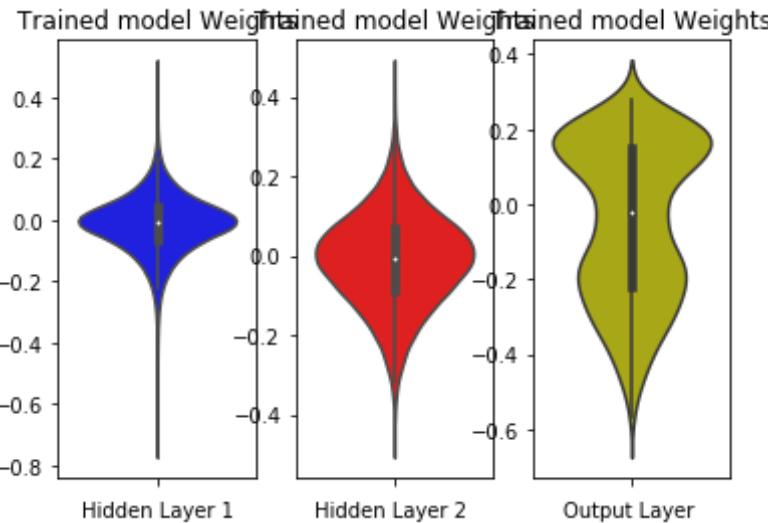
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



#### Model 4: relu activation with Dropout + GradientDescentOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_29"

Layer (type)	Output Shape	Param #
dense_79 (Dense)	(None, 501)	393285
dropout_3 (Dropout)	(None, 501)	0
dense_80 (Dense)	(None, 101)	50702
dropout_4 (Dropout)	(None, 101)	0
dense_81 (Dense)	(None, 10)	1020

Total params: 445,007  
Trainable params: 445,007  
Non-trainable params: 0

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 7s 123us/step - loss: 1.  
5733 - acc: 0.4800 - val\_loss: 0.7011 - val\_acc: 0.8353  
Epoch 2/20  
60000/60000 [=====] - 5s 88us/step - loss: 0.8  
652 - acc: 0.7283 - val\_loss: 0.4606 - val\_acc: 0.8812  
Epoch 3/20  
60000/60000 [=====] - 5s 87us/step - loss: 0.6  
788 - acc: 0.7924 - val\_loss: 0.3780 - val\_acc: 0.8968  
Epoch 4/20  
60000/60000 [=====] - 5s 87us/step - loss: 0.5  
825 - acc: 0.8250 - val\_loss: 0.3346 - val\_acc: 0.9066  
Epoch 5/20  
60000/60000 [=====] - 5s 891us/step - loss: 0.5

```
230 - acc: 0.8443 - val_loss: 0.3061 - val_acc: 0.9135
Epoch 6/20
60000/60000 [=====] - 6s 92us/step - loss: 0.4
789 - acc: 0.8579 - val_loss: 0.2844 - val_acc: 0.9185
Epoch 7/20
60000/60000 [=====] - 5s 89us/step - loss: 0.4
452 - acc: 0.8700 - val_loss: 0.2661 - val_acc: 0.9226
Epoch 8/20
60000/60000 [=====] - 5s 90us/step - loss: 0.4
228 - acc: 0.8772 - val_loss: 0.2518 - val_acc: 0.9268
Epoch 9/20
60000/60000 [=====] - 5s 90us/step - loss: 0.3
962 - acc: 0.8852 - val_loss: 0.2388 - val_acc: 0.9300
Epoch 10/20
60000/60000 [=====] - 5s 91us/step - loss: 0.3
795 - acc: 0.8899 - val_loss: 0.2287 - val_acc: 0.9328
Epoch 11/20
60000/60000 [=====] - 5s 89us/step - loss: 0.3
594 - acc: 0.8956 - val_loss: 0.2206 - val_acc: 0.9347
Epoch 12/20
60000/60000 [=====] - 5s 88us/step - loss: 0.3
441 - acc: 0.9007 - val_loss: 0.2117 - val_acc: 0.9374
Epoch 13/20
60000/60000 [=====] - 5s 90us/step - loss: 0.3
343 - acc: 0.9040 - val_loss: 0.2049 - val_acc: 0.9382
Epoch 14/20
60000/60000 [=====] - 5s 90us/step - loss: 0.3
223 - acc: 0.9075 - val_loss: 0.1972 - val_acc: 0.9407
Epoch 15/20
60000/60000 [=====] - 5s 88us/step - loss: 0.3
086 - acc: 0.9102 - val_loss: 0.1907 - val_acc: 0.9429
Epoch 16/20
60000/60000 [=====] - 5s 90us/step - loss: 0.2
993 - acc: 0.9134 - val_loss: 0.1850 - val_acc: 0.9457
Epoch 17/20
60000/60000 [=====] - 5s 91us/step - loss: 0.2
927 - acc: 0.9163 - val_loss: 0.1790 - val_acc: 0.9467
Epoch 18/20
60000/60000 [=====] - 5s 89us/step - loss: 0.2
```

```
827 - acc: 0.9172 - val_loss: 0.1747 - val_acc: 0.9488
Epoch 19/20
60000/60000 [=====] - 5s 88us/step - loss: 0.2
750 - acc: 0.9205 - val_loss: 0.1693 - val_acc: 0.9502
Epoch 20/20
60000/60000 [=====] - 5s 88us/step - loss: 0.2
707 - acc: 0.9221 - val_loss: 0.1651 - val_acc: 0.9512
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

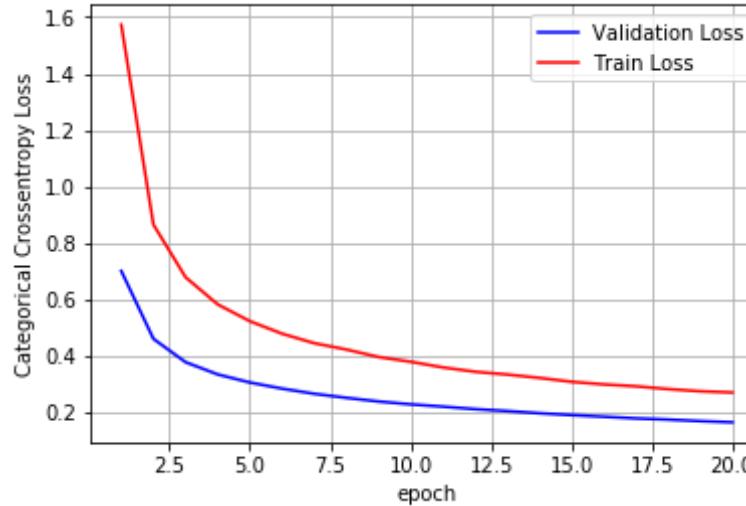
```
Test score: 0.16511225905790924
Test accuracy: 0.9512
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

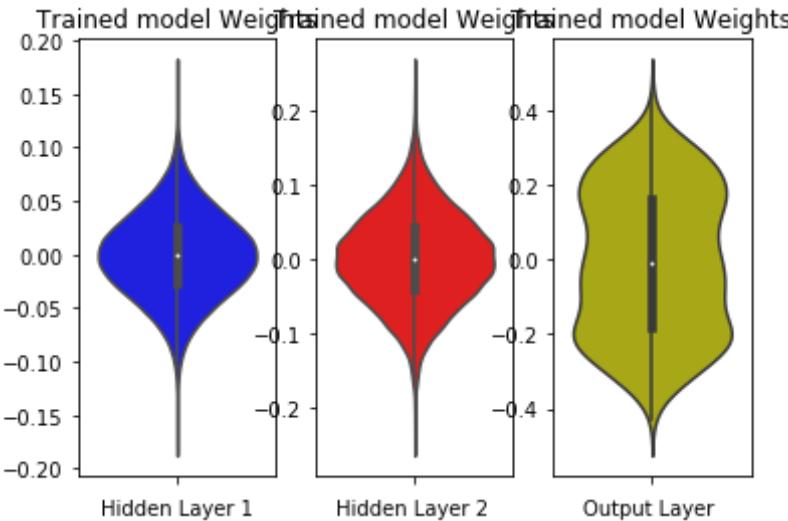
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



Model 4: relu activation with Dropout + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_30"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```

=====
dense_82 (Dense)           (None, 501)          393285
dropout_5 (Dropout)         (None, 501)          0
dense_83 (Dense)           (None, 101)          50702
dropout_6 (Dropout)         (None, 101)          0
dense_84 (Dense)           (None, 10)           1020
=====
Total params: 445,007
Trainable params: 445,007
Non-trainable params: 0

```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 144us/step - loss: 0.
6131 - acc: 0.8089 - val_loss: 0.1817 - val_acc: 0.9450
Epoch 2/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
2652 - acc: 0.9215 - val_loss: 0.1312 - val_acc: 0.9595
Epoch 3/20
60000/60000 [=====] - 6s 108us/step - loss: 0.
2044 - acc: 0.9403 - val_loss: 0.1069 - val_acc: 0.9672
Epoch 4/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
1690 - acc: 0.9506 - val_loss: 0.0954 - val_acc: 0.9714
Epoch 5/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
1521 - acc: 0.9559 - val_loss: 0.0826 - val_acc: 0.9768
Epoch 6/20
60000/60000 [=====] - 6s 107us/step - loss: 0.
```

```
1344 - acc: 0.9603 - val_loss: 0.0822 - val_acc: 0.9764
Epoch 7/20
60000/60000 [=====] - 6s 108us/step - loss: 0.
1214 - acc: 0.9642 - val_loss: 0.0844 - val_acc: 0.9763
Epoch 8/20
60000/60000 [=====] - 6s 107us/step - loss: 0.
1140 - acc: 0.9664 - val_loss: 0.0815 - val_acc: 0.9776
Epoch 9/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
1021 - acc: 0.9691 - val_loss: 0.0741 - val_acc: 0.9795
Epoch 10/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0945 - acc: 0.9718 - val_loss: 0.0707 - val_acc: 0.9782
Epoch 11/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0880 - acc: 0.9737 - val_loss: 0.0747 - val_acc: 0.9803
Epoch 12/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0845 - acc: 0.9751 - val_loss: 0.0668 - val_acc: 0.9812
Epoch 13/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0790 - acc: 0.9757 - val_loss: 0.0719 - val_acc: 0.9798
Epoch 14/20
60000/60000 [=====] - 7s 110us/step - loss: 0.
0775 - acc: 0.9764 - val_loss: 0.0734 - val_acc: 0.9812
Epoch 15/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0709 - acc: 0.9787 - val_loss: 0.0729 - val_acc: 0.9798
Epoch 16/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0722 - acc: 0.9775 - val_loss: 0.0700 - val_acc: 0.9814
Epoch 17/20
60000/60000 [=====] - 7s 109us/step - loss: 0.
0703 - acc: 0.9789 - val_loss: 0.0693 - val_acc: 0.9826
Epoch 18/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
0621 - acc: 0.9812 - val_loss: 0.0741 - val_acc: 0.9816
Epoch 19/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
```

```
0617 - acc: 0.9813 - val_loss: 0.0712 - val_acc: 0.9816
Epoch 20/20
60000/60000 [=====] - 7s 110us/step - loss: 0.
0622 - acc: 0.9815 - val_loss: 0.0694 - val_acc: 0.9827
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

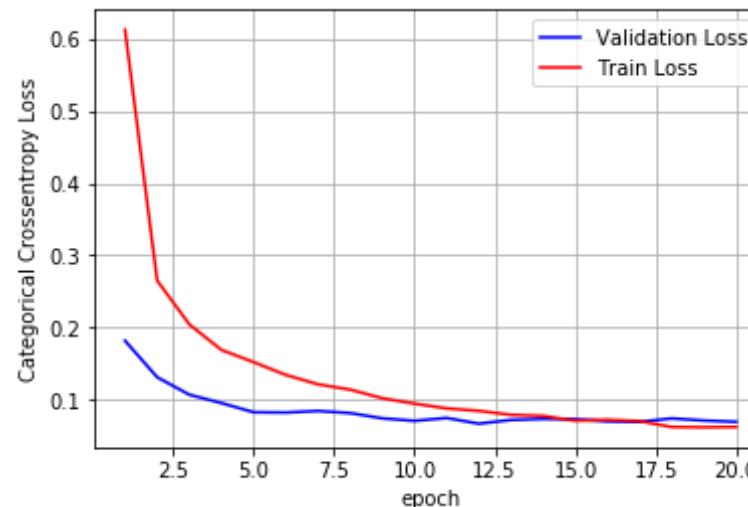
```
Test score: 0.06936063391418429
Test accuracy: 0.9827
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



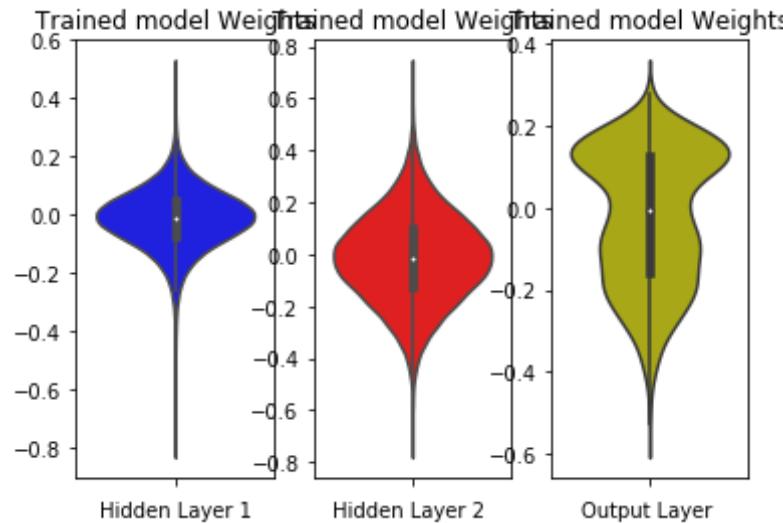
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 4: relu activation with Dropout + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='relu'))

model_1.summary()
```

Model: "sequential\_31"

Layer (type)	Output Shape	Param #
<hr/>		

dense_85 (Dense)	(None, 501)	393285
dropout_7 (Dropout)	(None, 501)	0
dense_86 (Dense)	(None, 101)	50702
dropout_8 (Dropout)	(None, 101)	0
dense_87 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 8s 127us/step - loss: 2.
4836 - acc: 0.1349 - val_loss: 2.1475 - val_acc: 0.1301
Epoch 2/20
60000/60000 [=====] - 5s 91us/step - loss: 2.1
063 - acc: 0.2343 - val_loss: 1.8386 - val_acc: 0.6029
Epoch 3/20
60000/60000 [=====] - 5s 91us/step - loss: 1.8
665 - acc: 0.4130 - val_loss: 1.5459 - val_acc: 0.6107
Epoch 4/20
60000/60000 [=====] - 6s 93us/step - loss: 1.6
604 - acc: 0.5121 - val_loss: 1.2064 - val_acc: 0.7669
Epoch 5/20
60000/60000 [=====] - 5s 91us/step - loss: 1.9
830 - acc: 0.2503 - val_loss: 1.6435 - val_acc: 0.5173
Epoch 6/20
60000/60000 [=====] - 5s 91us/step - loss: 1.7
051 - acc: 0.4325 - val_loss: 1.4205 - val_acc: 0.4527
```

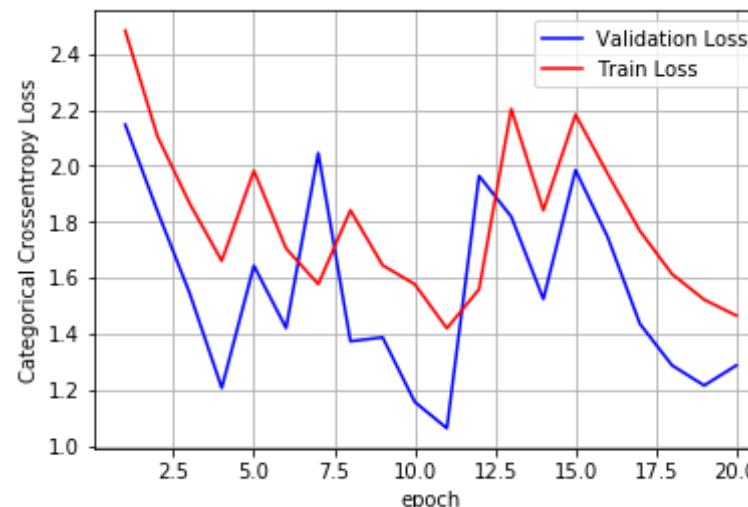
```
- -  
Epoch 7/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.5  
773 - acc: 0.5022 - val_loss: 2.0462 - val_acc: 0.1000  
Epoch 8/20  
60000/60000 [=====] - 6s 93us/step - loss: 1.8  
416 - acc: 0.3090 - val_loss: 1.3732 - val_acc: 0.6522  
Epoch 9/20  
60000/60000 [=====] - 6s 92us/step - loss: 1.6  
453 - acc: 0.4453 - val_loss: 1.3872 - val_acc: 0.4796  
Epoch 10/20  
60000/60000 [=====] - 6s 93us/step - loss: 1.5  
771 - acc: 0.4732 - val_loss: 1.1565 - val_acc: 0.7442  
Epoch 11/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.4  
196 - acc: 0.5886 - val_loss: 1.0621 - val_acc: 0.7410  
Epoch 12/20  
60000/60000 [=====] - 6s 92us/step - loss: 1.5  
589 - acc: 0.5573 - val_loss: 1.9643 - val_acc: 0.0920  
Epoch 13/20  
60000/60000 [=====] - 6s 93us/step - loss: 2.2  
040 - acc: 0.1306 - val_loss: 1.8192 - val_acc: 0.1212  
Epoch 14/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.8  
417 - acc: 0.2398 - val_loss: 1.5240 - val_acc: 0.4515  
Epoch 15/20  
60000/60000 [=====] - 5s 91us/step - loss: 2.1  
836 - acc: 0.1138 - val_loss: 1.9853 - val_acc: 0.1003  
Epoch 16/20  
60000/60000 [=====] - 6s 93us/step - loss: 1.9  
718 - acc: 0.1395 - val_loss: 1.7452 - val_acc: 0.1468  
Epoch 17/20  
60000/60000 [=====] - 5s 90us/step - loss: 1.7  
680 - acc: 0.2767 - val_loss: 1.4358 - val_acc: 0.5677  
Epoch 18/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.6  
138 - acc: 0.4260 - val_loss: 1.2872 - val_acc: 0.6961  
Epoch 19/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.5  
220 - acc: 0.4956 - val_loss: 1.2151 - val_acc: 0.7132
```

```
Epoch 20/20  
60000/60000 [=====] - 5s 91us/step - loss: 1.4  
650 - acc: 0.5422 - val_loss: 1.2870 - val_acc: 0.6801
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 1.2869609331130982  
Test accuracy: 0.6801
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



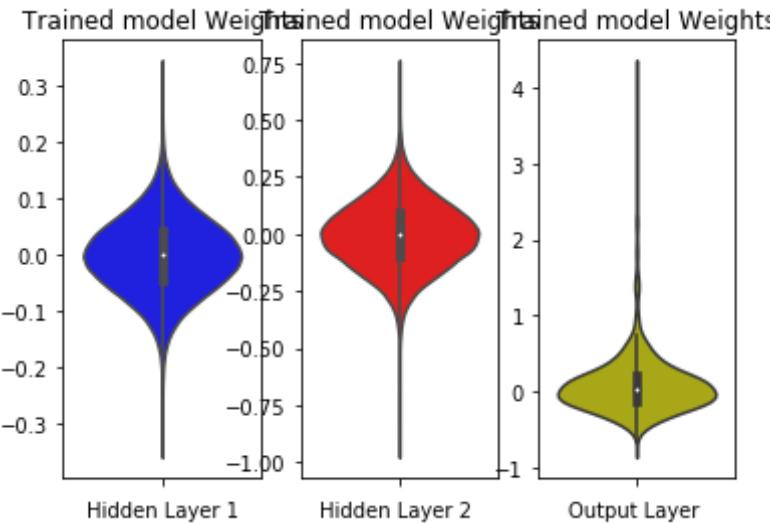
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 5: relu activation with BatchNormalization + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_36"

Layer (type)	Output Shape	Param #
dense_100 (Dense)	(None, 501)	393285
batch_normalization_23 (Batch Normalization)	(None, 501)	2004
dense_101 (Dense)	(None, 101)	50702
batch_normalization_24 (Batch Normalization)	(None, 101)	404
dense_102 (Dense)	(None, 10)	1020
<hr/>		
Total params: 447,415		
Trainable params: 446,211		
Non-trainable params: 1,204		

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 10s 169us/step - loss: 0.1878 - acc: 0.9446 - val_loss: 0.1002 - val_acc: 0.9689
Epoch 2/20
60000/60000 [=====] - 7s 121us/step - loss: 0.0724 - acc: 0.9784 - val_loss: 0.0814 - val_acc: 0.9749
Epoch 3/20
60000/60000 [=====] - 7s 120us/step - loss: 0.0495 - acc: 0.9845 - val_loss: 0.0801 - val_acc: 0.9733
Epoch 4/20
60000/60000 [=====] - 7s 120us/step - loss: 0.0370 - acc: 0.9886 - val_loss: 0.0759 - val_acc: 0.9775
Epoch 5/20
60000/60000 [=====] - 7s 121us/step - loss: 0.0268 - acc: 0.9914 - val_loss: 0.0737 - val_acc: 0.9776
Epoch 6/20
```

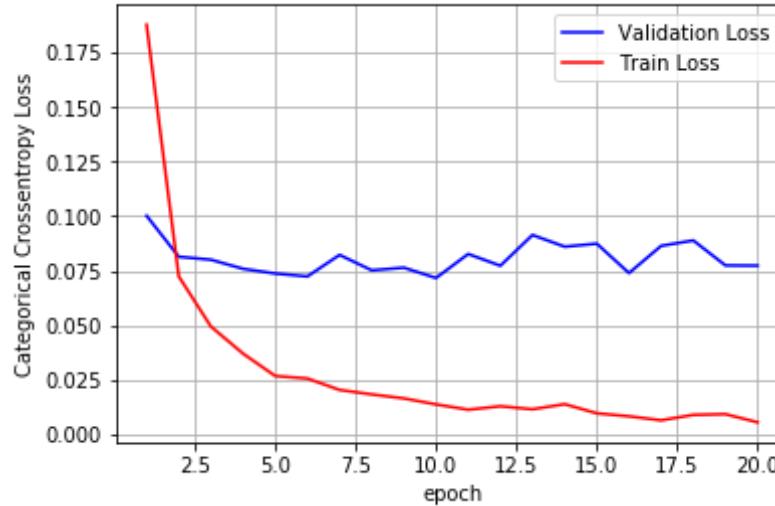
```
60000/60000 [=====] - 7s 120us/step - loss: 0.  
0256 - acc: 0.9914 - val_loss: 0.0724 - val_acc: 0.9778  
Epoch 7/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
0204 - acc: 0.9936 - val_loss: 0.0823 - val_acc: 0.9774  
Epoch 8/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0183 - acc: 0.9938 - val_loss: 0.0752 - val_acc: 0.9789  
Epoch 9/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
0165 - acc: 0.9944 - val_loss: 0.0764 - val_acc: 0.9777  
Epoch 10/20  
60000/60000 [=====] - 7s 123us/step - loss: 0.  
0138 - acc: 0.9958 - val_loss: 0.0717 - val_acc: 0.9789  
Epoch 11/20  
60000/60000 [=====] - 7s 119us/step - loss: 0.  
0113 - acc: 0.9965 - val_loss: 0.0827 - val_acc: 0.9791  
Epoch 12/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
0129 - acc: 0.9960 - val_loss: 0.0773 - val_acc: 0.9792  
Epoch 13/20  
60000/60000 [=====] - 7s 125us/step - loss: 0.  
0115 - acc: 0.9960 - val_loss: 0.0914 - val_acc: 0.9769  
Epoch 14/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0139 - acc: 0.9955 - val_loss: 0.0860 - val_acc: 0.9781  
Epoch 15/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
0097 - acc: 0.9968 - val_loss: 0.0874 - val_acc: 0.9788  
Epoch 16/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0083 - acc: 0.9971 - val_loss: 0.0739 - val_acc: 0.9810  
Epoch 17/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0065 - acc: 0.9980 - val_loss: 0.0864 - val_acc: 0.9787  
Epoch 18/20  
60000/60000 [=====] - 7s 122us/step - loss: 0.  
0090 - acc: 0.9969 - val_loss: 0.0889 - val_acc: 0.9773  
Epoch 19/20
```

```
60000/60000 [=====] - 7s 122us/step - loss: 0.  
0093 - acc: 0.9968 - val_loss: 0.0774 - val_acc: 0.9790  
Epoch 20/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0056 - acc: 0.9982 - val_loss: 0.0774 - val_acc: 0.9810
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.07735706123386735  
Test accuracy: 0.981
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

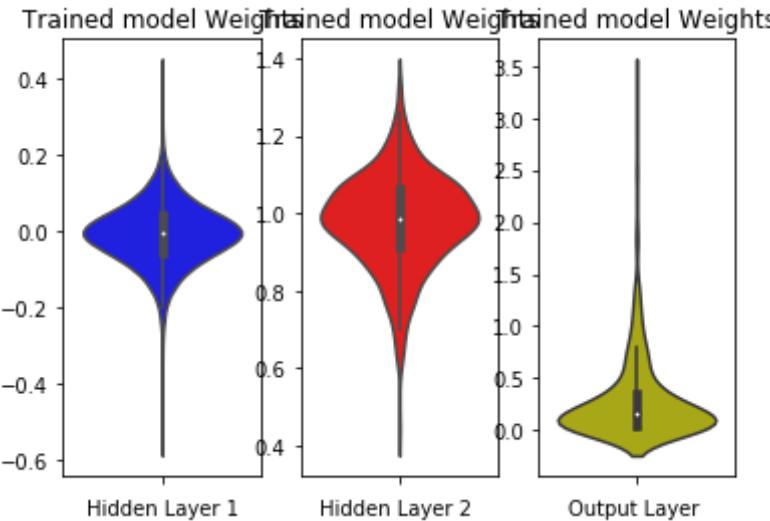
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



## Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_37"

Layer (type)	Output Shape	Param #
dense_103 (Dense)	(None, 501)	393285
batch_normalization_25 (Batch Normalization)	(None, 501)	2004
dense_104 (Dense)	(None, 101)	50702
batch_normalization_26 (Batch Normalization)	(None, 101)	404
dense_105 (Dense)	(None, 10)	1020
<hr/>		
Total params: 447,415		
Trainable params: 446,211		
Non-trainable params: 1,204		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 150us/step - loss: 0.
4448 - acc: 0.8724 - val_loss: 0.2391 - val_acc: 0.9323
Epoch 2/20
60000/60000 [=====] - 6s 98us/step - loss: 0.2
120 - acc: 0.9419 - val_loss: 0.1804 - val_acc: 0.9472
Epoch 3/20
60000/60000 [=====] - 6s 96us/step - loss: 0.1
629 - acc: 0.9563 - val_loss: 0.1512 - val_acc: 0.9564
Epoch 4/20
60000/60000 [=====] - 6s 97us/step - loss: 0.1
350 - acc: 0.9636 - val_loss: 0.1349 - val_acc: 0.9605
Epoch 5/20
60000/60000 [=====] - 6s 101us/step - loss: 0.
1147 - acc: 0.9693 - val_loss: 0.1232 - val_acc: 0.9634
```

```
- -  
Epoch 6/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
997 - acc: 0.9735 - val_loss: 0.1151 - val_acc: 0.9660  
Epoch 7/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
883 - acc: 0.9765 - val_loss: 0.1079 - val_acc: 0.9683  
Epoch 8/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
785 - acc: 0.9801 - val_loss: 0.1034 - val_acc: 0.9686  
Epoch 9/20  
60000/60000 [=====] - 6s 96us/step - loss: 0.0  
698 - acc: 0.9826 - val_loss: 0.0989 - val_acc: 0.9701  
Epoch 10/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
633 - acc: 0.9848 - val_loss: 0.0948 - val_acc: 0.9715  
Epoch 11/20  
60000/60000 [=====] - 6s 96us/step - loss: 0.0  
566 - acc: 0.9866 - val_loss: 0.0933 - val_acc: 0.9713  
Epoch 12/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
520 - acc: 0.9879 - val_loss: 0.0905 - val_acc: 0.9737  
Epoch 13/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
462 - acc: 0.9896 - val_loss: 0.0878 - val_acc: 0.9742  
Epoch 14/20  
60000/60000 [=====] - 6s 100us/step - loss: 0.  
0427 - acc: 0.9910 - val_loss: 0.0866 - val_acc: 0.9738  
Epoch 15/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
388 - acc: 0.9917 - val_loss: 0.0855 - val_acc: 0.9739  
Epoch 16/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
349 - acc: 0.9932 - val_loss: 0.0846 - val_acc: 0.9736  
Epoch 17/20  
60000/60000 [=====] - 6s 98us/step - loss: 0.0  
331 - acc: 0.9935 - val_loss: 0.0832 - val_acc: 0.9742  
Epoch 18/20  
60000/60000 [=====] - 6s 97us/step - loss: 0.0  
301 - acc: 0.9947 - val_loss: 0.0837 - val_acc: 0.9738
```

```
Epoch 19/20
60000/60000 [=====] - 6s 99us/step - loss: 0.0
279 - acc: 0.9951 - val_loss: 0.0822 - val_acc: 0.9736
Epoch 20/20
60000/60000 [=====] - 6s 98us/step - loss: 0.0
251 - acc: 0.9963 - val_loss: 0.0816 - val_acc: 0.9742
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

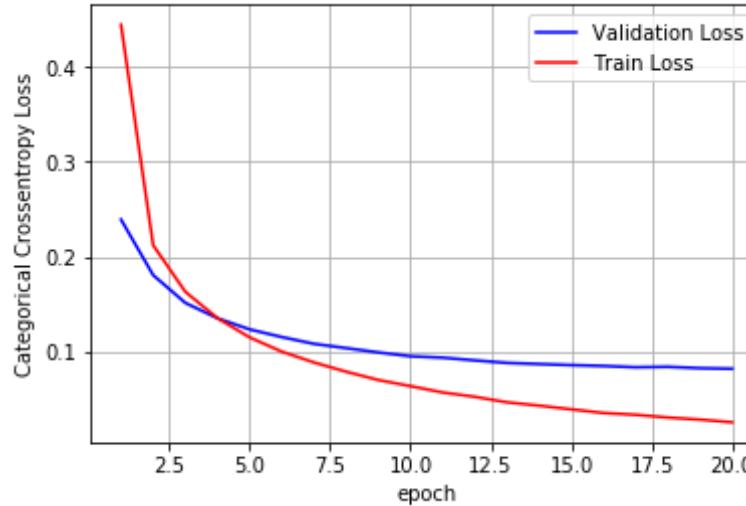
```
Test score: 0.0816448771264404
Test accuracy: 0.9742
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

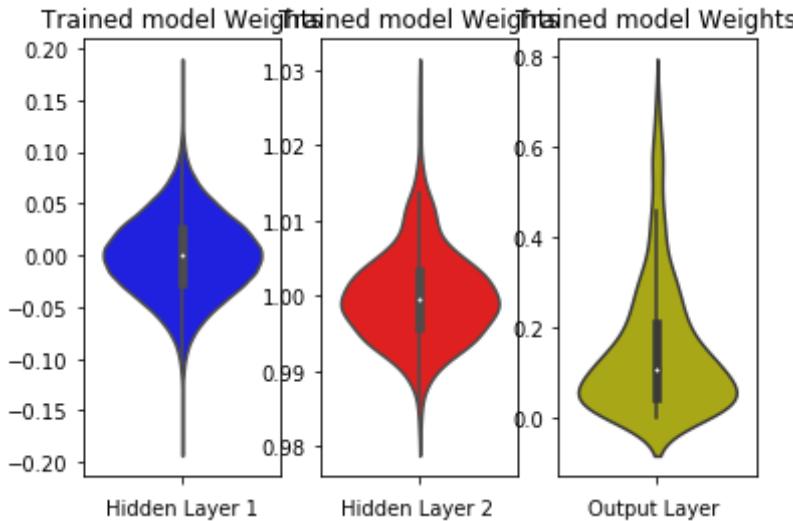
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



Model 5: relu activation with BatchNormalization + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_38"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```
=====
dense_106 (Dense)           (None, 501)          393285
batch_normalization_27 (Batch Normalization) (None, 501)      2004
dense_107 (Dense)           (None, 101)          50702
batch_normalization_28 (Batch Normalization) (None, 101)      404
dense_108 (Dense)           (None, 10)           1020
=====
Total params: 447,415
Trainable params: 446,211
Non-trainable params: 1,204
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 11s 178us/step - loss: 0.2129 - acc: 0.9362 - val_loss: 0.1227 - val_acc: 0.9652
Epoch 2/20
60000/60000 [=====] - 7s 120us/step - loss: 0.0773 - acc: 0.9769 - val_loss: 0.0909 - val_acc: 0.9716
Epoch 3/20
60000/60000 [=====] - 7s 121us/step - loss: 0.0486 - acc: 0.9848 - val_loss: 0.0830 - val_acc: 0.9741
Epoch 4/20
60000/60000 [=====] - 7s 122us/step - loss: 0.0335 - acc: 0.9897 - val_loss: 0.0767 - val_acc: 0.9763
Epoch 5/20
60000/60000 [=====] - 7s 122us/step - loss: 0.0240 - acc: 0.9924 - val_loss: 0.0815 - val_acc: 0.9761
Epoch 6/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
```

```
0196 - acc: 0.9939 - val_loss: 0.0772 - val_acc: 0.9767
Epoch 7/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0167 - acc: 0.9949 - val_loss: 0.0735 - val_acc: 0.9779
Epoch 8/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0149 - acc: 0.9951 - val_loss: 0.0792 - val_acc: 0.9771
Epoch 9/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0142 - acc: 0.9957 - val_loss: 0.0893 - val_acc: 0.9757
Epoch 10/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0119 - acc: 0.9961 - val_loss: 0.0813 - val_acc: 0.9781
Epoch 11/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0099 - acc: 0.9968 - val_loss: 0.0796 - val_acc: 0.9795
Epoch 12/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0121 - acc: 0.9963 - val_loss: 0.0777 - val_acc: 0.9780
Epoch 13/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0098 - acc: 0.9969 - val_loss: 0.0830 - val_acc: 0.9797
Epoch 14/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0082 - acc: 0.9973 - val_loss: 0.0905 - val_acc: 0.9765
Epoch 15/20
60000/60000 [=====] - 7s 121us/step - loss: 0.
0085 - acc: 0.9971 - val_loss: 0.0800 - val_acc: 0.9793
Epoch 16/20
60000/60000 [=====] - 7s 122us/step - loss: 0.
0093 - acc: 0.9968 - val_loss: 0.0869 - val_acc: 0.9792
Epoch 17/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0082 - acc: 0.9973 - val_loss: 0.0811 - val_acc: 0.9805
Epoch 18/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0060 - acc: 0.9981 - val_loss: 0.0773 - val_acc: 0.9816
Epoch 19/20
60000/60000 [=====] - 7s 120us/step - loss: 0.
```

```
0079 - acc: 0.9974 - val_loss: 0.0832 - val_acc: 0.9800
Epoch 20/20
60000/60000 [=====] - 7s 121us/step - loss: 0.
0055 - acc: 0.9981 - val_loss: 0.0760 - val_acc: 0.9814
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

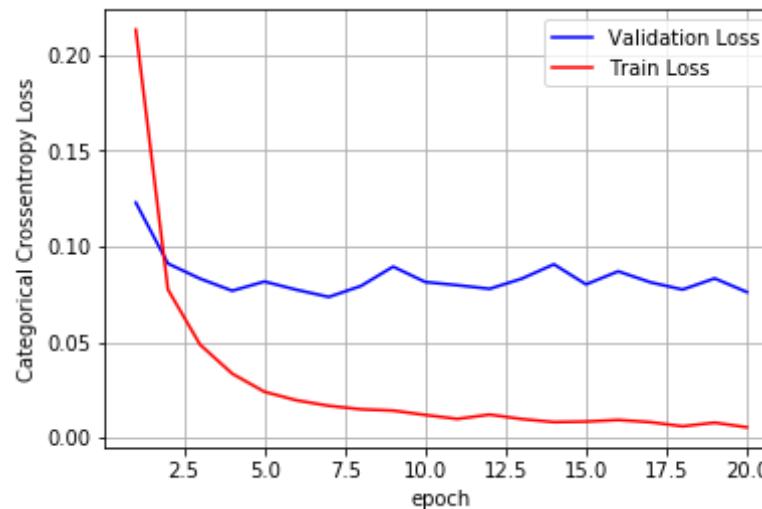
```
Test score: 0.0760320247169545
Test accuracy: 0.9814
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



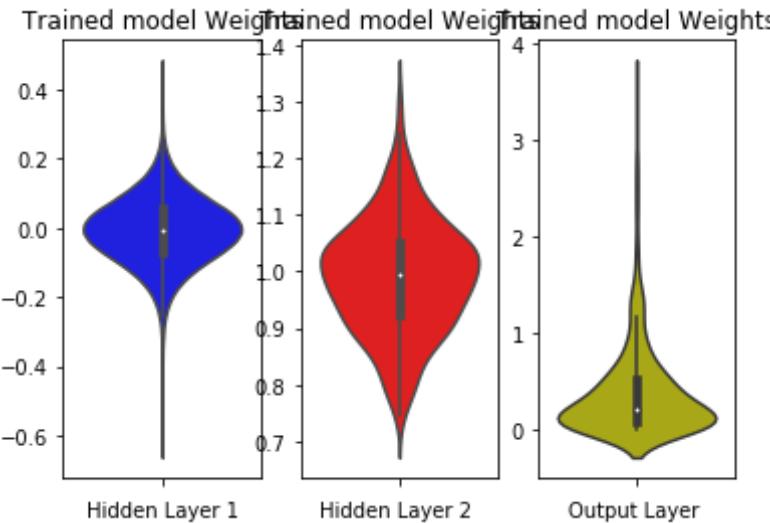
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



### Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(101, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.140, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_39"

Layer (type)	Output Shape	Param #
<hr/>		

dense_109 (Dense)	(None, 501)	393285
batch_normalization_29 (Batch Normalization)	(None, 501)	2004
dense_110 (Dense)	(None, 101)	50702
batch_normalization_30 (Batch Normalization)	(None, 101)	404
dense_111 (Dense)	(None, 10)	1020
<hr/>		
Total params: 447,415		
Trainable params: 446,211		
Non-trainable params: 1,204		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 156us/step - loss: 0.
6669 - acc: 0.7980 - val_loss: 0.3380 - val_acc: 0.9068
Epoch 2/20
60000/60000 [=====] - 6s 97us/step - loss: 0.3
145 - acc: 0.9114 - val_loss: 0.2554 - val_acc: 0.9283
Epoch 3/20
60000/60000 [=====] - 6s 95us/step - loss: 0.2
498 - acc: 0.9288 - val_loss: 0.2168 - val_acc: 0.9386
Epoch 4/20
60000/60000 [=====] - 6s 97us/step - loss: 0.2
120 - acc: 0.9394 - val_loss: 0.1932 - val_acc: 0.9421
Epoch 5/20
60000/60000 [=====] - 6s 95us/step - loss: 0.1
870 - acc: 0.9468 - val_loss: 0.1766 - val_acc: 0.9472
Epoch 6/20
60000/60000 [=====] - 6s 97us/step - loss: 0.1
```

```
676 - acc: 0.9529 - val_loss: 0.1636 - val_acc: 0.9500
Epoch 7/20
60000/60000 [=====] - 6s 99us/step - loss: 0.1
523 - acc: 0.9574 - val_loss: 0.1542 - val_acc: 0.9534
Epoch 8/20
60000/60000 [=====] - 6s 97us/step - loss: 0.1
405 - acc: 0.9608 - val_loss: 0.1458 - val_acc: 0.9555
Epoch 9/20
60000/60000 [=====] - 6s 101us/step - loss: 0.
1285 - acc: 0.9632 - val_loss: 0.1402 - val_acc: 0.9568
Epoch 10/20
60000/60000 [=====] - 6s 103us/step - loss: 0.
1202 - acc: 0.9661 - val_loss: 0.1338 - val_acc: 0.9593
Epoch 11/20
60000/60000 [=====] - 6s 101us/step - loss: 0.
1123 - acc: 0.9681 - val_loss: 0.1287 - val_acc: 0.9609
Epoch 12/20
60000/60000 [=====] - 6s 101us/step - loss: 0.
1054 - acc: 0.9710 - val_loss: 0.1247 - val_acc: 0.9621
Epoch 13/20
60000/60000 [=====] - 6s 102us/step - loss: 0.
0987 - acc: 0.9720 - val_loss: 0.1211 - val_acc: 0.9631
Epoch 14/20
60000/60000 [=====] - 6s 101us/step - loss: 0.
0924 - acc: 0.9746 - val_loss: 0.1176 - val_acc: 0.9653
Epoch 15/20
60000/60000 [=====] - 6s 99us/step - loss: 0.0
878 - acc: 0.9760 - val_loss: 0.1144 - val_acc: 0.9651
Epoch 16/20
60000/60000 [=====] - 6s 99us/step - loss: 0.0
833 - acc: 0.9770 - val_loss: 0.1118 - val_acc: 0.9659
Epoch 17/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
786 - acc: 0.9790 - val_loss: 0.1092 - val_acc: 0.9661
Epoch 18/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
748 - acc: 0.9801 - val_loss: 0.1071 - val_acc: 0.9674
Epoch 19/20
60000/60000 [=====] - 6s 97us/step - loss: 0.0
```

```
705 - acc: 0.9816 - val_loss: 0.1055 - val_acc: 0.9682
Epoch 20/20
60000/60000 [=====] - 6s 98us/step - loss: 0.0
676 - acc: 0.9820 - val_loss: 0.1036 - val_acc: 0.9679
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

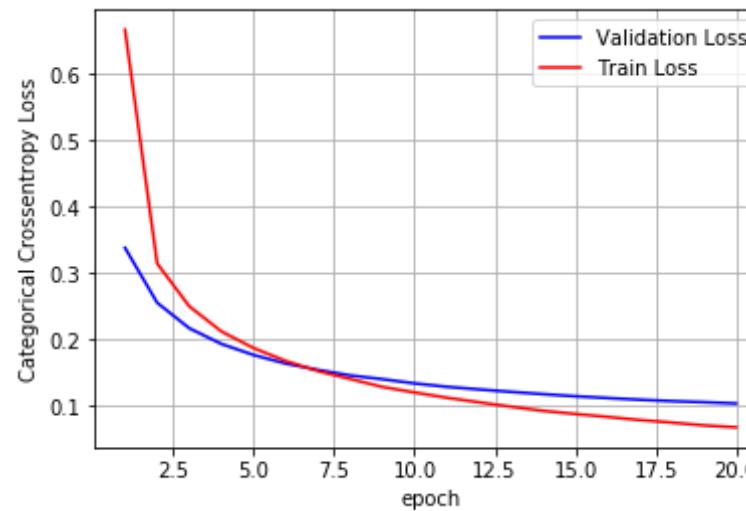
```
Test score: 0.10358885963968932
Test accuracy: 0.9679
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



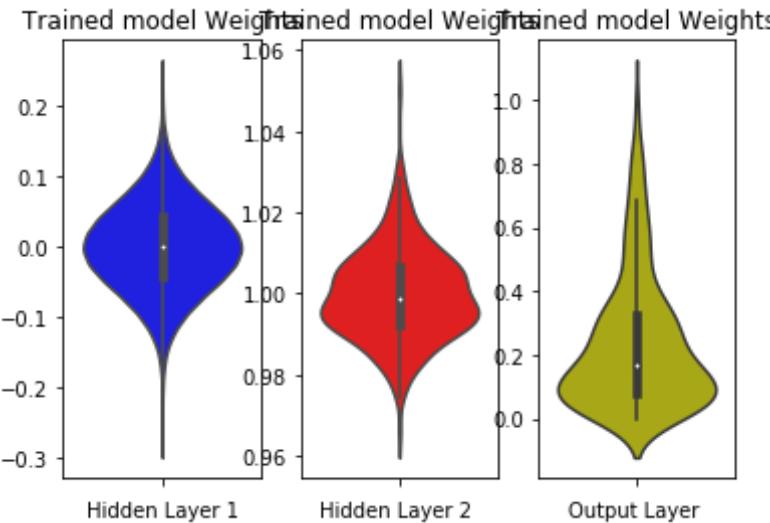
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 6: sigmoid activation with Dropout + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.058, seed=None) ))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_68"

Layer (type)	Output Shape	Param #
dense_220 (Dense)	(None, 501)	393285
dropout_43 (Dropout)	(None, 501)	0
dense_221 (Dense)	(None, 101)	50702
dropout_44 (Dropout)	(None, 101)	0
dense_222 (Dense)	(None, 10)	1020

---

Total params: 445,007  
Trainable params: 445,007  
Non-trainable params: 0

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 14s 234us/step - loss: 0.8513 - acc: 0.7398 - val_loss: 0.3079 - val_acc: 0.9126
Epoch 2/20
60000/60000 [=====] - 7s 119us/step - loss: 0.3761 - acc: 0.8914 - val_loss: 0.2277 - val_acc: 0.9295
Epoch 3/20
60000/60000 [=====] - 7s 117us/step - loss: 0.2942 - acc: 0.9155 - val_loss: 0.1868 - val_acc: 0.9426
Epoch 4/20
60000/60000 [=====] - 7s 118us/step - loss: 0.2443 - acc: 0.9305 - val_loss: 0.1563 - val_acc: 0.9501
Epoch 5/20
60000/60000 [=====] - 7s 120us/step - loss: 0.2115 - acc: 0.9380 - val_loss: 0.1347 - val_acc: 0.9583
Epoch 6/20
```

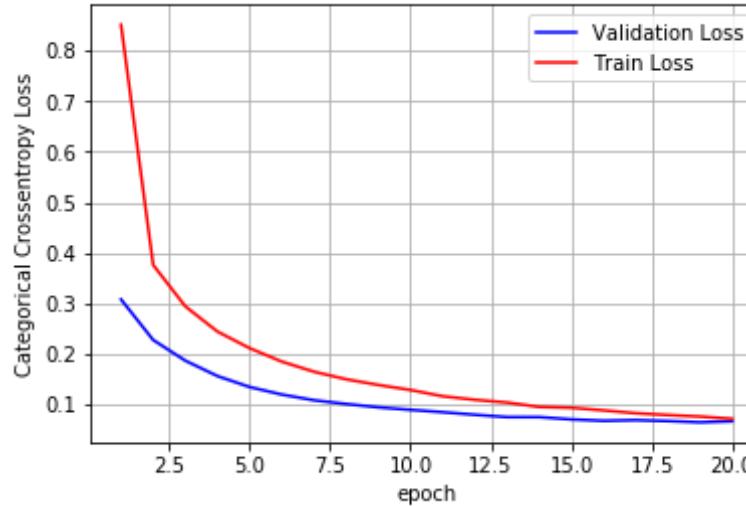
```
60000/60000 [=====] - 7s 120us/step - loss: 0.  
1849 - acc: 0.9470 - val_loss: 0.1198 - val_acc: 0.9635  
Epoch 7/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
1650 - acc: 0.9515 - val_loss: 0.1086 - val_acc: 0.9669  
Epoch 8/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
1500 - acc: 0.9563 - val_loss: 0.1012 - val_acc: 0.9692  
Epoch 9/20  
60000/60000 [=====] - 7s 119us/step - loss: 0.  
1388 - acc: 0.9593 - val_loss: 0.0945 - val_acc: 0.9713  
Epoch 10/20  
60000/60000 [=====] - 7s 119us/step - loss: 0.  
1289 - acc: 0.9623 - val_loss: 0.0896 - val_acc: 0.9725  
Epoch 11/20  
60000/60000 [=====] - 7s 119us/step - loss: 0.  
1165 - acc: 0.9653 - val_loss: 0.0850 - val_acc: 0.9741  
Epoch 12/20  
60000/60000 [=====] - 7s 120us/step - loss: 0.  
1095 - acc: 0.9675 - val_loss: 0.0799 - val_acc: 0.9746  
Epoch 13/20  
60000/60000 [=====] - 7s 122us/step - loss: 0.  
1041 - acc: 0.9686 - val_loss: 0.0753 - val_acc: 0.9769  
Epoch 14/20  
60000/60000 [=====] - 8s 126us/step - loss: 0.  
0953 - acc: 0.9714 - val_loss: 0.0752 - val_acc: 0.9769  
Epoch 15/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0937 - acc: 0.9720 - val_loss: 0.0707 - val_acc: 0.9783  
Epoch 16/20  
60000/60000 [=====] - 7s 122us/step - loss: 0.  
0885 - acc: 0.9737 - val_loss: 0.0679 - val_acc: 0.9786  
Epoch 17/20  
60000/60000 [=====] - 7s 122us/step - loss: 0.  
0829 - acc: 0.9750 - val_loss: 0.0692 - val_acc: 0.9779  
Epoch 18/20  
60000/60000 [=====] - 7s 119us/step - loss: 0.  
0793 - acc: 0.9762 - val_loss: 0.0674 - val_acc: 0.9792  
Epoch 19/20
```

```
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0763 - acc: 0.9766 - val_loss: 0.0649 - val_acc: 0.9807  
Epoch 20/20  
60000/60000 [=====] - 7s 121us/step - loss: 0.  
0718 - acc: 0.9784 - val_loss: 0.0669 - val_acc: 0.9799
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.06690293591122609  
Test accuracy: 0.9799
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

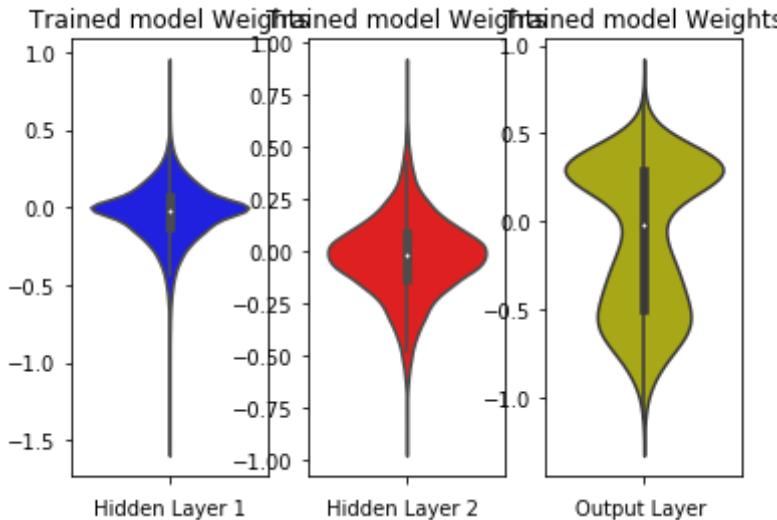
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



## Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.058, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

Model: "sequential_69"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_223 (Dense)	(None, 501)	393285
dropout_45 (Dropout)	(None, 501)	0
dense_224 (Dense)	(None, 101)	50702
dropout_46 (Dropout)	(None, 101)	0
dense_225 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 12s 205us/step - loss: 2.4230 - acc: 0.1096 - val_loss: 2.2459 - val_acc: 0.3144
Epoch 2/20
60000/60000 [=====] - 6s 100us/step - loss: 2.3073 - acc: 0.1349 - val_loss: 2.2150 - val_acc: 0.3575
Epoch 3/20
60000/60000 [=====] - 6s 101us/step - loss: 2.2586 - acc: 0.1629 - val_loss: 2.1813 - val_acc: 0.4956
Epoch 4/20
60000/60000 [=====] - 6s 98us/step - loss: 2.2145 - acc: 0.1935 - val_loss: 2.1318 - val_acc: 0.5269
Epoch 5/20
60000/60000 [=====] - 6s 99us/step - loss: 2.1645 - acc: 0.2266 - val_loss: 2.0569 - val_acc: 0.5315
```

```
- -  
Epoch 6/20  
60000/60000 [=====] - 6s 98us/step - loss: 2.0  
958 - acc: 0.2628 - val_loss: 1.9503 - val_acc: 0.5720  
Epoch 7/20  
60000/60000 [=====] - 6s 100us/step - loss: 2.  
0047 - acc: 0.3028 - val_loss: 1.8128 - val_acc: 0.5926  
Epoch 8/20  
60000/60000 [=====] - 6s 100us/step - loss: 1.  
8951 - acc: 0.3441 - val_loss: 1.6595 - val_acc: 0.6462  
Epoch 9/20  
60000/60000 [=====] - 6s 101us/step - loss: 1.  
7734 - acc: 0.3865 - val_loss: 1.5084 - val_acc: 0.6520  
Epoch 10/20  
60000/60000 [=====] - 6s 100us/step - loss: 1.  
6565 - acc: 0.4261 - val_loss: 1.3725 - val_acc: 0.6782  
Epoch 11/20  
60000/60000 [=====] - 6s 99us/step - loss: 1.5  
469 - acc: 0.4628 - val_loss: 1.2564 - val_acc: 0.6984  
Epoch 12/20  
60000/60000 [=====] - 6s 101us/step - loss: 1.  
4516 - acc: 0.4916 - val_loss: 1.1623 - val_acc: 0.7059  
Epoch 13/20  
60000/60000 [=====] - 6s 99us/step - loss: 1.3  
761 - acc: 0.5171 - val_loss: 1.0830 - val_acc: 0.7396  
Epoch 14/20  
60000/60000 [=====] - 6s 99us/step - loss: 1.3  
005 - acc: 0.5432 - val_loss: 1.0154 - val_acc: 0.7394  
Epoch 15/20  
60000/60000 [=====] - 6s 98us/step - loss: 1.2  
417 - acc: 0.5655 - val_loss: 0.9616 - val_acc: 0.7491  
Epoch 16/20  
60000/60000 [=====] - 6s 97us/step - loss: 1.1  
904 - acc: 0.5832 - val_loss: 0.9149 - val_acc: 0.7643  
Epoch 17/20  
60000/60000 [=====] - 6s 100us/step - loss: 1.  
1412 - acc: 0.6027 - val_loss: 0.8737 - val_acc: 0.7606  
Epoch 18/20  
60000/60000 [=====] - 6s 101us/step - loss: 1.  
1011 - acc: 0.6172 - val_loss: 0.8374 - val_acc: 0.7734
```

```
Epoch 19/20
60000/60000 [=====] - 6s 107us/step - loss: 1.
0623 - acc: 0.6328 - val_loss: 0.8055 - val_acc: 0.7815
Epoch 20/20
60000/60000 [=====] - 6s 101us/step - loss: 1.
0304 - acc: 0.6450 - val_loss: 0.7754 - val_acc: 0.7899
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

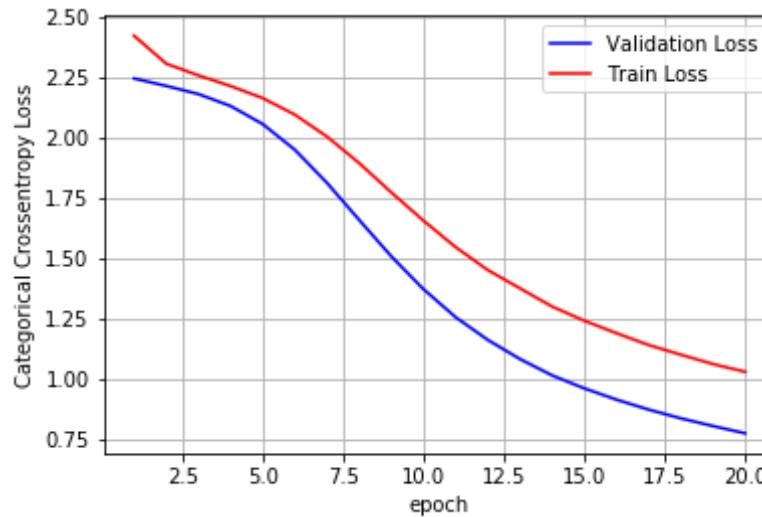
```
Test score: 0.7754171266555786
Test accuracy: 0.7899
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

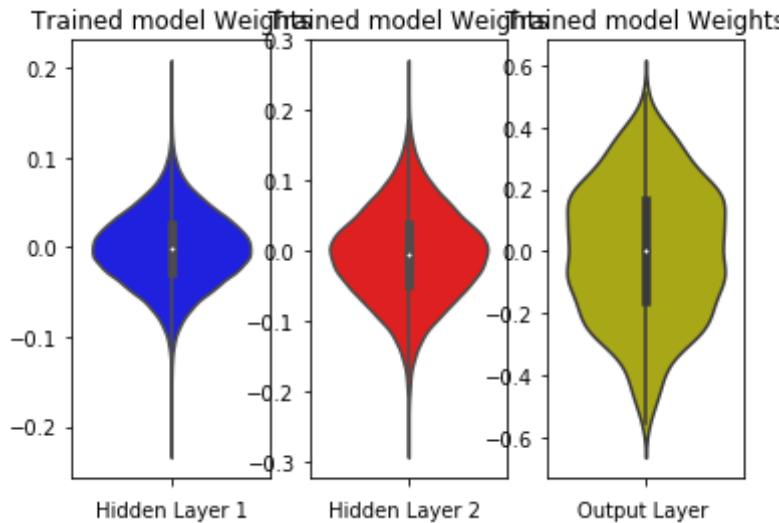
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
```

```
plt.xlabel('Output Layer ')
plt.show()
```



Model 6: sigmoid activation with Dropout + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.140, seed=None)) )
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_70"

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```
=====
dense_226 (Dense)           (None, 501)          393285
dropout_47 (Dropout)         (None, 501)          0
dense_227 (Dense)           (None, 101)          50702
dropout_48 (Dropout)         (None, 101)          0
dense_228 (Dense)           (None, 10)           1020
=====
Total params: 445,007
Trainable params: 445,007
Non-trainable params: 0
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 14s 235us/step - loss: 0.8554 - acc: 0.7380 - val_loss: 0.3123 - val_acc: 0.9111
Epoch 2/20
60000/60000 [=====] - 8s 126us/step - loss: 0.3896 - acc: 0.8885 - val_loss: 0.2285 - val_acc: 0.9296
Epoch 3/20
60000/60000 [=====] - 7s 122us/step - loss: 0.3027 - acc: 0.9133 - val_loss: 0.1835 - val_acc: 0.9455
Epoch 4/20
60000/60000 [=====] - 7s 122us/step - loss: 0.2491 - acc: 0.9288 - val_loss: 0.1546 - val_acc: 0.9524
Epoch 5/20
60000/60000 [=====] - 7s 123us/step - loss: 0.2110 - acc: 0.9395 - val_loss: 0.1343 - val_acc: 0.9596
Epoch 6/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
```

```
1883 - acc: 0.9461 - val_loss: 0.1225 - val_acc: 0.9613
Epoch 7/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
1698 - acc: 0.9510 - val_loss: 0.1089 - val_acc: 0.9670
Epoch 8/20
60000/60000 [=====] - 7s 121us/step - loss: 0.
1543 - acc: 0.9554 - val_loss: 0.0995 - val_acc: 0.9685
Epoch 9/20
60000/60000 [=====] - 7s 122us/step - loss: 0.
1357 - acc: 0.9608 - val_loss: 0.0931 - val_acc: 0.9703
Epoch 10/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
1280 - acc: 0.9627 - val_loss: 0.0899 - val_acc: 0.9722
Epoch 11/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
1188 - acc: 0.9650 - val_loss: 0.0873 - val_acc: 0.9722
Epoch 12/20
60000/60000 [=====] - 7s 122us/step - loss: 0.
1102 - acc: 0.9671 - val_loss: 0.0773 - val_acc: 0.9759
Epoch 13/20
60000/60000 [=====] - 7s 120us/step - loss: 0.
1024 - acc: 0.9699 - val_loss: 0.0762 - val_acc: 0.9758
Epoch 14/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
0985 - acc: 0.9706 - val_loss: 0.0758 - val_acc: 0.9770
Epoch 15/20
60000/60000 [=====] - 7s 123us/step - loss: 0.
0928 - acc: 0.9719 - val_loss: 0.0742 - val_acc: 0.9770
Epoch 16/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0875 - acc: 0.9733 - val_loss: 0.0720 - val_acc: 0.9772
Epoch 17/20
60000/60000 [=====] - 8s 126us/step - loss: 0.
0832 - acc: 0.9751 - val_loss: 0.0675 - val_acc: 0.9789
Epoch 18/20
60000/60000 [=====] - 7s 124us/step - loss: 0.
0790 - acc: 0.9764 - val_loss: 0.0683 - val_acc: 0.9780
Epoch 19/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
```

```
0755 - acc: 0.9773 - val_loss: 0.0676 - val_acc: 0.9796
Epoch 20/20
60000/60000 [=====] - 7s 119us/step - loss: 0.
0718 - acc: 0.9779 - val_loss: 0.0664 - val_acc: 0.9792
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

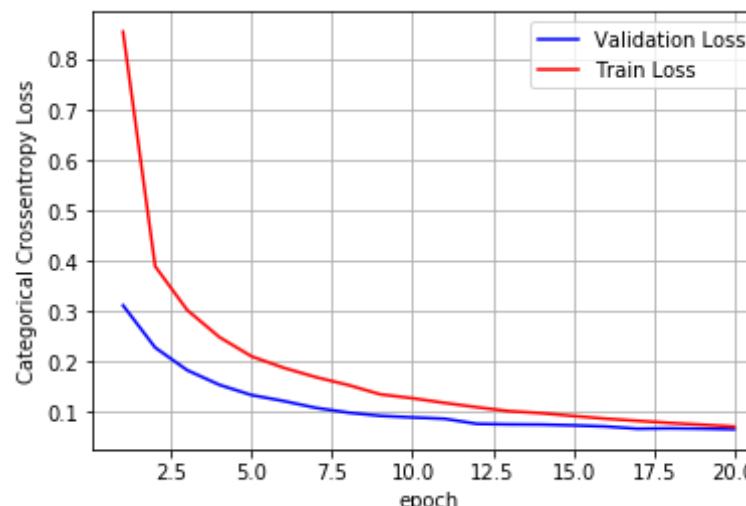
```
Test score: 0.0663997630263446
Test accuracy: 0.9792
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



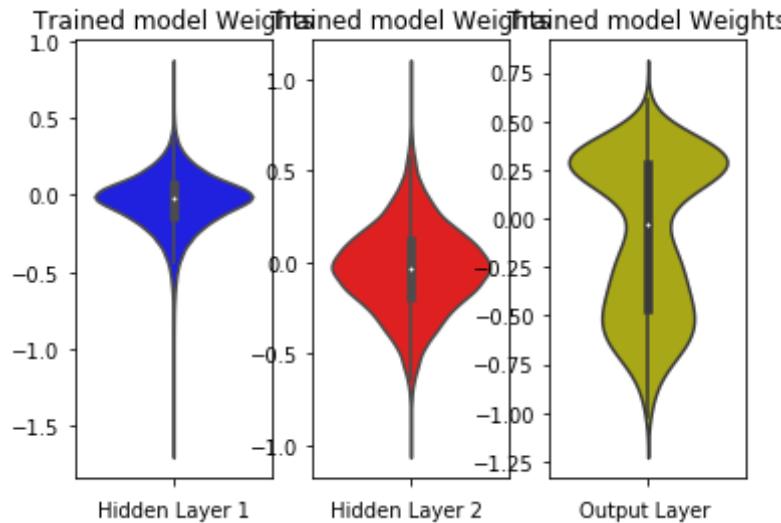
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(501, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.063, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(101, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.140, seed=None) ))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_71"

Layer (type)	Output Shape	Param #
<hr/>		

dense_229 (Dense)	(None, 501)	393285
dropout_49 (Dropout)	(None, 501)	0
dense_230 (Dense)	(None, 101)	50702
dropout_50 (Dropout)	(None, 101)	0
dense_231 (Dense)	(None, 10)	1020
<hr/>		
Total params: 445,007		
Trainable params: 445,007		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 13s 210us/step - loss: 2.5177 - acc: 0.1128 - val_loss: 2.1730 - val_acc: 0.3444
Epoch 2/20
60000/60000 [=====] - 6s 98us/step - loss: 2.3079 - acc: 0.1532 - val_loss: 2.0714 - val_acc: 0.4773
Epoch 3/20
60000/60000 [=====] - 6s 98us/step - loss: 2.1883 - acc: 0.2049 - val_loss: 1.9582 - val_acc: 0.5786
Epoch 4/20
60000/60000 [=====] - 6s 98us/step - loss: 2.0803 - acc: 0.2603 - val_loss: 1.8170 - val_acc: 0.6196
Epoch 5/20
60000/60000 [=====] - 6s 97us/step - loss: 1.9657 - acc: 0.3152 - val_loss: 1.6462 - val_acc: 0.6714
Epoch 6/20
60000/60000 [=====] - 6s 97us/step - loss: 1.8
```

```
343 - acc: 0.3668 - val_loss: 1.4731 - val_acc: 0.6891
Epoch 7/20
60000/60000 [=====] - 6s 97us/step - loss: 1.7
068 - acc: 0.4148 - val_loss: 1.3130 - val_acc: 0.7130
Epoch 8/20
60000/60000 [=====] - 6s 97us/step - loss: 1.5
861 - acc: 0.4574 - val_loss: 1.1809 - val_acc: 0.7324
Epoch 9/20
60000/60000 [=====] - 6s 98us/step - loss: 1.4
850 - acc: 0.4923 - val_loss: 1.0757 - val_acc: 0.7558
Epoch 10/20
60000/60000 [=====] - 6s 98us/step - loss: 1.3
891 - acc: 0.5260 - val_loss: 0.9930 - val_acc: 0.7540
Epoch 11/20
60000/60000 [=====] - 6s 97us/step - loss: 1.3
097 - acc: 0.5548 - val_loss: 0.9262 - val_acc: 0.7685
Epoch 12/20
60000/60000 [=====] - 6s 96us/step - loss: 1.2
470 - acc: 0.5750 - val_loss: 0.8708 - val_acc: 0.7791
Epoch 13/20
60000/60000 [=====] - 6s 96us/step - loss: 1.1
904 - acc: 0.5968 - val_loss: 0.8263 - val_acc: 0.7851
Epoch 14/20
60000/60000 [=====] - 6s 98us/step - loss: 1.1
408 - acc: 0.6131 - val_loss: 0.7872 - val_acc: 0.7909
Epoch 15/20
60000/60000 [=====] - 6s 98us/step - loss: 1.0
943 - acc: 0.6297 - val_loss: 0.7539 - val_acc: 0.7954
Epoch 16/20
60000/60000 [=====] - 6s 98us/step - loss: 1.0
594 - acc: 0.6416 - val_loss: 0.7229 - val_acc: 0.8033
Epoch 17/20
60000/60000 [=====] - 6s 98us/step - loss: 1.0
225 - acc: 0.6554 - val_loss: 0.6963 - val_acc: 0.8110
Epoch 18/20
60000/60000 [=====] - 6s 98us/step - loss: 0.9
936 - acc: 0.6665 - val_loss: 0.6744 - val_acc: 0.8124
Epoch 19/20
60000/60000 [=====] - 6s 99us/step - loss: 0.9
```

```
683 - acc: 0.6758 - val_loss: 0.6528 - val_acc: 0.8169
Epoch 20/20
60000/60000 [=====] - 6s 98us/step - loss: 0.9
424 - acc: 0.6862 - val_loss: 0.6328 - val_acc: 0.8226
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

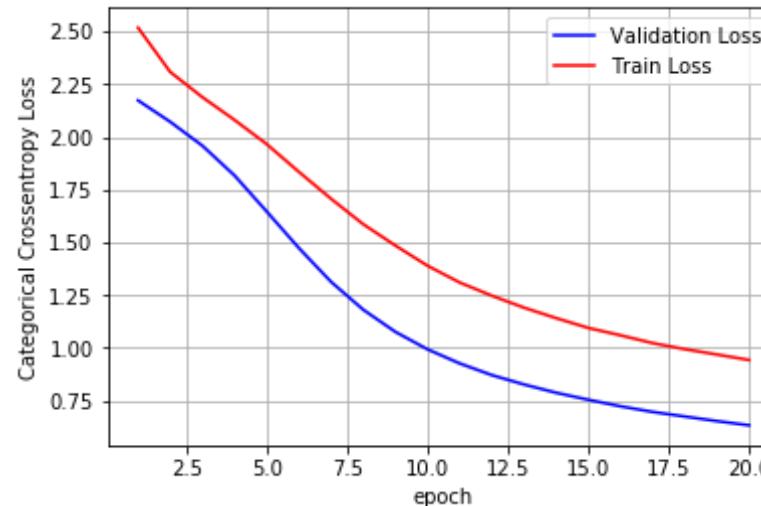
```
Test score: 0.632832354927063
Test accuracy: 0.8226
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



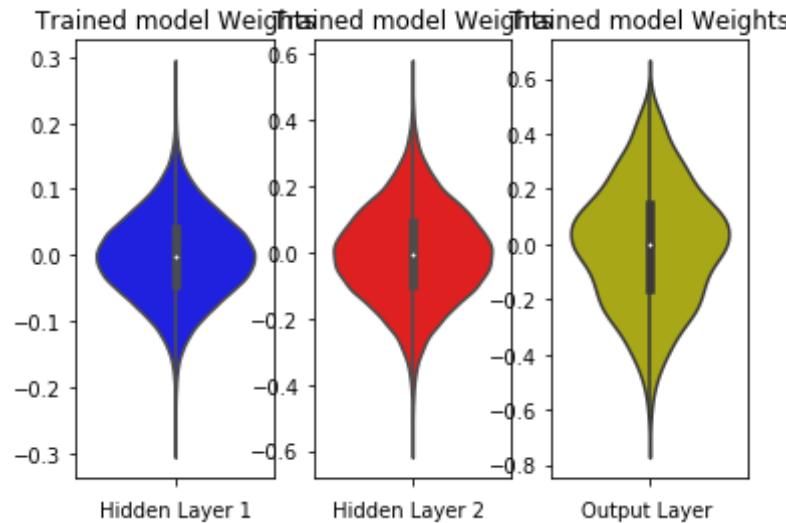
```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Architecture 2: 784-256-128-64-10

### Model 1: sigmoid activation + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.0072, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))
```

```
model_1.summary()
```

Model: "sequential\_44"

Layer (type)	Output Shape	Param #
<hr/>		
dense_124 (Dense)	(None, 256)	200960
dense_125 (Dense)	(None, 128)	32896
dense_126 (Dense)	(None, 64)	8256
dense_127 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 8s 125us/step - loss: 0.  
8760 - acc: 0.7448 - val\_loss: 0.3209 - val\_acc: 0.9153  
Epoch 2/20  
60000/60000 [=====] - 4s 67us/step - loss: 0.2  
588 - acc: 0.9281 - val\_loss: 0.2015 - val\_acc: 0.9423  
Epoch 3/20  
60000/60000 [=====] - 4s 65us/step - loss: 0.1  
746 - acc: 0.9507 - val\_loss: 0.1569 - val\_acc: 0.9545  
Epoch 4/20  
60000/60000 [=====] - 4s 66us/step - loss: 0.1  
323 - acc: 0.9616 - val\_loss: 0.1389 - val\_acc: 0.9566  
Epoch 5/20

```
60000/60000 [=====] - 4s 65us/step - loss: 0.1
047 - acc: 0.9701 - val_loss: 0.1176 - val_acc: 0.9650
Epoch 6/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
841 - acc: 0.9760 - val_loss: 0.1084 - val_acc: 0.9688
Epoch 7/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
686 - acc: 0.9797 - val_loss: 0.0899 - val_acc: 0.9714
Epoch 8/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
559 - acc: 0.9842 - val_loss: 0.0859 - val_acc: 0.9738
Epoch 9/20
60000/60000 [=====] - 4s 69us/step - loss: 0.0
465 - acc: 0.9864 - val_loss: 0.0804 - val_acc: 0.9760
Epoch 10/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
396 - acc: 0.9882 - val_loss: 0.0790 - val_acc: 0.9754
Epoch 11/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
307 - acc: 0.9914 - val_loss: 0.0744 - val_acc: 0.9776
Epoch 12/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
259 - acc: 0.9934 - val_loss: 0.0858 - val_acc: 0.9759
Epoch 13/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
226 - acc: 0.9936 - val_loss: 0.0813 - val_acc: 0.9766
Epoch 14/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
181 - acc: 0.9954 - val_loss: 0.0781 - val_acc: 0.9781
Epoch 15/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
148 - acc: 0.9960 - val_loss: 0.0789 - val_acc: 0.9767
Epoch 16/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
122 - acc: 0.9970 - val_loss: 0.0800 - val_acc: 0.9782
Epoch 17/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
087 - acc: 0.9982 - val_loss: 0.0871 - val_acc: 0.9769
Epoch 18/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
```

```
073 - acc: 0.9983 - val_loss: 0.0858 - val_acc: 0.9780
Epoch 19/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
086 - acc: 0.9979 - val_loss: 0.0973 - val_acc: 0.9759
Epoch 20/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
055 - acc: 0.9989 - val_loss: 0.0864 - val_acc: 0.9786
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

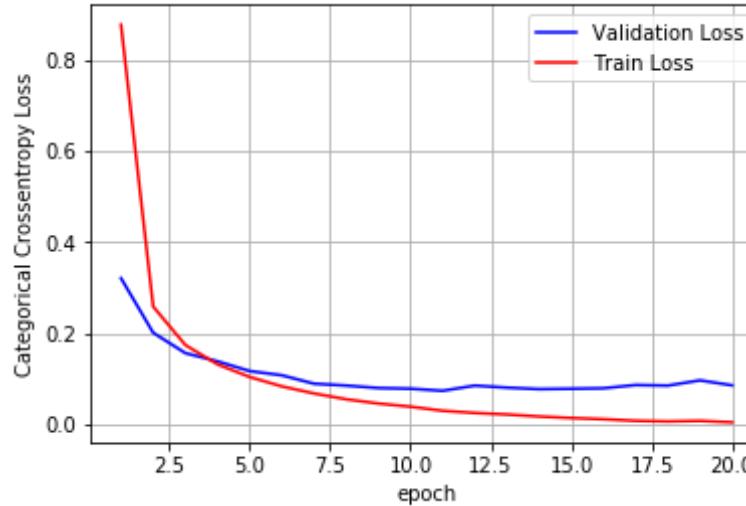
```
Test score: 0.08640015179301262
Test accuracy: 0.9786
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

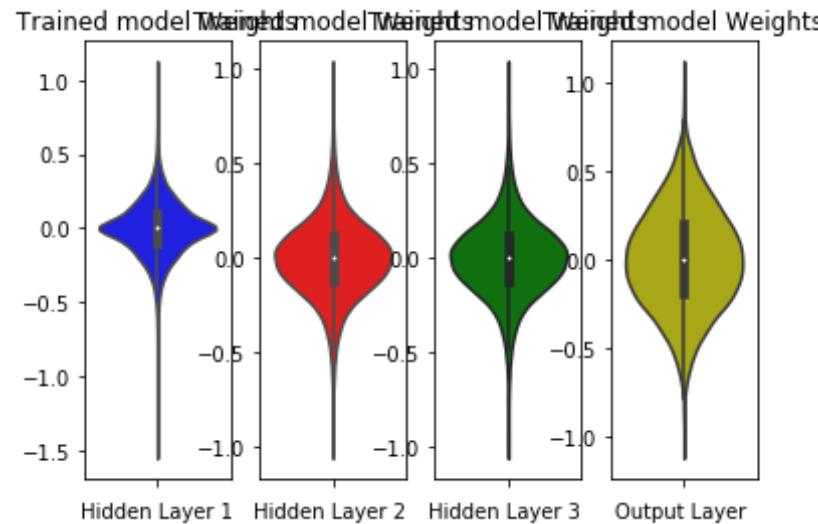
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : sigmoid activation + GradientDescentOptimizer with SGD Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo

```

```
    rmal(mean=0.0, stddev=0.102, seed=None)))  
  
    model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()  
  
Model: "sequential_45"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_128 (Dense)	(None, 256)	200960
dense_129 (Dense)	(None, 128)	32896
dense_130 (Dense)	(None, 64)	8256
dense_131 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 7s 113us/step - loss: 2.  
3099 - acc: 0.1117 - val_loss: 2.2968 - val_acc: 0.1135  
Epoch 2/20  
60000/60000 [=====] - 3s 54us/step - loss: 2.2  
953 - acc: 0.1162 - val_loss: 2.2928 - val_acc: 0.1330  
Epoch 3/20  
60000/60000 [=====] - 3s 53us/step - loss: 2.2  
909 - acc: 0.1320 - val_loss: 2.2887 - val_acc: 0.1135  
Epoch 4/20
```

```
60000/60000 [=====] - 3s 54us/step - loss: 2.2
863 - acc: 0.1271 - val_loss: 2.2830 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 3s 55us/step - loss: 2.2
811 - acc: 0.1346 - val_loss: 2.2775 - val_acc: 0.1838
Epoch 6/20
60000/60000 [=====] - 3s 54us/step - loss: 2.2
749 - acc: 0.1742 - val_loss: 2.2708 - val_acc: 0.1138
Epoch 7/20
60000/60000 [=====] - 3s 53us/step - loss: 2.2
678 - acc: 0.1795 - val_loss: 2.2627 - val_acc: 0.1875
Epoch 8/20
60000/60000 [=====] - 3s 55us/step - loss: 2.2
587 - acc: 0.2281 - val_loss: 2.2526 - val_acc: 0.2078
Epoch 9/20
60000/60000 [=====] - 3s 54us/step - loss: 2.2
475 - acc: 0.2674 - val_loss: 2.2395 - val_acc: 0.3304
Epoch 10/20
60000/60000 [=====] - 3s 54us/step - loss: 2.2
326 - acc: 0.3259 - val_loss: 2.2223 - val_acc: 0.3295
Epoch 11/20
60000/60000 [=====] - 3s 53us/step - loss: 2.2
124 - acc: 0.3704 - val_loss: 2.1989 - val_acc: 0.3255
Epoch 12/20
60000/60000 [=====] - 3s 53us/step - loss: 2.1
845 - acc: 0.4057 - val_loss: 2.1649 - val_acc: 0.3880
Epoch 13/20
60000/60000 [=====] - 3s 53us/step - loss: 2.1
444 - acc: 0.4356 - val_loss: 2.1167 - val_acc: 0.4742
Epoch 14/20
60000/60000 [=====] - 3s 54us/step - loss: 2.0
856 - acc: 0.4649 - val_loss: 2.0449 - val_acc: 0.4959
Epoch 15/20
60000/60000 [=====] - 3s 55us/step - loss: 1.9
996 - acc: 0.4912 - val_loss: 1.9418 - val_acc: 0.4750
Epoch 16/20
60000/60000 [=====] - 3s 54us/step - loss: 1.8
800 - acc: 0.5089 - val_loss: 1.8038 - val_acc: 0.5107
Epoch 17/20
60000/60000 [=====] - 3s 54us/step - loss: 1.7
```

```
304 - acc: 0.5324 - val_loss: 1.6435 - val_acc: 0.5348
Epoch 18/20
60000/60000 [=====] - 3s 53us/step - loss: 1.5
689 - acc: 0.5582 - val_loss: 1.4840 - val_acc: 0.5753
Epoch 19/20
60000/60000 [=====] - 3s 53us/step - loss: 1.4
178 - acc: 0.5870 - val_loss: 1.3438 - val_acc: 0.6069
Epoch 20/20
60000/60000 [=====] - 3s 54us/step - loss: 1.2
901 - acc: 0.6154 - val_loss: 1.2299 - val_acc: 0.6201
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

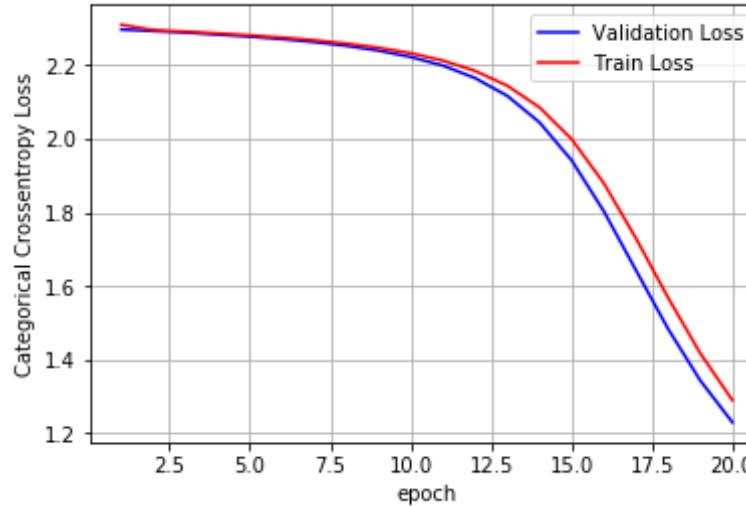
```
Test score: 1.229913805580139
Test accuracy: 0.6201
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

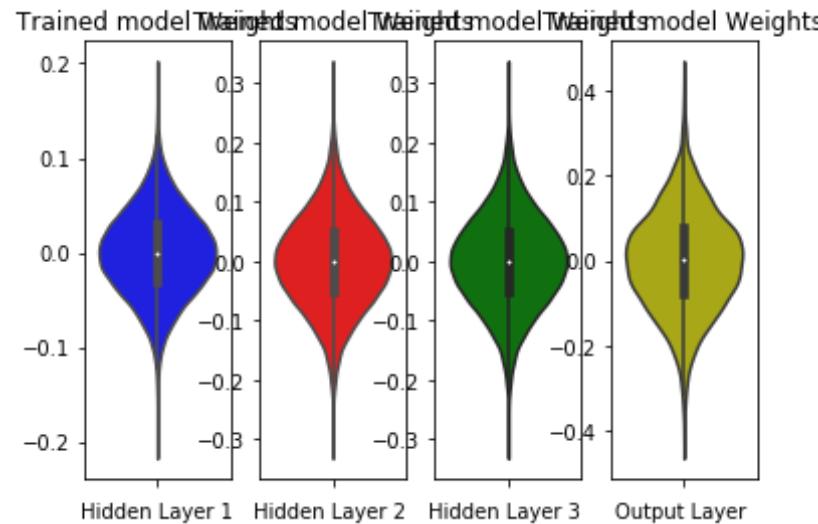
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : sigmoid activation + AdamOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))

```

```
model_1.add(Dense(output_dim, activation='softmax'))
```

```
model_1.summary()
```

Model: "sequential\_46"

Layer (type)	Output Shape	Param #
<hr/>		
dense_132 (Dense)	(None, 256)	200960
dense_133 (Dense)	(None, 128)	32896
dense_134 (Dense)	(None, 64)	8256
dense_135 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

In [0]: 

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/20

```
60000/60000 [=====] - 8s 127us/step - loss: 0.7385 - acc: 0.8276 - val_loss: 0.2796 - val_acc: 0.9212
```

Epoch 2/20

```
60000/60000 [=====] - 4s 65us/step - loss: 0.2314 - acc: 0.9343 - val_loss: 0.1906 - val_acc: 0.9441
```

Epoch 3/20

```
60000/60000 [=====] - 4s 66us/step - loss: 0.1616 - acc: 0.9536 - val_loss: 0.1458 - val_acc: 0.9572
```

Epoch 4/20

```
60000/60000 [=====] - 4s 65us/step - loss: 0.1
```

```
220 - acc: 0.9652 - val_loss: 0.1167 - val_acc: 0.9662
Epoch 5/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
949 - acc: 0.9723 - val_loss: 0.0996 - val_acc: 0.9705
Epoch 6/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
766 - acc: 0.9779 - val_loss: 0.0902 - val_acc: 0.9731
Epoch 7/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
627 - acc: 0.9817 - val_loss: 0.0860 - val_acc: 0.9742
Epoch 8/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
510 - acc: 0.9852 - val_loss: 0.0806 - val_acc: 0.9747
Epoch 9/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
416 - acc: 0.9883 - val_loss: 0.0754 - val_acc: 0.9769
Epoch 10/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
330 - acc: 0.9907 - val_loss: 0.0695 - val_acc: 0.9785
Epoch 11/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
287 - acc: 0.9918 - val_loss: 0.0728 - val_acc: 0.9784
Epoch 12/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
228 - acc: 0.9938 - val_loss: 0.0767 - val_acc: 0.9762
Epoch 13/20
60000/60000 [=====] - 4s 64us/step - loss: 0.0
182 - acc: 0.9956 - val_loss: 0.0685 - val_acc: 0.9800
Epoch 14/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
142 - acc: 0.9962 - val_loss: 0.0742 - val_acc: 0.9785
Epoch 15/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
122 - acc: 0.9970 - val_loss: 0.0688 - val_acc: 0.9805
Epoch 16/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
091 - acc: 0.9980 - val_loss: 0.0777 - val_acc: 0.9778
Epoch 17/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
084 - acc: 0.9981 - val_loss: 0.0733 - val_acc: 0.9802
```

```
Epoch 18/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
068 - acc: 0.9985 - val_loss: 0.0721 - val_acc: 0.9819
Epoch 19/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
051 - acc: 0.9990 - val_loss: 0.0720 - val_acc: 0.9808
Epoch 20/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
061 - acc: 0.9985 - val_loss: 0.0877 - val_acc: 0.9771
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

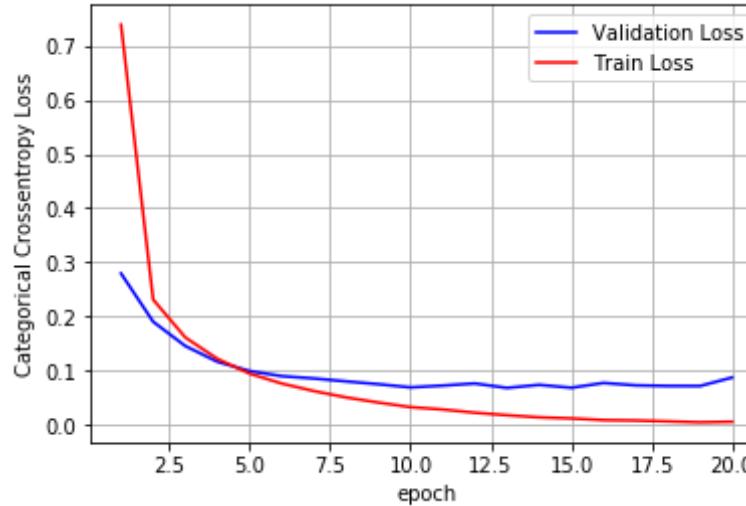
```
Test score: 0.08769395060807583
Test accuracy: 0.9771
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

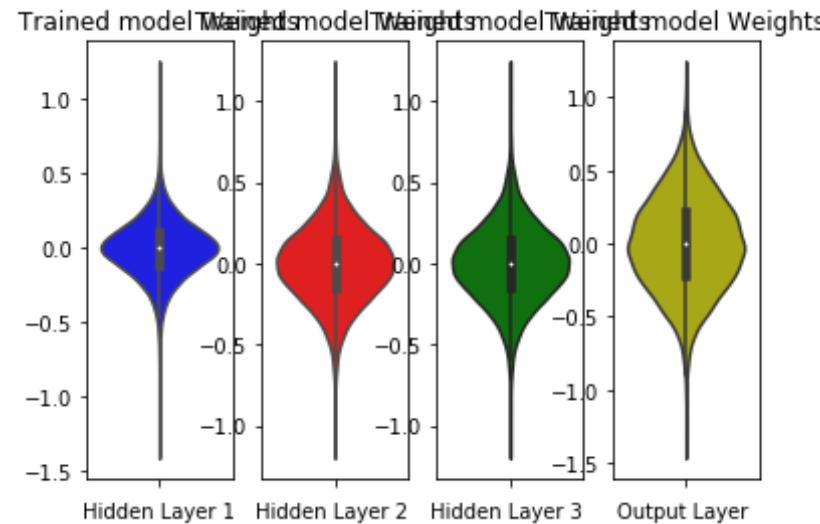
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : sigmoid activation + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo

```

```
    rmal(mean=0.0, stddev=0.175, seed=None)))  
  
    model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()  
  
Model: "sequential_47"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_136 (Dense)	(None, 256)	200960
dense_137 (Dense)	(None, 128)	32896
dense_138 (Dense)	(None, 64)	8256
dense_139 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 7s 117us/step - loss: 2.  
2985 - acc: 0.1428 - val_loss: 2.2770 - val_acc: 0.2314  
Epoch 2/20  
60000/60000 [=====] - 3s 56us/step - loss: 2.2  
641 - acc: 0.2536 - val_loss: 2.2480 - val_acc: 0.2746  
Epoch 3/20  
60000/60000 [=====] - 3s 55us/step - loss: 2.2  
321 - acc: 0.3406 - val_loss: 2.2109 - val_acc: 0.3880  
Epoch 4/20
```

```
60000/60000 [=====] - 3s 55us/step - loss: 2.1
887 - acc: 0.4152 - val_loss: 2.1580 - val_acc: 0.4814
Epoch 5/20
60000/60000 [=====] - 3s 56us/step - loss: 2.1
253 - acc: 0.4758 - val_loss: 2.0791 - val_acc: 0.5080
Epoch 6/20
60000/60000 [=====] - 3s 56us/step - loss: 2.0
311 - acc: 0.5068 - val_loss: 1.9640 - val_acc: 0.5120
Epoch 7/20
60000/60000 [=====] - 3s 54us/step - loss: 1.8
969 - acc: 0.5282 - val_loss: 1.8070 - val_acc: 0.5729
Epoch 8/20
60000/60000 [=====] - 3s 55us/step - loss: 1.7
277 - acc: 0.5593 - val_loss: 1.6251 - val_acc: 0.5898
Epoch 9/20
60000/60000 [=====] - 3s 55us/step - loss: 1.5
477 - acc: 0.5940 - val_loss: 1.4492 - val_acc: 0.6140
Epoch 10/20
60000/60000 [=====] - 3s 55us/step - loss: 1.3
847 - acc: 0.6326 - val_loss: 1.2990 - val_acc: 0.6525
Epoch 11/20
60000/60000 [=====] - 3s 56us/step - loss: 1.2
503 - acc: 0.6696 - val_loss: 1.1799 - val_acc: 0.6745
Epoch 12/20
60000/60000 [=====] - 3s 56us/step - loss: 1.1
435 - acc: 0.6981 - val_loss: 1.0844 - val_acc: 0.7256
Epoch 13/20
60000/60000 [=====] - 3s 55us/step - loss: 1.0
579 - acc: 0.7266 - val_loss: 1.0079 - val_acc: 0.7431
Epoch 14/20
60000/60000 [=====] - 3s 55us/step - loss: 0.9
873 - acc: 0.7461 - val_loss: 0.9431 - val_acc: 0.7652
Epoch 15/20
60000/60000 [=====] - 3s 55us/step - loss: 0.9
268 - acc: 0.7633 - val_loss: 0.8870 - val_acc: 0.7810
Epoch 16/20
60000/60000 [=====] - 3s 56us/step - loss: 0.8
733 - acc: 0.7769 - val_loss: 0.8363 - val_acc: 0.7926
Epoch 17/20
60000/60000 [=====] - 3s 55us/step - loss: 0.8
```

```
246 - acc: 0.7898 - val_loss: 0.7895 - val_acc: 0.8053
Epoch 18/20
60000/60000 [=====] - 3s 55us/step - loss: 0.7
798 - acc: 0.8022 - val_loss: 0.7471 - val_acc: 0.8171
Epoch 19/20
60000/60000 [=====] - 3s 55us/step - loss: 0.7
383 - acc: 0.8126 - val_loss: 0.7070 - val_acc: 0.8230
Epoch 20/20
60000/60000 [=====] - 3s 56us/step - loss: 0.6
999 - acc: 0.8226 - val_loss: 0.6695 - val_acc: 0.8342
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

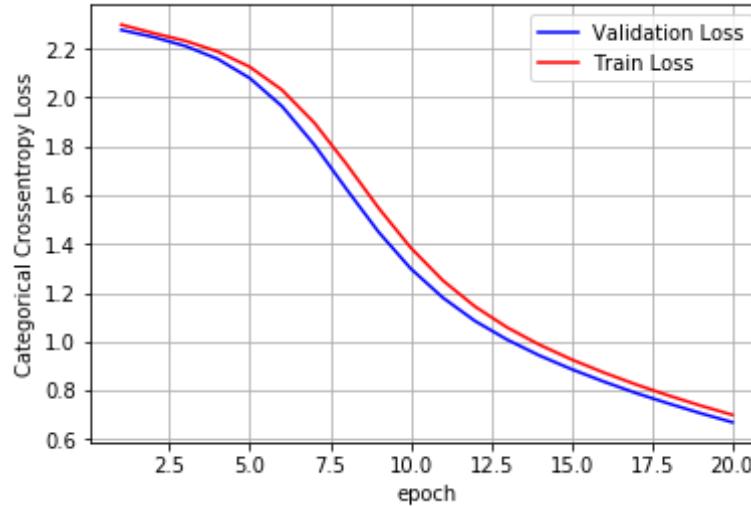
```
Test score: 0.669477147102356
Test accuracy: 0.8342
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

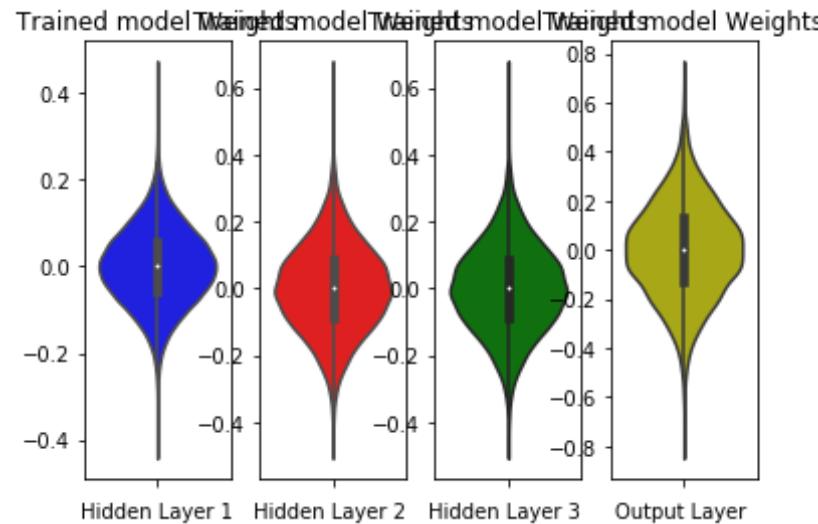
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## — Model 2: Relu activation + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))

```

```
model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

Model: "sequential_48"

Layer (type)          Output Shape         Param #
=====
```

Layer (type)	Output Shape	Param #
dense_140 (Dense)	(None, 256)	200960
dense_141 (Dense)	(None, 128)	32896
dense_142 (Dense)	(None, 64)	8256
dense_143 (Dense)	(None, 10)	650

```
Total params: 242,762
Trainable params: 242,762
Non-trainable params: 0
```

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 8s 129us/step - loss: 0.
2817 - acc: 0.9171 - val_loss: 0.1166 - val_acc: 0.9636
Epoch 2/20
60000/60000 [=====] - 4s 67us/step - loss: 0.1
032 - acc: 0.9687 - val_loss: 0.0982 - val_acc: 0.9691
Epoch 3/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
663 - acc: 0.9800 - val_loss: 0.0856 - val_acc: 0.9738
```

```
Epoch 4/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
497 - acc: 0.9842 - val_loss: 0.0852 - val_acc: 0.9749
Epoch 5/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
372 - acc: 0.9882 - val_loss: 0.0692 - val_acc: 0.9798
Epoch 6/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
305 - acc: 0.9899 - val_loss: 0.0849 - val_acc: 0.9768
Epoch 7/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
248 - acc: 0.9919 - val_loss: 0.0800 - val_acc: 0.9787
Epoch 8/20
60000/60000 [=====] - 4s 64us/step - loss: 0.0
200 - acc: 0.9935 - val_loss: 0.0797 - val_acc: 0.9796
Epoch 9/20
60000/60000 [=====] - 4s 64us/step - loss: 0.0
171 - acc: 0.9940 - val_loss: 0.0767 - val_acc: 0.9799
Epoch 10/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
177 - acc: 0.9940 - val_loss: 0.0880 - val_acc: 0.9766
Epoch 11/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
158 - acc: 0.9945 - val_loss: 0.0894 - val_acc: 0.9759
Epoch 12/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
137 - acc: 0.9952 - val_loss: 0.0951 - val_acc: 0.9790
Epoch 13/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
129 - acc: 0.9957 - val_loss: 0.0832 - val_acc: 0.9812
Epoch 14/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
105 - acc: 0.9964 - val_loss: 0.1152 - val_acc: 0.9769
Epoch 15/20
60000/60000 [=====] - 4s 70us/step - loss: 0.0
091 - acc: 0.9971 - val_loss: 0.0950 - val_acc: 0.9786
Epoch 16/20
60000/60000 [=====] - 4s 70us/step - loss: 0.0
089 - acc: 0.9972 - val_loss: 0.1186 - val_acc: 0.9759
Epoch 17/20
```

```
60000/60000 [=====] - 4s 68us/step - loss: 0.0
113 - acc: 0.9962 - val_loss: 0.0851 - val_acc: 0.9812
Epoch 18/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
111 - acc: 0.9964 - val_loss: 0.0889 - val_acc: 0.9809
Epoch 19/20
60000/60000 [=====] - 4s 65us/step - loss: 0.0
044 - acc: 0.9987 - val_loss: 0.1008 - val_acc: 0.9807
Epoch 20/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
123 - acc: 0.9960 - val_loss: 0.0992 - val_acc: 0.9788
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

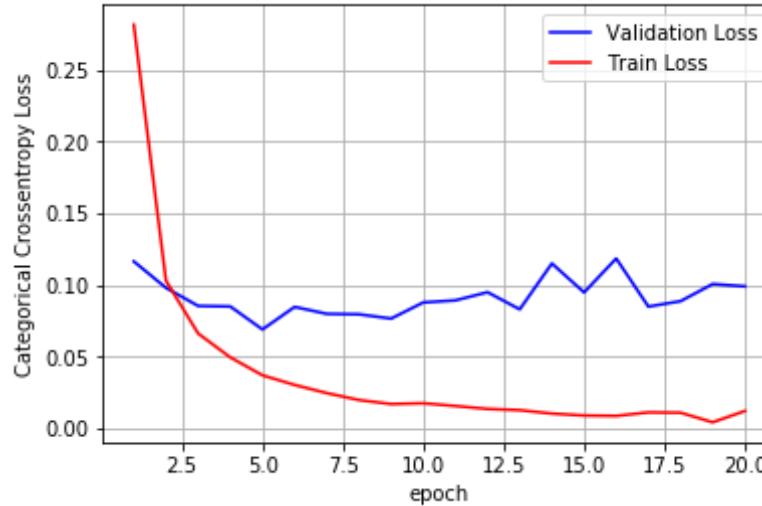
```
Test score: 0.09922539564829713
Test accuracy: 0.9788
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

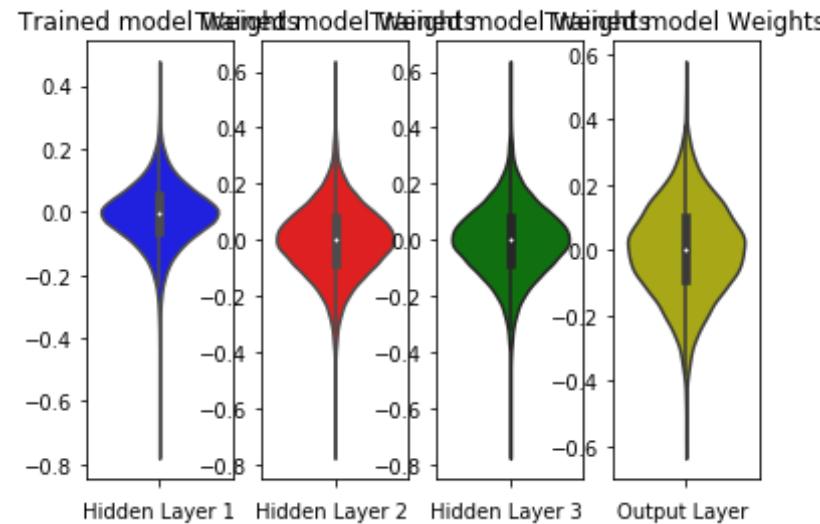
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 2 : relu activation + GradientDescentOptimizer with SGD Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(

```

```
l(mean=0.0, stddev=0.102, seed=None)))  
model_1.add(Dense(output_dim, activation='softmax'))  
model_1.summary()  
  
Model: "sequential_49"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_144 (Dense)	(None, 256)	200960
dense_145 (Dense)	(None, 128)	32896
dense_146 (Dense)	(None, 64)	8256
dense_147 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 7s 120us/step - loss: 1.  
3541 - acc: 0.6439 - val_loss: 0.5662 - val_acc: 0.8555  
Epoch 2/20  
60000/60000 [=====] - 3s 56us/step - loss: 0.4  
572 - acc: 0.8753 - val_loss: 0.3586 - val_acc: 0.8995  
Epoch 3/20  
60000/60000 [=====] - 3s 55us/step - loss: 0.3  
475 - acc: 0.9017 - val_loss: 0.3047 - val_acc: 0.9131  
Epoch 4/20
```

```
60000/60000 [=====] - 3s 57us/step - loss: 0.3
032 - acc: 0.9123 - val_loss: 0.2734 - val_acc: 0.9201
Epoch 5/20
60000/60000 [=====] - 3s 57us/step - loss: 0.2
749 - acc: 0.9211 - val_loss: 0.2541 - val_acc: 0.9261
Epoch 6/20
60000/60000 [=====] - 3s 56us/step - loss: 0.2
533 - acc: 0.9272 - val_loss: 0.2368 - val_acc: 0.9304
Epoch 7/20
60000/60000 [=====] - 3s 56us/step - loss: 0.2
359 - acc: 0.9321 - val_loss: 0.2216 - val_acc: 0.9362
Epoch 8/20
60000/60000 [=====] - 3s 58us/step - loss: 0.2
208 - acc: 0.9360 - val_loss: 0.2070 - val_acc: 0.9387
Epoch 9/20
60000/60000 [=====] - 3s 55us/step - loss: 0.2
080 - acc: 0.9406 - val_loss: 0.1952 - val_acc: 0.9423
Epoch 10/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
959 - acc: 0.9436 - val_loss: 0.1884 - val_acc: 0.9441
Epoch 11/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
858 - acc: 0.9468 - val_loss: 0.1792 - val_acc: 0.9465
Epoch 12/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
764 - acc: 0.9491 - val_loss: 0.1703 - val_acc: 0.9502
Epoch 13/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
676 - acc: 0.9518 - val_loss: 0.1668 - val_acc: 0.9505
Epoch 14/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
598 - acc: 0.9539 - val_loss: 0.1592 - val_acc: 0.9529
Epoch 15/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
525 - acc: 0.9560 - val_loss: 0.1549 - val_acc: 0.9546
Epoch 16/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
459 - acc: 0.9579 - val_loss: 0.1460 - val_acc: 0.9574
Epoch 17/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
```

```
395 - acc: 0.9601 - val_loss: 0.1412 - val_acc: 0.9583
Epoch 18/20
60000/60000 [=====] - 3s 57us/step - loss: 0.1
341 - acc: 0.9617 - val_loss: 0.1365 - val_acc: 0.9600
Epoch 19/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
288 - acc: 0.9628 - val_loss: 0.1356 - val_acc: 0.9601
Epoch 20/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
235 - acc: 0.9645 - val_loss: 0.1292 - val_acc: 0.9612
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

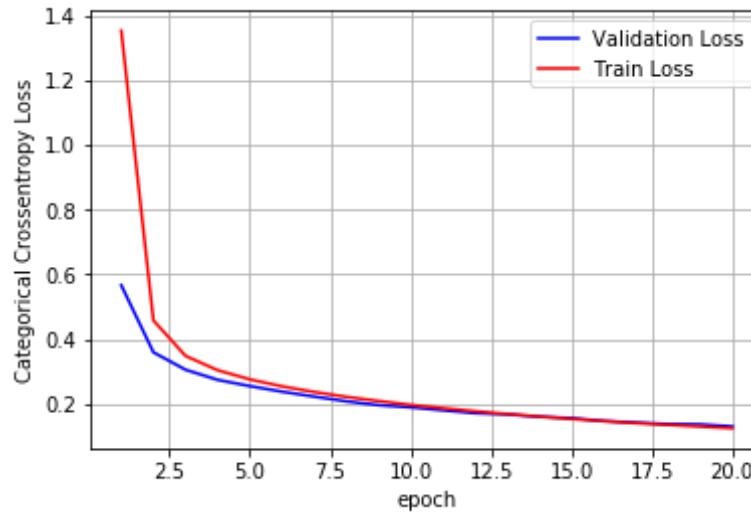
```
Test score: 0.1292183303233236
Test accuracy: 0.9612
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

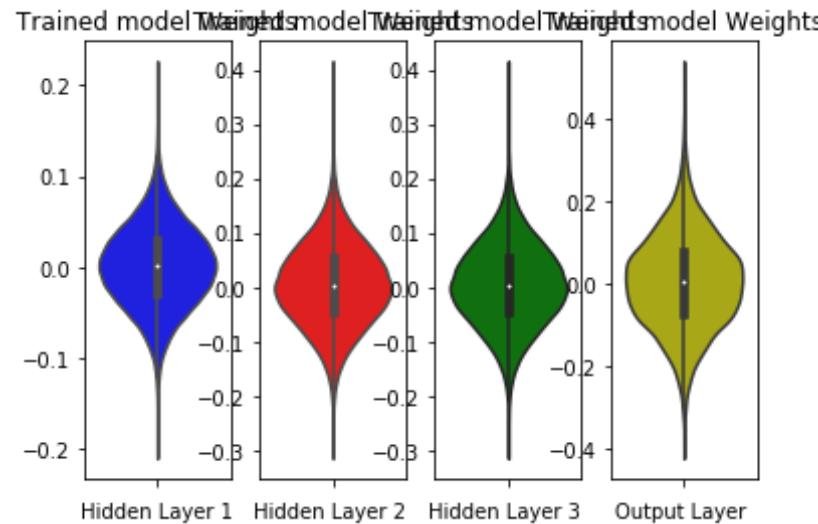
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : relu activation + AdamOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))

```

```
model_1.add(Dense(output_dim, activation='softmax'))
```

```
model_1.summary()
```

Model: "sequential\_50"

Layer (type)	Output Shape	Param #
<hr/>		
dense_148 (Dense)	(None, 256)	200960
dense_149 (Dense)	(None, 128)	32896
dense_150 (Dense)	(None, 64)	8256
dense_151 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

In [0]: 

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/20

```
60000/60000 [=====] - 8s 134us/step - loss: 0.
2832 - acc: 0.9147 - val_loss: 0.1477 - val_acc: 0.9520
```

Epoch 2/20

```
60000/60000 [=====] - 4s 66us/step - loss: 0.1
029 - acc: 0.9688 - val_loss: 0.0940 - val_acc: 0.9712
```

Epoch 3/20

```
60000/60000 [=====] - 4s 67us/step - loss: 0.0
679 - acc: 0.9786 - val_loss: 0.0928 - val_acc: 0.9710
```

Epoch 4/20

```
60000/60000 [=====] - 4s 68us/step - loss: 0.0
```

```
479 - acc: 0.9846 - val_loss: 0.0804 - val_acc: 0.9756
Epoch 5/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
338 - acc: 0.9892 - val_loss: 0.0970 - val_acc: 0.9723
Epoch 6/20
60000/60000 [=====] - 4s 70us/step - loss: 0.0
297 - acc: 0.9902 - val_loss: 0.0848 - val_acc: 0.9753
Epoch 7/20
60000/60000 [=====] - 4s 69us/step - loss: 0.0
200 - acc: 0.9935 - val_loss: 0.0804 - val_acc: 0.9772
Epoch 8/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
196 - acc: 0.9933 - val_loss: 0.1032 - val_acc: 0.9710
Epoch 9/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
205 - acc: 0.9932 - val_loss: 0.0952 - val_acc: 0.9776
Epoch 10/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
154 - acc: 0.9947 - val_loss: 0.0890 - val_acc: 0.9779
Epoch 11/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
120 - acc: 0.9961 - val_loss: 0.0909 - val_acc: 0.9779
Epoch 12/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
138 - acc: 0.9951 - val_loss: 0.0926 - val_acc: 0.9784
Epoch 13/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
125 - acc: 0.9960 - val_loss: 0.1222 - val_acc: 0.9739
Epoch 14/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
138 - acc: 0.9955 - val_loss: 0.1141 - val_acc: 0.9760
Epoch 15/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
116 - acc: 0.9961 - val_loss: 0.1070 - val_acc: 0.9783
Epoch 16/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
126 - acc: 0.9957 - val_loss: 0.1110 - val_acc: 0.9773
Epoch 17/20
60000/60000 [=====] - 4s 68us/step - loss: 0.0
139 - acc: 0.9953 - val_loss: 0.1146 - val_acc: 0.9758
```

```
Epoch 18/20
60000/60000 [=====] - 4s 70us/step - loss: 0.0
081 - acc: 0.9970 - val_loss: 0.1038 - val_acc: 0.9801
Epoch 19/20
60000/60000 [=====] - 4s 67us/step - loss: 0.0
088 - acc: 0.9971 - val_loss: 0.1017 - val_acc: 0.9787
Epoch 20/20
60000/60000 [=====] - 4s 66us/step - loss: 0.0
071 - acc: 0.9977 - val_loss: 0.1020 - val_acc: 0.9799
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

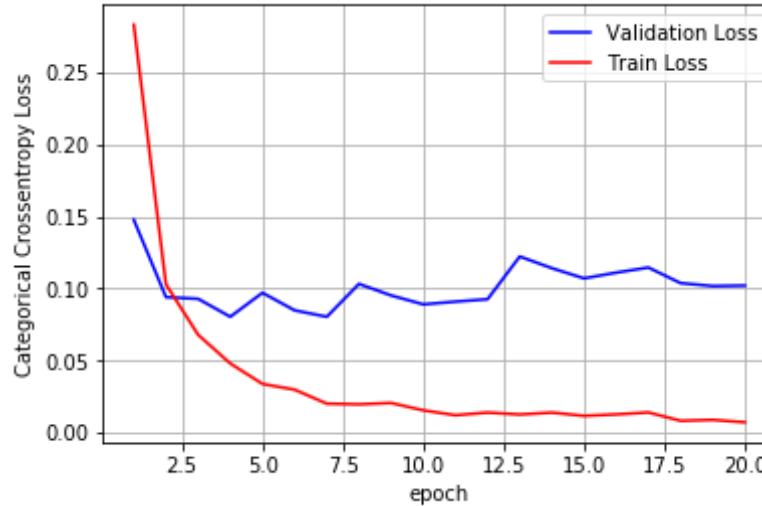
```
Test score: 0.10201168950064675
Test accuracy: 0.9799
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

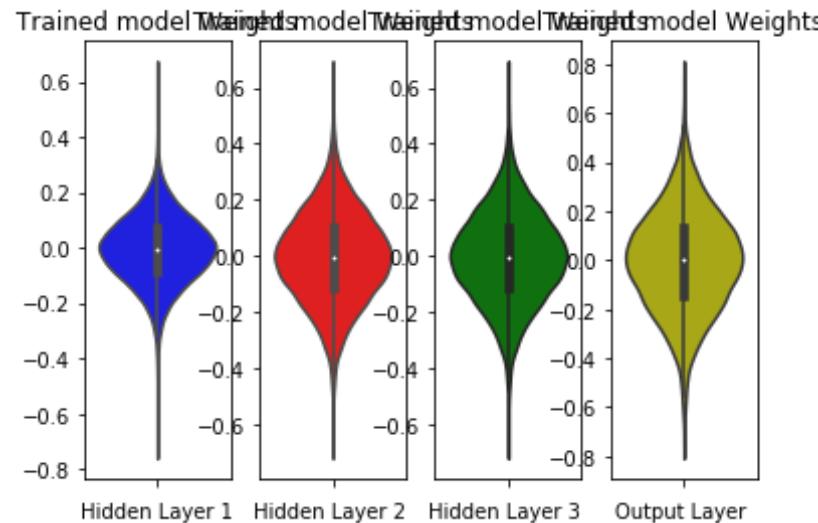
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 2 : relu activation + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))

```

```
model_1.add(Dense(output_dim, activation='softmax'))
```

```
model_1.summary()
```

Model: "sequential\_51"

Layer (type)	Output Shape	Param #
<hr/>		
dense_152 (Dense)	(None, 256)	200960
dense_153 (Dense)	(None, 128)	32896
dense_154 (Dense)	(None, 64)	8256
dense_155 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/20

```
60000/60000 [=====] - 7s 123us/step - loss: 0.7937 - acc: 0.7617 - val_loss: 0.3894 - val_acc: 0.8821
```

Epoch 2/20

```
60000/60000 [=====] - 3s 56us/step - loss: 0.3569 - acc: 0.8937 - val_loss: 0.2971 - val_acc: 0.9112
```

Epoch 3/20

```
60000/60000 [=====] - 3s 56us/step - loss: 0.2892 - acc: 0.9139 - val_loss: 0.2571 - val_acc: 0.9245
```

Epoch 4/20

```
60000/60000 [=====] - 3s 55us/step - loss: 0.2
```

```
524 - acc: 0.9249 - val_loss: 0.2282 - val_acc: 0.9327
Epoch 5/20
60000/60000 [=====] - 3s 56us/step - loss: 0.2
269 - acc: 0.9331 - val_loss: 0.2142 - val_acc: 0.9363
Epoch 6/20
60000/60000 [=====] - 3s 57us/step - loss: 0.2
074 - acc: 0.9388 - val_loss: 0.1967 - val_acc: 0.9404
Epoch 7/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
923 - acc: 0.9434 - val_loss: 0.1869 - val_acc: 0.9435
Epoch 8/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
790 - acc: 0.9473 - val_loss: 0.1753 - val_acc: 0.9479
Epoch 9/20
60000/60000 [=====] - 3s 57us/step - loss: 0.1
680 - acc: 0.9511 - val_loss: 0.1684 - val_acc: 0.9502
Epoch 10/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
584 - acc: 0.9537 - val_loss: 0.1612 - val_acc: 0.9504
Epoch 11/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
499 - acc: 0.9563 - val_loss: 0.1551 - val_acc: 0.9533
Epoch 12/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
424 - acc: 0.9587 - val_loss: 0.1501 - val_acc: 0.9535
Epoch 13/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
354 - acc: 0.9607 - val_loss: 0.1449 - val_acc: 0.9548
Epoch 14/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
292 - acc: 0.9625 - val_loss: 0.1443 - val_acc: 0.9550
Epoch 15/20
60000/60000 [=====] - 3s 55us/step - loss: 0.1
234 - acc: 0.9640 - val_loss: 0.1369 - val_acc: 0.9587
Epoch 16/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
183 - acc: 0.9663 - val_loss: 0.1333 - val_acc: 0.9586
Epoch 17/20
60000/60000 [=====] - 3s 57us/step - loss: 0.1
130 - acc: 0.9675 - val_loss: 0.1314 - val_acc: 0.9592
```

```
Epoch 18/20
60000/60000 [=====] - 3s 57us/step - loss: 0.1
089 - acc: 0.9686 - val_loss: 0.1282 - val_acc: 0.9592
Epoch 19/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
048 - acc: 0.9700 - val_loss: 0.1271 - val_acc: 0.9600
Epoch 20/20
60000/60000 [=====] - 3s 56us/step - loss: 0.1
009 - acc: 0.9712 - val_loss: 0.1257 - val_acc: 0.9617
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

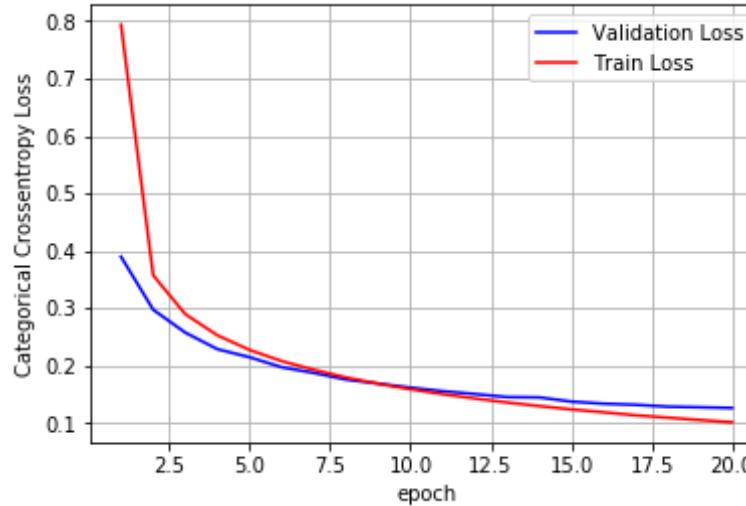
```
Test score: 0.12574072407316417
Test accuracy: 0.9617
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

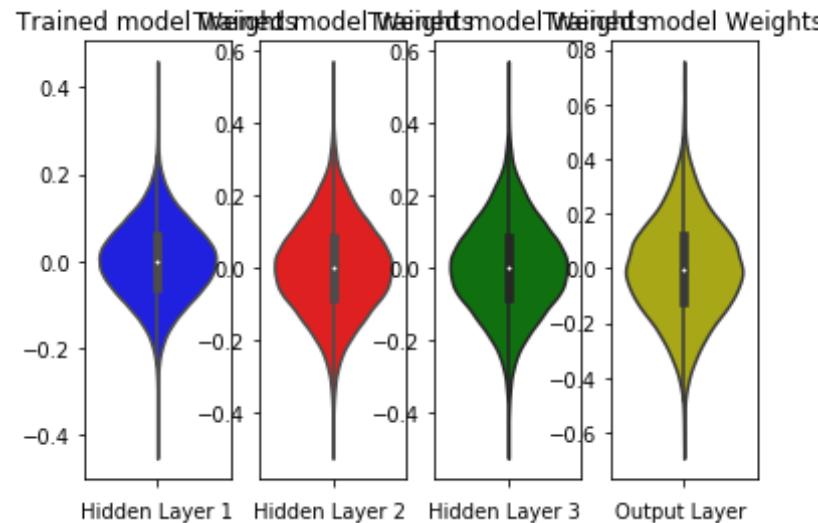
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN

```

```
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_52"

Layer (type)	Output Shape	Param #
dense_156 (Dense)	(None, 256)	200960
batch_normalization_31 (BatchNormalization)	(None, 256)	1024
dense_157 (Dense)	(None, 128)	32896
batch_normalization_32 (BatchNormalization)	(None, 128)	512
dense_158 (Dense)	(None, 64)	8256
batch_normalization_33 (BatchNormalization)	(None, 64)	256
dense_159 (Dense)	(None, 10)	650

Total params: 244,554

Trainable params: 243,658

Non-trainable params: 896

---

In [0]:

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 10s 166us/step - loss: 0.2908 - acc: 0.9151 - val_loss: 0.1803 - val_acc: 0.9445
Epoch 2/20
60000/60000 [=====] - 5s 84us/step - loss: 0.1422 - acc: 0.9582 - val_loss: 0.1245 - val_acc: 0.9623
Epoch 3/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0958 - acc: 0.9706 - val_loss: 0.0965 - val_acc: 0.9691
Epoch 4/20
60000/60000 [=====] - 5s 83us/step - loss: 0.0709 - acc: 0.9781 - val_loss: 0.0969 - val_acc: 0.9685
Epoch 5/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0545 - acc: 0.9828 - val_loss: 0.0946 - val_acc: 0.9705
Epoch 6/20
60000/60000 [=====] - 5s 84us/step - loss: 0.0430 - acc: 0.9860 - val_loss: 0.0905 - val_acc: 0.9718
Epoch 7/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0372 - acc: 0.9879 - val_loss: 0.0897 - val_acc: 0.9739
Epoch 8/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0291 - acc: 0.9907 - val_loss: 0.0845 - val_acc: 0.9747
Epoch 9/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0263 - acc: 0.9913 - val_loss: 0.0838 - val_acc: 0.9758
Epoch 10/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0228 - acc: 0.9926 - val_loss: 0.0919 - val_acc: 0.9742
Epoch 11/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0209 - acc: 0.9930 - val_loss: 0.0880 - val_acc: 0.9744
Epoch 12/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0176 - acc: 0.9942 - val_loss: 0.0934 - val_acc: 0.9741
Epoch 13/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0154 - acc: 0.9949 - val_loss: 0.0993 - val_acc: 0.9754
```

```
Epoch 14/20
60000/60000 [=====] - 5s 84us/step - loss: 0.0
157 - acc: 0.9946 - val_loss: 0.0940 - val_acc: 0.9755
Epoch 15/20
60000/60000 [=====] - 5s 82us/step - loss: 0.0
147 - acc: 0.9950 - val_loss: 0.0937 - val_acc: 0.9754
Epoch 16/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
119 - acc: 0.9960 - val_loss: 0.0997 - val_acc: 0.9748
Epoch 17/20
60000/60000 [=====] - 5s 84us/step - loss: 0.0
113 - acc: 0.9964 - val_loss: 0.1011 - val_acc: 0.9757
Epoch 18/20
60000/60000 [=====] - 5s 84us/step - loss: 0.0
121 - acc: 0.9959 - val_loss: 0.0972 - val_acc: 0.9755
Epoch 19/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
124 - acc: 0.9957 - val_loss: 0.0937 - val_acc: 0.9768
Epoch 20/20
60000/60000 [=====] - 5s 83us/step - loss: 0.0
093 - acc: 0.9968 - val_loss: 0.0933 - val_acc: 0.9786
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

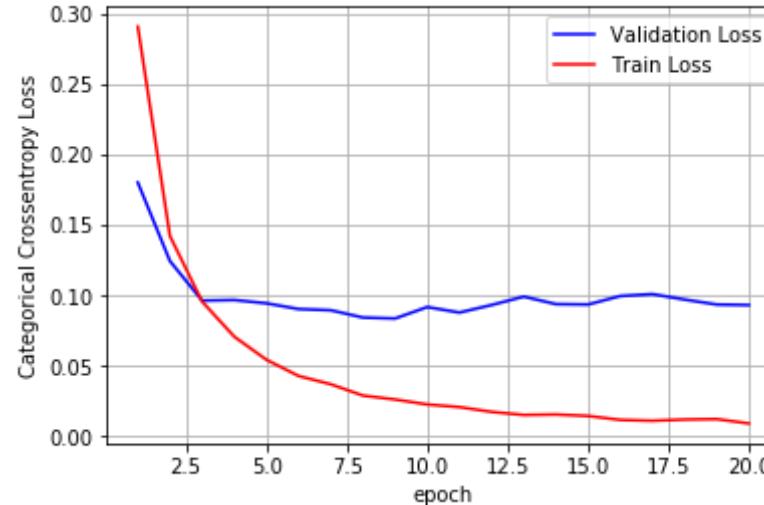
```
Test score: 0.09331667342400396
Test accuracy: 0.9786
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

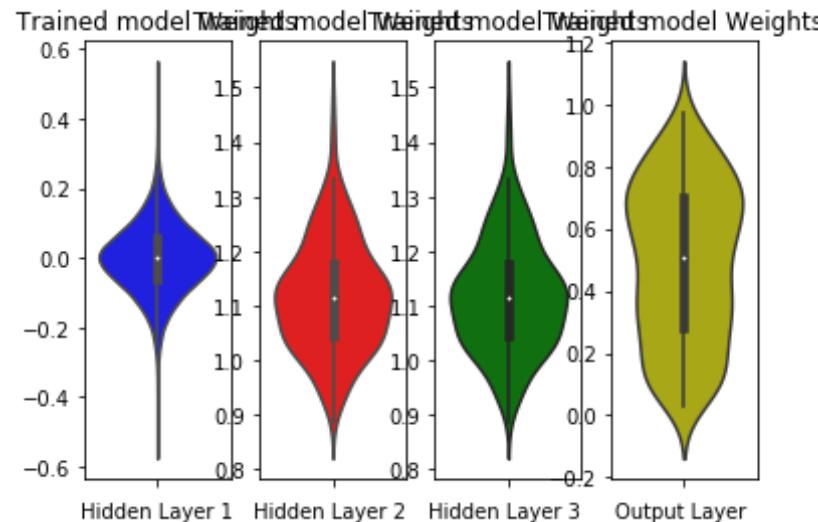
plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with SGD Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN

```

```
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_53"

Layer (type)	Output Shape	Param #
dense_160 (Dense)	(None, 256)	200960
batch_normalization_34 (Batch Normalization)	(None, 256)	1024
dense_161 (Dense)	(None, 128)	32896
batch_normalization_35 (Batch Normalization)	(None, 128)	512
dense_162 (Dense)	(None, 64)	8256
batch_normalization_36 (Batch Normalization)	(None, 64)	256
dense_163 (Dense)	(None, 10)	650

Total params: 244,554

Trainable params: 243,658

Non-trainable params: 896

---

In [0]:

```
model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 152us/step - loss: 0.
5440 - acc: 0.8426 - val_loss: 0.3466 - val_acc: 0.9038
Epoch 2/20
60000/60000 [=====] - 4s 72us/step - loss: 0.3
401 - acc: 0.9034 - val_loss: 0.2991 - val_acc: 0.9161
Epoch 3/20
60000/60000 [=====] - 4s 73us/step - loss: 0.3
022 - acc: 0.9135 - val_loss: 0.2760 - val_acc: 0.9198
Epoch 4/20
60000/60000 [=====] - 4s 74us/step - loss: 0.2
769 - acc: 0.9208 - val_loss: 0.2611 - val_acc: 0.9248
Epoch 5/20
60000/60000 [=====] - 4s 72us/step - loss: 0.2
596 - acc: 0.9264 - val_loss: 0.2501 - val_acc: 0.9288
Epoch 6/20
60000/60000 [=====] - 4s 73us/step - loss: 0.2
433 - acc: 0.9311 - val_loss: 0.2376 - val_acc: 0.9330
Epoch 7/20
60000/60000 [=====] - 4s 71us/step - loss: 0.2
296 - acc: 0.9355 - val_loss: 0.2248 - val_acc: 0.9368
Epoch 8/20
60000/60000 [=====] - 4s 71us/step - loss: 0.2
181 - acc: 0.9384 - val_loss: 0.2161 - val_acc: 0.9383
Epoch 9/20
60000/60000 [=====] - 4s 72us/step - loss: 0.2
043 - acc: 0.9419 - val_loss: 0.2059 - val_acc: 0.9412
Epoch 10/20
60000/60000 [=====] - 4s 72us/step - loss: 0.1
939 - acc: 0.9459 - val_loss: 0.1952 - val_acc: 0.9434
Epoch 11/20
60000/60000 [=====] - 4s 73us/step - loss: 0.1
817 - acc: 0.9486 - val_loss: 0.1868 - val_acc: 0.9457
Epoch 12/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
734 - acc: 0.9515 - val_loss: 0.1798 - val_acc: 0.9482
Epoch 13/20
60000/60000 [=====] - 4s 71us/step - loss: 0.1
640 - acc: 0.9533 - val_loss: 0.1720 - val_acc: 0.9496
```

```
Epoch 14/20
60000/60000 [=====] - 4s 73us/step - loss: 0.1
556 - acc: 0.9554 - val_loss: 0.1659 - val_acc: 0.9505
Epoch 15/20
60000/60000 [=====] - 4s 72us/step - loss: 0.1
480 - acc: 0.9581 - val_loss: 0.1575 - val_acc: 0.9544
Epoch 16/20
60000/60000 [=====] - 4s 71us/step - loss: 0.1
403 - acc: 0.9604 - val_loss: 0.1536 - val_acc: 0.9548
Epoch 17/20
60000/60000 [=====] - 4s 70us/step - loss: 0.1
334 - acc: 0.9625 - val_loss: 0.1484 - val_acc: 0.9558
Epoch 18/20
60000/60000 [=====] - 4s 71us/step - loss: 0.1
261 - acc: 0.9644 - val_loss: 0.1434 - val_acc: 0.9567
Epoch 19/20
60000/60000 [=====] - 4s 71us/step - loss: 0.1
206 - acc: 0.9656 - val_loss: 0.1362 - val_acc: 0.9595
Epoch 20/20
60000/60000 [=====] - 4s 70us/step - loss: 0.1
139 - acc: 0.9677 - val_loss: 0.1359 - val_acc: 0.9598
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

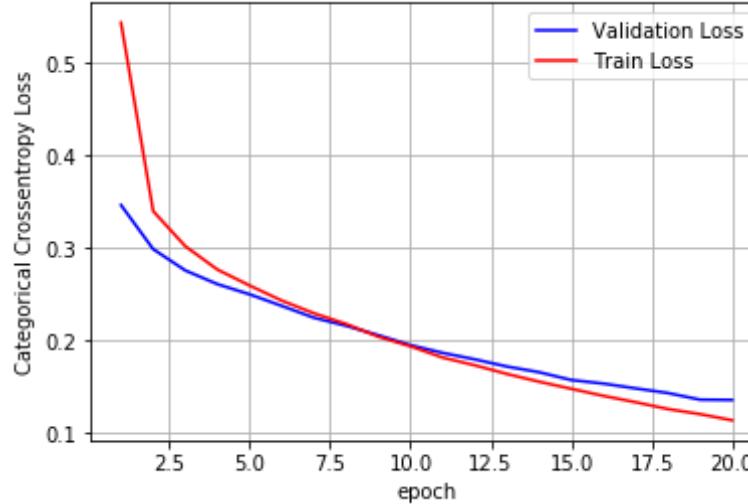
```
Test score: 0.1358664358958602
Test accuracy: 0.9598
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

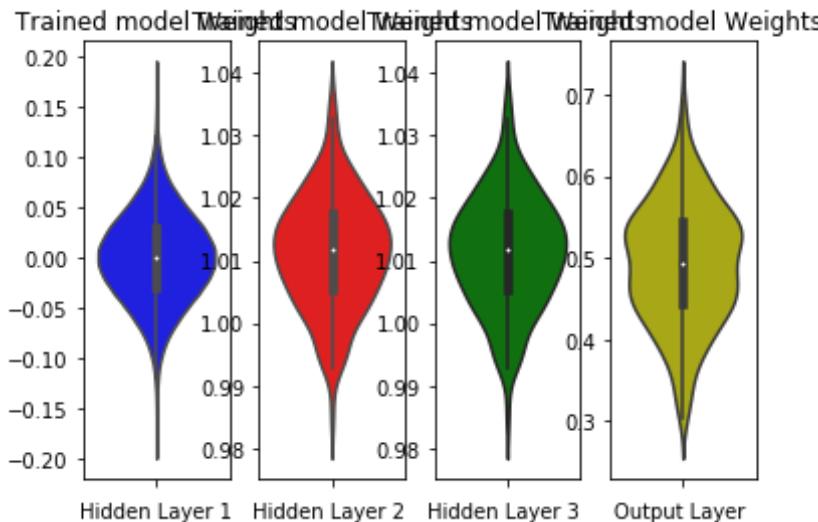
plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()

```



Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with Relu Weight</h3>

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

```

```
model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_54"

Layer (type)	Output Shape	Param #
dense_164 (Dense)	(None, 256)	200960
batch_normalization_37 (BatchNormalization)	(None, 256)	1024
dense_165 (Dense)	(None, 128)	32896
batch_normalization_38 (BatchNormalization)	(None, 128)	512
dense_166 (Dense)	(None, 64)	8256
batch_normalization_39 (BatchNormalization)	(None, 64)	256
dense_167 (Dense)	(None, 10)	650
<hr/>		
Total params: 244,554		
Trainable params: 243,658		
Non-trainable params: 896		

In [0]:

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20

```
60000/60000 [=====] - 10s 168us/step - loss:  
0.2980 - acc: 0.9131 - val_loss: 0.1682 - val_acc: 0.9521  
Epoch 2/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.1  
335 - acc: 0.9607 - val_loss: 0.1217 - val_acc: 0.9645  
Epoch 3/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0  
851 - acc: 0.9739 - val_loss: 0.0975 - val_acc: 0.9687  
Epoch 4/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0  
600 - acc: 0.9816 - val_loss: 0.0904 - val_acc: 0.9708  
Epoch 5/20  
60000/60000 [=====] - 5s 86us/step - loss: 0.0  
414 - acc: 0.9868 - val_loss: 0.0890 - val_acc: 0.9722  
Epoch 6/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0  
310 - acc: 0.9905 - val_loss: 0.0876 - val_acc: 0.9740  
Epoch 7/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.0  
269 - acc: 0.9917 - val_loss: 0.0847 - val_acc: 0.9755  
Epoch 8/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.0  
220 - acc: 0.9932 - val_loss: 0.0886 - val_acc: 0.9756  
Epoch 9/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0  
174 - acc: 0.9947 - val_loss: 0.0845 - val_acc: 0.9764  
Epoch 10/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.0  
146 - acc: 0.9954 - val_loss: 0.0976 - val_acc: 0.9745  
Epoch 11/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.0  
152 - acc: 0.9952 - val_loss: 0.0922 - val_acc: 0.9757  
Epoch 12/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0  
130 - acc: 0.9958 - val_loss: 0.0865 - val_acc: 0.9768  
Epoch 13/20  
60000/60000 [=====] - 5s 84us/step - loss: 0.0  
113 - acc: 0.9963 - val_loss: 0.0845 - val_acc: 0.9769  
Epoch 14/20  
60000/60000 [=====] - 5s 85us/step - loss: 0.0
```

```
114 - acc: 0.9961 - val_loss: 0.0988 - val_acc: 0.9758
Epoch 15/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
114 - acc: 0.9963 - val_loss: 0.0873 - val_acc: 0.9784
Epoch 16/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
116 - acc: 0.9962 - val_loss: 0.0886 - val_acc: 0.9780
Epoch 17/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
093 - acc: 0.9968 - val_loss: 0.0905 - val_acc: 0.9773
Epoch 18/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
083 - acc: 0.9972 - val_loss: 0.0876 - val_acc: 0.9796
Epoch 19/20
60000/60000 [=====] - 5s 84us/step - loss: 0.0
060 - acc: 0.9981 - val_loss: 0.0822 - val_acc: 0.9804
Epoch 20/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
107 - acc: 0.9964 - val_loss: 0.0987 - val_acc: 0.9785
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

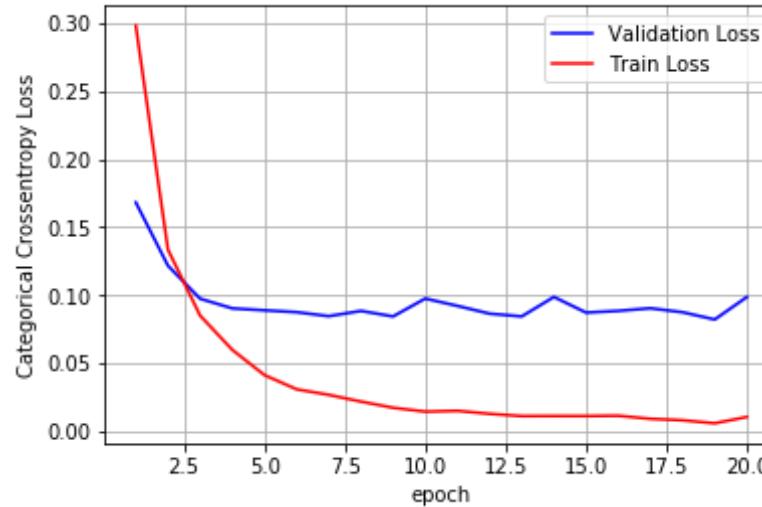
```
Test score: 0.09873631281140514
Test accuracy: 0.9785
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

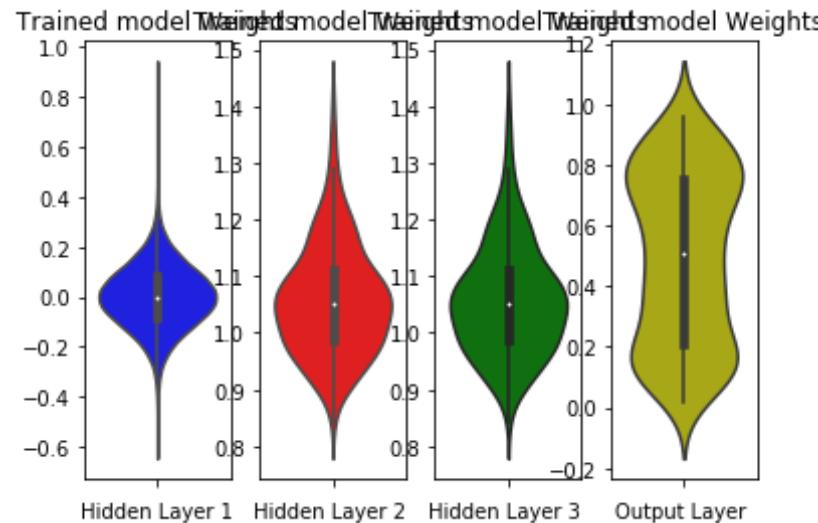
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

```

```
model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_55"

Layer (type)	Output Shape	Param #
dense_168 (Dense)	(None, 256)	200960
batch_normalization_40 (BatchNormalization)	(None, 256)	1024
dense_169 (Dense)	(None, 128)	32896
batch_normalization_41 (BatchNormalization)	(None, 128)	512
dense_170 (Dense)	(None, 64)	8256
batch_normalization_42 (BatchNormalization)	(None, 64)	256
dense_171 (Dense)	(None, 10)	650
<hr/>		
Total params: 244,554		
Trainable params: 243,658		
Non-trainable params: 896		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20

```
60000/60000 [=====] - 9s 158us/step - loss: 0.  
7626 - acc: 0.7702 - val_loss: 0.4319 - val_acc: 0.8769  
Epoch 2/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.4  
071 - acc: 0.8838 - val_loss: 0.3441 - val_acc: 0.9006  
Epoch 3/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.3  
439 - acc: 0.9008 - val_loss: 0.3051 - val_acc: 0.9110  
Epoch 4/20  
60000/60000 [=====] - 4s 72us/step - loss: 0.3  
082 - acc: 0.9106 - val_loss: 0.2811 - val_acc: 0.9187  
Epoch 5/20  
60000/60000 [=====] - 4s 74us/step - loss: 0.2  
837 - acc: 0.9188 - val_loss: 0.2628 - val_acc: 0.9237  
Epoch 6/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.2  
650 - acc: 0.9235 - val_loss: 0.2489 - val_acc: 0.9269  
Epoch 7/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.2  
498 - acc: 0.9277 - val_loss: 0.2363 - val_acc: 0.9308  
Epoch 8/20  
60000/60000 [=====] - 4s 74us/step - loss: 0.2  
372 - acc: 0.9322 - val_loss: 0.2262 - val_acc: 0.9332  
Epoch 9/20  
60000/60000 [=====] - 5s 78us/step - loss: 0.2  
252 - acc: 0.9355 - val_loss: 0.2177 - val_acc: 0.9358  
Epoch 10/20  
60000/60000 [=====] - 5s 78us/step - loss: 0.2  
157 - acc: 0.9376 - val_loss: 0.2088 - val_acc: 0.9380  
Epoch 11/20  
60000/60000 [=====] - 4s 74us/step - loss: 0.2  
047 - acc: 0.9408 - val_loss: 0.2012 - val_acc: 0.9391  
Epoch 12/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.1  
981 - acc: 0.9427 - val_loss: 0.1945 - val_acc: 0.9406  
Epoch 13/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.1  
893 - acc: 0.9452 - val_loss: 0.1892 - val_acc: 0.9422  
Epoch 14/20  
60000/60000 [=====] - 4s 73us/step - loss: 0.1
```

```
822 - acc: 0.9470 - val_loss: 0.1839 - val_acc: 0.9437
Epoch 15/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
763 - acc: 0.9498 - val_loss: 0.1786 - val_acc: 0.9449
Epoch 16/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
696 - acc: 0.9515 - val_loss: 0.1743 - val_acc: 0.9472
Epoch 17/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
644 - acc: 0.9523 - val_loss: 0.1697 - val_acc: 0.9482
Epoch 18/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
584 - acc: 0.9554 - val_loss: 0.1656 - val_acc: 0.9497
Epoch 19/20
60000/60000 [=====] - 4s 73us/step - loss: 0.1
531 - acc: 0.9562 - val_loss: 0.1624 - val_acc: 0.9516
Epoch 20/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
480 - acc: 0.9573 - val_loss: 0.1581 - val_acc: 0.9524
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

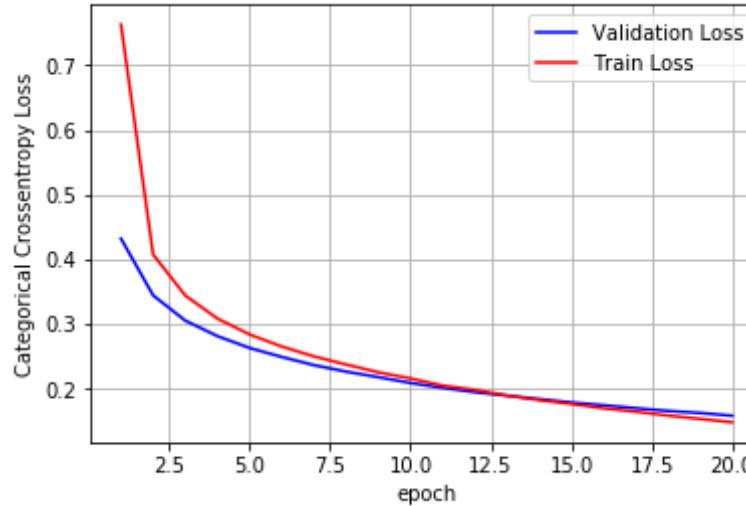
```
Test score: 0.15813388714268803
Test accuracy: 0.9524
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

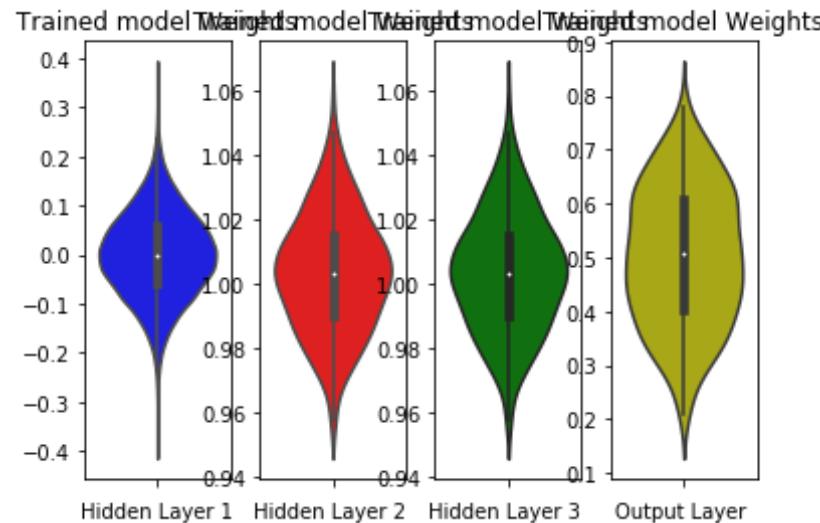
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 4: relu activation with Dropout + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(Dropout(0.5))

```

```
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_56"

Layer (type)	Output Shape	Param #
dense_172 (Dense)	(None, 256)	200960
dropout_19 (Dropout)	(None, 256)	0
dense_173 (Dense)	(None, 128)	32896
dropout_20 (Dropout)	(None, 128)	0
dense_174 (Dense)	(None, 64)	8256
dropout_21 (Dropout)	(None, 64)	0
dense_175 (Dense)	(None, 10)	650

Total params: 242,762

Trainable params: 242,762

Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))  
  
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 9s 158us/step - loss: 0.  
7965 - acc: 0.7478 - val_loss: 0.2032 - val_acc: 0.9425  
Epoch 2/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.3  
421 - acc: 0.9091 - val_loss: 0.1607 - val_acc: 0.9530  
Epoch 3/20  
60000/60000 [=====] - 4s 75us/step - loss: 0.2  
676 - acc: 0.9309 - val_loss: 0.1420 - val_acc: 0.9599  
Epoch 4/20  
60000/60000 [=====] - 5s 77us/step - loss: 0.2  
294 - acc: 0.9418 - val_loss: 0.1213 - val_acc: 0.9654  
Epoch 5/20  
60000/60000 [=====] - 5s 75us/step - loss: 0.2  
003 - acc: 0.9481 - val_loss: 0.1086 - val_acc: 0.9697  
Epoch 6/20  
60000/60000 [=====] - 5s 77us/step - loss: 0.1  
850 - acc: 0.9524 - val_loss: 0.0990 - val_acc: 0.9735  
Epoch 7/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
673 - acc: 0.9574 - val_loss: 0.1028 - val_acc: 0.9732  
Epoch 8/20  
60000/60000 [=====] - 5s 78us/step - loss: 0.1  
603 - acc: 0.9581 - val_loss: 0.0951 - val_acc: 0.9744  
Epoch 9/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
505 - acc: 0.9614 - val_loss: 0.0871 - val_acc: 0.9756  
Epoch 10/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
426 - acc: 0.9625 - val_loss: 0.0877 - val_acc: 0.9764  
Epoch 11/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
380 - acc: 0.9639 - val_loss: 0.0884 - val_acc: 0.9768  
Epoch 12/20  
60000/60000 [=====] - 5s 76us/step - loss: 0.1  
331 - acc: 0.9652 - val_loss: 0.0916 - val_acc: 0.9763
```

```
Epoch 13/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
264 - acc: 0.9667 - val_loss: 0.0889 - val_acc: 0.9775
Epoch 14/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
233 - acc: 0.9676 - val_loss: 0.0821 - val_acc: 0.9783
Epoch 15/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
175 - acc: 0.9692 - val_loss: 0.0856 - val_acc: 0.9789
Epoch 16/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
143 - acc: 0.9694 - val_loss: 0.0856 - val_acc: 0.9776
Epoch 17/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
149 - acc: 0.9698 - val_loss: 0.0840 - val_acc: 0.9790
Epoch 18/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
100 - acc: 0.9712 - val_loss: 0.0850 - val_acc: 0.9792
Epoch 19/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
095 - acc: 0.9708 - val_loss: 0.0837 - val_acc: 0.9792
Epoch 20/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
063 - acc: 0.9716 - val_loss: 0.0873 - val_acc: 0.9787
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

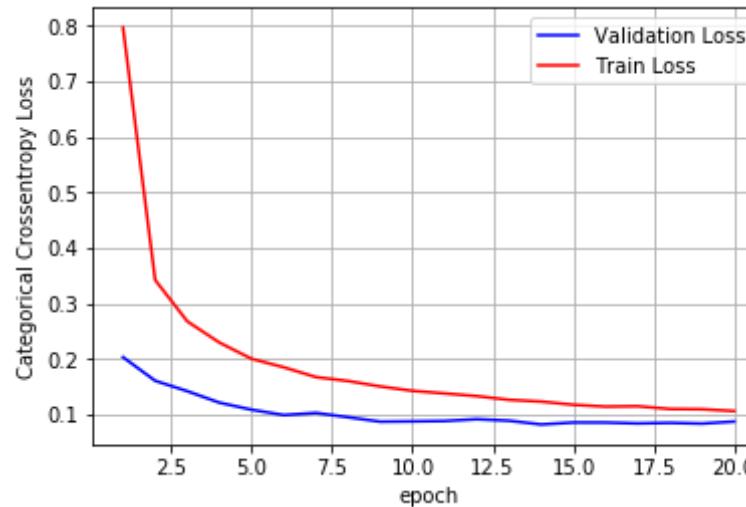
```
Test score: 0.08730042533672168
Test accuracy: 0.9787
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

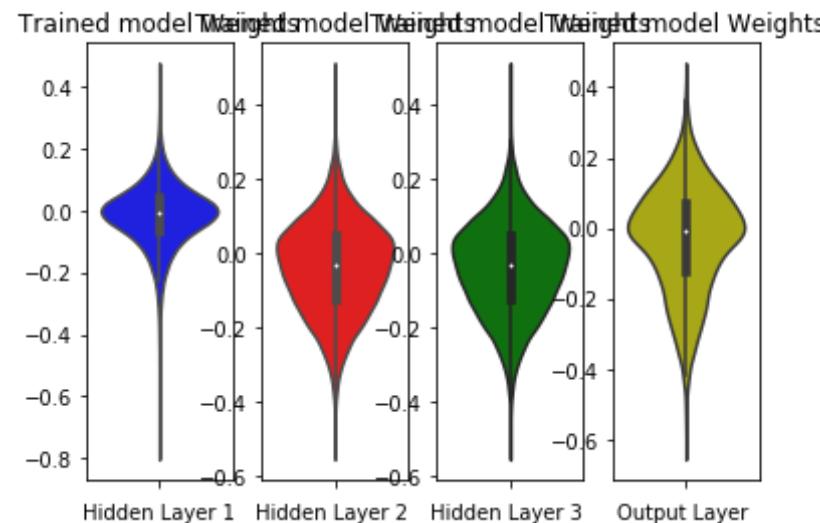
```

plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 4: relu activation with Dropout + GradientDescentOptimizer with SGD Weight

```

In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), ker

```

```
        kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
    mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(
    mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_57"

Layer (type)	Output Shape	Param #
dense_176 (Dense)	(None, 256)	200960
dropout_22 (Dropout)	(None, 256)	0
dense_177 (Dense)	(None, 128)	32896
dropout_23 (Dropout)	(None, 128)	0
dense_178 (Dense)	(None, 64)	8256
dropout_24 (Dropout)	(None, 64)	0
dense_179 (Dense)	(None, 10)	650

Total params: 242,762  
Trainable params: 242,762  
Non-trainable params: 0

In [0]: model\_1.compile(optimizer='sgd', loss='categorical\_crossentropy', metri

```
cs=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 147us/step - loss: 2.
1701 - acc: 0.2146 - val_loss: 1.6725 - val_acc: 0.6721
Epoch 2/20
60000/60000 [=====] - 4s 65us/step - loss: 1.6
120 - acc: 0.4465 - val_loss: 0.9077 - val_acc: 0.7901
Epoch 3/20
60000/60000 [=====] - 4s 65us/step - loss: 1.1
954 - acc: 0.5915 - val_loss: 0.6395 - val_acc: 0.8424
Epoch 4/20
60000/60000 [=====] - 4s 65us/step - loss: 0.9
695 - acc: 0.6790 - val_loss: 0.4995 - val_acc: 0.8754
Epoch 5/20
60000/60000 [=====] - 4s 65us/step - loss: 0.8
325 - acc: 0.7312 - val_loss: 0.4247 - val_acc: 0.8908
Epoch 6/20
60000/60000 [=====] - 4s 65us/step - loss: 0.7
422 - acc: 0.7670 - val_loss: 0.3742 - val_acc: 0.9013
Epoch 7/20
60000/60000 [=====] - 4s 65us/step - loss: 0.6
728 - acc: 0.7934 - val_loss: 0.3348 - val_acc: 0.9080
Epoch 8/20
60000/60000 [=====] - 4s 68us/step - loss: 0.6
187 - acc: 0.8152 - val_loss: 0.3075 - val_acc: 0.9139
Epoch 9/20
60000/60000 [=====] - 4s 69us/step - loss: 0.5
750 - acc: 0.8298 - val_loss: 0.2869 - val_acc: 0.9201
Epoch 10/20
60000/60000 [=====] - 4s 67us/step - loss: 0.5
378 - acc: 0.8443 - val_loss: 0.2695 - val_acc: 0.9247
Epoch 11/20
60000/60000 [=====] - 4s 65us/step - loss: 0.5
106 - acc: 0.8527 - val_loss: 0.2566 - val_acc: 0.9277
Epoch 12/20
60000/60000 [=====] - 4s 65us/step - loss: 0.5
200 - acc: 0.8530 - val_loss: 0.2566 - val_acc: 0.9277
```

```
60000/60000 [=====] - 4s 65us/step - loss: 0.4
883 - acc: 0.8613 - val_loss: 0.2423 - val_acc: 0.9312
Epoch 13/20
60000/60000 [=====] - 4s 66us/step - loss: 0.4
664 - acc: 0.8689 - val_loss: 0.2339 - val_acc: 0.9344
Epoch 14/20
60000/60000 [=====] - 4s 66us/step - loss: 0.4
468 - acc: 0.8744 - val_loss: 0.2230 - val_acc: 0.9370
Epoch 15/20
60000/60000 [=====] - 4s 67us/step - loss: 0.4
272 - acc: 0.8826 - val_loss: 0.2136 - val_acc: 0.9398
Epoch 16/20
60000/60000 [=====] - 4s 65us/step - loss: 0.4
110 - acc: 0.8864 - val_loss: 0.2083 - val_acc: 0.9430
Epoch 17/20
60000/60000 [=====] - 4s 67us/step - loss: 0.4
004 - acc: 0.8910 - val_loss: 0.2012 - val_acc: 0.9432
Epoch 18/20
60000/60000 [=====] - 4s 65us/step - loss: 0.3
847 - acc: 0.8939 - val_loss: 0.1956 - val_acc: 0.9448
Epoch 19/20
60000/60000 [=====] - 4s 64us/step - loss: 0.3
740 - acc: 0.8970 - val_loss: 0.1923 - val_acc: 0.9455
Epoch 20/20
60000/60000 [=====] - 4s 65us/step - loss: 0.3
623 - acc: 0.9019 - val_loss: 0.1837 - val_acc: 0.9489
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

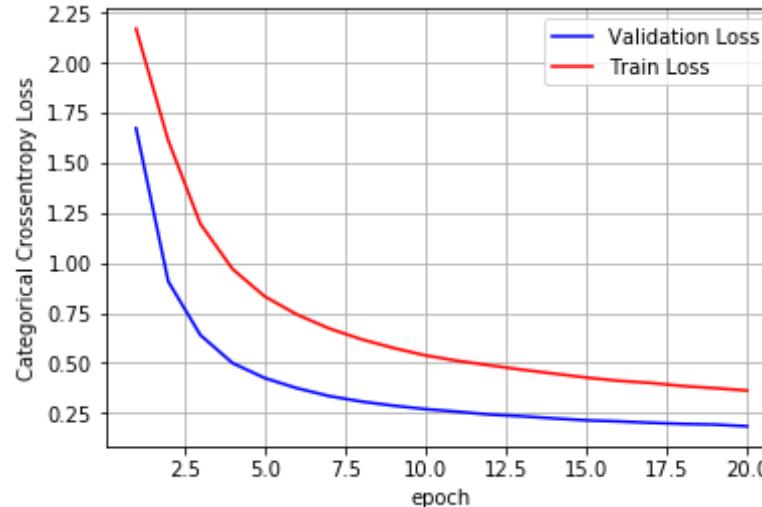
```
Test score: 0.18373567296117543
Test accuracy: 0.9489
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
```

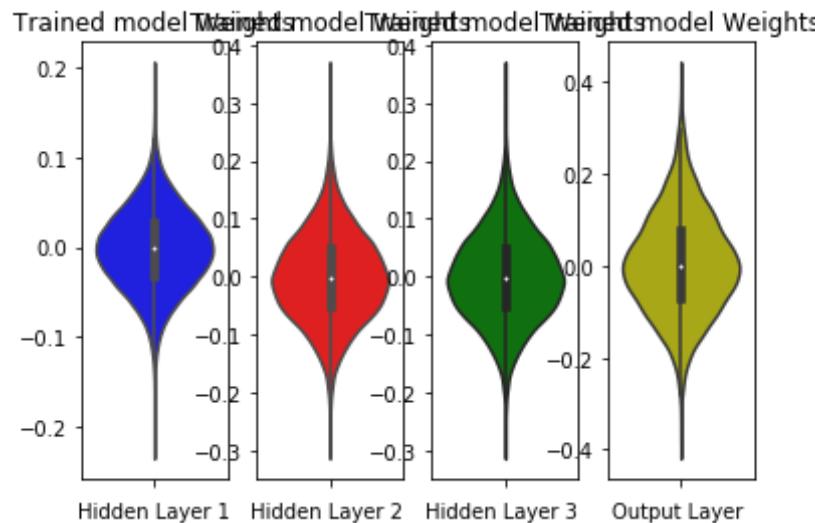
```

ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



Model 4: relu activation with Dropout + AdamOptimizer with Relu Weight</h3>

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

```

```
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_58"

Layer (type)	Output Shape	Param #
dense_180 (Dense)	(None, 256)	200960
dropout_25 (Dropout)	(None, 256)	0
dense_181 (Dense)	(None, 128)	32896
dropout_26 (Dropout)	(None, 128)	0
dense_182 (Dense)	(None, 64)	8256
dropout_27 (Dropout)	(None, 64)	0
dense_183 (Dense)	(None, 10)	650

Total params: 242,762

Trainable params: 242,762

Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 10s 162us/step - loss: 1.2960 - acc: 0.5797 - val_loss: 0.3377 - val_acc: 0.9141
Epoch 2/20
60000/60000 [=====] - 5s 76us/step - loss: 0.5141 - acc: 0.8522 - val_loss: 0.2182 - val_acc: 0.9406
Epoch 3/20
60000/60000 [=====] - 5s 79us/step - loss: 0.3821 - acc: 0.8988 - val_loss: 0.1788 - val_acc: 0.9507
Epoch 4/20
60000/60000 [=====] - 5s 76us/step - loss: 0.3148 - acc: 0.9180 - val_loss: 0.1619 - val_acc: 0.9560
Epoch 5/20
60000/60000 [=====] - 5s 77us/step - loss: 0.2691 - acc: 0.9321 - val_loss: 0.1402 - val_acc: 0.9614
Epoch 6/20
60000/60000 [=====] - 5s 77us/step - loss: 0.2427 - acc: 0.9374 - val_loss: 0.1321 - val_acc: 0.9652
Epoch 7/20
60000/60000 [=====] - 5s 77us/step - loss: 0.2238 - acc: 0.9431 - val_loss: 0.1249 - val_acc: 0.9672
Epoch 8/20
60000/60000 [=====] - 5s 77us/step - loss: 0.2070 - acc: 0.9474 - val_loss: 0.1155 - val_acc: 0.9688
Epoch 9/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1947 - acc: 0.9511 - val_loss: 0.1126 - val_acc: 0.9691
Epoch 10/20
60000/60000 [=====] - 5s 78us/step - loss: 0.1764 - acc: 0.9543 - val_loss: 0.1068 - val_acc: 0.9707
Epoch 11/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1728 - acc: 0.9568 - val_loss: 0.1051 - val_acc: 0.9724
Epoch 12/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1624 - acc: 0.9581 - val_loss: 0.1056 - val_acc: 0.9721
```

```
Epoch 13/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
615 - acc: 0.9584 - val_loss: 0.1016 - val_acc: 0.9736
Epoch 14/20
60000/60000 [=====] - 5s 78us/step - loss: 0.1
470 - acc: 0.9620 - val_loss: 0.0964 - val_acc: 0.9750
Epoch 15/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
460 - acc: 0.9634 - val_loss: 0.0976 - val_acc: 0.9757
Epoch 16/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
376 - acc: 0.9640 - val_loss: 0.0987 - val_acc: 0.9747
Epoch 17/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
336 - acc: 0.9663 - val_loss: 0.0988 - val_acc: 0.9742
Epoch 18/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1
308 - acc: 0.9656 - val_loss: 0.0986 - val_acc: 0.9747
Epoch 19/20
60000/60000 [=====] - 5s 78us/step - loss: 0.1
258 - acc: 0.9682 - val_loss: 0.0956 - val_acc: 0.9773
Epoch 20/20
60000/60000 [=====] - 5s 78us/step - loss: 0.1
208 - acc: 0.9682 - val_loss: 0.0971 - val_acc: 0.9758
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

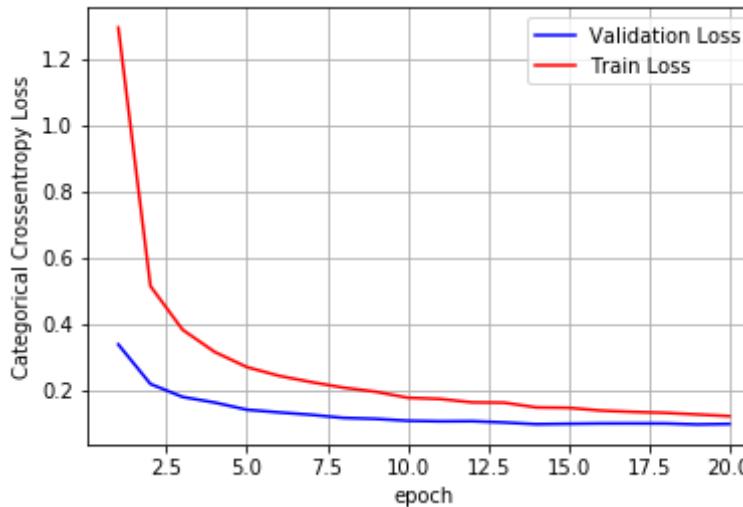
```
Test score: 0.09710459082686866
Test accuracy: 0.9758
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

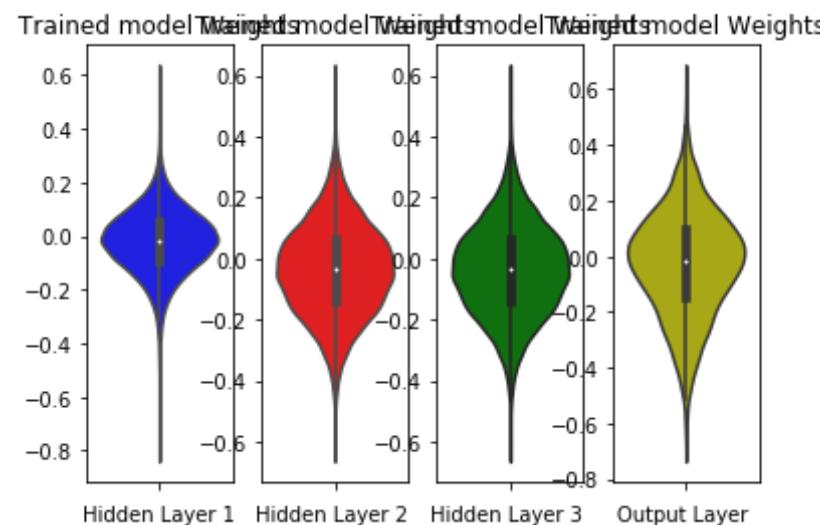
```

plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 4: relu activation with Dropout + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

```

```
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='relu'))

model_1.summary()
```

Model: "sequential\_59"

Layer (type)	Output Shape	Param #
<hr/>		
dense_184 (Dense)	(None, 256)	200960
dropout_28 (Dropout)	(None, 256)	0
dense_185 (Dense)	(None, 128)	32896
dropout_29 (Dropout)	(None, 128)	0
dense_186 (Dense)	(None, 64)	8256
dropout_30 (Dropout)	(None, 64)	0
dense_187 (Dense)	(None, 10)	650
<hr/>		

Total params: 242,762  
Trainable params: 242,762  
Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 9s 150us/step - loss: 3.
1887 - acc: 0.1117 - val_loss: 2.7026 - val_acc: 0.1135
Epoch 2/20
60000/60000 [=====] - 4s 66us/step - loss: 2.6
555 - acc: 0.1124 - val_loss: 2.6031 - val_acc: 0.1135
Epoch 3/20
60000/60000 [=====] - 4s 65us/step - loss: 2.5
769 - acc: 0.1124 - val_loss: 2.5491 - val_acc: 0.1135
Epoch 4/20
60000/60000 [=====] - 4s 65us/step - loss: 2.5
319 - acc: 0.1124 - val_loss: 2.5130 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 4s 66us/step - loss: 2.5
006 - acc: 0.1124 - val_loss: 2.4865 - val_acc: 0.1135
Epoch 6/20
60000/60000 [=====] - 4s 65us/step - loss: 2.4
778 - acc: 0.1123 - val_loss: 2.4658 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 4s 65us/step - loss: 2.4
587 - acc: 0.1124 - val_loss: 2.4491 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 4s 65us/step - loss: 2.4
422 - acc: 0.1124 - val_loss: 2.4351 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 4s 65us/step - loss: 2.4
300 - acc: 0.1124 - val_loss: 2.4233 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 4s 66us/step - loss: 2.4
193 - acc: 0.1123 - val_loss: 2.4130 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 4s 65us/step - loss: 2.4
087 - acc: 0.1124 - val_loss: 2.4041 - val_acc: 0.1135
Epoch 12/20
60000/60000 [=====] - 4s 66us/step - loss: 2.4
009 - acc: 0.1124 - val_loss: 2.3961 - val_acc: 0.1135
- 1/10/20
```

```
Epoch 13/20
60000/60000 [=====] - 4s 66us/step - loss: 2.3
931 - acc: 0.1123 - val_loss: 2.3890 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 4s 66us/step - loss: 2.3
861 - acc: 0.1124 - val_loss: 2.3826 - val_acc: 0.1135
Epoch 15/20
60000/60000 [=====] - 4s 66us/step - loss: 2.3
803 - acc: 0.1123 - val_loss: 2.3769 - val_acc: 0.1135
Epoch 16/20
60000/60000 [=====] - 4s 65us/step - loss: 2.3
747 - acc: 0.1124 - val_loss: 2.3716 - val_acc: 0.1135
Epoch 17/20
60000/60000 [=====] - 4s 66us/step - loss: 2.3
698 - acc: 0.1124 - val_loss: 2.3668 - val_acc: 0.1135
Epoch 18/20
60000/60000 [=====] - 4s 67us/step - loss: 2.3
647 - acc: 0.1124 - val_loss: 2.3624 - val_acc: 0.1135
Epoch 19/20
60000/60000 [=====] - 4s 65us/step - loss: 2.3
609 - acc: 0.1123 - val_loss: 2.3584 - val_acc: 0.1135
Epoch 20/20
60000/60000 [=====] - 4s 67us/step - loss: 2.3
568 - acc: 0.1123 - val_loss: 2.3547 - val_acc: 0.1135
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 2.3546517501831055
Test accuracy: 0.1135
```

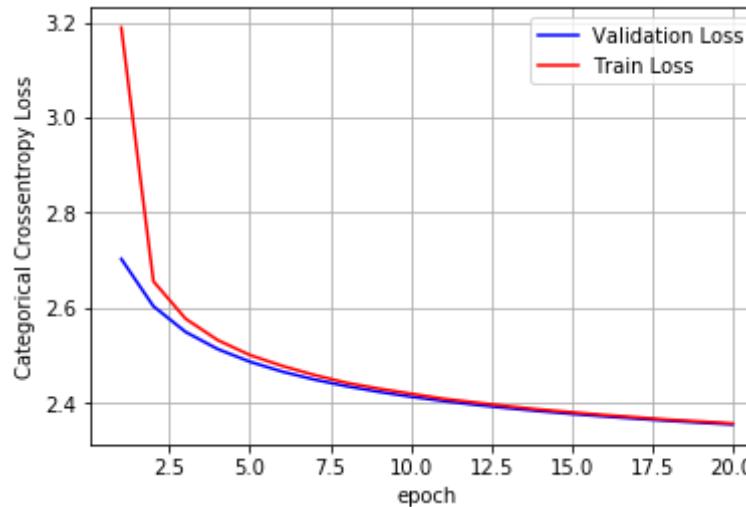
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
```

```
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

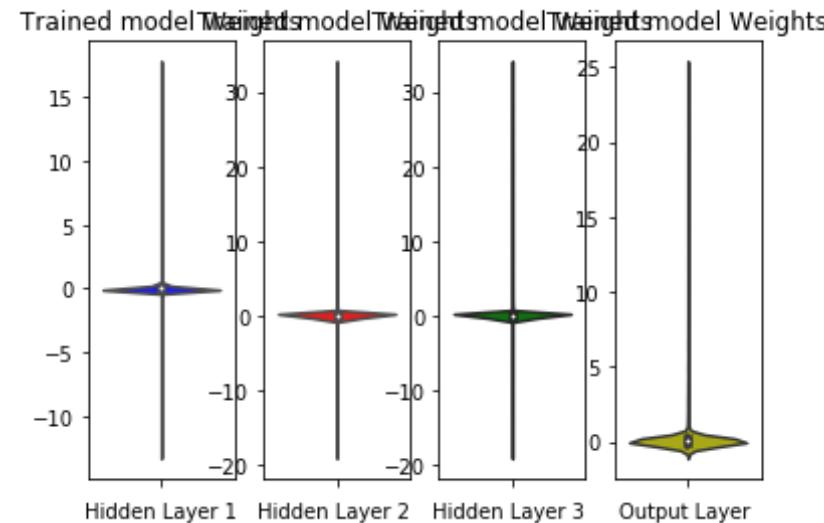
plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()

```



## Model 5: relu activation with BatchNormalization + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), ker

```

```

    kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
    mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(
    mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

```

Model: "sequential\_60"

Layer (type)	Output Shape	Param #
dense_188 (Dense)	(None, 256)	200960
batch_normalization_43 (Batch Normalization)	(None, 256)	1024
dense_189 (Dense)	(None, 128)	32896
batch_normalization_44 (Batch Normalization)	(None, 128)	512
dense_190 (Dense)	(None, 64)	8256
batch_normalization_45 (Batch Normalization)	(None, 64)	256
dense_191 (Dense)	(None, 10)	650

Total params: 244,554

Trainable params: 243,658

Non-trainable params: 896

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metr

```
    ics=[ 'accuracy' ])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 11s 181us/step - loss:
0.2296 - acc: 0.9319 - val_loss: 0.1219 - val_acc: 0.9621
Epoch 2/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
858 - acc: 0.9738 - val_loss: 0.0889 - val_acc: 0.9729
Epoch 3/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
555 - acc: 0.9826 - val_loss: 0.0923 - val_acc: 0.9718
Epoch 4/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0
433 - acc: 0.9864 - val_loss: 0.0810 - val_acc: 0.9762
Epoch 5/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0
326 - acc: 0.9896 - val_loss: 0.0906 - val_acc: 0.9729
Epoch 6/20
60000/60000 [=====] - 5s 85us/step - loss: 0.0
277 - acc: 0.9908 - val_loss: 0.0785 - val_acc: 0.9788
Epoch 7/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
265 - acc: 0.9912 - val_loss: 0.0749 - val_acc: 0.9781
Epoch 8/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
217 - acc: 0.9928 - val_loss: 0.0822 - val_acc: 0.9756
Epoch 9/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
187 - acc: 0.9938 - val_loss: 0.0886 - val_acc: 0.9762
Epoch 10/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0
185 - acc: 0.9938 - val_loss: 0.0756 - val_acc: 0.9801
Epoch 11/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
130 - acc: 0.9957 - val_loss: 0.0653 - val_acc: 0.9817
Epoch 12/20
60000/60000 [=====] - 5s 86us/step - loss: 0.0
130 - acc: 0.9957 - val_loss: 0.0653 - val_acc: 0.9817
```

```
60000/60000 [=====] - 5s 88us/step - loss: 0.0
167 - acc: 0.9948 - val_loss: 0.0781 - val_acc: 0.9794
Epoch 13/20
60000/60000 [=====] - 6s 93us/step - loss: 0.0
140 - acc: 0.9954 - val_loss: 0.0785 - val_acc: 0.9782
Epoch 14/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
107 - acc: 0.9962 - val_loss: 0.0816 - val_acc: 0.9794
Epoch 15/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0
133 - acc: 0.9956 - val_loss: 0.0749 - val_acc: 0.9802
Epoch 16/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
119 - acc: 0.9962 - val_loss: 0.0965 - val_acc: 0.9765
Epoch 17/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
118 - acc: 0.9960 - val_loss: 0.0766 - val_acc: 0.9805
Epoch 18/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
109 - acc: 0.9965 - val_loss: 0.0879 - val_acc: 0.9783
Epoch 19/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
108 - acc: 0.9962 - val_loss: 0.0961 - val_acc: 0.9776
Epoch 20/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0
088 - acc: 0.9971 - val_loss: 0.0840 - val_acc: 0.9808
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

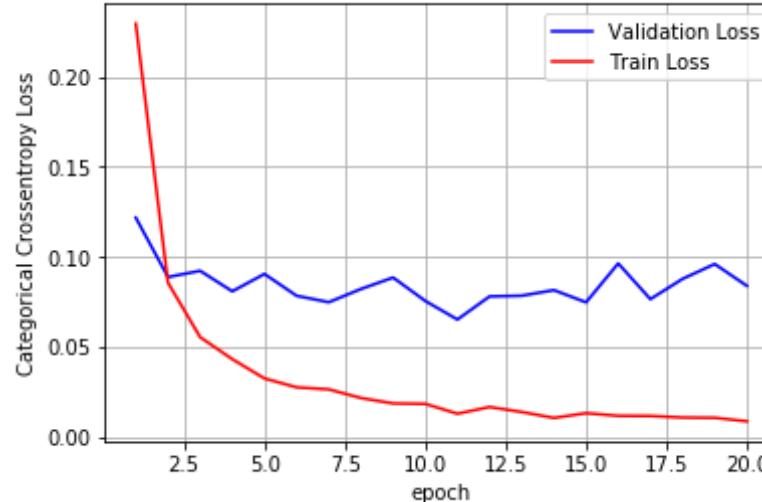
```
Test score: 0.08400668230069168
Test accuracy: 0.9808
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
```

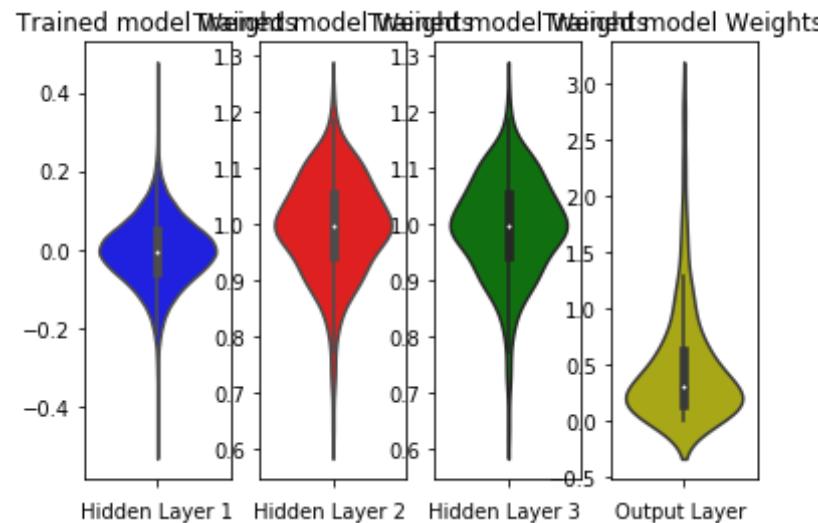
```

ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with SGD Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), ker

```

```
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_61"

Layer (type)	Output Shape	Param #
dense_192 (Dense)	(None, 256)	200960
batch_normalization_46 (Batch Normalization)	(None, 256)	1024
dense_193 (Dense)	(None, 128)	32896
batch_normalization_47 (Batch Normalization)	(None, 128)	512
dense_194 (Dense)	(None, 64)	8256
batch_normalization_48 (Batch Normalization)	(None, 64)	256
dense_195 (Dense)	(None, 10)	650

Total params: 244,554

Trainable params: 243,658

Non-trainable params: 896

In [0]: model\_1.compile(optimizer='sgd', loss='categorical\_crossentropy', metri

```
cs=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 10s 170us/step - loss:
0.5485 - acc: 0.8431 - val_loss: 0.2733 - val_acc: 0.9253
Epoch 2/20
60000/60000 [=====] - 5s 76us/step - loss: 0.2
490 - acc: 0.9311 - val_loss: 0.2015 - val_acc: 0.9417
Epoch 3/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
887 - acc: 0.9475 - val_loss: 0.1671 - val_acc: 0.9498
Epoch 4/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
544 - acc: 0.9561 - val_loss: 0.1473 - val_acc: 0.9568
Epoch 5/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
318 - acc: 0.9627 - val_loss: 0.1332 - val_acc: 0.9615
Epoch 6/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
149 - acc: 0.9682 - val_loss: 0.1251 - val_acc: 0.9641
Epoch 7/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
010 - acc: 0.9717 - val_loss: 0.1168 - val_acc: 0.9662
Epoch 8/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
899 - acc: 0.9757 - val_loss: 0.1121 - val_acc: 0.9679
Epoch 9/20
60000/60000 [=====] - 5s 75us/step - loss: 0.0
793 - acc: 0.9785 - val_loss: 0.1058 - val_acc: 0.9693
Epoch 10/20
60000/60000 [=====] - 4s 75us/step - loss: 0.0
725 - acc: 0.9805 - val_loss: 0.1038 - val_acc: 0.9700
Epoch 11/20
60000/60000 [=====] - 5s 75us/step - loss: 0.0
660 - acc: 0.9826 - val_loss: 0.0993 - val_acc: 0.9706
Epoch 12/20
60000/60000 [=====] - 4s 75us/step - loss: 0.0
600 - acc: 0.9845 - val_loss: 0.0958 - val_acc: 0.9715
```

```
60000/60000 [=====] - 4s 74us/step - loss: 0.0
593 - acc: 0.9845 - val_loss: 0.0971 - val_acc: 0.9721
Epoch 13/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
552 - acc: 0.9860 - val_loss: 0.0962 - val_acc: 0.9726
Epoch 14/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
497 - acc: 0.9878 - val_loss: 0.0930 - val_acc: 0.9732
Epoch 15/20
60000/60000 [=====] - 4s 75us/step - loss: 0.0
457 - acc: 0.9889 - val_loss: 0.0932 - val_acc: 0.9734
Epoch 16/20
60000/60000 [=====] - 5s 75us/step - loss: 0.0
422 - acc: 0.9901 - val_loss: 0.0902 - val_acc: 0.9736
Epoch 17/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
384 - acc: 0.9916 - val_loss: 0.0898 - val_acc: 0.9740
Epoch 18/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
356 - acc: 0.9919 - val_loss: 0.0891 - val_acc: 0.9735
Epoch 19/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
332 - acc: 0.9927 - val_loss: 0.0890 - val_acc: 0.9740
Epoch 20/20
60000/60000 [=====] - 4s 75us/step - loss: 0.0
310 - acc: 0.9932 - val_loss: 0.0864 - val_acc: 0.9749
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

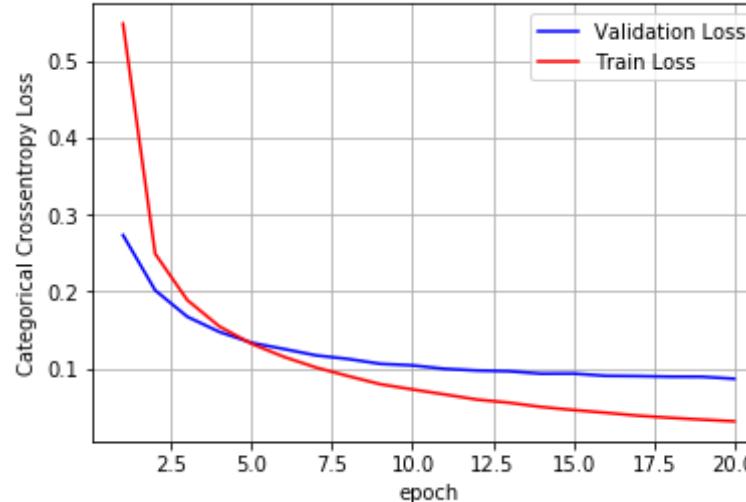
```
Test score: 0.0863677886866033
Test accuracy: 0.9749
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
```

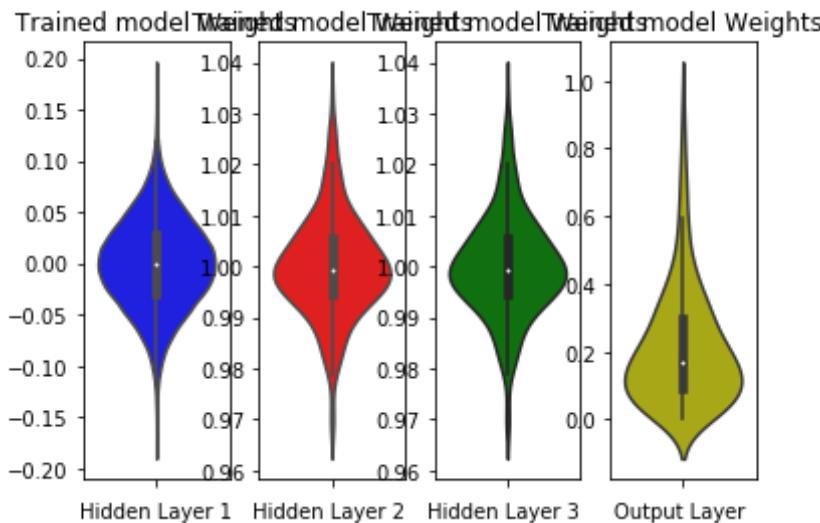
```

ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



Model 5: relu activation with BatchNormalization + AdamOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

```

```
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_62"

Layer (type)	Output Shape	Param #
dense_196 (Dense)	(None, 256)	200960
batch_normalization_49 (Batch Normalization)	(None, 256)	1024
dense_197 (Dense)	(None, 128)	32896
batch_normalization_50 (Batch Normalization)	(None, 128)	512
dense_198 (Dense)	(None, 64)	8256
batch_normalization_51 (Batch Normalization)	(None, 64)	256
dense_199 (Dense)	(None, 10)	650

Total params: 244,554  
Trainable params: 243,658  
Non-trainable params: 896

In [0]: `model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])`

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 12s 200us/step - loss: 0.2985 - acc: 0.9135 - val_loss: 0.1218 - val_acc: 0.9621
Epoch 2/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0 964 - acc: 0.9709 - val_loss: 0.0972 - val_acc: 0.9712
Epoch 3/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0 600 - acc: 0.9818 - val_loss: 0.0908 - val_acc: 0.9708
Epoch 4/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0 419 - acc: 0.9870 - val_loss: 0.0753 - val_acc: 0.9765
Epoch 5/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0 308 - acc: 0.9904 - val_loss: 0.0828 - val_acc: 0.9766
Epoch 6/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0 262 - acc: 0.9912 - val_loss: 0.0846 - val_acc: 0.9742
Epoch 7/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0 225 - acc: 0.9924 - val_loss: 0.0757 - val_acc: 0.9778
Epoch 8/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0 185 - acc: 0.9936 - val_loss: 0.0931 - val_acc: 0.9738
Epoch 9/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0 170 - acc: 0.9944 - val_loss: 0.0832 - val_acc: 0.9780
Epoch 10/20
60000/60000 [=====] - 5s 87us/step - loss: 0.0 155 - acc: 0.9950 - val_loss: 0.0781 - val_acc: 0.9781
Epoch 11/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0 139 - acc: 0.9952 - val_loss: 0.0788 - val_acc: 0.9798
Epoch 12/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0 122 - acc: 0.9957 - val_loss: 0.0924 - val_acc: 0.9777
```

```
Epoch 13/20
60000/60000 [=====] - 5s 90us/step - loss: 0.0
131 - acc: 0.9957 - val_loss: 0.0833 - val_acc: 0.9790
Epoch 14/20
60000/60000 [=====] - 6s 92us/step - loss: 0.0
124 - acc: 0.9955 - val_loss: 0.0897 - val_acc: 0.9780
Epoch 15/20
60000/60000 [=====] - 5s 91us/step - loss: 0.0
107 - acc: 0.9965 - val_loss: 0.0766 - val_acc: 0.9800
Epoch 16/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
099 - acc: 0.9967 - val_loss: 0.0840 - val_acc: 0.9787
Epoch 17/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0
091 - acc: 0.9970 - val_loss: 0.0855 - val_acc: 0.9781
Epoch 18/20
60000/60000 [=====] - 5s 88us/step - loss: 0.0
087 - acc: 0.9971 - val_loss: 0.0951 - val_acc: 0.9774
Epoch 19/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0
105 - acc: 0.9964 - val_loss: 0.0844 - val_acc: 0.9796
Epoch 20/20
60000/60000 [=====] - 5s 89us/step - loss: 0.0
073 - acc: 0.9979 - val_loss: 0.0922 - val_acc: 0.9781
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

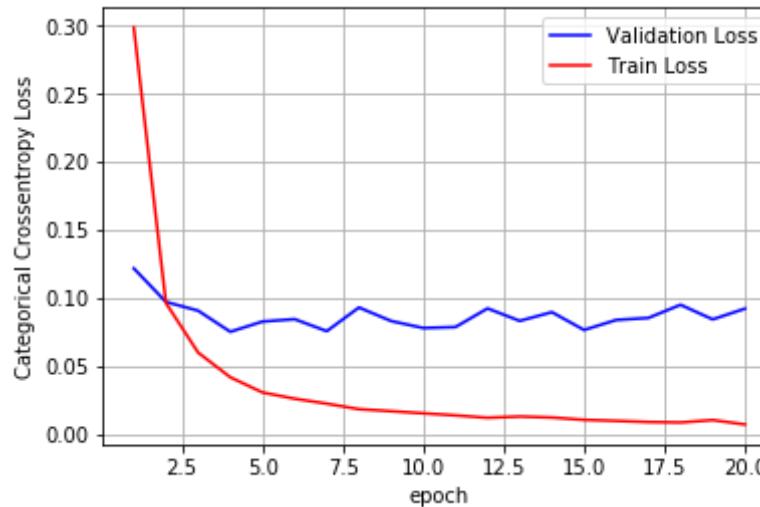
```
Test score: 0.09224662557971605
Test accuracy: 0.9781
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']
```

```
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

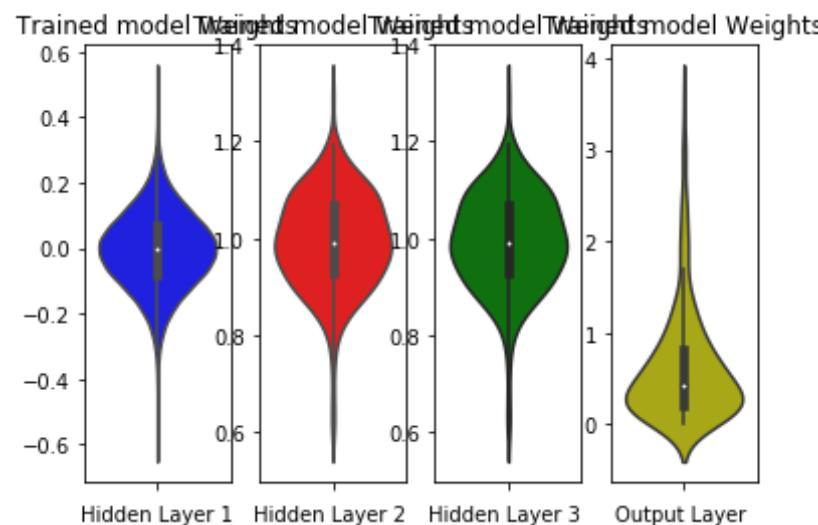
```

plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

```

```
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())
model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_63"

Layer (type)	Output Shape	Param #
dense_200 (Dense)	(None, 256)	200960
batch_normalization_52 (BatchNormalization)	(None, 256)	1024
dense_201 (Dense)	(None, 128)	32896
batch_normalization_53 (BatchNormalization)	(None, 128)	512
dense_202 (Dense)	(None, 64)	8256
batch_normalization_54 (BatchNormalization)	(None, 64)	256
dense_203 (Dense)	(None, 10)	650
<hr/>		
Total params: 244,554		
Trainable params: 243,658		
Non-trainable params: 896		

In [0]: `model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])`

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 11s 178us/step - loss: 0.8961 - acc: 0.7268 - val_loss: 0.4470 - val_acc: 0.8758
Epoch 2/20
60000/60000 [=====] - 5s 75us/step - loss: 0.4068 - acc: 0.8848 - val_loss: 0.3192 - val_acc: 0.9094
Epoch 3/20
60000/60000 [=====] - 5s 77us/step - loss: 0.3127 - acc: 0.9096 - val_loss: 0.2647 - val_acc: 0.9254
Epoch 4/20
60000/60000 [=====] - 5s 75us/step - loss: 0.2636 - acc: 0.9234 - val_loss: 0.2330 - val_acc: 0.9321
Epoch 5/20
60000/60000 [=====] - 5s 76us/step - loss: 0.2308 - acc: 0.9319 - val_loss: 0.2118 - val_acc: 0.9381
Epoch 6/20
60000/60000 [=====] - 5s 76us/step - loss: 0.2071 - acc: 0.9395 - val_loss: 0.1961 - val_acc: 0.9427
Epoch 7/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1900 - acc: 0.9439 - val_loss: 0.1838 - val_acc: 0.9463
Epoch 8/20
60000/60000 [=====] - 5s 77us/step - loss: 0.1734 - acc: 0.9492 - val_loss: 0.1745 - val_acc: 0.9489
Epoch 9/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1620 - acc: 0.9525 - val_loss: 0.1667 - val_acc: 0.9513
Epoch 10/20
60000/60000 [=====] - 5s 81us/step - loss: 0.1511 - acc: 0.9557 - val_loss: 0.1597 - val_acc: 0.9513
Epoch 11/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1427 - acc: 0.9581 - val_loss: 0.1546 - val_acc: 0.9531
Epoch 12/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1341 - acc: 0.9609 - val_loss: 0.1494 - val_acc: 0.9544
```

```
Epoch 13/20
60000/60000 [=====] - 5s 78us/step - loss: 0.1
275 - acc: 0.9621 - val_loss: 0.1453 - val_acc: 0.9551
Epoch 14/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
200 - acc: 0.9651 - val_loss: 0.1421 - val_acc: 0.9557
Epoch 15/20
60000/60000 [=====] - 5s 76us/step - loss: 0.1
146 - acc: 0.9666 - val_loss: 0.1382 - val_acc: 0.9565
Epoch 16/20
60000/60000 [=====] - 4s 75us/step - loss: 0.1
080 - acc: 0.9684 - val_loss: 0.1358 - val_acc: 0.9571
Epoch 17/20
60000/60000 [=====] - 4s 74us/step - loss: 0.1
042 - acc: 0.9695 - val_loss: 0.1335 - val_acc: 0.9590
Epoch 18/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
998 - acc: 0.9707 - val_loss: 0.1307 - val_acc: 0.9589
Epoch 19/20
60000/60000 [=====] - 4s 75us/step - loss: 0.0
959 - acc: 0.9719 - val_loss: 0.1286 - val_acc: 0.9584
Epoch 20/20
60000/60000 [=====] - 4s 74us/step - loss: 0.0
907 - acc: 0.9741 - val_loss: 0.1268 - val_acc: 0.9605
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

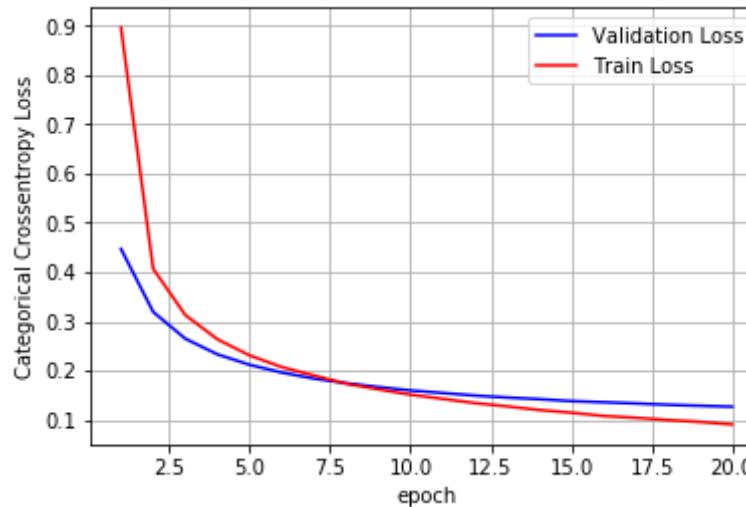
```
Test score: 0.12683080091178417
Test accuracy: 0.9605
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

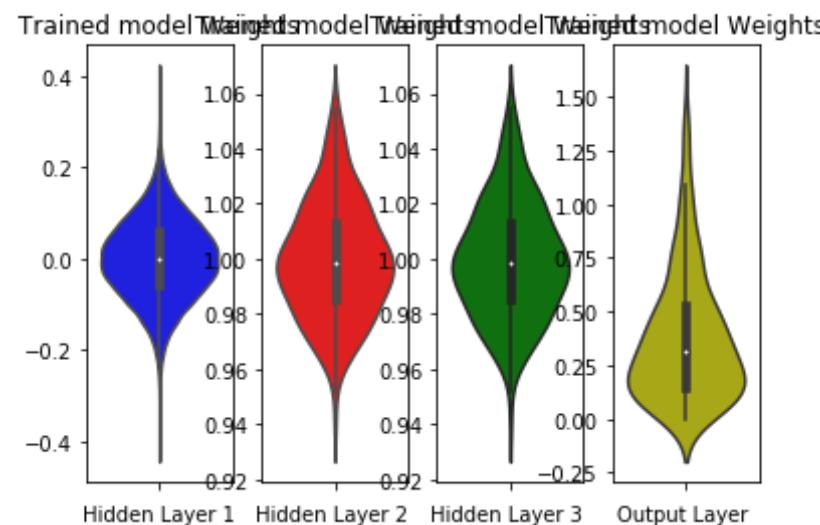
```

plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 6: sigmoid activation with Dropout + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),

```

```

        kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

```

Model: "sequential\_64"

Layer (type)	Output Shape	Param #
<hr/>		
dense_204 (Dense)	(None, 256)	200960
dropout_31 (Dropout)	(None, 256)	0
dense_205 (Dense)	(None, 128)	32896
dropout_32 (Dropout)	(None, 128)	0
dense_206 (Dense)	(None, 64)	8256
dropout_33 (Dropout)	(None, 64)	0
dense_207 (Dense)	(None, 10)	650
<hr/>		

Total params: 242,762  
Trainable params: 242,762  
Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metr

```
    ics=[ 'accuracy' ])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 11s 178us/step - loss:
1.3987 - acc: 0.5296 - val_loss: 0.4613 - val_acc: 0.8859
Epoch 2/20
60000/60000 [=====] - 5s 80us/step - loss: 0.5
704 - acc: 0.8409 - val_loss: 0.2848 - val_acc: 0.9204
Epoch 3/20
60000/60000 [=====] - 5s 80us/step - loss: 0.4
230 - acc: 0.8883 - val_loss: 0.2292 - val_acc: 0.9334
Epoch 4/20
60000/60000 [=====] - 5s 79us/step - loss: 0.3
520 - acc: 0.9083 - val_loss: 0.1947 - val_acc: 0.9436
Epoch 5/20
60000/60000 [=====] - 5s 79us/step - loss: 0.3
086 - acc: 0.9192 - val_loss: 0.1711 - val_acc: 0.9499
Epoch 6/20
60000/60000 [=====] - 5s 79us/step - loss: 0.2
774 - acc: 0.9289 - val_loss: 0.1532 - val_acc: 0.9566
Epoch 7/20
60000/60000 [=====] - 5s 78us/step - loss: 0.2
540 - acc: 0.9345 - val_loss: 0.1425 - val_acc: 0.9587
Epoch 8/20
60000/60000 [=====] - 5s 79us/step - loss: 0.2
433 - acc: 0.9367 - val_loss: 0.1349 - val_acc: 0.9620
Epoch 9/20
60000/60000 [=====] - 5s 80us/step - loss: 0.2
208 - acc: 0.9432 - val_loss: 0.1280 - val_acc: 0.9633
Epoch 10/20
60000/60000 [=====] - 5s 78us/step - loss: 0.2
101 - acc: 0.9457 - val_loss: 0.1211 - val_acc: 0.9654
Epoch 11/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
937 - acc: 0.9491 - val_loss: 0.1175 - val_acc: 0.9680
Epoch 12/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1
850 - acc: 0.9511 - val_loss: 0.1155 - val_acc: 0.9700
```

```
60000/60000 [=====] - 5s 78us/step - loss: 0.1
909 - acc: 0.9510 - val_loss: 0.1118 - val_acc: 0.9690
Epoch 13/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1
815 - acc: 0.9523 - val_loss: 0.1059 - val_acc: 0.9704
Epoch 14/20
60000/60000 [=====] - 5s 79us/step - loss: 0.1
766 - acc: 0.9541 - val_loss: 0.1036 - val_acc: 0.9708
Epoch 15/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
672 - acc: 0.9565 - val_loss: 0.1002 - val_acc: 0.9715
Epoch 16/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
591 - acc: 0.9575 - val_loss: 0.0988 - val_acc: 0.9736
Epoch 17/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
523 - acc: 0.9593 - val_loss: 0.0968 - val_acc: 0.9735
Epoch 18/20
60000/60000 [=====] - 5s 81us/step - loss: 0.1
478 - acc: 0.9613 - val_loss: 0.0925 - val_acc: 0.9744
Epoch 19/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
437 - acc: 0.9614 - val_loss: 0.0950 - val_acc: 0.9741
Epoch 20/20
60000/60000 [=====] - 5s 80us/step - loss: 0.1
418 - acc: 0.9621 - val_loss: 0.0926 - val_acc: 0.9750
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

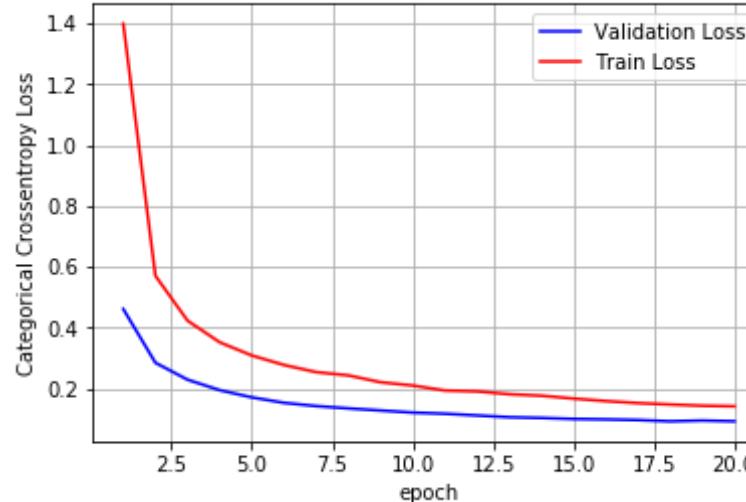
```
Test score: 0.09259034679569304
Test accuracy: 0.975
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
```

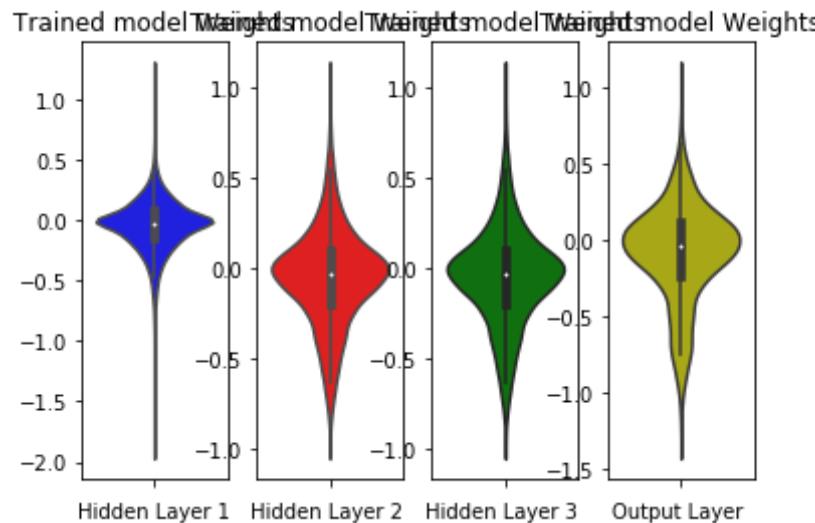
```

ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with SGD Weight

In [0]:

```

model_1 = Sequential()
model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(Dropout(0.5))

```

```
model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_65"

Layer (type)	Output Shape	Param #
<hr/>		
dense_208 (Dense)	(None, 256)	200960
dropout_34 (Dropout)	(None, 256)	0
dense_209 (Dense)	(None, 128)	32896
dropout_35 (Dropout)	(None, 128)	0
dense_210 (Dense)	(None, 64)	8256
dropout_36 (Dropout)	(None, 64)	0
dense_211 (Dense)	(None, 10)	650
<hr/>		

Total params: 242,762  
Trainable params: 242,762  
Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='sgd', loss='categorical\_crossentropy', metrics=['accuracy'])

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 10s 167us/step - loss: 2.4856 - acc: 0.1043 - val_loss: 2.2976 - val_acc: 0.1135
Epoch 2/20
60000/60000 [=====] - 4s 68us/step - loss: 2.4040 - acc: 0.1020 - val_loss: 2.2957 - val_acc: 0.1135
Epoch 3/20
60000/60000 [=====] - 4s 68us/step - loss: 2.3679 - acc: 0.1029 - val_loss: 2.2946 - val_acc: 0.1135
Epoch 4/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3450 - acc: 0.1051 - val_loss: 2.2938 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 4s 70us/step - loss: 2.3336 - acc: 0.1065 - val_loss: 2.2932 - val_acc: 0.1135
Epoch 6/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3249 - acc: 0.1063 - val_loss: 2.2930 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 4s 68us/step - loss: 2.3197 - acc: 0.1075 - val_loss: 2.2925 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 4s 70us/step - loss: 2.3165 - acc: 0.1068 - val_loss: 2.2922 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3112 - acc: 0.1102 - val_loss: 2.2919 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 4s 71us/step - loss: 2.3095 - acc: 0.1118 - val_loss: 2.2915 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 4s 70us/step - loss: 2.3070 - acc: 0.1143 - val_loss: 2.2912 - val_acc: 0.1135
Epoch 12/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3050 - acc: 0.1138 - val_loss: 2.2908 - val_acc: 0.1135
```

```
Epoch 13/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3
042 - acc: 0.1139 - val_loss: 2.2903 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3
020 - acc: 0.1161 - val_loss: 2.2897 - val_acc: 0.1193
Epoch 15/20
60000/60000 [=====] - 4s 69us/step - loss: 2.3
011 - acc: 0.1176 - val_loss: 2.2892 - val_acc: 0.1158
Epoch 16/20
60000/60000 [=====] - 4s 70us/step - loss: 2.2
995 - acc: 0.1179 - val_loss: 2.2884 - val_acc: 0.1135
Epoch 17/20
60000/60000 [=====] - 4s 68us/step - loss: 2.2
983 - acc: 0.1224 - val_loss: 2.2875 - val_acc: 0.1300
Epoch 18/20
60000/60000 [=====] - 4s 70us/step - loss: 2.2
979 - acc: 0.1220 - val_loss: 2.2867 - val_acc: 0.1353
Epoch 19/20
60000/60000 [=====] - 4s 68us/step - loss: 2.2
961 - acc: 0.1244 - val_loss: 2.2855 - val_acc: 0.1435
Epoch 20/20
60000/60000 [=====] - 4s 70us/step - loss: 2.2
952 - acc: 0.1264 - val_loss: 2.2844 - val_acc: 0.1497
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 2.2844085891723633
Test accuracy: 0.1497
```

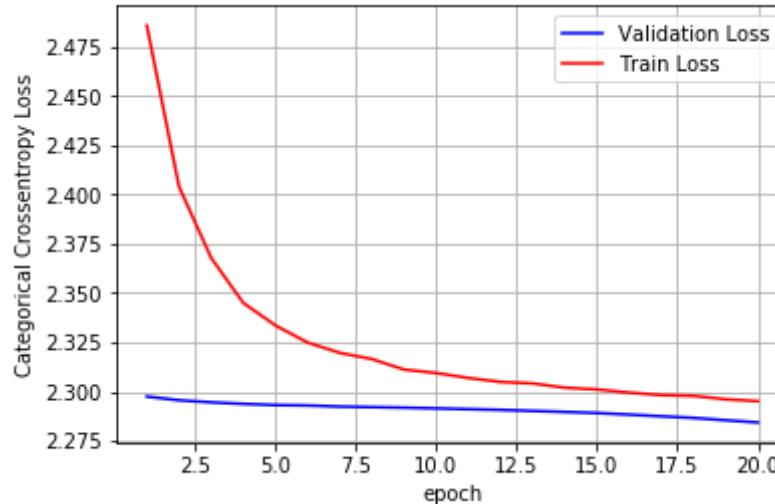
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
```

```
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

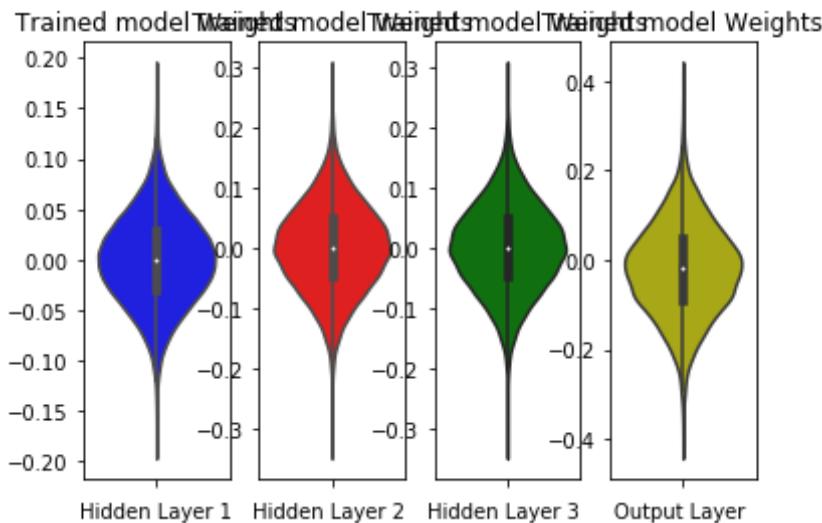
plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()

```



Model 6: sigmoid activation with Dropout + AdamOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))

```

```
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_66"

Layer (type)	Output Shape	Param #
dense_212 (Dense)	(None, 256)	200960
dropout_37 (Dropout)	(None, 256)	0
dense_213 (Dense)	(None, 128)	32896
dropout_38 (Dropout)	(None, 128)	0
dense_214 (Dense)	(None, 64)	8256
dropout_39 (Dropout)	(None, 64)	0
dense_215 (Dense)	(None, 10)	650

Total params: 242,762  
Trainable params: 242,762  
Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

history = model\_1.fit(X\_train, Y\_train, batch\_size=batch\_size, epochs=n\_b\_epoch, verbose=1, validation\_data=(X\_test, Y\_test))

Train on 60000 samples, validate on 10000 samples

```
Epoch 1/20
60000/60000 [=====] - 11s 183us/step - loss: 1.3694 - acc: 0.5323 - val_loss: 0.5083 - val_acc: 0.8744
Epoch 2/20
60000/60000 [=====] - 5s 80us/step - loss: 0.6173 - acc: 0.8213 - val_loss: 0.3068 - val_acc: 0.9156
Epoch 3/20
60000/60000 [=====] - 5s 81us/step - loss: 0.4543 - acc: 0.8774 - val_loss: 0.2334 - val_acc: 0.9352
Epoch 4/20
60000/60000 [=====] - 5s 79us/step - loss: 0.3723 - acc: 0.9027 - val_loss: 0.2074 - val_acc: 0.9399
Epoch 5/20
60000/60000 [=====] - 5s 79us/step - loss: 0.3267 - acc: 0.9141 - val_loss: 0.1795 - val_acc: 0.9487
Epoch 6/20
60000/60000 [=====] - 5s 80us/step - loss: 0.2898 - acc: 0.9245 - val_loss: 0.1637 - val_acc: 0.9531
Epoch 7/20
60000/60000 [=====] - 5s 80us/step - loss: 0.2653 - acc: 0.9307 - val_loss: 0.1516 - val_acc: 0.9576
Epoch 8/20
60000/60000 [=====] - 5s 81us/step - loss: 0.2434 - acc: 0.9373 - val_loss: 0.1386 - val_acc: 0.9606
Epoch 9/20
60000/60000 [=====] - 5s 80us/step - loss: 0.2289 - acc: 0.9406 - val_loss: 0.1326 - val_acc: 0.9621
Epoch 10/20
60000/60000 [=====] - 5s 81us/step - loss: 0.2150 - acc: 0.9443 - val_loss: 0.1263 - val_acc: 0.9637
Epoch 11/20
60000/60000 [=====] - 5s 82us/step - loss: 0.2020 - acc: 0.9480 - val_loss: 0.1195 - val_acc: 0.9655
Epoch 12/20
60000/60000 [=====] - 5s 85us/step - loss: 0.1943 - acc: 0.9492 - val_loss: 0.1129 - val_acc: 0.9682
Epoch 13/20
60000/60000 [=====] - 5s 87us/step - loss: 0.1806 - acc: 0.9520 - val_loss: 0.1148 - val_acc: 0.9670
Epoch 14/20
```

```
60000/60000 [=====] - 5s 82us/step - loss: 0.1
738 - acc: 0.9544 - val_loss: 0.1092 - val_acc: 0.9694
Epoch 15/20
60000/60000 [=====] - 5s 83us/step - loss: 0.1
685 - acc: 0.9562 - val_loss: 0.1049 - val_acc: 0.9709
Epoch 16/20
60000/60000 [=====] - 5s 81us/step - loss: 0.1
580 - acc: 0.9586 - val_loss: 0.1076 - val_acc: 0.9719
Epoch 17/20
60000/60000 [=====] - 5s 82us/step - loss: 0.1
541 - acc: 0.9600 - val_loss: 0.1019 - val_acc: 0.9719
Epoch 18/20
60000/60000 [=====] - 5s 83us/step - loss: 0.1
494 - acc: 0.9594 - val_loss: 0.0988 - val_acc: 0.9732
Epoch 19/20
60000/60000 [=====] - 5s 82us/step - loss: 0.1
441 - acc: 0.9616 - val_loss: 0.0970 - val_acc: 0.9729
Epoch 20/20
60000/60000 [=====] - 5s 83us/step - loss: 0.1
382 - acc: 0.9640 - val_loss: 0.1007 - val_acc: 0.9733
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

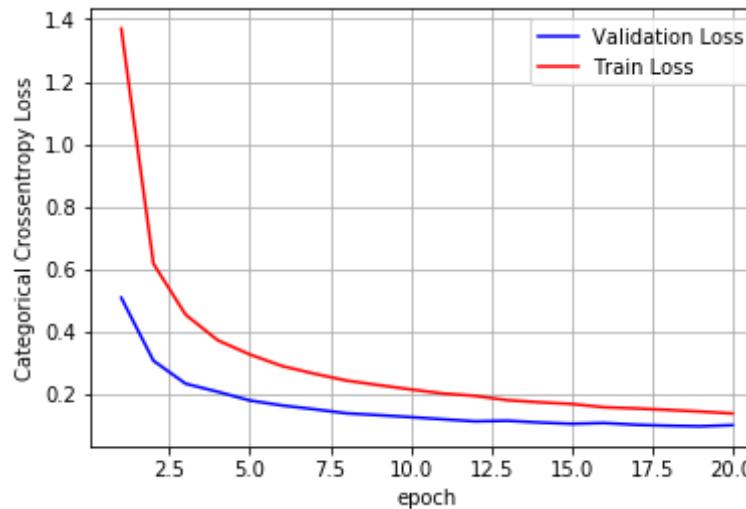
```
Test score: 0.10067519135973416
Test accuracy: 0.9733
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[3].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

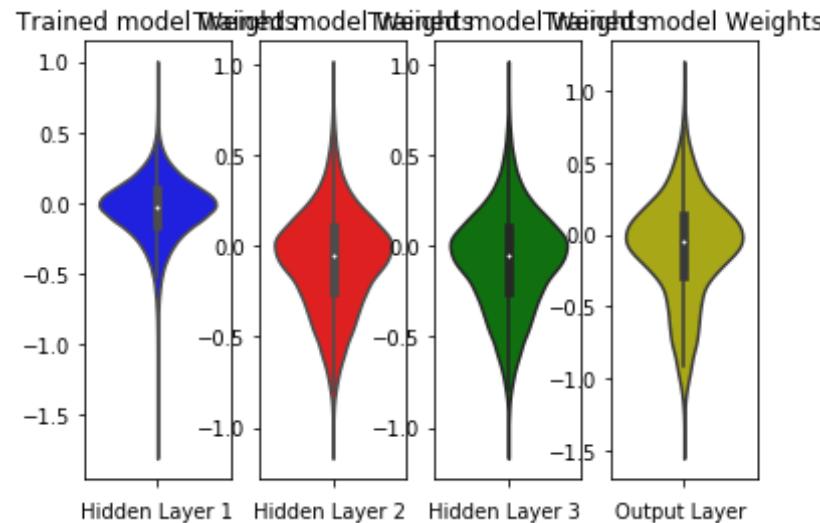
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with Relu Weight

```

In [0]: model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

```

```
model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_67"

Layer (type)	Output Shape	Param #
<hr/>		
dense_216 (Dense)	(None, 256)	200960
dropout_40 (Dropout)	(None, 256)	0
dense_217 (Dense)	(None, 128)	32896
dropout_41 (Dropout)	(None, 128)	0
dense_218 (Dense)	(None, 64)	8256
dropout_42 (Dropout)	(None, 64)	0
dense_219 (Dense)	(None, 10)	650
<hr/>		
Total params: 242,762		
Trainable params: 242,762		
Non-trainable params: 0		

In [0]:

```
model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20

```
60000/60000 [=====] - 10s 172us/step - loss:  
2.5109 - acc: 0.1016 - val_loss: 2.2875 - val_acc: 0.1222  
Epoch 2/20  
60000/60000 [=====] - 5s 77us/step - loss: 2.4  
089 - acc: 0.1045 - val_loss: 2.2787 - val_acc: 0.1135  
Epoch 3/20  
60000/60000 [=====] - 5s 75us/step - loss: 2.3  
652 - acc: 0.1096 - val_loss: 2.2738 - val_acc: 0.1796  
Epoch 4/20  
60000/60000 [=====] - 4s 75us/step - loss: 2.3  
371 - acc: 0.1128 - val_loss: 2.2703 - val_acc: 0.1225  
Epoch 5/20  
60000/60000 [=====] - 5s 80us/step - loss: 2.3  
203 - acc: 0.1162 - val_loss: 2.2664 - val_acc: 0.1360  
Epoch 6/20  
60000/60000 [=====] - 4s 70us/step - loss: 2.3  
113 - acc: 0.1190 - val_loss: 2.2633 - val_acc: 0.1322  
Epoch 7/20  
60000/60000 [=====] - 4s 70us/step - loss: 2.3  
031 - acc: 0.1226 - val_loss: 2.2594 - val_acc: 0.1528  
Epoch 8/20  
60000/60000 [=====] - 5s 81us/step - loss: 2.2  
948 - acc: 0.1290 - val_loss: 2.2545 - val_acc: 0.1804  
Epoch 9/20  
60000/60000 [=====] - 5s 83us/step - loss: 2.2  
867 - acc: 0.1367 - val_loss: 2.2483 - val_acc: 0.2451  
Epoch 10/20  
60000/60000 [=====] - 5s 83us/step - loss: 2.2  
828 - acc: 0.1369 - val_loss: 2.2410 - val_acc: 0.2121  
Epoch 11/20  
60000/60000 [=====] - 5s 86us/step - loss: 2.2  
743 - acc: 0.1459 - val_loss: 2.2318 - val_acc: 0.2507  
Epoch 12/20  
60000/60000 [=====] - 5s 83us/step - loss: 2.2  
670 - acc: 0.1526 - val_loss: 2.2203 - val_acc: 0.2868  
Epoch 13/20  
60000/60000 [=====] - 5s 84us/step - loss: 2.2  
594 - acc: 0.1580 - val_loss: 2.2052 - val_acc: 0.3557  
Epoch 14/20  
60000/60000 [=====] - 5s 84us/step - loss: 2.2
```

```
491 - acc: 0.1641 - val_loss: 2.1867 - val_acc: 0.4104
Epoch 15/20
60000/60000 [=====] - 5s 85us/step - loss: 2.2
374 - acc: 0.1734 - val_loss: 2.1629 - val_acc: 0.3751
Epoch 16/20
60000/60000 [=====] - 5s 85us/step - loss: 2.2
198 - acc: 0.1844 - val_loss: 2.1326 - val_acc: 0.3960
Epoch 17/20
60000/60000 [=====] - 5s 85us/step - loss: 2.1
995 - acc: 0.1931 - val_loss: 2.0953 - val_acc: 0.4303
Epoch 18/20
60000/60000 [=====] - 5s 85us/step - loss: 2.1
789 - acc: 0.2031 - val_loss: 2.0516 - val_acc: 0.4544
Epoch 19/20
60000/60000 [=====] - 5s 85us/step - loss: 2.1
507 - acc: 0.2175 - val_loss: 2.0006 - val_acc: 0.4463
Epoch 20/20
60000/60000 [=====] - 5s 85us/step - loss: 2.1
155 - acc: 0.2314 - val_loss: 1.9438 - val_acc: 0.4608
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

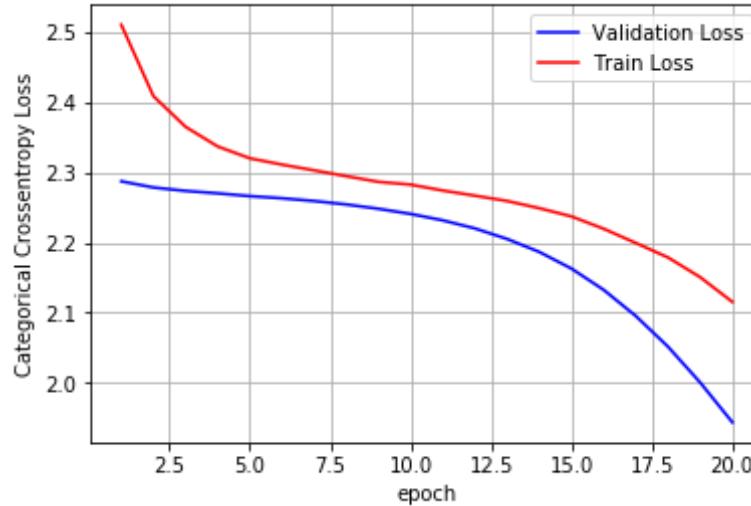
```
Test score: 1.9437882654190064
Test accuracy: 0.4608
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

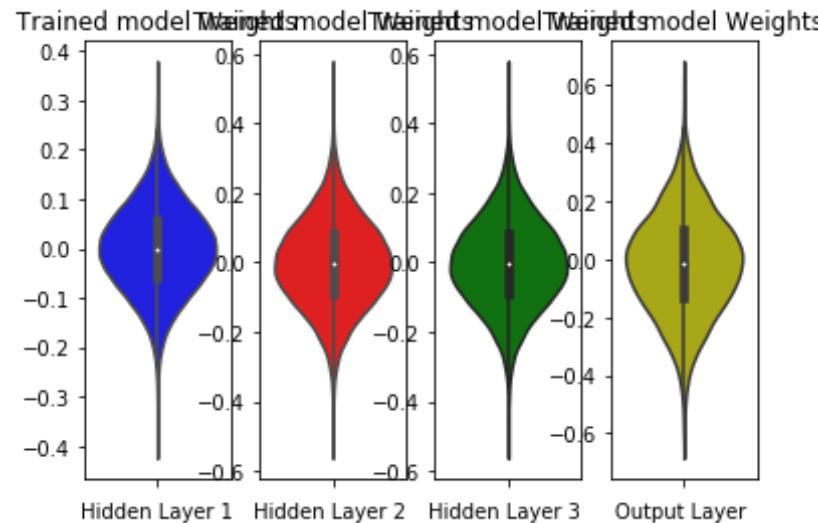
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
```

```

ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Architecture 3: 784-512-256-128-64-32-10

#### Model 1: sigmoid activation + AdamOptimizer with SGD Weight

```

In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

```

```
model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_72"

Layer (type)	Output Shape	Param #
dense_232 (Dense)	(None, 512)	401920
dense_233 (Dense)	(None, 256)	131328
dense_234 (Dense)	(None, 128)	32896
dense_235 (Dense)	(None, 64)	8256
dense_236 (Dense)	(None, 32)	2080
dense_237 (Dense)	(None, 10)	330

Total params: 576,810  
Trainable params: 576,810  
Non-trainable params: 0

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 16s 263us/step - loss:  
1.2712 - acc: 0.6198 - val_loss: 0.5745 - val_acc: 0.8861  
Epoch 2/20  
60000/60000 [=====] - 8s 135us/step - loss: 0.  
3896 - acc: 0.9166 - val_loss: 0.2913 - val_acc: 0.9354  
Epoch 3/20  
60000/60000 [=====] - 8s 136us/step - loss: 0.  
2297 - acc: 0.9460 - val_loss: 0.1945 - val_acc: 0.9533  
Epoch 4/20  
60000/60000 [=====] - 8s 136us/step - loss: 0.  
1603 - acc: 0.9611 - val_loss: 0.1773 - val_acc: 0.9536  
Epoch 5/20  
60000/60000 [=====] - 8s 138us/step - loss: 0.  
1197 - acc: 0.9702 - val_loss: 0.1351 - val_acc: 0.9649  
Epoch 6/20  
60000/60000 [=====] - 8s 138us/step - loss: 0.  
0957 - acc: 0.9755 - val_loss: 0.1363 - val_acc: 0.9639  
Epoch 7/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
0799 - acc: 0.9789 - val_loss: 0.1266 - val_acc: 0.9656  
Epoch 8/20  
60000/60000 [=====] - 8s 137us/step - loss: 0.  
0639 - acc: 0.9833 - val_loss: 0.1027 - val_acc: 0.9727  
Epoch 9/20  
60000/60000 [=====] - 8s 138us/step - loss: 0.  
0542 - acc: 0.9855 - val_loss: 0.1190 - val_acc: 0.9686  
Epoch 10/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
0467 - acc: 0.9874 - val_loss: 0.1099 - val_acc: 0.9714  
Epoch 11/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
0398 - acc: 0.9893 - val_loss: 0.1066 - val_acc: 0.9737  
Epoch 12/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.
```

```
0340 - acc: 0.9910 - val_loss: 0.0897 - val_acc: 0.9780
Epoch 13/20
60000/60000 [=====] - 8s 137us/step - loss: 0.
0302 - acc: 0.9914 - val_loss: 0.0950 - val_acc: 0.9760
Epoch 14/20
60000/60000 [=====] - 8s 137us/step - loss: 0.
0248 - acc: 0.9933 - val_loss: 0.1018 - val_acc: 0.9743
Epoch 15/20
60000/60000 [=====] - 8s 137us/step - loss: 0.
0235 - acc: 0.9936 - val_loss: 0.1162 - val_acc: 0.9726
Epoch 16/20
60000/60000 [=====] - 8s 136us/step - loss: 0.
0200 - acc: 0.9947 - val_loss: 0.0968 - val_acc: 0.9770
Epoch 17/20
60000/60000 [=====] - 8s 138us/step - loss: 0.
0175 - acc: 0.9952 - val_loss: 0.0993 - val_acc: 0.9776
Epoch 18/20
60000/60000 [=====] - 8s 138us/step - loss: 0.
0151 - acc: 0.9959 - val_loss: 0.1128 - val_acc: 0.9737
Epoch 19/20
60000/60000 [=====] - 8s 136us/step - loss: 0.
0169 - acc: 0.9950 - val_loss: 0.1149 - val_acc: 0.9750
Epoch 20/20
60000/60000 [=====] - 8s 137us/step - loss: 0.
0130 - acc: 0.9965 - val_loss: 0.1000 - val_acc: 0.9791
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

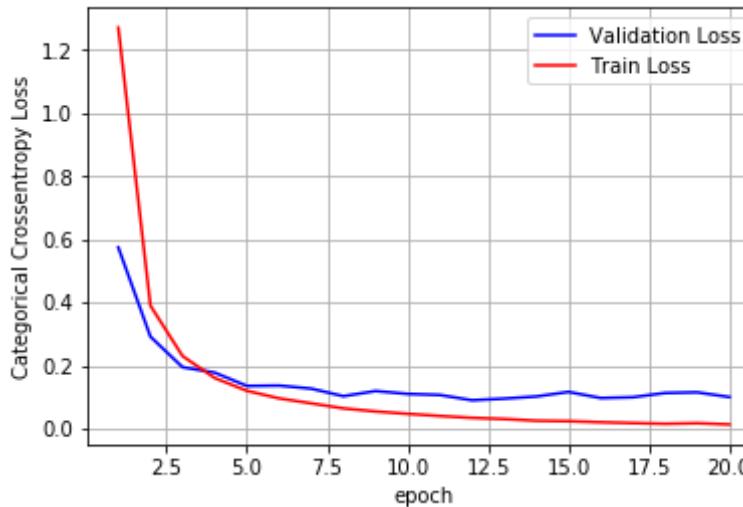
```
Test score: 0.09996185349542648
Test accuracy: 0.9791
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
```

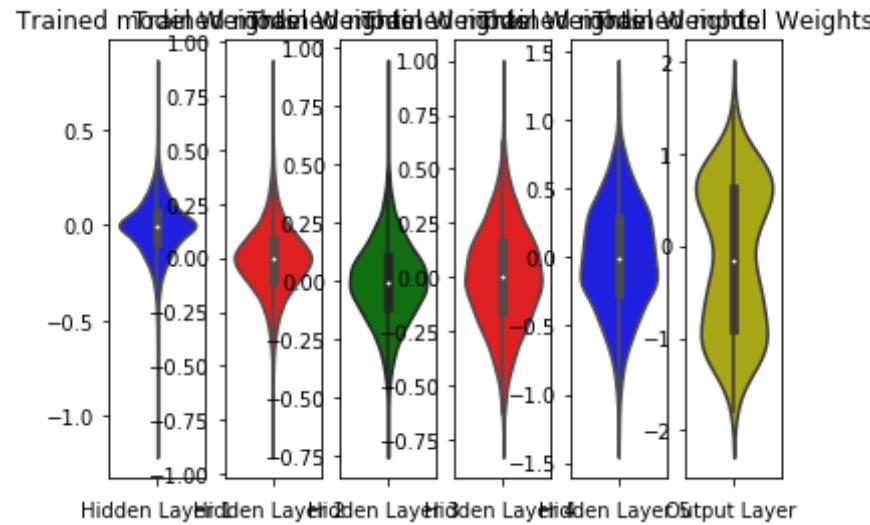
```
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 1 : sigmoid activation + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.051, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.144, seed=None)))
```

```
model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()  
  
Model: "sequential_73"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_238 (Dense)	(None, 512)	401920
dense_239 (Dense)	(None, 256)	131328
dense_240 (Dense)	(None, 128)	32896
dense_241 (Dense)	(None, 64)	8256
dense_242 (Dense)	(None, 32)	2080
dense_243 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 13s 223us/step - loss:  
2.3271 - acc: 0.1055 - val_loss: 2.3012 - val_acc: 0.1135  
Epoch 2/20  
60000/60000 [=====] - 7s 111us/step - loss: 2.  
3015 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135  
Epoch 3/20  
60000/60000 [=====] - 7s 111us/step - loss: 2.
```

```
3015 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 4/20
60000/60000 [=====] - 7s 110us/step - loss: 2.
3014 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 7s 111us/step - loss: 2.
3014 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 6/20
60000/60000 [=====] - 7s 110us/step - loss: 2.
3013 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
3014 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
3013 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
3013 - acc: 0.1124 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 7s 111us/step - loss: 2.
3013 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 12/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
3013 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 13/20
60000/60000 [=====] - 7s 111us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 15/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3008 - val_acc: 0.1135
Epoch 16/20
60000/60000 [=====] - 7s 111us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3008 - val_acc: 0.1135
```

```
Epoch 17/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
3012 - acc: 0.1124 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 18/20
60000/60000 [=====] - 7s 110us/step - loss: 2.
3011 - acc: 0.1124 - val_loss: 2.3009 - val_acc: 0.1135
Epoch 19/20
60000/60000 [=====] - 7s 110us/step - loss: 2.
3011 - acc: 0.1124 - val_loss: 2.3007 - val_acc: 0.1135
Epoch 20/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
3011 - acc: 0.1124 - val_loss: 2.3010 - val_acc: 0.1135
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

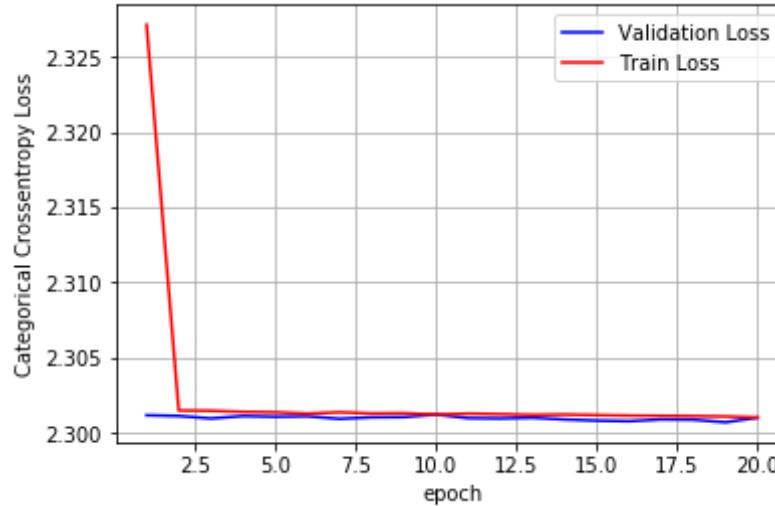
```
Test score: 2.3010412399291993
Test accuracy: 0.1135
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

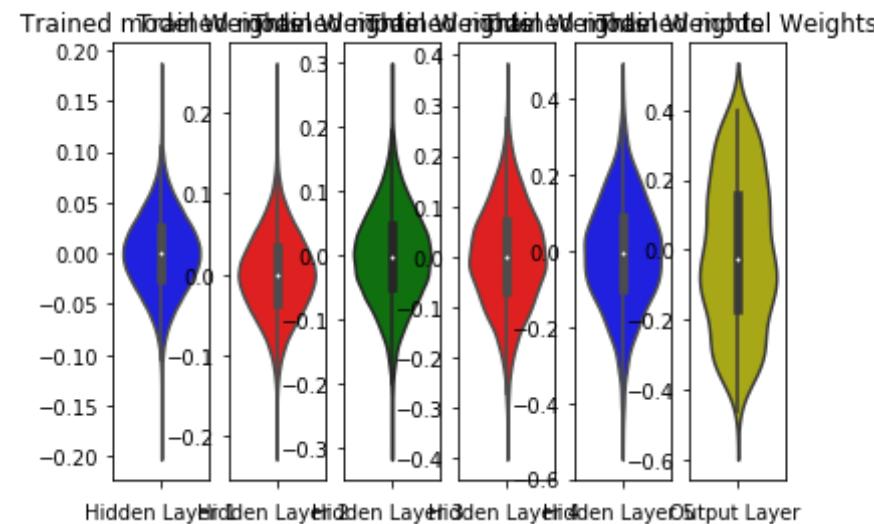
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : sigmoid activation + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.088, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.246, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_74"

Layer (type)	Output Shape	Param #
dense_244 (Dense)	(None, 512)	401920
dense_245 (Dense)	(None, 256)	131328
dense_246 (Dense)	(None, 128)	32896
dense_247 (Dense)	(None, 64)	8256
dense_248 (Dense)	(None, 32)	2080
dense_249 (Dense)	(None, 10)	330

```
Total params: 576,810  
Trainable params: 576,810  
Non-trainable params: 0
```

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 15s 253us/step - loss: 1.0441 - acc: 0.7439 - val_loss: 0.3445 - val_acc: 0.9273  
Epoch 2/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.2599 - acc: 0.9368 - val_loss: 0.1950 - val_acc: 0.9523  
Epoch 3/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.1617 - acc: 0.9587 - val_loss: 0.1471 - val_acc: 0.9618  
Epoch 4/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.1166 - acc: 0.9687 - val_loss: 0.1466 - val_acc: 0.9597  
Epoch 5/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.0900 - acc: 0.9760 - val_loss: 0.1027 - val_acc: 0.9727  
Epoch 6/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.0728 - acc: 0.9806 - val_loss: 0.1036 - val_acc: 0.9706  
Epoch 7/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.0591 - acc: 0.9838 - val_loss: 0.0923 - val_acc: 0.9749  
Epoch 8/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.0480 - acc: 0.9873 - val_loss: 0.0876 - val_acc: 0.9773  
Epoch 9/20  
60000/60000 [=====] - 9s 143us/step - loss: 0.0413 - acc: 0.9887 - val_loss: 0.0909 - val_acc: 0.9753  
Epoch 10/20
```

```
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0353 - acc: 0.9902 - val_loss: 0.0840 - val_acc: 0.9778  
Epoch 11/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0300 - acc: 0.9916 - val_loss: 0.0862 - val_acc: 0.9764  
Epoch 12/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.  
0256 - acc: 0.9929 - val_loss: 0.0790 - val_acc: 0.9794  
Epoch 13/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.  
0219 - acc: 0.9941 - val_loss: 0.0781 - val_acc: 0.9805  
Epoch 14/20  
60000/60000 [=====] - 9s 143us/step - loss: 0.  
0208 - acc: 0.9942 - val_loss: 0.0892 - val_acc: 0.9763  
Epoch 15/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0168 - acc: 0.9950 - val_loss: 0.0786 - val_acc: 0.9802  
Epoch 16/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0144 - acc: 0.9960 - val_loss: 0.0940 - val_acc: 0.9770  
Epoch 17/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0147 - acc: 0.9958 - val_loss: 0.1133 - val_acc: 0.9736  
Epoch 18/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.  
0128 - acc: 0.9966 - val_loss: 0.0849 - val_acc: 0.9795  
Epoch 19/20  
60000/60000 [=====] - 8s 142us/step - loss: 0.  
0114 - acc: 0.9966 - val_loss: 0.0920 - val_acc: 0.9792  
Epoch 20/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.  
0088 - acc: 0.9975 - val_loss: 0.0914 - val_acc: 0.9805
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

Test score: 0.09138628843901679

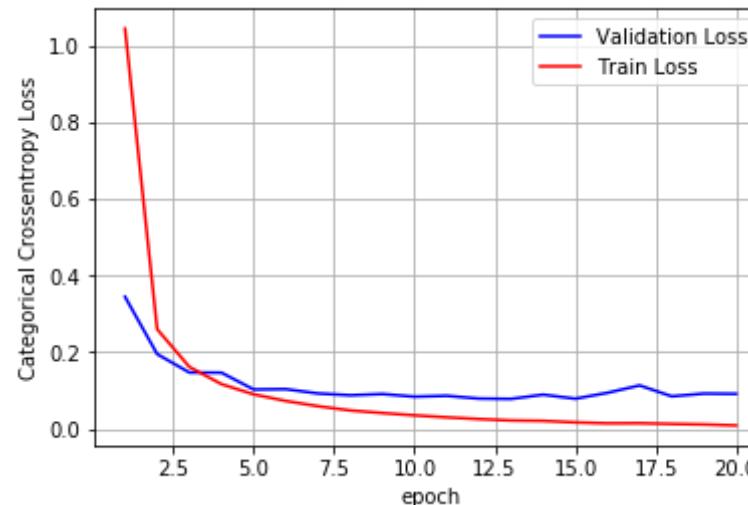
Test accuracy: 0.9805

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
```

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

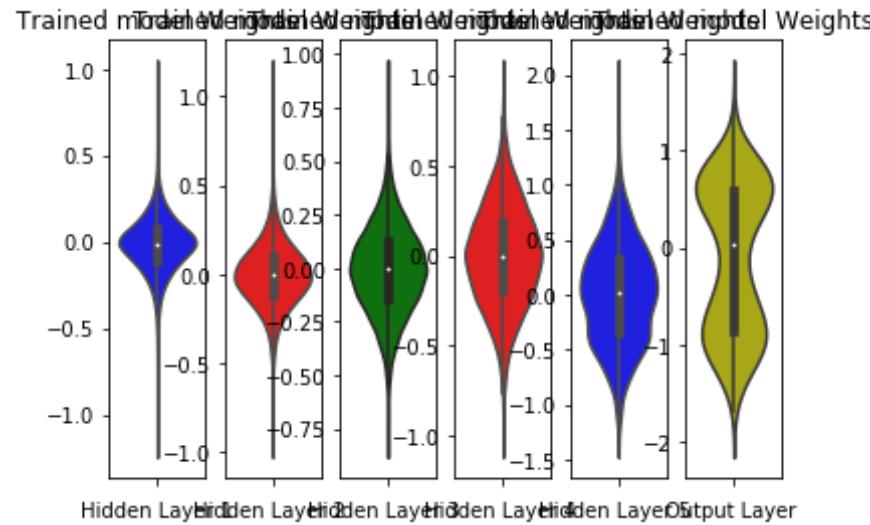
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 1 : sigmoid activation + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))
```

```
model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()  
  
Model: "sequential_75"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_250 (Dense)	(None, 512)	401920
dense_251 (Dense)	(None, 256)	131328
dense_252 (Dense)	(None, 128)	32896
dense_253 (Dense)	(None, 64)	8256
dense_254 (Dense)	(None, 32)	2080
dense_255 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 14s 226us/step - loss:  
2.3087 - acc: 0.1088 - val_loss: 2.3002 - val_acc: 0.1135  
Epoch 2/20  
60000/60000 [=====] - 7s 113us/step - loss: 2.  
3000 - acc: 0.1124 - val_loss: 2.2993 - val_acc: 0.1135  
Epoch 3/20  
60000/60000 [=====] - 7s 113us/step - loss: 2.
```

```
2993 - acc: 0.1124 - val_loss: 2.2987 - val_acc: 0.1135
Epoch 4/20
60000/60000 [=====] - 7s 115us/step - loss: 2.
2986 - acc: 0.1124 - val_loss: 2.2980 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 7s 114us/step - loss: 2.
2980 - acc: 0.1124 - val_loss: 2.2972 - val_acc: 0.1135
Epoch 6/20
60000/60000 [=====] - 7s 115us/step - loss: 2.
2972 - acc: 0.1131 - val_loss: 2.2965 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 7s 114us/step - loss: 2.
2964 - acc: 0.1124 - val_loss: 2.2955 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 7s 114us/step - loss: 2.
2955 - acc: 0.1124 - val_loss: 2.2949 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 7s 114us/step - loss: 2.
2947 - acc: 0.1124 - val_loss: 2.2938 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2936 - acc: 0.1131 - val_loss: 2.2925 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2925 - acc: 0.1128 - val_loss: 2.2914 - val_acc: 0.1135
Epoch 12/20
60000/60000 [=====] - 7s 111us/step - loss: 2.
2912 - acc: 0.1124 - val_loss: 2.2900 - val_acc: 0.1135
Epoch 13/20
60000/60000 [=====] - 7s 114us/step - loss: 2.
2897 - acc: 0.1134 - val_loss: 2.2884 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2880 - acc: 0.1185 - val_loss: 2.2865 - val_acc: 0.1135
Epoch 15/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2861 - acc: 0.1127 - val_loss: 2.2843 - val_acc: 0.1135
Epoch 16/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
2838 - acc: 0.1160 - val_loss: 2.2818 - val_acc: 0.2080
```

```
Epoch 17/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2810 - acc: 0.1246 - val_loss: 2.2787 - val_acc: 0.1153
Epoch 18/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
2777 - acc: 0.1322 - val_loss: 2.2750 - val_acc: 0.1147
Epoch 19/20
60000/60000 [=====] - 7s 112us/step - loss: 2.
2736 - acc: 0.1293 - val_loss: 2.2703 - val_acc: 0.1303
Epoch 20/20
60000/60000 [=====] - 7s 113us/step - loss: 2.
2685 - acc: 0.1623 - val_loss: 2.2643 - val_acc: 0.1830
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

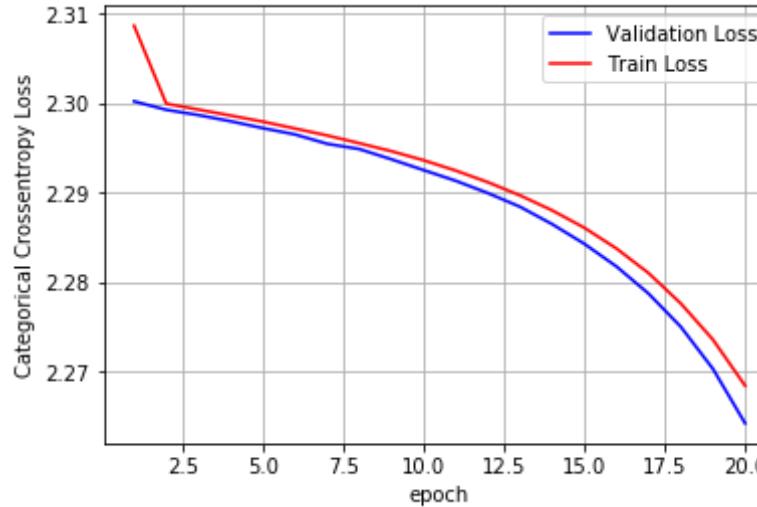
```
Test score: 2.264263961029053
Test accuracy: 0.183
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

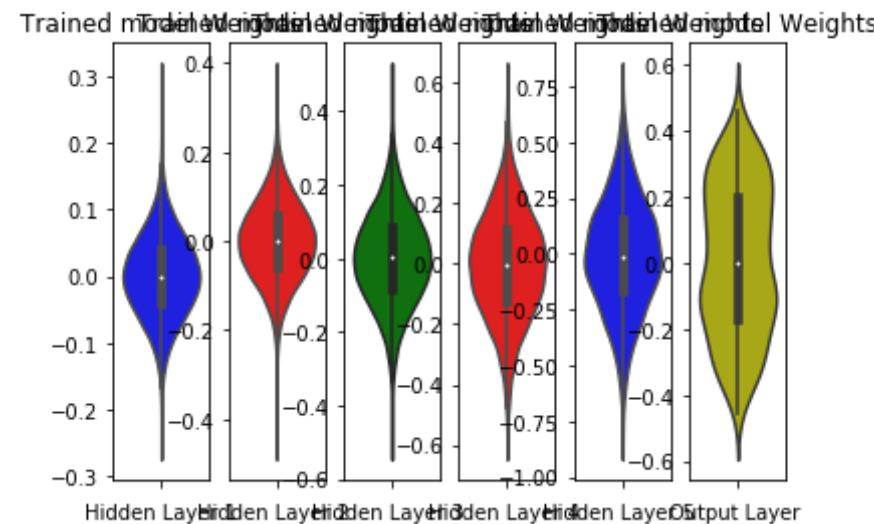
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 2: Relu activation + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_76"

Layer (type)	Output Shape	Param #
<hr/>		
dense_256 (Dense)	(None, 512)	401920
dense_257 (Dense)	(None, 256)	131328
dense_258 (Dense)	(None, 128)	32896
dense_259 (Dense)	(None, 64)	8256
dense_260 (Dense)	(None, 32)	2080
<hr/>		

```
dense_261 (Dense)           (None, 10)          330
=====
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 16s 265us/step - loss: 0.2845 - acc: 0.9142 - val_loss: 0.1108 - val_acc: 0.9663
Epoch 2/20
60000/60000 [=====] - 9s 143us/step - loss: 0.0967 - acc: 0.9707 - val_loss: 0.0917 - val_acc: 0.9708
Epoch 3/20
60000/60000 [=====] - 8s 140us/step - loss: 0.0615 - acc: 0.9810 - val_loss: 0.0792 - val_acc: 0.9761
Epoch 4/20
60000/60000 [=====] - 8s 140us/step - loss: 0.0471 - acc: 0.9851 - val_loss: 0.0773 - val_acc: 0.9767
Epoch 5/20
60000/60000 [=====] - 8s 141us/step - loss: 0.0388 - acc: 0.9877 - val_loss: 0.0809 - val_acc: 0.9781
Epoch 6/20
60000/60000 [=====] - 9s 145us/step - loss: 0.0308 - acc: 0.9900 - val_loss: 0.0755 - val_acc: 0.9793
Epoch 7/20
60000/60000 [=====] - 9s 145us/step - loss: 0.0253 - acc: 0.9922 - val_loss: 0.0802 - val_acc: 0.9823
Epoch 8/20
60000/60000 [=====] - 9s 143us/step - loss: 0.0234 - acc: 0.9924 - val_loss: 0.0925 - val_acc: 0.9772
Epoch 9/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
```

```
0226 - acc: 0.9929 - val_loss: 0.0879 - val_acc: 0.9792
Epoch 10/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
0187 - acc: 0.9944 - val_loss: 0.0884 - val_acc: 0.9794
Epoch 11/20
60000/60000 [=====] - 9s 144us/step - loss: 0.
0170 - acc: 0.9944 - val_loss: 0.0830 - val_acc: 0.9807
Epoch 12/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
0183 - acc: 0.9943 - val_loss: 0.0863 - val_acc: 0.9811
Epoch 13/20
60000/60000 [=====] - 9s 142us/step - loss: 0.
0148 - acc: 0.9953 - val_loss: 0.0837 - val_acc: 0.9801
Epoch 14/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
0151 - acc: 0.9955 - val_loss: 0.0861 - val_acc: 0.9797
Epoch 15/20
60000/60000 [=====] - 9s 142us/step - loss: 0.
0098 - acc: 0.9969 - val_loss: 0.0861 - val_acc: 0.9829
Epoch 16/20
60000/60000 [=====] - 9s 145us/step - loss: 0.
0107 - acc: 0.9967 - val_loss: 0.0980 - val_acc: 0.9803
Epoch 17/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
0130 - acc: 0.9960 - val_loss: 0.1090 - val_acc: 0.9784
Epoch 18/20
60000/60000 [=====] - 8s 141us/step - loss: 0.
0121 - acc: 0.9967 - val_loss: 0.0828 - val_acc: 0.9812
Epoch 19/20
60000/60000 [=====] - 9s 144us/step - loss: 0.
0087 - acc: 0.9975 - val_loss: 0.0827 - val_acc: 0.9834
Epoch 20/20
60000/60000 [=====] - 9s 143us/step - loss: 0.
0111 - acc: 0.9966 - val_loss: 0.0918 - val_acc: 0.9808
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

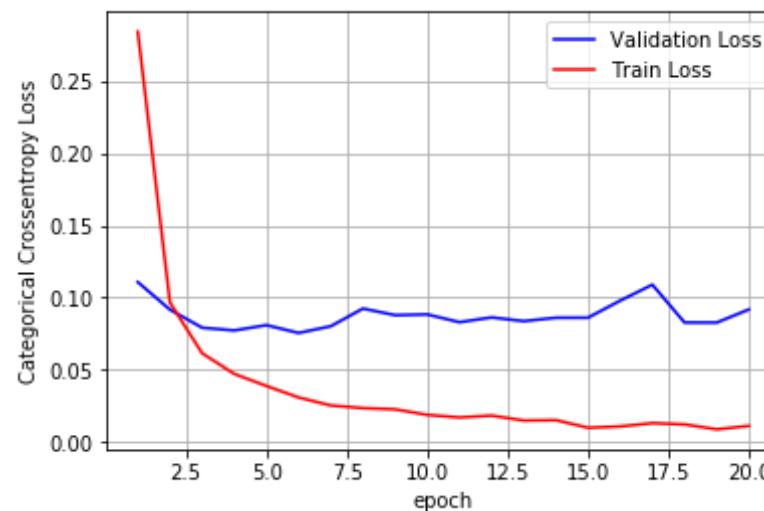
```
Test score: 0.09181766118927444
Test accuracy: 0.9808
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
```

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

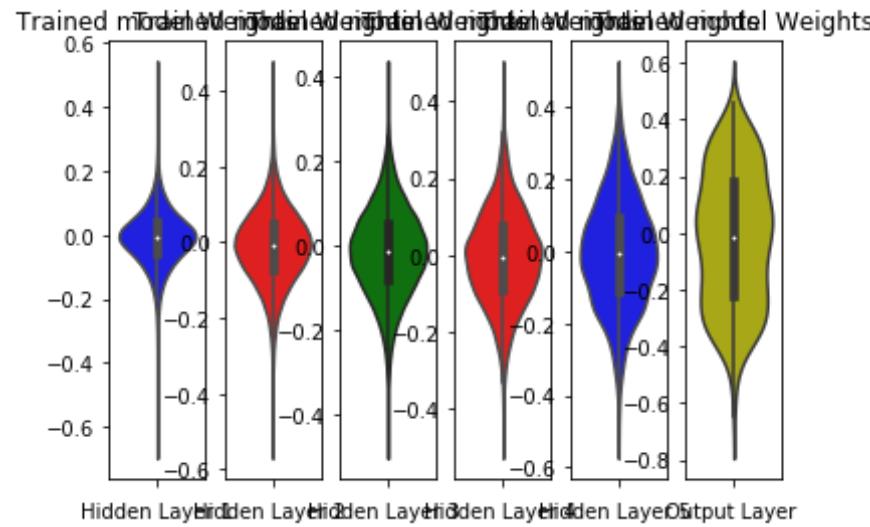
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 2 : relu activation + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))
```

```
model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()  
  
Model: "sequential_77"
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_262 (Dense)	(None, 512)	401920
dense_263 (Dense)	(None, 256)	131328
dense_264 (Dense)	(None, 128)	32896
dense_265 (Dense)	(None, 64)	8256
dense_266 (Dense)	(None, 32)	2080
dense_267 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])  
  
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n  
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 14s 233us/step - loss:  
1.3393 - acc: 0.6293 - val_loss: 0.5091 - val_acc: 0.8592  
Epoch 2/20  
60000/60000 [=====] - 7s 117us/step - loss: 0.  
4106 - acc: 0.8830 - val_loss: 0.3277 - val_acc: 0.9056  
Epoch 3/20  
60000/60000 [=====] - 7s 116us/step - loss: 0.
```

```
3083 - acc: 0.9101 - val_loss: 0.2764 - val_acc: 0.9201
Epoch 4/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
2594 - acc: 0.9239 - val_loss: 0.2346 - val_acc: 0.9318
Epoch 5/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
2243 - acc: 0.9340 - val_loss: 0.2072 - val_acc: 0.9396
Epoch 6/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
1979 - acc: 0.9425 - val_loss: 0.1891 - val_acc: 0.9439
Epoch 7/20
60000/60000 [=====] - 7s 119us/step - loss: 0.
1749 - acc: 0.9489 - val_loss: 0.1669 - val_acc: 0.9498
Epoch 8/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
1577 - acc: 0.9541 - val_loss: 0.1569 - val_acc: 0.9521
Epoch 9/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
1428 - acc: 0.9581 - val_loss: 0.1667 - val_acc: 0.9483
Epoch 10/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
1310 - acc: 0.9621 - val_loss: 0.1393 - val_acc: 0.9588
Epoch 11/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
1209 - acc: 0.9649 - val_loss: 0.1312 - val_acc: 0.9610
Epoch 12/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
1112 - acc: 0.9677 - val_loss: 0.1345 - val_acc: 0.9594
Epoch 13/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
1035 - acc: 0.9693 - val_loss: 0.1275 - val_acc: 0.9603
Epoch 14/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
0963 - acc: 0.9723 - val_loss: 0.1124 - val_acc: 0.9668
Epoch 15/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
0901 - acc: 0.9736 - val_loss: 0.1102 - val_acc: 0.9661
Epoch 16/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0841 - acc: 0.9756 - val_loss: 0.1156 - val_acc: 0.9649
```

```
Epoch 17/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
0786 - acc: 0.9769 - val_loss: 0.1054 - val_acc: 0.9678
Epoch 18/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
0736 - acc: 0.9786 - val_loss: 0.1048 - val_acc: 0.9686
Epoch 19/20
60000/60000 [=====] - 7s 118us/step - loss: 0.
0692 - acc: 0.9802 - val_loss: 0.1057 - val_acc: 0.9692
Epoch 20/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0648 - acc: 0.9810 - val_loss: 0.0989 - val_acc: 0.9698
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

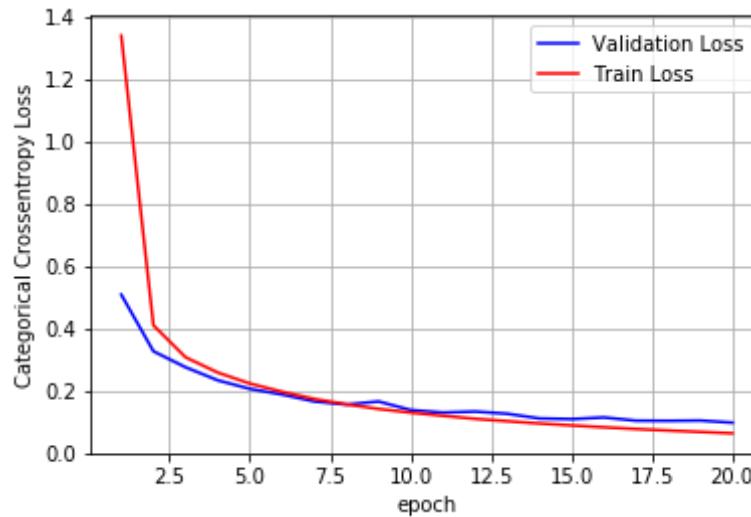
```
Test score: 0.09889934923872351
Test accuracy: 0.9698
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

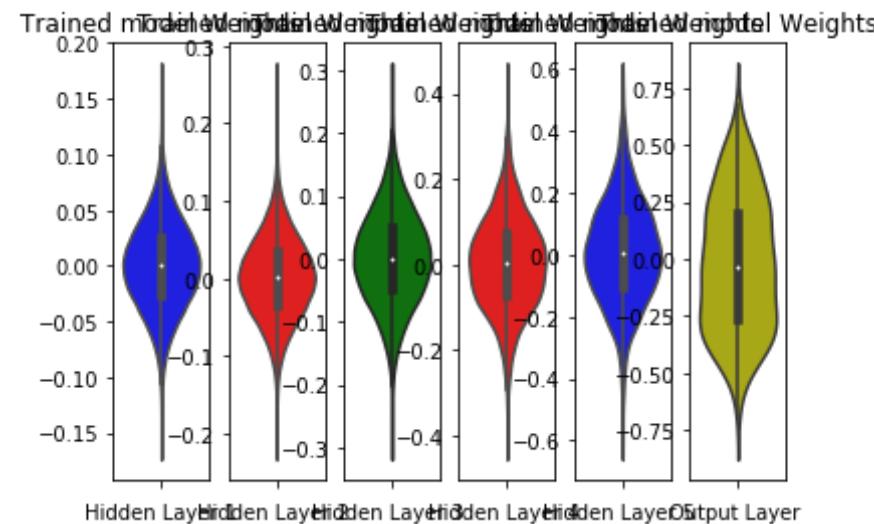
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 1 : relu activation + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_78"

Layer (type)	Output Shape	Param #
dense_268 (Dense)	(None, 512)	401920
dense_269 (Dense)	(None, 256)	131328
dense_270 (Dense)	(None, 128)	32896
dense_271 (Dense)	(None, 64)	8256
dense_272 (Dense)	(None, 32)	2080
dense_273 (Dense)	(None, 10)	330

```
Total params: 576,810  
Trainable params: 576,810  
Non-trainable params: 0
```

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 16s 262us/step - loss: 0.2790 - acc: 0.9162 - val_loss: 0.1251 - val_acc: 0.9616  
Epoch 2/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.0994 - acc: 0.9697 - val_loss: 0.0844 - val_acc: 0.9747  
Epoch 3/20  
60000/60000 [=====] - 9s 146us/step - loss: 0.0627 - acc: 0.9802 - val_loss: 0.0868 - val_acc: 0.9735  
Epoch 4/20  
60000/60000 [=====] - 9s 145us/step - loss: 0.0482 - acc: 0.9850 - val_loss: 0.0991 - val_acc: 0.9714  
Epoch 5/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.0383 - acc: 0.9878 - val_loss: 0.0938 - val_acc: 0.9732  
Epoch 6/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.0320 - acc: 0.9904 - val_loss: 0.1108 - val_acc: 0.9708  
Epoch 7/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.0337 - acc: 0.9890 - val_loss: 0.0799 - val_acc: 0.9801  
Epoch 8/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.0247 - acc: 0.9925 - val_loss: 0.0960 - val_acc: 0.9756  
Epoch 9/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.0249 - acc: 0.9922 - val_loss: 0.1052 - val_acc: 0.9725  
Epoch 10/20
```

```
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0187 - acc: 0.9938 - val_loss: 0.0739 - val_acc: 0.9820  
Epoch 11/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0189 - acc: 0.9938 - val_loss: 0.0913 - val_acc: 0.9786  
Epoch 12/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0186 - acc: 0.9939 - val_loss: 0.0848 - val_acc: 0.9807  
Epoch 13/20  
60000/60000 [=====] - 9s 142us/step - loss: 0.  
0217 - acc: 0.9927 - val_loss: 0.0923 - val_acc: 0.9797  
Epoch 14/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0161 - acc: 0.9951 - val_loss: 0.1004 - val_acc: 0.9765  
Epoch 15/20  
60000/60000 [=====] - 8s 137us/step - loss: 0.  
0131 - acc: 0.9958 - val_loss: 0.0961 - val_acc: 0.9787  
Epoch 16/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0168 - acc: 0.9948 - val_loss: 0.1009 - val_acc: 0.9749  
Epoch 17/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0112 - acc: 0.9965 - val_loss: 0.1025 - val_acc: 0.9781  
Epoch 18/20  
60000/60000 [=====] - 8s 141us/step - loss: 0.  
0138 - acc: 0.9956 - val_loss: 0.0949 - val_acc: 0.9804  
Epoch 19/20  
60000/60000 [=====] - 8s 140us/step - loss: 0.  
0100 - acc: 0.9970 - val_loss: 0.1130 - val_acc: 0.9775  
Epoch 20/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
0182 - acc: 0.9944 - val_loss: 0.0811 - val_acc: 0.9832
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

Test score: 0.08113763677500264

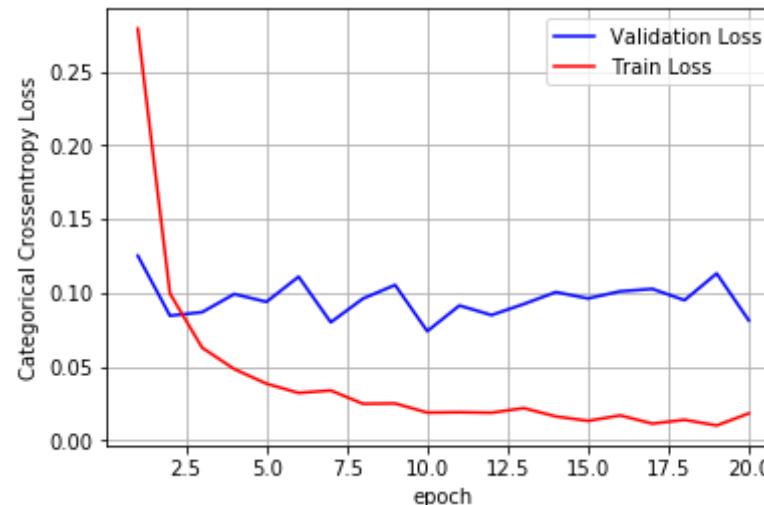
Test accuracy: 0.9832

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)
```

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

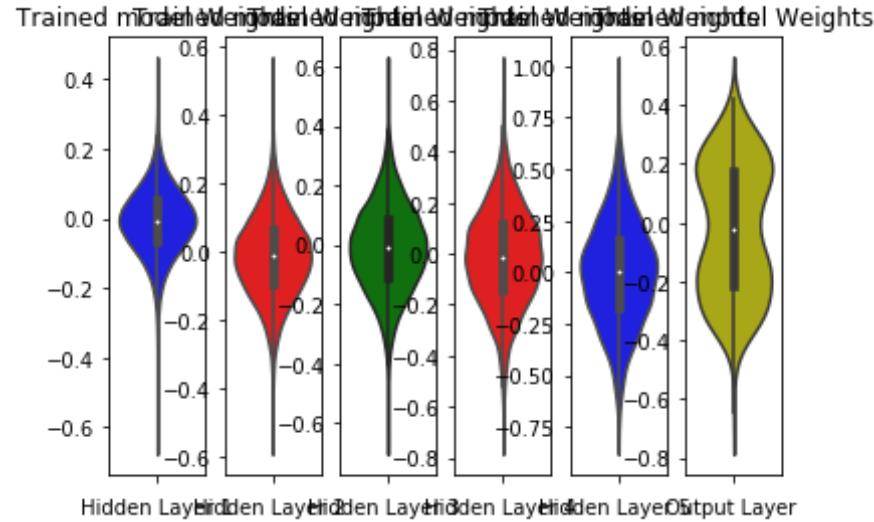
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 2 : relu activation + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))

model_1.add(Dense(output_dim, activation='softmax'))
```

```
model_1.summary()
```

Model: "sequential\_79"

Layer (type)	Output Shape	Param #
<hr/>		
dense_274 (Dense)	(None, 512)	401920
dense_275 (Dense)	(None, 256)	131328
dense_276 (Dense)	(None, 128)	32896
dense_277 (Dense)	(None, 64)	8256
dense_278 (Dense)	(None, 32)	2080
dense_279 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 14s 229us/step - loss: 0.7107 - acc: 0.7839 - val\_loss: 0.2985 - val\_acc: 0.9172  
Epoch 2/20  
60000/60000 [=====] - 7s 112us/step - loss: 0.2700 - acc: 0.9200 - val\_loss: 0.2416 - val\_acc: 0.9275  
Epoch 3/20  
60000/60000 [=====] - 7s 114us/step - loss: 0.2097 - acc: 0.9370 - val\_loss: 0.1927 - val\_acc: 0.9437

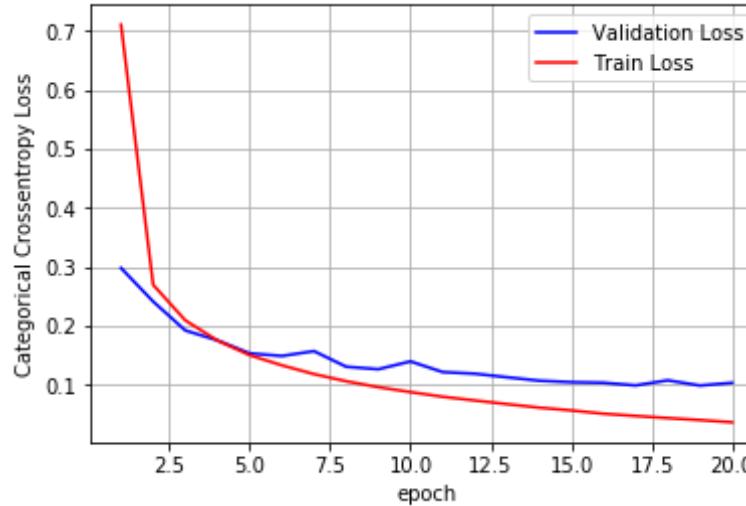
```
Epoch 4/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
1755 - acc: 0.9482 - val_loss: 0.1759 - val_acc: 0.9485
Epoch 5/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
1508 - acc: 0.9548 - val_loss: 0.1539 - val_acc: 0.9543
Epoch 6/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
1336 - acc: 0.9601 - val_loss: 0.1495 - val_acc: 0.9555
Epoch 7/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
1188 - acc: 0.9646 - val_loss: 0.1577 - val_acc: 0.9533
Epoch 8/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
1067 - acc: 0.9683 - val_loss: 0.1313 - val_acc: 0.9606
Epoch 9/20
60000/60000 [=====] - 7s 112us/step - loss: 0.
0966 - acc: 0.9716 - val_loss: 0.1270 - val_acc: 0.9617
Epoch 10/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
0883 - acc: 0.9738 - val_loss: 0.1403 - val_acc: 0.9572
Epoch 11/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
0804 - acc: 0.9769 - val_loss: 0.1222 - val_acc: 0.9637
Epoch 12/20
60000/60000 [=====] - 7s 113us/step - loss: 0.
0741 - acc: 0.9786 - val_loss: 0.1194 - val_acc: 0.9638
Epoch 13/20
60000/60000 [=====] - 7s 115us/step - loss: 0.
0679 - acc: 0.9806 - val_loss: 0.1134 - val_acc: 0.9634
Epoch 14/20
60000/60000 [=====] - 7s 114us/step - loss: 0.
0619 - acc: 0.9823 - val_loss: 0.1076 - val_acc: 0.9669
Epoch 15/20
60000/60000 [=====] - 7s 116us/step - loss: 0.
0572 - acc: 0.9843 - val_loss: 0.1051 - val_acc: 0.9665
Epoch 16/20
60000/60000 [=====] - 7s 117us/step - loss: 0.
0516 - acc: 0.9859 - val_loss: 0.1041 - val_acc: 0.9670
Epoch 17/20
```

```
60000/60000 [=====] - 7s 115us/step - loss: 0.  
0479 - acc: 0.9869 - val_loss: 0.0996 - val_acc: 0.9691  
Epoch 18/20  
60000/60000 [=====] - 7s 114us/step - loss: 0.  
0442 - acc: 0.9885 - val_loss: 0.1082 - val_acc: 0.9659  
Epoch 19/20  
60000/60000 [=====] - 7s 115us/step - loss: 0.  
0409 - acc: 0.9894 - val_loss: 0.0994 - val_acc: 0.9693  
Epoch 20/20  
60000/60000 [=====] - 7s 114us/step - loss: 0.  
0371 - acc: 0.9906 - val_loss: 0.1040 - val_acc: 0.9681
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.10397074847645127  
Test accuracy: 0.9681
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

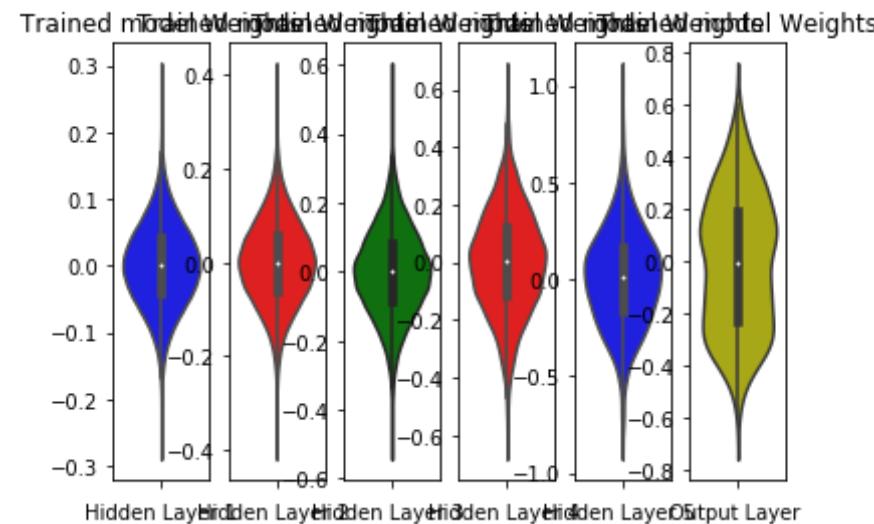
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers.normalization import BatchNormalization

model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_80"

Layer (type)	Output Shape	Param #
<hr/>		
dense_280 (Dense)	(None, 512)	401920
batch_normalization_55 (Batch Normalization)	(None, 512)	2048

dense_281 (Dense)	(None, 256)	131328
batch_normalization_56 (Batch Normalization)	(None, 256)	1024
dense_282 (Dense)	(None, 128)	32896
batch_normalization_57 (Batch Normalization)	(None, 128)	512
dense_283 (Dense)	(None, 64)	8256
batch_normalization_58 (Batch Normalization)	(None, 64)	256
dense_284 (Dense)	(None, 32)	2080
batch_normalization_59 (Batch Normalization)	(None, 32)	128
dense_285 (Dense)	(None, 10)	330
=====		
Total params: 580,778		
Trainable params: 578,794		
Non-trainable params: 1,984		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 19s 318us/step - loss: 0.2678 - acc: 0.9243 - val_loss: 0.1684 - val_acc: 0.9487
Epoch 2/20
60000/60000 [=====] - 10s 174us/step - loss: 0.1275 - acc: 0.9618 - val_loss: 0.1235 - val_acc: 0.9624
Epoch 3/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0873 - acc: 0.9733 - val_loss: 0.0939 - val_acc: 0.9709
Epoch 4/20
```

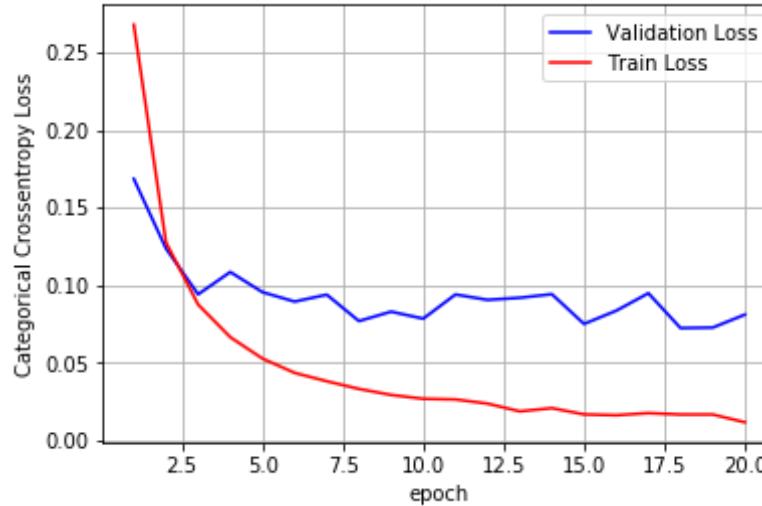
```
Epoch 4/20
60000/60000 [=====] - 10s 175us/step - loss: 0.0663 - acc: 0.9794 - val_loss: 0.1083 - val_acc: 0.9665
Epoch 5/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0525 - acc: 0.9832 - val_loss: 0.0953 - val_acc: 0.9729
Epoch 6/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0433 - acc: 0.9853 - val_loss: 0.0893 - val_acc: 0.9734
Epoch 7/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0379 - acc: 0.9876 - val_loss: 0.0937 - val_acc: 0.9732
Epoch 8/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0330 - acc: 0.9893 - val_loss: 0.0767 - val_acc: 0.9787
Epoch 9/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0291 - acc: 0.9903 - val_loss: 0.0828 - val_acc: 0.9753
Epoch 10/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0266 - acc: 0.9911 - val_loss: 0.0783 - val_acc: 0.9778
Epoch 11/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0262 - acc: 0.9916 - val_loss: 0.0938 - val_acc: 0.9739
Epoch 12/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0235 - acc: 0.9920 - val_loss: 0.0904 - val_acc: 0.9774
Epoch 13/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0185 - acc: 0.9937 - val_loss: 0.0917 - val_acc: 0.9773
Epoch 14/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0206 - acc: 0.9929 - val_loss: 0.0940 - val_acc: 0.9754
Epoch 15/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0165 - acc: 0.9945 - val_loss: 0.0749 - val_acc: 0.9798
Epoch 16/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0160 - acc: 0.9950 - val_loss: 0.0834 - val_acc: 0.9780
Epoch 17/20
```

```
60000/60000 [=====] - 11s 178us/step - loss:  
0.0173 - acc: 0.9942 - val_loss: 0.0947 - val_acc: 0.9771  
Epoch 18/20  
60000/60000 [=====] - 11s 180us/step - loss:  
0.0165 - acc: 0.9943 - val_loss: 0.0722 - val_acc: 0.9809  
Epoch 19/20  
60000/60000 [=====] - 11s 180us/step - loss:  
0.0165 - acc: 0.9943 - val_loss: 0.0725 - val_acc: 0.9807  
Epoch 20/20  
60000/60000 [=====] - 11s 180us/step - loss:  
0.0114 - acc: 0.9964 - val_loss: 0.0809 - val_acc: 0.9807
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.08085234306444763  
Test accuracy: 0.9807
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

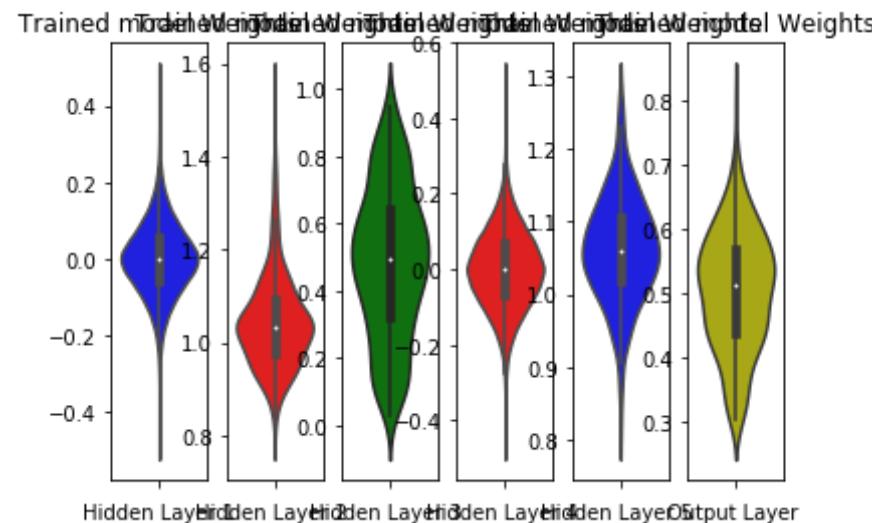
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
                 kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_81"

Layer (type)	Output Shape	Param #
dense_286 (Dense)	(None, 512)	401920
batch_normalization_60 (BatchNormalization)	(None, 512)	2048
dense_287 (Dense)	(None, 256)	131328

batch_normalization_61 (Batch Normalization)	(None, 256)	1024
dense_288 (Dense)	(None, 128)	32896
batch_normalization_62 (Batch Normalization)	(None, 128)	512
dense_289 (Dense)	(None, 64)	8256
batch_normalization_63 (Batch Normalization)	(None, 64)	256
dense_290 (Dense)	(None, 32)	2080
batch_normalization_64 (Batch Normalization)	(None, 32)	128
dense_291 (Dense)	(None, 10)	330
<hr/>		
Total params: 580,778		
Trainable params: 578,794		
Non-trainable params: 1,984		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 17s 286us/step - loss: 0.5219 - acc: 0.8586 - val_loss: 0.3287 - val_acc: 0.9134
Epoch 2/20
60000/60000 [=====] - 9s 151us/step - loss: 0.3122 - acc: 0.9158 - val_loss: 0.2630 - val_acc: 0.9289
Epoch 3/20
60000/60000 [=====] - 9s 149us/step - loss: 0.2545 - acc: 0.9308 - val_loss: 0.2264 - val_acc: 0.9380
Epoch 4/20
60000/60000 [=====] - 9s 149us/step - loss: 0.2162 - acc: 0.9406 - val_loss: 0.1953 - val_acc: 0.9440
```

```
- -  
Epoch 5/20  
60000/60000 [=====] - 9s 147us/step - loss: 0.  
1873 - acc: 0.9484 - val_loss: 0.1746 - val_acc: 0.9514  
Epoch 6/20  
60000/60000 [=====] - 9s 149us/step - loss: 0.  
1661 - acc: 0.9539 - val_loss: 0.1582 - val_acc: 0.9551  
Epoch 7/20  
60000/60000 [=====] - 9s 149us/step - loss: 0.  
1472 - acc: 0.9589 - val_loss: 0.1450 - val_acc: 0.9590  
Epoch 8/20  
60000/60000 [=====] - 9s 146us/step - loss: 0.  
1310 - acc: 0.9640 - val_loss: 0.1334 - val_acc: 0.9616  
Epoch 9/20  
60000/60000 [=====] - 9s 152us/step - loss: 0.  
1190 - acc: 0.9675 - val_loss: 0.1270 - val_acc: 0.9643  
Epoch 10/20  
60000/60000 [=====] - 9s 155us/step - loss: 0.  
1080 - acc: 0.9701 - val_loss: 0.1200 - val_acc: 0.9648  
Epoch 11/20  
60000/60000 [=====] - 9s 151us/step - loss: 0.  
0969 - acc: 0.9736 - val_loss: 0.1120 - val_acc: 0.9662  
Epoch 12/20  
60000/60000 [=====] - 9s 149us/step - loss: 0.  
0904 - acc: 0.9752 - val_loss: 0.1088 - val_acc: 0.9700  
Epoch 13/20  
60000/60000 [=====] - 9s 150us/step - loss: 0.  
0823 - acc: 0.9775 - val_loss: 0.1059 - val_acc: 0.9690  
Epoch 14/20  
60000/60000 [=====] - 9s 150us/step - loss: 0.  
0759 - acc: 0.9799 - val_loss: 0.1009 - val_acc: 0.9701  
Epoch 15/20  
60000/60000 [=====] - 9s 148us/step - loss: 0.  
0697 - acc: 0.9818 - val_loss: 0.0969 - val_acc: 0.9715  
Epoch 16/20  
60000/60000 [=====] - 9s 146us/step - loss: 0.  
0638 - acc: 0.9832 - val_loss: 0.0961 - val_acc: 0.9698  
Epoch 17/20  
60000/60000 [=====] - 9s 149us/step - loss: 0.  
0592 - acc: 0.9842 - val_loss: 0.0921 - val_acc: 0.9724
```

```
Epoch 18/20
60000/60000 [=====] - 9s 148us/step - loss: 0.
0543 - acc: 0.9861 - val_loss: 0.0910 - val_acc: 0.9718
Epoch 19/20
60000/60000 [=====] - 9s 147us/step - loss: 0.
0498 - acc: 0.9874 - val_loss: 0.0929 - val_acc: 0.9717
Epoch 20/20
60000/60000 [=====] - 9s 146us/step - loss: 0.
0468 - acc: 0.9883 - val_loss: 0.0877 - val_acc: 0.9724
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

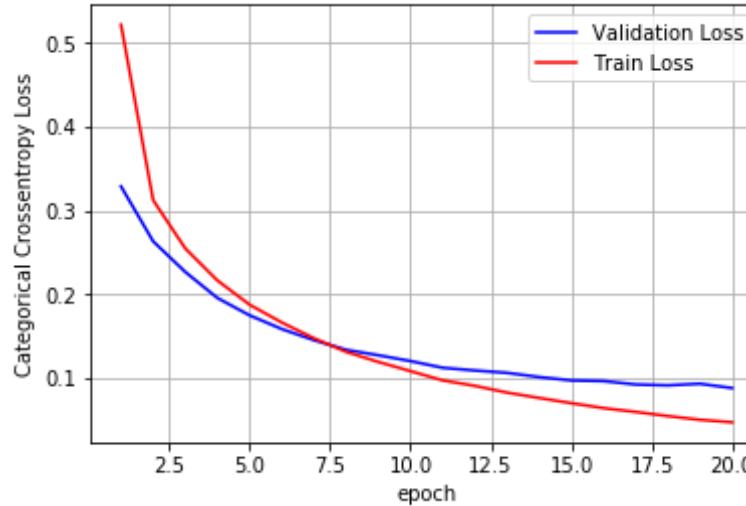
```
Test score: 0.08773159670177848
Test accuracy: 0.9724
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

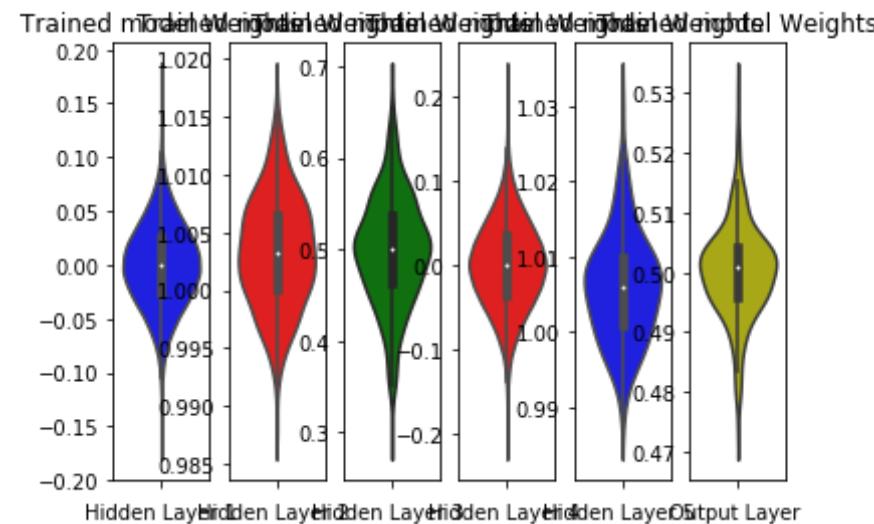
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_83"

Layer (type)	Output Shape	Param #
dense_296 (Dense)	(None, 512)	401920
batch_normalization_68 (BatchNormalization)	(None, 512)	2048
dense_297 (Dense)	(None, 256)	131328
batch_normalization_69 (BatchNormalization)	(None, 256)	1024

dense_298 (Dense)	(None, 128)	32896
batch_normalization_70 (Batch Normalization)	(None, 128)	512
dense_299 (Dense)	(None, 64)	8256
batch_normalization_71 (Batch Normalization)	(None, 64)	256
dense_300 (Dense)	(None, 32)	2080
batch_normalization_72 (Batch Normalization)	(None, 32)	128
dense_301 (Dense)	(None, 10)	330
=====		
Total params: 580,778		
Trainable params: 578,794		
Non-trainable params: 1,984		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 20s 327us/step - loss: 0.2701 - acc: 0.9236 - val_loss: 0.1536 - val_acc: 0.9563
Epoch 2/20
60000/60000 [=====] - 10s 174us/step - loss: 0.1160 - acc: 0.9662 - val_loss: 0.1135 - val_acc: 0.9655
Epoch 3/20
60000/60000 [=====] - 10s 173us/step - loss: 0.0746 - acc: 0.9773 - val_loss: 0.1076 - val_acc: 0.9680
Epoch 4/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0576 - acc: 0.9823 - val_loss: 0.0956 - val_acc: 0.9721
```

```
Epoch 5/20
60000/60000 [=====] - 10s 174us/step - loss: 0.0497 - acc: 0.9842 - val_loss: 0.0958 - val_acc: 0.9715
Epoch 6/20
60000/60000 [=====] - 10s 173us/step - loss: 0.0367 - acc: 0.9885 - val_loss: 0.0941 - val_acc: 0.9739
Epoch 7/20
60000/60000 [=====] - 10s 174us/step - loss: 0.0333 - acc: 0.9891 - val_loss: 0.1094 - val_acc: 0.9679
Epoch 8/20
60000/60000 [=====] - 10s 174us/step - loss: 0.0310 - acc: 0.9896 - val_loss: 0.1039 - val_acc: 0.9719
Epoch 9/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0264 - acc: 0.9914 - val_loss: 0.1113 - val_acc: 0.9709
Epoch 10/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0236 - acc: 0.9927 - val_loss: 0.0944 - val_acc: 0.9741
Epoch 11/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0244 - acc: 0.9923 - val_loss: 0.0977 - val_acc: 0.9744
Epoch 12/20
60000/60000 [=====] - 10s 174us/step - loss: 0.0200 - acc: 0.9934 - val_loss: 0.0869 - val_acc: 0.9762
Epoch 13/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0168 - acc: 0.9946 - val_loss: 0.0877 - val_acc: 0.9771
Epoch 14/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0187 - acc: 0.9938 - val_loss: 0.1039 - val_acc: 0.9754
Epoch 15/20
60000/60000 [=====] - 11s 177us/step - loss: 0.0193 - acc: 0.9935 - val_loss: 0.0953 - val_acc: 0.9756
Epoch 16/20
60000/60000 [=====] - 11s 176us/step - loss: 0.0158 - acc: 0.9949 - val_loss: 0.0998 - val_acc: 0.9765
Epoch 17/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0173 - acc: 0.9943 - val_loss: 0.0930 - val_acc: 0.9778
```

```
Epoch 18/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0126 - acc: 0.9957 - val_loss: 0.1017 - val_acc: 0.9774
Epoch 19/20
60000/60000 [=====] - 11s 181us/step - loss: 0.0141 - acc: 0.9953 - val_loss: 0.0999 - val_acc: 0.9768
Epoch 20/20
60000/60000 [=====] - 11s 179us/step - loss: 0.0143 - acc: 0.9954 - val_loss: 0.0921 - val_acc: 0.9768
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

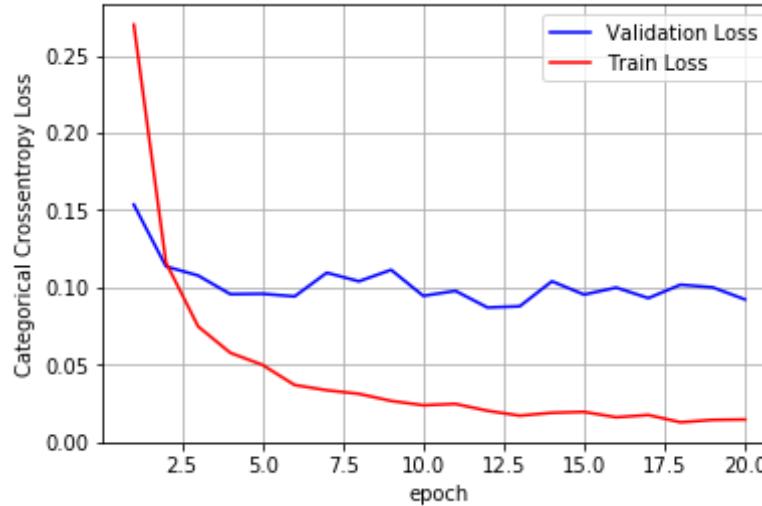
```
Test score: 0.09214983825360687
Test accuracy: 0.9768
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

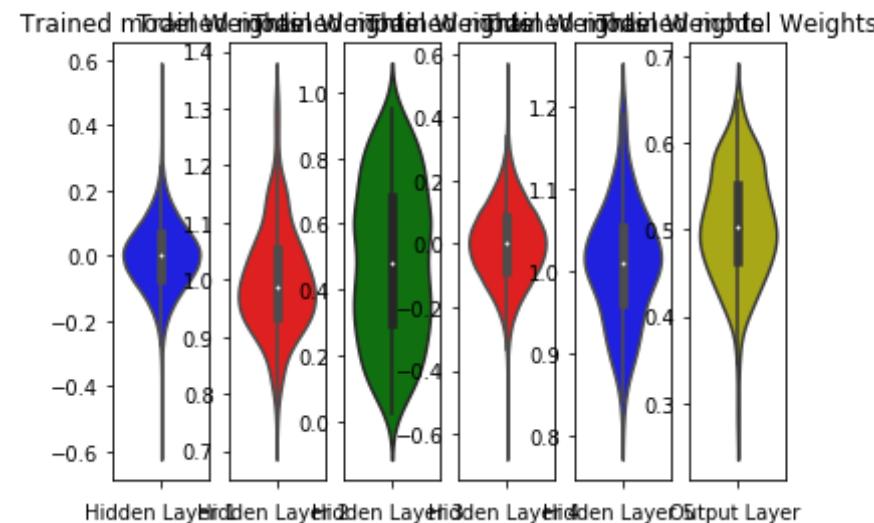
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



### Model 3: sigmoid activation with Batch Normalization + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_84"

Layer (type)	Output Shape	Param #
dense_302 (Dense)	(None, 512)	401920
batch_normalization_73 (BatchNormalization)	(None, 512)	2048
dense_303 (Dense)	(None, 256)	131328

batch_normalization_74 (Batch Normalization)	(None, 256)	1024
dense_304 (Dense)	(None, 128)	32896
batch_normalization_75 (Batch Normalization)	(None, 128)	512
dense_305 (Dense)	(None, 64)	8256
batch_normalization_76 (Batch Normalization)	(None, 64)	256
dense_306 (Dense)	(None, 32)	2080
batch_normalization_77 (Batch Normalization)	(None, 32)	128
dense_307 (Dense)	(None, 10)	330
<hr/>		
Total params: 580,778		
Trainable params: 578,794		
Non-trainable params: 1,984		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 18s 299us/step - loss: 0.6867 - acc: 0.8078 - val_loss: 0.3902 - val_acc: 0.8962
Epoch 2/20
60000/60000 [=====] - 9s 150us/step - loss: 0.3560 - acc: 0.9028 - val_loss: 0.2973 - val_acc: 0.9182
Epoch 3/20
60000/60000 [=====] - 9s 150us/step - loss: 0.2827 - acc: 0.9214 - val_loss: 0.2543 - val_acc: 0.9292
Epoch 4/20
60000/60000 [=====] - 9s 150us/step - loss: 0.2421 - acc: 0.9328 - val_loss: 0.2247 - val_acc: 0.9376
```

```
- -  
Epoch 5/20  
60000/60000 [=====] - 9s 150us/step - loss: 0.  
2133 - acc: 0.9401 - val_loss: 0.2044 - val_acc: 0.9415  
Epoch 6/20  
60000/60000 [=====] - 9s 151us/step - loss: 0.  
1903 - acc: 0.9469 - val_loss: 0.1916 - val_acc: 0.9442  
Epoch 7/20  
60000/60000 [=====] - 9s 148us/step - loss: 0.  
1724 - acc: 0.9524 - val_loss: 0.1779 - val_acc: 0.9486  
Epoch 8/20  
60000/60000 [=====] - 9s 147us/step - loss: 0.  
1571 - acc: 0.9563 - val_loss: 0.1684 - val_acc: 0.9519  
Epoch 9/20  
60000/60000 [=====] - 9s 147us/step - loss: 0.  
1453 - acc: 0.9600 - val_loss: 0.1603 - val_acc: 0.9527  
Epoch 10/20  
60000/60000 [=====] - 9s 150us/step - loss: 0.  
1340 - acc: 0.9628 - val_loss: 0.1536 - val_acc: 0.9545  
Epoch 11/20  
60000/60000 [=====] - 9s 149us/step - loss: 0.  
1243 - acc: 0.9658 - val_loss: 0.1479 - val_acc: 0.9566  
Epoch 12/20  
60000/60000 [=====] - 9s 146us/step - loss: 0.  
1160 - acc: 0.9679 - val_loss: 0.1441 - val_acc: 0.9580  
Epoch 13/20  
60000/60000 [=====] - 9s 147us/step - loss: 0.  
1069 - acc: 0.9708 - val_loss: 0.1389 - val_acc: 0.9593  
Epoch 14/20  
60000/60000 [=====] - 9s 146us/step - loss: 0.  
1002 - acc: 0.9724 - val_loss: 0.1361 - val_acc: 0.9597  
Epoch 15/20  
60000/60000 [=====] - 9s 147us/step - loss: 0.  
0932 - acc: 0.9742 - val_loss: 0.1319 - val_acc: 0.9614  
Epoch 16/20  
60000/60000 [=====] - 9s 145us/step - loss: 0.  
0869 - acc: 0.9759 - val_loss: 0.1293 - val_acc: 0.9613  
Epoch 17/20  
60000/60000 [=====] - 9s 145us/step - loss: 0.  
0824 - acc: 0.9780 - val_loss: 0.1276 - val_acc: 0.9632
```

```
Epoch 18/20
60000/60000 [=====] - 9s 145us/step - loss: 0.
0772 - acc: 0.9794 - val_loss: 0.1257 - val_acc: 0.9627
Epoch 19/20
60000/60000 [=====] - 9s 144us/step - loss: 0.
0708 - acc: 0.9816 - val_loss: 0.1231 - val_acc: 0.9640
Epoch 20/20
60000/60000 [=====] - 9s 147us/step - loss: 0.
0676 - acc: 0.9822 - val_loss: 0.1214 - val_acc: 0.9636
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

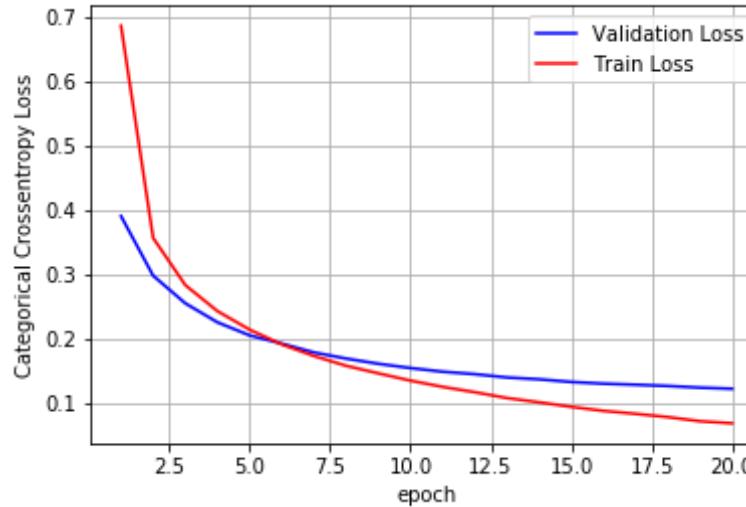
```
Test score: 0.12137281653508544
Test accuracy: 0.9636
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

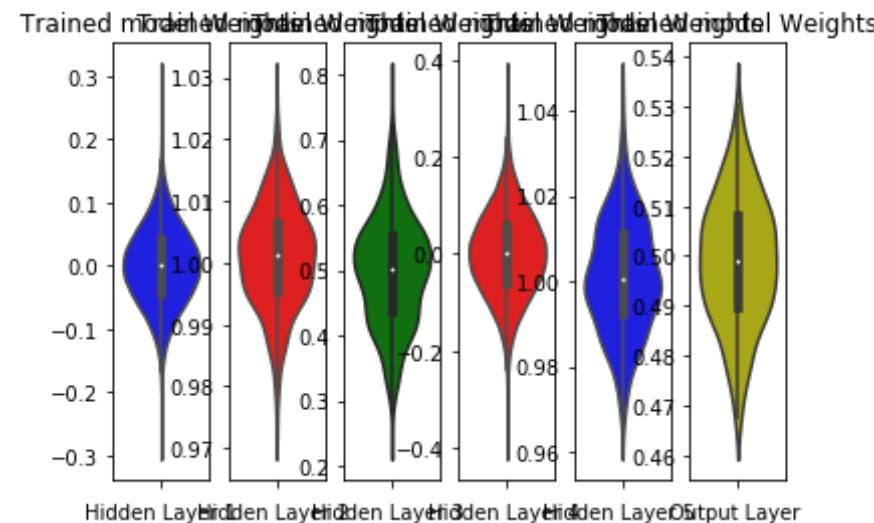
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 4: relu activation with Dropout + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_85"

Layer (type)	Output Shape	Param #
<hr/>		
dense_308 (Dense)	(None, 512)	401920
dropout_51 (Dropout)	(None, 512)	0

dense_309 (Dense)	(None, 256)	131328
dropout_52 (Dropout)	(None, 256)	0
dense_310 (Dense)	(None, 128)	32896
dropout_53 (Dropout)	(None, 128)	0
dense_311 (Dense)	(None, 64)	8256
dropout_54 (Dropout)	(None, 64)	0
dense_312 (Dense)	(None, 32)	2080
dropout_55 (Dropout)	(None, 32)	0
dense_313 (Dense)	(None, 10)	330
=====		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 19s 310us/step - loss:
1.3868 - acc: 0.5024 - val_loss: 0.4142 - val_acc: 0.8914
Epoch 2/20
60000/60000 [=====] - 10s 165us/step - loss:
0.6075 - acc: 0.8186 - val_loss: 0.2524 - val_acc: 0.9460
Epoch 3/20
60000/60000 [=====] - 10s 165us/step - loss:
0.4409 - acc: 0.8844 - val loss: 0.1889 - val acc: 0.9579
```

```
- -  
Epoch 4/20  
60000/60000 [=====] - 10s 166us/step - loss:  
0.3746 - acc: 0.9079 - val_loss: 0.1703 - val_acc: 0.9604  
Epoch 5/20  
60000/60000 [=====] - 10s 163us/step - loss:  
0.3320 - acc: 0.9201 - val_loss: 0.1655 - val_acc: 0.9616  
Epoch 6/20  
60000/60000 [=====] - 10s 165us/step - loss:  
0.2988 - acc: 0.9278 - val_loss: 0.1445 - val_acc: 0.9682  
Epoch 7/20  
60000/60000 [=====] - 10s 163us/step - loss:  
0.2785 - acc: 0.9329 - val_loss: 0.1413 - val_acc: 0.9682  
Epoch 8/20  
60000/60000 [=====] - 10s 167us/step - loss:  
0.2538 - acc: 0.9394 - val_loss: 0.1369 - val_acc: 0.9717  
Epoch 9/20  
60000/60000 [=====] - 10s 167us/step - loss:  
0.2447 - acc: 0.9410 - val_loss: 0.1225 - val_acc: 0.9741  
Epoch 10/20  
60000/60000 [=====] - 10s 165us/step - loss:  
0.2254 - acc: 0.9462 - val_loss: 0.1377 - val_acc: 0.9711  
Epoch 11/20  
60000/60000 [=====] - 10s 163us/step - loss:  
0.2174 - acc: 0.9485 - val_loss: 0.1332 - val_acc: 0.9724  
Epoch 12/20  
60000/60000 [=====] - 10s 161us/step - loss:  
0.2133 - acc: 0.9495 - val_loss: 0.1397 - val_acc: 0.9724  
Epoch 13/20  
60000/60000 [=====] - 10s 162us/step - loss:  
0.2100 - acc: 0.9504 - val_loss: 0.1161 - val_acc: 0.9755  
Epoch 14/20  
60000/60000 [=====] - 10s 163us/step - loss:  
0.2040 - acc: 0.9511 - val_loss: 0.1168 - val_acc: 0.9742  
Epoch 15/20  
60000/60000 [=====] - 10s 162us/step - loss:  
0.1957 - acc: 0.9510 - val_loss: 0.1081 - val_acc: 0.9773  
Epoch 16/20  
60000/60000 [=====] - 10s 165us/step - loss:  
0.1906 - acc: 0.9543 - val_loss: 0.1095 - val_acc: 0.9773
```

```
Epoch 17/20
60000/60000 [=====] - 10s 164us/step - loss:
0.1785 - acc: 0.9573 - val_loss: 0.1071 - val_acc: 0.9781
Epoch 18/20
60000/60000 [=====] - 10s 163us/step - loss:
0.1828 - acc: 0.9559 - val_loss: 0.1231 - val_acc: 0.9763
Epoch 19/20
60000/60000 [=====] - 10s 162us/step - loss:
0.1754 - acc: 0.9570 - val_loss: 0.1167 - val_acc: 0.9766
Epoch 20/20
60000/60000 [=====] - 10s 163us/step - loss:
0.1729 - acc: 0.9588 - val_loss: 0.1163 - val_acc: 0.9773
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

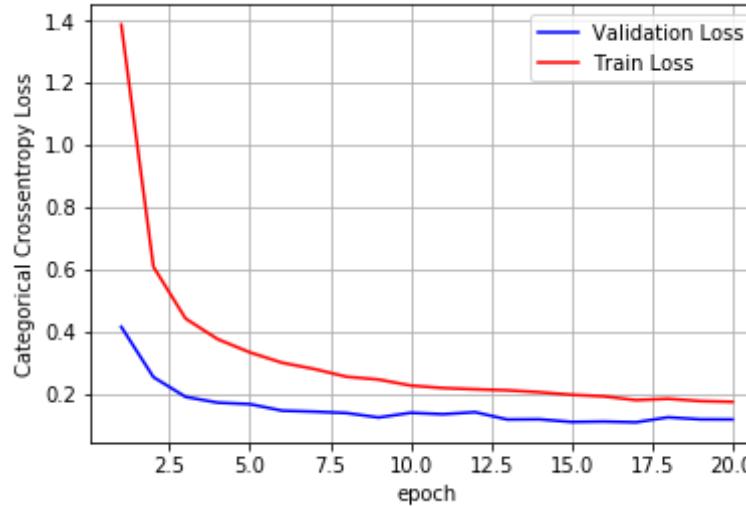
```
Test score: 0.11627653976904913
Test accuracy: 0.9773
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

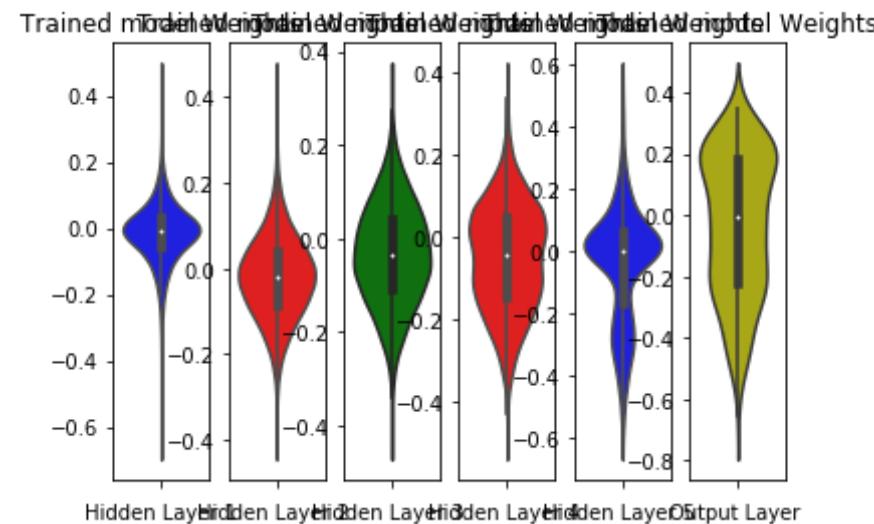
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



## Model 4: relu activation with Dropout + GradientDescentOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout  
  
model_1 = Sequential()  
  
model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))  
model_1.add(Dropout(0.5))  
  
model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))  
model_1.add(Dropout(0.5))  
  
model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))  
model_1.add(Dropout(0.5))  
  
model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))  
model_1.add(Dropout(0.5))  
  
model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))  
model_1.add(Dropout(0.5))  
  
model_1.add(Dense(output_dim, activation='softmax'))  
  
model_1.summary()
```

Model: "sequential\_86"

Layer (type)	Output Shape	Param #
<hr/>		
dense_314 (Dense)	(None, 512)	401920
dropout_56 (Dropout)	(None, 512)	0

dense_315 (Dense)	(None, 256)	131328
dropout_57 (Dropout)	(None, 256)	0
dense_316 (Dense)	(None, 128)	32896
dropout_58 (Dropout)	(None, 128)	0
dense_317 (Dense)	(None, 64)	8256
dropout_59 (Dropout)	(None, 64)	0
dense_318 (Dense)	(None, 32)	2080
dropout_60 (Dropout)	(None, 32)	0
dense_319 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 16s 273us/step - loss: 2.3355 - acc: 0.1121 - val\_loss: 2.2941 - val\_acc: 0.1811  
Epoch 2/20  
60000/60000 [=====] - 8s 131us/step - loss: 2.2741 - acc: 0.1477 - val\_loss: 2.2116 - val\_acc: 0.3238  
Epoch 3/20  
60000/60000 [=====] - 8s 130us/step - loss: 2.1666 - acc: 0.2108 - val\_loss: 1.9421 - val\_acc: 0.4012  
Epoch 4/20

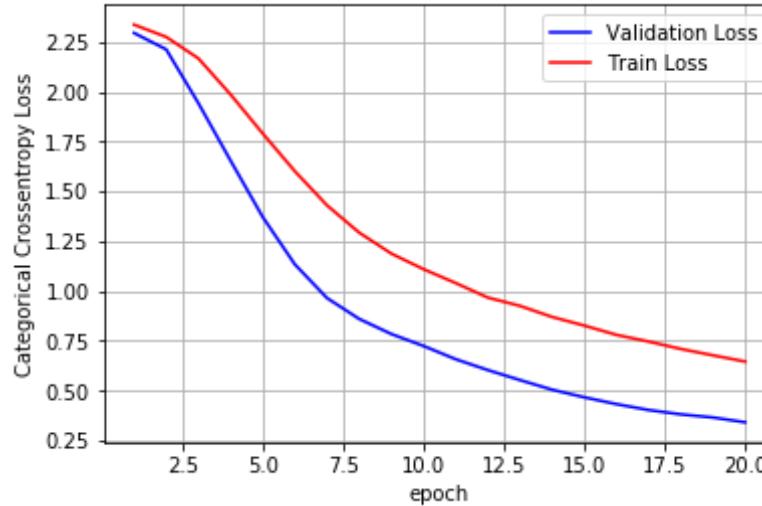
```
60000/60000 [=====] - 8s 136us/step - loss: 1.  
9858 - acc: 0.2903 - val_loss: 1.6541 - val_acc: 0.4895  
Epoch 5/20  
60000/60000 [=====] - 8s 137us/step - loss: 1.  
7900 - acc: 0.3592 - val_loss: 1.3712 - val_acc: 0.6041  
Epoch 6/20  
60000/60000 [=====] - 8s 140us/step - loss: 1.  
6006 - acc: 0.4216 - val_loss: 1.1334 - val_acc: 0.6617  
Epoch 7/20  
60000/60000 [=====] - 8s 140us/step - loss: 1.  
4296 - acc: 0.4829 - val_loss: 0.9647 - val_acc: 0.6792  
Epoch 8/20  
60000/60000 [=====] - 8s 137us/step - loss: 1.  
2928 - acc: 0.5341 - val_loss: 0.8601 - val_acc: 0.7181  
Epoch 9/20  
60000/60000 [=====] - 8s 134us/step - loss: 1.  
1877 - acc: 0.5769 - val_loss: 0.7844 - val_acc: 0.7552  
Epoch 10/20  
60000/60000 [=====] - 8s 132us/step - loss: 1.  
1095 - acc: 0.6113 - val_loss: 0.7245 - val_acc: 0.7836  
Epoch 11/20  
60000/60000 [=====] - 8s 135us/step - loss: 1.  
0400 - acc: 0.6437 - val_loss: 0.6581 - val_acc: 0.8330  
Epoch 12/20  
60000/60000 [=====] - 8s 138us/step - loss: 0.  
9670 - acc: 0.6780 - val_loss: 0.6037 - val_acc: 0.8492  
Epoch 13/20  
60000/60000 [=====] - 8s 134us/step - loss: 0.  
9254 - acc: 0.6983 - val_loss: 0.5535 - val_acc: 0.8681  
Epoch 14/20  
60000/60000 [=====] - 8s 132us/step - loss: 0.  
8698 - acc: 0.7241 - val_loss: 0.5051 - val_acc: 0.8900  
Epoch 15/20  
60000/60000 [=====] - 8s 135us/step - loss: 0.  
8267 - acc: 0.7411 - val_loss: 0.4671 - val_acc: 0.9015  
Epoch 16/20  
60000/60000 [=====] - 8s 139us/step - loss: 0.  
7800 - acc: 0.7584 - val_loss: 0.4328 - val_acc: 0.9107  
Epoch 17/20
```

```
60000/60000 [=====] - 8s 139us/step - loss: 0.  
7466 - acc: 0.7720 - val_loss: 0.4032 - val_acc: 0.9174  
Epoch 18/20  
60000/60000 [=====] - 8s 136us/step - loss: 0.  
7096 - acc: 0.7852 - val_loss: 0.3815 - val_acc: 0.9222  
Epoch 19/20  
60000/60000 [=====] - 8s 137us/step - loss: 0.  
6775 - acc: 0.7942 - val_loss: 0.3658 - val_acc: 0.9241  
Epoch 20/20  
60000/60000 [=====] - 8s 138us/step - loss: 0.  
6463 - acc: 0.8095 - val_loss: 0.3417 - val_acc: 0.9287
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.3417197764635086  
Test accuracy: 0.9287
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
```

```

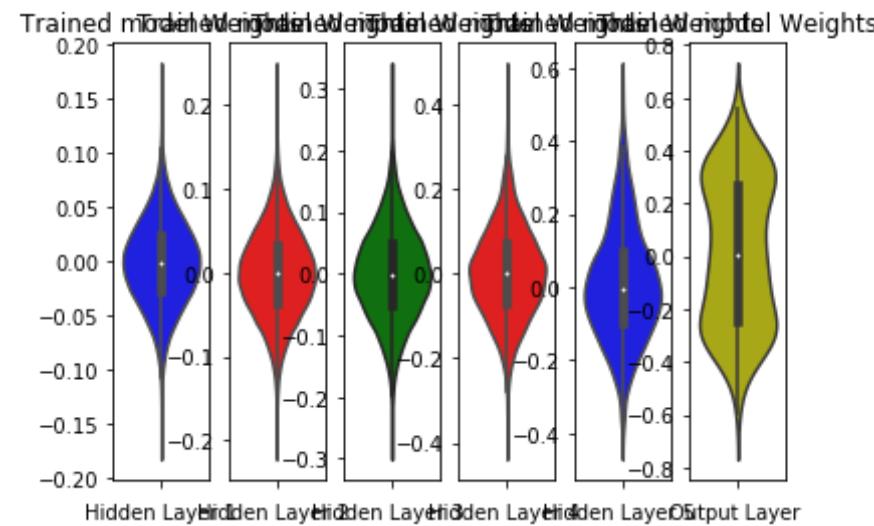
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()

```



#### Model 4: relu activation with Dropout + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(Dropout(0.5))
model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_87"

Layer (type)	Output Shape	Param #
<hr/>		
dense_320 (Dense)	(None, 512)	401920
dropout_61 (Dropout)	(None, 512)	0
dense_321 (Dense)	(None, 256)	131328
dropout_62 (Dropout)	(None, 256)	0
<hr/>		

dense_322 (Dense)	(None, 128)	32896
dropout_63 (Dropout)	(None, 128)	0
dense_323 (Dense)	(None, 64)	8256
dropout_64 (Dropout)	(None, 64)	0
dense_324 (Dense)	(None, 32)	2080
dropout_65 (Dropout)	(None, 32)	0
dense_325 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/20  
60000/60000 [=====] - 19s 315us/step - loss:  
2.5172 - acc: 0.1457 - val\_loss: 1.9460 - val\_acc: 0.4235  
Epoch 2/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.7438 - acc: 0.7852 - val\_loss: 0.3831 - val\_acc: 0.9258  
Epoch 5/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.6178 - acc: 0.8307 - val\_loss: 0.3184 - val\_acc: 0.9350  
Epoch 6/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.5332 - acc: 0.8617 - val\_loss: 0.2759 - val\_acc: 0.9450  
Epoch 7/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.4580 - acc: 0.8920 - val\_loss: 0.2359 - val\_acc: 0.9550

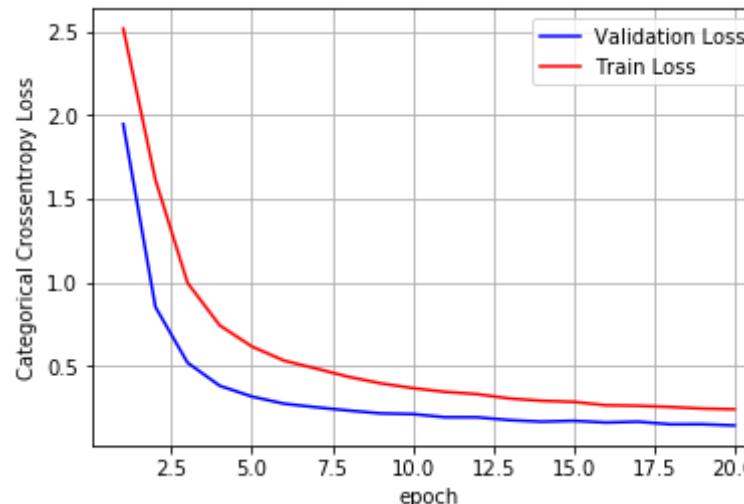
```
60000/60000 [=====] - 10s 160us/step - loss:  
0.4859 - acc: 0.8783 - val_loss: 0.2544 - val_acc: 0.9489  
Epoch 8/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.4368 - acc: 0.8917 - val_loss: 0.2352 - val_acc: 0.9515  
Epoch 9/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.3979 - acc: 0.9033 - val_loss: 0.2181 - val_acc: 0.9559  
Epoch 10/20  
60000/60000 [=====] - 10s 162us/step - loss:  
0.3690 - acc: 0.9118 - val_loss: 0.2140 - val_acc: 0.9577  
Epoch 11/20  
60000/60000 [=====] - 9s 158us/step - loss: 0.  
3467 - acc: 0.9174 - val_loss: 0.1950 - val_acc: 0.9610  
Epoch 12/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.3332 - acc: 0.9216 - val_loss: 0.1948 - val_acc: 0.9619  
Epoch 13/20  
60000/60000 [=====] - 10s 161us/step - loss:  
0.3071 - acc: 0.9259 - val_loss: 0.1779 - val_acc: 0.9625  
Epoch 14/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.2933 - acc: 0.9283 - val_loss: 0.1685 - val_acc: 0.9652  
Epoch 15/20  
60000/60000 [=====] - 9s 158us/step - loss: 0.  
2869 - acc: 0.9313 - val_loss: 0.1736 - val_acc: 0.9656  
Epoch 16/20  
60000/60000 [=====] - 9s 157us/step - loss: 0.  
2668 - acc: 0.9366 - val_loss: 0.1634 - val_acc: 0.9675  
Epoch 17/20  
60000/60000 [=====] - 10s 161us/step - loss:  
0.2635 - acc: 0.9357 - val_loss: 0.1684 - val_acc: 0.9678  
Epoch 18/20  
60000/60000 [=====] - 10s 161us/step - loss:  
0.2562 - acc: 0.9392 - val_loss: 0.1528 - val_acc: 0.9697  
Epoch 19/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.2469 - acc: 0.9422 - val_loss: 0.1532 - val_acc: 0.9684  
Epoch 20/20
```

```
600000/60000 [=====] - 10s 160us/step - loss:  
0.2424 - acc: 0.9428 - val_loss: 0.1458 - val_acc: 0.9713
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.14583226314671338  
Test accuracy: 0.9713
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))  
  
vy = history.history['val_loss']  
ty = history.history['loss']  
  
plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()
```

```
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

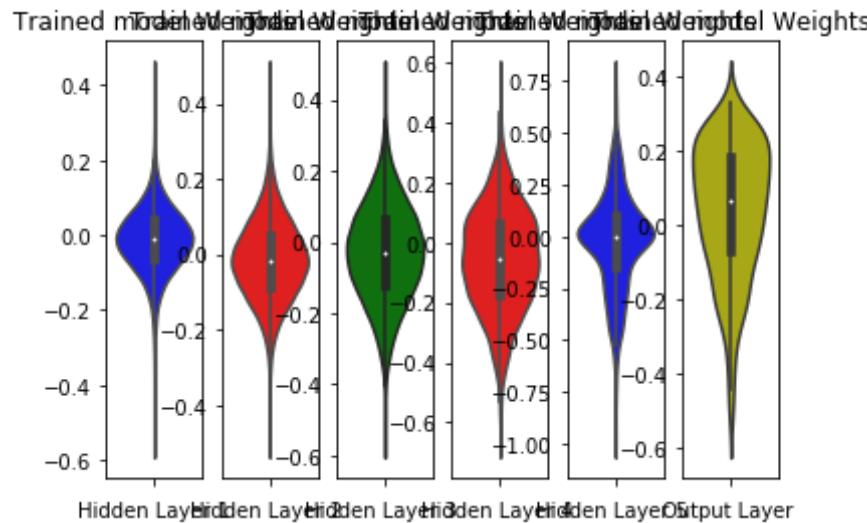
plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 4: relu activation with Dropout + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))
```

```
model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='relu'))

model_1.summary()
```

Model: "sequential\_88"

Layer (type)	Output Shape	Param #
dense_326 (Dense)	(None, 512)	401920
dropout_66 (Dropout)	(None, 512)	0
dense_327 (Dense)	(None, 256)	131328
dropout_67 (Dropout)	(None, 256)	0
dense_328 (Dense)	(None, 128)	32896
dropout_68 (Dropout)	(None, 128)	0
dense_329 (Dense)	(None, 64)	8256
dropout_69 (Dropout)	(None, 64)	0
dense_330 (Dense)	(None, 32)	2080
dropout_70 (Dropout)	(None, 32)	0
dense_331 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		

Non-trainable params: 0

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nbr_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 17s 279us/step - loss: 5.3447 - acc: 0.0981 - val_loss: 4.6554 - val_acc: 0.0982
Epoch 2/20
60000/60000 [=====] - 8s 136us/step - loss: 3.8970 - acc: 0.0976 - val_loss: 3.2631 - val_acc: 0.0982
Epoch 3/20
60000/60000 [=====] - 8s 133us/step - loss: 3.1754 - acc: 0.0978 - val_loss: 3.1298 - val_acc: 0.0982
Epoch 4/20
60000/60000 [=====] - 8s 130us/step - loss: 3.0757 - acc: 0.0980 - val_loss: 3.0471 - val_acc: 0.0982
Epoch 5/20
60000/60000 [=====] - 8s 130us/step - loss: 2.9984 - acc: 0.0977 - val_loss: 2.9858 - val_acc: 0.0982
Epoch 6/20
60000/60000 [=====] - 8s 131us/step - loss: 2.9473 - acc: 0.0974 - val_loss: 2.9390 - val_acc: 0.0982
Epoch 7/20
60000/60000 [=====] - 8s 130us/step - loss: 2.9067 - acc: 0.0979 - val_loss: 2.9007 - val_acc: 0.0982
Epoch 8/20
60000/60000 [=====] - 8s 131us/step - loss: 2.8713 - acc: 0.0975 - val_loss: 2.8684 - val_acc: 0.0982
Epoch 9/20
60000/60000 [=====] - 8s 130us/step - loss: 2.8418 - acc: 0.0977 - val_loss: 2.8404 - val_acc: 0.0982
Epoch 10/20
60000/60000 [=====] - 8s 131us/step - loss: 2.8111 - acc: 0.0978 - val_loss: 2.8352 - val_acc: 0.0982
```

```
8171 - acc: 0.0970 - val_loss: 2.8159 - val_acc: 0.0982
Epoch 11/20
60000/60000 [=====] - 8s 132us/step - loss: 2.
7918 - acc: 0.0971 - val_loss: 2.7941 - val_acc: 0.0982
Epoch 12/20
60000/60000 [=====] - 8s 131us/step - loss: 2.
7737 - acc: 0.0973 - val_loss: 2.7744 - val_acc: 0.0982
Epoch 13/20
60000/60000 [=====] - 8s 134us/step - loss: 2.
7519 - acc: 0.0979 - val_loss: 2.7566 - val_acc: 0.0982
Epoch 14/20
60000/60000 [=====] - 8s 137us/step - loss: 2.
7352 - acc: 0.0974 - val_loss: 2.7402 - val_acc: 0.0982
Epoch 15/20
60000/60000 [=====] - 8s 132us/step - loss: 2.
7209 - acc: 0.0973 - val_loss: 2.7250 - val_acc: 0.0982
Epoch 16/20
60000/60000 [=====] - 8s 134us/step - loss: 2.
7043 - acc: 0.0974 - val_loss: 2.7109 - val_acc: 0.0982
Epoch 17/20
60000/60000 [=====] - 8s 134us/step - loss: 2.
6911 - acc: 0.0975 - val_loss: 2.6979 - val_acc: 0.0982
Epoch 18/20
60000/60000 [=====] - 8s 137us/step - loss: 2.
6805 - acc: 0.0979 - val_loss: 2.6856 - val_acc: 0.0982
Epoch 19/20
60000/60000 [=====] - 8s 134us/step - loss: 2.
6666 - acc: 0.0981 - val_loss: 2.6742 - val_acc: 0.0982
Epoch 20/20
60000/60000 [=====] - 8s 136us/step - loss: 2.
6550 - acc: 0.0979 - val_loss: 2.6635 - val_acc: 0.0982
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

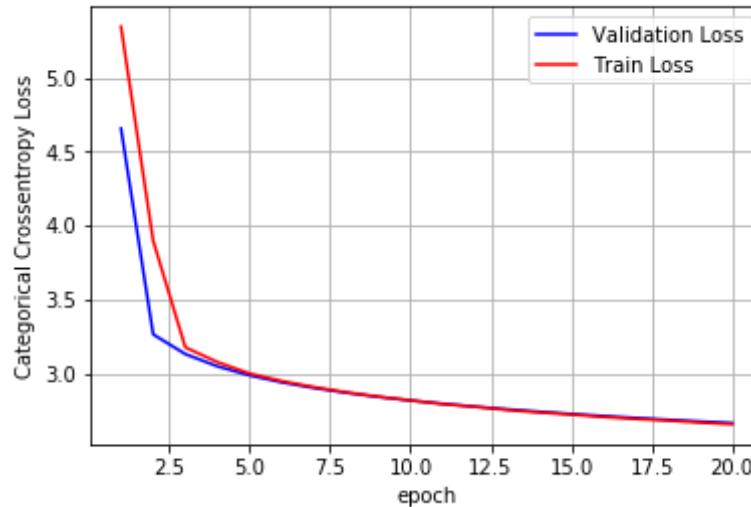
```
Test score: 2.6634529762268064
Test accuracy: 0.0982
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
```

```
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

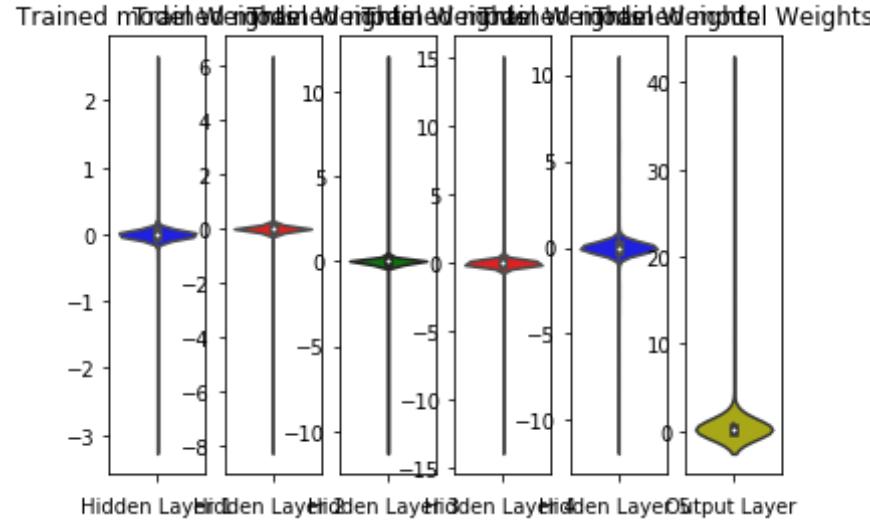
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 5: relu activation with BatchNormalization + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.09, seed=None)))
model_1.add(BatchNormalization())
```

```

l(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(
l(mean=0.0, stddev=0.144, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

```

Model: "sequential\_89"

Layer (type)	Output Shape	Param #
dense_332 (Dense)	(None, 512)	401920
batch_normalization_78 (Batch Normalization)	(None, 512)	2048
dense_333 (Dense)	(None, 256)	131328
batch_normalization_79 (Batch Normalization)	(None, 256)	1024
dense_334 (Dense)	(None, 128)	32896
batch_normalization_80 (Batch Normalization)	(None, 128)	512
dense_335 (Dense)	(None, 64)	8256
batch_normalization_81 (Batch Normalization)	(None, 64)	256
dense_336 (Dense)	(None, 32)	2080
batch_normalization_82 (Batch Normalization)	(None, 32)	128
dense_337 (Dense)	(None, 10)	330

Total params: 580,778  
Trainable params: 578,794

Non-trainable params: 1,984

---

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n_b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 20s 334us/step - loss: 0.2513 - acc: 0.9309 - val_loss: 0.1254 - val_acc: 0.9586
Epoch 2/20
60000/60000 [=====] - 11s 182us/step - loss: 0.0933 - acc: 0.9713 - val_loss: 0.1002 - val_acc: 0.9703
Epoch 3/20
60000/60000 [=====] - 11s 183us/step - loss: 0.0672 - acc: 0.9783 - val_loss: 0.0854 - val_acc: 0.9746
Epoch 4/20
60000/60000 [=====] - 11s 183us/step - loss: 0.0514 - acc: 0.9841 - val_loss: 0.0884 - val_acc: 0.9729
Epoch 5/20
60000/60000 [=====] - 11s 181us/step - loss: 0.0436 - acc: 0.9859 - val_loss: 0.0878 - val_acc: 0.9756
Epoch 6/20
60000/60000 [=====] - 11s 183us/step - loss: 0.0349 - acc: 0.9885 - val_loss: 0.0958 - val_acc: 0.9720
Epoch 7/20
60000/60000 [=====] - 11s 181us/step - loss: 0.0339 - acc: 0.9884 - val_loss: 0.0809 - val_acc: 0.9761
Epoch 8/20
60000/60000 [=====] - 11s 181us/step - loss: 0.0297 - acc: 0.9901 - val_loss: 0.0847 - val_acc: 0.9763
Epoch 9/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0288 - acc: 0.9908 - val_loss: 0.0812 - val_acc: 0.9773
Epoch 10/20
60000/60000 [=====] - 11s 178us/step - loss: 0.0248 - acc: 0.9918 - val_loss: 0.0887 - val_acc: 0.9743
```

```
Epoch 11/20  
60000/60000 [=====] - 11s 178us/step - loss:  
0.0201 - acc: 0.9935 - val_loss: 0.0694 - val_acc: 0.9800  
Epoch 12/20  
60000/60000 [=====] - 11s 179us/step - loss:  
0.0213 - acc: 0.9929 - val_loss: 0.0810 - val_acc: 0.9785  
Epoch 13/20  
60000/60000 [=====] - 11s 180us/step - loss:  
0.0181 - acc: 0.9939 - val_loss: 0.0772 - val_acc: 0.9803  
Epoch 14/20  
60000/60000 [=====] - 11s 184us/step - loss:  
0.0190 - acc: 0.9937 - val_loss: 0.0773 - val_acc: 0.9789  
Epoch 15/20  
60000/60000 [=====] - 11s 179us/step - loss:  
0.0156 - acc: 0.9952 - val_loss: 0.1029 - val_acc: 0.9754  
Epoch 16/20  
60000/60000 [=====] - 11s 181us/step - loss:  
0.0156 - acc: 0.9950 - val_loss: 0.0725 - val_acc: 0.9815  
Epoch 17/20  
60000/60000 [=====] - 11s 184us/step - loss:  
0.0144 - acc: 0.9955 - val_loss: 0.0671 - val_acc: 0.9817  
Epoch 18/20  
60000/60000 [=====] - 11s 183us/step - loss:  
0.0127 - acc: 0.9957 - val_loss: 0.0740 - val_acc: 0.9815  
Epoch 19/20  
60000/60000 [=====] - 11s 185us/step - loss:  
0.0147 - acc: 0.9953 - val_loss: 0.0741 - val_acc: 0.9810  
Epoch 20/20  
60000/60000 [=====] - 11s 182us/step - loss:  
0.0134 - acc: 0.9956 - val_loss: 0.0714 - val_acc: 0.9823
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

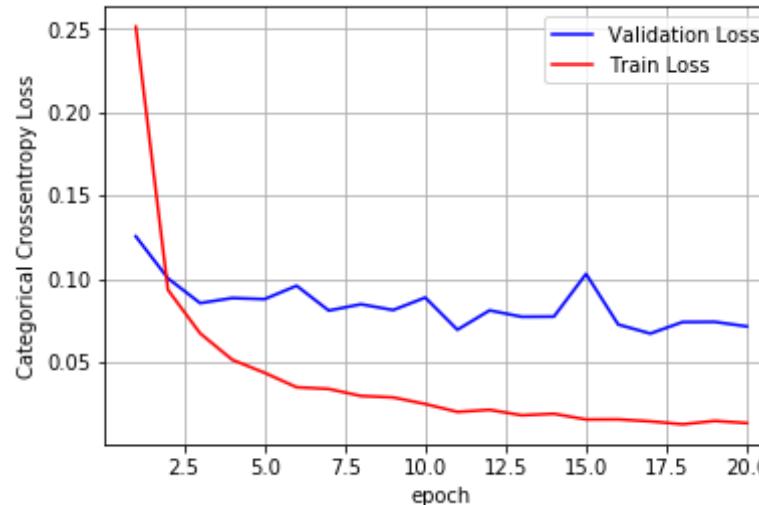
```
Test score: 0.07139411058052793  
Test accuracy: 0.9823
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
```

```
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

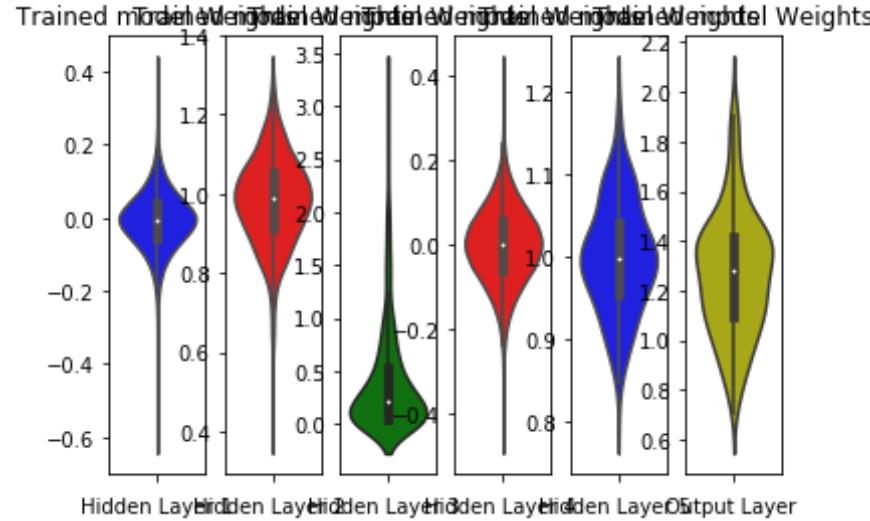
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(BatchNormalization())
```

```
model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_90"

Layer (type)	Output Shape	Param #
dense_338 (Dense)	(None, 512)	401920
batch_normalization_83 (Batch Normalization)	(None, 512)	2048
dense_339 (Dense)	(None, 256)	131328
batch_normalization_84 (Batch Normalization)	(None, 256)	1024
dense_340 (Dense)	(None, 128)	32896
batch_normalization_85 (Batch Normalization)	(None, 128)	512
dense_341 (Dense)	(None, 64)	8256
batch_normalization_86 (Batch Normalization)	(None, 64)	256
dense_342 (Dense)	(None, 32)	2080
batch_normalization_87 (Batch Normalization)	(None, 32)	128
dense_343 (Dense)	(None, 10)	330
<hr/>		
Total params: 580,778		
Trainable params: 578,794		
Non-trainable params: 1,984		

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 19s 323us/step - loss: 0.5982 - acc: 0.8405 - val_loss: 0.2706 - val_acc: 0.9323
Epoch 2/20
60000/60000 [=====] - 9s 155us/step - loss: 0.2386 - acc: 0.9396 - val_loss: 0.1834 - val_acc: 0.9505
Epoch 3/20
60000/60000 [=====] - 9s 152us/step - loss: 0.1648 - acc: 0.9574 - val_loss: 0.1468 - val_acc: 0.9578
Epoch 4/20
60000/60000 [=====] - 9s 155us/step - loss: 0.1257 - acc: 0.9668 - val_loss: 0.1307 - val_acc: 0.9603
Epoch 5/20
60000/60000 [=====] - 9s 156us/step - loss: 0.1002 - acc: 0.9746 - val_loss: 0.1165 - val_acc: 0.9636
Epoch 6/20
60000/60000 [=====] - 9s 153us/step - loss: 0.0821 - acc: 0.9793 - val_loss: 0.1080 - val_acc: 0.9667
Epoch 7/20
60000/60000 [=====] - 9s 153us/step - loss: 0.0682 - acc: 0.9832 - val_loss: 0.1024 - val_acc: 0.9684
Epoch 8/20
60000/60000 [=====] - 9s 151us/step - loss: 0.0570 - acc: 0.9869 - val_loss: 0.0997 - val_acc: 0.9683
Epoch 9/20
60000/60000 [=====] - 9s 153us/step - loss: 0.0474 - acc: 0.9894 - val_loss: 0.0954 - val_acc: 0.9699
Epoch 10/20
60000/60000 [=====] - 9s 153us/step - loss: 0.0414 - acc: 0.9908 - val_loss: 0.0933 - val_acc: 0.9698
Epoch 11/20
60000/60000 [=====] - 9s 154us/step - loss: 0.0364 - acc: 0.9918 - val_loss: 0.0918 - val_acc: 0.9698
```

```
Epoch 12/20
60000/60000 [=====] - 9s 152us/step - loss: 0.
0308 - acc: 0.9934 - val_loss: 0.0922 - val_acc: 0.9695
Epoch 13/20
60000/60000 [=====] - 9s 151us/step - loss: 0.
0268 - acc: 0.9950 - val_loss: 0.0892 - val_acc: 0.9715
Epoch 14/20
60000/60000 [=====] - 9s 153us/step - loss: 0.
0236 - acc: 0.9959 - val_loss: 0.0884 - val_acc: 0.9719
Epoch 15/20
60000/60000 [=====] - 9s 154us/step - loss: 0.
0218 - acc: 0.9961 - val_loss: 0.0872 - val_acc: 0.9717
Epoch 16/20
60000/60000 [=====] - 9s 157us/step - loss: 0.
0184 - acc: 0.9974 - val_loss: 0.0873 - val_acc: 0.9715
Epoch 17/20
60000/60000 [=====] - 9s 157us/step - loss: 0.
0172 - acc: 0.9975 - val_loss: 0.0872 - val_acc: 0.9709
Epoch 18/20
60000/60000 [=====] - 9s 158us/step - loss: 0.
0156 - acc: 0.9976 - val_loss: 0.0847 - val_acc: 0.9724
Epoch 19/20
60000/60000 [=====] - 10s 161us/step - loss:
0.0139 - acc: 0.9981 - val_loss: 0.0860 - val_acc: 0.9722
Epoch 20/20
60000/60000 [=====] - 10s 159us/step - loss:
0.0135 - acc: 0.9981 - val_loss: 0.0873 - val_acc: 0.9717
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.08727886506225914
Test accuracy: 0.9717
```

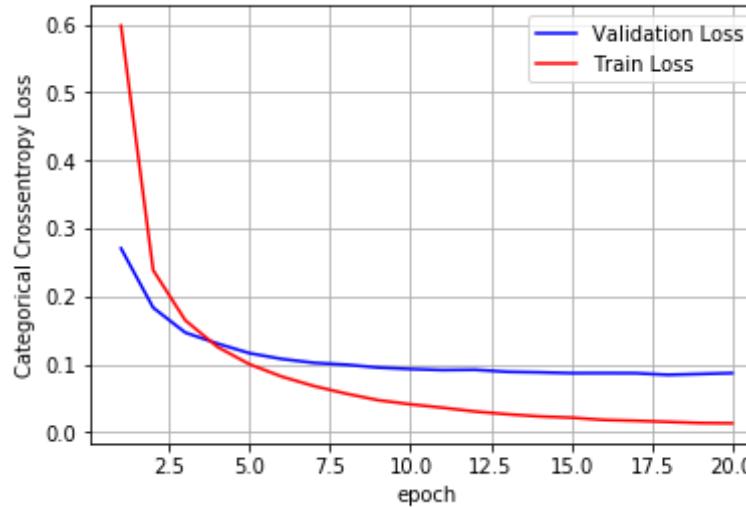
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

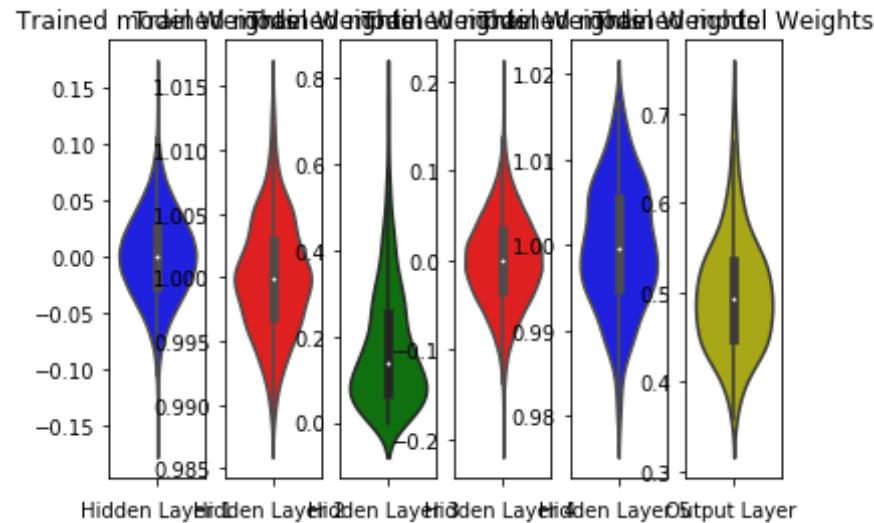
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



Model 5: relu activation with BatchNormalization + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.22, seed=None)))
```

```
l(mean=0.0, stddev=0.246, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_91"

Layer (type)	Output Shape	Param #
dense_344 (Dense)	(None, 512)	401920
batch_normalization_88 (Batch Normalization)	(None, 512)	2048
dense_345 (Dense)	(None, 256)	131328
batch_normalization_89 (Batch Normalization)	(None, 256)	1024
dense_346 (Dense)	(None, 128)	32896
batch_normalization_90 (Batch Normalization)	(None, 128)	512
dense_347 (Dense)	(None, 64)	8256
batch_normalization_91 (Batch Normalization)	(None, 64)	256
dense_348 (Dense)	(None, 32)	2080
batch_normalization_92 (Batch Normalization)	(None, 32)	128
dense_349 (Dense)	(None, 10)	330

Total params: 580,778

Trainable params: 578,794

Non-trainable params: 1,984

---

In [0]: model\_1.compile(optimizer='adam', loss='categorical\_crossentropy', metr

```
    ics=[ 'accuracy' ])
```

```
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 22s 363us/step - loss:
0.2827 - acc: 0.9230 - val_loss: 0.1201 - val_acc: 0.9652
Epoch 2/20
60000/60000 [=====] - 11s 188us/step - loss:
0.0895 - acc: 0.9736 - val_loss: 0.0947 - val_acc: 0.9724
Epoch 3/20
60000/60000 [=====] - 11s 186us/step - loss:
0.0621 - acc: 0.9814 - val_loss: 0.0809 - val_acc: 0.9741
Epoch 4/20
60000/60000 [=====] - 11s 188us/step - loss:
0.0486 - acc: 0.9850 - val_loss: 0.0956 - val_acc: 0.9721
Epoch 5/20
60000/60000 [=====] - 11s 190us/step - loss:
0.0372 - acc: 0.9887 - val_loss: 0.0878 - val_acc: 0.9757
Epoch 6/20
60000/60000 [=====] - 11s 190us/step - loss:
0.0337 - acc: 0.9891 - val_loss: 0.0906 - val_acc: 0.9735
Epoch 7/20
60000/60000 [=====] - 11s 190us/step - loss:
0.0299 - acc: 0.9902 - val_loss: 0.0814 - val_acc: 0.9773
Epoch 8/20
60000/60000 [=====] - 11s 188us/step - loss:
0.0258 - acc: 0.9915 - val_loss: 0.0697 - val_acc: 0.9799
Epoch 9/20
60000/60000 [=====] - 11s 187us/step - loss:
0.0225 - acc: 0.9927 - val_loss: 0.0693 - val_acc: 0.9807
Epoch 10/20
60000/60000 [=====] - 11s 188us/step - loss:
0.0209 - acc: 0.9936 - val_loss: 0.0817 - val_acc: 0.9762
Epoch 11/20
60000/60000 [=====] - 11s 190us/step - loss:
0.0197 - acc: 0.9935 - val_loss: 0.0822 - val_acc: 0.9785
Epoch 12/20
60000/60000 [=====] - 11s 190us/step - loss:
```

```
60000/60000 [=====] - 11s 188us/step - loss:  
0.0178 - acc: 0.9943 - val_loss: 0.0690 - val_acc: 0.9814  
Epoch 13/20  
60000/60000 [=====] - 12s 193us/step - loss:  
0.0187 - acc: 0.9938 - val_loss: 0.0737 - val_acc: 0.9803  
Epoch 14/20  
60000/60000 [=====] - 11s 188us/step - loss:  
0.0158 - acc: 0.9947 - val_loss: 0.0799 - val_acc: 0.9790  
Epoch 15/20  
60000/60000 [=====] - 11s 187us/step - loss:  
0.0144 - acc: 0.9950 - val_loss: 0.0785 - val_acc: 0.9806  
Epoch 16/20  
60000/60000 [=====] - 11s 190us/step - loss:  
0.0144 - acc: 0.9952 - val_loss: 0.0745 - val_acc: 0.9814  
Epoch 17/20  
60000/60000 [=====] - 11s 190us/step - loss:  
0.0139 - acc: 0.9954 - val_loss: 0.0766 - val_acc: 0.9807  
Epoch 18/20  
60000/60000 [=====] - 11s 190us/step - loss:  
0.0123 - acc: 0.9961 - val_loss: 0.0787 - val_acc: 0.9815  
Epoch 19/20  
60000/60000 [=====] - 12s 193us/step - loss:  
0.0109 - acc: 0.9963 - val_loss: 0.0665 - val_acc: 0.9833  
Epoch 20/20  
60000/60000 [=====] - 11s 191us/step - loss:  
0.0090 - acc: 0.9969 - val_loss: 0.0815 - val_acc: 0.9805
```

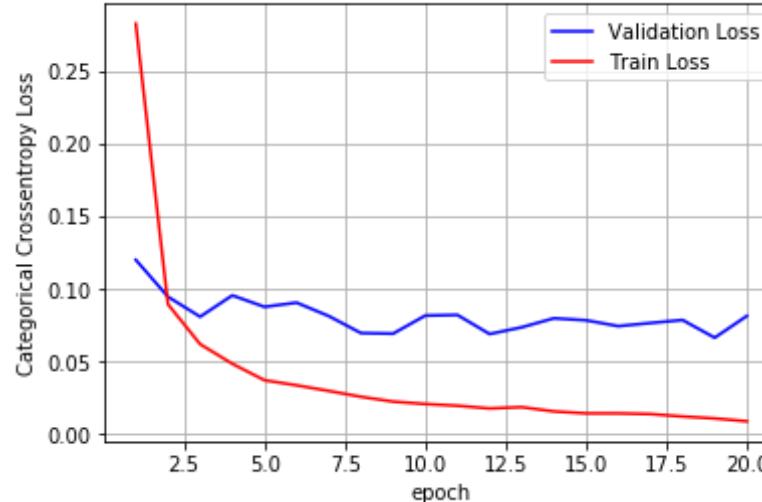
```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.08148554847669438  
Test accuracy: 0.9805
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

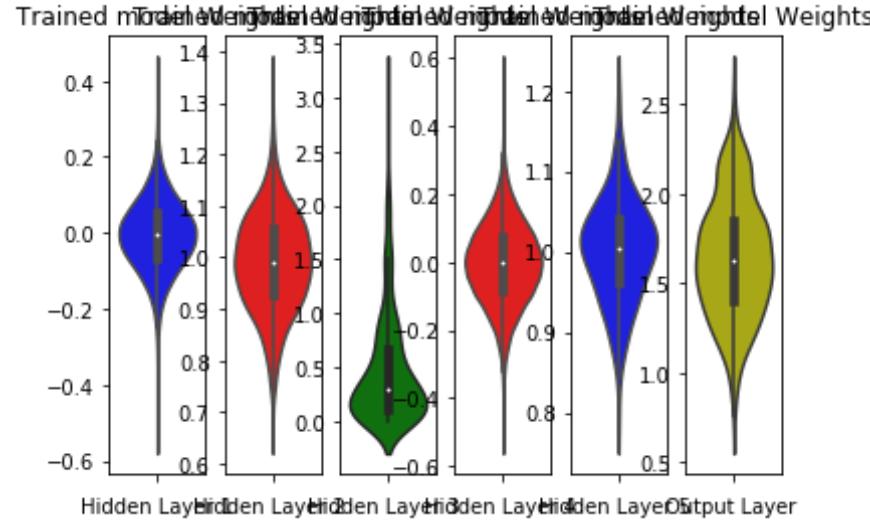
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



### Model 5: relu activation with BatchNormalization + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(BatchNormalization())

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(BatchNormalization())
```

```
model_1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(BatchNormalization())
model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_92"

Layer (type)	Output Shape	Param #
dense_350 (Dense)	(None, 512)	401920
batch_normalization_93 (BatchNormalization)	(None, 512)	2048
dense_351 (Dense)	(None, 256)	131328
batch_normalization_94 (BatchNormalization)	(None, 256)	1024
dense_352 (Dense)	(None, 128)	32896
batch_normalization_95 (BatchNormalization)	(None, 128)	512
dense_353 (Dense)	(None, 64)	8256
batch_normalization_96 (BatchNormalization)	(None, 64)	256
dense_354 (Dense)	(None, 32)	2080
batch_normalization_97 (BatchNormalization)	(None, 32)	128
dense_355 (Dense)	(None, 10)	330

Total params: 580,778

Trainable params: 578,794

Non-trainable params: 1,984

In [0]: model\_1.compile(optimizer='sgd', loss='categorical\_crossentropy', metri

```
cs=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 20s 341us/step - loss:
0.8386 - acc: 0.7594 - val_loss: 0.4028 - val_acc: 0.8928
Epoch 2/20
60000/60000 [=====] - 10s 161us/step - loss:
0.3508 - acc: 0.9055 - val_loss: 0.2728 - val_acc: 0.9247
Epoch 3/20
60000/60000 [=====] - 10s 165us/step - loss:
0.2513 - acc: 0.9306 - val_loss: 0.2216 - val_acc: 0.9385
Epoch 4/20
60000/60000 [=====] - 10s 159us/step - loss:
0.2027 - acc: 0.9429 - val_loss: 0.1915 - val_acc: 0.9449
Epoch 5/20
60000/60000 [=====] - 10s 159us/step - loss:
0.1697 - acc: 0.9518 - val_loss: 0.1733 - val_acc: 0.9498
Epoch 6/20
60000/60000 [=====] - 10s 160us/step - loss:
0.1485 - acc: 0.9579 - val_loss: 0.1610 - val_acc: 0.9534
Epoch 7/20
60000/60000 [=====] - 10s 165us/step - loss:
0.1296 - acc: 0.9633 - val_loss: 0.1502 - val_acc: 0.9552
Epoch 8/20
60000/60000 [=====] - 10s 163us/step - loss:
0.1162 - acc: 0.9670 - val_loss: 0.1422 - val_acc: 0.9569
Epoch 9/20
60000/60000 [=====] - 10s 160us/step - loss:
0.1037 - acc: 0.9705 - val_loss: 0.1360 - val_acc: 0.9599
Epoch 10/20
60000/60000 [=====] - 10s 159us/step - loss:
0.0934 - acc: 0.9738 - val_loss: 0.1312 - val_acc: 0.9608
Epoch 11/20
60000/60000 [=====] - 10s 164us/step - loss:
0.0836 - acc: 0.9775 - val_loss: 0.1290 - val_acc: 0.9611
Epoch 12/20
60000/60000 [=====] - 10s 160us/step - loss:
0.0744 - acc: 0.9808 - val_loss: 0.1270 - val_acc: 0.9615
```

```
60000/60000 [=====] - 10s 160us/step - loss:  
0.0760 - acc: 0.9790 - val_loss: 0.1254 - val_acc: 0.9616  
Epoch 13/20  
60000/60000 [=====] - 10s 162us/step - loss:  
0.0701 - acc: 0.9814 - val_loss: 0.1231 - val_acc: 0.9630  
Epoch 14/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.0640 - acc: 0.9825 - val_loss: 0.1204 - val_acc: 0.9643  
Epoch 15/20  
60000/60000 [=====] - 10s 161us/step - loss:  
0.0579 - acc: 0.9846 - val_loss: 0.1188 - val_acc: 0.9645  
Epoch 16/20  
60000/60000 [=====] - 10s 159us/step - loss:  
0.0546 - acc: 0.9856 - val_loss: 0.1180 - val_acc: 0.9641  
Epoch 17/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.0487 - acc: 0.9874 - val_loss: 0.1159 - val_acc: 0.9658  
Epoch 18/20  
60000/60000 [=====] - 10s 160us/step - loss:  
0.0440 - acc: 0.9892 - val_loss: 0.1155 - val_acc: 0.9654  
Epoch 19/20  
60000/60000 [=====] - 10s 167us/step - loss:  
0.0413 - acc: 0.9895 - val_loss: 0.1146 - val_acc: 0.9660  
Epoch 20/20  
60000/60000 [=====] - 9s 158us/step - loss: 0.  
0376 - acc: 0.9912 - val_loss: 0.1143 - val_acc: 0.9666
```

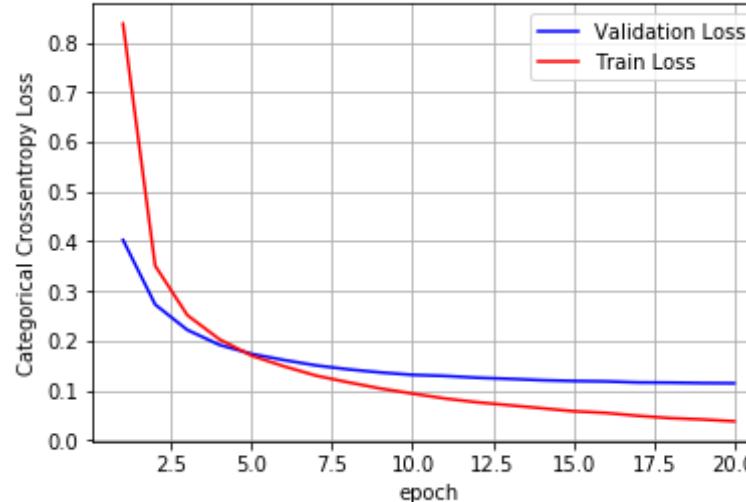
```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)  
print('Test score:', score[0])  
print('Test accuracy:', score[1])
```

```
Test score: 0.11425882778484374  
Test accuracy: 0.9666
```

```
In [0]: fig,ax = plt.subplots(1,1)  
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')  
  
# list of epoch numbers  
x = list(range(1,nb_epoch+1))
```

```
vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

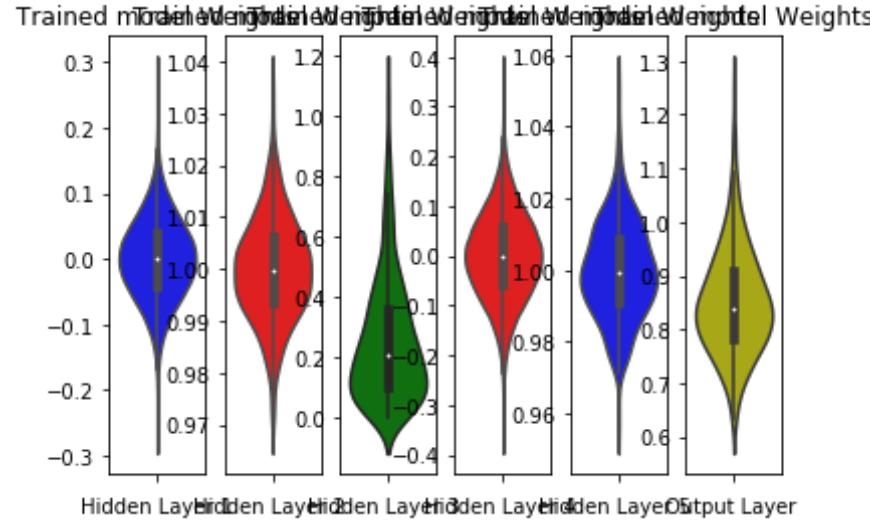
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 6: sigmoid activation with Dropout + AdamOptimizer with SGD Weight

```
In [0]: from keras.layers import Dropout

model_1 = Sequential()

model_1.add(Dense(256, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.044, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))
```

```

model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()

```

Model: "sequential\_93"

Layer (type)	Output Shape	Param #
<hr/>		
dense_356 (Dense)	(None, 256)	200960
dropout_71 (Dropout)	(None, 256)	0
dense_357 (Dense)	(None, 512)	131584
dropout_72 (Dropout)	(None, 512)	0
dense_358 (Dense)	(None, 256)	131328
dropout_73 (Dropout)	(None, 256)	0
dense_359 (Dense)	(None, 128)	32896
dropout_74 (Dropout)	(None, 128)	0
dense_360 (Dense)	(None, 64)	8256
dropout_75 (Dropout)	(None, 64)	0
dense_361 (Dense)	(None, 32)	2080

```
dropout_76 (Dropout)           (None, 32)          0
dense_362 (Dense)             (None, 10)         330
=====
Total params: 507,434
Trainable params: 507,434
Non-trainable params: 0
```

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 20s 339us/step - loss: 2.3372 - acc: 0.1035 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 2/20
60000/60000 [=====] - 10s 172us/step - loss: 2.0959 - acc: 0.1831 - val_loss: 1.5800 - val_acc: 0.3579
Epoch 3/20
60000/60000 [=====] - 10s 173us/step - loss: 1.4680 - acc: 0.3686 - val_loss: 1.1738 - val_acc: 0.4790
Epoch 4/20
60000/60000 [=====] - 10s 171us/step - loss: 1.2212 - acc: 0.4466 - val_loss: 1.0062 - val_acc: 0.4952
Epoch 5/20
60000/60000 [=====] - 10s 172us/step - loss: 1.1057 - acc: 0.4717 - val_loss: 0.9403 - val_acc: 0.5080
Epoch 6/20
60000/60000 [=====] - 10s 172us/step - loss: 1.0494 - acc: 0.4777 - val_loss: 0.9066 - val_acc: 0.5083
Epoch 7/20
60000/60000 [=====] - 10s 174us/step - loss: 1.0124 - acc: 0.4877 - val_loss: 0.8961 - val_acc: 0.5114
Epoch 8/20
60000/60000 [=====] - 10s 171us/step - loss: 0.9823 - acc: 0.4960 - val_loss: 0.8721 - val_acc: 0.5244
```

```
v1.3.2rc3      acc: 0.7900      val_loss: 0.4724      val_acc: 0.7277
Epoch 9/20
60000/60000 [=====] - 10s 173us/step - loss: 0.9326 - acc: 0.5586 - val_loss: 0.6792 - val_acc: 0.7030
Epoch 10/20
60000/60000 [=====] - 10s 173us/step - loss: 0.7907 - acc: 0.6649 - val_loss: 0.5380 - val_acc: 0.7696
Epoch 11/20
60000/60000 [=====] - 10s 173us/step - loss: 0.7085 - acc: 0.7152 - val_loss: 0.4835 - val_acc: 0.7790
Epoch 12/20
60000/60000 [=====] - 10s 170us/step - loss: 0.6586 - acc: 0.7366 - val_loss: 0.4614 - val_acc: 0.7852
Epoch 13/20
60000/60000 [=====] - 10s 170us/step - loss: 0.6246 - acc: 0.7478 - val_loss: 0.4447 - val_acc: 0.7962
Epoch 14/20
60000/60000 [=====] - 10s 169us/step - loss: 0.5967 - acc: 0.7622 - val_loss: 0.4096 - val_acc: 0.8429
Epoch 15/20
60000/60000 [=====] - 10s 171us/step - loss: 0.5633 - acc: 0.7919 - val_loss: 0.3643 - val_acc: 0.8678
Epoch 16/20
60000/60000 [=====] - 10s 174us/step - loss: 0.5323 - acc: 0.8072 - val_loss: 0.3406 - val_acc: 0.8707
Epoch 17/20
60000/60000 [=====] - 10s 174us/step - loss: 0.5022 - acc: 0.8181 - val_loss: 0.3221 - val_acc: 0.8726
Epoch 18/20
60000/60000 [=====] - 11s 175us/step - loss: 0.4764 - acc: 0.8272 - val_loss: 0.3175 - val_acc: 0.8742
Epoch 19/20
60000/60000 [=====] - 11s 176us/step - loss: 0.4574 - acc: 0.8314 - val_loss: 0.3075 - val_acc: 0.8777
Epoch 20/20
60000/60000 [=====] - 10s 173us/step - loss: 0.4466 - acc: 0.8344 - val_loss: 0.2954 - val_acc: 0.8802
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

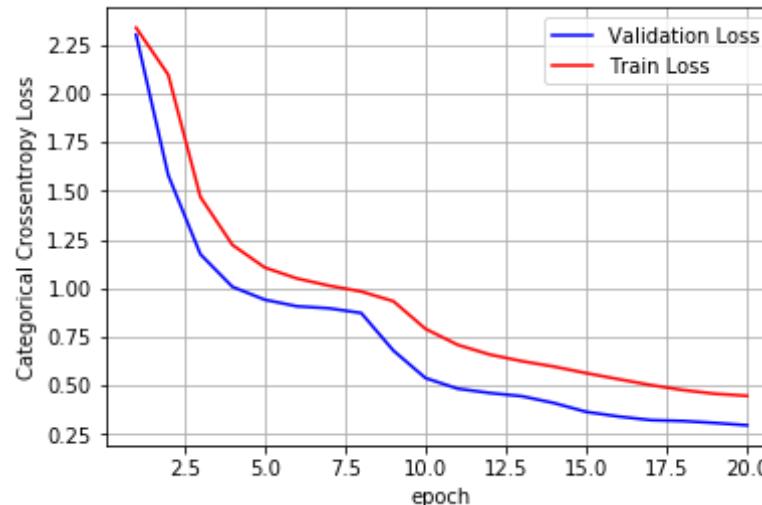
```
Test score: 0.2954470875740051
Test accuracy: 0.8802
```

```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
```

```
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

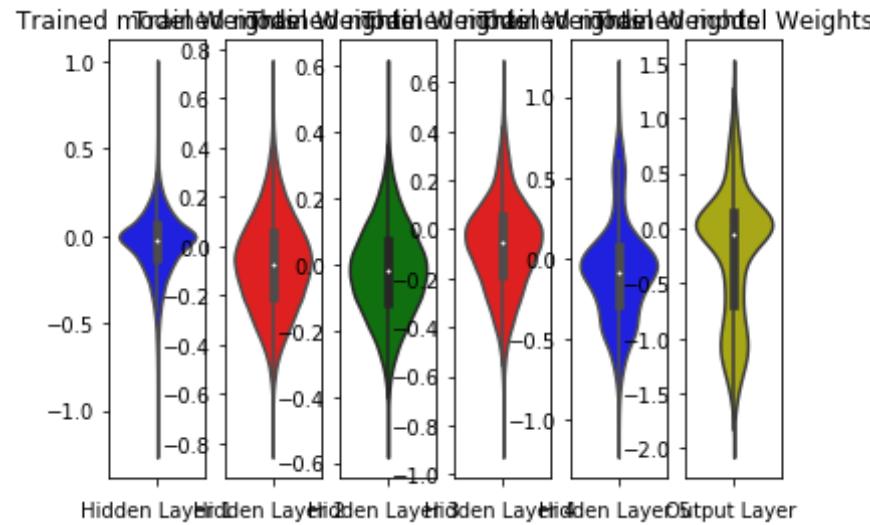
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



### Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with SGD Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.051, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.072, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.102, seed=None)))
model_1.add(Dropout(0.5))
```

```
model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.144, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_94"

Layer (type)	Output Shape	Param #
<hr/>		
dense_363 (Dense)	(None, 512)	401920
dropout_77 (Dropout)	(None, 512)	0
dense_364 (Dense)	(None, 256)	131328
dropout_78 (Dropout)	(None, 256)	0
dense_365 (Dense)	(None, 128)	32896
dropout_79 (Dropout)	(None, 128)	0
dense_366 (Dense)	(None, 64)	8256
dropout_80 (Dropout)	(None, 64)	0
dense_367 (Dense)	(None, 32)	2080
dropout_81 (Dropout)	(None, 32)	0
dense_368 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 19s 315us/step - loss: 2.4703 - acc: 0.1019 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 2/20
60000/60000 [=====] - 9s 144us/step - loss: 2.3927 - acc: 0.1027 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 3/20
60000/60000 [=====] - 9s 145us/step - loss: 2.3628 - acc: 0.1034 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 4/20
60000/60000 [=====] - 9s 144us/step - loss: 2.3476 - acc: 0.1009 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 5/20
60000/60000 [=====] - 9s 144us/step - loss: 2.3369 - acc: 0.0988 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 6/20
60000/60000 [=====] - 9s 144us/step - loss: 2.3286 - acc: 0.0999 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 9s 147us/step - loss: 2.3236 - acc: 0.1033 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 9s 146us/step - loss: 2.3205 - acc: 0.1026 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 9s 146us/step - loss: 2.3169 - acc: 0.1014 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 9s 148us/step - loss: 2.3151 - acc: 0.1026 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 9s 146us/step - loss: 2.3118 - acc: 0.1047 - val_loss: 2.3011 - val_acc: 0.1135
```

```
Epoch 12/20
60000/60000 [=====] - 9s 148us/step - loss: 2.
3110 - acc: 0.1018 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 13/20
60000/60000 [=====] - 9s 154us/step - loss: 2.
3105 - acc: 0.1033 - val_loss: 2.3010 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 9s 153us/step - loss: 2.
3090 - acc: 0.1035 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 15/20
60000/60000 [=====] - 9s 155us/step - loss: 2.
3077 - acc: 0.1038 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 16/20
60000/60000 [=====] - 9s 151us/step - loss: 2.
3078 - acc: 0.1036 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 17/20
60000/60000 [=====] - 9s 158us/step - loss: 2.
3073 - acc: 0.1039 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 18/20
60000/60000 [=====] - 9s 152us/step - loss: 2.
3066 - acc: 0.1043 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 19/20
60000/60000 [=====] - 9s 152us/step - loss: 2.
3064 - acc: 0.1045 - val_loss: 2.3011 - val_acc: 0.1135
Epoch 20/20
60000/60000 [=====] - 9s 150us/step - loss: 2.
3060 - acc: 0.1047 - val_loss: 2.3011 - val_acc: 0.1135
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 2.3010885906219483
Test accuracy: 0.1135
```

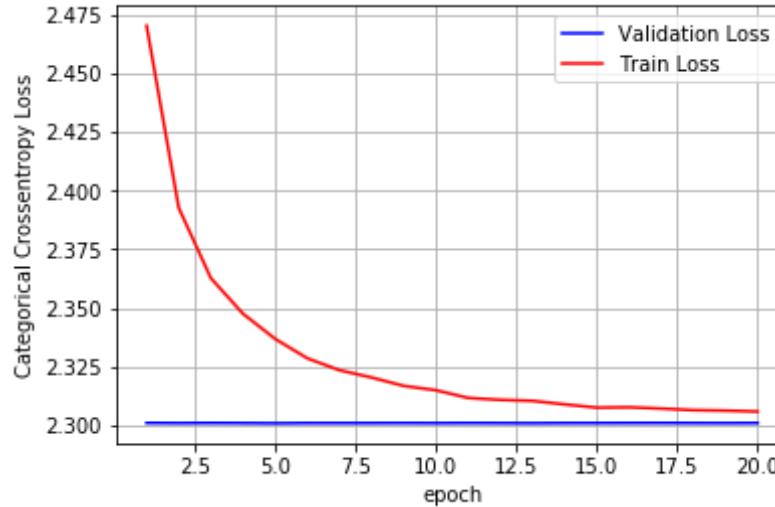
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

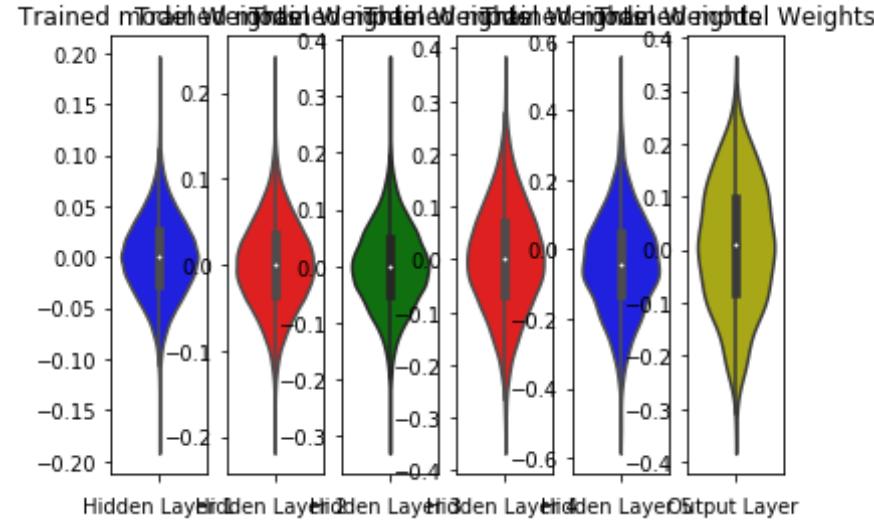
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



Model 6: sigmoid activation with Dropout + AdamOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNo
```

```
    rmal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_95"

Layer (type)	Output Shape	Param #
<hr/>		
dense_369 (Dense)	(None, 512)	401920
dropout_82 (Dropout)	(None, 512)	0
dense_370 (Dense)	(None, 256)	131328
dropout_83 (Dropout)	(None, 256)	0
dense_371 (Dense)	(None, 128)	32896
dropout_84 (Dropout)	(None, 128)	0
dense_372 (Dense)	(None, 64)	8256
dropout_85 (Dropout)	(None, 64)	0
dense_373 (Dense)	(None, 32)	2080
dropout_86 (Dropout)	(None, 32)	0
dense_374 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

```
In [0]: model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 21s 348us/step - loss:
2.0744 - acc: 0.2115 - val_loss: 1.3061 - val_acc: 0.6436
Epoch 2/20
60000/60000 [=====] - 10s 174us/step - loss:
1.2352 - acc: 0.5555 - val_loss: 0.6935 - val_acc: 0.8672
Epoch 3/20
60000/60000 [=====] - 11s 175us/step - loss:
0.8730 - acc: 0.7302 - val_loss: 0.4499 - val_acc: 0.9161
Epoch 4/20
60000/60000 [=====] - 11s 177us/step - loss:
0.7012 - acc: 0.8031 - val_loss: 0.3391 - val_acc: 0.9337
Epoch 5/20
60000/60000 [=====] - 11s 175us/step - loss:
0.6018 - acc: 0.8397 - val_loss: 0.2778 - val_acc: 0.9450
Epoch 6/20
60000/60000 [=====] - 11s 177us/step - loss:
0.5321 - acc: 0.8613 - val_loss: 0.2446 - val_acc: 0.9501
Epoch 7/20
60000/60000 [=====] - 10s 175us/step - loss:
0.4930 - acc: 0.8725 - val_loss: 0.2225 - val_acc: 0.9545
Epoch 8/20
60000/60000 [=====] - 11s 177us/step - loss:
0.4541 - acc: 0.8843 - val_loss: 0.2084 - val_acc: 0.9574
Epoch 9/20
60000/60000 [=====] - 11s 178us/step - loss:
0.4275 - acc: 0.8885 - val_loss: 0.2020 - val_acc: 0.9580
Epoch 10/20
60000/60000 [=====] - 11s 179us/step - loss:
0.4015 - acc: 0.8950 - val_loss: 0.1830 - val_acc: 0.9614
Epoch 11/20
60000/60000 [=====] - 11s 178us/step - loss:
0.3784 - acc: 0.9008 - val_loss: 0.1742 - val_acc: 0.9643
```

```
Epoch 12/20
60000/60000 [=====] - 11s 180us/step - loss:
0.3600 - acc: 0.9047 - val_loss: 0.1658 - val_acc: 0.9655
Epoch 13/20
60000/60000 [=====] - 11s 179us/step - loss:
0.3451 - acc: 0.9086 - val_loss: 0.1592 - val_acc: 0.9680
Epoch 14/20
60000/60000 [=====] - 11s 178us/step - loss:
0.3293 - acc: 0.9123 - val_loss: 0.1526 - val_acc: 0.9699
Epoch 15/20
60000/60000 [=====] - 11s 179us/step - loss:
0.3206 - acc: 0.9138 - val_loss: 0.1478 - val_acc: 0.9682
Epoch 16/20
60000/60000 [=====] - 11s 179us/step - loss:
0.3058 - acc: 0.9182 - val_loss: 0.1443 - val_acc: 0.9707
Epoch 17/20
60000/60000 [=====] - 11s 185us/step - loss:
0.2947 - acc: 0.9219 - val_loss: 0.1437 - val_acc: 0.9712
Epoch 18/20
60000/60000 [=====] - 11s 181us/step - loss:
0.2872 - acc: 0.9223 - val_loss: 0.1351 - val_acc: 0.9733
Epoch 19/20
60000/60000 [=====] - 11s 180us/step - loss:
0.2791 - acc: 0.9248 - val_loss: 0.1390 - val_acc: 0.9715
Epoch 20/20
60000/60000 [=====] - 11s 178us/step - loss:
0.2691 - acc: 0.9271 - val_loss: 0.1258 - val_acc: 0.9731
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.12581829980649054
Test accuracy: 0.9731
```

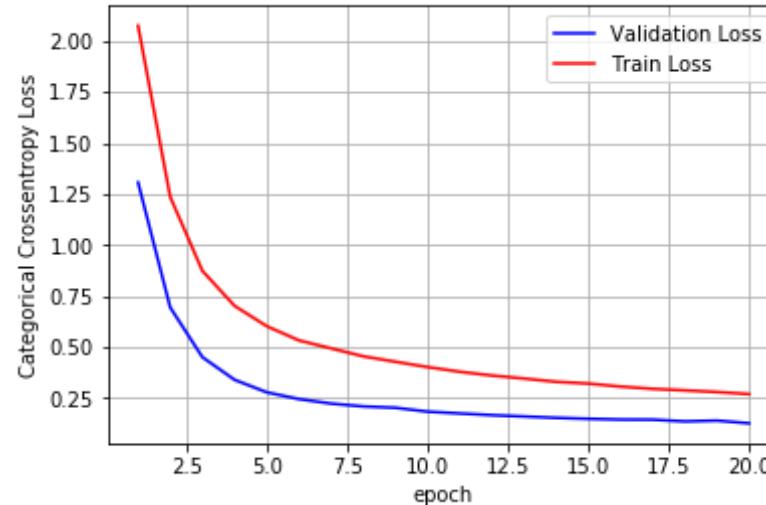
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

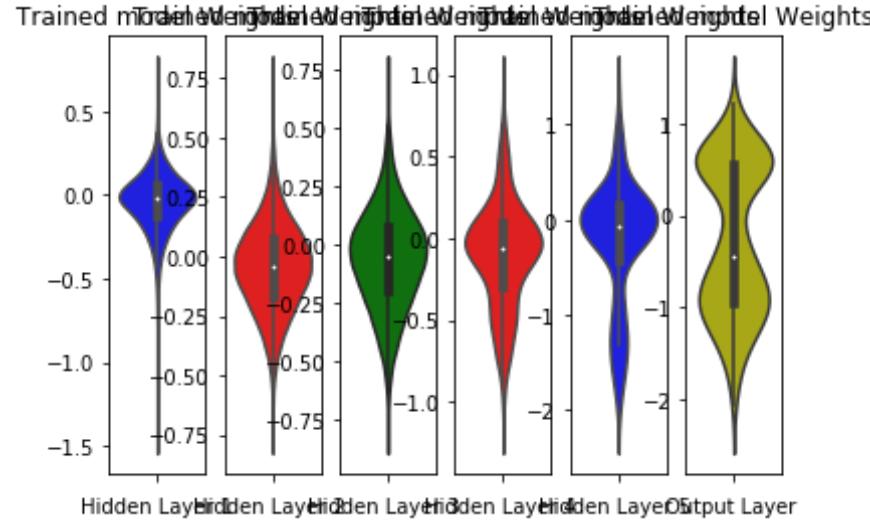
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



## Model 6: sigmoid activation with Dropout + GradientDescentOptimizer with Relu Weight

```
In [0]: model_1 = Sequential()

model_1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(256, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.088, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.175, seed=None)))
model_1.add(Dropout(0.5))
```

```
model_1.add(Dense(32, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.246, seed=None)))
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

Model: "sequential\_96"

Layer (type)	Output Shape	Param #
<hr/>		
dense_375 (Dense)	(None, 512)	401920
dropout_87 (Dropout)	(None, 512)	0
dense_376 (Dense)	(None, 256)	131328
dropout_88 (Dropout)	(None, 256)	0
dense_377 (Dense)	(None, 128)	32896
dropout_89 (Dropout)	(None, 128)	0
dense_378 (Dense)	(None, 64)	8256
dropout_90 (Dropout)	(None, 64)	0
dense_379 (Dense)	(None, 32)	2080
dropout_91 (Dropout)	(None, 32)	0
dense_380 (Dense)	(None, 10)	330
<hr/>		
Total params: 576,810		
Trainable params: 576,810		
Non-trainable params: 0		

---

```
In [0]: model_1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [=====] - 20s 326us/step - loss: 2.4863 - acc: 0.1019 - val_loss: 2.3030 - val_acc: 0.0962
Epoch 2/20
60000/60000 [=====] - 9s 152us/step - loss: 2.4109 - acc: 0.0986 - val_loss: 2.3032 - val_acc: 0.1028
Epoch 3/20
60000/60000 [=====] - 9s 152us/step - loss: 2.3715 - acc: 0.1015 - val_loss: 2.3029 - val_acc: 0.1190
Epoch 4/20
60000/60000 [=====] - 9s 149us/step - loss: 2.3520 - acc: 0.1006 - val_loss: 2.3021 - val_acc: 0.1045
Epoch 5/20
60000/60000 [=====] - 9s 150us/step - loss: 2.3386 - acc: 0.1019 - val_loss: 2.3024 - val_acc: 0.1039
Epoch 6/20
60000/60000 [=====] - 9s 151us/step - loss: 2.3284 - acc: 0.1035 - val_loss: 2.3017 - val_acc: 0.1135
Epoch 7/20
60000/60000 [=====] - 9s 150us/step - loss: 2.3255 - acc: 0.1012 - val_loss: 2.3018 - val_acc: 0.1135
Epoch 8/20
60000/60000 [=====] - 9s 151us/step - loss: 2.3208 - acc: 0.1008 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 9/20
60000/60000 [=====] - 9s 152us/step - loss: 2.3175 - acc: 0.1018 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 10/20
60000/60000 [=====] - 9s 150us/step - loss: 2.3147 - acc: 0.1041 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 11/20
60000/60000 [=====] - 9s 149us/step - loss: 2.3119 - acc: 0.1041 - val_loss: 2.3014 - val_acc: 0.1135
```

```
Epoch 12/20
60000/60000 [=====] - 9s 147us/step - loss: 2.
3105 - acc: 0.1029 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 13/20
60000/60000 [=====] - 9s 146us/step - loss: 2.
3103 - acc: 0.1050 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 14/20
60000/60000 [=====] - 9s 146us/step - loss: 2.
3086 - acc: 0.1041 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 15/20
60000/60000 [=====] - 9s 145us/step - loss: 2.
3081 - acc: 0.1054 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 16/20
60000/60000 [=====] - 9s 142us/step - loss: 2.
3074 - acc: 0.1054 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 17/20
60000/60000 [=====] - 9s 143us/step - loss: 2.
3069 - acc: 0.1047 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 18/20
60000/60000 [=====] - 9s 143us/step - loss: 2.
3070 - acc: 0.1048 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 19/20
60000/60000 [=====] - 9s 142us/step - loss: 2.
3058 - acc: 0.1044 - val_loss: 2.3012 - val_acc: 0.1135
Epoch 20/20
60000/60000 [=====] - 8s 139us/step - loss: 2.
3055 - acc: 0.1045 - val_loss: 2.3012 - val_acc: 0.1135
```

```
In [0]: score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 2.3011954772949217
Test accuracy: 0.1135
```

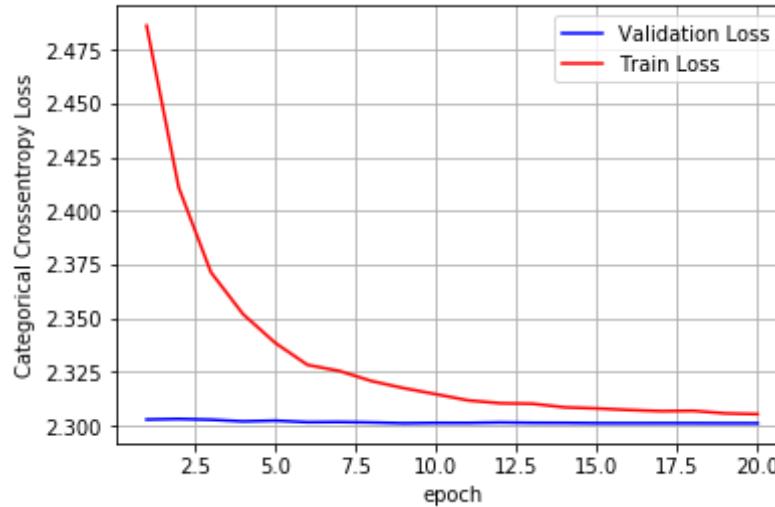
```
In [0]: fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch'); ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']

plt_dynamic(x, vy, ty, ax)
```



```
In [0]: w_after = model_1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)

fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
```

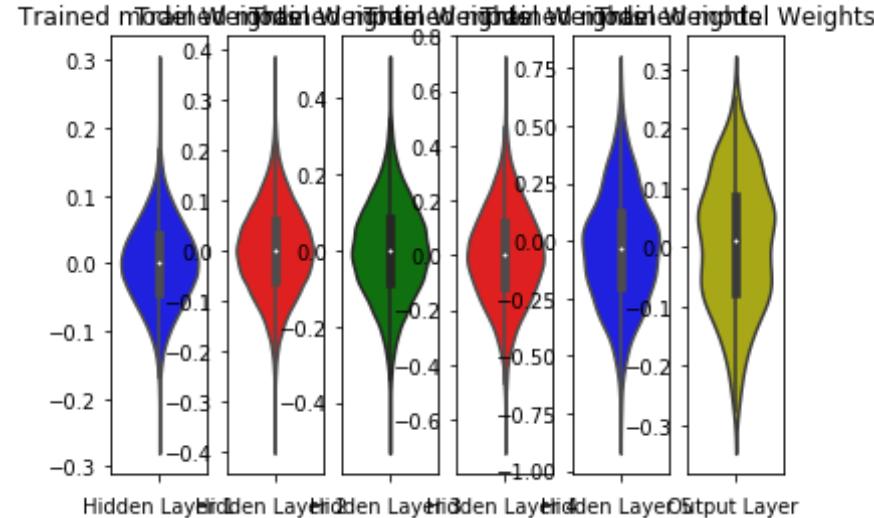
```
plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



```
In [0]: # http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Model Name", "Accuracy(%)", "Test Cost"]
x.add_row(["Architecture 1: 784-501-101-10", "input-sigmoid(501)-sigmoid(101)-sigmoid(output)-AdamOptimizer-SGD Weight", 98.19, 0.07])
x.add_row(["Architecture 1: 784-501-101-10", "input-sigmoid(501)-sigmoid(101)-sigmoid(output)-GradientDescentOptimizer-SGD Weight", 87.68, 0.47])
x.add_row(["Architecture 1: 784-501-101-10", "input-sigmoid(501)-sigmoid(101)-sigmoid(output)-AdamOptimizer-Relu Weight", 97.89, 0.07])
x.add_row(["Architecture 1: 784-501-101-10", "input-sigmoid(501)-sigmoid(101)-sigmoid(output)-GradientDescentOptimizer-Relu Weight", 89.99, 0.38])
x.add_row(["Architecture 1: 784-501-101-10", "input(784)-ReLU(501)-ReLU(101)-sigmoid(output 10)-AdamOptimizer-SGD Weight", 98.09, 0.09])
x.add_row(["Architecture 1: 784-501-101-10", "input(784)-ReLU(501)-ReLU(101)-sigmoid(output 10)-GradientDescentOptimizer-Relu Weight", 89.99, 0.38])
```

```
(101)-sigmoid(output 10)-AdamOptimizer-GradientDescentOptimizer-SGD Wei
ght", 95.82, 0.15])
x.add_row(["Architecture 1: 784-501-101-10", "input(784)-ReLU(501)-ReLU
(101)-sigmoid(output 10)-AdamOptimizer-AdamOptimizer-Relu Weight", 98.0
6, 0.1])
x.add_row(["Architecture 1: 784-501-101-10", "input(784)-ReLU(501)-ReLU
(101)-sigmoid(output 10)-AdamOptimizer-GradientDescentOptimizer-Relu We
ight", 96.35, 0.12])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Sigmoid(BN(501))-Si
gmoid(BN(101))-Sigmoid(output)-AdamOptimizer-SGD Weight", 97.82,
0.08])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Sigmoid(BN(501))-Si
gmoid(BN(101))-Sigmoid(output))-GradientDescentOptimizer-SGD Weight", 9
5.55, 0.16])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Sigmoid(BN(501))-Si
gmoid(BN(101))-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.8, 0.09])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Sigmoid(BN(501))-Si
gmoid(BN(101))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 9
5.31, 0.16])
x.add_row(["Architecture 1: 784-501-101-10", "Input-ReLU(501)-Dropout-R
eLu(101)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight", 98.15, 0.07
])
x.add_row(["Architecture 1: 784-501-101-10", "Input-ReLU(501)-Dropout-R
eLu(101)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight",
95.12, 0.17])
x.add_row(["Architecture 1: 784-501-101-10", "Input-ReLU(501)-Dropout-R
eLu(101)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight", 98.27, 0.0
7])
x.add_row(["Architecture 1: 784-501-101-10", "Input-ReLU(501)-Dropout-R
eLu(101)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight",
68.01, 1.29])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Relu(BN(501))-Relu
(BN(101))-sigmoid(output)-AdamOptimizer-SGD Weight", 98.1, 0.08])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Relu(BN(501))-Relu
(BN(101))-sigmoid(output)-GradientDescentOptimizer-SGD Weight", 97.42,
0.08])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Relu(BN(501))-Relu
(BN(101))-sigmoid(output)-AdamOptimizer-Relu Weight", 98.14, 0.08])
x.add_row(["Architecture 1: 784-501-101-10", "Input-Relu(BN(501))-Relu
```

```

(BN(101))-sigmoid(output)-GradientDescentOptimizer-Relu Weight", 96.79,
    0.1])
x.add_row(["Architecture 1: 784-501-101-10", "Input-sigmoid(501)-Dropou
t-sigmoid(101)-Dropout-sigmoid(output)-AdamOptimizer-SGD Weight", 97.99
, 0.07])
x.add_row(["Architecture 1: 784-501-101-10", "Input-sigmoid(501)-Dropou
t-sigmoid(101)-Dropout-sigmoid(output)-GradientDescentOptimizer-SGD Wei
ght", 78.99, 0.78])
x.add_row(["Architecture 1: 784-501-101-10", "Input-sigmoid(501)-Dropou
t-sigmoid(101)-Dropout-sigmoid(output)-AdamOptimizer-Relu Weight", 97.9
2, 0.07])
x.add_row(["Architecture 1: 784-501-101-10", "Input-sigmoid(501)-Dropou
t-sigmoid(101)-Dropout-sigmoid(output)-GradientDescentOptimizer-Relu We
ight", 82.26, 0.63])
print(x)

```

Architecture		Model Name	Accu
racy(%)	Test Cost		
8.19	0.07	input-sigmoid(501)-sig moid(101)-sigmoid(output)-AdamOptimizer-SGD Weight	9
7.68	0.47	input-sigmoid(501)-sig moid(101)-sigmoid(output)-GradientDescentOptimizer-SGD Weight	8
7.89	0.07	input-sigmoid(501)-sig moid(101)-sigmoid(output)-AdamOptimizer-Relu Weight	9
9.99	0.38	input-sigmoid(501)-sig moid(101)-sigmoid(output)-GradientDescentOptimizer-Relu Weight	8
~ ~	~ ~	input(784)-ReLU(501)-Re Lu(101)-sigmoid(output 10)-AdamOptimizer-SGD Weight	9

```
8.09 | 0.09 |
| Architecture 1: 784-501-101-10 | input(784)-ReLU(501)-ReLU(101)-sigmoid(output 10)-AdamOptimizer-GradientDescentOptimizer-SGD Weight | 9
5.82 | 0.15 |
| Architecture 1: 784-501-101-10 | input(784)-ReLU(501)-ReLU(101)-sigmoid(output 10)-AdamOptimizer-AdamOptimizer-Relu Weight | 9
8.06 | 0.1 |
| Architecture 1: 784-501-101-10 | input(784)-ReLU(501)-ReLU(101)-sigmoid(output 10)-AdamOptimizer-GradientDescentOptimizer-Relu Weight | 9
6.35 | 0.12 |
| Architecture 1: 784-501-101-10 | Input-Sigmoid(BN(501))-Sigmoid(BN(101))-Sigmoid(output)-AdamOptimizer-SGD Weight | 9
7.82 | 0.08 |
| Architecture 1: 784-501-101-10 | Input-Sigmoid(BN(501))-Sigmoid(BN(101))-Sigmoid(output))-GradientDescentOptimizer-SGD Weight | 9
5.55 | 0.16 |
| Architecture 1: 784-501-101-10 | Input-Sigmoid(BN(501))-Sigmoid(BN(101))-Sigmoid(output)-AdamOptimizer-Relu Weight | 9

97.8 | 0.09 |
| Architecture 1: 784-501-101-10 | Input-Sigmoid(BN(501))-Sigmoid(BN(101))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight | 9
5.31 | 0.16 |
| Architecture 1: 784-501-101-10 | Input-ReLU(501)-Dropout-ReLU(101)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight | 9
8.15 | 0.07 |
| Architecture 1: 784-501-101-10 | Input-ReLU(501)-Dropout-ReLU(101)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight | 9
5.12 | 0.17 |
| Architecture 1: 784-501-101-10 | Input-ReLU(501)-Dropout-ReLU(101)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight | 9
8.27 | 0.07 |
| Architecture 1: 784-501-101-10 | Input-ReLU(501)-Dropout-ReLU(101)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight | 6
8.01 | 1.29 |
| Architecture 1: 784-501-101-10 | Input-ReLU(BN(501))-ReLU(BN(101))-sigmoid(output)-AdamOptimizer-SGD Weight | 9
98.1 | 0.08 |
| Architecture 1: 784-501-101-10 | Input-ReLU(BN(501))-ReLU(BN(101))-sigmoid(output)-GradientDescentOptimizer-SGD Weight | 9
```

7.42		0.08				
	Architecture 1: 784-501-101-10		Input-Relu(BN(501))-Relu			
(BN(101))-sigmoid(output)-AdamOptimizer-Relu Weight						9
8.14		0.08				
	Architecture 1: 784-501-101-10		Input-Relu(BN(501))-Relu(BN(1			
01))-sigmoid(output)-GradientDescentOptimizer-Relu Weight			01))			9
6.79		0.1				
	Architecture 1: 784-501-101-10		Input-sigmoid(501)-Dropout-sig			
moid(101)-Dropout-sigmoid(output)-AdamOptimizer-SGD Weight			moid(101))			9
7.99		0.07				
	Architecture 1: 784-501-101-10		Input-sigmoid(501)-Dropout-sigmoid(1			
01)-Dropout-sigmoid(output)-GradientDescentOptimizer-SGD Weight			01))			7
8.99		0.78				
	Architecture 1: 784-501-101-10		Input-sigmoid(501)-Dropout-sigm			
oid(101)-Dropout-sigmoid(output)-AdamOptimizer-Relu Weight			oid(101))			9
7.92		0.07				
	Architecture 1: 784-501-101-10		Input-sigmoid(501)-Dropout-sigmoid(1			
01)-Dropout-sigmoid(output)-GradientDescentOptimizer-Relu Weight			01))			8
2.26		0.63				
+-----+-----+						
- - - - +-----+-----+						
- - - - +-----+-----+						

```
In [0]: # http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Model Name", "Accuracy(%)", "Test Cost"]
x.add_row(["Architecture 2: 784-256-128-64-10", "input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-AdamOptimizer-SGD Weight", 97.86, 0.09,])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-GradientDescentOptimizer-SGD Weight", 62.01, 1.23])
```

```
x.add_row(["Architecture 2: 784-256-128-64-10", "input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-AdamOptimizer-Relu Weight", 97.71, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-GradientDescentOptimizer-Relu Weight", 83.42, 0.67])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-SGD Weight", 97.88, 0.1])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-SGD Weight", 96.12, 0.13])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-AdamOptimizer-Relu Weight", 97.99, 0.1])
x.add_row(["Architecture 2: 784-256-128-64-10", "input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-Relu Weight", 96.17, 0.13])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-AdamOptimizer-SGD Weight", 97.86, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 95.98, 0.14])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.85, 0.1])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 95.24, 0.16])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight", 97.87, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 94.89, 0.18])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.58, 0.1])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Dropou
```

```

t-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 11.35, 2.35])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(BN(256))-ReLu(BN(128))-Relu(BN(64))-Sigmoid(output)-AdamOptimizer-SGD Weight", 98.08, 0.08])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(BN(256))-ReLu(BN(128))-Relu(BN(64))-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 97.49, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(BN(256))-ReLu(BN(128))-Relu(BN(64))-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.81, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(BN(256))-ReLu(BN(128))-Relu(BN(64))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 96.05, 0.13])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight", 97.5, 0.09])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 14.97, 2.28])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.33, 0.1])
x.add_row(["Architecture 2: 784-256-128-64-10", "Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 46.08, 1.94])
print(x)

```

Architecture Model Name	Accuracy(%)	Test Cost
Architecture 2: 784-256-128-64-10   input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-AdamOptimizer-SGD Weight	97.86	0.09

Architecture 2: 784-256-128-64-10	input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-GradientDescentOptimizer-SGD Weight
62.01   1.23	
Architecture 2: 784-256-128-64-10	input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-AdamOptimizer-Relu Weight
97.71   0.09	
Architecture 2: 784-256-128-64-10	input-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(output)-GradientDescentOptimizer-Relu Weight
83.42   0.67	
Architecture 2: 784-256-128-64-10	input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-SGD Weight
97.88   0.1	
Architecture 2: 784-256-128-64-10	input-ReLu(256)-ReLu(128)-ReLu(64)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-SGD Weight
96.12   0.13	
Architecture 2: 784-256-128-64-10	input-ReLu(256)-ReLU(128)-ReLU(64)-sigmoid(output)-AdamOptimizer-AdamOptimizer-Relu Weight
97.99   0.1	
Architecture 2: 784-256-128-64-10	input-ReLu(256)-ReLU(128)-ReLU(64)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-Relu Weight
96.17   0.13	
Architecture 2: 784-256-128-64-10	Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-AdamOptimizer-SGD Weight
97.86   0.09	
Architecture 2: 784-256-128-64-10	Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output))-GradientDescentOptimizer-SGD Weight
95.98   0.14	
Architecture 2: 784-256-128-64-10	Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-AdamOptimizer-Relu Weight
97.85   0.1	
Architecture 2: 784-256-128-64-10	Input-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight
95.24   0.16	
Architecture 2: 784-256-128-64-10	Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight
97.87   0.09	
Architecture 2: 784-256-128-64-10	Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight
94.89   0.18	
Architecture 2: 784-256-128-64-10	Input-ReLu(256)-Dropout-ReL

u(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight	97.58	0.1	
Architecture 2: 784-256-128-64-10   Input-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	11.35	2.35	
Architecture 2: 784-256-128-64-10   Input-ReLu(BN(256))-Relu(BN(128))-Relu(BN(64))-Sigmoid(output)-AdamOptimizer-SGD Weight	98.08	0.08	
Architecture 2: 784-256-128-64-10   Input-ReLu(BN(256))-Relu(BN(128))-Relu(BN(64))-Sigmoid(output)-GradientDescentOptimizer-SGD Weight	97.49	0.09	
Architecture 2: 784-256-128-64-10   Input-ReLu(BN(256))-Relu(BN(128))-Relu(BN(64))-Sigmoid(output)-AdamOptimizer-Relu Weight	97.81	0.09	
Architecture 2: 784-256-128-64-10   Input-ReLu(BN(256))-Relu(BN(128))-Relu(BN(64))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	96.05	0.13	
Architecture 2: 784-256-128-64-10   Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight	97.5	0.09	
Architecture 2: 784-256-128-64-10   Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight	14.97	2.28	
Architecture 2: 784-256-128-64-10   Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight	97.33	0.1	
Architecture 2: 784-256-128-64-10   Input-ReLu(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	46.08	1.94	
+-----+-----+-----+			
- - - - + - - - - + - - - - +			
- - - - + - - - - + - - - - +			

```
In [0]: # http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
```

```
x.field_names = ["Architecture", "Model Name", "Accuracy(%)", "Test Cost"]

x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-AdamOptimizer-SGD Weight", 97.91, 0.1])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-GradientDescentOptimizer-SGD Weight", 11.35, 2.30])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-AdamOptimizer-Relu Weight", 98.05, 0.09])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-GradientDescentOptimizer-Relu Weight", 18.3, 2.26])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-AdamOptimizer-SGD Weight", 98.08, 0.09])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-SGD Weight", 96.98, 0.1])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-AdamOptimizer-AdamOptimizer-Relu Weight", 98.32, 0.08])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-Relu Weight", 96.81, 0.1])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-AdamOptimizer-SGD Weight", 98.07, 0.08])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output))-GradientDescentOptimizer-SGD Weight", 97.24, 0.09])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.68, 0.09])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output))-GradientDescentOptimizer-Relu Weight", 97.24, 0.09])
```

```
N(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(3  
2))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 96.36, 0.12  
])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(512)  
-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-  
Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight", 97.73, 0.12])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(512)  
-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-  
Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 92.87,  
0.34])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(512)  
-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-  
Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.13, 0.15])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(512)  
-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-  
Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 9.82, 2.  
66])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(BN(5  
12))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(outp  
ut)-AdamOptimizer-SGD Weight", 98.23, 0.07])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(BN(5  
12))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(outp  
ut)-GradientDescentOptimizer-SGD Weight", 97.17, 0.09])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(BN(5  
12))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(outp  
ut)-AdamOptimizer-Relu Weight", 98.05, 0.08])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-ReLu(BN(5  
12))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(outp  
ut)-GradientDescentOptimizer-Relu Weight", 96.66, 0.11])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(5  
12)-Dropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmo  
id(32)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight", 88.02, 0.3])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(5  
12)-Dropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmo  
id(32)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight", 1  
1.35, 2.30])  
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(5  
12)-Dropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmo  
id(32)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight", 97.31,
```

```

    0.13])
x.add_row(["Architecture 3: 784-512-256-128-64-32-10", "Input-Sigmoid(5
12)-Dropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmo
id(32)-Dropout-Sigmoid(output)-GradientDescentOptimizer-Relu Weight", 1
1.35, 2.30])
print(x)

```

Architecture	Model Name	Accuracy(%)
Test Cost		
0.13]		
Architecture 3: 784-512-256-128-64-32-10   input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-AdamOptimizer-SGD Weight   97.91		
0.1		
Architecture 3: 784-512-256-128-64-32-10   input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-GradientDescentOptimizer-SGD Weight   11.35		
2.3		
Architecture 3: 784-512-256-128-64-32-10   input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-AdamOptimizer-Relu Weight   98.05		
0.09		
Architecture 3: 784-512-256-128-64-32-10   input-sigmoid(512)-sigmoid(256)-sigmoid(128)-sigmoid(64)-sigmoid(32)-sigmoid(output)-GradientDescentOptimizer-Relu Weight   18.3		
2.26		
Architecture 3: 784-512-256-128-64-32-10   input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-AdamOptimizer-SGD Weight   98.08		
0.09		
Architecture 3: 784-512-256-128-64-32-10   input-ReLu(512)-ReLu(256)-ReLu(128)-ReLu(64)-ReLu(32)-sigmoid(output)-GradientDescentOptimizer-Relu Weight   98.08		

Architecture 3: 784-512-256-128-64-32-10	Input-Relu(512)-ReLU(256)-ReLU(128)-ReLU(64)-ReLU(32)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-SGD Weight	96.98
0.1		
Architecture 3: 784-512-256-128-64-32-10	Input-Relu(512)-ReLU(256)-ReLU(128)-ReLU(64)-ReLU(32)-sigmoid(output)-AdamOptimizer-AdamOptimizer-Relu Weight	98.32
0.08		
Architecture 3: 784-512-256-128-64-32-10	Input-ReLU(512)-ReLU(256)-ReLU(128)-ReLU(64)-ReLU(32)-sigmoid(output)-AdamOptimizer-GradientDescentOptimizer-Relu Weight	96.81
0.1		
Architecture 3: 784-512-256-128-64-32-10	Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-AdamOptimizer-SGD Weight	98.07
0.08		
Architecture 3: 784-512-256-128-64-32-10	Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-GradientDescentOptimizer-SGD Weight	97.24
0.09		
Architecture 3: 784-512-256-128-64-32-10	Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-AdamOptimizer-Relu Weight	97.68
0.09		
Architecture 3: 784-512-256-128-64-32-10	Input-Sigmoid(BN(512))-Sigmoid(BN(256))-Sigmoid(BN(128))-Sigmoid(BN(64))-Sigmoid(BN(32))-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	96.36
0.12		
Architecture 3: 784-512-256-128-64-32-10	Input-ReLu(512)-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight	97.73
0.12		
Architecture 3: 784-512-256-128-64-32-10	Input-ReLu(512)-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-Dropout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight	92.87
0.34		
Architecture 3: 784-512-256-128-64-32-10	Input-ReLu(512)-Dropout-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight	97.13
0.15		

Architecture 3: 784-512-256-128-64-32-10	Input-ReLu(512)-Dropou	
t-ReLu(256)-Dropout-ReLu(128)-Dropout-ReLu(64)-Dropout-ReLu(32)-Dropout		
-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	9.82	
		Input-
Architecture 3: 784-512-256-128-64-32-10	Relu(BN(512))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sig	
	moid(output)-AdamOptimizer-SGD Weight	98.23
		Input-Relu(B
Architecture 3: 784-512-256-128-64-32-10	N(512))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(o	
	utput)-GradientDescentOptimizer-SGD Weight	97.17
		Input-R
Architecture 3: 784-512-256-128-64-32-10	elu(BN(512))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sig	
	moid(output)-AdamOptimizer-Relu Weight	98.05
		Input-Relu(B
Architecture 3: 784-512-256-128-64-32-10	N(512))-Relu(BN(256))-Relu(BN(128))-Relu(BN(64))-Relu(BN(32))-Sigmoid(o	
	utput)-GradientDescentOptimizer-Relu Weight	96.66
		Input-Sigmoid(512)-D
Architecture 3: 784-512-256-128-64-32-10	ropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(3	
	2)-Dropout-Sigmoid(output)-AdamOptimizer-SGD Weight	88.02
		Input-Sigmoid(512)-Dropou
Architecture 3: 784-512-256-128-64-32-10	t-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(32)-Dro	
	pout-Sigmoid(output)-GradientDescentOptimizer-SGD Weight	11.35
		Input-Sigmoid(512)-D
Architecture 3: 784-512-256-128-64-32-10	ropout-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(3	
	2)-Dropout-Sigmoid(output)-AdamOptimizer-Relu Weight	97.31
		Input-Sigmoid(512)-Dropout
Architecture 3: 784-512-256-128-64-32-10	-Sigmoid(256)-Sigmoid(128)-Dropout-Sigmoid(64)-Dropout-Sigmoid(32)-Drop	
	out-Sigmoid(output)-GradientDescentOptimizer-Relu Weight	11.35
+-----+-----+		
- - - - -		
- - - - -		

+-----+