Introduction

MNIST ("Modified National Institute of Standards and Technology") is the de facto “Hello World” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Objective is to correctly identify digits from a dataset of tens of thousands of handwritten images

Approach

For this, we will be using Keras (with TensorFlow as our backend) as the main package to create a simple neural network to predict, as accurately as we can, digits from handwritten images. In particular, we will be calling the Functional Model API of Keras, and creating a 2-layered, 3-layered and 5-layered neural network.

Also, we will be experimenting with various optimizers: the plain vanilla Stochastic Gradient Descent optimizer and the Adam optimizer. However, there are many other parameters, such as training epochs which will we will not be experimenting with.

In addition, the choice of hidden layer units are completely arbitrary and may not be optimal. This is yet another parameter which we will not attempt to tinker with. Lastly, we introduce dropout, a form of regularisation, in our neural networks to prevent overfitting.

Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
print("Printing the version of Tensorflow installed - ", tf.__version__
)
```

```
Printing the version of Tensorflow installed -  1.13.1
```

Importing the dataset

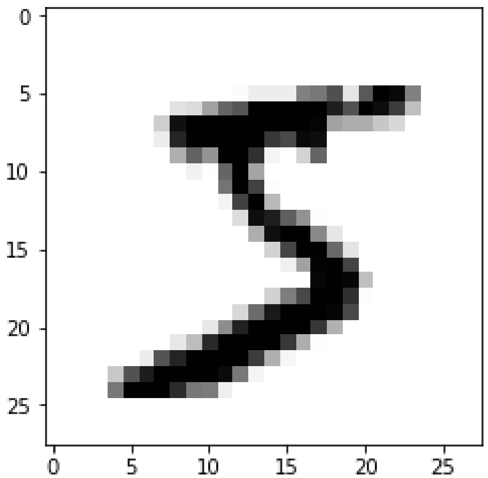
In [2]:
```
mnist_data = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist_data.load_data()

print("Number of training examples :", x_train.shape[0], "and each imag
e 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)
```

Plotting

In [3]:
```
print("Printing the label for the first image :", y_train[0])
plt.imshow(x_train[0], cmap=plt.cm.binary)
plt.show()
```

```
Printing the label for the first image : 5
```



Data preprocessing and Data cleaning

In [4]:
```
x_train.shape
```

Out[4]:
```
(60000, 28, 28)
```

Observation - As you can see, the above dataset contains 3D array with each row containing 28x28 matrix. We will have to change this such that each row will contain 784(28*28) columns.

In [5]:
```
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
])

print("The shape of Training data now becomes : ", x_train.shape)
print("The shape of Testing data now becomes  : ", x_test.shape)
```

```
The shape of Training data now becomes :  (60000, 784)
The shape of Testing data now becomes  :  (10000, 784)
```

In [6]:
```
x_train[0]
```

Out[6]:
```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  18,  18,  18,
       126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
       253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
       253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
       253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   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, 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,  11, 190, 253,  70,
         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,  35,
       241, 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,  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,  45, 186, 253, 253, 150,  27,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  16,  93, 252, 253, 187,
         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, 249,
```

```
        253, 249,  64,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  46, 130,
        183, 253, 253, 207,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39, 148,
        229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114,
        221, 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  23,  66,
        213, 253, 253, 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 171,
        219, 253, 253, 253, 253, 195,  80,   9,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  55, 172,
        226, 253, 253, 253, 253, 244, 133,  11,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        136, 253, 253, 253, 212, 135, 132,  16,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0], dtype=uint8)
```

Observation - We will have to normalize the data before training the MLP model.

There are two approaches for normalization-

1. Divide the entire dataset by 255.0 (Since in rgb 255 is the maximum value)
2. We can use the inbuilt normalization library of tensorflow - utils.normalize()

In [0]:
```
x_train = tf.keras.utils.normalize(x_train)
x_test = tf.keras.utils.normalize(x_test)
```

In [8]:
```
# After normalizing
x_train[0]
```

```
Out[8]: array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00123091, 0.00738549, 0.00738549,
       0.00738549, 0.0516984 , 0.05580145, 0.07180334, 0.01066792,
       0.0681106 , 0.10462772, 0.10134528, 0.05210871, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01230914, 0.01477097, 0.03856865, 0.06318694,
       0.06975182, 0.10380711, 0.10380711, 0.10380711, 0.10380711,
       0.10380711, 0.09231858, 0.07057243, 0.10380711, 0.09929376,
       0.08000944, 0.02625951, 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02010494, 0.09765254,
0.10380711, 0.10380711, 0.10380711, 0.10380711, 0.10380711,
0.10380711, 0.10380711, 0.10380711, 0.1029865 , 0.03815835,
0.03364499, 0.03364499, 0.02297707, 0.01600189, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.00738549, 0.08985675, 0.10380711, 0.10380711,
0.10380711, 0.10380711, 0.10380711, 0.08124035, 0.07467547,
0.10134528, 0.09888346, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.03282438, 0.06400755, 0.04390261, 0.10380711, 0.10380711,
0.08411248, 0.00451335, 0.        , 0.01764311, 0.06318694,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.00574427,
0.0004103 , 0.06318694, 0.10380711, 0.03692743, 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.        , 0.05703237,
0.10380711, 0.07795791, 0.00082061, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.00451335, 0.07795791, 0.10380711,
0.02872134, 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.        , 0.01436067, 0.09888346, 0.09231858, 0.06564877,
0.04431292, 0.0004103 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.03323469, 0.09847315, 0.10380711, 0.10380711, 0.04882627,
0.01025762, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.01846372,
0.07631669, 0.10380711, 0.10380711, 0.06154572, 0.01107823,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.00656488, 0.03815835,
0.10339681, 0.10380711, 0.076727  , 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.        , 0.10216589, 0.10380711,
0.10216589, 0.02625951, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.01887402, 0.05333962,
0.07508578, 0.10380711, 0.10380711, 0.08493309, 0.00082061,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.01600189,
0.06072511, 0.0939598 , 0.10380711, 0.10380711, 0.10380711,
0.1025762 , 0.07467547, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.00984732, 0.04677475, 0.09067736, 0.10380711, 0.10380711,
0.10380711, 0.10380711, 0.08247126, 0.03200377, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.00943701, 0.02708012, 0.08739492, 0.10380711,
0.10380711, 0.10380711, 0.10380711, 0.08124035, 0.03323469,
0.00082061, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.00738549, 0.07016212, 0.08985675,
0.10380711, 0.10380711, 0.10380711, 0.10380711, 0.08000944,
0.03282438, 0.00369274, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.02256676, 0.07057243,
0.09272888, 0.10380711, 0.10380711, 0.10380711, 0.10380711,
0.10011437, 0.05457054, 0.00451335, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.05580145, 0.10380711, 0.10380711, 0.10380711,
0.08698462, 0.05539115, 0.05416023, 0.00656488, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.         , 0.         , 0.         , 0.         , 0.         ,
0.         , 0.         , 0.         , 0.         ])
```

One hot encoding

In [9]:
```
print("Class label for the first image ", y_train[0])

y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

print("Class label after one hot encoding : ", y_train[0])
```

```
Class label for the first image  5
Class label after one hot encoding :  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

MLP Architectures on dataset using Keras

In [0]:
```
# Some paramaters to be predefined
n_epochs = 20
batchsize = 1024

output_dim = 10
input_dim = x_train.shape[1]
```

In [0]:
```
final_output = pd.DataFrame(columns=["#Layers", "Model", "Layer-Archite
cture", "Optimizer", "BN-Present",
                                     "Dropout-Present",
                                     "Train-loss", "Test-loss", "Train-
accuracy", "Test-Accuracy"])
```

In [0]:
```
# 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")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

2 Hidden Layers architecture

2 ReLU hidden Layers (512-128) + ADAM

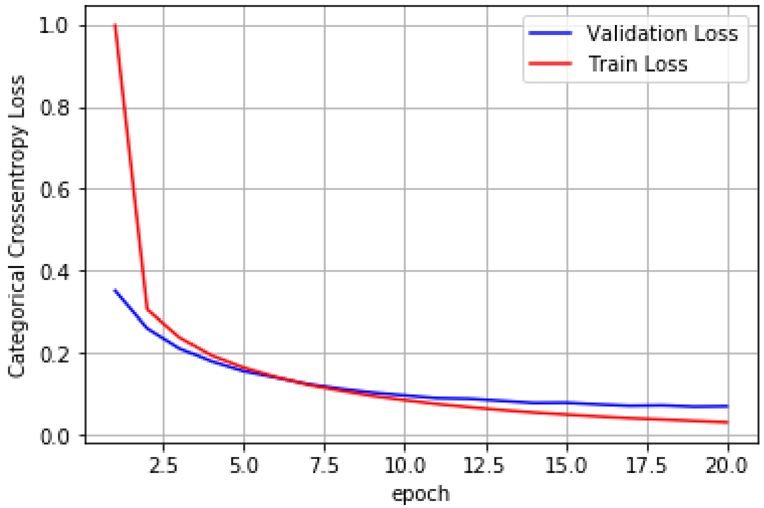
In [13]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("************************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
```

```
> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/ops/resource_variable_ops.py:435: colocate_with (from tensorf
low.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
Train on 60000 samples, validate on 10000 samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.ma
th_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.9974 - acc: 0.7768 - val_loss: 0.3515 - val_acc: 0.9000
Epoch 2/20
```

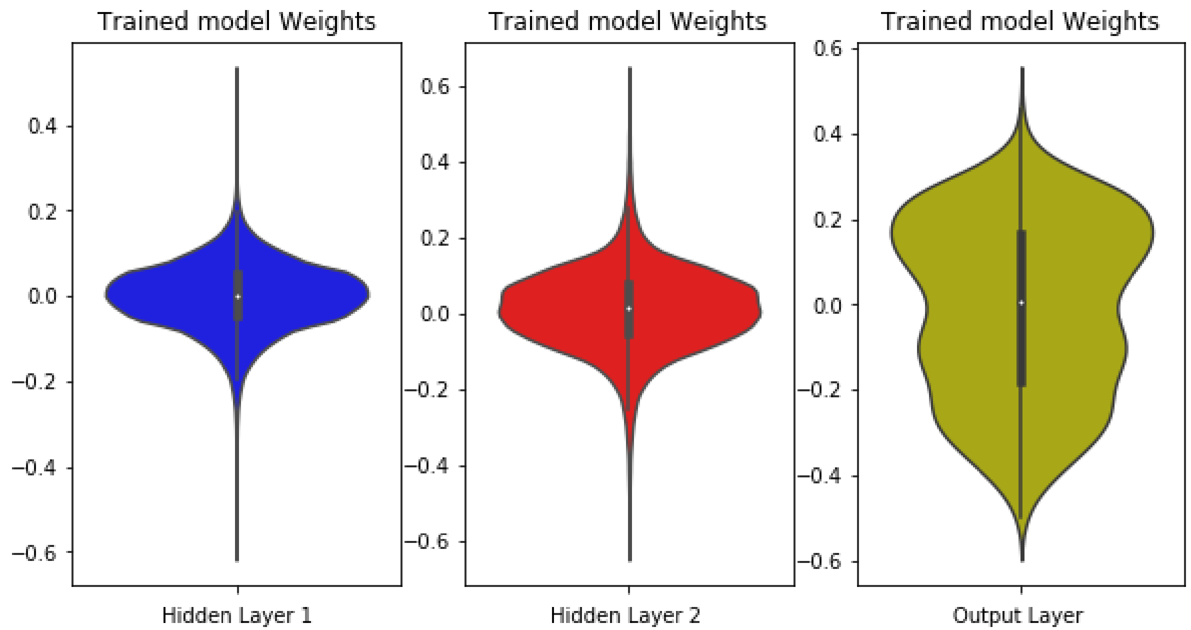
```
60000/60000 [==============================] - 4s 59us/sample - loss:
0.3060 - acc: 0.9112 - val_loss: 0.2601 - val_acc: 0.9253
Epoch 3/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.2380 - acc: 0.9311 - val_loss: 0.2112 - val_acc: 0.9393
Epoch 4/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.1939 - acc: 0.9434 - val_loss: 0.1797 - val_acc: 0.9492
Epoch 5/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.1644 - acc: 0.9522 - val_loss: 0.1556 - val_acc: 0.9533
Epoch 6/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.1415 - acc: 0.9592 - val_loss: 0.1402 - val_acc: 0.9573
Epoch 7/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1224 - acc: 0.9644 - val_loss: 0.1242 - val_acc: 0.9629
Epoch 8/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.1080 - acc: 0.9688 - val_loss: 0.1127 - val_acc: 0.9663
Epoch 9/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0950 - acc: 0.9728 - val_loss: 0.1033 - val_acc: 0.9699
Epoch 10/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0848 - acc: 0.9748 - val_loss: 0.0965 - val_acc: 0.9710
Epoch 11/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0754 - acc: 0.9787 - val_loss: 0.0897 - val_acc: 0.9727
Epoch 12/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0678 - acc: 0.9804 - val_loss: 0.0883 - val_acc: 0.9738
Epoch 13/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0607 - acc: 0.9823 - val_loss: 0.0831 - val_acc: 0.9752
Epoch 14/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0546 - acc: 0.9844 - val_loss: 0.0781 - val_acc: 0.9763
Epoch 15/20
```

```
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0497 - acc: 0.9861 - val_loss: 0.0786 - val_acc: 0.9758
Epoch 16/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0449 - acc: 0.9876 - val_loss: 0.0745 - val_acc: 0.9773
Epoch 17/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0407 - acc: 0.9887 - val_loss: 0.0711 - val_acc: 0.9786
Epoch 18/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0375 - acc: 0.9899 - val_loss: 0.0722 - val_acc: 0.9779
Epoch 19/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0341 - acc: 0.9911 - val_loss: 0.0690 - val_acc: 0.9787
Epoch 20/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0306 - acc: 0.9920 - val_loss: 0.0699 - val_acc: 0.9784
***********************************************
Printing the Model Summary

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 74us/sample - loss:
0.0699 - acc: 0.9784
Test score: 0.06986413442818448
Test accuracy: 0.9784
```



In [14]:
```
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



2 ReLU hidden Layers (256-256) + ADAM

In [15]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(256, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))
```

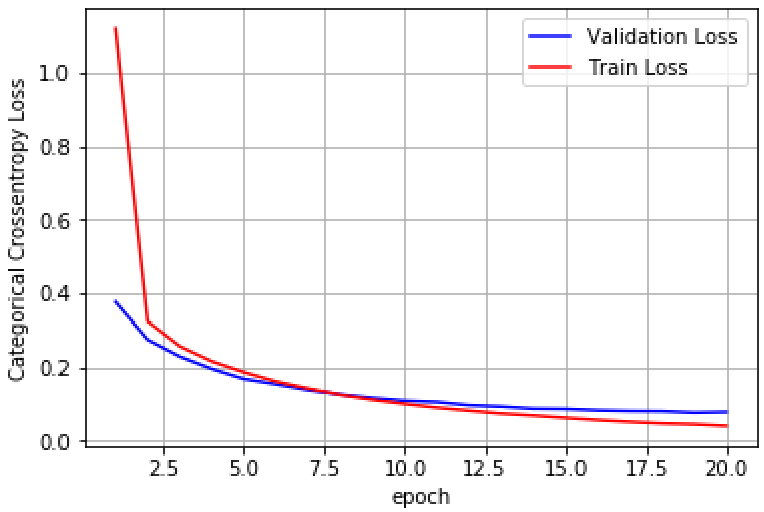
```
print("**********************************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("**********************************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 256 -
> 256 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 45us/sample - loss:
```

```
1.1184 - acc: 0.7268 - val_loss: 0.3768 - val_acc: 0.8929
Epoch 2/20
60000/60000 [==============================] - 3s 42us/sample - loss:
0.3240 - acc: 0.9049 - val_loss: 0.2743 - val_acc: 0.9183
Epoch 3/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.2565 - acc: 0.9250 - val_loss: 0.2287 - val_acc: 0.9324
Epoch 4/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.2157 - acc: 0.9383 - val_loss: 0.1954 - val_acc: 0.9418
Epoch 5/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.1863 - acc: 0.9456 - val_loss: 0.1679 - val_acc: 0.9506
Epoch 6/20
60000/60000 [==============================] - 2s 40us/sample - loss:
0.1610 - acc: 0.9529 - val_loss: 0.1536 - val_acc: 0.9540
Epoch 7/20
60000/60000 [==============================] - 2s 39us/sample - loss:
0.1419 - acc: 0.9582 - val_loss: 0.1375 - val_acc: 0.9579
Epoch 8/20
60000/60000 [==============================] - 2s 39us/sample - loss:
0.1245 - acc: 0.9635 - val_loss: 0.1254 - val_acc: 0.9609
Epoch 9/20
60000/60000 [==============================] - 2s 40us/sample - loss:
0.1110 - acc: 0.9676 - val_loss: 0.1161 - val_acc: 0.9634
Epoch 10/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.1000 - acc: 0.9708 - val_loss: 0.1085 - val_acc: 0.9659
Epoch 11/20
60000/60000 [==============================] - 2s 40us/sample - loss:
0.0897 - acc: 0.9741 - val_loss: 0.1051 - val_acc: 0.9673
Epoch 12/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.0816 - acc: 0.9756 - val_loss: 0.0963 - val_acc: 0.9697
Epoch 13/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.0737 - acc: 0.9786 - val_loss: 0.0928 - val_acc: 0.9710

Epoch 14/20
60000/60000 [==============================] - 2s 41us/sample - loss:
```

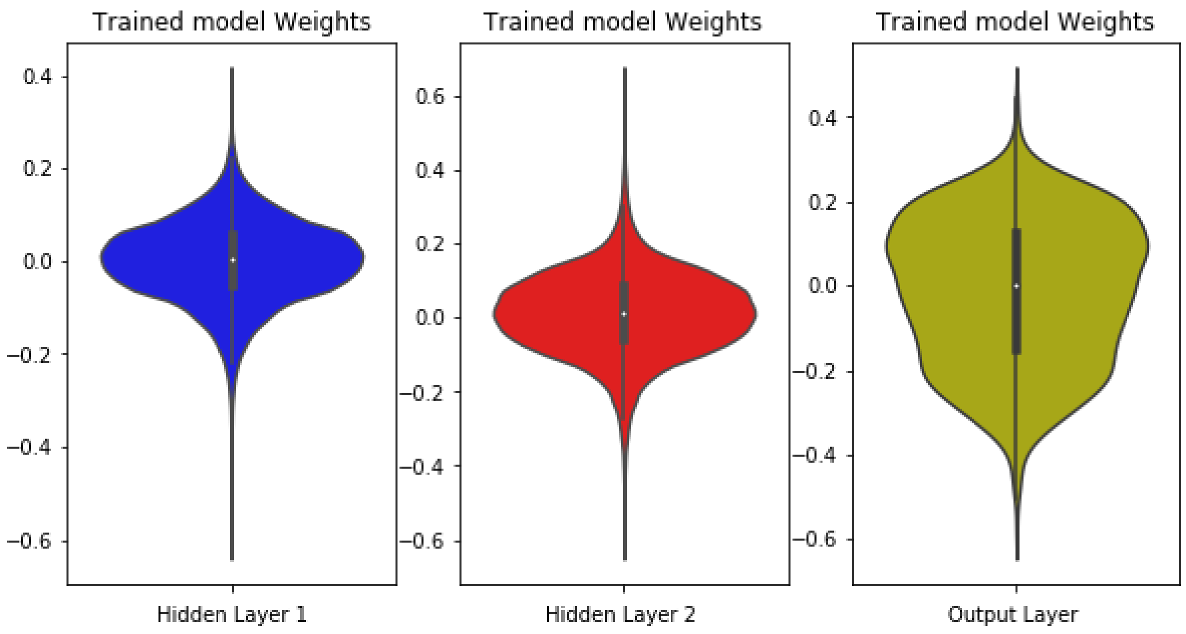
```
0.0684 - acc: 0.9800 - val_loss: 0.0870 - val_acc: 0.9718
Epoch 15/20
60000/60000 [==============================] - 3s 42us/sample - loss:
0.0621 - acc: 0.9819 - val_loss: 0.0862 - val_acc: 0.9734
Epoch 16/20
60000/60000 [==============================] - 3s 42us/sample - loss:
0.0561 - acc: 0.9839 - val_loss: 0.0824 - val_acc: 0.9748
Epoch 17/20
60000/60000 [==============================] - 2s 42us/sample - loss:
0.0512 - acc: 0.9855 - val_loss: 0.0806 - val_acc: 0.9742
Epoch 18/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.0470 - acc: 0.9866 - val_loss: 0.0800 - val_acc: 0.9743
Epoch 19/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.0447 - acc: 0.9874 - val_loss: 0.0762 - val_acc: 0.9759
Epoch 20/20
60000/60000 [==============================] - 2s 41us/sample - loss:
0.0402 - acc: 0.9887 - val_loss: 0.0780 - val_acc: 0.9763
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_5 (Dense)              (None, 10)                2570
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 51us/sample - loss:

0.0780 - acc: 0.9763
Test score: 0.077971170845069
```

```
Test accuracy: 0.9763
```



In [16]:
```python
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



2 ReLU hidden Layers (384-128) + ADAM

In [17]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(384, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))
```

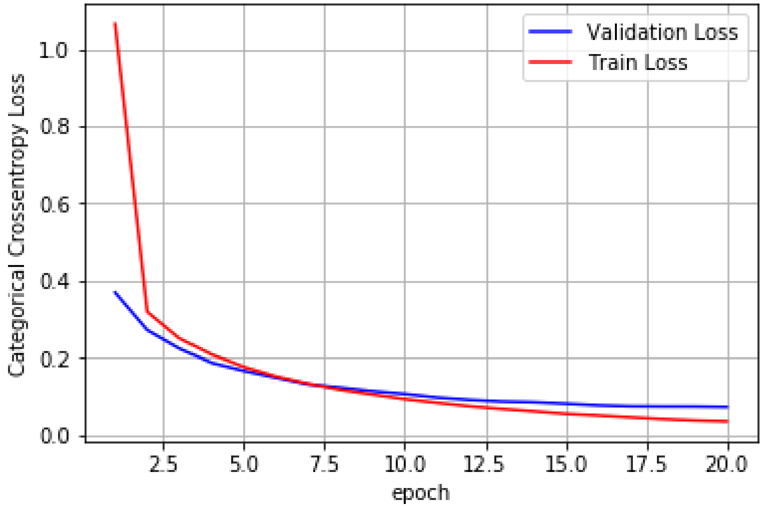
```
print("**************************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("**************************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 384 -
> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

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

```
60000/60000 [==============================] - 3s 49us/sample - loss:
1.0646 - acc: 0.7678 - val_loss: 0.3695 - val_acc: 0.8988
Epoch 2/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.3195 - acc: 0.9073 - val_loss: 0.2722 - val_acc: 0.9203
Epoch 3/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.2506 - acc: 0.9271 - val_loss: 0.2243 - val_acc: 0.9339
Epoch 4/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.2089 - acc: 0.9398 - val_loss: 0.1871 - val_acc: 0.9460
Epoch 5/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.1766 - acc: 0.9483 - val_loss: 0.1664 - val_acc: 0.9527
Epoch 6/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.1520 - acc: 0.9552 - val_loss: 0.1488 - val_acc: 0.9554
Epoch 7/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.1332 - acc: 0.9612 - val_loss: 0.1313 - val_acc: 0.9620
Epoch 8/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.1172 - acc: 0.9660 - val_loss: 0.1229 - val_acc: 0.9636
Epoch 9/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.1051 - acc: 0.9694 - val_loss: 0.1138 - val_acc: 0.9650
Epoch 10/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.0933 - acc: 0.9727 - val_loss: 0.1066 - val_acc: 0.9677
Epoch 11/20
60000/60000 [==============================] - 3s 47us/sample - loss:
0.0838 - acc: 0.9754 - val_loss: 0.0974 - val_acc: 0.9696
Epoch 12/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0748 - acc: 0.9783 - val_loss: 0.0913 - val_acc: 0.9713
Epoch 13/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0678 - acc: 0.9806 - val_loss: 0.0870 - val_acc: 0.9732

Epoch 14/20
```

```
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0618 - acc: 0.9825 - val_loss: 0.0852 - val_acc: 0.9735
Epoch 15/20
60000/60000 [==============================] - 3s 45us/sample - loss:
0.0551 - acc: 0.9846 - val_loss: 0.0815 - val_acc: 0.9745
Epoch 16/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0508 - acc: 0.9857 - val_loss: 0.0771 - val_acc: 0.9759
Epoch 17/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0460 - acc: 0.9874 - val_loss: 0.0747 - val_acc: 0.9760
Epoch 18/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0415 - acc: 0.9890 - val_loss: 0.0741 - val_acc: 0.9786
Epoch 19/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0378 - acc: 0.9899 - val_loss: 0.0740 - val_acc: 0.9769
Epoch 20/20
60000/60000 [==============================] - 3s 46us/sample - loss:
0.0355 - acc: 0.9904 - val_loss: 0.0729 - val_acc: 0.9767
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 384)               301440
_________________________________________________________________
dense_7 (Dense)              (None, 128)               49280
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 352,010
Trainable params: 352,010
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 57us/sample - loss:

0.0729 - acc: 0.9767
```

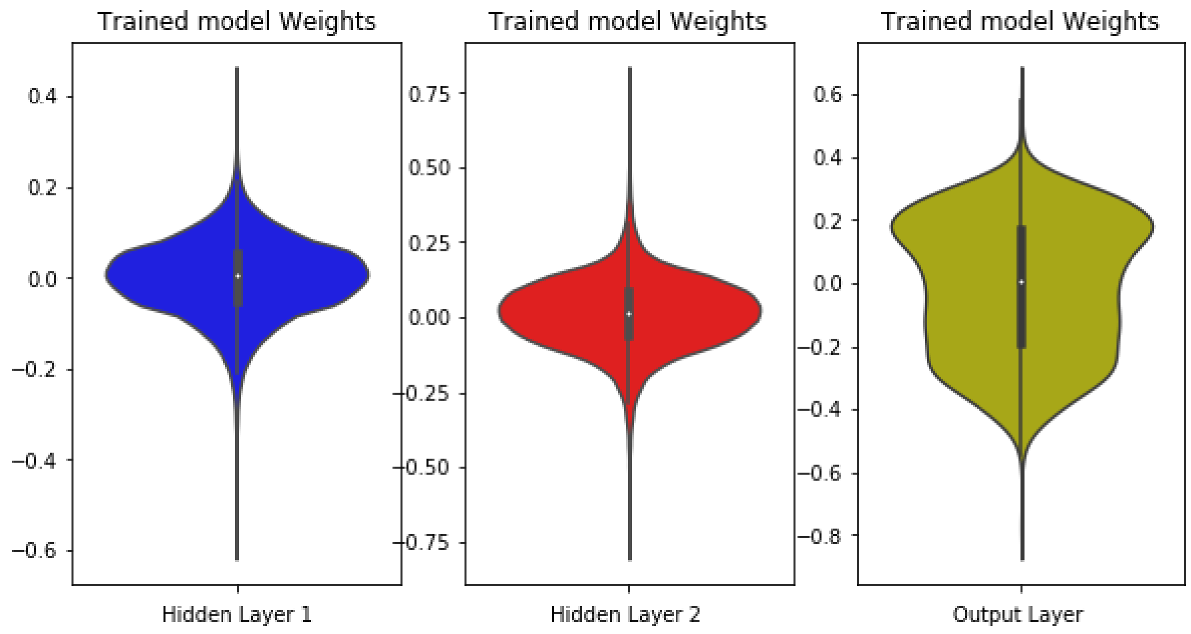
```
Test score: 0.07292742731682957
Test accuracy: 0.9767
```



In [18]:

```
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



2 ReLU hidden Layers (512-128) + BatchNormalization + Dropout + ADAM

In [19]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))
```

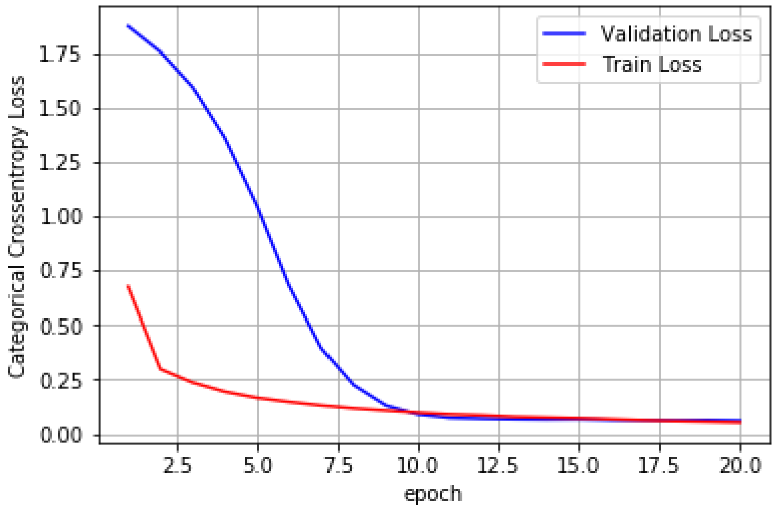
```
relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**********************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("**********************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

```python
vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/keras/layers/core.py:143: calling dropout (from tensorflow.py
thon.ops.nn_ops) with keep_prob is deprecated and will be removed in a
future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 87us/sample - loss:
0.6757 - acc: 0.7988 - val_loss: 1.8734 - val_acc: 0.3371
Epoch 2/20
60000/60000 [==============================] - 5s 75us/sample - loss:
0.2986 - acc: 0.9108 - val_loss: 1.7556 - val_acc: 0.2403
Epoch 3/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.2363 - acc: 0.9294 - val_loss: 1.5945 - val_acc: 0.2704
Epoch 4/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1938 - acc: 0.9416 - val_loss: 1.3605 - val_acc: 0.4257
Epoch 5/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1655 - acc: 0.9500 - val_loss: 1.0498 - val_acc: 0.7379
Epoch 6/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1466 - acc: 0.9566 - val_loss: 0.6820 - val_acc: 0.9064
Epoch 7/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.1305 - acc: 0.9608 - val_loss: 0.3947 - val_acc: 0.9472
Epoch 8/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.1169 - acc: 0.9643 - val_loss: 0.2245 - val_acc: 0.9594
Epoch 9/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.1070 - acc: 0.9673 - val_loss: 0.1303 - val_acc: 0.9742
```

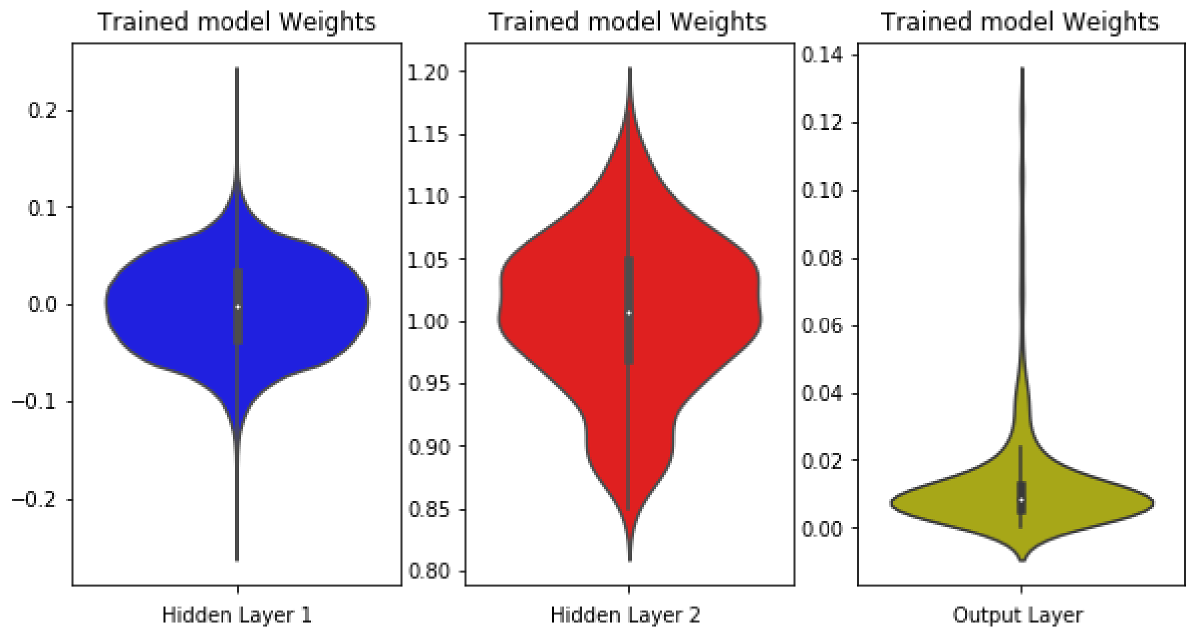
```
Epoch 10/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0971 - acc: 0.9697 - val_loss: 0.0888 - val_acc: 0.9762
Epoch 11/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0890 - acc: 0.9731 - val_loss: 0.0727 - val_acc: 0.9778
Epoch 12/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0843 - acc: 0.9743 - val_loss: 0.0689 - val_acc: 0.9776
Epoch 13/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0774 - acc: 0.9757 - val_loss: 0.0665 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.0741 - acc: 0.9767 - val_loss: 0.0651 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0708 - acc: 0.9776 - val_loss: 0.0658 - val_acc: 0.9805
Epoch 16/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0665 - acc: 0.9790 - val_loss: 0.0623 - val_acc: 0.9817
Epoch 17/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0616 - acc: 0.9800 - val_loss: 0.0615 - val_acc: 0.9811
Epoch 18/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0579 - acc: 0.9816 - val_loss: 0.0604 - val_acc: 0.9807
Epoch 19/20
60000/60000 [==============================] - 5s 75us/sample - loss:
0.0549 - acc: 0.9827 - val_loss: 0.0618 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 5s 75us/sample - loss:
0.0516 - acc: 0.9832 - val_loss: 0.0597 - val_acc: 0.9823
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 512)               401920
```

```
_________________________________________________________________
batch_normalization_v1 (Batc (None, 512)               2048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_v1_1 (Ba (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 72us/sample - loss:
0.0597 - acc: 0.9823
Test score: 0.05972711388359894
Test accuracy: 0.9823
```



In [20]:

```
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



2 ReLU hidden Layers (256-256) + BatchNormalization + Dropout + ADAM

In [21]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(256, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
```

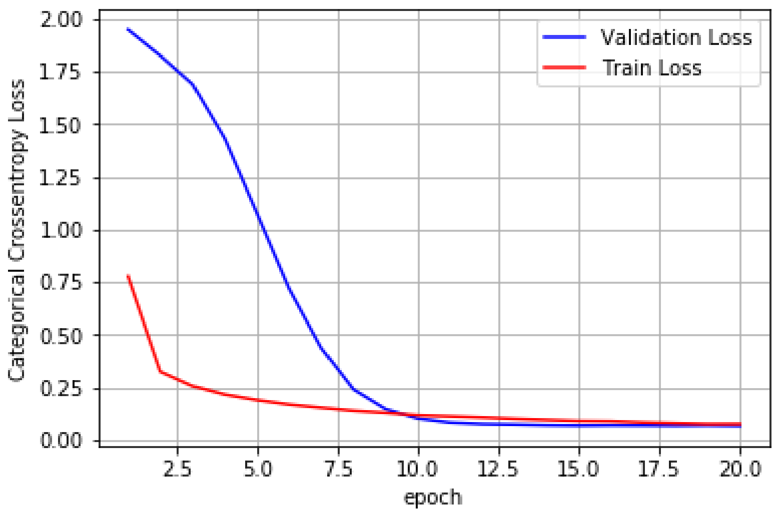
```
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**************************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("**************************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 256 -
> 256 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/20
60000/60000 [==============================] - 4s 68us/sample - loss:
0.7771 - acc: 0.7660 - val_loss: 1.9486 - val_acc: 0.1833
Epoch 2/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.3251 - acc: 0.9003 - val_loss: 1.8251 - val_acc: 0.1206
Epoch 3/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.2563 - acc: 0.9229 - val_loss: 1.6905 - val_acc: 0.1385
Epoch 4/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.2158 - acc: 0.9355 - val_loss: 1.4314 - val_acc: 0.2957
Epoch 5/20
60000/60000 [==============================] - 4s 64us/sample - loss:
0.1894 - acc: 0.9422 - val_loss: 1.0783 - val_acc: 0.6259
Epoch 6/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1683 - acc: 0.9496 - val_loss: 0.7199 - val_acc: 0.8310
Epoch 7/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1531 - acc: 0.9531 - val_loss: 0.4377 - val_acc: 0.9161
Epoch 8/20
60000/60000 [==============================] - 4s 66us/sample - loss:
0.1384 - acc: 0.9571 - val_loss: 0.2402 - val_acc: 0.9458
Epoch 9/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1287 - acc: 0.9596 - val_loss: 0.1465 - val_acc: 0.9665
Epoch 10/20
60000/60000 [==============================] - 4s 64us/sample - loss:
0.1168 - acc: 0.9634 - val_loss: 0.1015 - val_acc: 0.9724
Epoch 11/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.1117 - acc: 0.9648 - val_loss: 0.0820 - val_acc: 0.9751
Epoch 12/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.1063 - acc: 0.9669 - val_loss: 0.0749 - val_acc: 0.9764
Epoch 13/20
60000/60000 [==============================] - 4s 62us/sample - loss:

0.0998 - acc: 0.9680 - val_loss: 0.0724 - val_acc: 0.9772
```

```
Epoch 14/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0946 - acc: 0.9709 - val_loss: 0.0693 - val_acc: 0.9784
Epoch 15/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0894 - acc: 0.9724 - val_loss: 0.0677 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0882 - acc: 0.9728 - val_loss: 0.0688 - val_acc: 0.9784
Epoch 17/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0823 - acc: 0.9732 - val_loss: 0.0683 - val_acc: 0.9783
Epoch 18/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0785 - acc: 0.9745 - val_loss: 0.0672 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0748 - acc: 0.9762 - val_loss: 0.0683 - val_acc: 0.9796
Epoch 20/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0751 - acc: 0.9757 - val_loss: 0.0664 - val_acc: 0.9802
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 256)               200960
_________________________________________________________________
batch_normalization_v1_2 (Ba (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               65792
_________________________________________________________________
batch_normalization_v1_3 (Ba (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                2570
```

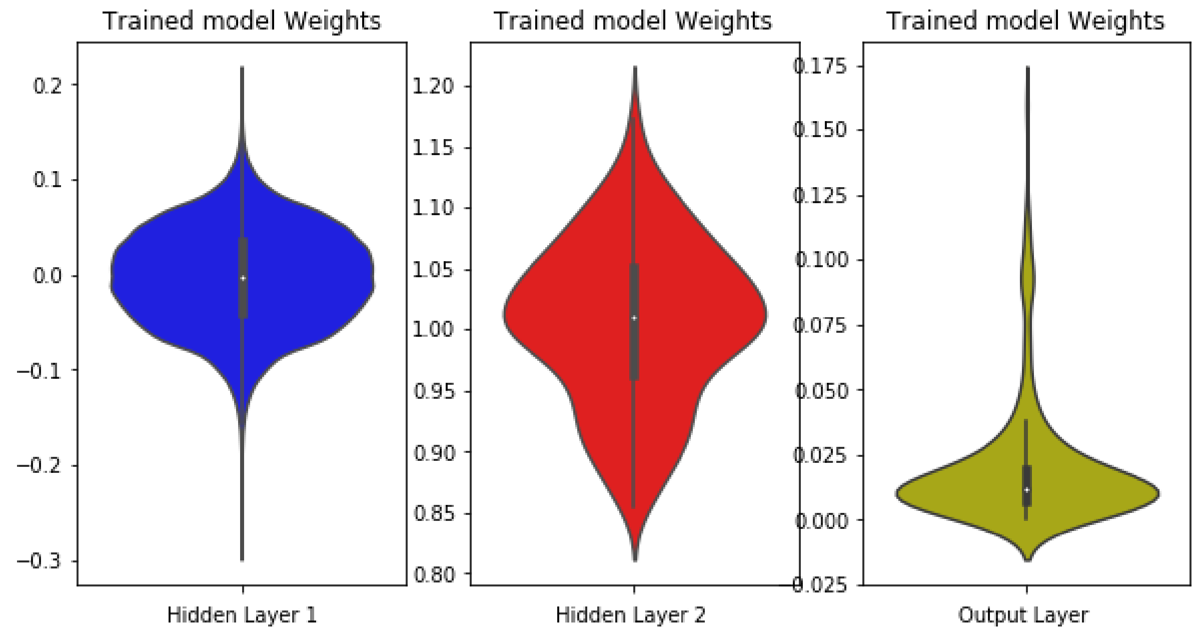
```
=================================================================
Total params: 271,370
Trainable params: 270,346
Non-trainable params: 1,024
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 69us/sample - loss:
0.0664 - acc: 0.9802
Test score: 0.06643414922667434
Test accuracy: 0.9802
```



In [22]:

```python
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



2 ReLU hidden Layers (384-128) + BatchNormalization + Dropout + ADAM

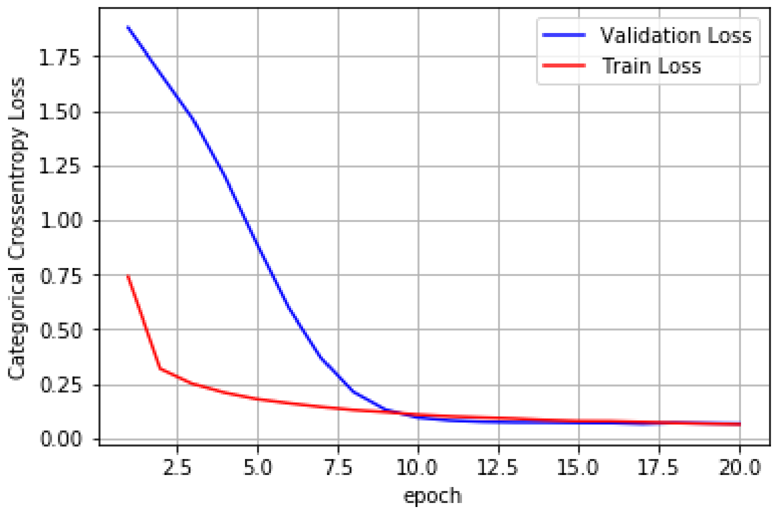
In [23]:
```
relumodel_2 = tf.keras.models.Sequential()
relumodel_2.add(tf.keras.layers.Dense(384, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_2.add(tf.keras.layers.BatchNormalization())
relumodel_2.add(tf.keras.layers.Dropout(0.5))
relumodel_2.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_2.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**************************************************")
print("Printing the Model Summary")
print(relumodel_2.summary())
print("**************************************************")

score = relumodel_2.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 2,
                                    "Model": "2-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 384 -
> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
```

```
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 75us/sample - loss:
0.7401 - acc: 0.7775 - val_loss: 1.8800 - val_acc: 0.5990
Epoch 2/20
60000/60000 [==============================] - 4s 67us/sample - loss:
0.3188 - acc: 0.9033 - val_loss: 1.6718 - val_acc: 0.6008
Epoch 3/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.2496 - acc: 0.9271 - val_loss: 1.4658 - val_acc: 0.7006
Epoch 4/20
60000/60000 [==============================] - 4s 69us/sample - loss:
0.2086 - acc: 0.9377 - val_loss: 1.2000 - val_acc: 0.8156
Epoch 5/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1793 - acc: 0.9449 - val_loss: 0.8920 - val_acc: 0.9021
Epoch 6/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1600 - acc: 0.9522 - val_loss: 0.5971 - val_acc: 0.9418
Epoch 7/20
60000/60000 [==============================] - 4s 66us/sample - loss:
0.1432 - acc: 0.9571 - val_loss: 0.3685 - val_acc: 0.9577
Epoch 8/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.1290 - acc: 0.9616 - val_loss: 0.2124 - val_acc: 0.9699
```

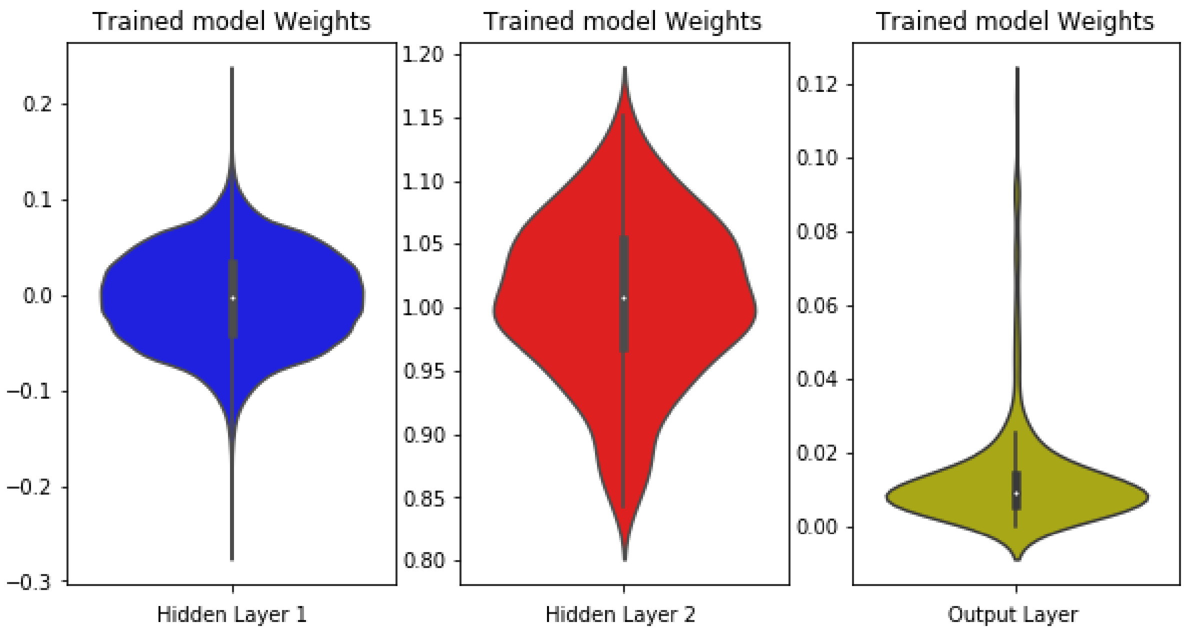
```
Epoch 9/20
60000/60000 [==============================] - 4s 64us/sample - loss:
0.1195 - acc: 0.9630 - val_loss: 0.1309 - val_acc: 0.9731
Epoch 10/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.1077 - acc: 0.9676 - val_loss: 0.0941 - val_acc: 0.9756
Epoch 11/20
60000/60000 [==============================] - 4s 65us/sample - loss:
0.0997 - acc: 0.9694 - val_loss: 0.0809 - val_acc: 0.9760
Epoch 12/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0945 - acc: 0.9711 - val_loss: 0.0747 - val_acc: 0.9770
Epoch 13/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0897 - acc: 0.9724 - val_loss: 0.0721 - val_acc: 0.9768
Epoch 14/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0828 - acc: 0.9742 - val_loss: 0.0715 - val_acc: 0.9775
Epoch 15/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0782 - acc: 0.9751 - val_loss: 0.0701 - val_acc: 0.9796
Epoch 16/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0775 - acc: 0.9755 - val_loss: 0.0692 - val_acc: 0.9788
Epoch 17/20
60000/60000 [==============================] - 4s 64us/sample - loss:
0.0735 - acc: 0.9767 - val_loss: 0.0652 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0693 - acc: 0.9782 - val_loss: 0.0698 - val_acc: 0.9793
Epoch 19/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0661 - acc: 0.9789 - val_loss: 0.0679 - val_acc: 0.9792
Epoch 20/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0631 - acc: 0.9801 - val_loss: 0.0662 - val_acc: 0.9803
***********************************************
Printing the Model Summary
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 384)               301440
_________________________________________________________________
batch_normalization_v1_4 (Ba (None, 384)               1536
_________________________________________________________________
dropout_4 (Dropout)          (None, 384)               0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               49280
_________________________________________________________________
batch_normalization_v1_5 (Ba (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290
=================================================================
Total params: 354,058
Trainable params: 353,034
Non-trainable params: 1,024
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 69us/sample - loss:
0.0662 - acc: 0.9803
Test score: 0.06616050415177015
Test accuracy: 0.9803
```



In [24]:
```
w_after = relumodel_2.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(figsize=(10, 5))
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()
```



3 Hidden Layers architecture

In [0]:
```
final_output = final_output.append({"#Layers": "--",
                                    "Model": "--",
                                    "Layer-Architecture": "--",
                                    "Optimizer": "--", "BN-Present": "-
-",
                                    "Dropout-Present": "--",
                                    "Train-loss": "--",
                                    "Test-loss": "--",
                                    "Train-accuracy": "--",
                                    "Test-Accuracy": "--"}, ignore_inde
x=True)
```

3 ReLU hidden Layers (512-256-128) + ADAM

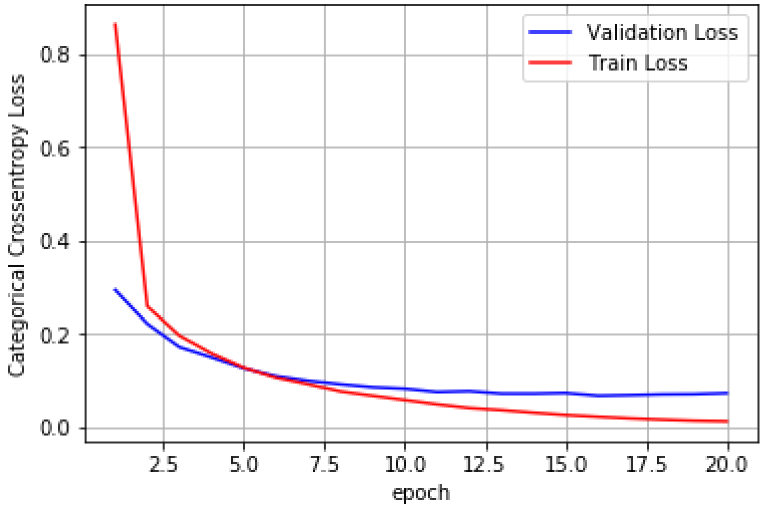
In [26]:
```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_3.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_3.summary())
print("************************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 256 -> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
```

```
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.8634 - acc: 0.7693 - val_loss: 0.2947 - val_acc: 0.9133
Epoch 2/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.2597 - acc: 0.9237 - val_loss: 0.2218 - val_acc: 0.9331
Epoch 3/20
60000/60000 [==============================] - 4s 72us/sample - loss:
0.1966 - acc: 0.9419 - val_loss: 0.1723 - val_acc: 0.9481
Epoch 4/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.1584 - acc: 0.9532 - val_loss: 0.1499 - val_acc: 0.9547
Epoch 5/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.1279 - acc: 0.9622 - val_loss: 0.1265 - val_acc: 0.9615
Epoch 6/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.1064 - acc: 0.9686 - val_loss: 0.1099 - val_acc: 0.9656
Epoch 7/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0922 - acc: 0.9725 - val_loss: 0.0996 - val_acc: 0.9692
Epoch 8/20
60000/60000 [==============================] - 4s 73us/sample - loss:

0.0765 - acc: 0.9771 - val_loss: 0.0919 - val_acc: 0.9717
Epoch 9/20
```

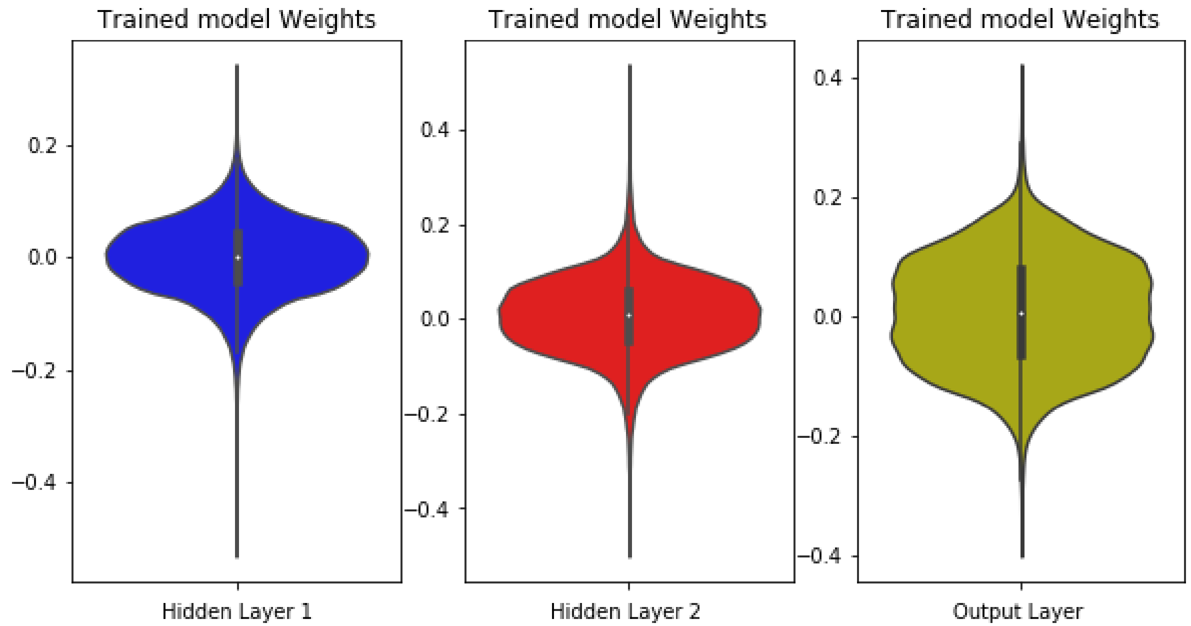
```
Epoch 9/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0672 - acc: 0.9800 - val_loss: 0.0857 - val_acc: 0.9726
Epoch 10/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0580 - acc: 0.9824 - val_loss: 0.0825 - val_acc: 0.9735
Epoch 11/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0489 - acc: 0.9856 - val_loss: 0.0755 - val_acc: 0.9758
Epoch 12/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0410 - acc: 0.9885 - val_loss: 0.0772 - val_acc: 0.9752
Epoch 13/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0365 - acc: 0.9892 - val_loss: 0.0720 - val_acc: 0.9774
Epoch 14/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0309 - acc: 0.9913 - val_loss: 0.0719 - val_acc: 0.9787
Epoch 15/20
60000/60000 [==============================] - 4s 72us/sample - loss:
0.0260 - acc: 0.9930 - val_loss: 0.0733 - val_acc: 0.9773
Epoch 16/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0221 - acc: 0.9942 - val_loss: 0.0674 - val_acc: 0.9797
Epoch 17/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0187 - acc: 0.9954 - val_loss: 0.0687 - val_acc: 0.9788
Epoch 18/20
60000/60000 [==============================] - 4s 72us/sample - loss:
0.0160 - acc: 0.9960 - val_loss: 0.0704 - val_acc: 0.9783
Epoch 19/20
60000/60000 [==============================] - 4s 72us/sample - loss:
0.0137 - acc: 0.9967 - val_loss: 0.0708 - val_acc: 0.9786
Epoch 20/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0124 - acc: 0.9971 - val_loss: 0.0729 - val_acc: 0.9782
***********************************************
Printing the Model Summary


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

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_19 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 89us/sample - loss:
0.0729 - acc: 0.9782
Test score: 0.07288095771700609
Test accuracy: 0.9782
```



In [27]:
```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



3 ReLU hidden Layers (512-128-64) + ADAM

In [28]:
```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
```

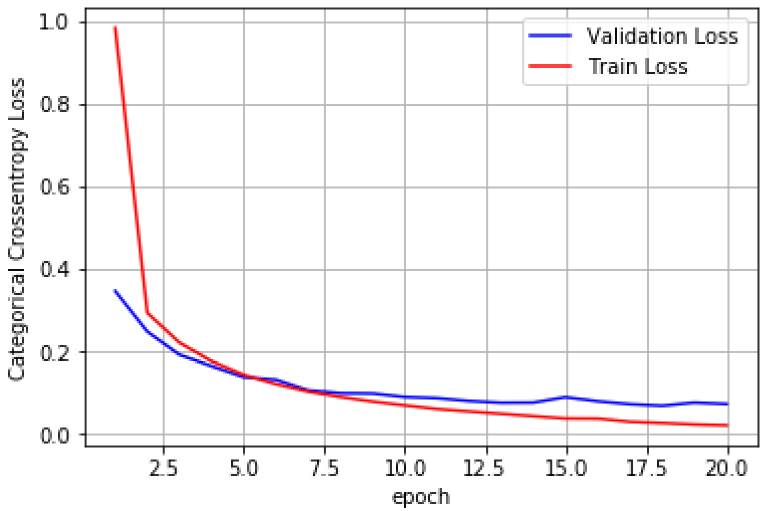
```
print("Printing the Model Summary")
print(relumodel_3.summary())
print("***********************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 128 -> 64 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 68us/sample - loss:
0.9820 - acc: 0.7609 - val_loss: 0.3465 - val_acc: 0.8991
```

```
Epoch 2/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.2937 - acc: 0.9139 - val_loss: 0.2482 - val_acc: 0.9270
Epoch 3/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.2216 - acc: 0.9358 - val_loss: 0.1922 - val_acc: 0.9430
Epoch 4/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1764 - acc: 0.9481 - val_loss: 0.1647 - val_acc: 0.9502
Epoch 5/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.1438 - acc: 0.9576 - val_loss: 0.1383 - val_acc: 0.9591
Epoch 6/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1215 - acc: 0.9636 - val_loss: 0.1322 - val_acc: 0.9607
Epoch 7/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1036 - acc: 0.9696 - val_loss: 0.1066 - val_acc: 0.9683
Epoch 8/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0900 - acc: 0.9732 - val_loss: 0.0995 - val_acc: 0.9696
Epoch 9/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0790 - acc: 0.9765 - val_loss: 0.0988 - val_acc: 0.9698
Epoch 10/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0699 - acc: 0.9793 - val_loss: 0.0903 - val_acc: 0.9729
Epoch 11/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0611 - acc: 0.9824 - val_loss: 0.0875 - val_acc: 0.9728
Epoch 12/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0551 - acc: 0.9837 - val_loss: 0.0805 - val_acc: 0.9757
Epoch 13/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0494 - acc: 0.9860 - val_loss: 0.0763 - val_acc: 0.9776
Epoch 14/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0437 - acc: 0.9875 - val_loss: 0.0767 - val_acc: 0.9778
```

```
Epoch 15/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0381 - acc: 0.9895 - val_loss: 0.0899 - val_acc: 0.9733
Epoch 16/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0378 - acc: 0.9888 - val_loss: 0.0797 - val_acc: 0.9772
Epoch 17/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0300 - acc: 0.9918 - val_loss: 0.0728 - val_acc: 0.9790
Epoch 18/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0270 - acc: 0.9929 - val_loss: 0.0689 - val_acc: 0.9802
Epoch 19/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0235 - acc: 0.9939 - val_loss: 0.0766 - val_acc: 0.9780
Epoch 20/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0215 - acc: 0.9948 - val_loss: 0.0732 - val_acc: 0.9800
**********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_23 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_24 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_25 (Dense)             (None, 10)                650
=================================================================
Total params: 476,490
Trainable params: 476,490
Non-trainable params: 0
_________________________________________________________________
None
**********************************************
10000/10000 [==============================] - 1s 69us/sample - loss:
0.0732 - acc: 0.9800
```

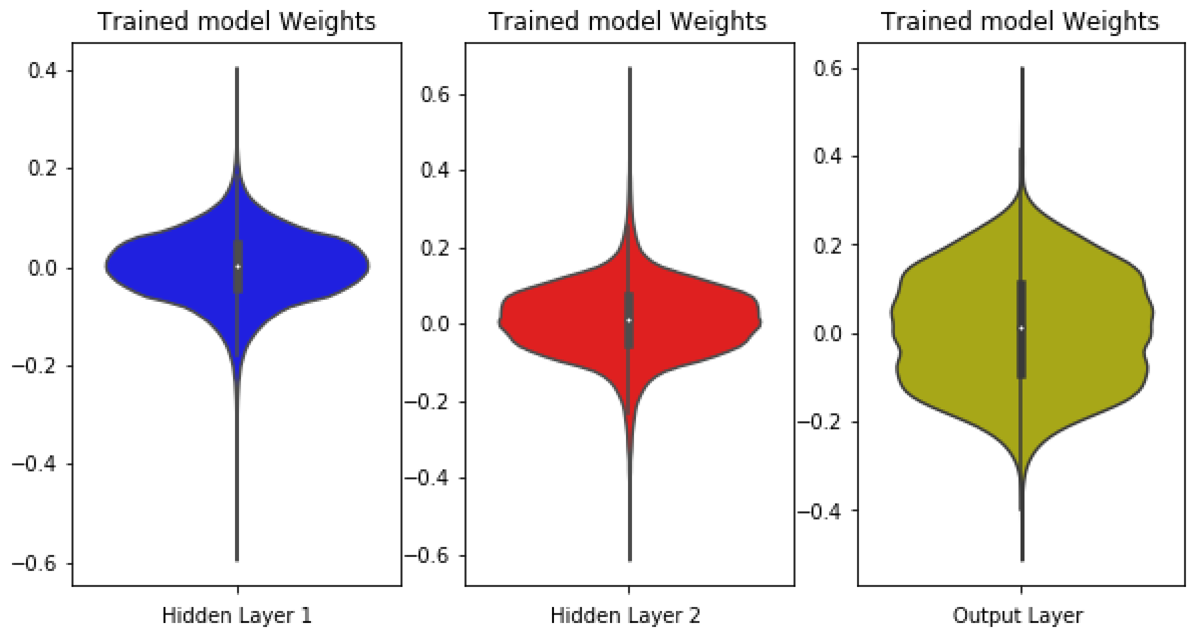
```
Test score: 0.07322663756180554
Test accuracy: 0.98
```



In [29]:

```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



3 ReLU hidden Layers (384-256-128) + ADAM

In [30]:

```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(384, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_3.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

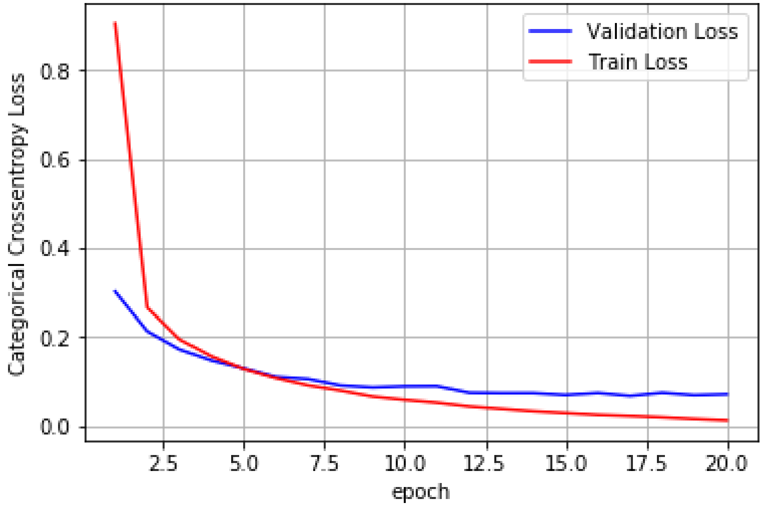
```
model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**************************************************")
print("Printing the Model Summary")
print(relumodel_3.summary())
print("**************************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 384 -
> 256 -> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
```

```
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 66us/sample - loss:
0.9044 - acc: 0.7855 - val_loss: 0.3027 - val_acc: 0.9101
Epoch 2/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2669 - acc: 0.9223 - val_loss: 0.2134 - val_acc: 0.9357
Epoch 3/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.1952 - acc: 0.9424 - val_loss: 0.1727 - val_acc: 0.9489
Epoch 4/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.1569 - acc: 0.9532 - val_loss: 0.1476 - val_acc: 0.9549
Epoch 5/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.1287 - acc: 0.9616 - val_loss: 0.1300 - val_acc: 0.9606
Epoch 6/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.1081 - acc: 0.9675 - val_loss: 0.1110 - val_acc: 0.9661
Epoch 7/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.0918 - acc: 0.9719 - val_loss: 0.1061 - val_acc: 0.9662
Epoch 8/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0800 - acc: 0.9759 - val_loss: 0.0916 - val_acc: 0.9708
Epoch 9/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.0666 - acc: 0.9803 - val_loss: 0.0872 - val_acc: 0.9723
Epoch 10/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0592 - acc: 0.9823 - val_loss: 0.0897 - val_acc: 0.9716
Epoch 11/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0528 - acc: 0.9844 - val_loss: 0.0900 - val_acc: 0.9711
Epoch 12/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0442 - acc: 0.9875 - val_loss: 0.0752 - val_acc: 0.9757
```

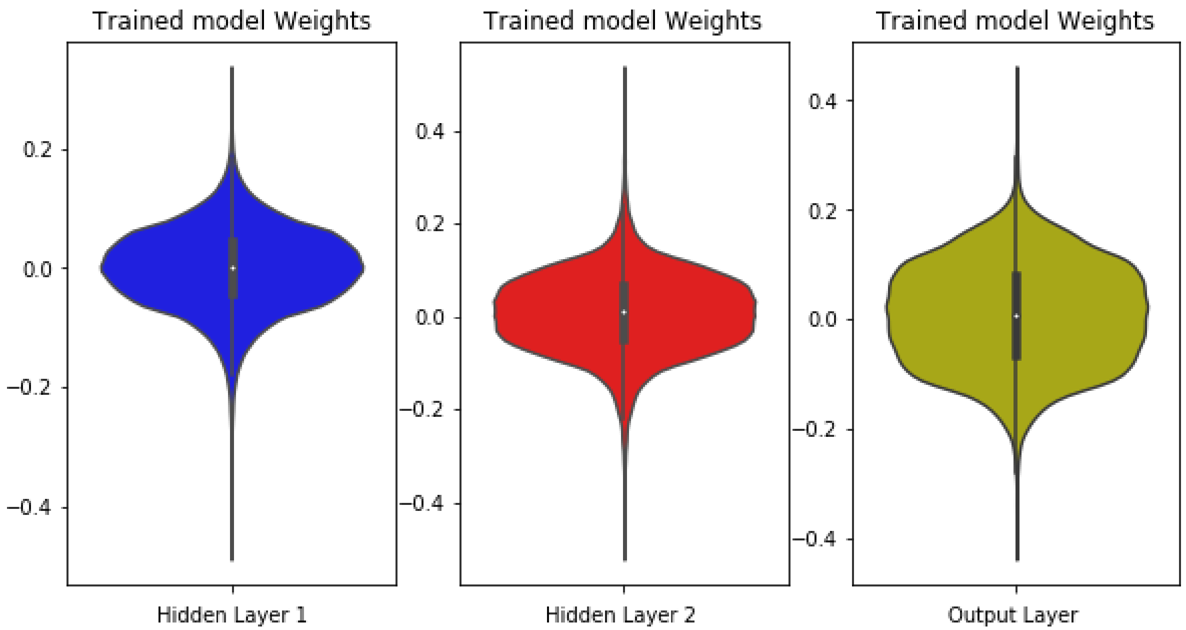
```
Epoch 13/20

60000/60000 [==============================] - 4s 60us/sample - loss:
0.0387 - acc: 0.9890 - val_loss: 0.0743 - val_acc: 0.9764
Epoch 14/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0334 - acc: 0.9907 - val_loss: 0.0743 - val_acc: 0.9772
Epoch 15/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.0293 - acc: 0.9916 - val_loss: 0.0702 - val_acc: 0.9795
Epoch 16/20
60000/60000 [==============================] - 4s 59us/sample - loss:
0.0253 - acc: 0.9935 - val_loss: 0.0746 - val_acc: 0.9771
Epoch 17/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0228 - acc: 0.9940 - val_loss: 0.0681 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 4s 69us/sample - loss:
0.0198 - acc: 0.9948 - val_loss: 0.0751 - val_acc: 0.9770
Epoch 19/20
60000/60000 [==============================] - 4s 69us/sample - loss:
0.0161 - acc: 0.9961 - val_loss: 0.0697 - val_acc: 0.9787
Epoch 20/20
60000/60000 [==============================] - 4s 58us/sample - loss:
0.0130 - acc: 0.9971 - val_loss: 0.0717 - val_acc: 0.9786
***********************************************
Printing the Model Summary

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 384)               301440
_________________________________________________________________
dense_27 (Dense)             (None, 256)               98560
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_29 (Dense)             (None, 10)                1290
=================================================================
Total params: 434,186
Trainable params: 434,186
Non-trainable params: 0
```

```
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 69us/sample - loss:
0.0717 - acc: 0.9786
Test score: 0.07172299538475926
Test accuracy: 0.9786
```



In [31]:
```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



3 ReLU hidden Layers (512-256-128) + BatchNormalization + Dropout + ADAM

In [32]:
```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
```

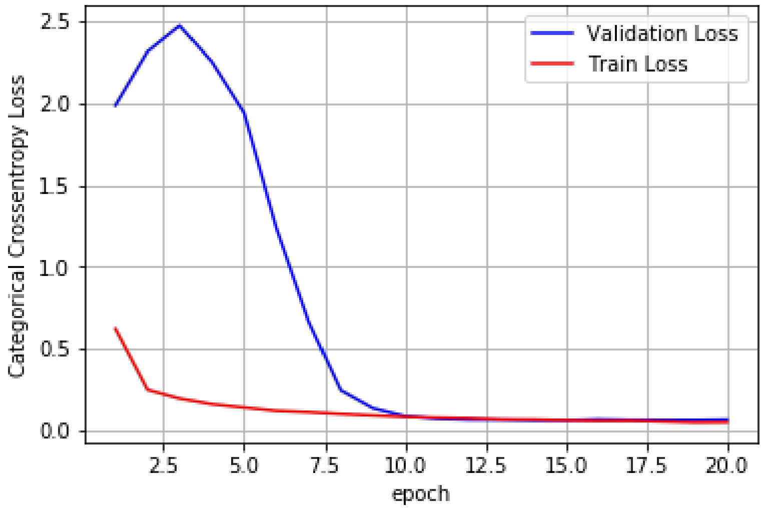
```
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("***********************************************")
print("Printing the Model Summary")
print(relumodel_3.summary())
print("***********************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 256 -> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
```

```
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/sample - loss:
0.6203 - acc: 0.8054 - val_loss: 1.9858 - val_acc: 0.0978
Epoch 2/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.2467 - acc: 0.9247 - val_loss: 2.3146 - val_acc: 0.0974
Epoch 3/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.1933 - acc: 0.9406 - val_loss: 2.4727 - val_acc: 0.0979
Epoch 4/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.1586 - acc: 0.9510 - val_loss: 2.2506 - val_acc: 0.1620
Epoch 5/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.1383 - acc: 0.9575 - val_loss: 1.9394 - val_acc: 0.1974
Epoch 6/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.1181 - acc: 0.9625 - val_loss: 1.2399 - val_acc: 0.4169
Epoch 7/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.1099 - acc: 0.9666 - val_loss: 0.6685 - val_acc: 0.7272
Epoch 8/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.0987 - acc: 0.9695 - val_loss: 0.2438 - val_acc: 0.9250
Epoch 9/20
60000/60000 [==============================] - 6s 95us/sample - loss:
```

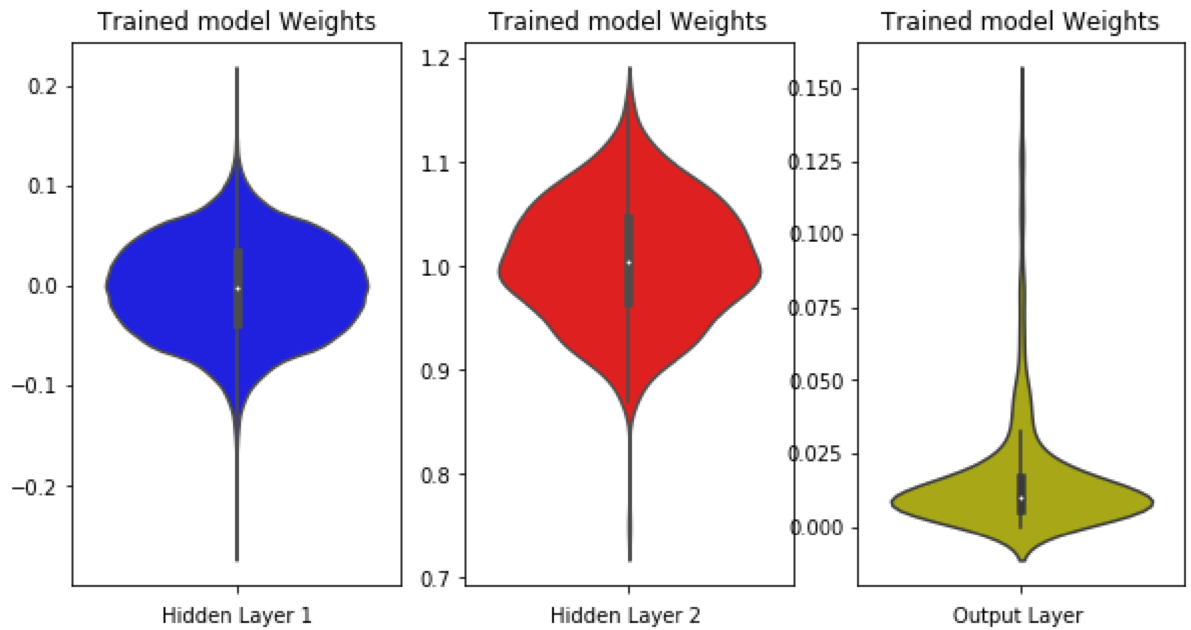
```
0.0898 - acc: 0.9711 - val_loss: 0.1340 - val_acc: 0.9588
Epoch 10/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0809 - acc: 0.9738 - val_loss: 0.0859 - val_acc: 0.9711
Epoch 11/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.0764 - acc: 0.9759 - val_loss: 0.0698 - val_acc: 0.9781
Epoch 12/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.0725 - acc: 0.9766 - val_loss: 0.0635 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.0651 - acc: 0.9787 - val_loss: 0.0630 - val_acc: 0.9801
Epoch 14/20
60000/60000 [==============================] - 6s 96us/sample - loss:
0.0643 - acc: 0.9789 - val_loss: 0.0594 - val_acc: 0.9814
Epoch 15/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0600 - acc: 0.9807 - val_loss: 0.0601 - val_acc: 0.9818
Epoch 16/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0564 - acc: 0.9817 - val_loss: 0.0655 - val_acc: 0.9802
Epoch 17/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0573 - acc: 0.9810 - val_loss: 0.0623 - val_acc: 0.9808
Epoch 18/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0526 - acc: 0.9829 - val_loss: 0.0615 - val_acc: 0.9819
Epoch 19/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0471 - acc: 0.9841 - val_loss: 0.0613 - val_acc: 0.9811
Epoch 20/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0478 - acc: 0.9845 - val_loss: 0.0640 - val_acc: 0.9819
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_30 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_v1_6 (Ba (None, 512)               2048
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_31 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_v1_7 (Ba (None, 256)               1024
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_32 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_33 (Dense)             (None, 10)                1290
=================================================================
Total params: 570,506
Trainable params: 568,970
Non-trainable params: 1,536
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 106us/sample - loss:
0.0640 - acc: 0.9819
Test score: 0.0640360280782188
Test accuracy: 0.9819
```



In [33]:
```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



3 ReLU hidden Layers (512-128-64) + BatchNormalization + Dropout + ADAM

In [34]:

```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))
```

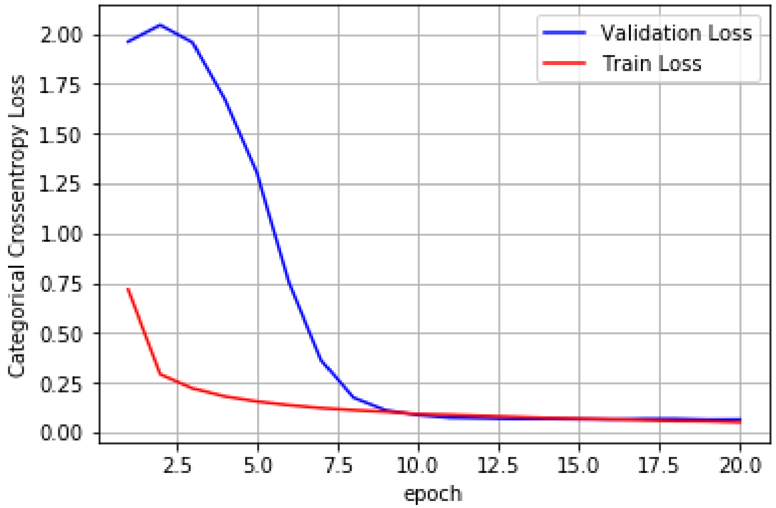
```
relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_3.summary())
print("************************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 128 -> 64 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

```
vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 92us/sample - loss:
0.7178 - acc: 0.7768 - val_loss: 1.9599 - val_acc: 0.1297
Epoch 2/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.2933 - acc: 0.9117 - val_loss: 2.0435 - val_acc: 0.0982
Epoch 3/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.2226 - acc: 0.9337 - val_loss: 1.9565 - val_acc: 0.0989
Epoch 4/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.1820 - acc: 0.9456 - val_loss: 1.6727 - val_acc: 0.1564
Epoch 5/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.1566 - acc: 0.9530 - val_loss: 1.3032 - val_acc: 0.3902
Epoch 6/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.1382 - acc: 0.9586 - val_loss: 0.7529 - val_acc: 0.7091
Epoch 7/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.1233 - acc: 0.9630 - val_loss: 0.3635 - val_acc: 0.9012
Epoch 8/20
60000/60000 [==============================] - 5s 79us/sample - loss:
0.1130 - acc: 0.9647 - val_loss: 0.1758 - val_acc: 0.9527
Epoch 9/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.1048 - acc: 0.9676 - val_loss: 0.1125 - val_acc: 0.9683
Epoch 10/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.0934 - acc: 0.9713 - val_loss: 0.0882 - val_acc: 0.9721
Epoch 11/20
60000/60000 [==============================] - 5s 78us/sample - loss:
0.0895 - acc: 0.9722 - val_loss: 0.0748 - val_acc: 0.9771
Epoch 12/20
60000/60000 [==============================] - 5s 77us/sample - loss:
```

```
0.0841 - acc: 0.9740 - val_loss: 0.0722 - val_acc: 0.9784
Epoch 13/20
60000/60000 [==============================] - 5s 78us/sample - loss: 
0.0791 - acc: 0.9752 - val_loss: 0.0687 - val_acc: 0.9795
Epoch 14/20
60000/60000 [==============================] - 5s 78us/sample - loss: 
0.0737 - acc: 0.9768 - val_loss: 0.0689 - val_acc: 0.9784
Epoch 15/20
60000/60000 [==============================] - 5s 79us/sample - loss: 
0.0701 - acc: 0.9780 - val_loss: 0.0672 - val_acc: 0.9803
Epoch 16/20
60000/60000 [==============================] - 5s 80us/sample - loss: 
0.0672 - acc: 0.9791 - val_loss: 0.0656 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 5s 80us/sample - loss: 
0.0621 - acc: 0.9801 - val_loss: 0.0690 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 5s 79us/sample - loss: 
0.0582 - acc: 0.9815 - val_loss: 0.0692 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 5s 80us/sample - loss: 
0.0567 - acc: 0.9818 - val_loss: 0.0654 - val_acc: 0.9813
Epoch 20/20
60000/60000 [==============================] - 5s 79us/sample - loss: 
0.0520 - acc: 0.9833 - val_loss: 0.0659 - val_acc: 0.9812
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_34 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_v1_8 (Ba (None, 512)               2048      
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_v1_9 (Ba (None, 128)               512       
```

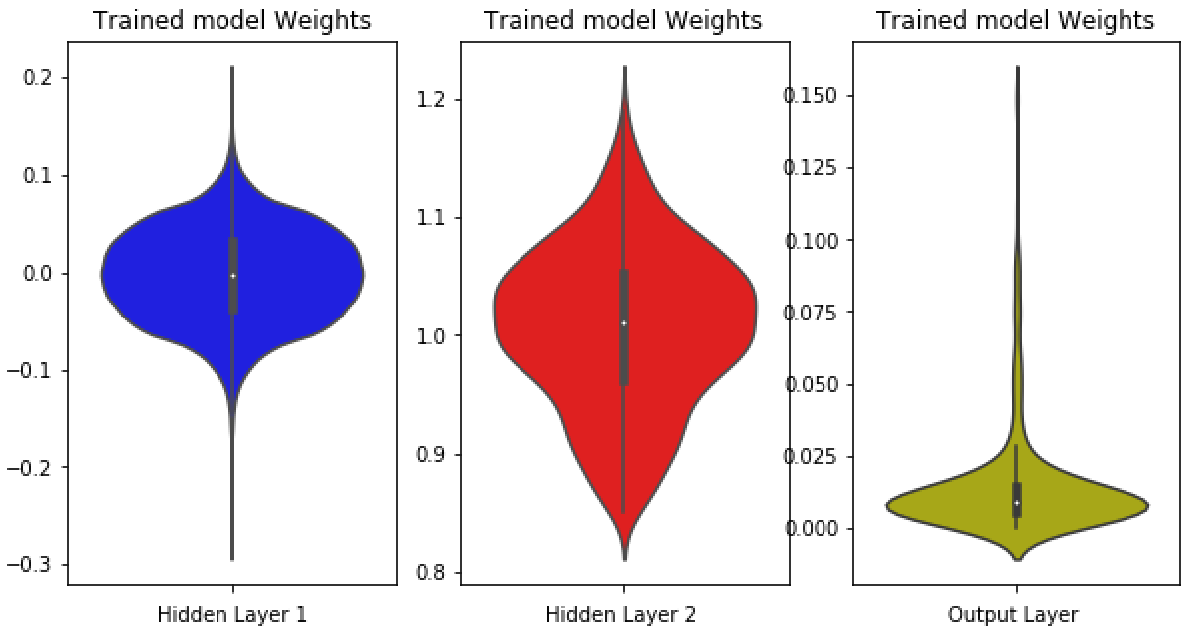
```
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_36 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_37 (Dense)             (None, 10)                650
=================================================================
Total params: 479,050
Trainable params: 477,770
Non-trainable params: 1,280
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 102us/sample - loss:
0.0659 - acc: 0.9812
Test score: 0.06589906297894195
Test accuracy: 0.9812
```



In [35]:

```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



3 ReLU hidden Layers (384-256-128) + BatchNormalization + Dropout + ADAM

In [36]:

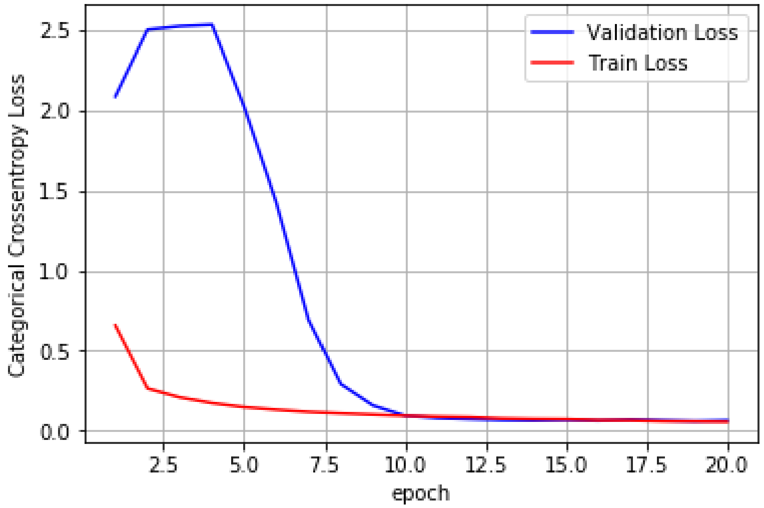
```
relumodel_3 = tf.keras.models.Sequential()
relumodel_3.add(tf.keras.layers.Dense(384, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.BatchNormalization())
relumodel_3.add(tf.keras.layers.Dropout(0.5))
relumodel_3.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_3.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_3.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_3.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_3.summary())
print("************************************************")

score = relumodel_3.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 3,
                                    "Model": "3-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 384 -
> 256 -> 128 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
```

```
                                                      "Dropout-Present": True,
                                                      "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                                      "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                                      "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                                      "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/sample - loss:
0.6575 - acc: 0.7929 - val_loss: 2.0845 - val_acc: 0.0976
Epoch 2/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.2646 - acc: 0.9195 - val_loss: 2.5004 - val_acc: 0.0974
Epoch 3/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.2092 - acc: 0.9355 - val_loss: 2.5218 - val_acc: 0.0975
Epoch 4/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1732 - acc: 0.9473 - val_loss: 2.5314 - val_acc: 0.0994
Epoch 5/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1480 - acc: 0.9542 - val_loss: 2.0204 - val_acc: 0.2185
Epoch 6/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.1317 - acc: 0.9594 - val_loss: 1.4259 - val_acc: 0.3795
```

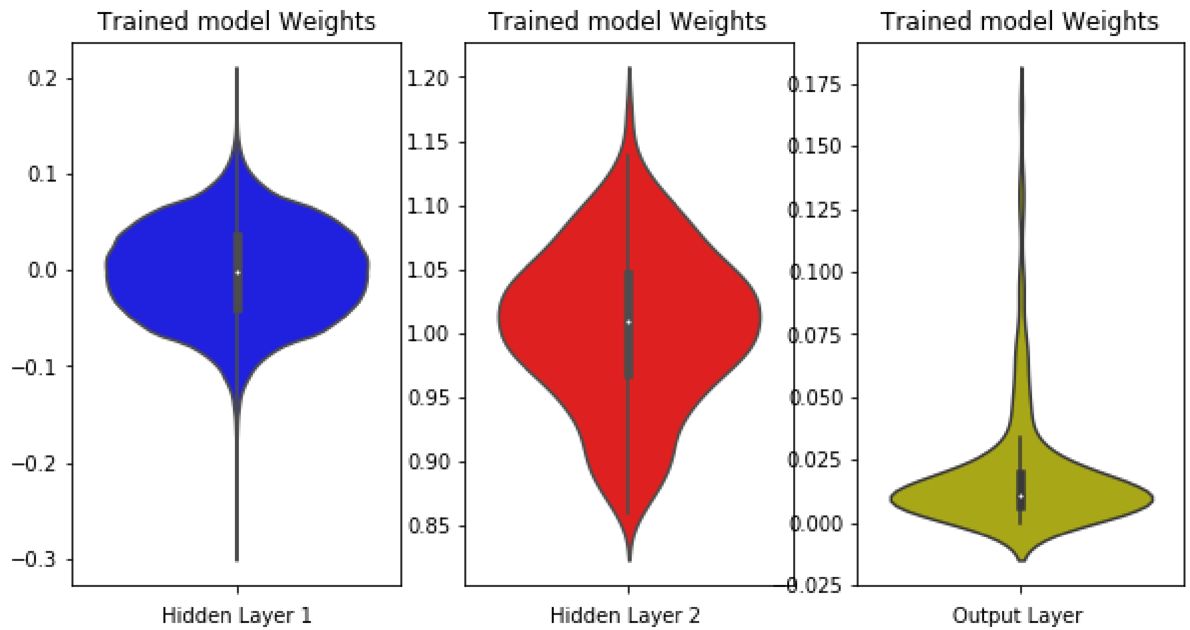
```
Epoch 7/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.1181 - acc: 0.9630 - val_loss: 0.6956 - val_acc: 0.7306
Epoch 8/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.1091 - acc: 0.9663 - val_loss: 0.2920 - val_acc: 0.9046
Epoch 9/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.1014 - acc: 0.9682 - val_loss: 0.1586 - val_acc: 0.9481
Epoch 10/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0937 - acc: 0.9708 - val_loss: 0.0946 - val_acc: 0.9706
Epoch 11/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0886 - acc: 0.9713 - val_loss: 0.0796 - val_acc: 0.9744
Epoch 12/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0852 - acc: 0.9730 - val_loss: 0.0729 - val_acc: 0.9770
Epoch 13/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.0769 - acc: 0.9758 - val_loss: 0.0673 - val_acc: 0.9784
Epoch 14/20
60000/60000 [==============================] - 5s 77us/sample - loss:
0.0735 - acc: 0.9761 - val_loss: 0.0648 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 5s 92us/sample - loss:
0.0722 - acc: 0.9770 - val_loss: 0.0678 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 5s 90us/sample - loss:
0.0663 - acc: 0.9779 - val_loss: 0.0633 - val_acc: 0.9799
Epoch 17/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0656 - acc: 0.9790 - val_loss: 0.0694 - val_acc: 0.9801
Epoch 18/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0599 - acc: 0.9808 - val_loss: 0.0641 - val_acc: 0.9805
Epoch 19/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0587 - acc: 0.9809 - val_loss: 0.0600 - val_acc: 0.9819
```

```
Epoch 20/20
60000/60000 [==============================] - 5s 76us/sample - loss:
0.0557 - acc: 0.9824 - val_loss: 0.0643 - val_acc: 0.9819
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_38 (Dense)             (None, 384)               301440
_________________________________________________________________
batch_normalization_v1_10 (B (None, 384)               1536
_________________________________________________________________
dropout_10 (Dropout)         (None, 384)               0
_________________________________________________________________
dense_39 (Dense)             (None, 256)               98560
_________________________________________________________________
batch_normalization_v1_11 (B (None, 256)               1024
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_40 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_41 (Dense)             (None, 10)                1290
=================================================================
Total params: 436,746
Trainable params: 435,466
Non-trainable params: 1,280
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 99us/sample - loss:
0.0643 - acc: 0.9819
Test score: 0.06429926064036845
Test accuracy: 0.9819
```



In [37]:
```
w_after = relumodel_3.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(figsize=(10, 5))
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()
```



5 Hidden Layers architecture

In [0]:

```
final_output = final_output.append({"#Layers": "--",
                              "Model": "--",
                              "Layer-Architecture": "--",
                              "Optimizer": "--", "BN-Present": "-
-",
                              "Dropout-Present": "--",
                              "Train-loss": "--",
                              "Test-loss": "--",
                              "Train-accuracy": "--",
                              "Test-Accuracy": "--"}, ignore_inde
x=True)
```

5 ReLU hidden Layers (512-384-256-128-64) + ADAM

In [39]:

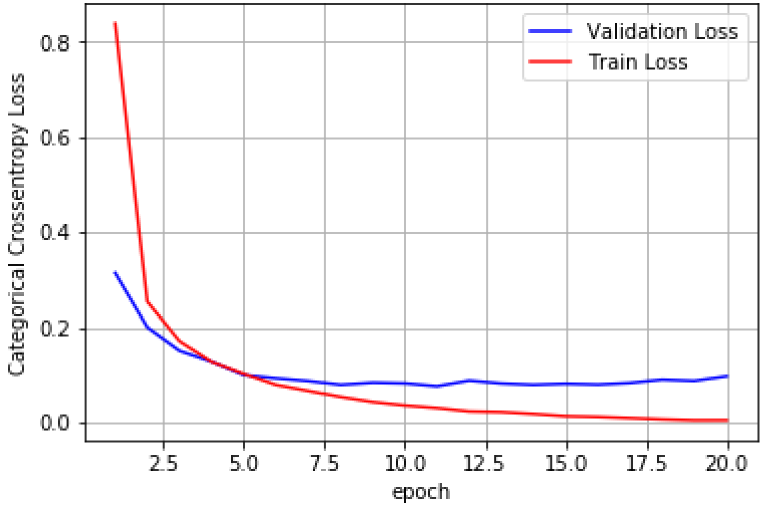
```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.Dense(384, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("*************************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("*************************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 384 -> 256 -> 128 -> 64 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
```

```
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 106us/sample - loss:
0.8376 - acc: 0.7474 - val_loss: 0.3146 - val_acc: 0.9047
Epoch 2/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.2546 - acc: 0.9247 - val_loss: 0.2004 - val_acc: 0.9381
Epoch 3/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.1716 - acc: 0.9494 - val_loss: 0.1515 - val_acc: 0.9550
Epoch 4/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.1279 - acc: 0.9617 - val_loss: 0.1282 - val_acc: 0.9596
Epoch 5/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.1027 - acc: 0.9691 - val_loss: 0.1003 - val_acc: 0.9683
Epoch 6/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0792 - acc: 0.9761 - val_loss: 0.0932 - val_acc: 0.9711
Epoch 7/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0665 - acc: 0.9795 - val_loss: 0.0875 - val_acc: 0.9739
Epoch 8/20
60000/60000 [==============================] - 6s 94us/sample - loss:
```

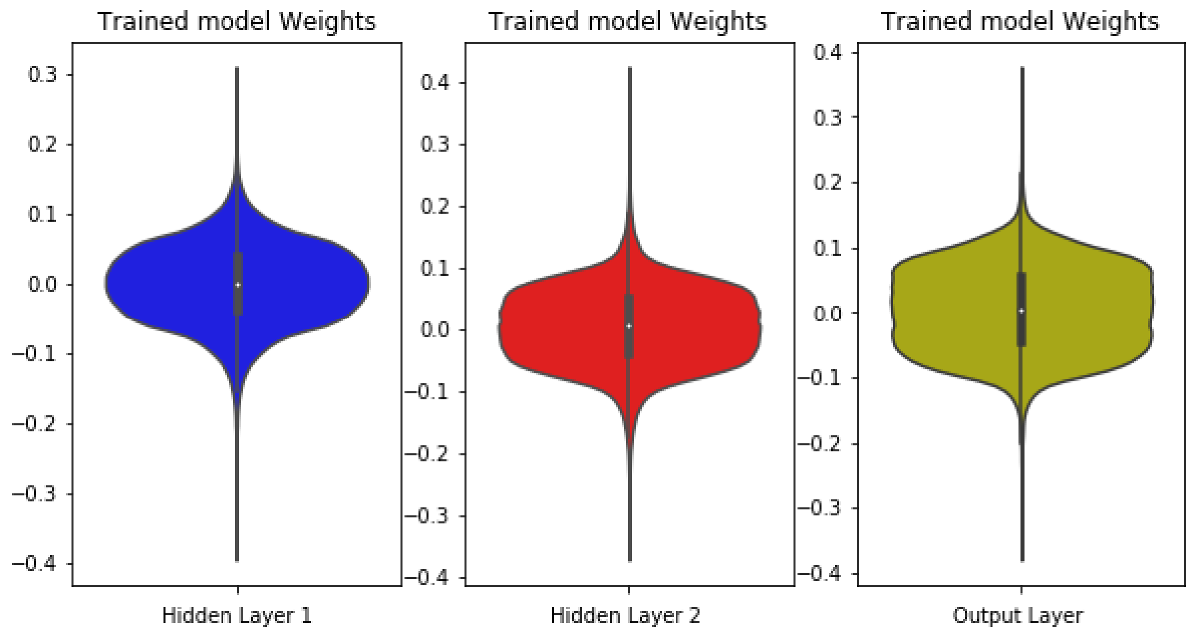
```
0.0538 - acc: 0.9838 - val_loss: 0.0794 - val_acc: 0.9760
Epoch 9/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0431 - acc: 0.9869 - val_loss: 0.0839 - val_acc: 0.9742
Epoch 10/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0358 - acc: 0.9896 - val_loss: 0.0825 - val_acc: 0.9756
Epoch 11/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0305 - acc: 0.9909 - val_loss: 0.0763 - val_acc: 0.9772
Epoch 12/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0233 - acc: 0.9933 - val_loss: 0.0883 - val_acc: 0.9747
Epoch 13/20
60000/60000 [==============================] - 6s 95us/sample - loss:
0.0219 - acc: 0.9936 - val_loss: 0.0822 - val_acc: 0.9763
Epoch 14/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0180 - acc: 0.9946 - val_loss: 0.0794 - val_acc: 0.9777
Epoch 15/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0134 - acc: 0.9963 - val_loss: 0.0816 - val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0118 - acc: 0.9966 - val_loss: 0.0799 - val_acc: 0.9794
Epoch 17/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0093 - acc: 0.9976 - val_loss: 0.0833 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] - 6s 93us/sample - loss:
0.0067 - acc: 0.9984 - val_loss: 0.0902 - val_acc: 0.9787
Epoch 19/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0048 - acc: 0.9989 - val_loss: 0.0876 - val_acc: 0.9793
Epoch 20/20
60000/60000 [==============================] - 6s 94us/sample - loss:
0.0048 - acc: 0.9989 - val_loss: 0.0976 - val_acc: 0.9779
**********************************************
Printing the Model Summary
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_42 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_43 (Dense)             (None, 384)               196992
_________________________________________________________________
dense_44 (Dense)             (None, 256)               98560
_________________________________________________________________
dense_45 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_46 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_47 (Dense)             (None, 10)                650
=================================================================
Total params: 739,274
Trainable params: 739,274
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 123us/sample - loss:
0.0976 - acc: 0.9779
Test score: 0.09762186423194326
Test accuracy: 0.9779
```



In [40]:
```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



5 ReLU hidden Layers (512-256-128-64-32) + ADAM

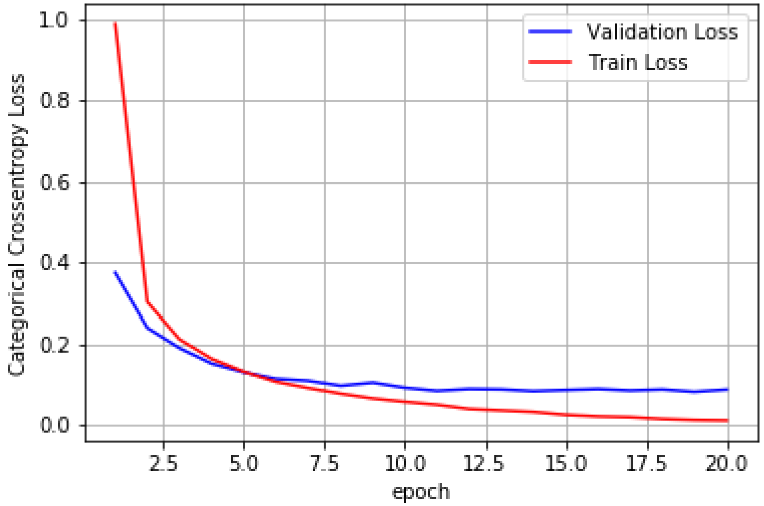
In [41]:
```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(32, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("***********************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("***********************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 256 -> 128 -> 64 -> 32 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 87us/sample - loss:
0.9865 - acc: 0.7222 - val_loss: 0.3746 - val_acc: 0.8934
Epoch 2/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.3030 - acc: 0.9122 - val_loss: 0.2393 - val_acc: 0.9280
Epoch 3/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.2113 - acc: 0.9388 - val_loss: 0.1898 - val_acc: 0.9431
Epoch 4/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.1631 - acc: 0.9519 - val_loss: 0.1514 - val_acc: 0.9548
Epoch 5/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.1315 - acc: 0.9614 - val_loss: 0.1302 - val_acc: 0.9616
Epoch 6/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.1066 - acc: 0.9682 - val_loss: 0.1140 - val_acc: 0.9642
Epoch 7/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0913 - acc: 0.9722 - val_loss: 0.1091 - val_acc: 0.9658
Epoch 8/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0771 - acc: 0.9765 - val_loss: 0.0967 - val_acc: 0.9692
Epoch 9/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0653 - acc: 0.9811 - val_loss: 0.1044 - val_acc: 0.9686
Epoch 10/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0571 - acc: 0.9829 - val_loss: 0.0917 - val_acc: 0.9699
Epoch 11/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0499 - acc: 0.9851 - val_loss: 0.0842 - val_acc: 0.9751
Epoch 12/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0394 - acc: 0.9884 - val_loss: 0.0888 - val_acc: 0.9728
Epoch 13/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0358 - acc: 0.9896 - val_loss: 0.0878 - val_acc: 0.9726
```

```
Epoch 14/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0321 - acc: 0.9905 - val_loss: 0.0835 - val_acc: 0.9750
Epoch 15/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0250 - acc: 0.9931 - val_loss: 0.0862 - val_acc: 0.9752
Epoch 16/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0210 - acc: 0.9944 - val_loss: 0.0887 - val_acc: 0.9746
Epoch 17/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0192 - acc: 0.9949 - val_loss: 0.0850 - val_acc: 0.9761
Epoch 18/20
60000/60000 [==============================] - 4s 73us/sample - loss:
0.0147 - acc: 0.9961 - val_loss: 0.0876 - val_acc: 0.9769
Epoch 19/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0122 - acc: 0.9970 - val_loss: 0.0819 - val_acc: 0.9778
Epoch 20/20
60000/60000 [==============================] - 4s 74us/sample - loss:
0.0111 - acc: 0.9971 - val_loss: 0.0873 - val_acc: 0.9776
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_48 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_49 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_50 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_51 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_52 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_53 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
```

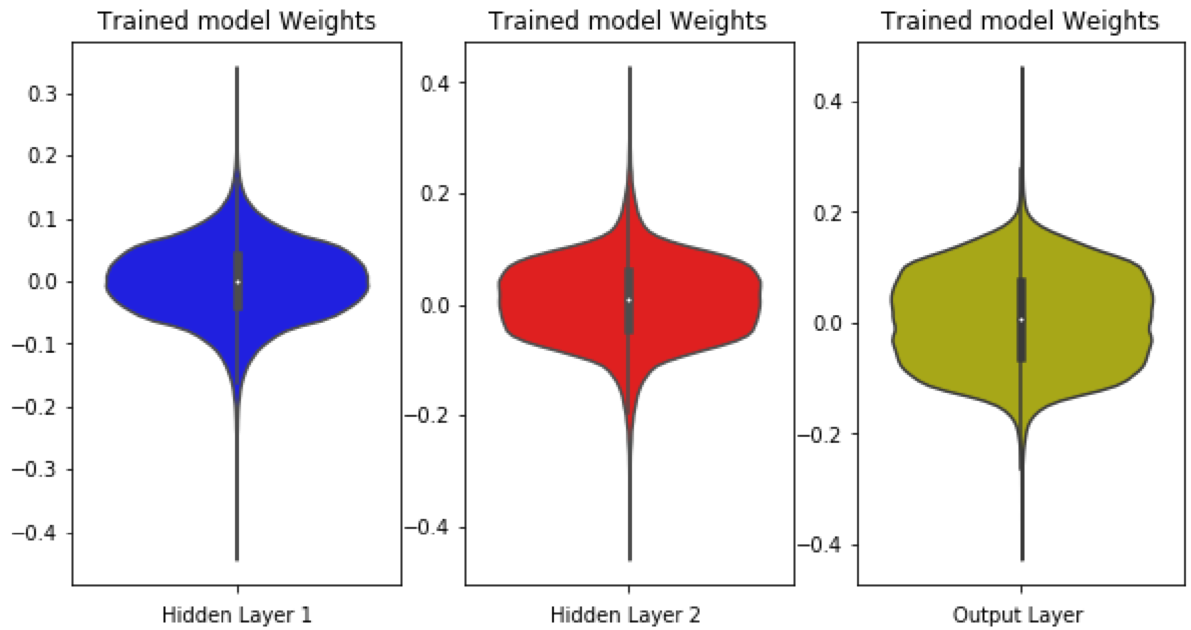
```
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 97us/sample - loss:
0.0873 - acc: 0.9776
Test score: 0.08732793643142504
Test accuracy: 0.9776
```



In [42]:

```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



5 ReLU hidden Layers (512-128-64-32-16) + ADAM

In [43]:
```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

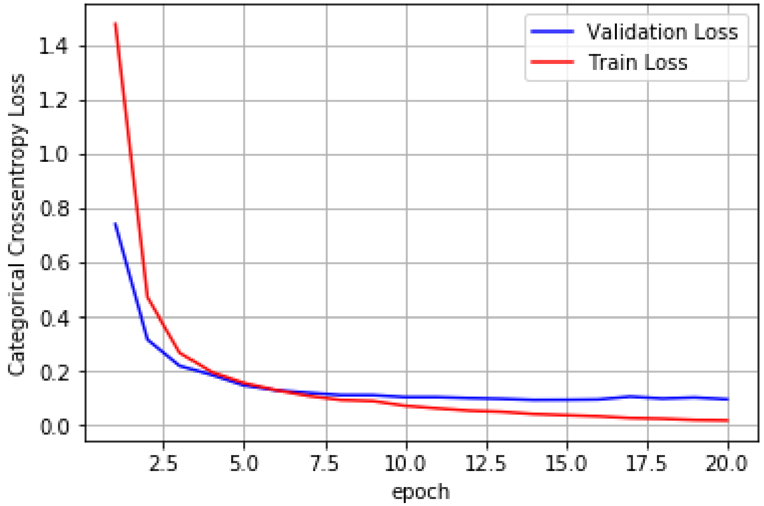
```
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(32, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(16, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**************************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("**************************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 128 -> 64 -> 32 -> 16 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
False,
                                    "Dropout-Present": False,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
```

```python
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 74us/sample - loss:
1.4770 - acc: 0.5357 - val_loss: 0.7409 - val_acc: 0.7723
Epoch 2/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.4721 - acc: 0.8718 - val_loss: 0.3171 - val_acc: 0.9142
Epoch 3/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.2684 - acc: 0.9276 - val_loss: 0.2203 - val_acc: 0.9381
Epoch 4/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1965 - acc: 0.9452 - val_loss: 0.1864 - val_acc: 0.9453
Epoch 5/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.1565 - acc: 0.9559 - val_loss: 0.1476 - val_acc: 0.9586
Epoch 6/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1301 - acc: 0.9627 - val_loss: 0.1297 - val_acc: 0.9622
Epoch 7/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.1091 - acc: 0.9683 - val_loss: 0.1208 - val_acc: 0.9665
Epoch 8/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0940 - acc: 0.9732 - val_loss: 0.1123 - val_acc: 0.9678
Epoch 9/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0902 - acc: 0.9737 - val_loss: 0.1121 - val_acc: 0.9674
Epoch 10/20
60000/60000 [==============================] - 4s 61us/sample - loss:
```

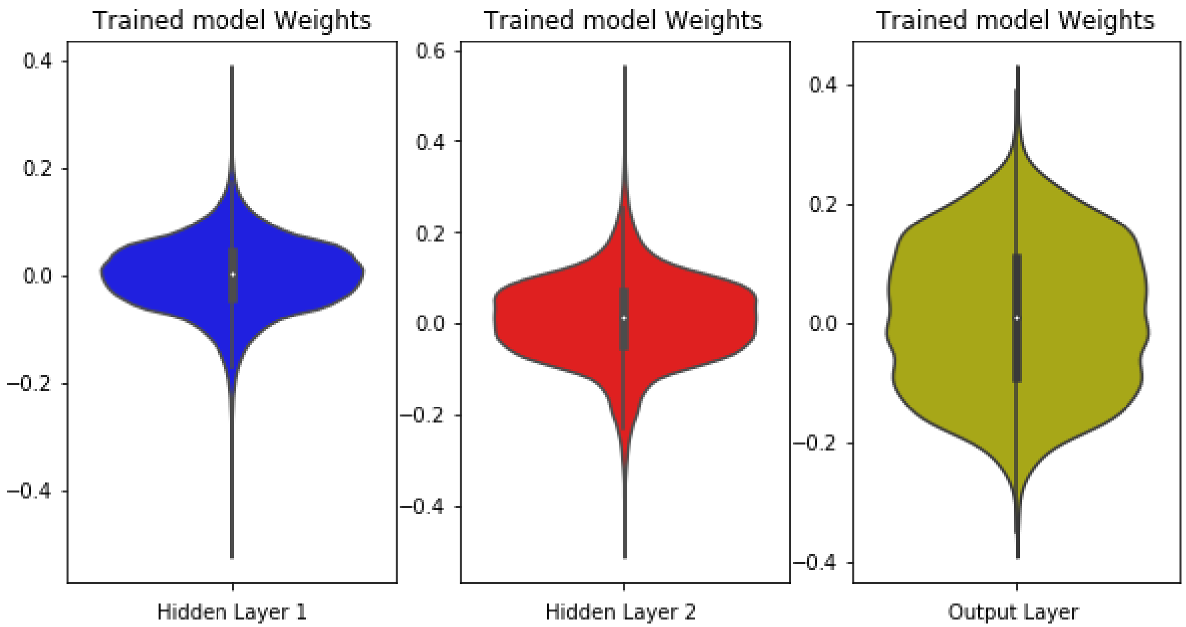
```
0.0728 - acc: 0.9786 - val_loss: 0.1048 - val_acc: 0.9708
Epoch 11/20
60000/60000 [==============================] - 4s 62us/sample - loss:
0.0632 - acc: 0.9820 - val_loss: 0.1047 - val_acc: 0.9700
Epoch 12/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0547 - acc: 0.9844 - val_loss: 0.1009 - val_acc: 0.9716
Epoch 13/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0505 - acc: 0.9854 - val_loss: 0.0984 - val_acc: 0.9728
Epoch 14/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0422 - acc: 0.9878 - val_loss: 0.0943 - val_acc: 0.9722
Epoch 15/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0380 - acc: 0.9891 - val_loss: 0.0947 - val_acc: 0.9728
Epoch 16/20
60000/60000 [==============================] - 4s 75us/sample - loss:
0.0338 - acc: 0.9905 - val_loss: 0.0961 - val_acc: 0.9731
Epoch 17/20
60000/60000 [==============================] - 4s 66us/sample - loss:
0.0274 - acc: 0.9929 - val_loss: 0.1066 - val_acc: 0.9708
Epoch 18/20
60000/60000 [==============================] - 4s 63us/sample - loss:
0.0249 - acc: 0.9934 - val_loss: 0.0993 - val_acc: 0.9724
Epoch 19/20
60000/60000 [==============================] - 4s 61us/sample - loss:
0.0203 - acc: 0.9949 - val_loss: 0.1033 - val_acc: 0.9723
Epoch 20/20
60000/60000 [==============================] - 4s 60us/sample - loss:
0.0186 - acc: 0.9952 - val_loss: 0.0968 - val_acc: 0.9739
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_54 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_55 (Dense)             (None, 128)               65664
```

```
dense_56 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_57 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_58 (Dense)             (None, 16)                528
_________________________________________________________________
dense_59 (Dense)             (None, 10)                170
=================================================================
Total params: 478,618
Trainable params: 478,618
Non-trainable params: 0
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 84us/sample - loss:
0.0968 - acc: 0.9739
Test score: 0.09677388302332256
Test accuracy: 0.9739
```



In [44]:
```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



5 ReLU hidden Layers (512-384-256-128-64) + BatchNormalization + Dropout + ADAM

In [45]:
```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(384, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
```

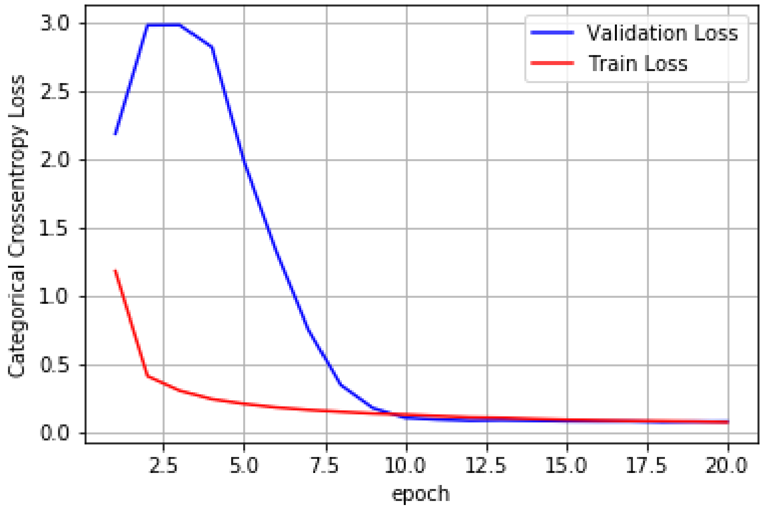
```
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("**************************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("**************************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 384 -> 256 -> 128 -> 64 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')
```

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 154us/sample - loss:
1.1776 - acc: 0.6162 - val_loss: 2.1850 - val_acc: 0.1435
Epoch 2/20
60000/60000 [==============================] - 8s 128us/sample - loss:
0.4080 - acc: 0.8788 - val_loss: 2.9801 - val_acc: 0.0974
Epoch 3/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.3010 - acc: 0.9135 - val_loss: 2.9817 - val_acc: 0.0980
Epoch 4/20
60000/60000 [==============================] - 8s 128us/sample - loss:
0.2378 - acc: 0.9312 - val_loss: 2.8179 - val_acc: 0.1386
Epoch 5/20
60000/60000 [==============================] - 8s 127us/sample - loss:
0.2035 - acc: 0.9415 - val_loss: 1.9801 - val_acc: 0.3267
Epoch 6/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.1769 - acc: 0.9487 - val_loss: 1.3237 - val_acc: 0.5080
Epoch 7/20
60000/60000 [==============================] - 8s 128us/sample - loss:
0.1589 - acc: 0.9538 - val_loss: 0.7492 - val_acc: 0.7186
Epoch 8/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.1448 - acc: 0.9576 - val_loss: 0.3424 - val_acc: 0.8903
Epoch 9/20
60000/60000 [==============================] - 8s 130us/sample - loss:
0.1324 - acc: 0.9613 - val_loss: 0.1728 - val_acc: 0.9449
Epoch 10/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.1248 - acc: 0.9650 - val_loss: 0.1008 - val_acc: 0.9679
Epoch 11/20
60000/60000 [==============================] - 8s 130us/sample - loss:
```

```
0.1141 - acc: 0.9669 - val_loss: 0.0880 - val_acc: 0.9734
Epoch 12/20
60000/60000 [==============================] - 8s 130us/sample - loss:
0.1042 - acc: 0.9695 - val_loss: 0.0800 - val_acc: 0.9776
Epoch 13/20
60000/60000 [==============================] - 8s 130us/sample - loss:
0.1005 - acc: 0.9705 - val_loss: 0.0827 - val_acc: 0.9780
Epoch 14/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.0926 - acc: 0.9731 - val_loss: 0.0796 - val_acc: 0.9782
Epoch 15/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.0864 - acc: 0.9749 - val_loss: 0.0757 - val_acc: 0.9797
Epoch 16/20
60000/60000 [==============================] - 8s 131us/sample - loss:
0.0821 - acc: 0.9757 - val_loss: 0.0744 - val_acc: 0.9804
Epoch 17/20
60000/60000 [==============================] - 8s 130us/sample - loss:
0.0784 - acc: 0.9771 - val_loss: 0.0743 - val_acc: 0.9800
Epoch 18/20
60000/60000 [==============================] - 8s 130us/sample - loss:
0.0768 - acc: 0.9775 - val_loss: 0.0695 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.0734 - acc: 0.9779 - val_loss: 0.0721 - val_acc: 0.9814
Epoch 20/20
60000/60000 [==============================] - 8s 129us/sample - loss:
0.0680 - acc: 0.9805 - val_loss: 0.0734 - val_acc: 0.9817
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_60 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_v1_12 (B (None, 512)               2048
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0
_________________________________________________________________
```

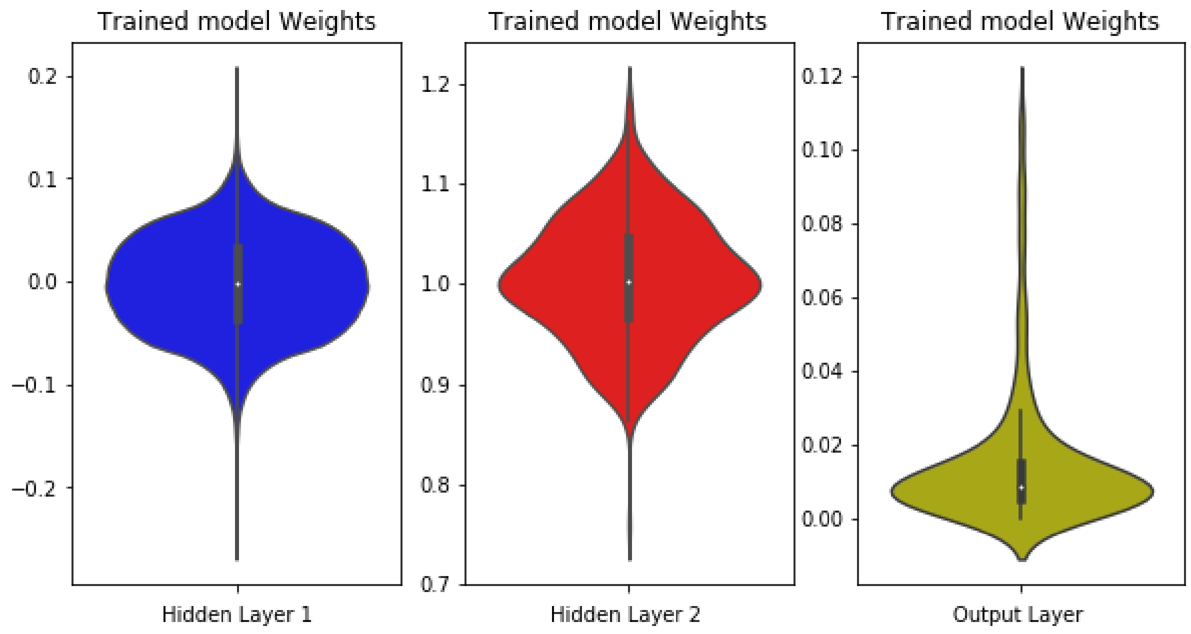
```
dense_61 (Dense)             (None, 384)               196992
_________________________________________________________________
batch_normalization_v1_13 (B (None, 384)               1536
_________________________________________________________________
dropout_13 (Dropout)         (None, 384)               0
_________________________________________________________________
dense_62 (Dense)             (None, 256)               98560
_________________________________________________________________
batch_normalization_v1_14 (B (None, 256)               1024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_63 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_v1_15 (B (None, 128)               512
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_64 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_65 (Dense)             (None, 10)                650
=================================================================
Total params: 744,394
Trainable params: 741,834
Non-trainable params: 2,560
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 148us/sample - loss:
0.0734 - acc: 0.9817
Test score: 0.07335510681418236
Test accuracy: 0.9817
```



In [46]:

```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



5 ReLU hidden Layers (512-256-128-64-32) + BatchNormalization + Dropout + ADAM

In [47]:

```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
```

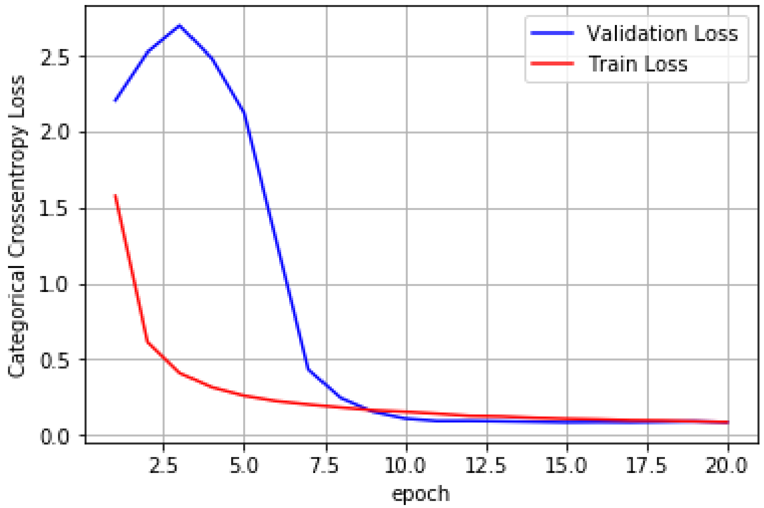
```
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(32, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("************************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 256 -> 128 -> 64 -> 32 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
```

```
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 127us/sample - loss:
1.5776 - acc: 0.4854 - val_loss: 2.2071 - val_acc: 0.3203
Epoch 2/20
60000/60000 [==============================] - 6s 102us/sample - loss:
0.6154 - acc: 0.8227 - val_loss: 2.5244 - val_acc: 0.1348
Epoch 3/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.4099 - acc: 0.8871 - val_loss: 2.6993 - val_acc: 0.1823
Epoch 4/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.3159 - acc: 0.9140 - val_loss: 2.4828 - val_acc: 0.2420
Epoch 5/20
60000/60000 [==============================] - 6s 100us/sample - loss:
0.2600 - acc: 0.9298 - val_loss: 2.1246 - val_acc: 0.2918
Epoch 6/20
60000/60000 [==============================] - 6s 100us/sample - loss:
0.2246 - acc: 0.9401 - val_loss: 1.2837 - val_acc: 0.5370
Epoch 7/20
60000/60000 [==============================] - 6s 100us/sample - loss:
0.2021 - acc: 0.9465 - val_loss: 0.4334 - val_acc: 0.8613
Epoch 8/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.1823 - acc: 0.9507 - val_loss: 0.2455 - val_acc: 0.9234
Epoch 9/20
60000/60000 [==============================] - 6s 100us/sample - loss:
0.1640 - acc: 0.9564 - val_loss: 0.1540 - val_acc: 0.9529
Epoch 10/20
60000/60000 [==============================] - 6s 101us/sample - loss:
```

```
0.1538 - acc: 0.9589 - val_loss: 0.1094 - val_acc: 0.9671
Epoch 11/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.1414 - acc: 0.9630 - val_loss: 0.0929 - val_acc: 0.9744
Epoch 12/20
60000/60000 [==============================] - 6s 100us/sample - loss:
0.1267 - acc: 0.9661 - val_loss: 0.0938 - val_acc: 0.9740
Epoch 13/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.1230 - acc: 0.9676 - val_loss: 0.0906 - val_acc: 0.9761
Epoch 14/20
60000/60000 [==============================] - 6s 102us/sample - loss:
0.1140 - acc: 0.9700 - val_loss: 0.0874 - val_acc: 0.9776
Epoch 15/20
60000/60000 [==============================] - 6s 103us/sample - loss:
0.1078 - acc: 0.9718 - val_loss: 0.0839 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] - 6s 102us/sample - loss:
0.1037 - acc: 0.9721 - val_loss: 0.0850 - val_acc: 0.9793
Epoch 17/20
60000/60000 [==============================] - 6s 101us/sample - loss:
0.0976 - acc: 0.9740 - val_loss: 0.0839 - val_acc: 0.9780
Epoch 18/20
60000/60000 [==============================] - 6s 102us/sample - loss:
0.0957 - acc: 0.9744 - val_loss: 0.0869 - val_acc: 0.9775
Epoch 19/20
60000/60000 [==============================] - 6s 108us/sample - loss:
0.0908 - acc: 0.9761 - val_loss: 0.0902 - val_acc: 0.9788
Epoch 20/20
60000/60000 [==============================] - 7s 110us/sample - loss:
0.0851 - acc: 0.9769 - val_loss: 0.0845 - val_acc: 0.9792
***********************************************
Printing the Model Summary

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_66 (Dense)             (None, 512)               401920

_________________________________________________________________
batch_normalization_v1_16 (B (None, 512)               2048
```

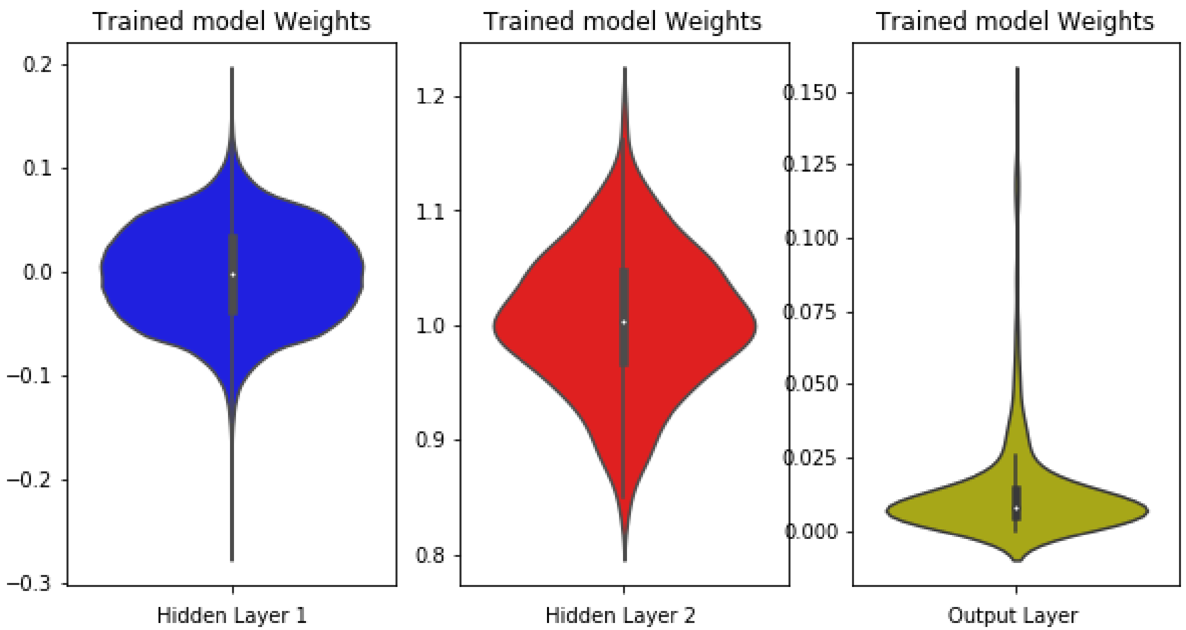
```
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_67 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_v1_17 (B (None, 256)               1024
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_68 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_v1_18 (B (None, 128)               512
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_69 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_19 (B (None, 64)                256
_________________________________________________________________
dropout_19 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_70 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_71 (Dense)             (None, 10)                330
=================================================================
Total params: 580,650
Trainable params: 578,730
Non-trainable params: 1,920
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 2s 165us/sample - loss:
0.0845 - acc: 0.9792
Test score: 0.08452142000179738
Test accuracy: 0.9792
```



In [48]:

```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



5 ReLU hidden Layers (512-128-64-32-16) + BatchNormalization + Dropout + ADAM

In [49]:

```
relumodel_5 = tf.keras.models.Sequential()
relumodel_5.add(tf.keras.layers.Dense(512, activation=tf.nn.relu, input
_shape=(input_dim, )))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(64, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(32, activation=tf.nn.relu))
```

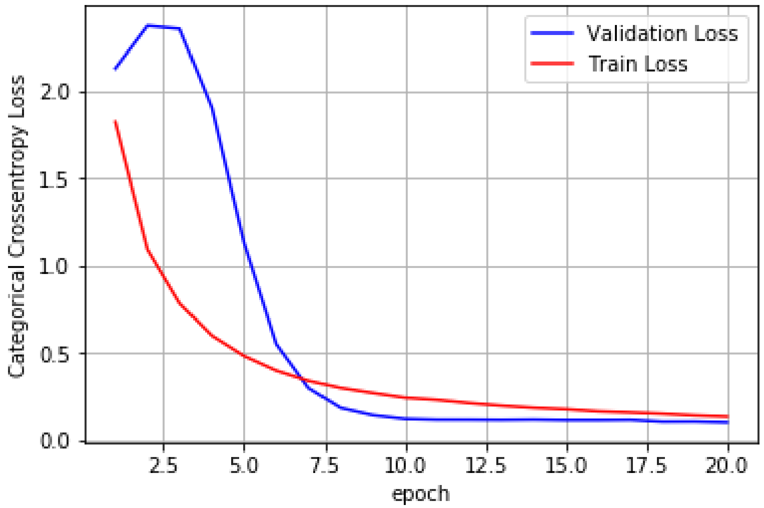
```
relumodel_5.add(tf.keras.layers.BatchNormalization())
relumodel_5.add(tf.keras.layers.Dropout(0.5))
relumodel_5.add(tf.keras.layers.Dense(16, activation=tf.nn.relu))
relumodel_5.add(tf.keras.layers.Dense(output_dim, activation=tf.nn.soft
max))

relumodel_5.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model = relumodel_5.fit(x_train, y_train, epochs=n_epochs, batch_size=b
atchsize, verbose=1, validation_data=(x_test, y_test))

print("************************************************")
print("Printing the Model Summary")
print(relumodel_5.summary())
print("************************************************")

score = relumodel_5.evaluate(x_test, y_test)

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

final_output = final_output.append({"#Layers": 5,
                                    "Model": "5-ReLU + Softmax",
                                    "Layer-Architecture": "784 -> 512 -
> 128 -> 64 -> 32 -> 16 -> 10",
                                    "Optimizer": "ADAM", "BN-Present":
True,
                                    "Dropout-Present": True,
                                    "Train-loss": '{:.5f}'.format(model
.history["loss"][n_epochs-1]),
                                    "Test-loss": '{:.5f}'.format(model.
history["val_loss"][n_epochs-1]),
                                    "Train-accuracy": '{:.5f}'.format(m
odel.history["acc"][n_epochs-1]),
                                    "Test-Accuracy": '{:.5f}'.format(mo
del.history["val_acc"][n_epochs-1])}, ignore_index=True)

fig,ax = plt.subplots(1,1)
```

```python
ax.set_xlabel('epoch')
ax.set_ylabel('Categorical Crossentropy Loss')

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

vy = model.history['val_loss']
ty = model.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 111us/sample - loss:
1.8243 - acc: 0.3702 - val_loss: 2.1310 - val_acc: 0.2047
Epoch 2/20
60000/60000 [==============================] - 5s 84us/sample - loss:
1.0923 - acc: 0.6401 - val_loss: 2.3769 - val_acc: 0.1413
Epoch 3/20
60000/60000 [==============================] - 5s 84us/sample - loss:
0.7828 - acc: 0.7556 - val_loss: 2.3591 - val_acc: 0.1599
Epoch 4/20
60000/60000 [==============================] - 5s 84us/sample - loss:
0.5977 - acc: 0.8222 - val_loss: 1.9045 - val_acc: 0.1981
Epoch 5/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.4813 - acc: 0.8649 - val_loss: 1.1301 - val_acc: 0.5712
Epoch 6/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.3978 - acc: 0.8934 - val_loss: 0.5505 - val_acc: 0.8466
Epoch 7/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.3391 - acc: 0.9116 - val_loss: 0.2975 - val_acc: 0.9232
Epoch 8/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.2975 - acc: 0.9240 - val_loss: 0.1832 - val_acc: 0.9515
Epoch 9/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.2686 - acc: 0.9325 - val_loss: 0.1413 - val_acc: 0.9633
Epoch 10/20
60000/60000 [==============================] - 5s 83us/sample - loss:
```

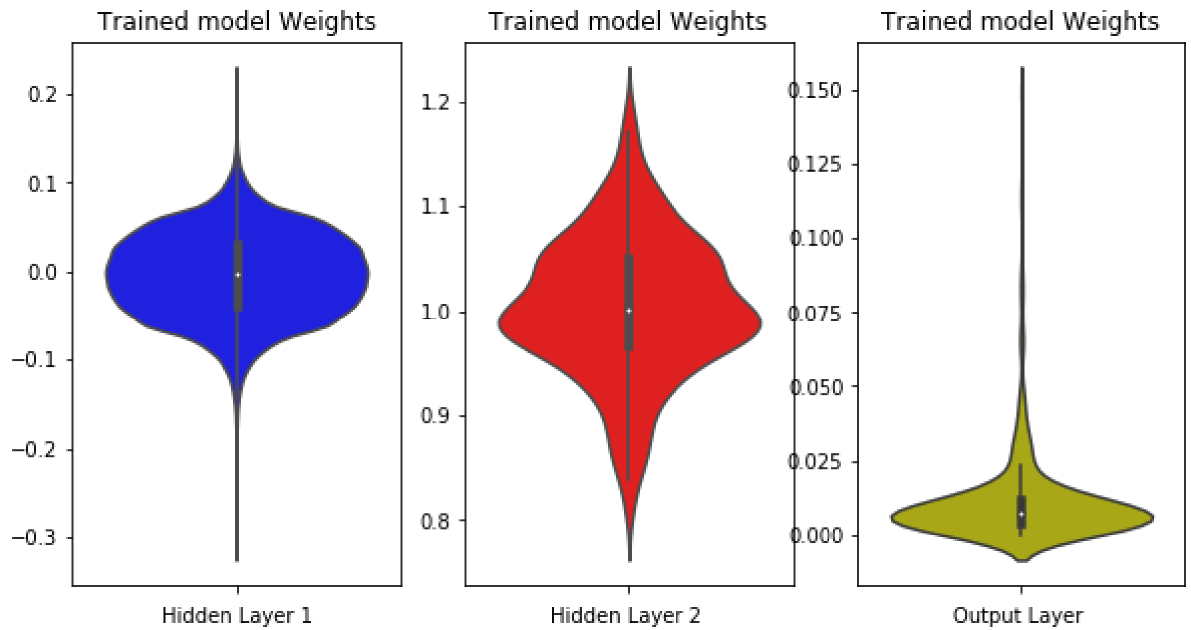
```
0.2417 - acc: 0.9394 - val_loss: 0.1198 - val_acc: 0.9694
Epoch 11/20
60000/60000 [==============================] - 5s 84us/sample - loss:
0.2292 - acc: 0.9432 - val_loss: 0.1151 - val_acc: 0.9714
Epoch 12/20
60000/60000 [==============================] - 5s 85us/sample - loss:
0.2107 - acc: 0.9483 - val_loss: 0.1149 - val_acc: 0.9716
Epoch 13/20
60000/60000 [==============================] - 5s 84us/sample - loss:
0.1958 - acc: 0.9520 - val_loss: 0.1137 - val_acc: 0.9733
Epoch 14/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.1838 - acc: 0.9554 - val_loss: 0.1154 - val_acc: 0.9732
Epoch 15/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.1755 - acc: 0.9575 - val_loss: 0.1127 - val_acc: 0.9753
Epoch 16/20
60000/60000 [==============================] - 5s 83us/sample - loss:
0.1631 - acc: 0.9603 - val_loss: 0.1126 - val_acc: 0.9762
Epoch 17/20
60000/60000 [==============================] - 5s 85us/sample - loss:
0.1568 - acc: 0.9626 - val_loss: 0.1140 - val_acc: 0.9764
Epoch 18/20
60000/60000 [==============================] - 5s 85us/sample - loss:
0.1495 - acc: 0.9637 - val_loss: 0.1035 - val_acc: 0.9778
Epoch 19/20
60000/60000 [==============================] - 5s 84us/sample - loss:
0.1400 - acc: 0.9659 - val_loss: 0.1041 - val_acc: 0.9778
Epoch 20/20
60000/60000 [==============================] - 5s 85us/sample - loss:
0.1333 - acc: 0.9679 - val_loss: 0.0996 - val_acc: 0.9794
***********************************************
Printing the Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_72 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_v1_20 (B (None, 512)               2048
```

```
_________________________________________________________________
dropout_20 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_73 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_v1_21 (B (None, 128)               512
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_74 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_v1_22 (B (None, 64)                256
_________________________________________________________________
dropout_22 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_75 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_v1_23 (B (None, 32)                128
_________________________________________________________________
dropout_23 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_76 (Dense)             (None, 16)                528
_________________________________________________________________
dense_77 (Dense)             (None, 10)                170
=================================================================
Total params: 481,562
Trainable params: 480,090
Non-trainable params: 1,472
_________________________________________________________________
None
***********************************************
10000/10000 [==============================] - 1s 141us/sample - loss:
0.0996 - acc: 0.9794
Test score: 0.09960540832220577
Test accuracy: 0.9794
```



In [50]:
```
w_after = relumodel_5.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(figsize=(10, 5))
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()
```



Conclusion

Steps Summary -

- The dataset that is taken for analysing MLP techniques is MNIST which contains data about the handwritten images. Bascially, we have to classify the handwritten numbers from the image to numeric.
- A number of architectures were deployed - ***2 hidden layers, 3 hidden layers, 5 hidden layers***.
- In this, we specifically took Rectified Linear Unit (**ReLU**) as our default activation function with 'AdaM' optimizer.
- For each of the architecture, we tried BatchNormalization(**BN**) and **Dropout** (With dropout rate to 0.5) to see whether our model performs better or not.

In [51]:
```
print("The below output summarizes the number of architecture performances that were deployed on MNIST Dataset")
print("*****************************************************************************************************")
final_output
```

```
The below output summarizes the number of architecture performances that were deployed on MNIST Dataset
*****************************************************************************************
```

Out[51]:

	#Layers	Model	Layer-Architecture	Optimizer	BN-Present	Dropout-Present	Train-loss	Test-loss	Train-accuracy	Ac
0	2	2-ReLU + Softmax	784 -> 512 -> 128 -> 10	ADAM	False	False	0.03062	0.06986	0.99197	(
1	2	2-ReLU + Softmax	784 -> 256 -> 256 -> 10	ADAM	False	False	0.04022	0.07797	0.98873	(
2	2	2-ReLU + Softmax	784 -> 384 -> 128 -> 10	ADAM	False	False	0.03553	0.07293	0.99040	(
3	2	2-ReLU + Softmax	784 -> 512 -> 128 -> 10	ADAM	True	True	0.05164	0.05973	0.98323	(
4	2	2-ReLU + Softmax	784 -> 256 -> 256 -> 10	ADAM	True	True	0.07511	0.06643	0.97567	(
5	2	2-ReLU + Softmax	784 -> 384 -> 128 -> 10	ADAM	True	True	0.06306	0.06616	0.98013	(
6	--	--	--	--	--	--	--	--	--	
7	3	3-ReLU + Softmax	784 -> 512 -> 256 -> 128 -> 10	ADAM	False	False	0.01240	0.07288	0.99708	(

	#Layers	Model	Layer-Architecture	Optimizer	BN-Present	Dropout-Present	Train-loss	Test-loss	Train-accuracy	Ac
8	3	3-ReLU + Softmax	784 -> 512 -> 128 -> 64 -> 10	ADAM	False	False	0.02145	0.07323	0.99483	(
9	3	3-ReLU + Softmax	784 -> 384 -> 256 -> 128 -> 10	ADAM	False	False	0.01295	0.07172	0.99713	(
10	3	3-ReLU + Softmax	784 -> 512 -> 256 -> 128 -> 10	ADAM	True	True	0.04785	0.06404	0.98450	(
11	3	3-ReLU + Softmax	784 -> 512 -> 128 -> 64 -> 10	ADAM	True	True	0.05202	0.06590	0.98335	(
12	3	3-ReLU + Softmax	784 -> 384 -> 256 -> 128 -> 10	ADAM	True	True	0.05572	0.06430	0.98243	(
13	--	--	--	--	--	--	--	--	--	
14	5	5-ReLU + Softmax	784 -> 512 -> 384 -> 256 -> 128 -> 64 -> 10	ADAM	False	False	0.00483	0.09762	0.99890	(
15	5	5-ReLU + Softmax	784 -> 512 -> 256 -> 128 -> 64 -> 32 -> 10	ADAM	False	False	0.01107	0.08733	0.99713	(
16	5	5-ReLU + Softmax	784 -> 512 -> 128 -> 64 -> 32 -> 16 -> 10	ADAM	False	False	0.01862	0.09677	0.99525	(
17	5	5-ReLU + Softmax	784 -> 512 -> 384 -> 256 -> 128 -> 64 -> 10	ADAM	True	True	0.06797	0.07336	0.98052	(
18	5	5-ReLU + Softmax	784 -> 512 -> 256 -> 128 -> 64 -> 32 -> 10	ADAM	True	True	0.08507	0.08452	0.97692	(

	#Layers	Model	Layer-Architecture	Optimizer	BN-Present	Dropout-Present	Train-loss	Test-loss	Train-accuracy	Ac
19	5	5-ReLU + Softmax	784 -> 512 -> 128 -> 64 -> 32 -> 16 -> 10	ADAM	True	True	0.13327	0.09961	0.96795	(

Output Representation -

- **Layers** -> The Number of Hidden Layers that were used.
- **Model** -> The Models that were used.
- **Layer Architecture** -> Number of neurons present in each hidden layer
- **Optimizer** -> The Type of Optimizer that was used.
- **BN-Present** -> Batch Normalization was used or not
- **Dropout-Present** -> Dropout rate was applied to the model or not

Output Conclusion -

By looking at the above table, We can conclude that after applying batch normalization and dropout, the loss(train & test) becomes significantly less different. Also, the accuracy for train as well test does not differs a lot.