Different MLP Arcitectures on MNIST dataset

Loading data

In [2]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" u
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

In [0]:
```
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid(True)
    fig.canvas.draw()
```

In [4]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz (https://
s3.amazonaws.com/img-datasets/mnist.npz)
11493376/11490434 [==============================] - 1s 0us/step
```

In [5]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of sh
print("Number of training examples :", X_test.shape[0], "and each image is of sha
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of training examples : 10000 and each image is of shape (28, 28)
```

In [0]:
```
# if you observe the input shape its 3 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

In [7]:
```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each image is of sh
print("Number of training examples :", X_test.shape[0], "and each image is of sha
```

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

In [8]:
```
# An example data point
print(X_train[0])
```

```
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   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]
```

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

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

X_train = X_train/255
X_test = X_test/255
```

In [10]:
```
# example data point after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

In [11]:
```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

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

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

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

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

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

# some model parameters
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

[1] MLP with 2 hidden layers

[1.1] Without Batch Normalization and Dropout

In [15]:
```
# Multilayer perceptron

model = Sequential()
model.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 364)               285740    
_________________________________________________________________
dense_5 (Dense)              (None, 52)                18980     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                530       
=================================================================
Total params: 305,250
Trainable params: 305,250
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

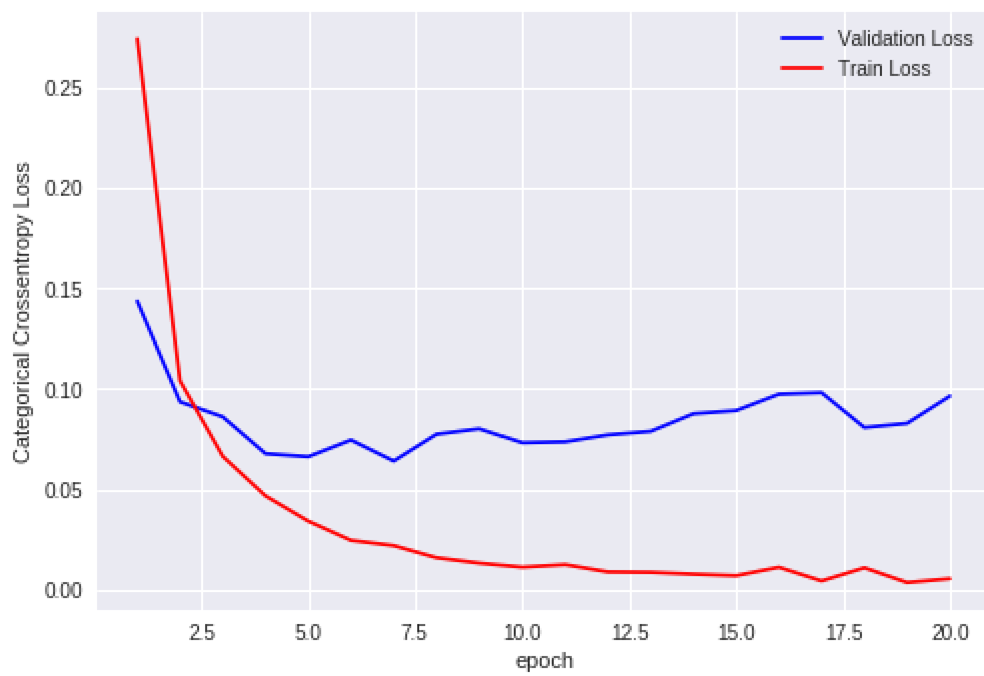
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2741 - ac
c: 0.9213 - val_loss: 0.1435 - val_acc: 0.9563
Epoch 2/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1041 - ac
c: 0.9690 - val_loss: 0.0936 - val_acc: 0.9729
Epoch 3/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0665 - ac
c: 0.9796 - val_loss: 0.0862 - val_acc: 0.9755
Epoch 4/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0468 - ac
c: 0.9855 - val_loss: 0.0678 - val_acc: 0.9787
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0343 - ac
c: 0.9892 - val_loss: 0.0663 - val_acc: 0.9803
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0247 - ac
c: 0.9924 - val_loss: 0.0746 - val_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0221 - ac
c: 0.9933 - val_loss: 0.0642 - val_acc: 0.9820
Epoch 8/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0161 - ac
c: 0.9946 - val_loss: 0.0775 - val_acc: 0.9788
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0134 - ac
c: 0.9957 - val_loss: 0.0802 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0113 - ac
c: 0.9964 - val_loss: 0.0733 - val_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0126 - ac
c: 0.9960 - val_loss: 0.0737 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0090 - ac
c: 0.9973 - val_loss: 0.0772 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0088 - ac
c: 0.9971 - val_loss: 0.0790 - val_acc: 0.9812
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0079 - ac
c: 0.9975 - val_loss: 0.0877 - val_acc: 0.9808
Epoch 15/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0071 - ac
c: 0.9976 - val_loss: 0.0893 - val_acc: 0.9781
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0113 - ac
c: 0.9965 - val_loss: 0.0975 - val_acc: 0.9787
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0045 - ac
c: 0.9986 - val_loss: 0.0982 - val_acc: 0.9792
Epoch 18/20
```

```
60000/60000 [==============================] - 4s 73us/step - loss: 0.0110 - ac
c: 0.9965 - val_loss: 0.0809 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0037 - ac
c: 0.9990 - val_loss: 0.0829 - val_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0056 - ac
c: 0.9982 - val_loss: 0.0965 - val_acc: 0.9793
```

In [18]:

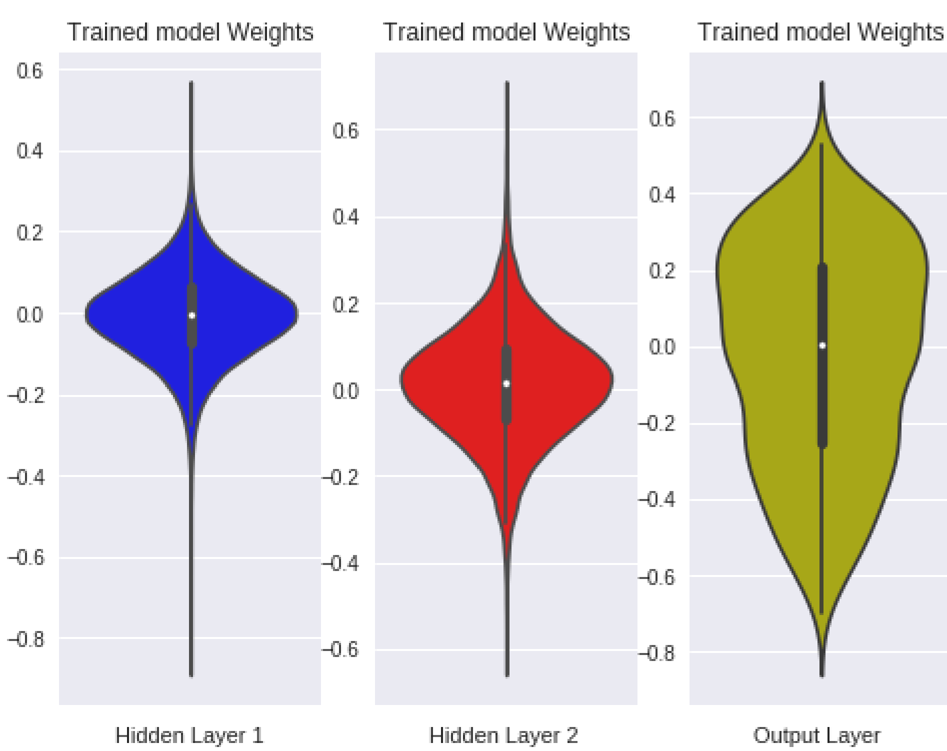
```python
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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


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

```
Test score: 0.09650619211766216
Test accuracy: 0.9793
```



In [19]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



In [0]:
```

```

[1.2] With Batch Normalization and Dropout

In [21]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model = Sequential()
model.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 364)               285740    
_________________________________________________________________
batch_normalization_1 (Batch (None, 364)               1456      
_________________________________________________________________
dropout_1 (Dropout)          (None, 364)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 52)                18980     
_________________________________________________________________
batch_normalization_2 (Batch (None, 52)                208       
_________________________________________________________________
dropout_2 (Dropout)          (None, 52)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                530       
=================================================================
Total params: 306,914
Trainable params: 306,082
Non-trainable params: 832
_________________________________________________________________
```

In [22]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

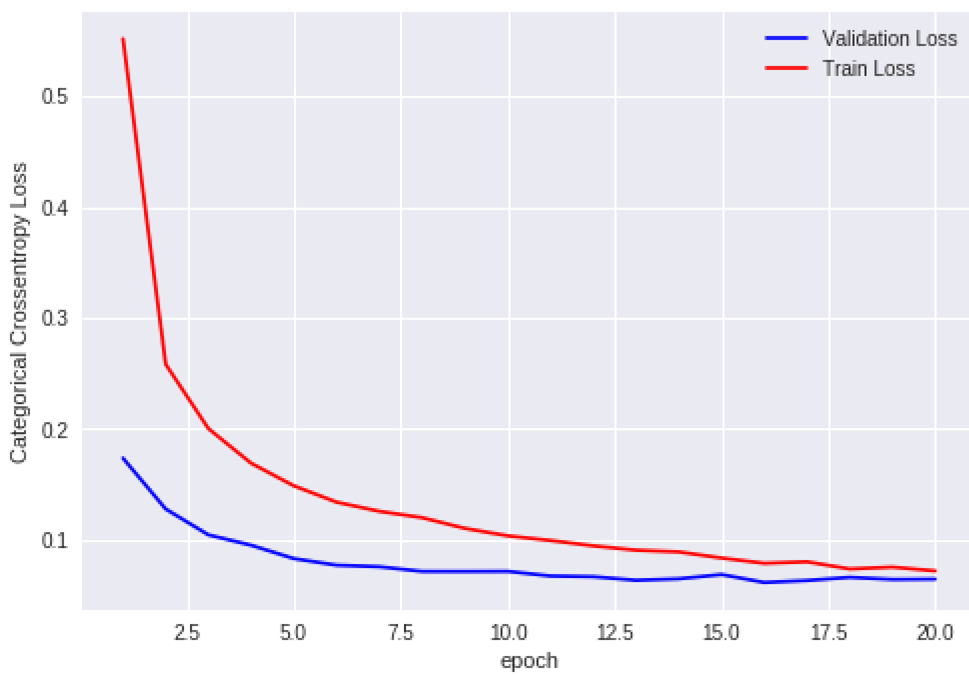
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.5517 - a
cc: 0.8330 - val_loss: 0.1736 - val_acc: 0.9462
Epoch 2/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.2584 - ac
c: 0.9258 - val_loss: 0.1278 - val_acc: 0.9600
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2003 - ac
c: 0.9423 - val_loss: 0.1044 - val_acc: 0.9666
Epoch 4/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1691 - ac
c: 0.9502 - val_loss: 0.0951 - val_acc: 0.9706
Epoch 5/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1487 - ac
c: 0.9580 - val_loss: 0.0830 - val_acc: 0.9750
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1339 - ac
c: 0.9607 - val_loss: 0.0770 - val_acc: 0.9758
Epoch 7/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1257 - ac
c: 0.9637 - val_loss: 0.0756 - val_acc: 0.9786
Epoch 8/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1200 - ac
c: 0.9646 - val_loss: 0.0714 - val_acc: 0.9774
Epoch 9/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1103 - ac
c: 0.9667 - val_loss: 0.0714 - val_acc: 0.9779
Epoch 10/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1035 - ac
c: 0.9689 - val_loss: 0.0715 - val_acc: 0.9781
Epoch 11/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0993 - a
cc: 0.9704 - val_loss: 0.0673 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0945 - ac
c: 0.9718 - val_loss: 0.0667 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0905 - ac
c: 0.9728 - val_loss: 0.0634 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0890 - ac
c: 0.9733 - val_loss: 0.0647 - val_acc: 0.9805
Epoch 15/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0835 - ac
c: 0.9753 - val_loss: 0.0686 - val_acc: 0.9796
Epoch 16/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0787 - ac
c: 0.9765 - val_loss: 0.0614 - val_acc: 0.9815
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0801 - ac
c: 0.9764 - val_loss: 0.0632 - val_acc: 0.9816
Epoch 18/20
```

```
60000/60000 [==============================] - 6s 95us/step - loss: 0.0736 - ac
c: 0.9782 - val_loss: 0.0661 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0753 - ac
c: 0.9775 - val_loss: 0.0642 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0719 - ac
c: 0.9784 - val_loss: 0.0645 - val_acc: 0.9819
```

In [23]:

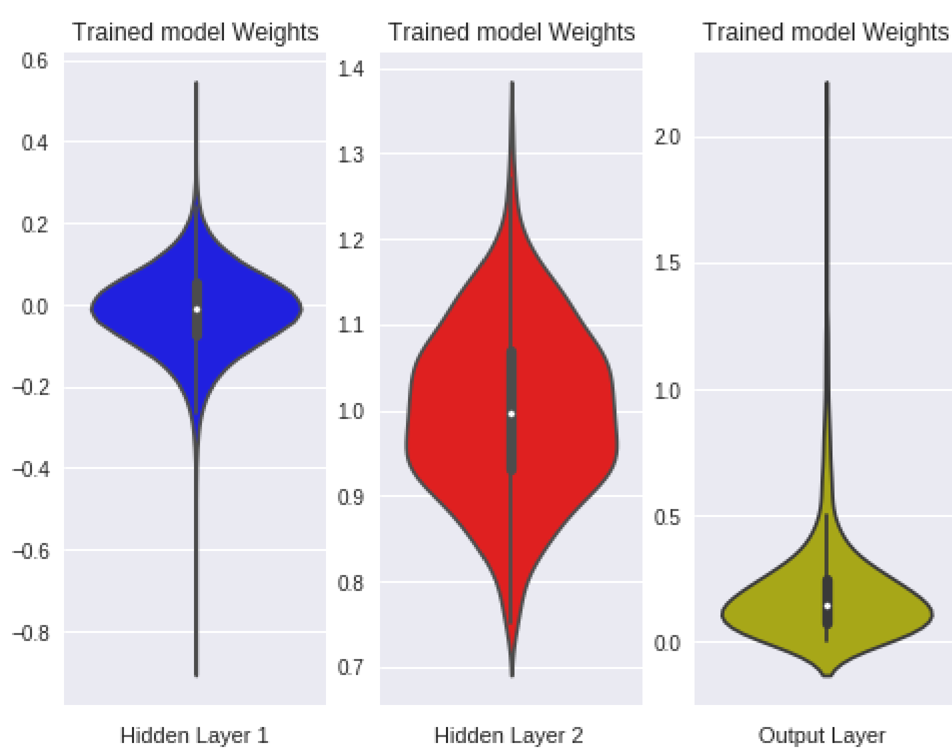
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.06447895890461514
Test accuracy: 0.9819
```



In [24]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



[2] MLP with 3 hidden layers

[2.1] Without Batch Normalization and Dropout

In [25]:
```
# Multilayer perceptron

model = Sequential()
model.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(Dense(196, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(98, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 392)               307720
_________________________________________________________________
dense_11 (Dense)             (None, 196)               77028
_________________________________________________________________
dense_12 (Dense)             (None, 98)                19306
_________________________________________________________________
dense_13 (Dense)             (None, 10)                990
=================================================================
Total params: 405,044
Trainable params: 405,044
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

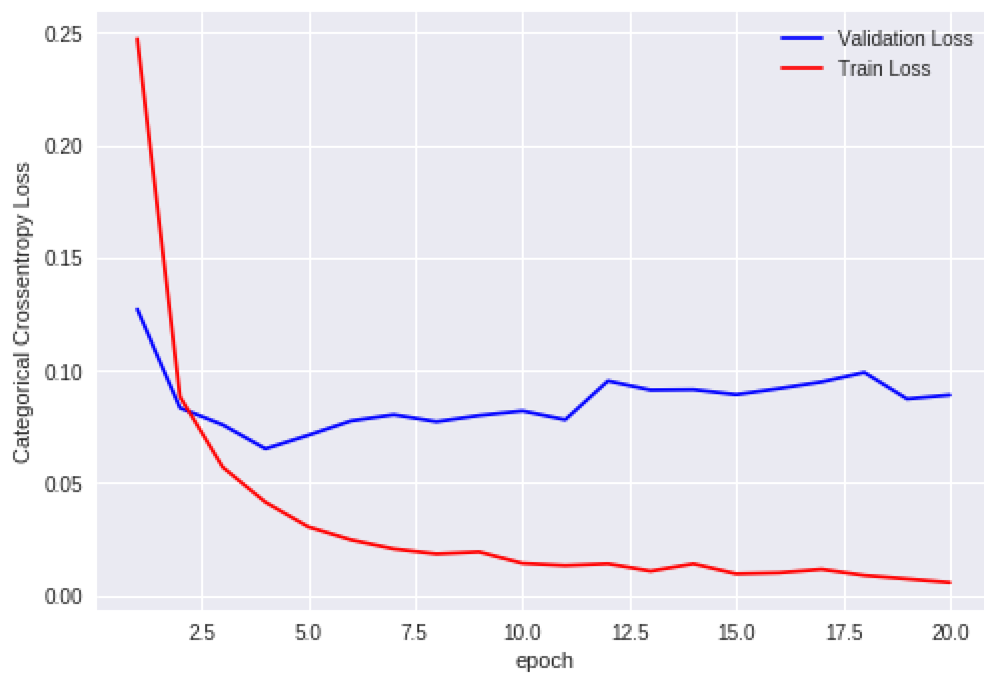
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.2474 - a
cc: 0.9268 - val_loss: 0.1271 - val_acc: 0.9592
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0884 - ac
c: 0.9727 - val_loss: 0.0834 - val_acc: 0.9732
Epoch 3/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0570 - ac
c: 0.9820 - val_loss: 0.0759 - val_acc: 0.9760
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0415 - ac
c: 0.9869 - val_loss: 0.0653 - val_acc: 0.9793
Epoch 5/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0305 - ac
c: 0.9900 - val_loss: 0.0713 - val_acc: 0.9800
Epoch 6/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0247 - ac
c: 0.9919 - val_loss: 0.0776 - val_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0207 - ac
c: 0.9929 - val_loss: 0.0804 - val_acc: 0.9792
Epoch 8/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0184 - ac
c: 0.9938 - val_loss: 0.0773 - val_acc: 0.9804
Epoch 9/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0194 - ac
c: 0.9934 - val_loss: 0.0800 - val_acc: 0.9778
Epoch 10/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0143 - ac
c: 0.9951 - val_loss: 0.0821 - val_acc: 0.9789
Epoch 11/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0133 - ac
c: 0.9956 - val_loss: 0.0781 - val_acc: 0.9820
Epoch 12/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0141 - ac
c: 0.9951 - val_loss: 0.0954 - val_acc: 0.9775
Epoch 13/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0108 - ac
c: 0.9967 - val_loss: 0.0913 - val_acc: 0.9775
Epoch 14/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0141 - ac
c: 0.9954 - val_loss: 0.0915 - val_acc: 0.9779
Epoch 15/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0096 - ac
c: 0.9968 - val_loss: 0.0893 - val_acc: 0.9795
Epoch 16/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0101 - ac
c: 0.9970 - val_loss: 0.0920 - val_acc: 0.9800
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0116 - ac
c: 0.9965 - val_loss: 0.0950 - val_acc: 0.9808
Epoch 18/20
```

```
60000/60000 [==============================] - 6s 96us/step - loss: 0.0089 - ac
c: 0.9972 - val_loss: 0.0992 - val_acc: 0.9789
Epoch 19/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0073 - ac
c: 0.9976 - val_loss: 0.0874 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0057 - ac
c: 0.9982 - val_loss: 0.0892 - val_acc: 0.9828
```

In [27]:

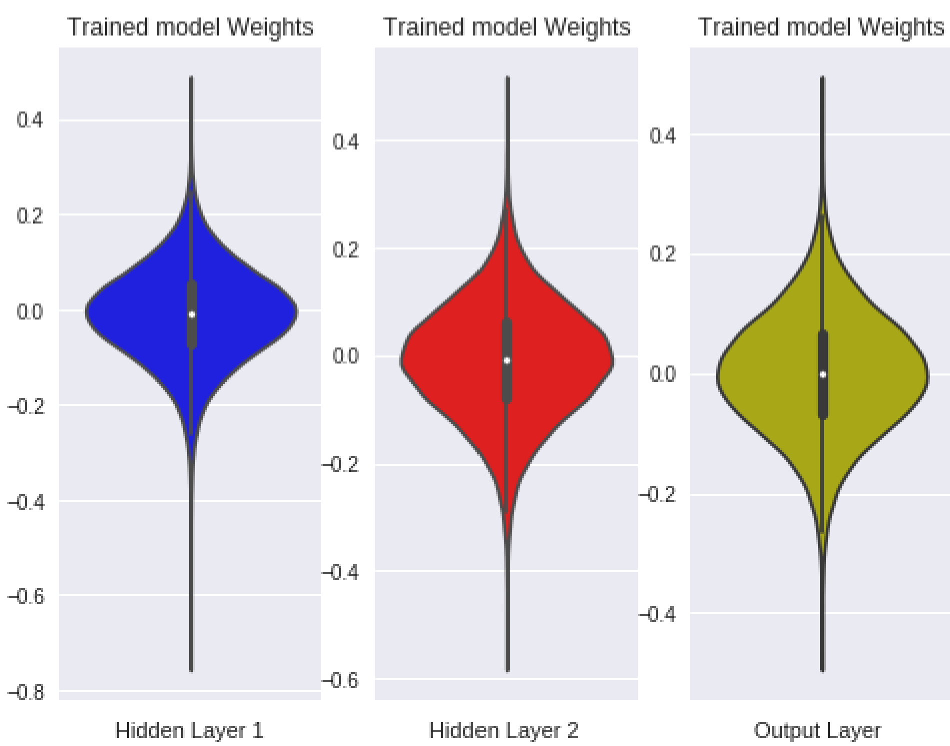
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.08915272126807804
Test accuracy: 0.9828
```



In [28]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



[2.2] With Batch Normalization and Dropout

In [29]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model = Sequential()
model.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(52, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 364)               285740    
_________________________________________________________________
batch_normalization_3 (Batch (None, 364)               1456      
_________________________________________________________________
dropout_3 (Dropout)          (None, 364)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 52)                18980     
_________________________________________________________________
batch_normalization_4 (Batch (None, 52)                208       
_________________________________________________________________
dropout_4 (Dropout)          (None, 52)                0         
_________________________________________________________________
dense_16 (Dense)             (None, 64)                3392      
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)                256       
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_17 (Dense)             (None, 10)                650       
=================================================================
Total params: 310,682
Trainable params: 309,722
Non-trainable params: 960
_________________________________________________________________
```

In [31]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

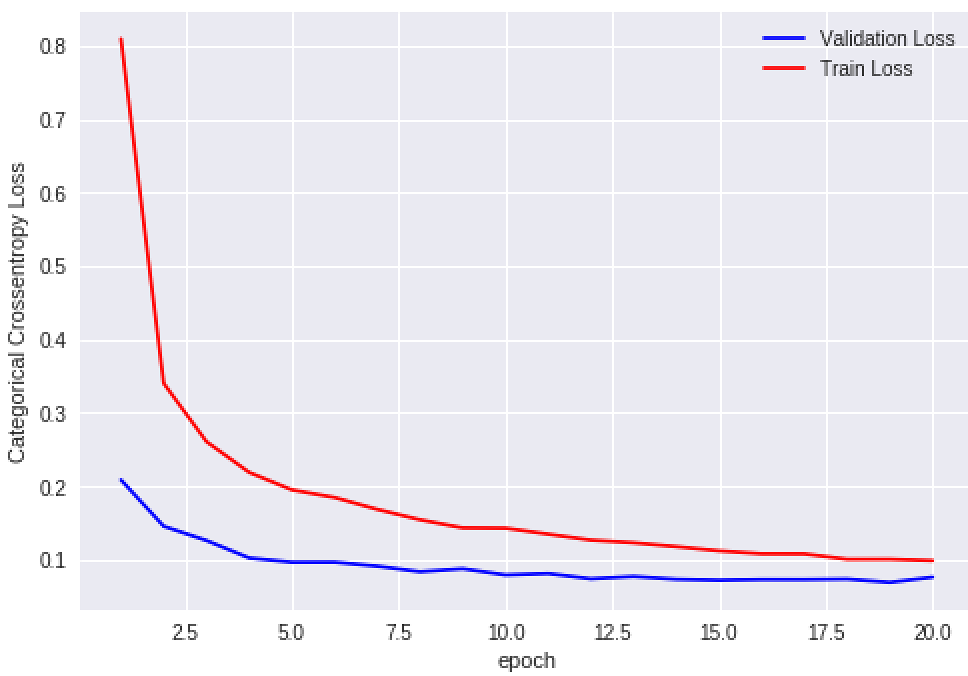
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.8095 - a
cc: 0.7476 - val_loss: 0.2088 - val_acc: 0.9344
Epoch 2/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.3403 - a
cc: 0.9033 - val_loss: 0.1457 - val_acc: 0.9562
Epoch 3/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.2610 - a
cc: 0.9285 - val_loss: 0.1263 - val_acc: 0.9634
Epoch 4/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.2190 - a
cc: 0.9390 - val_loss: 0.1025 - val_acc: 0.9711
Epoch 5/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1953 - a
cc: 0.9472 - val_loss: 0.0967 - val_acc: 0.9721
Epoch 6/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1848 - a
cc: 0.9492 - val_loss: 0.0968 - val_acc: 0.9724
Epoch 7/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1687 - a
cc: 0.9546 - val_loss: 0.0914 - val_acc: 0.9737
Epoch 8/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1545 - a
cc: 0.9578 - val_loss: 0.0838 - val_acc: 0.9762
Epoch 9/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1435 - a
cc: 0.9606 - val_loss: 0.0879 - val_acc: 0.9756
Epoch 10/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1431 - a
cc: 0.9607 - val_loss: 0.0794 - val_acc: 0.9787
Epoch 11/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.1350 - a
cc: 0.9622 - val_loss: 0.0813 - val_acc: 0.9769
Epoch 12/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.1268 - a
cc: 0.9652 - val_loss: 0.0741 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1232 - a
cc: 0.9660 - val_loss: 0.0773 - val_acc: 0.9792
Epoch 14/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.1180 - a
cc: 0.9678 - val_loss: 0.0735 - val_acc: 0.9793
Epoch 15/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.1122 - a
cc: 0.9694 - val_loss: 0.0723 - val_acc: 0.9808
Epoch 16/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.1082 - a
cc: 0.9704 - val_loss: 0.0732 - val_acc: 0.9804
Epoch 17/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.1081 - a
cc: 0.9701 - val_loss: 0.0731 - val_acc: 0.9805
Epoch 18/20
```

```
60000/60000 [==============================] - 7s 108us/step - loss: 0.1008 - a
cc: 0.9727 - val_loss: 0.0738 - val_acc: 0.9801
Epoch 19/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1008 - a
cc: 0.9720 - val_loss: 0.0692 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0989 - a
cc: 0.9724 - val_loss: 0.0764 - val_acc: 0.9795
```

In [32]:

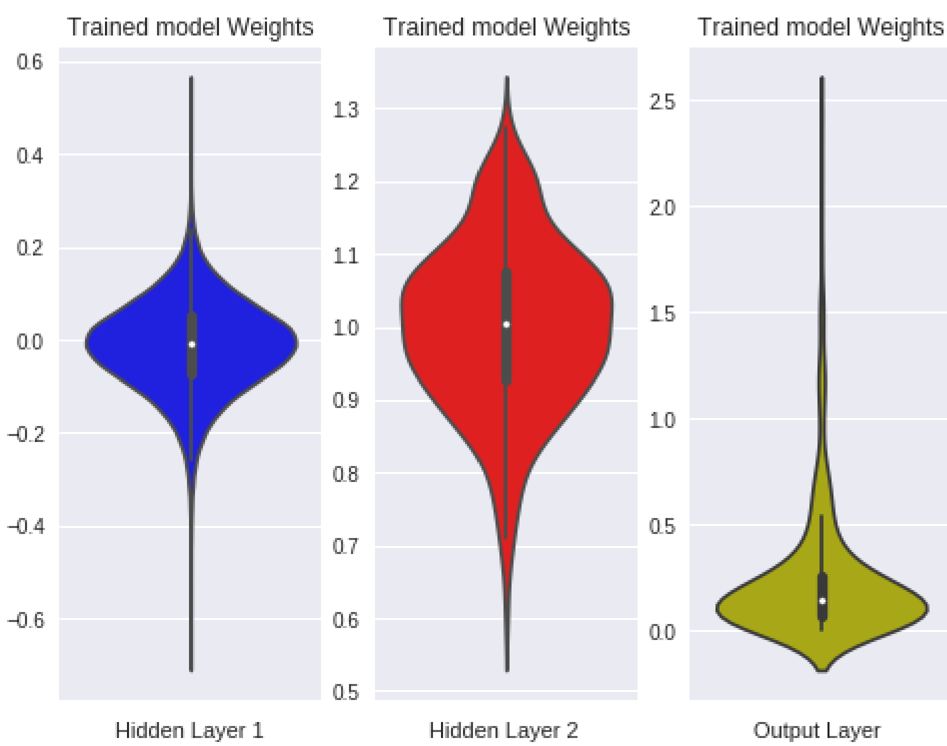
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.07640664648028324
Test accuracy: 0.9795
```



In [33]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



[2] MLP with 5 hidden layers

[3.1] Without Batch Normalization and Dropout

In [34]:
```
# Multilayer perceptron

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
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, 64)                8256
_________________________________________________________________
dense_22 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_23 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

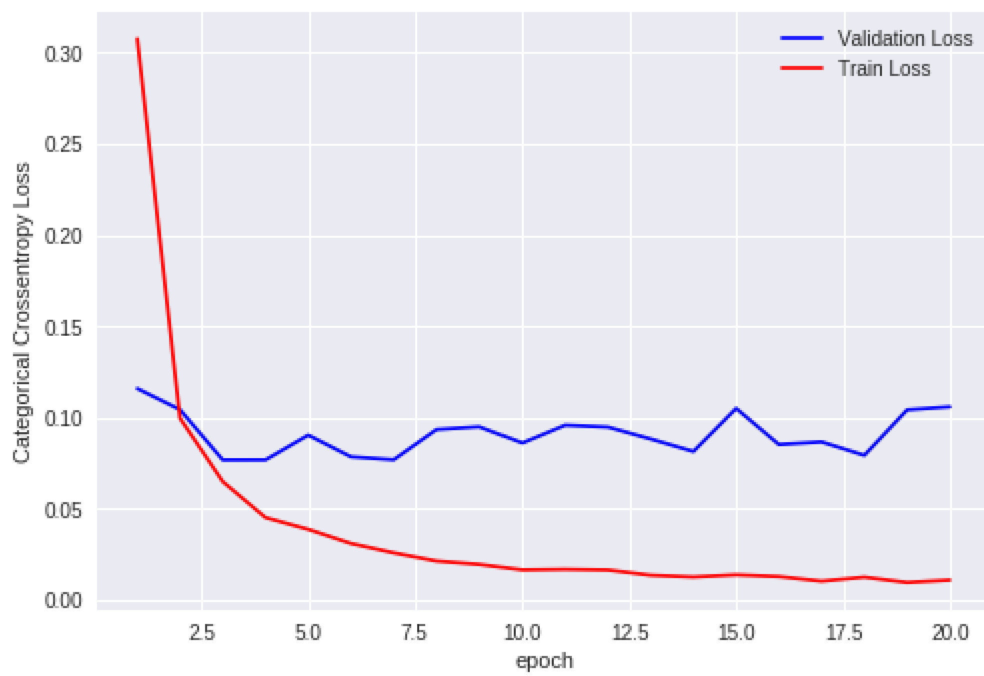
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.3075 - a
cc: 0.9062 - val_loss: 0.1160 - val_acc: 0.9641
Epoch 2/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0998 - a
cc: 0.9698 - val_loss: 0.1045 - val_acc: 0.9681
Epoch 3/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0650 - a
cc: 0.9800 - val_loss: 0.0769 - val_acc: 0.9760
Epoch 4/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0453 - a
cc: 0.9860 - val_loss: 0.0769 - val_acc: 0.9772
Epoch 5/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0388 - a
cc: 0.9875 - val_loss: 0.0905 - val_acc: 0.9762
Epoch 6/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0311 - a
cc: 0.9901 - val_loss: 0.0786 - val_acc: 0.9794
Epoch 7/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0260 - a
cc: 0.9920 - val_loss: 0.0770 - val_acc: 0.9793
Epoch 8/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0215 - a
cc: 0.9934 - val_loss: 0.0936 - val_acc: 0.9768
Epoch 9/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0196 - a
cc: 0.9937 - val_loss: 0.0951 - val_acc: 0.9768
Epoch 10/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0166 - a
cc: 0.9945 - val_loss: 0.0862 - val_acc: 0.9799
Epoch 11/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0169 - a
cc: 0.9948 - val_loss: 0.0959 - val_acc: 0.9774
Epoch 12/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0166 - a
cc: 0.9948 - val_loss: 0.0949 - val_acc: 0.9774
Epoch 13/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0137 - a
cc: 0.9956 - val_loss: 0.0884 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0127 - a
cc: 0.9960 - val_loss: 0.0816 - val_acc: 0.9808
Epoch 15/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0139 - a
cc: 0.9958 - val_loss: 0.1052 - val_acc: 0.9788
Epoch 16/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0130 - a
cc: 0.9963 - val_loss: 0.0854 - val_acc: 0.9832
Epoch 17/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0104 - a
cc: 0.9971 - val_loss: 0.0868 - val_acc: 0.9818
Epoch 18/20
```

```
60000/60000 [==============================] - 8s 133us/step - loss: 0.0126 - a
cc: 0.9963 - val_loss: 0.0794 - val_acc: 0.9822
Epoch 19/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0097 - a
cc: 0.9973 - val_loss: 0.1044 - val_acc: 0.9797
Epoch 20/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0110 - a
cc: 0.9968 - val_loss: 0.1060 - val_acc: 0.9785
```

In [36]:

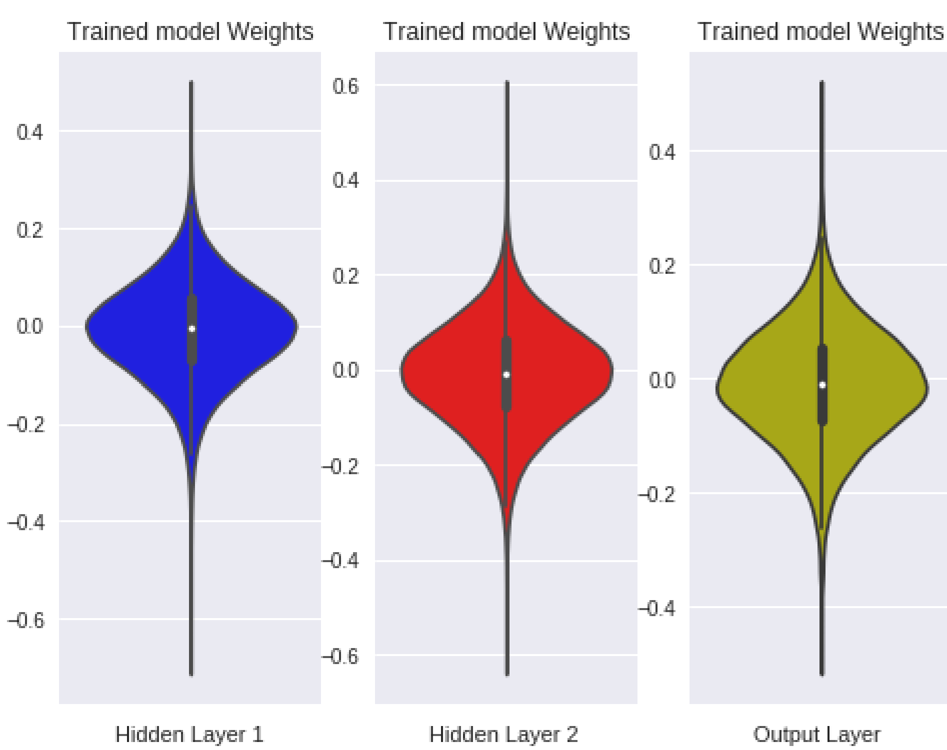
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.10602583270173854
Test accuracy: 0.9785
```



In [37]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



[3.2] With Batch Normalization and Dropout

In [38]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(output_dim, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_24 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_25 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_26 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512       
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256       
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_28 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128       
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0         
_________________________________________________________________
```

```
dense_29 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
```

In [39]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

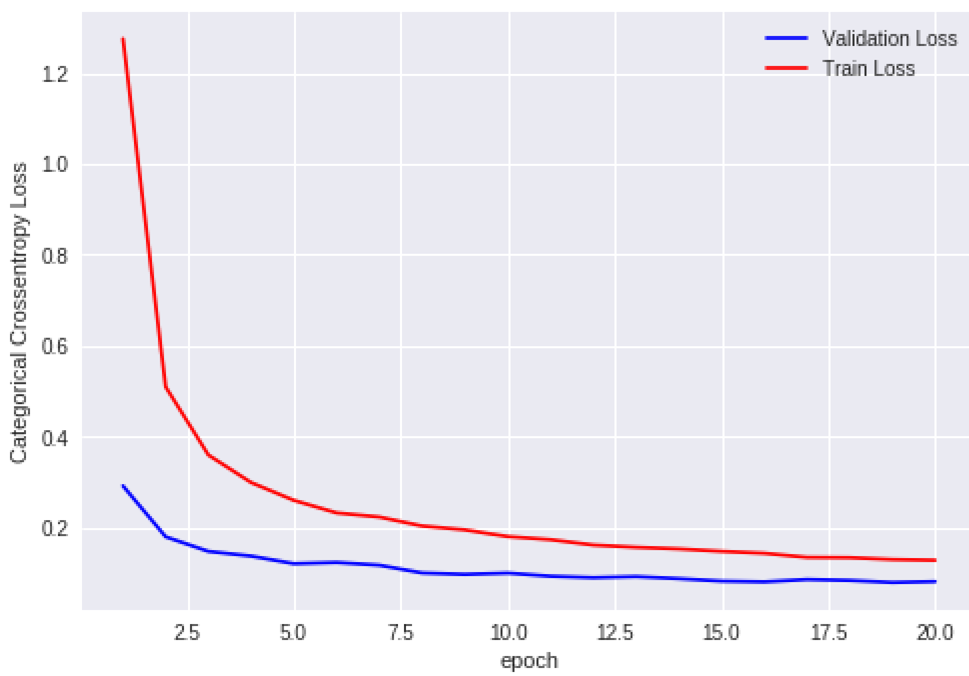
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 224us/step - loss: 1.2762 -
acc: 0.5948 - val_loss: 0.2912 - val_acc: 0.9224
Epoch 2/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.5094 -
acc: 0.8618 - val_loss: 0.1788 - val_acc: 0.9523
Epoch 3/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.3592 -
acc: 0.9095 - val_loss: 0.1466 - val_acc: 0.9610
Epoch 4/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.2985 -
acc: 0.9285 - val_loss: 0.1363 - val_acc: 0.9660
Epoch 5/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.2592 -
acc: 0.9387 - val_loss: 0.1195 - val_acc: 0.9697
Epoch 6/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.2315 -
acc: 0.9456 - val_loss: 0.1222 - val_acc: 0.9695
Epoch 7/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.2225 -
acc: 0.9487 - val_loss: 0.1162 - val_acc: 0.9709
Epoch 8/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.2027 -
acc: 0.9530 - val_loss: 0.0990 - val_acc: 0.9757
Epoch 9/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.1941 -
acc: 0.9557 - val_loss: 0.0959 - val_acc: 0.9758
Epoch 10/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.1793 -
acc: 0.9581 - val_loss: 0.0988 - val_acc: 0.9770
Epoch 11/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1725 -
acc: 0.9602 - val_loss: 0.0918 - val_acc: 0.9790
Epoch 12/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1606 -
acc: 0.9622 - val_loss: 0.0887 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.1556 -
acc: 0.9647 - val_loss: 0.0910 - val_acc: 0.9787
Epoch 14/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.1521 -
acc: 0.9651 - val_loss: 0.0865 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1467 -
acc: 0.9675 - val_loss: 0.0810 - val_acc: 0.9808
Epoch 16/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1424 -
acc: 0.9677 - val_loss: 0.0794 - val_acc: 0.9807
Epoch 17/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.1334 -
acc: 0.9695 - val_loss: 0.0847 - val_acc: 0.9816
Epoch 18/20
```

```
60000/60000 [==============================] - 11s 190us/step - loss: 0.1328 - acc: 0.9700 - val_loss: 0.0826 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.1287 - acc: 0.9709 - val_loss: 0.0780 - val_acc: 0.9822
Epoch 20/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.1271 - acc: 0.9715 - val_loss: 0.0801 - val_acc: 0.9819
```

In [40]:

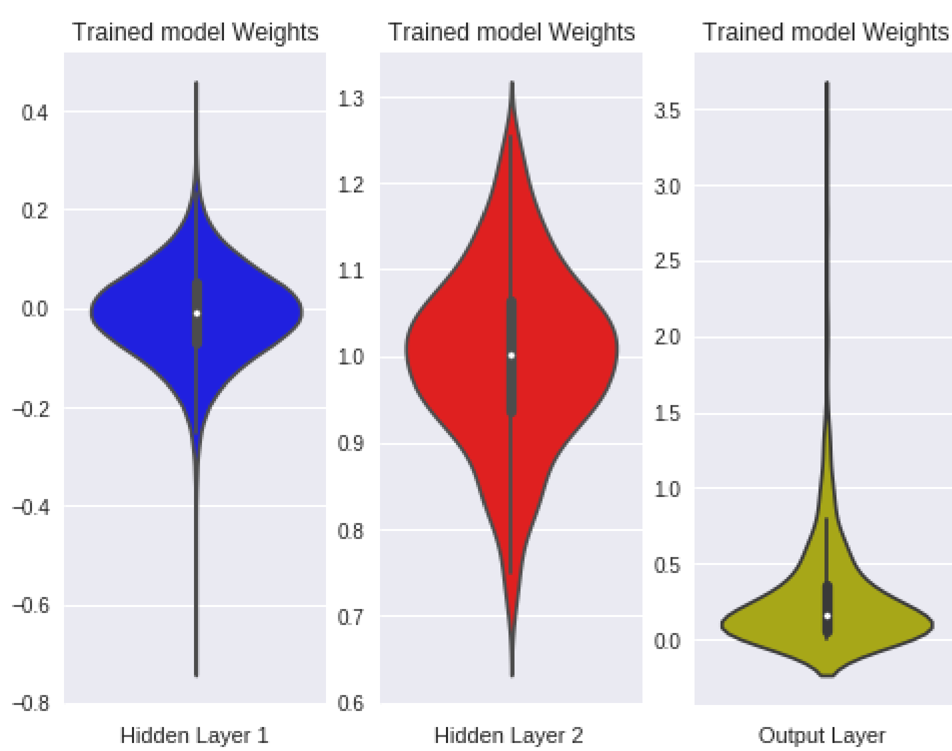
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.080058406015974447
Test accuracy: 0.9819
```



In [41]:
```
w_after = model.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=(8, 6))
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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarnin
g: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



In [47]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["MLP_MODEL", "TRAIN_ACCURACY", "TEST_ACCURACY"]
x.add_row(["MLP(2-hidden layers)", 0.870, 0.875])
x.add_row(["MLP(2-hidden layers) With Dropout and Batch Normalization", 0.978, 0.
x.add_row(["MLP(3-hidden layers)", 0.998, 0.982])
x.add_row(["MLP(3-hidden layers) With Dropout and Batch Normalization", 0.972, 0.
x.add_row(["MLP(5-hidden layers)", 0.996, 0.978])
x.add_row(["MLP(5-hidden layers) With Dropout and Batch Normalization", 0.971, 0.

print('\t\t\t\tMLP WITH DIFFERNET ARCHITECTURES')
print(x)
```

```
                                MLP WITH DIFFERNET ARCHITECTURES
+----------------------------------------------------------+----------------+-
--------------+
|                        MLP_MODEL                         | TRAIN_ACCURACY |
TEST_ACCURACY |
+----------------------------------------------------------+----------------+-
--------------+
|                   MLP(2-hidden layers)                   |      0.87      |
    0.875     |
| MLP(2-hidden layers) With Dropout and Batch Normalization |     0.978      |
    0.981     |
|                   MLP(3-hidden layers)                   |     0.998      |
    0.982     |
| MLP(3-hidden layers) With Dropout and Batch Normalization |     0.972      |
    0.979     |
|                   MLP(5-hidden layers)                   |     0.996      |
    0.978     |
| MLP(5-hidden layers) With Dropout and Batch Normalization |     0.971      |
    0.981     |
+----------------------------------------------------------+----------------+-
--------------+
```

Conclusion

1. The Best train and test (0.998 | 0.982) accuracy is obtained by MLP(3-hidden layers) without Dropout and Batch Normalization.
2. With Dropout and Batch Normalization the NN is less overfitted than the NN without Dropout and Batch Normalization.