In [3]:
```
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
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()
```

In [5]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

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

In [6]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of
 shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image is of
 shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

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

In [0]:
```
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 [8]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of
 shape (%d)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image is of
 shape (%d)"%(X_test.shape[1]))
```

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

In [9]:
```
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]:
```
X_train = X_train/255
X_test = X_test/255
```

In [11]:
```
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.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         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 [12]:
```
print("Class label of first image :", y_train[0])
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
from keras.initializers import he_normal
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
```

In [0]:
```
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

MLP + ReLu + Adam + BN + Dropout (2 Layer Architecture 784-168-472-10)

In [15]:
```
model = Sequential()
model.add(Dense(168, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(472, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/pyt
hon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.fr
amework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_op
s) 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 - k
eep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 168)               131880
_________________________________________________________________
batch_normalization_1 (Batch (None, 168)               672
_________________________________________________________________
dropout_1 (Dropout)          (None, 168)               0
_________________________________________________________________
dense_2 (Dense)              (None, 472)               79768
_________________________________________________________________
batch_normalization_2 (Batch (None, 472)               1888
_________________________________________________________________
dropout_2 (Dropout)          (None, 472)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                4730
=================================================================
Total params: 218,938
Trainable params: 217,658
Non-trainable params: 1,280
_________________________________________________________________
```

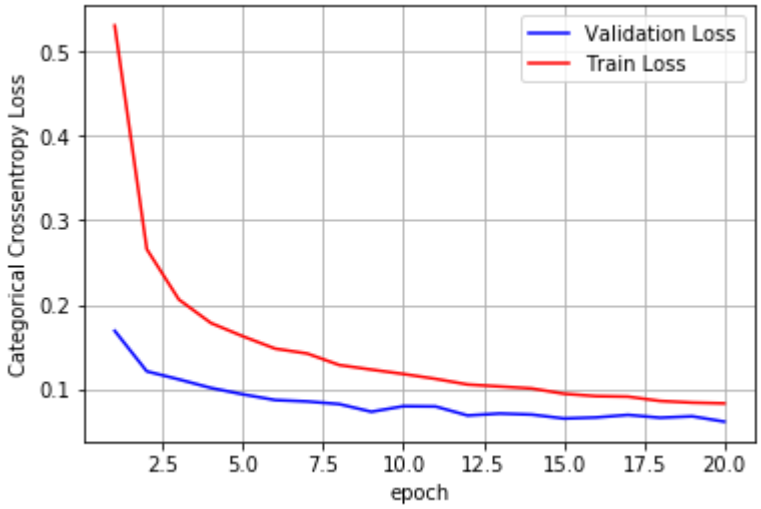
In [16]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history12 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/pyt
hon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is d
eprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.5309 -
acc: 0.8404 - val_loss: 0.1689 - val_acc: 0.9465
Epoch 2/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.2657 -
acc: 0.9189 - val_loss: 0.1209 - val_acc: 0.9628
Epoch 3/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2060 -
acc: 0.9380 - val_loss: 0.1113 - val_acc: 0.9667
Epoch 4/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.1781 -
acc: 0.9449 - val_loss: 0.1012 - val_acc: 0.9698
Epoch 5/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1624 -
acc: 0.9501 - val_loss: 0.0936 - val_acc: 0.9709
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1478 -
acc: 0.9546 - val_loss: 0.0869 - val_acc: 0.9730
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1420 -
acc: 0.9557 - val_loss: 0.0852 - val_acc: 0.9741
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1284 -
acc: 0.9599 - val_loss: 0.0820 - val_acc: 0.9747
Epoch 9/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1229 -
acc: 0.9618 - val_loss: 0.0730 - val_acc: 0.9766
Epoch 10/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1177 -
acc: 0.9638 - val_loss: 0.0796 - val_acc: 0.9757
Epoch 11/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1120 -
acc: 0.9653 - val_loss: 0.0792 - val_acc: 0.9762
Epoch 12/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1053 -
acc: 0.9672 - val_loss: 0.0685 - val_acc: 0.9776
Epoch 13/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1030 -
acc: 0.9673 - val_loss: 0.0709 - val_acc: 0.9785
Epoch 14/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1006 -
acc: 0.9684 - val_loss: 0.0696 - val_acc: 0.9783
Epoch 15/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0943 -
acc: 0.9697 - val_loss: 0.0650 - val_acc: 0.9794
Epoch 16/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0915 -
acc: 0.9707 - val_loss: 0.0662 - val_acc: 0.9789
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0909 -
acc: 0.9715 - val_loss: 0.0692 - val_acc: 0.9799
```

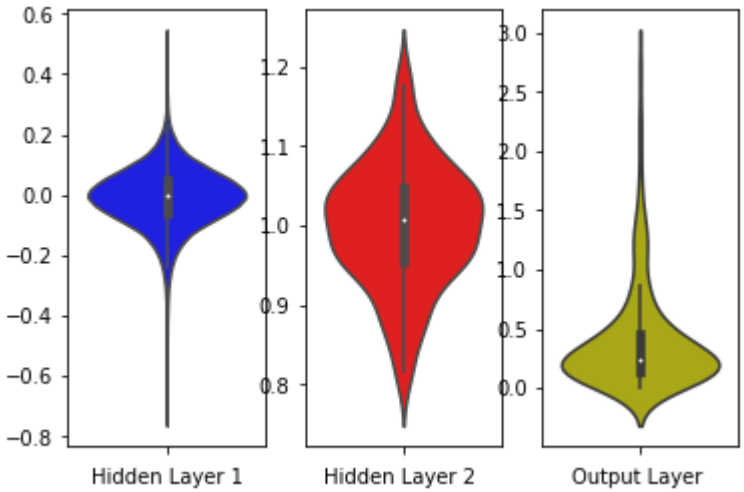
```
Epoch 18/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0857 -
acc: 0.9733 - val_loss: 0.0659 - val_acc: 0.9798
Epoch 19/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0838 -
acc: 0.9733 - val_loss: 0.0676 - val_acc: 0.9797
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0829 -
acc: 0.9736 - val_loss: 0.0613 - val_acc: 0.9811
```

In [17]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history12.history['val_loss']
ty = history12.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06125014309256803
Test accuracy: 0.9811
```



In [18]:
```
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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



In [19]:
```
model = Sequential()
model.add(Dense(168, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(472, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 168)               131880
_________________________________________________________________
batch_normalization_3 (Batch (None, 168)               672
_________________________________________________________________
dropout_3 (Dropout)          (None, 168)               0
_________________________________________________________________
dense_5 (Dense)              (None, 472)               79768
_________________________________________________________________
batch_normalization_4 (Batch (None, 472)               1888
_________________________________________________________________
dropout_4 (Dropout)          (None, 472)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                4730
=================================================================
Total params: 218,938
Trainable params: 217,658
Non-trainable params: 1,280
_________________________________________________________________
```

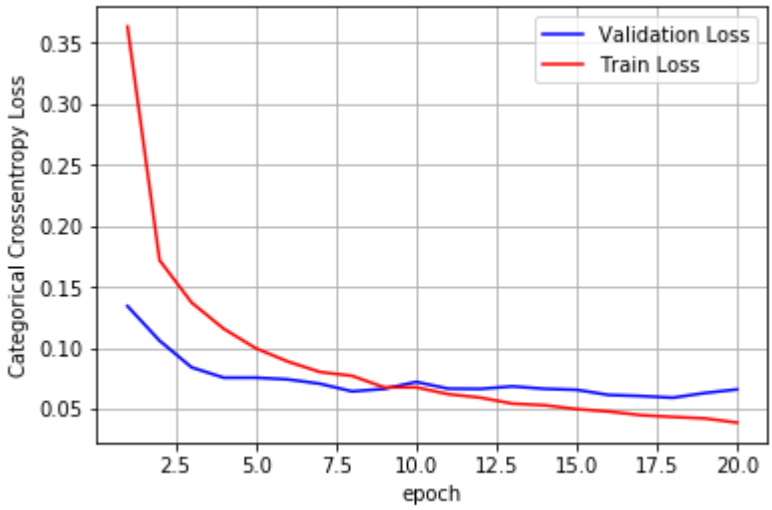
In [20]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc
uracy'])
history12 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch
, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.3632 -
acc: 0.8885 - val_loss: 0.1343 - val_acc: 0.9599
Epoch 2/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1718 -
acc: 0.9475 - val_loss: 0.1060 - val_acc: 0.9678
Epoch 3/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1371 -
acc: 0.9572 - val_loss: 0.0841 - val_acc: 0.9740
Epoch 4/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1157 -
acc: 0.9639 - val_loss: 0.0757 - val_acc: 0.9753
Epoch 5/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0997 -
acc: 0.9682 - val_loss: 0.0757 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0889 -
acc: 0.9715 - val_loss: 0.0743 - val_acc: 0.9754
Epoch 7/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0803 -
acc: 0.9745 - val_loss: 0.0707 - val_acc: 0.9774
Epoch 8/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0772 -
acc: 0.9740 - val_loss: 0.0645 - val_acc: 0.9800
Epoch 9/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0678 -
acc: 0.9782 - val_loss: 0.0665 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0678 -
acc: 0.9775 - val_loss: 0.0722 - val_acc: 0.9779
Epoch 11/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0622 -
acc: 0.9795 - val_loss: 0.0667 - val_acc: 0.9799
Epoch 12/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0593 -
acc: 0.9805 - val_loss: 0.0665 - val_acc: 0.9800
Epoch 13/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0544 -
acc: 0.9821 - val_loss: 0.0686 - val_acc: 0.9793
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0531 -
acc: 0.9823 - val_loss: 0.0666 - val_acc: 0.9802
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0499 -
acc: 0.9831 - val_loss: 0.0657 - val_acc: 0.9801
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0480 -
acc: 0.9837 - val_loss: 0.0615 - val_acc: 0.9814
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0450 -
acc: 0.9850 - val_loss: 0.0607 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0435 -
acc: 0.9857 - val_loss: 0.0593 - val_acc: 0.9822
Epoch 19/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0423 -
```

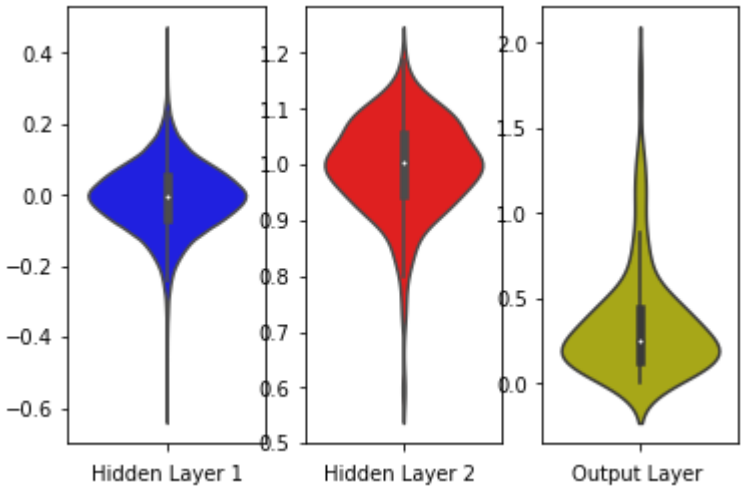
```
acc: 0.9859 - val_loss: 0.0631 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0389 -
acc: 0.9863 - val_loss: 0.0660 - val_acc: 0.9815
```

In [21]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history12.history['val_loss']
ty = history12.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06603560918954463
Test accuracy: 0.9815
```



In [22]:
```
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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



In [23]:
```
model = Sequential()
model.add(Dense(168, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(472, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 168)               131880
_________________________________________________________________
batch_normalization_5 (Batch (None, 168)               672
_________________________________________________________________
dropout_5 (Dropout)          (None, 168)               0
_________________________________________________________________
dense_8 (Dense)              (None, 472)               79768
_________________________________________________________________
batch_normalization_6 (Batch (None, 472)               1888
_________________________________________________________________
dropout_6 (Dropout)          (None, 472)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                4730
=================================================================
Total params: 218,938
Trainable params: 217,658
Non-trainable params: 1,280
_________________________________________________________________
```

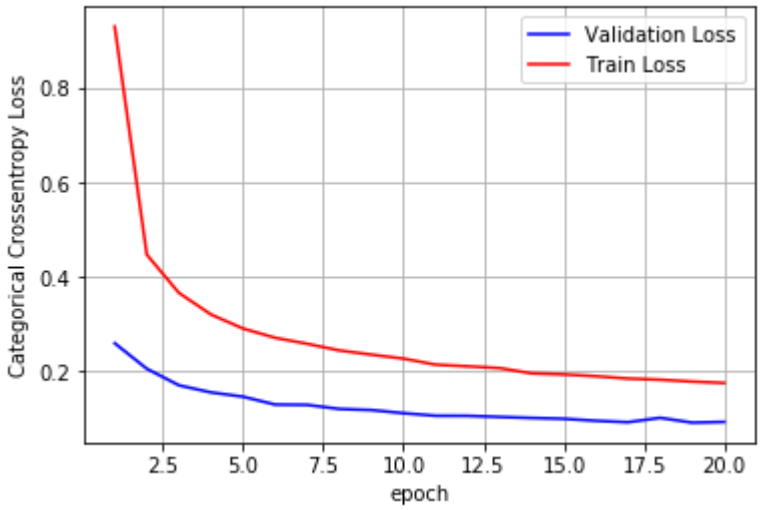
In [24]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc
uracy'])
history12 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch
, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.9308 -
acc: 0.7284 - val_loss: 0.2600 - val_acc: 0.9229
Epoch 2/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.4476 -
acc: 0.8645 - val_loss: 0.2060 - val_acc: 0.9379
Epoch 3/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.3667 -
acc: 0.8892 - val_loss: 0.1710 - val_acc: 0.9472
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.3209 -
acc: 0.9027 - val_loss: 0.1559 - val_acc: 0.9508
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2912 -
acc: 0.9120 - val_loss: 0.1468 - val_acc: 0.9554
Epoch 6/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2716 -
acc: 0.9183 - val_loss: 0.1301 - val_acc: 0.9608
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.2589 -
acc: 0.9222 - val_loss: 0.1295 - val_acc: 0.9595
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2448 -
acc: 0.9268 - val_loss: 0.1209 - val_acc: 0.9623
Epoch 9/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.2359 -
acc: 0.9277 - val_loss: 0.1185 - val_acc: 0.9651
Epoch 10/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.2275 -
acc: 0.9317 - val_loss: 0.1119 - val_acc: 0.9653
Epoch 11/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2147 -
acc: 0.9349 - val_loss: 0.1065 - val_acc: 0.9664
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2112 -
acc: 0.9364 - val_loss: 0.1063 - val_acc: 0.9669
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2074 -
acc: 0.9381 - val_loss: 0.1039 - val_acc: 0.9675
Epoch 14/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1963 -
acc: 0.9413 - val_loss: 0.1018 - val_acc: 0.9687
Epoch 15/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1940 -
acc: 0.9415 - val_loss: 0.1000 - val_acc: 0.9695
Epoch 16/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.1902 -
acc: 0.9430 - val_loss: 0.0958 - val_acc: 0.9715
Epoch 17/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1853 -
acc: 0.9438 - val_loss: 0.0928 - val_acc: 0.9712
Epoch 18/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1828 -
acc: 0.9452 - val_loss: 0.1021 - val_acc: 0.9702
Epoch 19/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1788 -
```

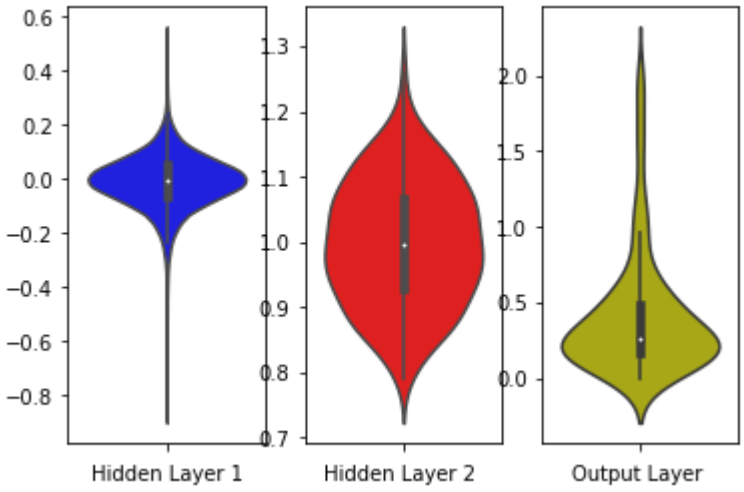
```
acc: 0.9452 - val_loss: 0.0919 - val_acc: 0.9728
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1759 -
acc: 0.9467 - val_loss: 0.0934 - val_acc: 0.9734
```

In [25]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history12.history['val_loss']
ty = history12.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.09341368079553358
Test accuracy: 0.9734
```



In [26]:
```
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()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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



MLP + ReLu + Adam + BN + Dropout (3 Layer Architecture 784-352-164-124-10)

In [27]:
```
model=Sequential()
model.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(164, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 352)               276320
_________________________________________________________________
batch_normalization_7 (Batch (None, 352)               1408
_________________________________________________________________
dropout_7 (Dropout)          (None, 352)               0
_________________________________________________________________
dense_11 (Dense)             (None, 164)               57892
_________________________________________________________________
batch_normalization_8 (Batch (None, 164)               656
_________________________________________________________________
dropout_8 (Dropout)          (None, 164)               0
_________________________________________________________________
dense_12 (Dense)             (None, 124)               20460
_________________________________________________________________
batch_normalization_9 (Batch (None, 124)               496
_________________________________________________________________
dropout_9 (Dropout)          (None, 124)               0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1250
=================================================================
Total params: 358,482
Trainable params: 357,202
Non-trainable params: 1,280
_________________________________________________________________
```

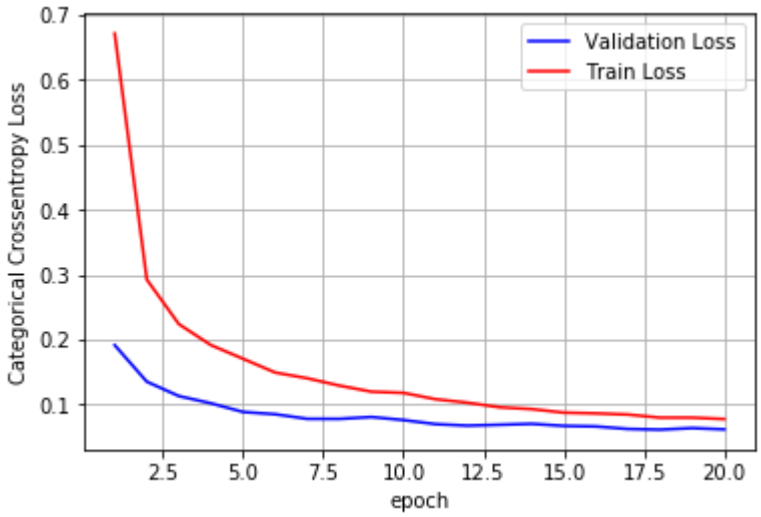
In [28]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.6714 -
acc: 0.7932 - val_loss: 0.1917 - val_acc: 0.9394
Epoch 2/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.2927 -
acc: 0.9146 - val_loss: 0.1355 - val_acc: 0.9571
Epoch 3/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.2240 -
acc: 0.9342 - val_loss: 0.1133 - val_acc: 0.9652
Epoch 4/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1917 -
acc: 0.9447 - val_loss: 0.1022 - val_acc: 0.9695
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.1707 -
acc: 0.9501 - val_loss: 0.0888 - val_acc: 0.9744
Epoch 6/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.1495 -
acc: 0.9568 - val_loss: 0.0853 - val_acc: 0.9756
Epoch 7/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1407 -
acc: 0.9587 - val_loss: 0.0783 - val_acc: 0.9771
Epoch 8/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1294 -
acc: 0.9621 - val_loss: 0.0782 - val_acc: 0.9776
Epoch 9/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1201 -
acc: 0.9643 - val_loss: 0.0811 - val_acc: 0.9775
Epoch 10/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.1182 -
acc: 0.9650 - val_loss: 0.0764 - val_acc: 0.9776
Epoch 11/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.1085 -
acc: 0.9678 - val_loss: 0.0701 - val_acc: 0.9790
Epoch 12/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.1031 -
acc: 0.9701 - val_loss: 0.0678 - val_acc: 0.9802
Epoch 13/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0960 -
acc: 0.9710 - val_loss: 0.0692 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0932 -
acc: 0.9723 - val_loss: 0.0706 - val_acc: 0.9798
Epoch 15/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0878 -
acc: 0.9744 - val_loss: 0.0674 - val_acc: 0.9810
Epoch 16/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0867 -
acc: 0.9739 - val_loss: 0.0665 - val_acc: 0.9812
Epoch 17/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0848 -
acc: 0.9749 - val_loss: 0.0627 - val_acc: 0.9823
Epoch 18/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0801 -
acc: 0.9756 - val_loss: 0.0616 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0801 -
```

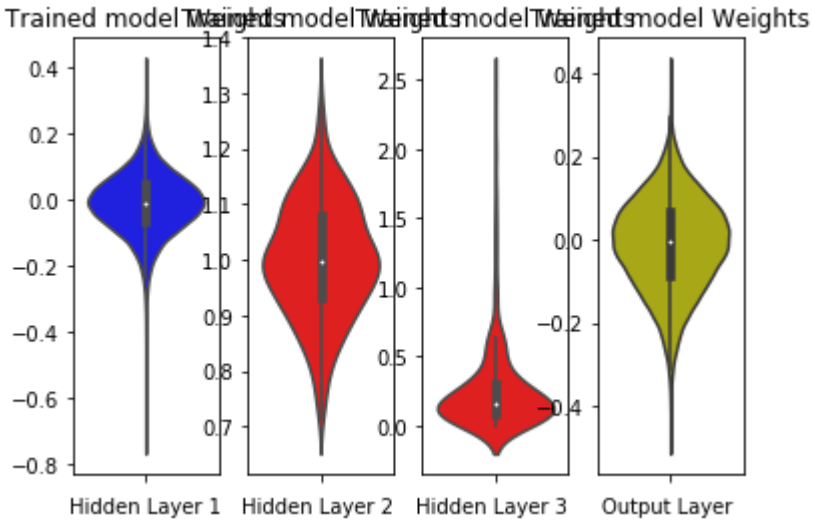
```
acc: 0.9763 - val_loss: 0.0640 - val_acc: 0.9815
Epoch 20/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0779 -
acc: 0.9767 - val_loss: 0.0620 - val_acc: 0.9826
```

In [29]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0620166264391155
Test accuracy: 0.9826
```



In [30]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)


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

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

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

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



In [31]:
```
model=Sequential()
model.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(164, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 352)               276320
_________________________________________________________________
batch_normalization_10 (Batc (None, 352)               1408
_________________________________________________________________
dropout_10 (Dropout)         (None, 352)               0
_________________________________________________________________
dense_15 (Dense)             (None, 164)               57892
_________________________________________________________________
batch_normalization_11 (Batc (None, 164)               656
_________________________________________________________________
dropout_11 (Dropout)         (None, 164)               0
_________________________________________________________________
dense_16 (Dense)             (None, 124)               20460
_________________________________________________________________
batch_normalization_12 (Batc (None, 124)               496
_________________________________________________________________
dropout_12 (Dropout)         (None, 124)               0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1250
=================================================================
Total params: 358,482
Trainable params: 357,202
Non-trainable params: 1,280
_________________________________________________________________
```

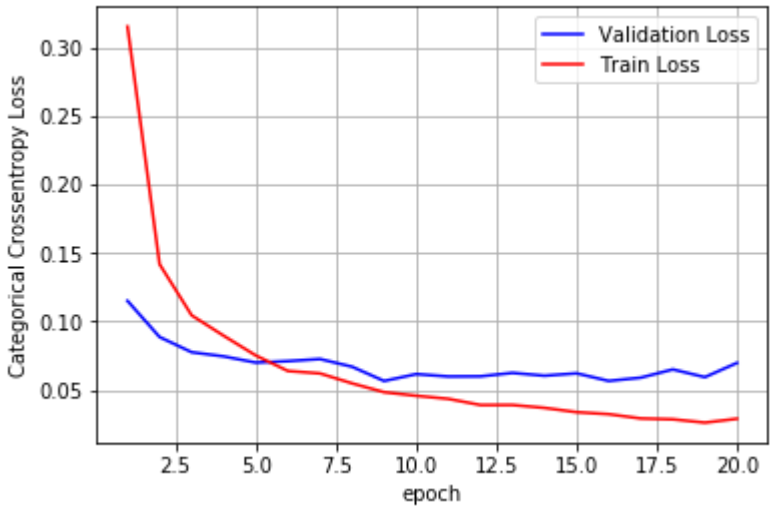
In [32]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.3156 -
acc: 0.9035 - val_loss: 0.1150 - val_acc: 0.9647
Epoch 2/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.1418 -
acc: 0.9563 - val_loss: 0.0886 - val_acc: 0.9722
Epoch 3/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1045 -
acc: 0.9683 - val_loss: 0.0775 - val_acc: 0.9756
Epoch 4/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0895 -
acc: 0.9719 - val_loss: 0.0744 - val_acc: 0.9778
Epoch 5/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0751 -
acc: 0.9763 - val_loss: 0.0699 - val_acc: 0.9778
Epoch 6/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0638 -
acc: 0.9796 - val_loss: 0.0710 - val_acc: 0.9781
Epoch 7/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0620 -
acc: 0.9796 - val_loss: 0.0725 - val_acc: 0.9789
Epoch 8/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0547 -
acc: 0.9819 - val_loss: 0.0669 - val_acc: 0.9786
Epoch 9/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0483 -
acc: 0.9844 - val_loss: 0.0565 - val_acc: 0.9825
Epoch 10/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0456 -
acc: 0.9850 - val_loss: 0.0615 - val_acc: 0.9823
Epoch 11/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0434 -
acc: 0.9860 - val_loss: 0.0597 - val_acc: 0.9822
Epoch 12/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0389 -
acc: 0.9876 - val_loss: 0.0598 - val_acc: 0.9832
Epoch 13/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0389 -
acc: 0.9872 - val_loss: 0.0624 - val_acc: 0.9819
Epoch 14/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0368 -
acc: 0.9879 - val_loss: 0.0603 - val_acc: 0.9818
Epoch 15/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0337 -
acc: 0.9884 - val_loss: 0.0620 - val_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0322 -
acc: 0.9895 - val_loss: 0.0565 - val_acc: 0.9835
Epoch 17/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0290 -
acc: 0.9904 - val_loss: 0.0589 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0284 -
acc: 0.9909 - val_loss: 0.0649 - val_acc: 0.9817
Epoch 19/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0260 -
```

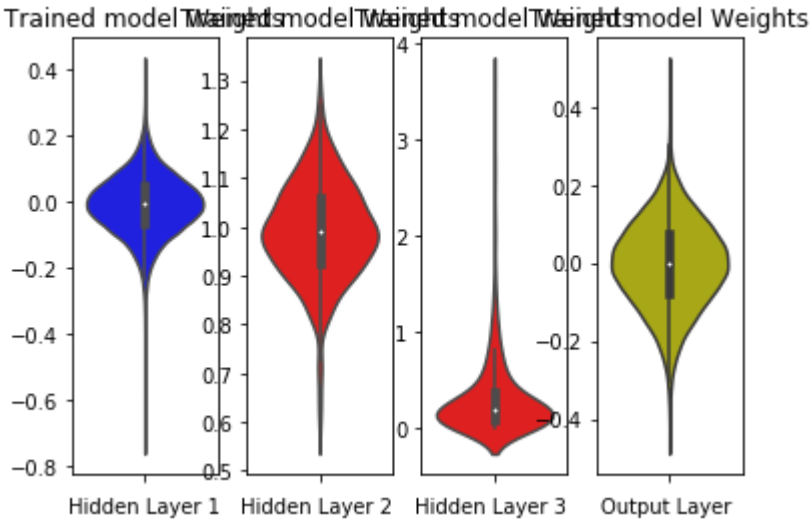
```
acc: 0.9915 - val_loss: 0.0593 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0288 -
acc: 0.9900 - val_loss: 0.0697 - val_acc: 0.9801
```

In [33]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0696625495184242
Test accuracy: 0.9801
```



In [34]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)


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

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

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

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



In [35]:
```
model=Sequential()
model.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(164, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 352)               276320
_________________________________________________________________
batch_normalization_13 (Batc (None, 352)               1408
_________________________________________________________________
dropout_13 (Dropout)         (None, 352)               0
_________________________________________________________________
dense_19 (Dense)             (None, 164)               57892
_________________________________________________________________
batch_normalization_14 (Batc (None, 164)               656
_________________________________________________________________
dropout_14 (Dropout)         (None, 164)               0
_________________________________________________________________
dense_20 (Dense)             (None, 124)               20460
_________________________________________________________________
batch_normalization_15 (Batc (None, 124)               496
_________________________________________________________________
dropout_15 (Dropout)         (None, 124)               0
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1250
=================================================================
Total params: 358,482
Trainable params: 357,202
Non-trainable params: 1,280
_________________________________________________________________
```

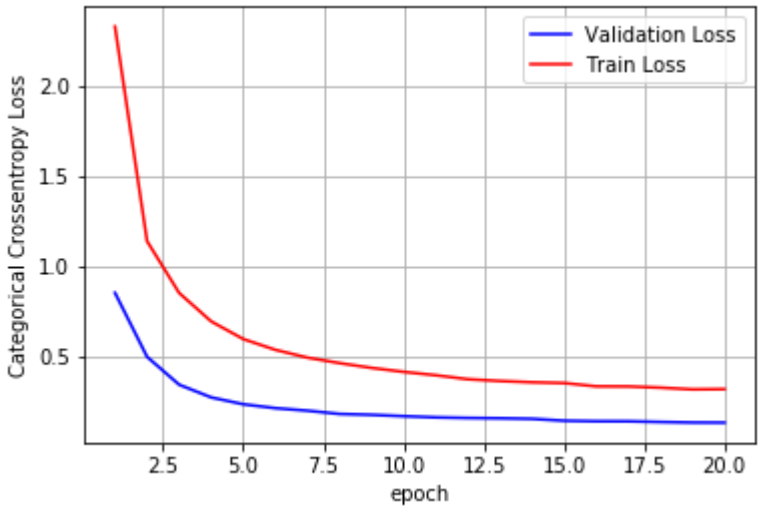
In [36]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 159us/step - loss: 2.3291
- acc: 0.3203 - val_loss: 0.8520 - val_acc: 0.7836
Epoch 2/20
60000/60000 [==============================] - 8s 127us/step - loss: 1.1387 -
acc: 0.5952 - val_loss: 0.4971 - val_acc: 0.8783
Epoch 3/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.8513 -
acc: 0.7194 - val_loss: 0.3424 - val_acc: 0.9115
Epoch 4/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.6934 -
acc: 0.7846 - val_loss: 0.2719 - val_acc: 0.9247
Epoch 5/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.5955 -
acc: 0.8227 - val_loss: 0.2339 - val_acc: 0.9323
Epoch 6/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.5358 -
acc: 0.8453 - val_loss: 0.2122 - val_acc: 0.9393
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.4925 -
acc: 0.8616 - val_loss: 0.1982 - val_acc: 0.9441
Epoch 8/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.4621 -
acc: 0.8727 - val_loss: 0.1801 - val_acc: 0.9483
Epoch 9/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.4356 -
acc: 0.8825 - val_loss: 0.1753 - val_acc: 0.9501
Epoch 10/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.4135 -
acc: 0.8893 - val_loss: 0.1677 - val_acc: 0.9522
Epoch 11/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3948 -
acc: 0.8960 - val_loss: 0.1611 - val_acc: 0.9558
Epoch 12/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.3727 -
acc: 0.9021 - val_loss: 0.1578 - val_acc: 0.9570
Epoch 13/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.3628 -
acc: 0.9047 - val_loss: 0.1560 - val_acc: 0.9564
Epoch 14/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3557 -
acc: 0.9083 - val_loss: 0.1530 - val_acc: 0.9576
Epoch 15/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.3515 -
acc: 0.9090 - val_loss: 0.1425 - val_acc: 0.9610
Epoch 16/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3332 -
acc: 0.9128 - val_loss: 0.1398 - val_acc: 0.9627
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.3325 -
acc: 0.9134 - val_loss: 0.1394 - val_acc: 0.9617
Epoch 18/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3260 -
acc: 0.9153 - val_loss: 0.1355 - val_acc: 0.9635
Epoch 19/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.3165 -
```

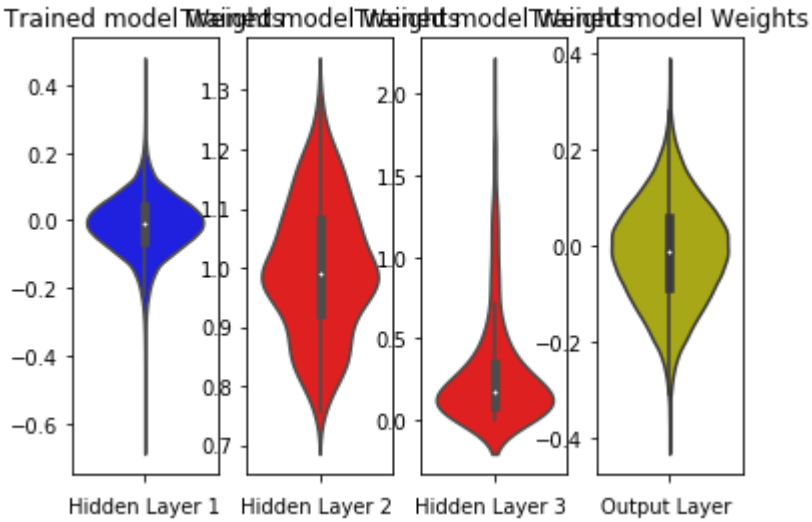
```
acc: 0.9186 - val_loss: 0.1325 - val_acc: 0.9649
Epoch 20/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.3178 -
acc: 0.9192 - val_loss: 0.1319 - val_acc: 0.9647
```

In [37]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.13190246427785604
Test accuracy: 0.9647
```



In [38]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)


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

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

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

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



MLP + ReLu + Adam + BN + Dropout (5 Layer Architecture 784-216-170-136-80-38-10)

In [39]:
```
model=Sequential()
model.add(Dense(216, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(170, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(136, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(80, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 216)               169560
_________________________________________________________________
batch_normalization_16 (Batc (None, 216)               864
_________________________________________________________________
dropout_16 (Dropout)         (None, 216)               0
_________________________________________________________________
dense_23 (Dense)             (None, 170)               36890
_________________________________________________________________
batch_normalization_17 (Batc (None, 170)               680
_________________________________________________________________
dropout_17 (Dropout)         (None, 170)               0
_________________________________________________________________
dense_24 (Dense)             (None, 136)               23256
_________________________________________________________________
batch_normalization_18 (Batc (None, 136)               544
_________________________________________________________________
dropout_18 (Dropout)         (None, 136)               0
_________________________________________________________________
dense_25 (Dense)             (None, 80)                10960
_________________________________________________________________
batch_normalization_19 (Batc (None, 80)                320
_________________________________________________________________
dropout_19 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_26 (Dense)             (None, 38)                3078
_________________________________________________________________
batch_normalization_20 (Batc (None, 38)                152
_________________________________________________________________
dropout_20 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_27 (Dense)             (None, 10)                390
=================================================================
Total params: 246,694
Trainable params: 245,414
Non-trainable params: 1,280
_________________________________________________________________
```

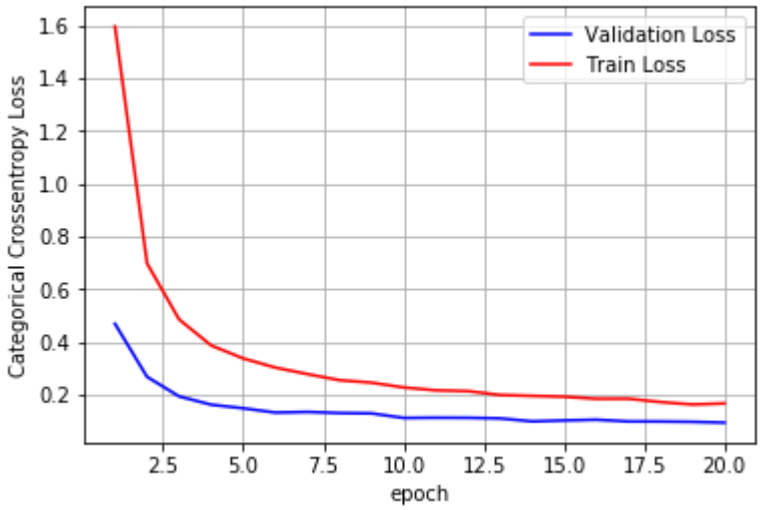
In [40]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc
uracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch
, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 171us/step - loss: 1.5948
- acc: 0.4750 - val_loss: 0.4692 - val_acc: 0.8639
Epoch 2/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.6977 -
acc: 0.7849 - val_loss: 0.2683 - val_acc: 0.9261
Epoch 3/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.4856 -
acc: 0.8690 - val_loss: 0.1942 - val_acc: 0.9458
Epoch 4/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.3871 -
acc: 0.9016 - val_loss: 0.1629 - val_acc: 0.9579
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.3379 -
acc: 0.9157 - val_loss: 0.1492 - val_acc: 0.9595
Epoch 6/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.3034 -
acc: 0.9250 - val_loss: 0.1330 - val_acc: 0.9646
Epoch 7/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.2787 -
acc: 0.9325 - val_loss: 0.1354 - val_acc: 0.9653
Epoch 8/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.2555 -
acc: 0.9383 - val_loss: 0.1313 - val_acc: 0.9671
Epoch 9/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.2462 -
acc: 0.9416 - val_loss: 0.1300 - val_acc: 0.9686
Epoch 10/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.2281 -
acc: 0.9458 - val_loss: 0.1127 - val_acc: 0.9712
Epoch 11/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.2172 -
acc: 0.9484 - val_loss: 0.1135 - val_acc: 0.9707
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2142 -
acc: 0.9491 - val_loss: 0.1131 - val_acc: 0.9733
Epoch 13/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.1997 -
acc: 0.9539 - val_loss: 0.1110 - val_acc: 0.9725
Epoch 14/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.1961 -
acc: 0.9546 - val_loss: 0.0997 - val_acc: 0.9758
Epoch 15/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.1932 -
acc: 0.9542 - val_loss: 0.1032 - val_acc: 0.9762
Epoch 16/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.1851 -
acc: 0.9567 - val_loss: 0.1059 - val_acc: 0.9757
Epoch 17/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.1854 -
acc: 0.9574 - val_loss: 0.0996 - val_acc: 0.9763
Epoch 18/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1729 -
acc: 0.9593 - val_loss: 0.0992 - val_acc: 0.9761
Epoch 19/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.1636 -
```

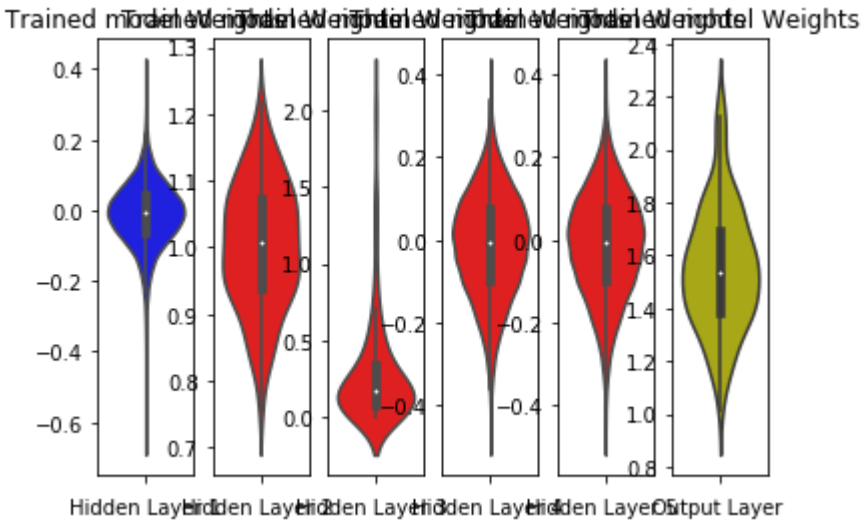
```
acc: 0.9610 - val_loss: 0.0979 - val_acc: 0.9775
Epoch 20/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.1676 -
acc: 0.9616 - val_loss: 0.0949 - val_acc: 0.9769
```

In [41]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.09488777558589354
Test accuracy: 0.9769
```



In [42]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```


Trained model Weights
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Hidden Layer 4
Hidden Layer 5
Output Layer

In [43]:
```
model=Sequential()
model.add(Dense(216, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(170, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(136, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(80, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.2))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 216)               169560
_________________________________________________________________
batch_normalization_21 (Batc (None, 216)               864
_________________________________________________________________
dropout_21 (Dropout)         (None, 216)               0
_________________________________________________________________
dense_29 (Dense)             (None, 170)               36890
_________________________________________________________________
batch_normalization_22 (Batc (None, 170)               680
_________________________________________________________________
dropout_22 (Dropout)         (None, 170)               0
_________________________________________________________________
dense_30 (Dense)             (None, 136)               23256
_________________________________________________________________
batch_normalization_23 (Batc (None, 136)               544
_________________________________________________________________
dropout_23 (Dropout)         (None, 136)               0
_________________________________________________________________
dense_31 (Dense)             (None, 80)                10960
_________________________________________________________________
batch_normalization_24 (Batc (None, 80)                320
_________________________________________________________________
dropout_24 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_32 (Dense)             (None, 38)                3078
_________________________________________________________________
batch_normalization_25 (Batc (None, 38)                152
_________________________________________________________________
dropout_25 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_33 (Dense)             (None, 10)                390
=================================================================
Total params: 246,694
Trainable params: 245,414
Non-trainable params: 1,280
_________________________________________________________________
```

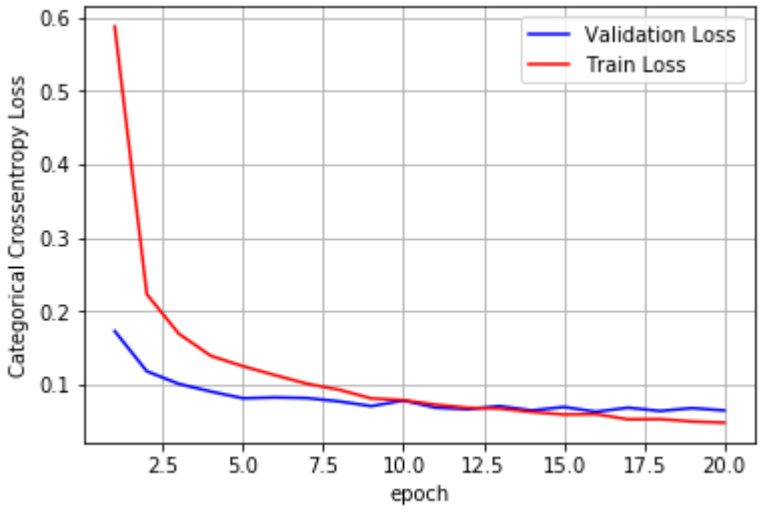
In [44]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.5880
- acc: 0.8248 - val_loss: 0.1732 - val_acc: 0.9498
Epoch 2/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.2234 -
acc: 0.9370 - val_loss: 0.1184 - val_acc: 0.9661
Epoch 3/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1694 -
acc: 0.9514 - val_loss: 0.1012 - val_acc: 0.9715
Epoch 4/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1393 -
acc: 0.9600 - val_loss: 0.0908 - val_acc: 0.9727
Epoch 5/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1253 -
acc: 0.9645 - val_loss: 0.0816 - val_acc: 0.9767
Epoch 6/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1131 -
acc: 0.9676 - val_loss: 0.0828 - val_acc: 0.9766
Epoch 7/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.1014 -
acc: 0.9713 - val_loss: 0.0819 - val_acc: 0.9758
Epoch 8/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0932 -
acc: 0.9739 - val_loss: 0.0773 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0814 -
acc: 0.9763 - val_loss: 0.0709 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0790 -
acc: 0.9769 - val_loss: 0.0786 - val_acc: 0.9777
Epoch 11/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0729 -
acc: 0.9785 - val_loss: 0.0688 - val_acc: 0.9812
Epoch 12/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0686 -
acc: 0.9797 - val_loss: 0.0667 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0676 -
acc: 0.9804 - val_loss: 0.0710 - val_acc: 0.9797
Epoch 14/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0629 -
acc: 0.9821 - val_loss: 0.0650 - val_acc: 0.9821
Epoch 15/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0595 -
acc: 0.9825 - val_loss: 0.0698 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0600 -
acc: 0.9825 - val_loss: 0.0633 - val_acc: 0.9824
Epoch 17/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0529 -
acc: 0.9850 - val_loss: 0.0688 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0532 -
acc: 0.9843 - val_loss: 0.0643 - val_acc: 0.9814
Epoch 19/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0500 -
```

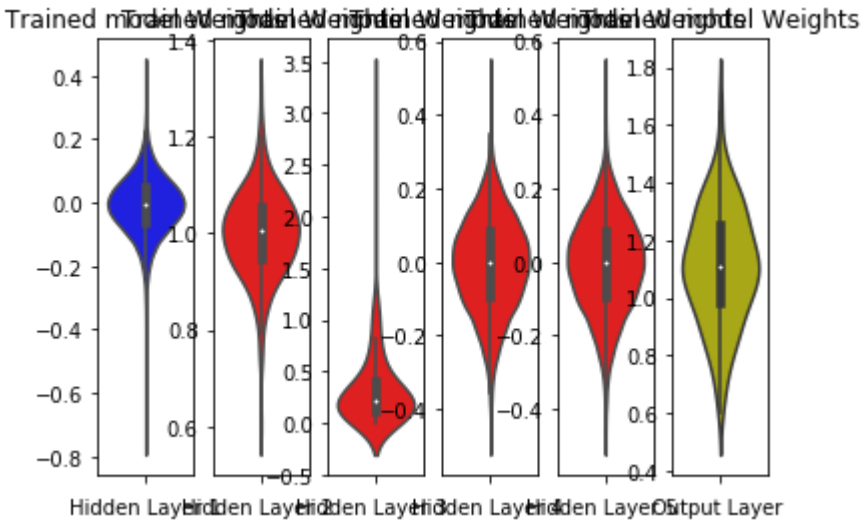
```
acc: 0.9854 - val_loss: 0.0682 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0484 -
acc: 0.9861 - val_loss: 0.0650 - val_acc: 0.9834
```

In [45]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06503258371697739
Test accuracy: 0.9834
```



In [46]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```


Trained model Weights
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Hidden Layer 4
Hidden Layer 5
Output Layer

In [47]:
```
model=Sequential()
model.add(Dense(216, activation='relu', input_shape=(input_dim,), kernel_initi
alizer=he_normal(seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(170, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(136, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(80, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

model.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None
)) )
model.add(BatchNormalization())
model.add(Dropout(0.8))

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

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 216)               169560
_________________________________________________________________
batch_normalization_26 (Batc (None, 216)               864
_________________________________________________________________
dropout_26 (Dropout)         (None, 216)               0
_________________________________________________________________
dense_35 (Dense)             (None, 170)               36890
_________________________________________________________________
batch_normalization_27 (Batc (None, 170)               680
_________________________________________________________________
dropout_27 (Dropout)         (None, 170)               0
_________________________________________________________________
dense_36 (Dense)             (None, 136)               23256
_________________________________________________________________
batch_normalization_28 (Batc (None, 136)               544
_________________________________________________________________
dropout_28 (Dropout)         (None, 136)               0
_________________________________________________________________
dense_37 (Dense)             (None, 80)                10960
_________________________________________________________________
batch_normalization_29 (Batc (None, 80)                320
_________________________________________________________________
dropout_29 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_38 (Dense)             (None, 38)                3078
_________________________________________________________________
batch_normalization_30 (Batc (None, 38)                152
_________________________________________________________________
dropout_30 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_39 (Dense)             (None, 10)                390
=================================================================
Total params: 246,694
Trainable params: 245,414
Non-trainable params: 1,280
_________________________________________________________________
```

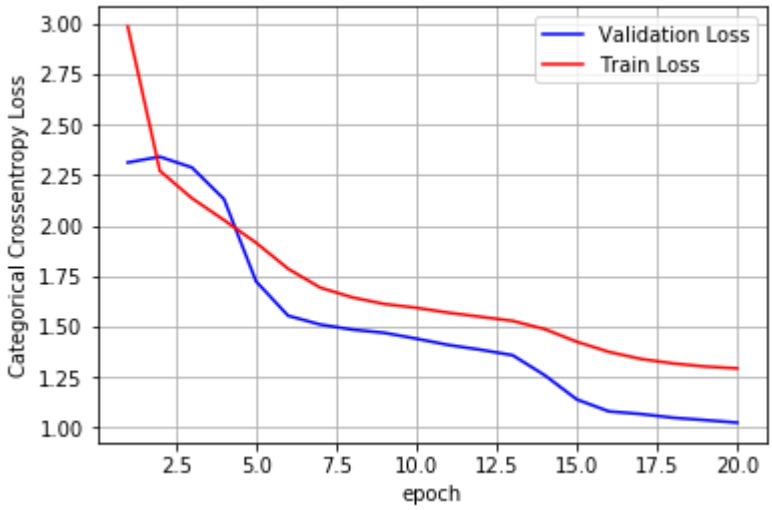
In [48]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history23 = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 188us/step - loss: 2.9871
- acc: 0.1053 - val_loss: 2.3135 - val_acc: 0.1135
Epoch 2/20
60000/60000 [==============================] - 8s 130us/step - loss: 2.2723 -
acc: 0.1475 - val_loss: 2.3420 - val_acc: 0.1135
Epoch 3/20
60000/60000 [==============================] - 8s 126us/step - loss: 2.1364 -
acc: 0.1918 - val_loss: 2.2887 - val_acc: 0.1358
Epoch 4/20
60000/60000 [==============================] - 8s 126us/step - loss: 2.0281 -
acc: 0.2190 - val_loss: 2.1321 - val_acc: 0.1821
Epoch 5/20
60000/60000 [==============================] - 7s 125us/step - loss: 1.9156 -
acc: 0.2463 - val_loss: 1.7242 - val_acc: 0.3085
Epoch 6/20
60000/60000 [==============================] - 8s 125us/step - loss: 1.7868 -
acc: 0.2807 - val_loss: 1.5540 - val_acc: 0.3138
Epoch 7/20
60000/60000 [==============================] - 8s 126us/step - loss: 1.6934 -
acc: 0.3031 - val_loss: 1.5104 - val_acc: 0.3679
Epoch 8/20
60000/60000 [==============================] - 7s 123us/step - loss: 1.6449 -
acc: 0.3204 - val_loss: 1.4852 - val_acc: 0.3799
Epoch 9/20
60000/60000 [==============================] - 7s 124us/step - loss: 1.6120 -
acc: 0.3313 - val_loss: 1.4692 - val_acc: 0.3778
Epoch 10/20
60000/60000 [==============================] - 8s 131us/step - loss: 1.5934 -
acc: 0.3406 - val_loss: 1.4402 - val_acc: 0.3839
Epoch 11/20
60000/60000 [==============================] - 8s 131us/step - loss: 1.5690 -
acc: 0.3511 - val_loss: 1.4090 - val_acc: 0.3861
Epoch 12/20
60000/60000 [==============================] - 7s 123us/step - loss: 1.5488 -
acc: 0.3598 - val_loss: 1.3855 - val_acc: 0.3953
Epoch 13/20
60000/60000 [==============================] - 7s 124us/step - loss: 1.5284 -
acc: 0.3707 - val_loss: 1.3582 - val_acc: 0.4097
Epoch 14/20
60000/60000 [==============================] - 7s 123us/step - loss: 1.4876 -
acc: 0.3903 - val_loss: 1.2590 - val_acc: 0.5016
Epoch 15/20
60000/60000 [==============================] - 8s 127us/step - loss: 1.4249 -
acc: 0.4100 - val_loss: 1.1390 - val_acc: 0.5219
Epoch 16/20
60000/60000 [==============================] - 7s 123us/step - loss: 1.3747 -
acc: 0.4209 - val_loss: 1.0800 - val_acc: 0.5192
Epoch 17/20
60000/60000 [==============================] - 8s 126us/step - loss: 1.3397 -
acc: 0.4303 - val_loss: 1.0669 - val_acc: 0.5370
Epoch 18/20
60000/60000 [==============================] - 7s 124us/step - loss: 1.3176 -
acc: 0.4322 - val_loss: 1.0485 - val_acc: 0.5151
Epoch 19/20
60000/60000 [==============================] - 7s 123us/step - loss: 1.3023 -
```

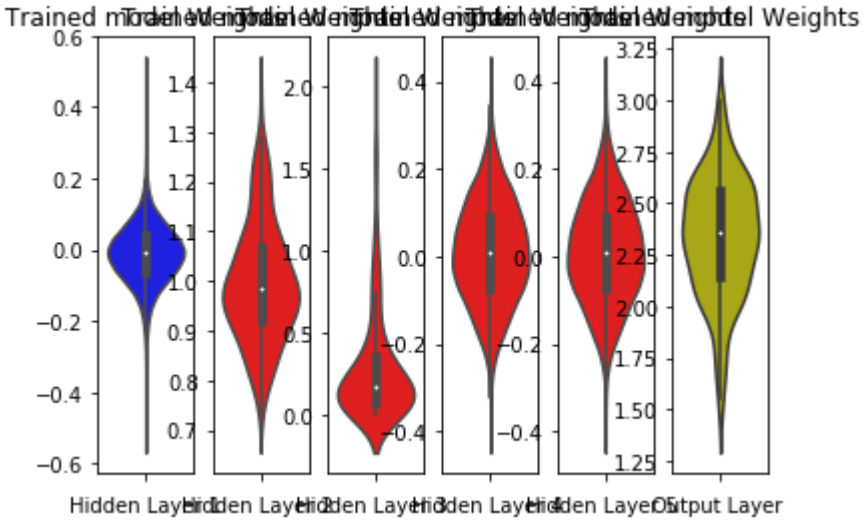
```
acc: 0.4380 - val_loss: 1.0368 - val_acc: 0.5301
Epoch 20/20
60000/60000 [==============================] - 8s 126us/step - loss: 1.2931 -
acc: 0.4378 - val_loss: 1.0242 - val_acc: 0.5750
```

In [49]:
```
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')
x = list(range(1,nb_epoch+1))
vy = history23.history['val_loss']
ty = history23.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 1.0241631193161012
Test accuracy: 0.575
```



In [50]:
```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```


Trained model Weights
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Hidden Layer 4
Hidden Layer 5
Output Layer

In [51]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Architecture", "Dropout Rate", "Test Score", "Test Accuracy"
]
x.add_row(["2 Layer Architecture 784-168-472-10", "0.5 ", 0.06, 0.98])
x.add_row(["2 Layer Architecture 784-168-472-10", "0.2 ", 0.06, 0.98])
x.add_row(["2 Layer Architecture 784-168-472-10", "0.8 ", 0.09, 0.97])
x.add_row(["3 Layer Architecture 784-352-164-124-10", "0.5", 0.06, 0.98])
x.add_row(["3 Layer Architecture 784-352-164-124-10", "0.2", 0.06, 0.98])
x.add_row(["3 Layer Architecture 784-352-164-124-10","0.8", 0.13, 0.96])
x.add_row(["5 Layer Architecture 784-216-170-136-80-38-10", "0.5", 0.09, 0.97
])
x.add_row(["5 Layer Architecture 784-216-170-136-80-38-10", "0.2", 0.06, 0.98
])
x.add_row(["5 Layer Architecture 784-216-170-136-80-38-10", "0.8", 1.02, 0.57
])
print(x)
```

```
+-----------------------------------------------+--------------+------------+
---------------+
|                  Architecture                 | Dropout Rate | Test Score |
Test Accuracy |
+-----------------------------------------------+--------------+------------+
---------------+
|      2 Layer Architecture 784-168-472-10      |     0.5      |    0.06    |
      0.98     |
|      2 Layer Architecture 784-168-472-10      |     0.2      |    0.06    |
      0.98     |
|      2 Layer Architecture 784-168-472-10      |     0.8      |    0.09    |
      0.97     |
|    3 Layer Architecture 784-352-164-124-10    |     0.5      |    0.06    |
      0.98     |
|    3 Layer Architecture 784-352-164-124-10    |     0.2      |    0.06    |
      0.98     |
|    3 Layer Architecture 784-352-164-124-10    |     0.8      |    0.13    |
      0.96     |
| 5 Layer Architecture 784-216-170-136-80-38-10 |     0.5      |    0.09    |
      0.97     |
| 5 Layer Architecture 784-216-170-136-80-38-10 |     0.2      |    0.06    |
      0.98     |
| 5 Layer Architecture 784-216-170-136-80-38-10 |     0.8      |    1.02    |
      0.57     |
+-----------------------------------------------+--------------+------------+
---------------+
```