Assignment-12: TensorFlow and Keras: Build various MLP architectures for MNIST dataset

In [1]:

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

```
Using TensorFlow backend.
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:516:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:517:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:518:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: Passing (type, 1) or '1t
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [2]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
```

```python
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()
    fig.canvas.draw()
```

In [3]:

```python
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:

```python
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 [5]:

```python
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

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

In [6]:

```python
# after converting the input images from 3d to 2d vectors

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

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

In [7]:

```python
# 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 [8]:

```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [9]:

```
# 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.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         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.          0.17647059 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.          0.0627451  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.          0.97647059 0.99215686 0.97647059 0.25098039 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.          0.15294118 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.
 0.09411765 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.         0.         0.
 0.         0.         0.         0.        ]
```

In [10]:

```
# 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] (one hot 
representation)
# 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.]
```

Softmax classifier

In [11]:

```
# https://keras.io/getting-started/sequential-model-guide/

# The Sequential model is a linear stack of layers.
# you can create a Sequential model by passing a list of layer instances to the constructor:

# model = Sequential([
#     Dense(32, input_shape=(784,)),
#     Activation('relu'),
#     Dense(10),
#     Activation('softmax'),
# ])

# You can also simply add layers via the .add() method:

# model = Sequential()
# model.add(Dense(32, input_dim=784))
# model.add(Activation('relu'))

###

# https://keras.io/layers/core/

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', 
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, 
activity_regularizer=None, 
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bias) where
```

```
# Dense implements the operation: output = activation(dot(input, kernel) + bias) where
# activation is the element-wise activation function passed as the activation argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bias is True).

# output = activation(dot(input, kernel) + bias)  => y = activation(WT. X + b)

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers:

# from keras.layers import Activation, Dense

# model.add(Dense(64))
# model.add(Activation('tanh'))

# This is equivalent to:
# model.add(Dense(64, activation='tanh'))

# there are many activation functions ar available ex: tanh, relu, softmax


from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [12]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [13]:

```
print('input dimensions:',input_dim,)
```

```
input dimensions: 784
```

1) 2-Hidden layer Architecture (784-472-168-10)

1.1 MLP + ReLU activation function + ADAM optimizer

In [14]:

```
from keras.initializers import he_normal
import warnings
warnings.filterwarnings("ignore")
model_relu = Sequential()
model_relu.add(Dense(472, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history11 = model_relu.fit(X_train, Y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, Y_test))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0818 11:16:39.769354 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:74: The nam
e tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0818 11:16:39.781115 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:517: The na
me tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

W0818 11:16:39.783540 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:4185: The n
ame tf.truncated_normal is deprecated. Please use tf.random.truncated_normal instead.

W0818 11:16:39.800450 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:4138: The n
ame tf.random_uniform is deprecated. Please use tf.random.uniform instead.

W0818 11:16:39.810351 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/optimizers.py:790: The name
tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

W0818 11:16:39.827991 140217894131520 deprecation_wrapper.py:119] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3295: The n
ame tf.log is deprecated. Please use tf.math.log instead.

W0818 11:16:39.893717 140217894131520 deprecation.py:323] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/ops/math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

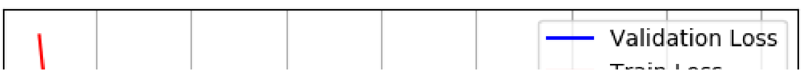
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 472)               370520    
_________________________________________________________________
dense_2 (Dense)              (None, 168)               79464     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1690      
=================================================================
Total params: 451,674
Trainable params: 451,674
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.2344 - acc: 0.9320 -
val_loss: 0.1033 - val_acc: 0.9694
Epoch 2/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0872 - acc: 0.9738 -
val_loss: 0.0851 - val_acc: 0.9757
Epoch 3/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0558 - acc: 0.9827 -
val_loss: 0.0665 - val_acc: 0.9795
Epoch 4/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0365 - acc: 0.9881 -
val_loss: 0.0719 - val_acc: 0.9770
Epoch 5/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0281 - acc: 0.9911 -
val_loss: 0.0605 - val_acc: 0.9811
Epoch 6/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0214 - acc: 0.9930 -
val_loss: 0.0674 - val_acc: 0.9808
Epoch 7/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0178 - acc: 0.9940 -
val_loss: 0.0681 - val_acc: 0.9809
Epoch 8/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0135 - acc: 0.9955 -
val_loss: 0.0706 - val_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0122 - acc: 0.9960 -
val_loss: 0.0721 - val_acc: 0.9819
Epoch 10/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0133 - acc: 0.9959 -
```

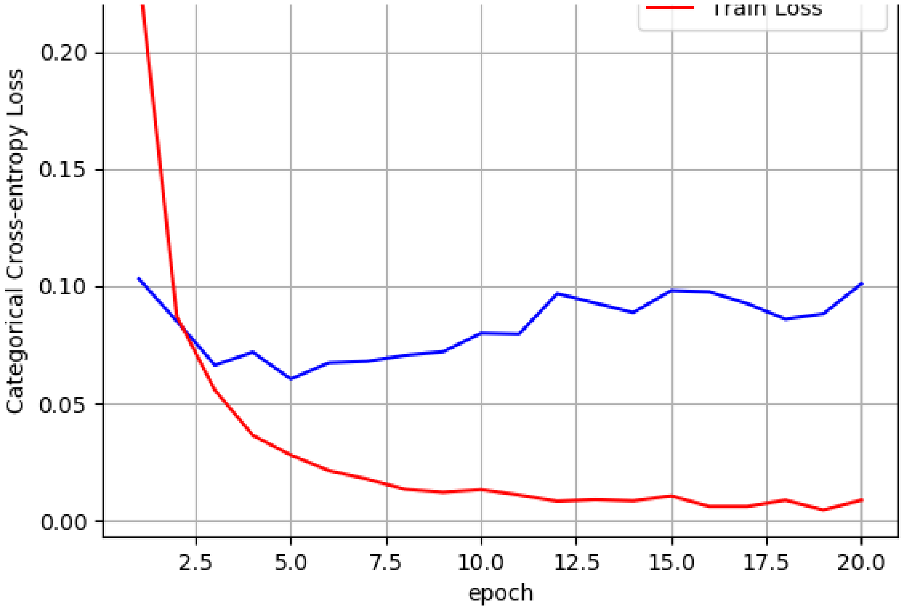
```
60000/60000 [==============================] - 1s 18us/step - loss: 0.0133 - acc: 0.9959 -
val_loss: 0.0800 - val_acc: 0.9800
Epoch 11/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0110 - acc: 0.9963 -
val_loss: 0.0796 - val_acc: 0.9819
Epoch 12/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0084 - acc: 0.9971 -
val_loss: 0.0969 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0091 - acc: 0.9968 -
val_loss: 0.0929 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0086 - acc: 0.9970 -
val_loss: 0.0888 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0106 - acc: 0.9965 -
val_loss: 0.0982 - val_acc: 0.9788
Epoch 16/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0062 - acc: 0.9980 -
val_loss: 0.0976 - val_acc: 0.9789
Epoch 17/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0062 - acc: 0.9981 -
val_loss: 0.0926 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0088 - acc: 0.9971 -
val_loss: 0.0860 - val_acc: 0.9814
Epoch 19/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0047 - acc: 0.9986 -
val_loss: 0.0883 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0088 - acc: 0.9971 -
val_loss: 0.1011 - val_acc: 0.9790
```

In [15]:

```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score1=score[0]
score2=score[1]
train_acc1=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax11 = plt.subplots(1,1)
ax11.set_xlabel('epoch') ; ax11.set_ylabel('Categorical Cross-entropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(x_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(x_test, y_test))

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy11 = history11.history['val_loss']
ty11 = history11.history['loss']
plt_dynamic(x, vy11, ty11, ax11)
```

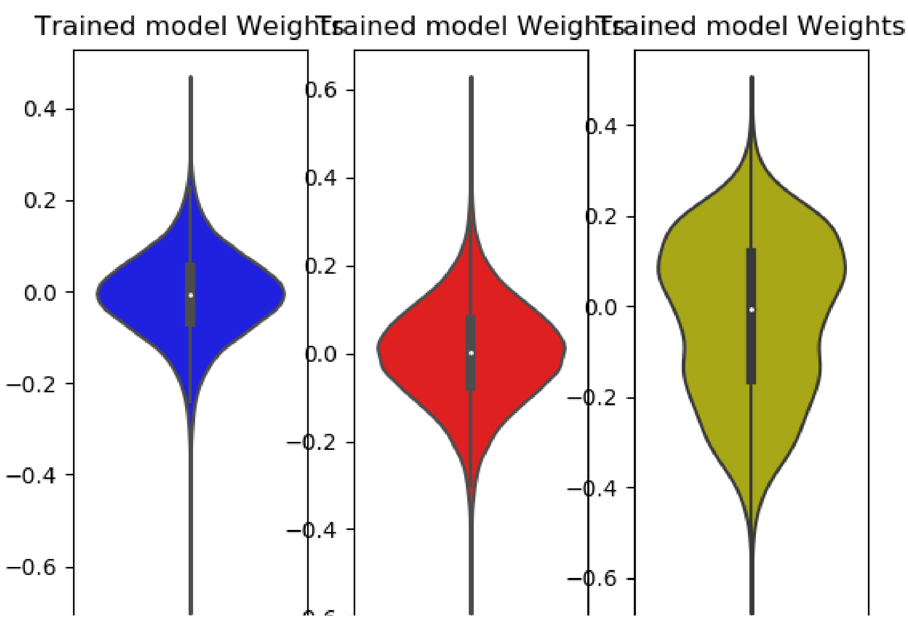
```
Test score: 0.10111289650749382
Test accuracy: 0.979
```

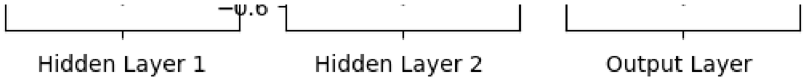




In [16]:

```
w_after = model_relu.get_weights()

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


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

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

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





MLP + ReLU activation function + RMSprop optimizer

In [17]:

```
from keras.initializers import he_normal
import warnings
warnings.filterwarnings("ignore")
model_relu = Sequential()
model_relu.add(Dense(472, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='rmsprop',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history11 = model_relu.fit(X_train, Y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, Y_test))
```

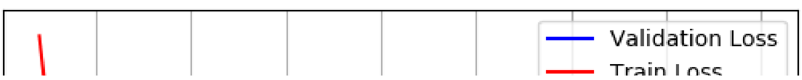
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 472)               370520    
_________________________________________________________________
dense_5 (Dense)              (None, 168)               79464     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1690      
=================================================================
Total params: 451,674
Trainable params: 451,674
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.2273 - acc: 0.9312 -
val_loss: 0.1277 - val_acc: 0.9582
Epoch 2/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0854 - acc: 0.9732 -
val_loss: 0.0855 - val_acc: 0.9714
Epoch 3/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0568 - acc: 0.9825 -
val_loss: 0.0737 - val_acc: 0.9769
Epoch 4/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0405 - acc: 0.9873 -
val_loss: 0.0619 - val_acc: 0.9794
Epoch 5/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.0295 - acc: 0.9906 -
val_loss: 0.0759 - val_acc: 0.9796
Epoch 6/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0230 - acc: 0.9928 -
val_loss: 0.0871 - val_acc: 0.9763
Epoch 7/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0174 - acc: 0.9942 -
val_loss: 0.0786 - val_acc: 0.9821
Epoch 8/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0146 - acc: 0.9959 -
val_loss: 0.0832 - val_acc: 0.9823
Epoch 9/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.0121 - acc: 0.9963 -
val_loss: 0.0794 - val_acc: 0.9830
Epoch 10/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0096 - acc: 0.9969 -
```

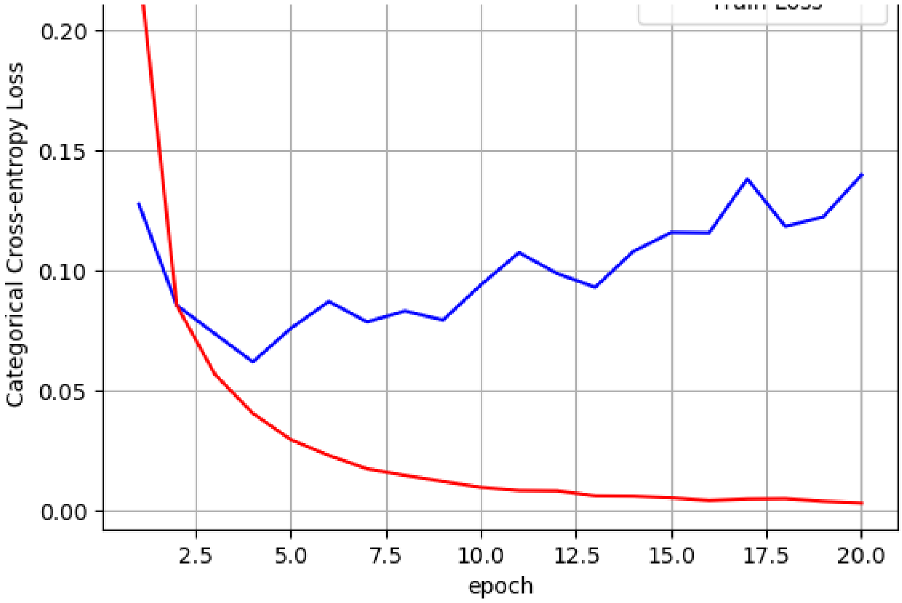
```
val_loss: 0.0941 - val_acc: 0.9788
Epoch 11/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0083 - acc: 0.9973 -
val_loss: 0.1075 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0082 - acc: 0.9976 -
val_loss: 0.0988 - val_acc: 0.9817
Epoch 13/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.0061 - acc: 0.9981 -
val_loss: 0.0931 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0060 - acc: 0.9983 -
val_loss: 0.1080 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0053 - acc: 0.9984 -
val_loss: 0.1159 - val_acc: 0.9807
Epoch 16/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0042 - acc: 0.9988 -
val_loss: 0.1157 - val_acc: 0.9817
Epoch 17/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.0048 - acc: 0.9986 -
val_loss: 0.1382 - val_acc: 0.9806
Epoch 18/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0050 - acc: 0.9983 -
val_loss: 0.1184 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0038 - acc: 0.9989 -
val_loss: 0.1224 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.0031 - acc: 0.9991 -
val_loss: 0.1398 - val_acc: 0.9809
```

In [18]:

```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score1=score[0]
score2=score[1]
train_acc1=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax11 = plt.subplots(1,1)
ax11.set_xlabel('epoch') ; ax11.set_ylabel('Categorical Cross-entropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(x_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(x_test, y_test))

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy11 = history11.history['val_loss']
ty11 = history11.history['loss']
plt_dynamic(x, vy11, ty11, ax11)
```

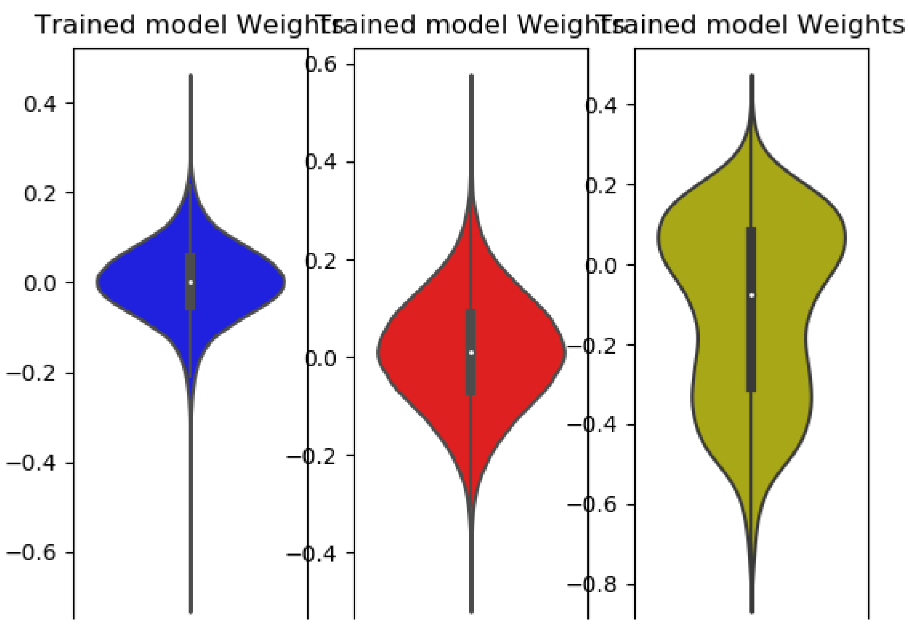
```
Test score: 0.13976741824084157
Test accuracy: 0.9809
```

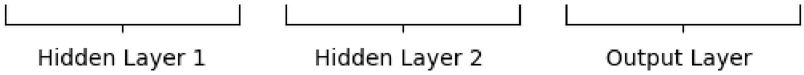




In [19]:

```
w_after = model_relu.get_weights()

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


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

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

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





1.2 MLP + Batch-Norm on hidden Layers + Adam Optimizer

In [20]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(472, activation='relu',
                      input_shape=(input_dim,),
                      kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(168, activation='relu',
                      kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 472)               370520    
_________________________________________________________________
batch_normalization_1 (Batch (None, 472)               1888      
_________________________________________________________________
dense_8 (Dense)              (None, 168)               79464     
_________________________________________________________________
batch_normalization_2 (Batch (None, 168)               672       
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1690      
=================================================================
Total params: 454,234
Trainable params: 452,954
Non-trainable params: 1,280
_________________________________________________________________
```

In [21]:

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

history12 = model_batch.fit(X_train, Y_train,
                            batch_size=batch_size,
                            epochs=nb_epoch, verbose=1,
                            validation_data=(X_test, Y_test))
```

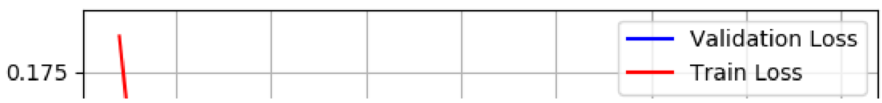
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1872 - acc: 0.9437 -
val_loss: 0.1048 - val_acc: 0.9668
Epoch 2/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0699 - acc: 0.9789 -
val_loss: 0.0926 - val_acc: 0.9712
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0446 - acc: 0.9866 -
val_loss: 0.0824 - val_acc: 0.9739
Epoch 4/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0322 - acc: 0.9902 -
val_loss: 0.0861 - val_acc: 0.9754
Epoch 5/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0273 - acc: 0.9913 -
val_loss: 0.0790 - val_acc: 0.9755
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0211 - acc: 0.9930 -
```

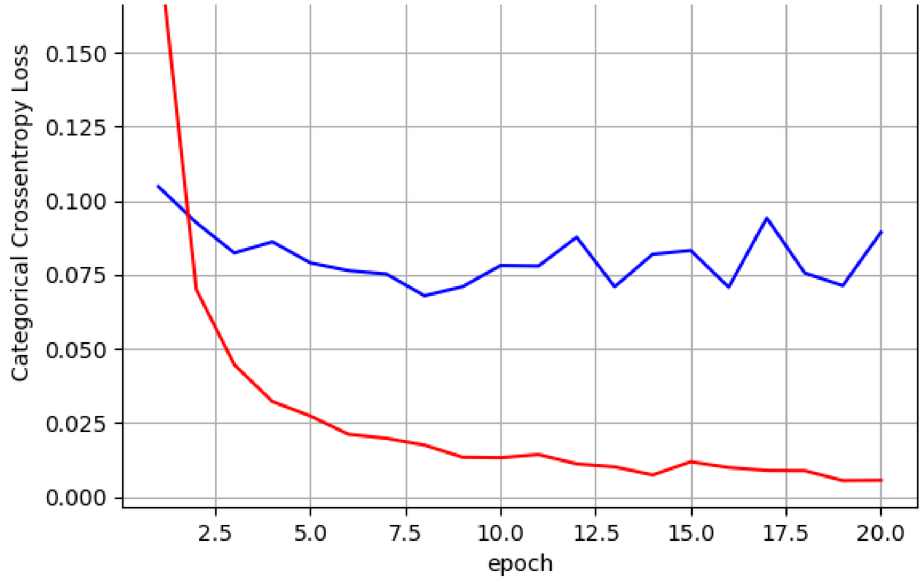
```
60000/60000 [==============================] - 2s 31us/step - loss: 0.0211 - acc: 0.9930
val_loss: 0.0764 - val_acc: 0.9787
Epoch 7/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0197 - acc: 0.9933 -
val_loss: 0.0752 - val_acc: 0.9793
Epoch 8/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0175 - acc: 0.9944 -
val_loss: 0.0679 - val_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0134 - acc: 0.9955 -
val_loss: 0.0710 - val_acc: 0.9818
Epoch 10/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0132 - acc: 0.9960 -
val_loss: 0.0781 - val_acc: 0.9791
Epoch 11/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0143 - acc: 0.9952 -
val_loss: 0.0780 - val_acc: 0.9774
Epoch 12/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0111 - acc: 0.9963 -
val_loss: 0.0877 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0101 - acc: 0.9968 -
val_loss: 0.0709 - val_acc: 0.9817
Epoch 14/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0074 - acc: 0.9977 -
val_loss: 0.0819 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0118 - acc: 0.9960 -
val_loss: 0.0832 - val_acc: 0.9799
Epoch 16/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0099 - acc: 0.9965 -
val_loss: 0.0708 - val_acc: 0.9816
Epoch 17/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0089 - acc: 0.9968 -
val_loss: 0.0942 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0088 - acc: 0.9971 -
val_loss: 0.0756 - val_acc: 0.9807
Epoch 19/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0055 - acc: 0.9982 -
val_loss: 0.0713 - val_acc: 0.9829
Epoch 20/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0056 - acc: 0.9982 -
val_loss: 0.0893 - val_acc: 0.9797
```

In [22]:

```python
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
score3=score[0]
score4=score[1]
train_acc2=history11.history['acc']

fig,ax12 = plt.subplots(1,1)
ax12.set_xlabel('epoch') ; ax12.set_ylabel('Categorical Crossentropy Loss')

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


vy12 = history12.history['val_loss']
ty12 = history12.history['loss']
plt_dynamic(x, vy12, ty12, ax12)
```

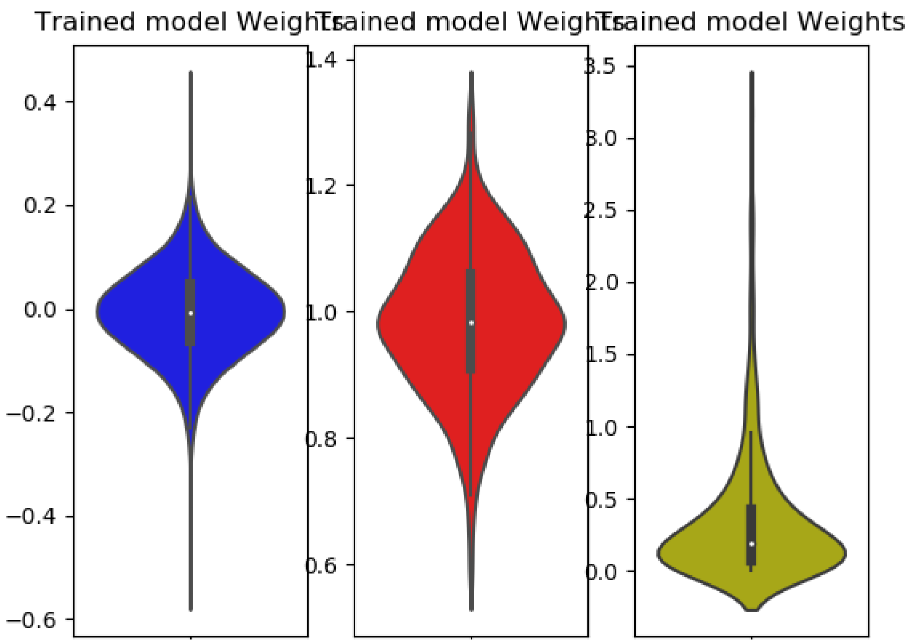
```
Test score: 0.08932151498529384
Test accuracy: 0.9797
```





In [23]:

```
w_after = model_batch.get_weights()

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


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

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

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



MLP + Batch-Norm on hidden Layers + Adagrad Optimizer

In [24]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(472, activation='relu',
                      input_shape=(input_dim,),
                      kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(168, activation='relu',
                      kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

model_batch.compile(optimizer='adagrad', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history12 = model_batch.fit(X_train, Y_train,
                            batch_size=batch_size,
                            epochs=nb_epoch, verbose=1,
                            validation_data=(X_test, Y_test))
```

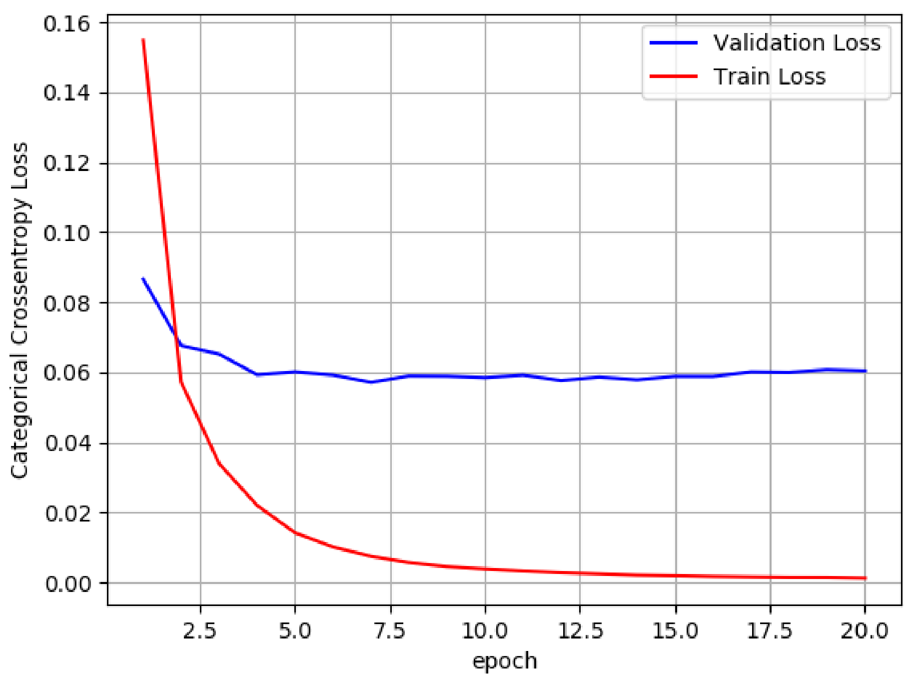
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1549 - acc: 0.9541 -
val_loss: 0.0866 - val_acc: 0.9746
Epoch 2/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0572 - acc: 0.9836 -
val_loss: 0.0676 - val_acc: 0.9795
Epoch 3/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0339 - acc: 0.9912 -
val_loss: 0.0652 - val_acc: 0.9803
Epoch 4/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0220 - acc: 0.9950 -
val_loss: 0.0593 - val_acc: 0.9823
Epoch 5/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0142 - acc: 0.9976 -
val_loss: 0.0601 - val_acc: 0.9813
Epoch 6/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0101 - acc: 0.9986 -
val_loss: 0.0592 - val_acc: 0.9826
Epoch 7/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0074 - acc: 0.9992 -
val_loss: 0.0571 - val_acc: 0.9833
Epoch 8/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0057 - acc: 0.9994 -
val_loss: 0.0589 - val_acc: 0.9829
Epoch 9/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0045 - acc: 0.9998 -
val_loss: 0.0589 - val_acc: 0.9831
Epoch 10/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0038 - acc: 0.9999 -
val_loss: 0.0584 - val_acc: 0.9836
Epoch 11/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0033 - acc: 0.9999 -
val_loss: 0.0592 - val_acc: 0.9833
Epoch 12/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0028 - acc: 0.9999 -
val_loss: 0.0576 - val_acc: 0.9831
Epoch 13/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0024 - acc: 1.0000 -
val_loss: 0.0586 - val_acc: 0.9825
Epoch 14/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0021 - acc: 1.0000 -
val_loss: 0.0578 - val_acc: 0.9836
```

```
Epoch 15/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.0019 - acc: 1.0000 -
val_loss: 0.0588 - val_acc: 0.9833
Epoch 16/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.0017 - acc: 1.0000 -
val_loss: 0.0588 - val_acc: 0.9842
Epoch 17/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0015 - acc: 1.0000 -
val_loss: 0.0601 - val_acc: 0.9841
Epoch 18/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0014 - acc: 1.0000 -
val_loss: 0.0599 - val_acc: 0.9844
Epoch 19/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0014 - acc: 1.0000 -
val_loss: 0.0607 - val_acc: 0.9836
Epoch 20/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0012 - acc: 1.0000 -
val_loss: 0.0604 - val_acc: 0.9843
```

In [25]:

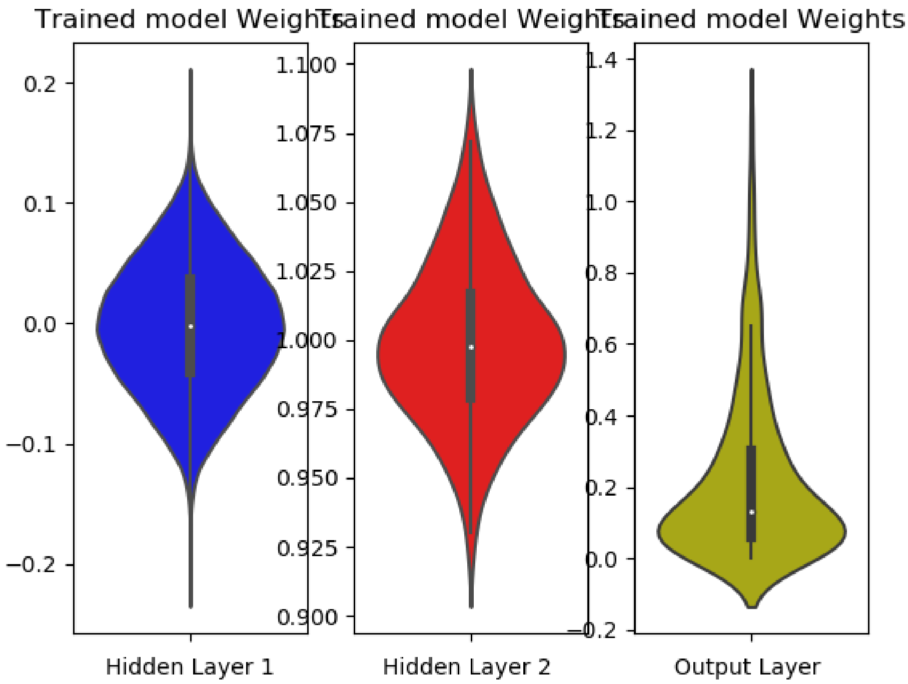
```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
score3=score[0]
score4=score[1]
train_acc2=history11.history['acc']

fig,ax12 = plt.subplots(1,1)
ax12.set_xlabel('epoch') ; ax12.set_ylabel('Categorical Crossentropy Loss')

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


vy12 = history12.history['val_loss']
ty12 = history12.history['loss']
plt_dynamic(x, vy12, ty12, ax12)
```

```
Test score: 0.060404907803970856
Test accuracy: 0.9843
```



In [26]:

```
w_after = model_batch.get_weights()

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


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

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

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



1.3 MLP + Dropout + AdamOptimizer

In [27]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-
keras

from keras.layers import Dropout

model_drop = Sequential()

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

model_drop.add(Dense(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
```

```
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
W0818 11:21:10.975695 140217894131520 deprecation.py:506] From
/home/ubuntu/anaconda3/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3445:
calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 472)               370520
_________________________________________________________________
batch_normalization_5 (Batch (None, 472)               1888
_________________________________________________________________
dropout_1 (Dropout)          (None, 472)               0
_________________________________________________________________
dense_14 (Dense)             (None, 168)               79464
_________________________________________________________________
batch_normalization_6 (Batch (None, 168)               672
_________________________________________________________________
dropout_2 (Dropout)          (None, 168)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                1690
=================================================================
Total params: 454,234
Trainable params: 452,954
Non-trainable params: 1,280
_________________________________________________________________
```

In [28]:

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

history13 = model_drop.fit(X_train, Y_train,
                         batch_size=batch_size,

                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, Y_test))
```

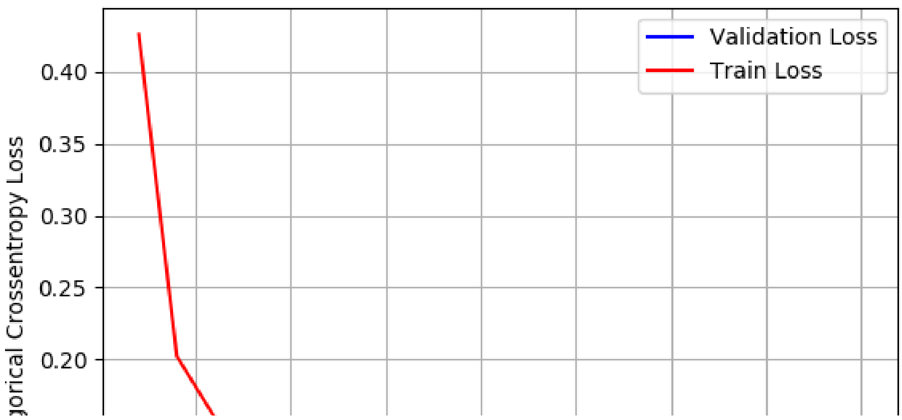
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.4263 - acc: 0.8711 -
val_loss: 0.1414 - val_acc: 0.9557
Epoch 2/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.2022 - acc: 0.9400 -
val_loss: 0.1110 - val_acc: 0.9667
Epoch 3/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.1594 - acc: 0.9509 -
val_loss: 0.0941 - val_acc: 0.9705
Epoch 4/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1349 - acc: 0.9581 -
val_loss: 0.0790 - val_acc: 0.9745
Epoch 5/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1176 - acc: 0.9637 -
val_loss: 0.0763 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.1087 - acc: 0.9663 -
val_loss: 0.0764 - val_acc: 0.9776
Epoch 7/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0976 - acc: 0.9701 -
val_loss: 0.0635 - val_acc: 0.9806
Epoch 8/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0899 - acc: 0.9726 -
val_loss: 0.0679 - val_acc: 0.9797
```

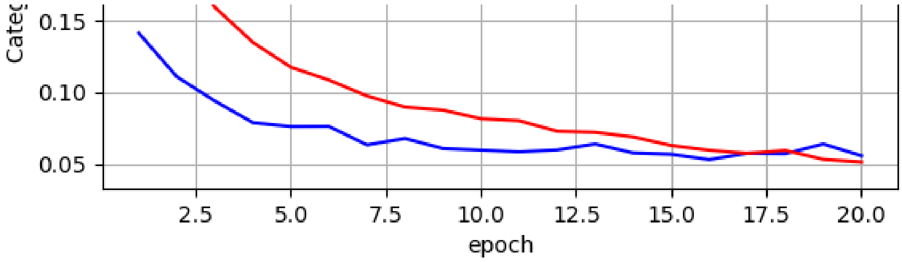
```
Epoch 9/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0877 - acc: 0.9728 -
val_loss: 0.0610 - val_acc: 0.9810
Epoch 10/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0817 - acc: 0.9742 -
val_loss: 0.0598 - val_acc: 0.9823
Epoch 11/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0803 - acc: 0.9746 -
val_loss: 0.0587 - val_acc: 0.9812
Epoch 12/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0730 - acc: 0.9769 -
val_loss: 0.0600 - val_acc: 0.9811
Epoch 13/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0723 - acc: 0.9773 -
val_loss: 0.0641 - val_acc: 0.9807
Epoch 14/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0690 - acc: 0.9780 -
val_loss: 0.0579 - val_acc: 0.9827
Epoch 15/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0631 - acc: 0.9801 -
val_loss: 0.0569 - val_acc: 0.9828
Epoch 16/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0598 - acc: 0.9808 -
val_loss: 0.0532 - val_acc: 0.9831
Epoch 17/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0576 - acc: 0.9810 -
val_loss: 0.0576 - val_acc: 0.9830
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0596 - acc: 0.9811 -
val_loss: 0.0573 - val_acc: 0.9825
Epoch 19/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0534 - acc: 0.9830 -
val_loss: 0.0641 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0515 - acc: 0.9834 -
val_loss: 0.0559 - val_acc: 0.9836
```

In [29]:

```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score5=score[0]
score6=score[1]
train_acc3=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax13 = plt.subplots(1,1)
ax13.set_xlabel('epoch') ; ax13.set_ylabel('Categorical Crossentropy Loss')

vy13 = history13.history['val_loss']
ty13 = history13.history['loss']
plt_dynamic(x, vy13, ty13, ax13)
```

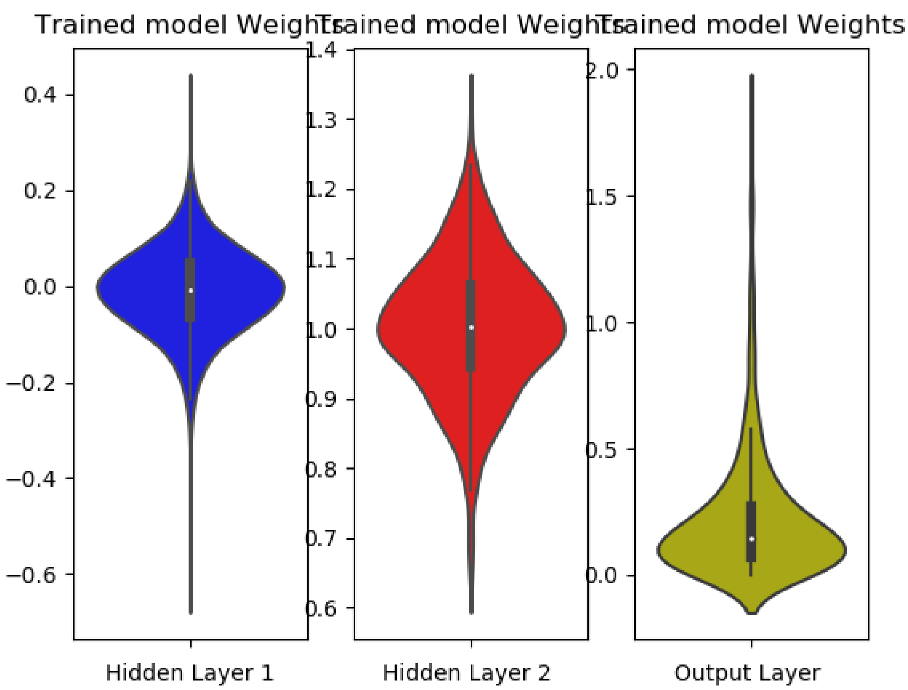
```
Test score: 0.05593528219778964
Test accuracy: 0.9836
```





In [30]:

```
w_after = model_drop.get_weights()

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


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

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

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



MLP + Dropout + Adadelta Optimizer

In [31]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-
keras
```

```python
keras

from keras.layers import Dropout

model_drop = Sequential()

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

model_drop.add(Dense(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.4))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_16 (Dense)             (None, 472)               370520    
_________________________________________________________________
batch_normalization_7 (Batch (None, 472)               1888      
_________________________________________________________________
dropout_3 (Dropout)          (None, 472)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 168)               79464     
_________________________________________________________________
batch_normalization_8 (Batch (None, 168)               672       
_________________________________________________________________
dropout_4 (Dropout)          (None, 168)               0         
_________________________________________________________________
dense_18 (Dense)             (None, 10)                1690      
=================================================================
Total params: 454,234
Trainable params: 452,954
Non-trainable params: 1,280
_________________________________________________________________
```

In [32]:

```python
model_drop.compile(optimizer='adadelta',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history13 = model_drop.fit(X_train, Y_train,
                           batch_size=batch_size,

                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, Y_test))
```

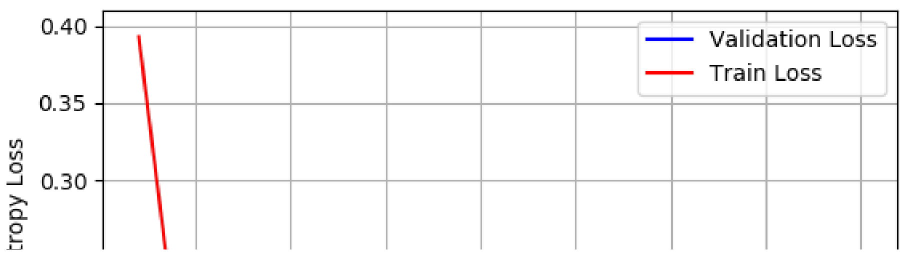
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3931 - acc: 0.8808 - val_loss: 0.1345 - val_acc: 0.9589
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.1973 - acc: 0.9409 - val_loss: 0.1051 - val_acc: 0.9677
Epoch 3/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1523 - acc: 0.9533 - val_loss: 0.0853 - val_acc: 0.9736
Epoch 4/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1245 - acc: 0.9615 - val_loss: 0.0800 - val_acc: 0.9757
Epoch 5/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.1114 - acc: 0.9657 - val_loss: 0.0712 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0985 - acc: 0.9698 - val_loss: 0.0675 - val_acc: 0.9793
```

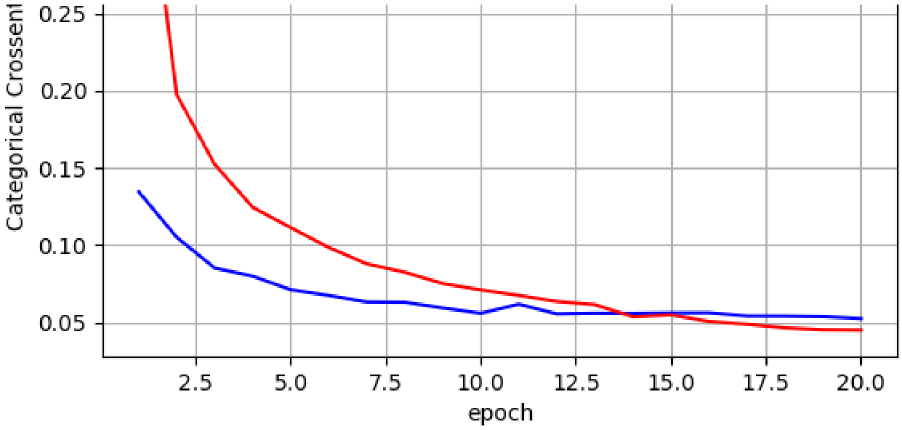
```
Epoch 7/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.0880 - acc: 0.9724 -
val_loss: 0.0633 - val_acc: 0.9805
Epoch 8/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0825 - acc: 0.9747 -
val_loss: 0.0631 - val_acc: 0.9807
Epoch 9/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0752 - acc: 0.9770 -
val_loss: 0.0594 - val_acc: 0.9816
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0711 - acc: 0.9779 -
val_loss: 0.0560 - val_acc: 0.9828
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0675 - acc: 0.9789 -
val_loss: 0.0619 - val_acc: 0.9810
Epoch 12/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0636 - acc: 0.9798 -
val_loss: 0.0556 - val_acc: 0.9831
Epoch 13/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.0616 - acc: 0.9802 -
val_loss: 0.0560 - val_acc: 0.9831
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0537 - acc: 0.9829 -
val_loss: 0.0559 - val_acc: 0.9834
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.0549 - acc: 0.9825 -
val_loss: 0.0562 - val_acc: 0.9832
Epoch 16/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0507 - acc: 0.9838 -
val_loss: 0.0563 - val_acc: 0.9833
Epoch 17/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0489 - acc: 0.9847 -
val_loss: 0.0542 - val_acc: 0.9838
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0465 - acc: 0.9850 -
val_loss: 0.0542 - val_acc: 0.9839
Epoch 19/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0453 - acc: 0.9853 -
val_loss: 0.0539 - val_acc: 0.9847
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.0451 - acc: 0.9849 -
val_loss: 0.0525 - val_acc: 0.9840
```

In [33]:

```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score5=score[0]
score6=score[1]
train_acc3=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax13 = plt.subplots(1,1)
ax13.set_xlabel('epoch') ; ax13.set_ylabel('Categorical Crossentropy Loss')

vy13 = history13.history['val_loss']
ty13 = history13.history['loss']
plt_dynamic(x, vy13, ty13, ax13)
```

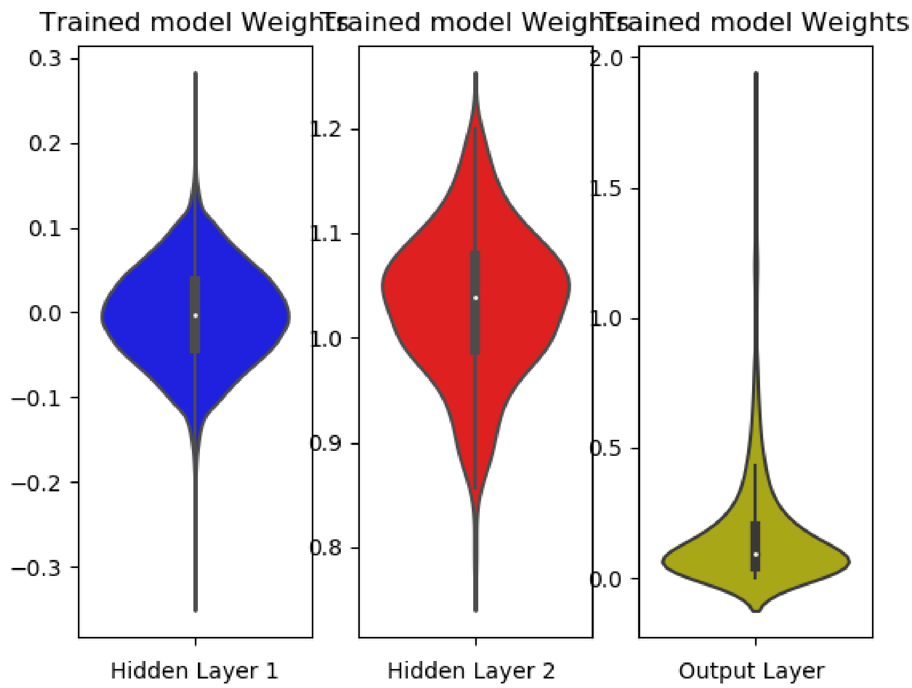
```
Test score: 0.05253026305373642
Test accuracy: 0.984
```





In [34]:

```
w_after = model_drop.get_weights()

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


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

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

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



2) 3-Hidden layer architecture (784-352-164-124 architecture)

2.1 MLP + ReLU + ADAM

In [35]:

```
model_relu = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history21 = model_relu.fit(X_train, Y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, Y_test))
```

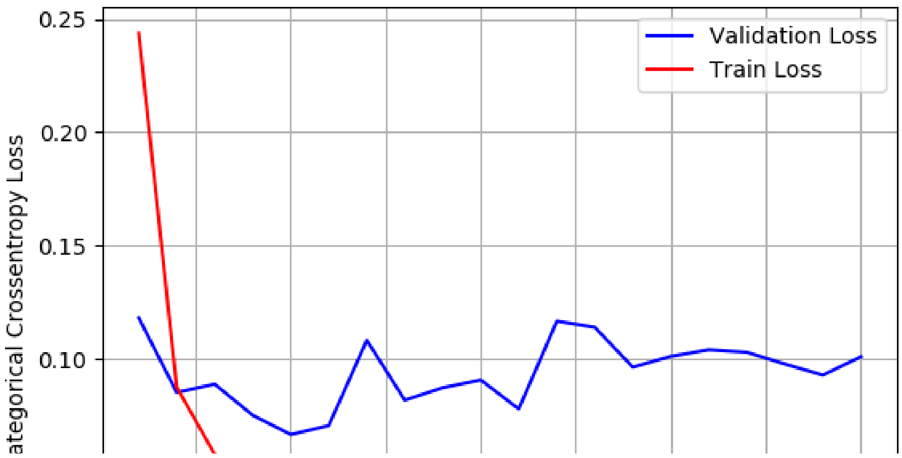
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_19 (Dense)             (None, 352)               276320    
_________________________________________________________________
dense_20 (Dense)             (None, 164)               57892     
_________________________________________________________________
dense_21 (Dense)             (None, 124)               20460     
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1250      
=================================================================
Total params: 355,922
Trainable params: 355,922
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.2438 - acc: 0.9264 -
val_loss: 0.1182 - val_acc: 0.9642
Epoch 2/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0875 - acc: 0.9728 -
val_loss: 0.0852 - val_acc: 0.9727
Epoch 3/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0578 - acc: 0.9819 -
val_loss: 0.0889 - val_acc: 0.9729
Epoch 4/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0429 - acc: 0.9865 -
val_loss: 0.0751 - val_acc: 0.9788
Epoch 5/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0309 - acc: 0.9896 -
val_loss: 0.0666 - val_acc: 0.9815
Epoch 6/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0241 - acc: 0.9918 -
val_loss: 0.0705 - val_acc: 0.9808
Epoch 7/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0204 - acc: 0.9932 -
val_loss: 0.1083 - val_acc: 0.9745
Epoch 8/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0199 - acc: 0.9933 -
val_loss: 0.0818 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0168 - acc: 0.9944 -
val_loss: 0.0873 - val_acc: 0.9789
Epoch 10/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0152 - acc: 0.9948 -
val_loss: 0.0907 - val_acc: 0.9777
```

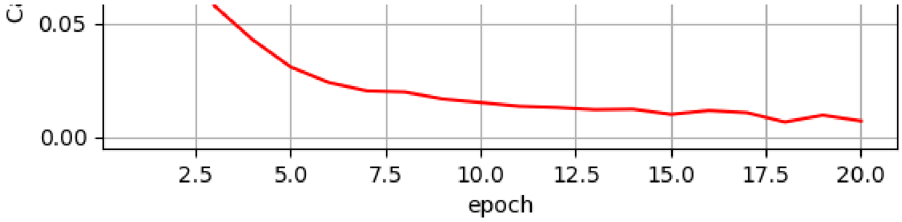
```
Epoch 11/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0136 - acc: 0.9957 -
val_loss: 0.0779 - val_acc: 0.9815
Epoch 12/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0131 - acc: 0.9956 -
val_loss: 0.1167 - val_acc: 0.9751
Epoch 13/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0121 - acc: 0.9959 -
val_loss: 0.1140 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0123 - acc: 0.9957 -
val_loss: 0.0964 - val_acc: 0.9797
Epoch 15/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.0100 - acc: 0.9968 -
val_loss: 0.1011 - val_acc: 0.9794
Epoch 16/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0117 - acc: 0.9960 -
val_loss: 0.1040 - val_acc: 0.9803
Epoch 17/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.0108 - acc: 0.9965 -
val_loss: 0.1029 - val_acc: 0.9791
Epoch 18/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0066 - acc: 0.9979 -
val_loss: 0.0977 - val_acc: 0.9798
Epoch 19/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0097 - acc: 0.9969 -
val_loss: 0.0929 - val_acc: 0.9825
Epoch 20/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0071 - acc: 0.9977 -
val_loss: 0.1009 - val_acc: 0.9798
```

In [36]:

```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score7=score[0]
score8=score[1]
train_acc4=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax21 = plt.subplots(1,1)
ax21.set_xlabel('epoch') ; ax21.set_ylabel('Categorical Crossentropy Loss')

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


vy21 = history21.history['val_loss']
ty21 = history21.history['loss']
plt_dynamic(x, vy21, ty21, ax21)
```

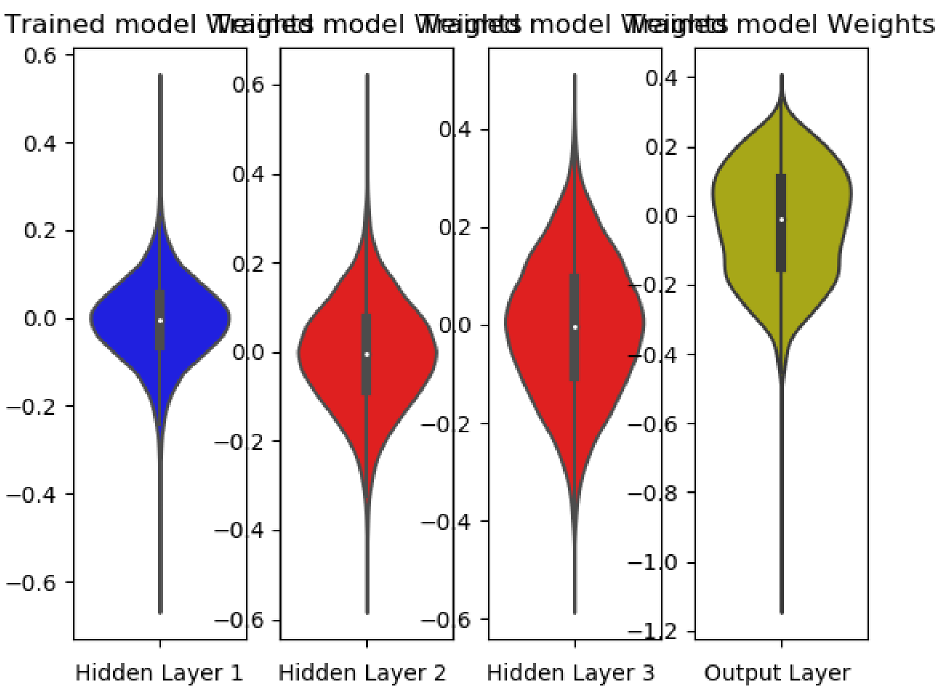
```
Test score: 0.10094069364849065
Test accuracy: 0.9798
```





In [38]:

```
w_after = model_relu.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 + Adadelta

In [39]:

```
model_relu = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adadelta',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history21 = model_relu.fit(X_train, Y_train,
                          batch_size=batch_size,
                          epochs=nb_epoch, verbose=1,
                          validation_data=(X_test, Y_test))
```

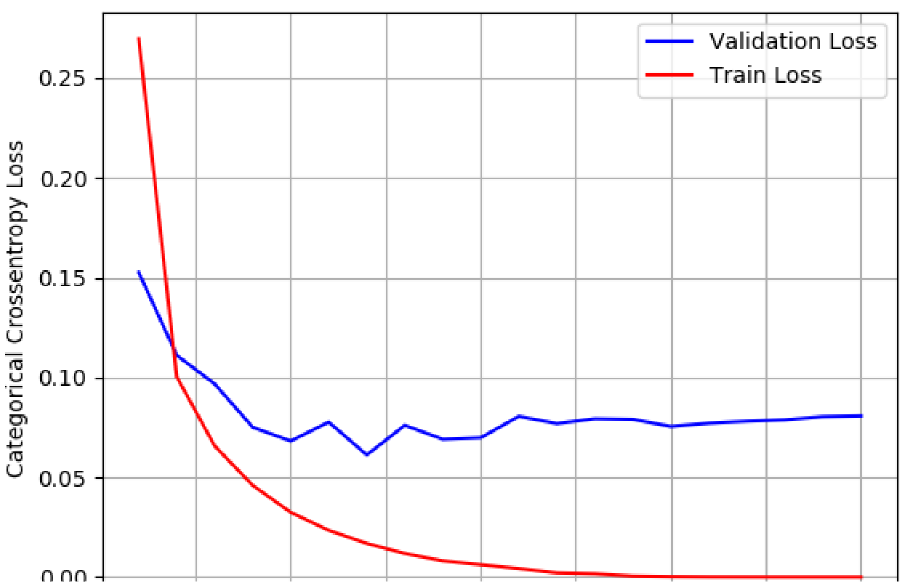
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 352)               276320
_________________________________________________________________
dense_24 (Dense)             (None, 164)               57892
_________________________________________________________________
dense_25 (Dense)             (None, 124)               20460
_________________________________________________________________
dense_26 (Dense)             (None, 10)                1250
=================================================================
Total params: 355,922
Trainable params: 355,922
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.2696 - acc: 0.9177 -
val_loss: 0.1526 - val_acc: 0.9511
Epoch 2/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.1003 - acc: 0.9694 -
val_loss: 0.1111 - val_acc: 0.9633
Epoch 3/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0657 - acc: 0.9804 -
val_loss: 0.0967 - val_acc: 0.9707
Epoch 4/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0460 - acc: 0.9857 -
val_loss: 0.0750 - val_acc: 0.9766
Epoch 5/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0325 - acc: 0.9897 -
val_loss: 0.0682 - val_acc: 0.9802
Epoch 6/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0234 - acc: 0.9927 -
val_loss: 0.0776 - val_acc: 0.9785
Epoch 7/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0169 - acc: 0.9949 -
val_loss: 0.0612 - val_acc: 0.9828
Epoch 8/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.0119 - acc: 0.9964 -
val_loss: 0.0760 - val_acc: 0.9798
Epoch 9/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0082 - acc: 0.9977 -
val_loss: 0.0690 - val_acc: 0.9829
Epoch 10/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0062 - acc: 0.9983 -
val_loss: 0.0698 - val_acc: 0.9841
Epoch 11/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0043 - acc: 0.9988 -
val_loss: 0.0805 - val_acc: 0.9817
Epoch 12/20
```

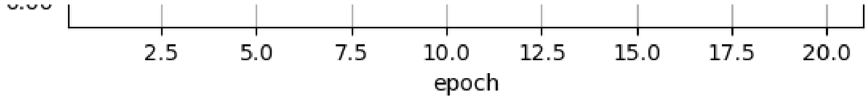
```
60000/60000 [==============================] - 1s 22us/step - loss: 0.0021 - acc: 0.9995 -
val_loss: 0.0769 - val_acc: 0.9828
Epoch 13/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.0017 - acc: 0.9996 -
val_loss: 0.0793 - val_acc: 0.9824
Epoch 14/20
60000/60000 [==============================] - 1s 24us/step - loss: 5.2830e-04 - acc: 0.9999 - val
_loss: 0.0790 - val_acc: 0.9837
Epoch 15/20
60000/60000 [==============================] - 1s 22us/step - loss: 2.4163e-04 - acc: 1.0000 - val
_loss: 0.0754 - val_acc: 0.9857
Epoch 16/20
60000/60000 [==============================] - 1s 22us/step - loss: 1.1272e-04 - acc: 1.0000 - val
_loss: 0.0771 - val_acc: 0.9851
Epoch 17/20
60000/60000 [==============================] - 1s 22us/step - loss: 7.6344e-05 - acc: 1.0000 - val
_loss: 0.0780 - val_acc: 0.9852
Epoch 18/20
60000/60000 [==============================] - 1s 24us/step - loss: 6.4472e-05 - acc: 1.0000 - val
_loss: 0.0787 - val_acc: 0.9853
Epoch 19/20
60000/60000 [==============================] - 2s 25us/step - loss: 5.5575e-05 - acc: 1.0000 - val
_loss: 0.0803 - val_acc: 0.9852
Epoch 20/20
60000/60000 [==============================] - 2s 25us/step - loss: 5.0057e-05 - acc: 1.0000 - val
_loss: 0.0807 - val_acc: 0.9857
```

In [40]:

```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score7=score[0]
score8=score[1]
train_acc4=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax21 = plt.subplots(1,1)
ax21.set_xlabel('epoch') ; ax21.set_ylabel('Categorical Crossentropy Loss')

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


vy21 = history21.history['val_loss']
ty21 = history21.history['loss']
plt_dynamic(x, vy21, ty21, ax21)
```

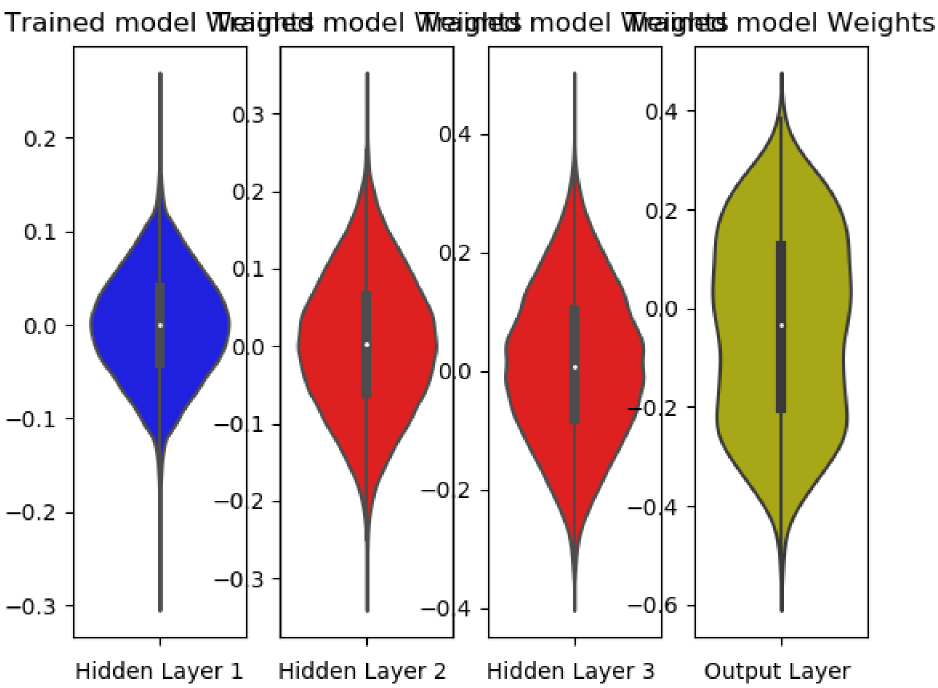
```
Test score: 0.08068556088549227
Test accuracy: 0.9857
```





In [41]:

```
w_after = model_relu.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()
```



2.2 MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [42]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(output_dim, activation='softmax'))
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history22 = model_batch.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 [==============================] - 3s 46us/step - loss: 0.4860 - acc: 0.8565 -
val_loss: 0.6053 - val_acc: 0.8472
Epoch 2/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.3029 - acc: 0.9134 -
val_loss: 2.6513 - val_acc: 0.4629
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2857 - acc: 0.9189 -
val_loss: 0.8559 - val_acc: 0.7858
Epoch 4/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2774 - acc: 0.9219 -
val_loss: 0.8852 - val_acc: 0.7914
Epoch 5/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2712 - acc: 0.9236 -
val_loss: 7.8901 - val_acc: 0.1965
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2689 - acc: 0.9243 -
val_loss: 1.0264 - val_acc: 0.7722
Epoch 7/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2669 - acc: 0.9258 -
val_loss: 1.1807 - val_acc: 0.7158
Epoch 8/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2635 - acc: 0.9253 -
val_loss: 1.4079 - val_acc: 0.6595
Epoch 9/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2624 - acc: 0.9263 -
val_loss: 1.6847 - val_acc: 0.6442
Epoch 10/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2591 - acc: 0.9277 -
val_loss: 2.0835 - val_acc: 0.5957
Epoch 11/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.2597 - acc: 0.9271 -
val_loss: 0.7050 - val_acc: 0.8390
Epoch 12/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2570 - acc: 0.9278 -
val_loss: 5.7033 - val_acc: 0.4569
Epoch 13/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2560 - acc: 0.9289 -
val_loss: 2.0083 - val_acc: 0.5987
Epoch 14/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2576 - acc: 0.9276 -
val_loss: 1.5552 - val_acc: 0.6973
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2544 - acc: 0.9284 -
val_loss: 9.8770 - val_acc: 0.1527
Epoch 16/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2553 - acc: 0.9274 -
val_loss: 0.7170 - val_acc: 0.8289
Epoch 17/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2518 - acc: 0.9290 -
val_loss: 3.4736 - val_acc: 0.4885
Epoch 18/20
```

```
Epoch 18/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.2524 - acc: 0.9293 -
val_loss: 2.1239 - val_acc: 0.5799
Epoch 19/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2517 - acc: 0.9287 -
val_loss: 0.6194 - val_acc: 0.8697
Epoch 20/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2522 - acc: 0.9295 -
val_loss: 1.0895 - val_acc: 0.7509
```

In [43]:

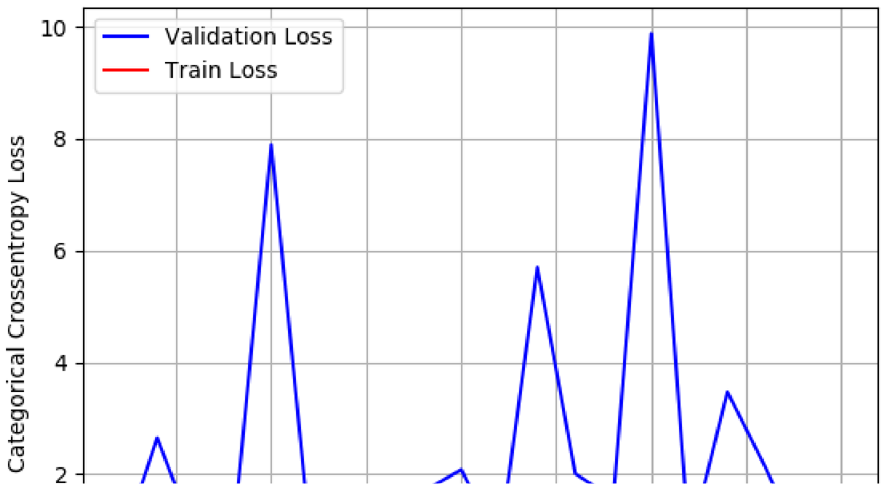
```
model_batch.summary()
```

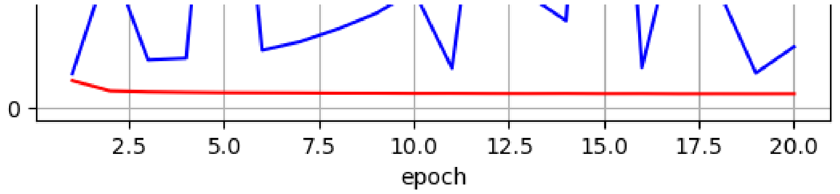
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_9 (Batch (None, 784)               3136      
_________________________________________________________________
batch_normalization_10 (Batc (None, 784)               3136      
_________________________________________________________________
batch_normalization_11 (Batc (None, 784)               3136      
_________________________________________________________________
dense_30 (Dense)             (None, 10)                7850      
=================================================================
Total params: 17,258
Trainable params: 12,554
Non-trainable params: 4,704
_________________________________________________________________
```

In [44]:

```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
score9=score[0]
score10=score[1]
train_acc5=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax22 = plt.subplots(1,1)
ax22.set_xlabel('epoch') ; ax22.set_ylabel('Categorical Crossentropy Loss')

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

vy22 = history22.history['val_loss']
ty22 = history22.history['loss']
plt_dynamic(x, vy22, ty22, ax22)
```

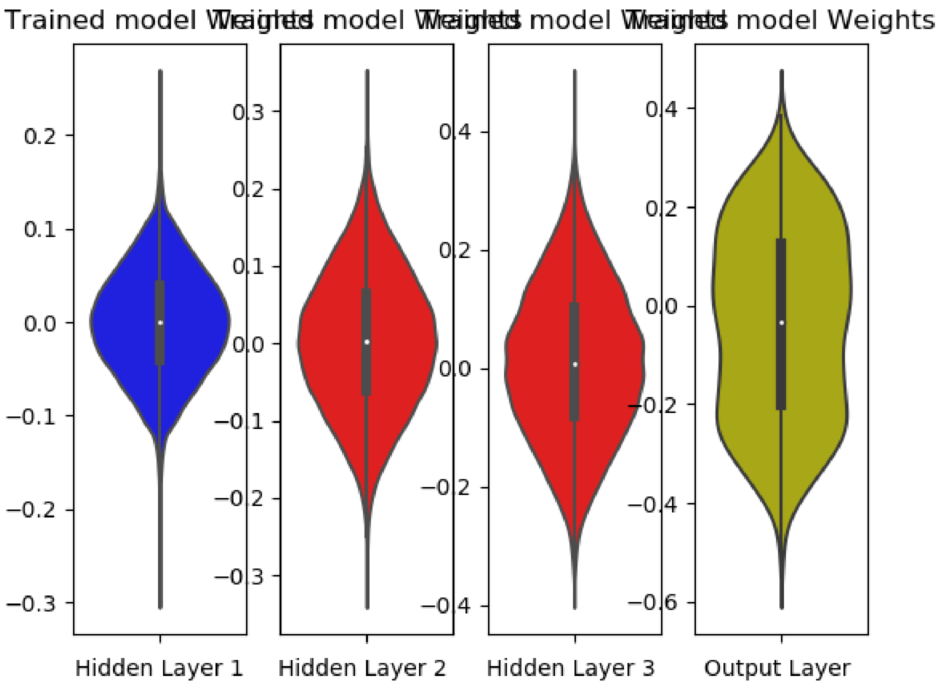
```
Test score: 1.089544097495079
Test accuracy: 0.7509
```





In [45]:

```
w_after = model_relu.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 + Batch-Norm on hidden Layers + RMS Prop Optimizer

In [46]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(output_dim, activation='softmax'))
model_batch.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history22 = model_batch.fit(X_train, Y_train,
                            batch_size=batch_size,
                            epochs=nb_epoch, verbose=1,
                            validation_data=(X_test, Y_test))
```

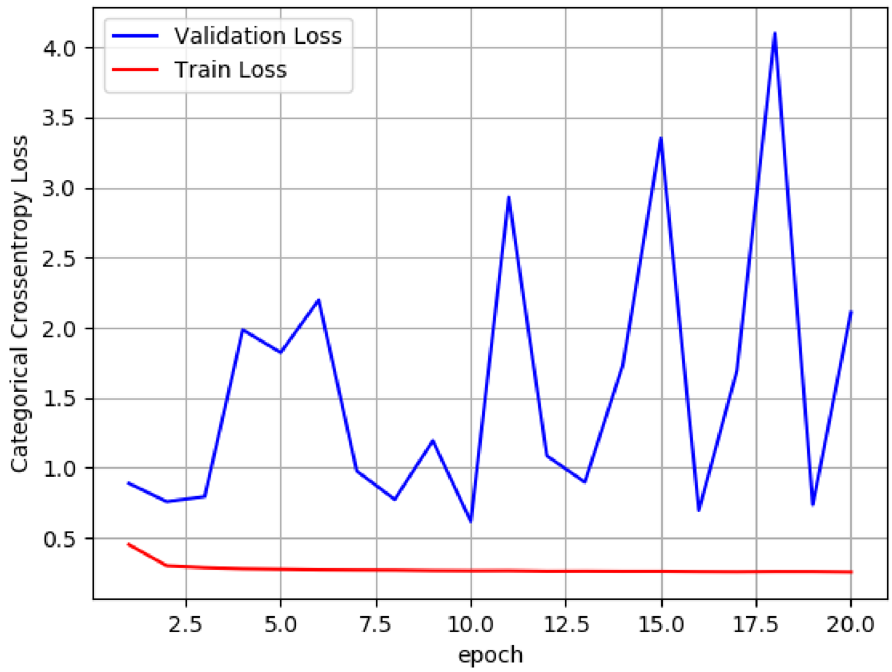
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.4528 - acc: 0.8681 -
val_loss: 0.8893 - val_acc: 0.7956
Epoch 2/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.3014 - acc: 0.9146 -
val_loss: 0.7591 - val_acc: 0.7998
Epoch 3/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.2882 - acc: 0.9201 -
val_loss: 0.7948 - val_acc: 0.7697
Epoch 4/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.2804 - acc: 0.9223 -
val_loss: 1.9841 - val_acc: 0.6465
Epoch 5/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2769 - acc: 0.9234 -
val_loss: 1.8222 - val_acc: 0.5741
Epoch 6/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2731 - acc: 0.9244 -
val_loss: 2.1971 - val_acc: 0.5198
Epoch 7/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2717 - acc: 0.9256 -
val_loss: 0.9766 - val_acc: 0.8075
Epoch 8/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2707 - acc: 0.9262 -
val_loss: 0.7744 - val_acc: 0.8175
Epoch 9/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2674 - acc: 0.9268 -
val_loss: 1.1930 - val_acc: 0.7625
Epoch 10/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.2660 - acc: 0.9273 -
val_loss: 0.6178 - val_acc: 0.8532
Epoch 11/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2664 - acc: 0.9272 -
val_loss: 2.9305 - val_acc: 0.5530
Epoch 12/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2630 - acc: 0.9278 -
val_loss: 1.0858 - val_acc: 0.7395
Epoch 13/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2635 - acc: 0.9283 -
val_loss: 0.8976 - val_acc: 0.8095
Epoch 14/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2624 - acc: 0.9295 -
val_loss: 1.7319 - val_acc: 0.6818
Epoch 15/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2624 - acc: 0.9292 -
val_loss: 3.3493 - val_acc: 0.5509
Epoch 16/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2602 - acc: 0.9294 -
val_loss: 0.6959 - val_acc: 0.8367
```

```
Epoch 17/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.2591 - acc: 0.9293 -
val_loss: 1.6922 - val_acc: 0.6514
Epoch 18/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.2607 - acc: 0.9294 -
val_loss: 4.0993 - val_acc: 0.4954
Epoch 19/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.2604 - acc: 0.9295 -
val_loss: 0.7377 - val_acc: 0.8301
Epoch 20/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.2577 - acc: 0.9292 -
val_loss: 2.1088 - val_acc: 0.5618
```

In [47]:

```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
score9=score[0]
score10=score[1]
train_acc5=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax22 = plt.subplots(1,1)
ax22.set_xlabel('epoch') ; ax22.set_ylabel('Categorical Crossentropy Loss')

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

vy22 = history22.history['val_loss']
ty22 = history22.history['loss']
plt_dynamic(x, vy22, ty22, ax22)
```

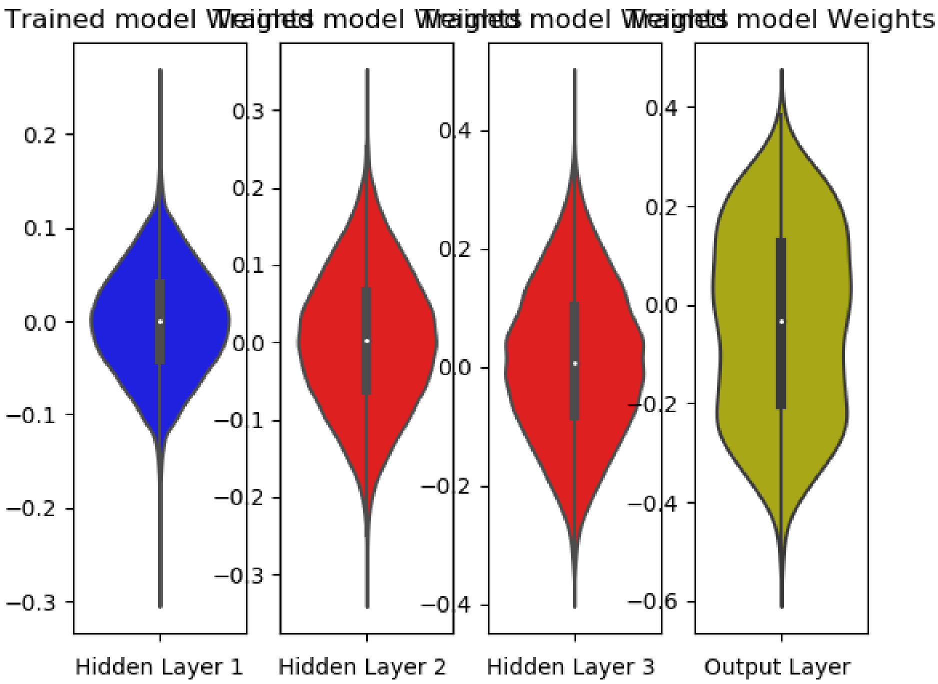
```
Test score: 2.1087711837768555
Test accuracy: 0.5618
```



In [48]:

```
w_after = model_relu.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()
```



2.3 MLP + Dropout + AdamOptimizer

In [49]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))

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

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

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

In [50]:

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

history23 = model_drop.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 [==============================] - 3s 54us/step - loss: 1.3786 - acc: 0.5656 -
val_loss: 0.5066 - val_acc: 0.8767
Epoch 2/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8568 - acc: 0.7204 -
val_loss: 0.4589 - val_acc: 0.8899
Epoch 3/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8374 - acc: 0.7253 -
val_loss: 0.4385 - val_acc: 0.8927
Epoch 4/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8287 - acc: 0.7279 -
val_loss: 0.4260 - val_acc: 0.8960
Epoch 5/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.8302 - acc: 0.7317 -
val_loss: 0.4264 - val_acc: 0.8952
Epoch 6/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8247 - acc: 0.7319 -
val_loss: 0.4230 - val_acc: 0.8946
Epoch 7/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.8203 - acc: 0.7321 -
val_loss: 0.4159 - val_acc: 0.8973
Epoch 8/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8212 - acc: 0.7347 -
val_loss: 0.4140 - val_acc: 0.8924
Epoch 9/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.8182 - acc: 0.7335 -
val_loss: 0.4209 - val_acc: 0.8962
Epoch 10/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8195 - acc: 0.7338 -
val_loss: 0.4100 - val_acc: 0.8968
Epoch 11/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8209 - acc: 0.7330 -
val_loss: 0.4054 - val_acc: 0.8982
Epoch 12/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8197 - acc: 0.7331 -
val_loss: 0.4080 - val_acc: 0.8968
Epoch 13/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.8123 - acc: 0.7372 -
val_loss: 0.4037 - val_acc: 0.8977
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.8109 - acc: 0.7377 -
val_loss: 0.4071 - val_acc: 0.8982
Epoch 15/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8165 - acc: 0.7362 -
val_loss: 0.4074 - val_acc: 0.8971
Epoch 16/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8080 - acc: 0.7385 -
val_loss: 0.4075 - val_acc: 0.8977
Epoch 17/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8045 - acc: 0.7400 -
val_loss: 0.4034 - val_acc: 0.8983
Epoch 18/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8125 - acc: 0.7369 -
val_loss: 0.4040 - val_acc: 0.9003
```

```
val_loss: 0.4040 - val_acc: 0.9003
Epoch 19/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.8083 - acc: 0.7378 -
val_loss: 0.4047 - val_acc: 0.8975
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.8052 - acc: 0.7413 -
val_loss: 0.4010 - val_acc: 0.8984
```

In [51]:

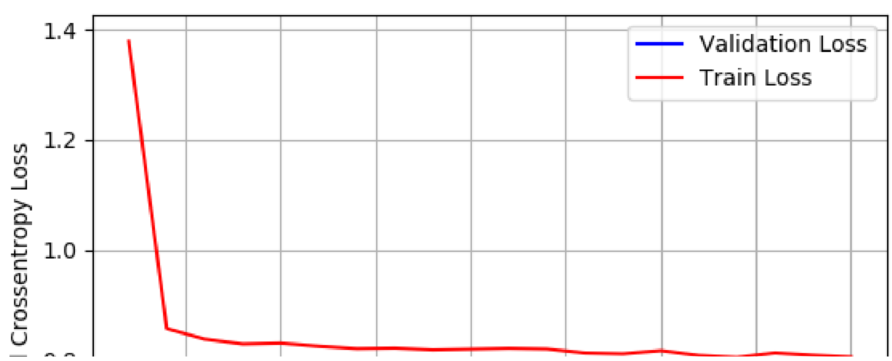
```python
model_drop.summary()
```

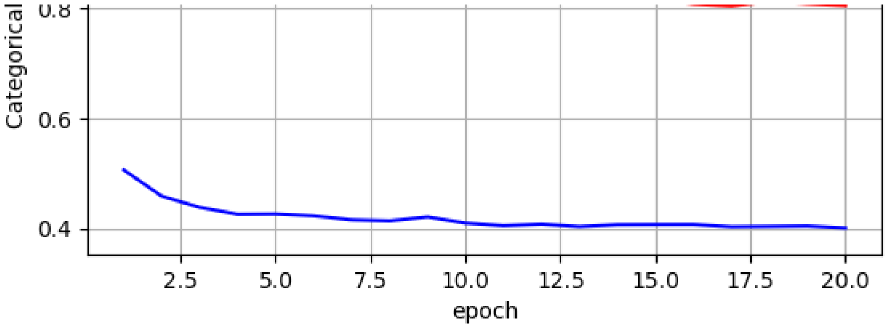
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_15 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_5 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_16 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_6 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_17 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_7 (Dropout)          (None, 784)               0         
_________________________________________________________________
dense_38 (Dense)             (None, 10)                7850      
=================================================================
Total params: 17,258
Trainable params: 12,554
Non-trainable params: 4,704
_________________________________________________________________
```

In [52]:

```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score11=score[0]
score12=score[1]
train_acc6=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax23 = plt.subplots(1,1)
ax23.set_xlabel('epoch') ; ax23.set_ylabel('Categorical Crossentropy Loss')

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


vy23 = history23.history['val_loss']
ty23 = history23.history['loss']
plt_dynamic(x, vy23, ty23, ax23)
```

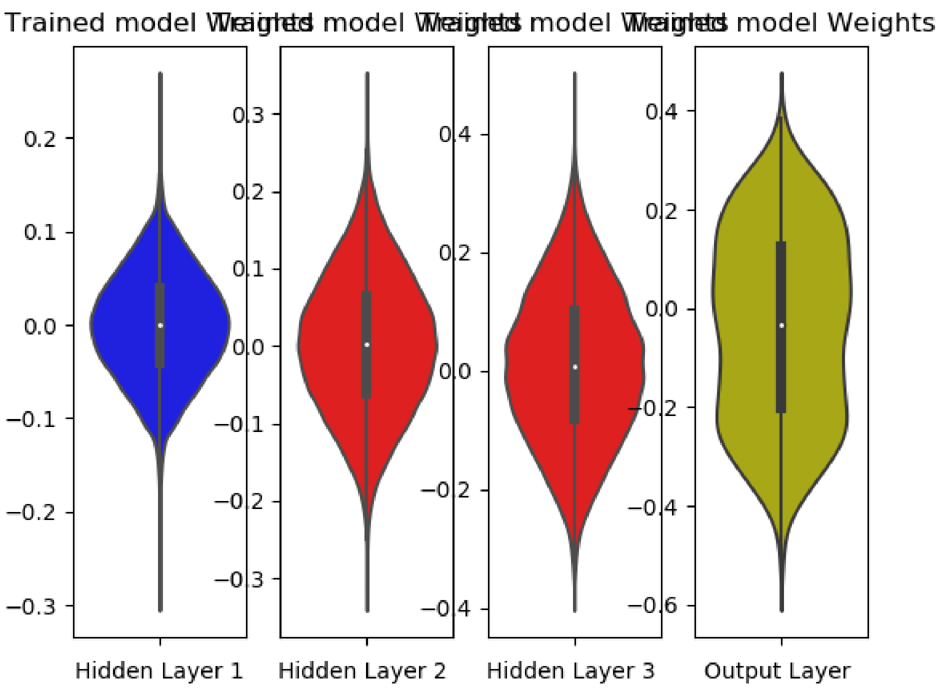
```
Test score: 0.40103902480602266
Test accuracy: 0.8984
```





In [53]:

```
w_after = model_relu.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 + with Different dropout rates + RMS Prop Optimizer

In [54]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-
keras

from keras.layers import Dropout

model_drop = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))

model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.3))


model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.4))

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

model_drop.compile(optimizer='rmsprop',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history23 = model_drop.fit(X_train, Y_train,
                           batch_size=batch_size,

                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, Y_test))
```

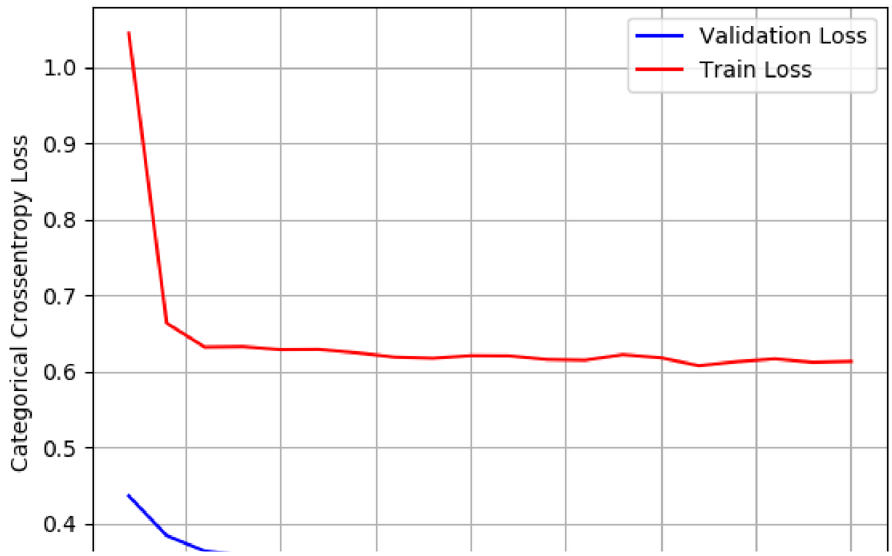
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 53us/step - loss: 1.0449 - acc: 0.6674 -
val_loss: 0.4368 - val_acc: 0.8891
Epoch 2/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6637 - acc: 0.7915 -
val_loss: 0.3845 - val_acc: 0.8983
Epoch 3/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6323 - acc: 0.8020 -
val_loss: 0.3641 - val_acc: 0.9003
Epoch 4/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6329 - acc: 0.8011 -
val_loss: 0.3580 - val_acc: 0.9036
Epoch 5/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6290 - acc: 0.8030 -
val_loss: 0.3571 - val_acc: 0.9047
Epoch 6/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6293 - acc: 0.8032 -
val_loss: 0.3534 - val_acc: 0.9050
Epoch 7/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6248 - acc: 0.8047 -
val_loss: 0.3550 - val_acc: 0.9031
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.6191 - acc: 0.8059 -
val_loss: 0.3503 - val_acc: 0.9058
Epoch 9/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6176 - acc: 0.8060 -
val_loss: 0.3455 - val_acc: 0.9072
Epoch 10/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6211 - acc: 0.8079 -
val_loss: 0.3505 - val_acc: 0.9052
Epoch 11/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6208 - acc: 0.8065 -
val_loss: 0.3435 - val_acc: 0.9051
```

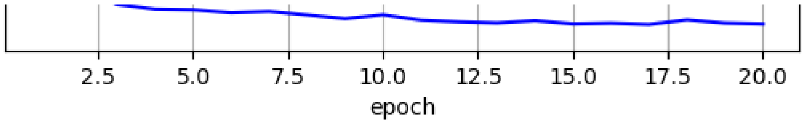
```
Epoch 12/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6160 - acc: 0.8074 -
val_loss: 0.3414 - val_acc: 0.9057
Epoch 13/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6151 - acc: 0.8092 -
val_loss: 0.3399 - val_acc: 0.9087
Epoch 14/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6223 - acc: 0.8082 -
val_loss: 0.3429 - val_acc: 0.9075
Epoch 15/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6183 - acc: 0.8070 -
val_loss: 0.3386 - val_acc: 0.9094
Epoch 16/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.6078 - acc: 0.8110 -
val_loss: 0.3395 - val_acc: 0.9069
Epoch 17/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6131 - acc: 0.8093 -
val_loss: 0.3380 - val_acc: 0.9068
Epoch 18/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6169 - acc: 0.8086 -
val_loss: 0.3439 - val_acc: 0.9047
Epoch 19/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6122 - acc: 0.8114 -
val_loss: 0.3397 - val_acc: 0.9092
Epoch 20/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.6135 - acc: 0.8109 -
val_loss: 0.3385 - val_acc: 0.9068
```

In [55]:

```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score11=score[0]
score12=score[1]
train_acc6=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax23 = plt.subplots(1,1)
ax23.set_xlabel('epoch') ; ax23.set_ylabel('Categorical Crossentropy Loss')

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


vy23 = history23.history['val_loss']
ty23 = history23.history['loss']
plt_dynamic(x, vy23, ty23, ax23)
```

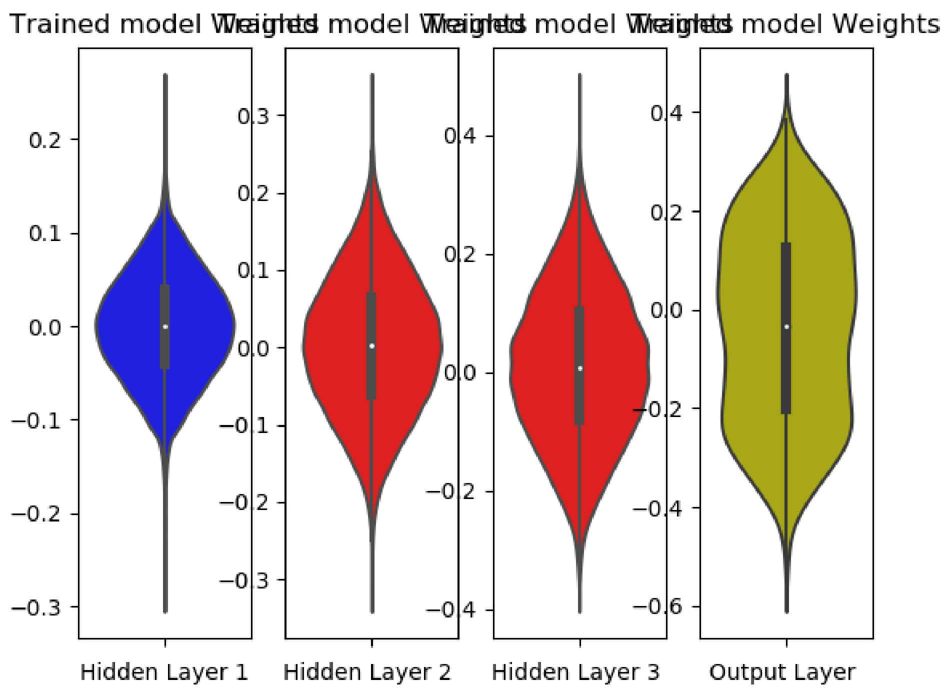
```
Test score: 0.33852629685401914
Test accuracy: 0.9068
```





In [56]:

```
w_after = model_relu.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()
```



3) 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture)

3.1 MLP + ReLU + ADAM

In [57]:

```
model_relu = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history31 = model_relu.fit(X_train, Y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, Y_test))
```

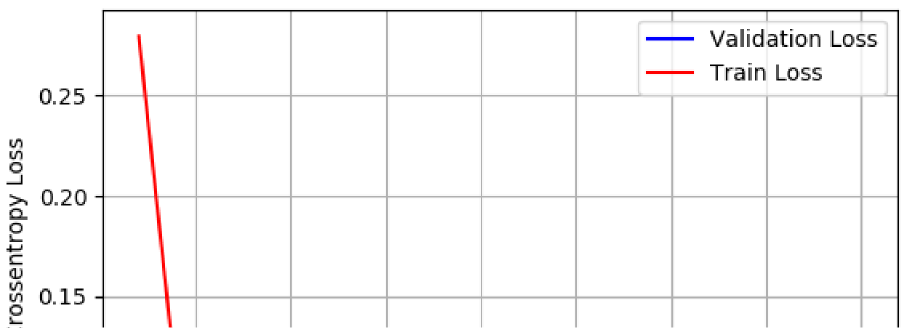
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_43 (Dense)             (None, 216)               169560    
_________________________________________________________________
dense_44 (Dense)             (None, 170)               36890     
_________________________________________________________________
dense_45 (Dense)             (None, 136)               23256     
_________________________________________________________________
dense_46 (Dense)             (None, 80)                10960     
_________________________________________________________________
dense_47 (Dense)             (None, 38)                3078      
_________________________________________________________________
dense_48 (Dense)             (None, 10)                390       
=================================================================
Total params: 244,134
Trainable params: 244,134
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2795 - acc: 0.9164 - val_loss: 0.1212 - val_acc: 0.9622
Epoch 2/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.1043 - acc: 0.9682 - val_loss: 0.1048 - val_acc: 0.9681
Epoch 3/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0724 - acc: 0.9777 - val_loss: 0.0897 - val_acc: 0.9719
Epoch 4/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0546 - acc: 0.9829 - val_loss: 0.0902 - val_acc: 0.9736
Epoch 5/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.0425 - acc: 0.9867 - val_loss: 0.0855 - val_acc: 0.9751
Epoch 6/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.0352 - acc: 0.9886 - val_loss: 0.0867 - val_acc: 0.9770
Epoch 7/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.0343 - acc: 0.9890 - val_loss: 0.0845 - val_acc: 0.9758
Epoch 8/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0269 - acc: 0.9910 - val_loss: 0.0978 - val_acc: 0.9738
Epoch 9/20
```

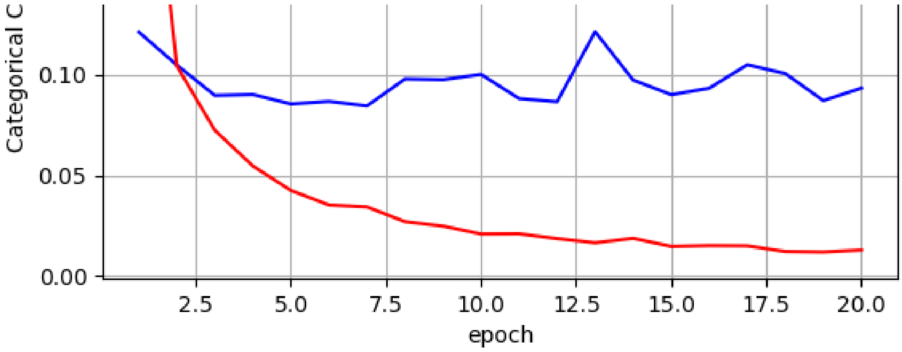
```
60000/60000 [==============================] - 2s 26us/step - loss: 0.0247 - acc: 0.9919 -
val_loss: 0.0975 - val_acc: 0.9766
Epoch 10/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0208 - acc: 0.9929 -
val_loss: 0.1001 - val_acc: 0.9743
Epoch 11/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.0209 - acc: 0.9932 -
val_loss: 0.0882 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.0185 - acc: 0.9943 -
val_loss: 0.0866 - val_acc: 0.9787
Epoch 13/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.0164 - acc: 0.9947 -
val_loss: 0.1214 - val_acc: 0.9752
Epoch 14/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0186 - acc: 0.9940 -
val_loss: 0.0973 - val_acc: 0.9767
Epoch 15/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0147 - acc: 0.9952 -
val_loss: 0.0901 - val_acc: 0.9800
Epoch 16/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0151 - acc: 0.9953 -
val_loss: 0.0932 - val_acc: 0.9779
Epoch 17/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0149 - acc: 0.9952 -
val_loss: 0.1049 - val_acc: 0.9758
Epoch 18/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0121 - acc: 0.9960 -
val_loss: 0.1006 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0119 - acc: 0.9963 -
val_loss: 0.0871 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0129 - acc: 0.9961 -
val_loss: 0.0933 - val_acc: 0.9778
```

In [58]:

```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score13=score[0]
score14=score[1]
train_acc7=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax31 = plt.subplots(1,1)
ax31.set_xlabel('epoch') ; ax31.set_ylabel('Categorical Crossentropy Loss')

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


vy31 = history31.history['val_loss']
ty31 = history31.history['loss']
plt_dynamic(x, vy31, ty31, ax31)
```

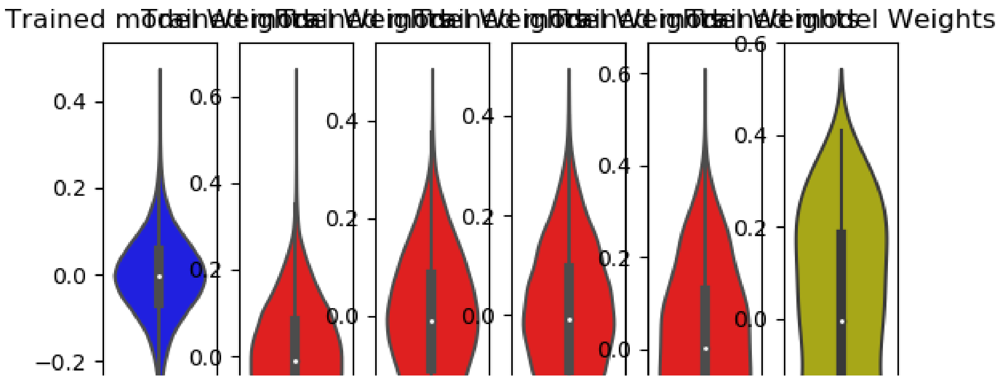
```
Test score: 0.09329951099789031
Test accuracy: 0.9778
```

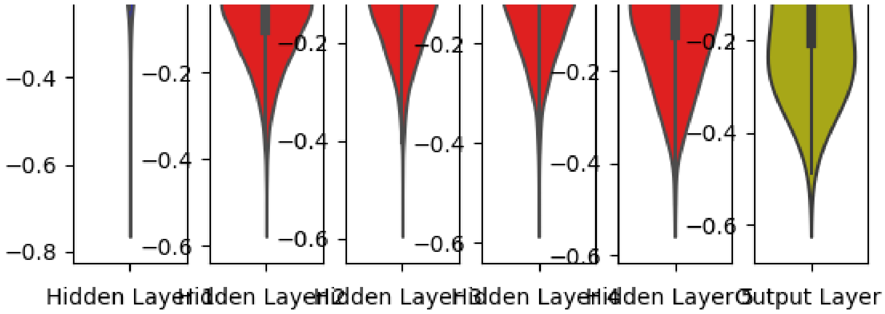




In [60]:

```
w_after = model_relu.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=h5_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()
```





MLP + ReLU + rmsprop

In [61]:

```
model_relu = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='rmsprop',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history31 = model_relu.fit(X_train, Y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, Y_test))
```

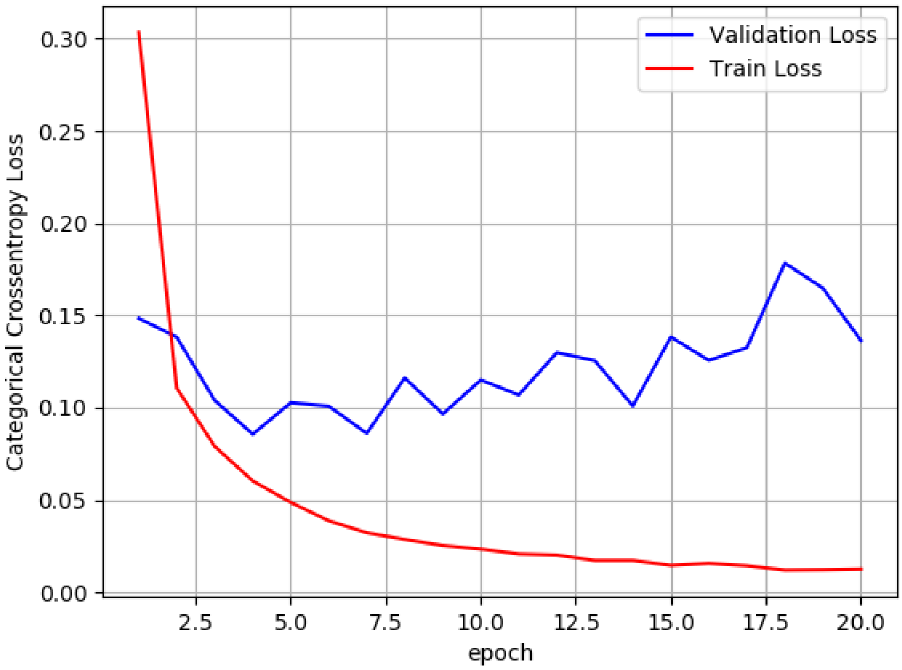
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_49 (Dense)             (None, 216)               169560
_________________________________________________________________
dense_50 (Dense)             (None, 170)               36890
_________________________________________________________________
dense_51 (Dense)             (None, 136)               23256
_________________________________________________________________
dense_52 (Dense)             (None, 80)                10960
_________________________________________________________________
dense_53 (Dense)             (None, 38)                3078
_________________________________________________________________
dense_54 (Dense)             (None, 10)                390
=================================================================
Total params: 244,134
Trainable params: 244,134
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.3035 - acc: 0.9089 -
val_loss: 0.1484 - val_acc: 0.9545
Epoch 2/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.1106 - acc: 0.9665 -
val_loss: 0.1382 - val_acc: 0.9611
Epoch 3/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.0790 - acc: 0.9763 -
val_loss: 0.1039 - val_acc: 0.9671
Epoch 4/20
```

```
Epoch 4/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0602 - acc: 0.9813 -
val_loss: 0.0856 - val_acc: 0.9762
Epoch 5/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.0487 - acc: 0.9856 -
val_loss: 0.1027 - val_acc: 0.9721
Epoch 6/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.0387 - acc: 0.9877 -
val_loss: 0.1008 - val_acc: 0.9748
Epoch 7/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.0322 - acc: 0.9900 -
val_loss: 0.0860 - val_acc: 0.9792
Epoch 8/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0286 - acc: 0.9913 -
val_loss: 0.1161 - val_acc: 0.9760
Epoch 9/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0253 - acc: 0.9924 -
val_loss: 0.0966 - val_acc: 0.9795
Epoch 10/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0233 - acc: 0.9934 -
val_loss: 0.1150 - val_acc: 0.9767
Epoch 11/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0207 - acc: 0.9940 -
val_loss: 0.1069 - val_acc: 0.9789
Epoch 12/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0201 - acc: 0.9943 -
val_loss: 0.1299 - val_acc: 0.9768
Epoch 13/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0171 - acc: 0.9953 -
val_loss: 0.1255 - val_acc: 0.9801
Epoch 14/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0172 - acc: 0.9949 -
val_loss: 0.1009 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0145 - acc: 0.9959 -
val_loss: 0.1383 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0156 - acc: 0.9959 -
val_loss: 0.1256 - val_acc: 0.9781
Epoch 17/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.0143 - acc: 0.9964 -
val_loss: 0.1326 - val_acc: 0.9807
Epoch 18/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0120 - acc: 0.9969 -
val_loss: 0.1784 - val_acc: 0.9779
Epoch 19/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0121 - acc: 0.9968 -
val_loss: 0.1648 - val_acc: 0.9780
Epoch 20/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0125 - acc: 0.9968 -
val_loss: 0.1364 - val_acc: 0.9792
```

In [62]:

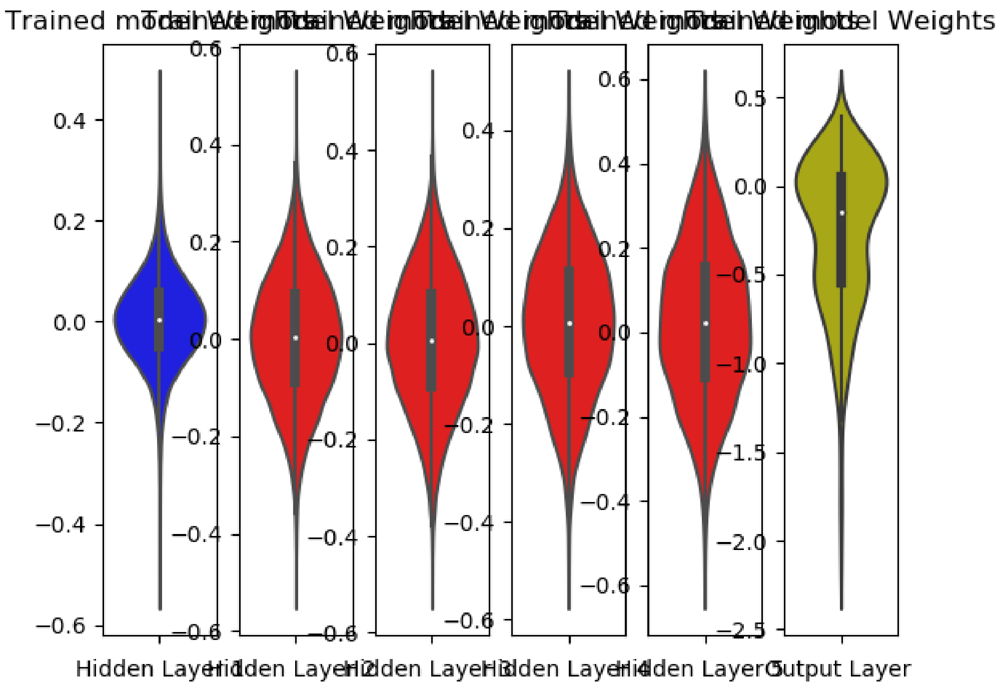
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score13=score[0]
score14=score[1]
train_acc7=history11.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax31 = plt.subplots(1,1)
ax31.set_xlabel('epoch') ; ax31.set_ylabel('Categorical Crossentropy Loss')

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


vy31 = history31.history['val_loss']
ty31 = history31.history['loss']
plt_dynamic(x, vy31, ty31, ax31)
```

```
Test score: 0.13639087163173536
Test accuracy: 0.9792
```



In [63]:

```
w_after = model_relu.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=h5_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()
```



3.2 MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [64]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

In [65]:

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

history32 = model_batch.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 [==============================] - 4s 74us/step - loss: 0.4931 - acc: 0.8537 -
val_loss: 14.2806 - val_acc: 0.1140
Epoch 2/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.3048 - acc: 0.9124 -
```

```
val_loss: 14.5466 - val_acc: 0.0975
Epoch 3/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2858 - acc: 0.9185 -
val_loss: 14.4644 - val_acc: 0.1026
Epoch 4/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2785 - acc: 0.9198 -
val_loss: 14.4126 - val_acc: 0.1058
Epoch 5/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2734 - acc: 0.9234 -
val_loss: 14.2581 - val_acc: 0.1154
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2683 - acc: 0.9232 -
val_loss: 14.2887 - val_acc: 0.1135
Epoch 7/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2662 - acc: 0.9248 -
val_loss: 14.5337 - val_acc: 0.0983
Epoch 8/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2645 - acc: 0.9260 -
val_loss: 14.2887 - val_acc: 0.1135
Epoch 9/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2615 - acc: 0.9261 -
val_loss: 14.5498 - val_acc: 0.0973
Epoch 10/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2603 - acc: 0.9273 -
val_loss: 14.4902 - val_acc: 0.1010
Epoch 11/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2595 - acc: 0.9271 -
val_loss: 14.2855 - val_acc: 0.1137
Epoch 12/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.2561 - acc: 0.9278 -
val_loss: 14.4740 - val_acc: 0.1020
Epoch 13/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2563 - acc: 0.9276 -
val_loss: 14.4878 - val_acc: 0.1011
Epoch 14/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2550 - acc: 0.9285 -
val_loss: 14.5450 - val_acc: 0.0976
Epoch 15/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2546 - acc: 0.9289 -
val_loss: 14.4660 - val_acc: 0.1025
Epoch 16/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2529 - acc: 0.9285 -
val_loss: 14.5514 - val_acc: 0.0972
Epoch 17/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2541 - acc: 0.9288 -
val_loss: 14.4886 - val_acc: 0.1011
Epoch 18/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2515 - acc: 0.9299 -
val_loss: 14.6723 - val_acc: 0.0897
Epoch 19/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2518 - acc: 0.9300 -
val_loss: 14.2887 - val_acc: 0.1135
Epoch 20/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2533 - acc: 0.9285 -
val_loss: 14.5466 - val_acc: 0.0975
```

In [66]:

```
model_batch.summary()
```

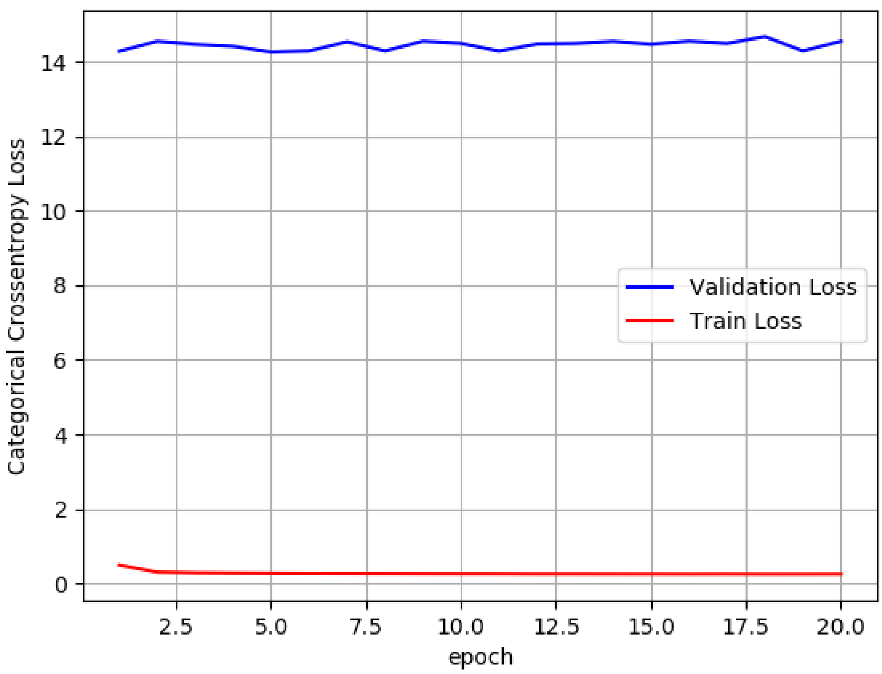
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_21 (Batc (None, 784)               3136
_________________________________________________________________
batch_normalization_22 (Batc (None, 784)               3136
_________________________________________________________________
batch_normalization_23 (Batc (None, 784)               3136
_________________________________________________________________
batch_normalization_24 (Batc (None, 784)               3136
_________________________________________________________________
batch_normalization_25 (Batc (None, 784)               3136
_________________________________________________________________
dense_60 (Dense)             (None, 10)                7850
=================================================================
```

```
Total params: 23,530
Trainable params: 15,690
Non-trainable params: 7,840
_________________________________________________________________
```

In [67]:

```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
score15=score[0]
score16=score[1]
train_acc8=history32.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax32 = plt.subplots(1,1)
ax32.set_xlabel('epoch') ; ax32.set_ylabel('Categorical Crossentropy Loss')

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

vy32 = history32.history['val_loss']
ty32 = history32.history['loss']
plt_dynamic(x, vy32, ty32, ax32)
```

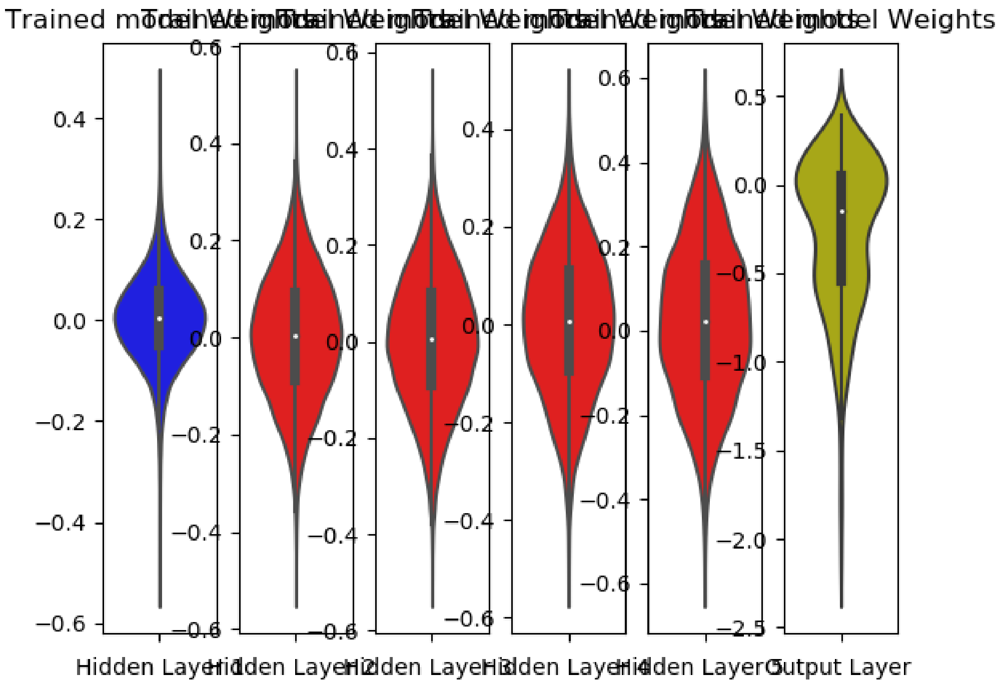
```
Test score: 14.546580853271484
Test accuracy: 0.0975
```



In [68]:

```
w_after = model_relu.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=h5_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()
```



MLP + Batch-Norm on hidden Layers + Adadelta

In [69]:

```
model_batch = Sequential()

model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
```

```
model_relu.add(Dense(80, activation='relu',
                    kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(38, activation='relu',
                    kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

In [70]:

```
model_batch.compile(optimizer='adadelta', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history32 = model_batch.fit(X_train, Y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, Y_test))
```

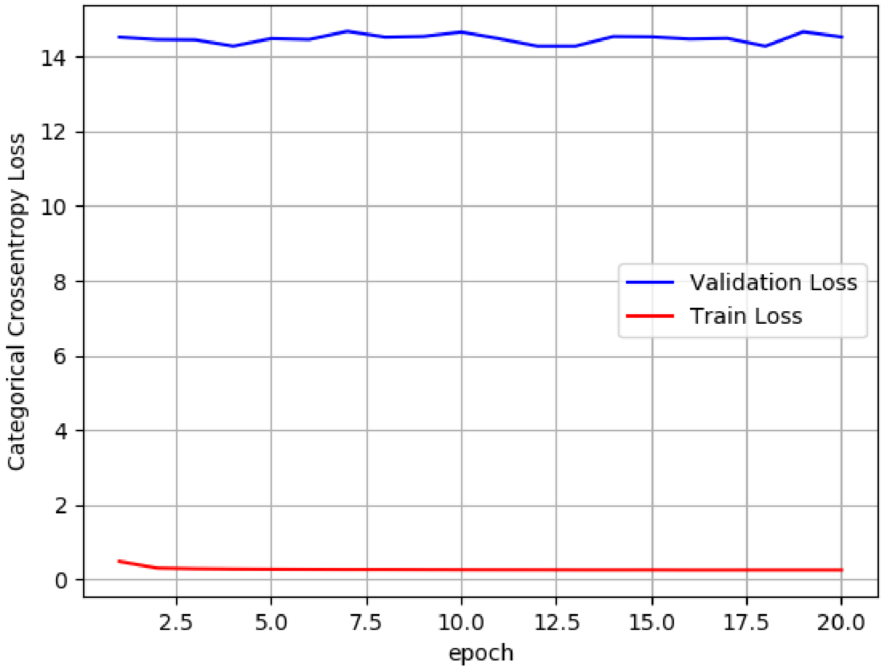
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.4841 - acc: 0.8597 -
val_loss: 14.5256 - val_acc: 0.0988
Epoch 2/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.3068 - acc: 0.9132 -
val_loss: 14.4612 - val_acc: 0.1028
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.2891 - acc: 0.9184 -
val_loss: 14.4525 - val_acc: 0.1032
Epoch 4/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2803 - acc: 0.9213 -
val_loss: 14.2806 - val_acc: 0.1140
Epoch 5/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2754 - acc: 0.9231 -
val_loss: 14.4886 - val_acc: 0.1011
Epoch 6/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2723 - acc: 0.9242 -
val_loss: 14.4628 - val_acc: 0.1027
Epoch 7/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.2694 - acc: 0.9256 -
val_loss: 14.6820 - val_acc: 0.0891
Epoch 8/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.2693 - acc: 0.9252 -
val_loss: 14.5233 - val_acc: 0.0989
Epoch 9/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.2664 - acc: 0.9268 -
val_loss: 14.5422 - val_acc: 0.0977
Epoch 10/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2642 - acc: 0.9260 -
val_loss: 14.6614 - val_acc: 0.0903
Epoch 11/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2625 - acc: 0.9281 -
val_loss: 14.4832 - val_acc: 0.1013
Epoch 12/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2619 - acc: 0.9269 -
val_loss: 14.2806 - val_acc: 0.1140
Epoch 13/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2598 - acc: 0.9288 -
val_loss: 14.2806 - val_acc: 0.1140
Epoch 14/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2590 - acc: 0.9287 -
val_loss: 14.5417 - val_acc: 0.0978
Epoch 15/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2594 - acc: 0.9279 -
val_loss: 14.5337 - val_acc: 0.0983
Epoch 16/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2565 - acc: 0.9295 -
val_loss: 14.4757 - val_acc: 0.1019
Epoch 17/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.2565 - acc: 0.9300 -
val_loss: 14.4943 - val_acc: 0.1007
Epoch 18/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2571 - acc: 0.9294 -
val_loss: 14.2790 - val_acc: 0.1141
Epoch 19/20
```

```
60000/60000 [==============================] - 3s 47us/step - loss: 0.2569 - acc: 0.9288 -
val_loss: 14.6698 - val_acc: 0.0897
Epoch 20/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.2567 - acc: 0.9287 -
val_loss: 14.5289 - val_acc: 0.0986
```

In [71]:

```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
score15=score[0]
score16=score[1]
train_acc8=history32.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax32 = plt.subplots(1,1)
ax32.set_xlabel('epoch') ; ax32.set_ylabel('Categorical Crossentropy Loss')

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

vy32 = history32.history['val_loss']
ty32 = history32.history['loss']
plt_dynamic(x, vy32, ty32, ax32)
```

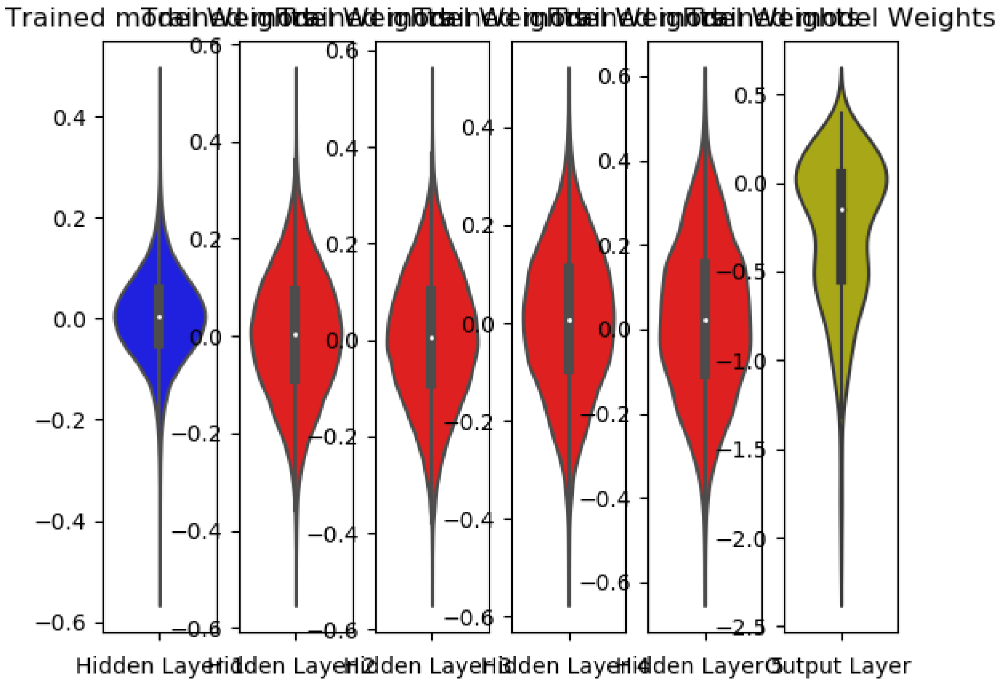
```
Test score: 14.528850985717773
Test accuracy: 0.0986
```



In [72]:

```
w_after = model_relu.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")
```

```
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=h5_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()
```



3.3 MLP + Dropout + AdamOptimizer

In [73]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                    kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(170, activation='relu',
```

```
                         kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_relu.add(Dense(136, activation='relu',
                         kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(80, activation='relu',
                         kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

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

In [74]:

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

history33 = model_drop.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 [==============================] - 5s 85us/step - loss: 2.1472 - acc: 0.3123 -
val_loss: 1.0609 - val_acc: 0.8265
Epoch 2/20
60000/60000 [==============================] - 3s 49us/step - loss: 1.6095 - acc: 0.4413 -
val_loss: 0.9769 - val_acc: 0.8420
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step - loss: 1.5983 - acc: 0.4471 -
val_loss: 0.9571 - val_acc: 0.8477
Epoch 4/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5921 - acc: 0.4479 -
val_loss: 0.9511 - val_acc: 0.8460
Epoch 5/20
60000/60000 [==============================] - 3s 58us/step - loss: 1.5854 - acc: 0.4501 -
val_loss: 0.9482 - val_acc: 0.8460
Epoch 6/20
60000/60000 [==============================] - 4s 59us/step - loss: 1.5794 - acc: 0.4549 -
val_loss: 0.9463 - val_acc: 0.8516
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 1.5878 - acc: 0.4474 -
val_loss: 0.9550 - val_acc: 0.8501
Epoch 8/20
60000/60000 [==============================] - 3s 58us/step - loss: 1.5732 - acc: 0.4545 -
val_loss: 0.9427 - val_acc: 0.8478
Epoch 9/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5755 - acc: 0.4537 -
val_loss: 0.9401 - val_acc: 0.8461
Epoch 10/20
60000/60000 [==============================] - 3s 54us/step - loss: 1.5732 - acc: 0.4532 -
val_loss: 0.9415 - val_acc: 0.8513
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 1.5702 - acc: 0.4559 -
val_loss: 0.9337 - val_acc: 0.8507
Epoch 12/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5701 - acc: 0.4573 -
val_loss: 0.9341 - val_acc: 0.8509
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5623 - acc: 0.4591 -
val_loss: 0.9330 - val_acc: 0.8524
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5663 - acc: 0.4601 -
val_loss: 0.9318 - val_acc: 0.8501
```

```
Epoch 15/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5658 - acc: 0.4596 - val_loss: 0.9342 - val_acc: 0.8515
Epoch 16/20
60000/60000 [==============================] - 3s 56us/step - loss: 1.5580 - acc: 0.4600 - val_loss: 0.9315 - val_acc: 0.8568
Epoch 17/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5593 - acc: 0.4587 - val_loss: 0.9311 - val_acc: 0.8504
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step - loss: 1.5611 - acc: 0.4602 - val_loss: 0.9332 - val_acc: 0.8496
Epoch 19/20
60000/60000 [==============================] - 3s 54us/step - loss: 1.5560 - acc: 0.4608 - val_loss: 0.9331 - val_acc: 0.8521
Epoch 20/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.5554 - acc: 0.4596 - val_loss: 0.9358 - val_acc: 0.8527
```

In [75]:

```python
model_drop.summary()
```

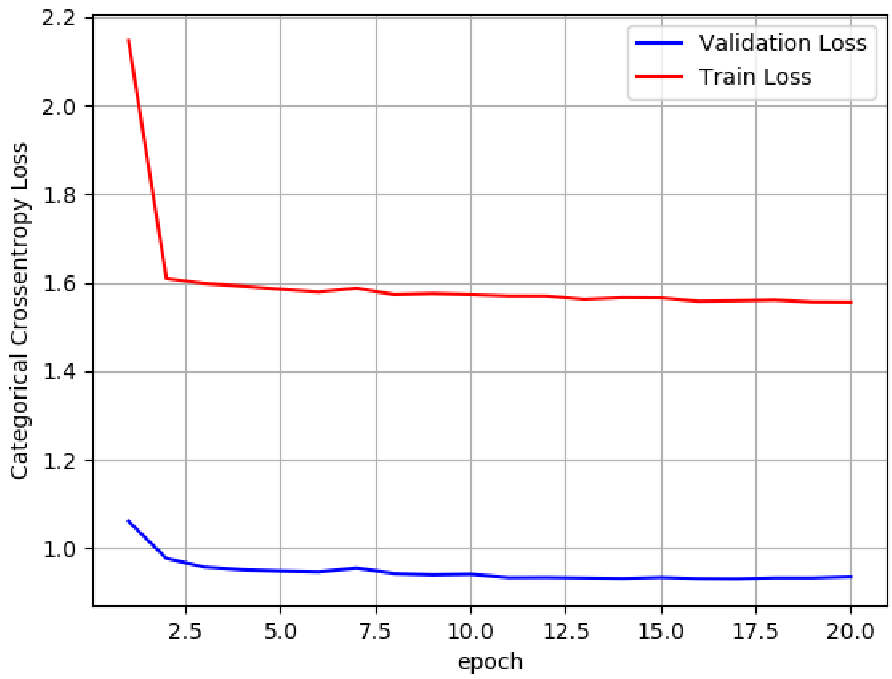
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_31 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_11 (Dropout)         (None, 784)               0         
_________________________________________________________________
batch_normalization_32 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_12 (Dropout)         (None, 784)               0         
_________________________________________________________________
batch_normalization_33 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_13 (Dropout)         (None, 784)               0         
_________________________________________________________________
batch_normalization_34 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_14 (Dropout)         (None, 784)               0         
_________________________________________________________________
batch_normalization_35 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_15 (Dropout)         (None, 784)               0         
_________________________________________________________________
dense_72 (Dense)             (None, 10)                7850      
=================================================================
Total params: 23,530
Trainable params: 15,690
Non-trainable params: 7,840
_________________________________________________________________
```

In [76]:

```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score17=score[0]
score18=score[1]
train_acc9=history33.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax33 = plt.subplots(1,1)
ax33.set_xlabel('epoch') ; ax33.set_ylabel('Categorical Crossentropy Loss')

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


vy33 = history33.history['val_loss']
ty33 = history33.history['loss']
plt_dynamic(x, vy33, ty33, ax33)
```

```
Test score: 0.9357627007484436
```

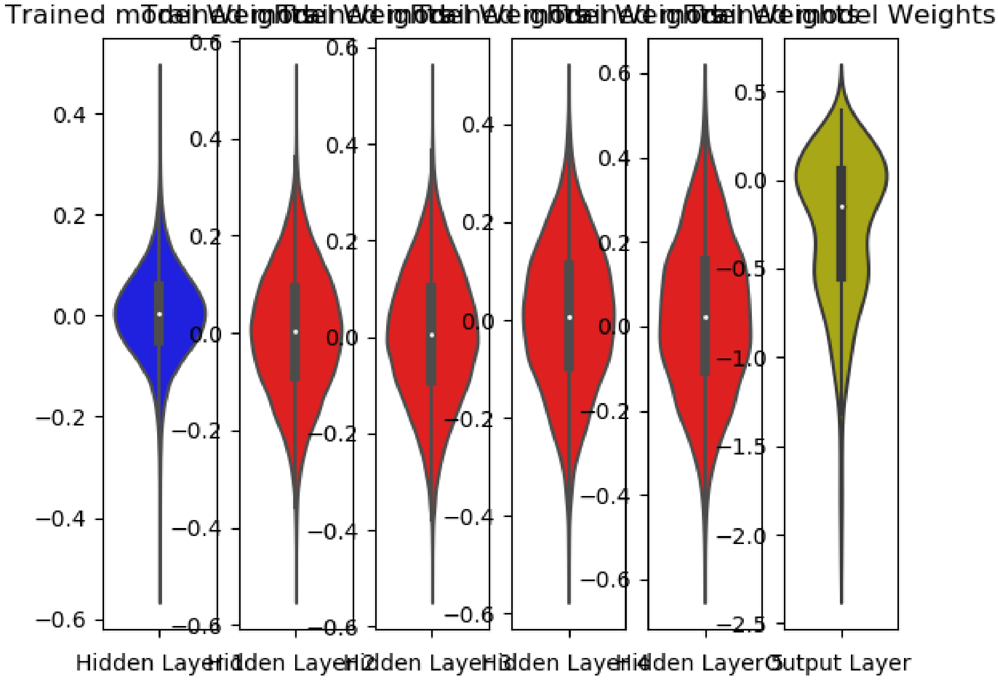
```
Test score: 0.9337027007484436
Test accuracy: 0.8527
```



In [77]:

```
w_after = model_relu.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=h5_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()
```



MLP + different drop out rates + rmsprop optimizer

In [78]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.3))

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.4))

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

In [79]:

```
model_drop.compile(optimizer='rmsprop',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history22 = model_drop.fit(X_train, Y_train,
```

```
history33 = model_drop.fit(X_train, Y_train,
                          batch_size=batch_size,

                          epochs=nb_epoch, verbose=1,
                          validation_data=(X_test, Y_test))
```

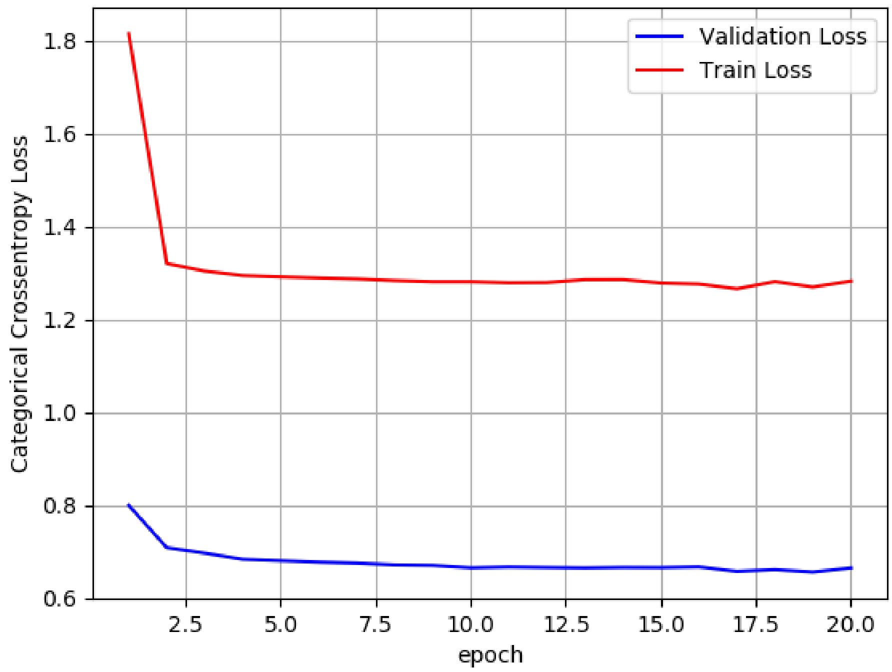
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 82us/step - loss: 1.8151 - acc: 0.4056 -
val_loss: 0.7996 - val_acc: 0.8484
Epoch 2/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.3204 - acc: 0.5537 -
val_loss: 0.7085 - val_acc: 0.8581
Epoch 3/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.3041 - acc: 0.5607 -
val_loss: 0.6967 - val_acc: 0.8635
Epoch 4/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.2945 - acc: 0.5616 -
val_loss: 0.6833 - val_acc: 0.8666
Epoch 5/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.2916 - acc: 0.5623 -
val_loss: 0.6804 - val_acc: 0.8624
Epoch 6/20
60000/60000 [==============================] - 3s 49us/step - loss: 1.2892 - acc: 0.5648 -
val_loss: 0.6773 - val_acc: 0.8664
Epoch 7/20
60000/60000 [==============================] - 3s 48us/step - loss: 1.2872 - acc: 0.5626 -
val_loss: 0.6751 - val_acc: 0.8645
Epoch 8/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.2836 - acc: 0.5665 -
val_loss: 0.6713 - val_acc: 0.8674
Epoch 9/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.2811 - acc: 0.5667 -
val_loss: 0.6704 - val_acc: 0.8646
Epoch 10/20
60000/60000 [==============================] - 3s 53us/step - loss: 1.2811 - acc: 0.5662 -
val_loss: 0.6653 - val_acc: 0.8684
Epoch 11/20
60000/60000 [==============================] - 4s 59us/step - loss: 1.2789 - acc: 0.5666 -
val_loss: 0.6669 - val_acc: 0.8662
Epoch 12/20
60000/60000 [==============================] - 3s 56us/step - loss: 1.2793 - acc: 0.5675 -
val_loss: 0.6659 - val_acc: 0.8765
Epoch 13/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.2857 - acc: 0.5632 -
val_loss: 0.6651 - val_acc: 0.8716
Epoch 14/20
60000/60000 [==============================] - 3s 46us/step - loss: 1.2858 - acc: 0.5653 -
val_loss: 0.6660 - val_acc: 0.8682
Epoch 15/20
60000/60000 [==============================] - 3s 54us/step - loss: 1.2783 - acc: 0.5683 -
val_loss: 0.6657 - val_acc: 0.8697
Epoch 16/20
60000/60000 [==============================] - 3s 51us/step - loss: 1.2762 - acc: 0.5671 -
val_loss: 0.6672 - val_acc: 0.8665
Epoch 17/20
60000/60000 [==============================] - 3s 47us/step - loss: 1.2660 - acc: 0.5719 -
val_loss: 0.6576 - val_acc: 0.8665
Epoch 18/20
60000/60000 [==============================] - 4s 64us/step - loss: 1.2814 - acc: 0.5652 -
val_loss: 0.6613 - val_acc: 0.8648
Epoch 19/20
60000/60000 [==============================] - 3s 50us/step - loss: 1.2698 - acc: 0.5697 -
val_loss: 0.6561 - val_acc: 0.8738
Epoch 20/20
60000/60000 [==============================] - 3s 48us/step - loss: 1.2824 - acc: 0.5673 -
val_loss: 0.6649 - val_acc: 0.8705
```

In [80]:

```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
score17=score[0]
score18=score[1]
train_acc9=history33.history['acc']
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

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

fig,ax33 = plt.subplots(1,1)
ax33.set_xlabel('epoch') ; ax33.set_ylabel('Categorical Crossentropy Loss')

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


vy33 = history33.history['val_loss']
ty33 = history33.history['loss']
plt_dynamic(x, vy33, ty33, ax33)
```

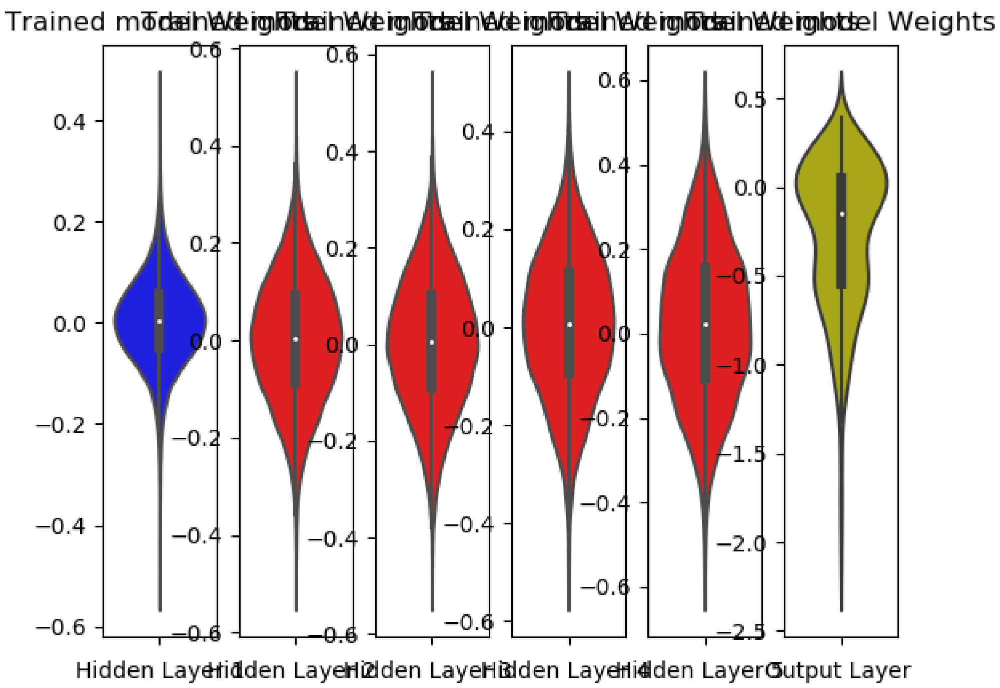
```
Test score: 0.6648854620933533
Test accuracy: 0.8705
```



In [81]:

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

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='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()
```



Summarizing all the models performance using Pretty Table

In [82]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Model", "Test-acccuracy"]

x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + ReLU activation function + ADAM opt
imizer",  0.979])
x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + ReLU activation function + RMSprop
optimizer", 0.9809])
x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + Batch-Norm on hidden Layers + Adam
Optimizer", 0.9797])
x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + Batch-Norm on hidden Layers + Adagr
ad Optimizer", 0.9843])
x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + Dropout + AdamOptimizer", 0.9836])
x.add_row(["2-Hidden layer Architecture (784-472-168-10):MLP + Dropout + Adadelta Optimizer", 0.98
4])
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP+ReLU+Adam", 0.9798])
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP + ReLU + Adadelta", 0.9
857])
```

```
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP + Batch-Norm on hidden 
Layers + AdamOptimizer", 0.7509])
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP + Batch-Norm on hidden 
Layers + RMS Prop Optimizer", 0.56])
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP + Dropout + 
AdamOptimizer", 0.8984])
x.add_row(["3-Hidden layer architecture (784-352-164-124 architecture):MLP + with Different dropou
t rates + RMS Prop Optimizer", 0.9068])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + ReLU + ADAM"
, 0.9778])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + ReLU + rmspr
op", 0.9792])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Batch-Norm o
n hidden Layers + AdamOptimizer", 0.975])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Batch-Norm o
n hidden Layers + Adadelta", 0.986])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Dropout + Ad
amOptimizer", 0.8527])
x.add_row(["5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + different dr
op out rates + rmsprop optimizer", 0.8705])

print(x)
```

```
+------------------------------------------------------------------------------------------------------
------------------+----------------+
|                                                          Model                                        
| Test-acccuracy |
+------------------------------------------------------------------------------------------------------
------------------+----------------+
|              2-Hidden layer Architecture (784-472-168-10):MLP + ReLU activation function + ADAM o
ptimizer              |     0.979      |
|            2-Hidden layer Architecture (784-472-168-10):MLP + ReLU activation function + RMSprop 
optimizer             |     0.9809     |
|            2-Hidden layer Architecture (784-472-168-10):MLP + Batch-Norm on hidden Layers + Adam 
Optimizer             |     0.9797     |
|           2-Hidden layer Architecture (784-472-168-10):MLP + Batch-Norm on hidden Layers + Adagra
d Optimizer           |     0.9843     |
|                        2-Hidden layer Architecture (784-472-168-10):MLP + Dropout + AdamOptimizer 
|     0.9836     |
|                     2-Hidden layer Architecture (784-472-168-10):MLP + Dropout + Adadelta 
Optimizer                     |     0.984      |
|                         3-Hidden layer architecture (784-352-164-124 architecture):MLP+ReLU+Adam 
|     0.9798     |
|                     3-Hidden layer architecture (784-352-164-124 architecture):MLP + ReLU + 
Adadelta                    |     0.9857     |
|      3-Hidden layer architecture (784-352-164-124 architecture):MLP + Batch-Norm on hidden 
Layers + AdamOptimizer      |     0.7509     |
|   3-Hidden layer architecture (784-352-164-124 architecture):MLP + Batch-Norm on hidden Layers + 
RMS Prop Optimizer    |      0.56      |
|                 3-Hidden layer architecture (784-352-164-124 architecture):MLP + Dropout + AdamOpt
imizer                |     0.8984     |
|   3-Hidden layer architecture (784-352-164-124 architecture):MLP + with Different dropout rates 
+ RMS Prop Optimizer   |     0.9068     |
|                  5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + ReLU + 
ADAM                  |     0.9778     |
|                 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + ReLU + r
msprop                |     0.9792     |
| 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Batch-Norm on hidden L
ayers + AdamOptimizer  |     0.975      |
|    5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Batch-Norm on 
hidden Layers + Adadelta    |     0.986      |
|           5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + Dropout + Ada
mOptimizer            |     0.8527     |
| 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture):MLP + different drop out rat
es + rmsprop optimizer |     0.8705     |
+------------------------------------------------------------------------------------------------------
------------------+----------------+
```

Conclusion

-- We find that the model with 3-Hidden layer architecture (784-352-164-124 architecture):MLP + ReLU + Adadelta has the higest test accuracy and it has outperformed compared to other models with different architectures.