Keras On MNIST

In [1]:

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

In [66]:

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
from prettytable import PrettyTable
```

In [8]:

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

In [9]:

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

In [10]:

```
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 [11]:

```
# 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 [12]:

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

```python
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 [13]:

```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 [14]:

```python
# 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 [15]:

```python
# 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.         0.         0.         0.
 0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
 0.58823529 0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.0627451   0.36470588  0.98823529  0.99215686  0.73333333
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.97647059  0.99215686  0.97647059  0.25098039  0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.18039216  0.50980392  0.71764706  0.99215686
 0.99215686  0.81176471  0.00784314  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.15294118  0.58039216
 0.89803922  0.99215686  0.99215686  0.99215686  0.98039216  0.71372549
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.09411765  0.44705882  0.86666667  0.99215686  0.99215686  0.99215686
 0.99215686  0.78823529  0.30588235  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.09019608  0.25882353  0.83529412  0.99215686
 0.99215686  0.99215686  0.99215686  0.77647059  0.31764706  0.00784314
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.07058824  0.67058824
 0.85882353  0.99215686  0.99215686  0.99215686  0.99215686  0.76470588
 0.31372549  0.03529412  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.21568627  0.6745098   0.88627451  0.99215686  0.99215686  0.99215686
 0.99215686  0.95686275  0.52156863  0.04313725  0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.53333333  0.99215686
 0.99215686  0.99215686  0.83137255  0.52941176  0.51764706  0.0627451
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

In [16]:

```
# here we are having a class number for each image
print("Class label of third image :", y_train[2])

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

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

print("After converting the output into a vector : ",Y_train[2])
```

```
Class label of third image : 4
After converting the output into a vector :  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

2 Layered Architecture

Relu + Adam

In [17]:

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

In [18]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 30
```

In [25]:

```
# start building a model
model_relu_2 = Sequential()
model_relu_2.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.074, seed=None)))
model_relu_2.add(Dense(212, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
97, seed=None)) )
model_relu_2.add(Dense(output_dim, activation='softmax'))

print(model_relu_2.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_11 (Dense)             (None, 212)               77380
_________________________________________________________________
dense_12 (Dense)             (None, 10)                2130
=================================================================
Total params: 365,250
Trainable params: 365,250
Non-trainable params: 0
_________________________________________________________________
None
```

In [26]:

```
model_relu_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

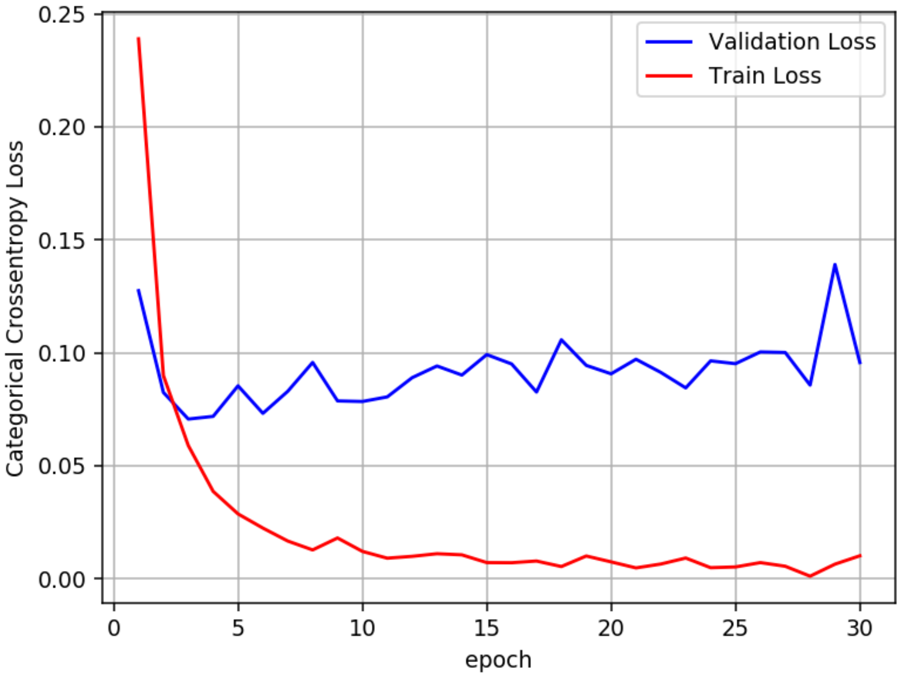
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 6s 102us/step - loss: 0.2389 - acc: 0.9287 -
val_loss: 0.1275 - val_acc: 0.9625
Epoch 2/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0899 - acc: 0.9726 -
val_loss: 0.0824 - val_acc: 0.9746
Epoch 3/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0589 - acc: 0.9822 -
val_loss: 0.0706 - val_acc: 0.9772
Epoch 4/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0386 - acc: 0.9880 -
val_loss: 0.0718 - val_acc: 0.9787
Epoch 5/30
60000/60000 [==============================] - 6s 96us/step - loss: 0.0286 - acc: 0.9908 -
val_loss: 0.0853 - val_acc: 0.9759
Epoch 6/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0224 - acc: 0.9929 -
```

```
val_loss: 0.0732 - val_acc: 0.9787
Epoch 7/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0166 - acc: 0.9950 -
val_loss: 0.0829 - val_acc: 0.9759
Epoch 8/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0127 - acc: 0.9960 -
val_loss: 0.0957 - val_acc: 0.9752
Epoch 9/30
60000/60000 [==============================] - 7s 110us/step - loss: 0.0180 - acc: 0.9939 -
val_loss: 0.0787 - val_acc: 0.9791
Epoch 10/30
60000/60000 [==============================] - 6s 98us/step - loss: 0.0121 - acc: 0.9959 -
val_loss: 0.0784 - val_acc: 0.9804
Epoch 11/30
60000/60000 [==============================] - 6s 97us/step - loss: 0.0090 - acc: 0.9970 -
val_loss: 0.0804 - val_acc: 0.9799
Epoch 12/30
60000/60000 [==============================] - 6s 99us/step - loss: 0.0099 - acc: 0.9968 -
val_loss: 0.0889 - val_acc: 0.9788
Epoch 13/30
60000/60000 [==============================] - 6s 97us/step - loss: 0.0110 - acc: 0.9960 -
val_loss: 0.0941 - val_acc: 0.9776
Epoch 14/30
60000/60000 [==============================] - 6s 105us/step - loss: 0.0105 - acc: 0.9964 -
val_loss: 0.0901 - val_acc: 0.9785
Epoch 15/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0071 - acc: 0.9978 -
val_loss: 0.0992 - val_acc: 0.9787
Epoch 16/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0070 - acc: 0.9979 -
val_loss: 0.0950 - val_acc: 0.9774
Epoch 17/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0078 - acc: 0.9975 -
val_loss: 0.0826 - val_acc: 0.9829
Epoch 18/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0053 - acc: 0.9982 -
val_loss: 0.1057 - val_acc: 0.9796
Epoch 19/30
60000/60000 [==============================] - 6s 102us/step - loss: 0.0100 - acc: 0.9967 -
val_loss: 0.0944 - val_acc: 0.9796
Epoch 20/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0074 - acc: 0.9974 -
val_loss: 0.0907 - val_acc: 0.9818
Epoch 21/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0047 - acc: 0.9984 -
val_loss: 0.0971 - val_acc: 0.9818
Epoch 22/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0065 - acc: 0.9977 -
val_loss: 0.0913 - val_acc: 0.9824
Epoch 23/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0091 - acc: 0.9972 -
val_loss: 0.0844 - val_acc: 0.9822
Epoch 24/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0048 - acc: 0.9985 -
val_loss: 0.0964 - val_acc: 0.9818
Epoch 25/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0052 - acc: 0.9983 -
val_loss: 0.0951 - val_acc: 0.9819
Epoch 26/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0071 - acc: 0.9979 -
val_loss: 0.1003 - val_acc: 0.9788
Epoch 27/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0055 - acc: 0.9982 -
val_loss: 0.1001 - val_acc: 0.9821
Epoch 28/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0011 - acc: 0.9996 -
val_loss: 0.0857 - val_acc: 0.9832
Epoch 29/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0064 - acc: 0.9980 -
val_loss: 0.1391 - val_acc: 0.9754
Epoch 30/30
60000/60000 [==============================] - 6s 105us/step - loss: 0.0101 - acc: 0.9970 -
val_loss: 0.0956 - val_acc: 0.9817
```

In [28]:

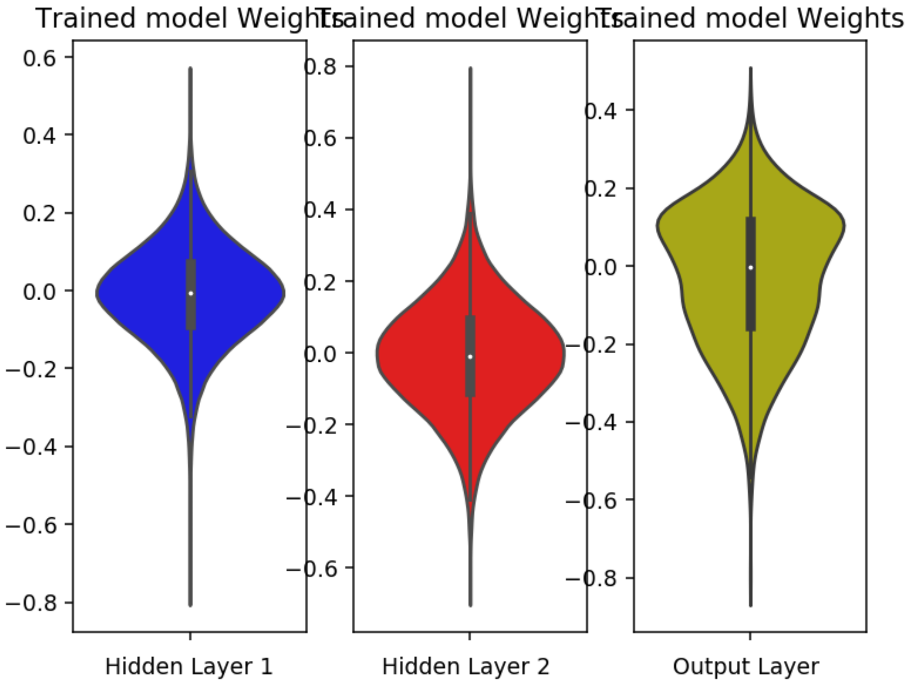
```
score_relu_2 = model_relu_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_2[0])
print('Test accuracy:', score_relu_2[1])

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

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

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

```
Test score: 0.09564156731741441
Test accuracy: 0.9817
```



In [29]:

```
w_after = model_relu_2.get_weights()

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


fig = plt.figure()
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()
```



Batch normalization on hidden layers + Relu + Adam

In [30]:

```
model_batch_2 = Sequential()

model_batch_2.add(Dense(532, activation='relu', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.061, seed=None)))
model_batch_2.add(BatchNormalization())

model_batch_2.add(Dense(384, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
072, seed=None)) )
model_batch_2.add(BatchNormalization())

model_batch_2.add(Dense(output_dim, activation='softmax'))


model_batch_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 532)               417620    
_________________________________________________________________
batch_normalization_7 (Batch (None, 532)               2128      
_________________________________________________________________
dense_14 (Dense)             (None, 384)               204672    
_________________________________________________________________
batch_normalization_8 (Batch (None, 384)               1536      
_________________________________________________________________
dense_15 (Dense)             (None, 10)                3850      
=================================================================
Total params: 629,806
Trainable params: 627,974
Non-trainable params: 1,832
_________________________________________________________________
```

In [31]:

```
model_batch_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))
```

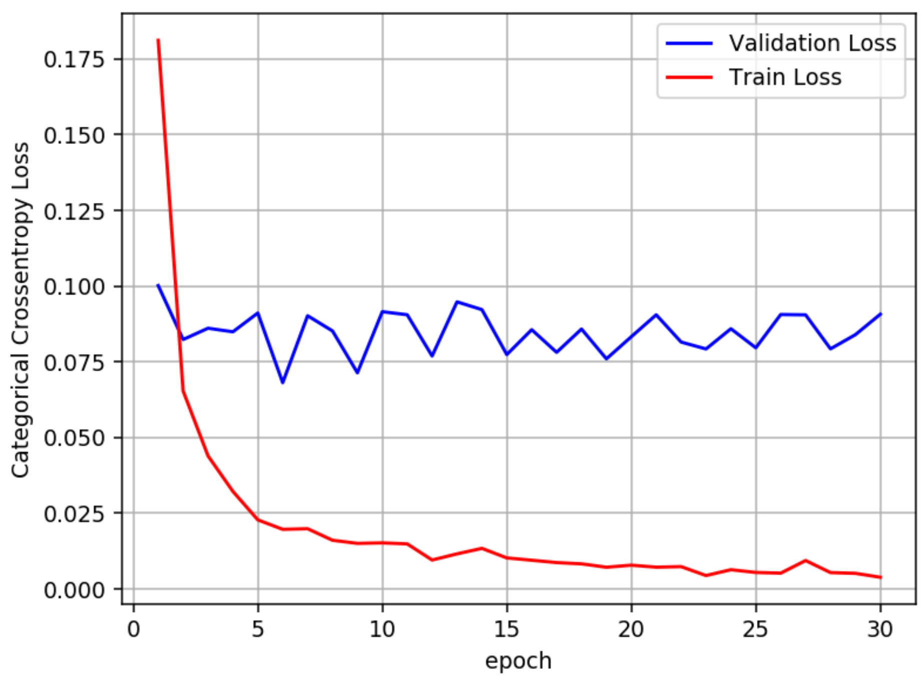
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 11s 189us/step - loss: 0.1812 - acc: 0.9441 - val_l
oss: 0.1002 - val_acc: 0.9688oss
Epoch 2/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0654 - acc: 0.9801 - val_l
oss: 0.0824 - val_acc: 0.9750
Epoch 3/30
60000/60000 [==============================] - 10s 159us/step - loss: 0.0439 - acc: 0.9862 - val_l
oss: 0.0861 - val_acc: 0.9724
Epoch 4/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0322 - acc: 0.9898 - val_l
oss: 0.0849 - val_acc: 0.9738
Epoch 5/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0228 - acc: 0.9928 - val_l
oss: 0.0911 - val_acc: 0.9728
Epoch 6/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.0197 - acc: 0.9933 - val_l
oss: 0.0681 - val_acc: 0.9807
Epoch 7/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0199 - acc: 0.9936 - val_l
oss: 0.0902 - val_acc: 0.9789
Epoch 8/30
60000/60000 [==============================] - 10s 161us/step - loss: 0.0161 - acc: 0.9949 - val_l
oss: 0.0852 - val_acc: 0.9764
Epoch 9/30
60000/60000 [==============================] - 10s 161us/step - loss: 0.0150 - acc: 0.9950 - val_l
oss: 0.0714 - val_acc: 0.9814
Epoch 10/30
60000/60000 [==============================] - 10s 163us/step - loss: 0.0152 - acc: 0.9948 - val_l
oss: 0.0916 - val_acc: 0.9765
Epoch 11/30
60000/60000 [==============================] - 10s 173us/step - loss: 0.0149 - acc: 0.9947 - val_l
oss: 0.0905 - val_acc: 0.9782
Epoch 12/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0095 - acc: 0.9969 - val_l
oss: 0.0769 - val_acc: 0.9804
Epoch 13/30
60000/60000 [==============================] - 10s 167us/step - loss: 0.0116 - acc: 0.9959 - val_l
oss: 0.0948 - val_acc: 0.9754
Epoch 14/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0134 - acc: 0.9956 - val_l
oss: 0.0922 - val_acc: 0.9777
Epoch 15/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.0102 - acc: 0.9962 - val_l
oss: 0.0774 - val_acc: 0.9818
Epoch 16/30
60000/60000 [==============================] - 10s 166us/step - loss: 0.0095 - acc: 0.9970 - val_l
oss: 0.0856 - val_acc: 0.9800
Epoch 17/30
60000/60000 [==============================] - 10s 166us/step - loss: 0.0087 - acc: 0.9970 - val_l
oss: 0.0781 - val_acc: 0.9815
Epoch 18/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.0083 - acc: 0.9973 - val_l
oss: 0.0858 - val_acc: 0.9808
Epoch 19/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0072 - acc: 0.9976 - val_l
oss: 0.0760 - val_acc: 0.9822
Epoch 20/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0078 - acc: 0.9973 - val_l
oss: 0.0833 - val_acc: 0.9823
Epoch 21/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.0072 - acc: 0.9978 - val_l
oss: 0.0905 - val_acc: 0.9803
Epoch 22/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0074 - acc: 0.9975 - val_l
oss: 0.0815 - val_acc: 0.9813
Epoch 23/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.0044 - acc: 0.9985 - val_l
oss: 0.0792 - val_acc: 0.9809
Epoch 24/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0063 - acc: 0.9979 - val_l
oss: 0.0859 - val_acc: 0.9808
Epoch 25/30
60000/60000 [==============================] - 10s 167us/step - loss: 0.0054 - acc: 0.9984 - val_l
oss: 0.0796 - val_acc: 0.9820
```

```
Epoch 26/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.0052 - acc: 0.9985 - val_l
oss: 0.0906 - val_acc: 0.9824
Epoch 27/30
60000/60000 [==============================] - 10s 168us/step - loss: 0.0094 - acc: 0.9969 - val_l
oss: 0.0905 - val_acc: 0.9814
Epoch 28/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0054 - acc: 0.9983 - val_l
oss: 0.0793 - val_acc: 0.9825
Epoch 29/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0051 - acc: 0.9984 - val_l
oss: 0.0840 - val_acc: 0.9813
Epoch 30/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.0038 - acc: 0.9987 - val_l
oss: 0.0907 - val_acc: 0.9804
```

In [32]:

```python
score_batch_2 = model_batch_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_batch_2[0])
print('Test accuracy:', score_batch_2[1])

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

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

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

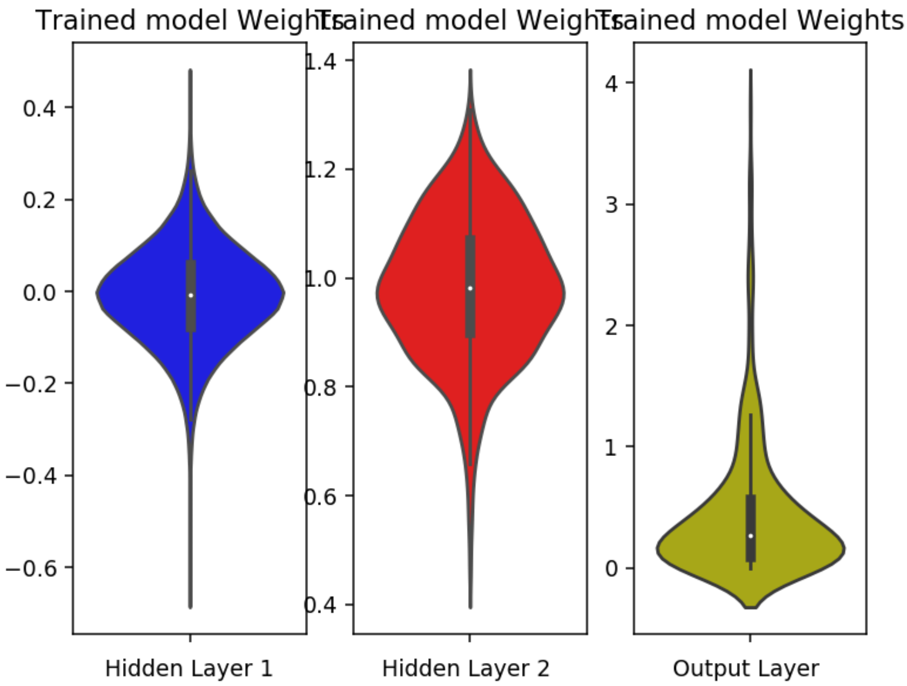
```
Test score: 0.09071999986333348
Test accuracy: 0.9804
```



In [33]:

```python
w_after = model_batch_2.get_weights()

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

```
fig = plt.figure()
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()
```



Dropout + Relu + Adam

In [34]:

```
model_drop_2 = Sequential()

model_drop_2.add(Dense(532, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.061, seed=None)))
model_drop_2.add(BatchNormalization())
model_drop_2.add(Dropout(0.5))

model_drop_2.add(Dense(384, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
72, seed=None)) )
model_drop_2.add(BatchNormalization())
model_drop_2.add(Dropout(0.5))

model_drop_2.add(Dense(output_dim, activation='softmax'))


model_drop_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_16 (Dense)             (None, 532)               417620    
```

```
_________________________________________________________________
batch_normalization_9 (Batch (None, 532)              2128
_________________________________________________________________
dropout_7 (Dropout)          (None, 532)              0
_________________________________________________________________
dense_17 (Dense)             (None, 384)              204672
_________________________________________________________________
batch_normalization_10 (Batc (None, 384)              1536
_________________________________________________________________
dropout_8 (Dropout)          (None, 384)              0
_________________________________________________________________
dense_18 (Dense)             (None, 10)               3850
=================================================================
Total params: 629,806
Trainable params: 627,974
Non-trainable params: 1,832
_________________________________________________________________
```

In [35]:

```
model_drop_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

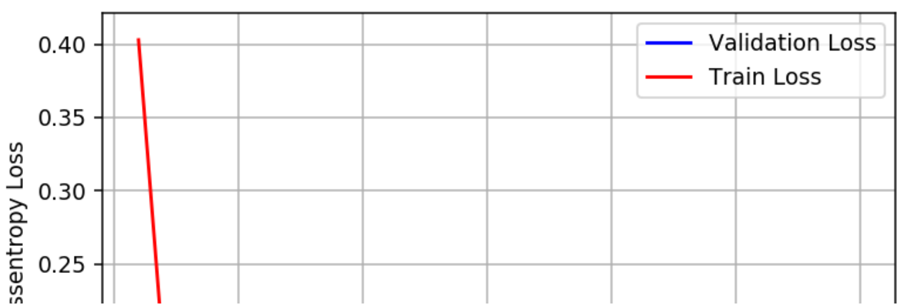
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 13s 215us/step - loss: 0.4029 - acc: 0.8781 - val_l
oss: 0.1331 - val_acc: 0.9583
Epoch 2/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.1890 - acc: 0.9423 - val_l
oss: 0.0954 - val_acc: 0.9706
Epoch 3/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.1462 - acc: 0.9547 - val_l
oss: 0.0829 - val_acc: 0.9731
Epoch 4/30
60000/60000 [==============================] - 12s 194us/step - loss: 0.1256 - acc: 0.9605 - val_l
oss: 0.0772 - val_acc: 0.9757
Epoch 5/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.1101 - acc: 0.9656 - val_l
oss: 0.0702 - val_acc: 0.9787
Epoch 6/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0987 - acc: 0.9698 - val_l
oss: 0.0701 - val_acc: 0.9791
Epoch 7/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0910 - acc: 0.9712 - val_l
oss: 0.0678 - val_acc: 0.9793
Epoch 8/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0872 - acc: 0.9717 - val_l
oss: 0.0648 - val_acc: 0.9789
Epoch 9/30
60000/60000 [==============================] - 12s 194us/step - loss: 0.0812 - acc: 0.9742 - val_l
oss: 0.0609 - val_acc: 0.9811
Epoch 10/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.0758 - acc: 0.9758 - val_l
oss: 0.0581 - val_acc: 0.9814
Epoch 11/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.0673 - acc: 0.9784 - val_l
oss: 0.0573 - val_acc: 0.9826
Epoch 12/30
60000/60000 [==============================] - 11s 190us/step - loss: 0.0668 - acc: 0.9778 - val_l
oss: 0.0605 - val_acc: 0.9826
Epoch 13/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0647 - acc: 0.9797 - val_l
oss: 0.0559 - val_acc: 0.9827
Epoch 14/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.0619 - acc: 0.9799 - val_l
oss: 0.0556 - val_acc: 0.9830
Epoch 15/30
60000/60000 [==============================] - 12s 197us/step - loss: 0.0566 - acc: 0.9816 - val_l
oss: 0.0577 - val_acc: 0.9825
Epoch 16/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0558 - acc: 0.9820 - val_l
oss: 0.0589 - val_acc: 0.9823
Epoch 17/30
60000/60000 [==============================] - 12s 196us/step - loss: 0.0536 - acc: 0.9823 - val_l
```

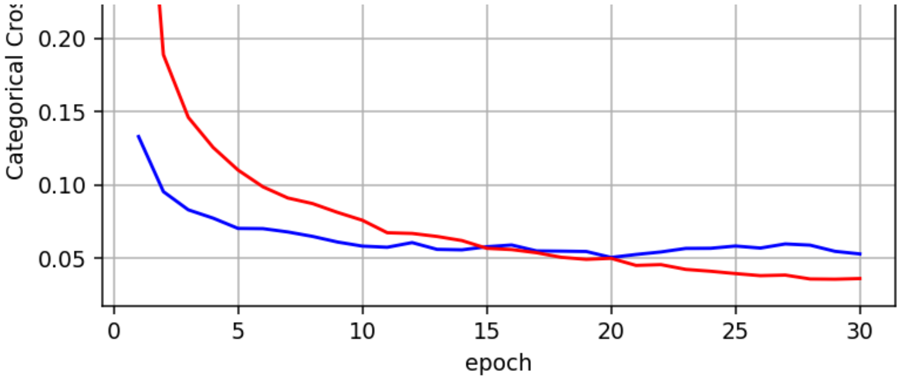
```
oss: 0.0548 - val_acc: 0.9843
Epoch 18/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0505 - acc: 0.9835 - val_l
oss: 0.0547 - val_acc: 0.9843
Epoch 19/30
60000/60000 [==============================] - 12s 192us/step - loss: 0.0491 - acc: 0.9842 - val_l
oss: 0.0545 - val_acc: 0.9832
Epoch 20/30
60000/60000 [==============================] - 11s 188us/step - loss: 0.0498 - acc: 0.9841 - val_l
oss: 0.0504 - val_acc: 0.9853loss: 0
Epoch 21/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0449 - acc: 0.9849 - val_l
oss: 0.0524 - val_acc: 0.9856
Epoch 22/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.0455 - acc: 0.9848 - val_l
oss: 0.0541 - val_acc: 0.9847
Epoch 23/30
60000/60000 [==============================] - 11s 186us/step - loss: 0.0422 - acc: 0.9859 - val_l
oss: 0.0566 - val_acc: 0.9833
Epoch 24/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0409 - acc: 0.9862 - val_l
oss: 0.0566 - val_acc: 0.9826
Epoch 25/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0394 - acc: 0.9868 - val_l
oss: 0.0582 - val_acc: 0.9836
Epoch 26/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.0379 - acc: 0.9876 - val_l
oss: 0.0568 - val_acc: 0.9842
Epoch 27/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.0383 - acc: 0.9871 - val_l
oss: 0.0596 - val_acc: 0.9836
Epoch 28/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.0357 - acc: 0.9880 - val_l
oss: 0.0588 - val_acc: 0.9829
Epoch 29/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.0355 - acc: 0.9879 - val_l
oss: 0.0546 - val_acc: 0.9841
Epoch 30/30
60000/60000 [==============================] - 10s 175us/step - loss: 0.0360 - acc: 0.9883 - val_l
oss: 0.0528 - val_acc: 0.9848
```

In [38]:

```python
score_drop_2 = model_drop_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_drop_2[0])
print('Test accuracy:', score_drop_2[1])

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

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

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

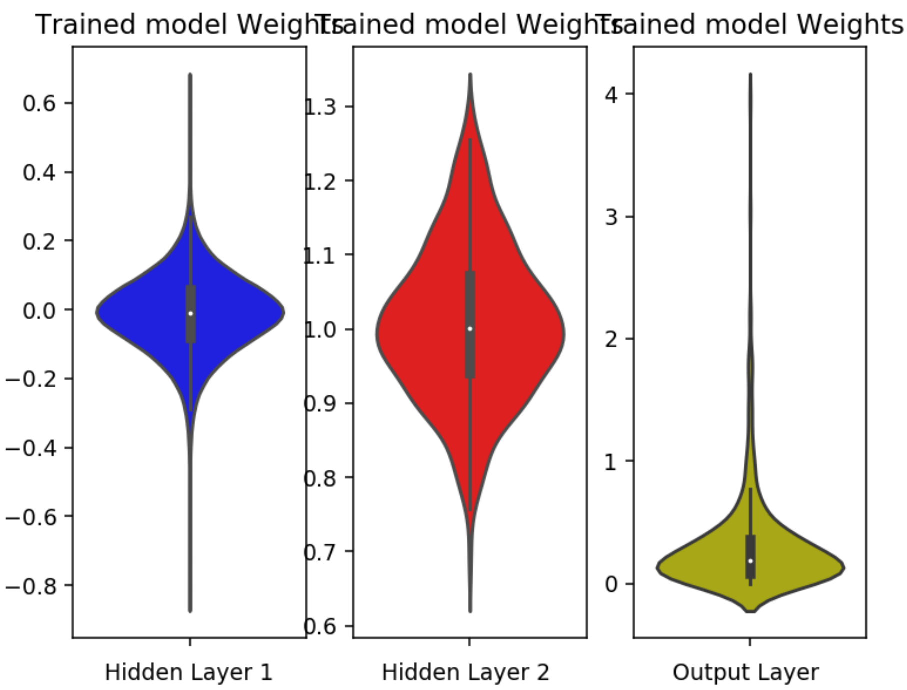
```
Test score: 0.052775146638258594
Test accuracy: 0.9848
```





In [39]:

```
w_after = model_drop_2.get_weights()

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


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

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

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



3 Layered Architecture

Relu + Adam

In [40]:

```
# start building a model
model_relu_3 = Sequential()
model_relu_3.add(Dense(621, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.056, seed=None)))
model_relu_3.add(Dense(333, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
77, seed=None)) )
model_relu_3.add(Dense(161, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
11, seed=None)) )
model_relu_3.add(Dense(output_dim, activation='softmax'))

print(model_relu_3.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 621)               487485
_________________________________________________________________
dense_20 (Dense)             (None, 333)               207126
_________________________________________________________________
dense_21 (Dense)             (None, 161)               53774
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1620
=================================================================
Total params: 750,005
Trainable params: 750,005
Non-trainable params: 0
_________________________________________________________________
None
```

In [41]:

```
model_relu_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

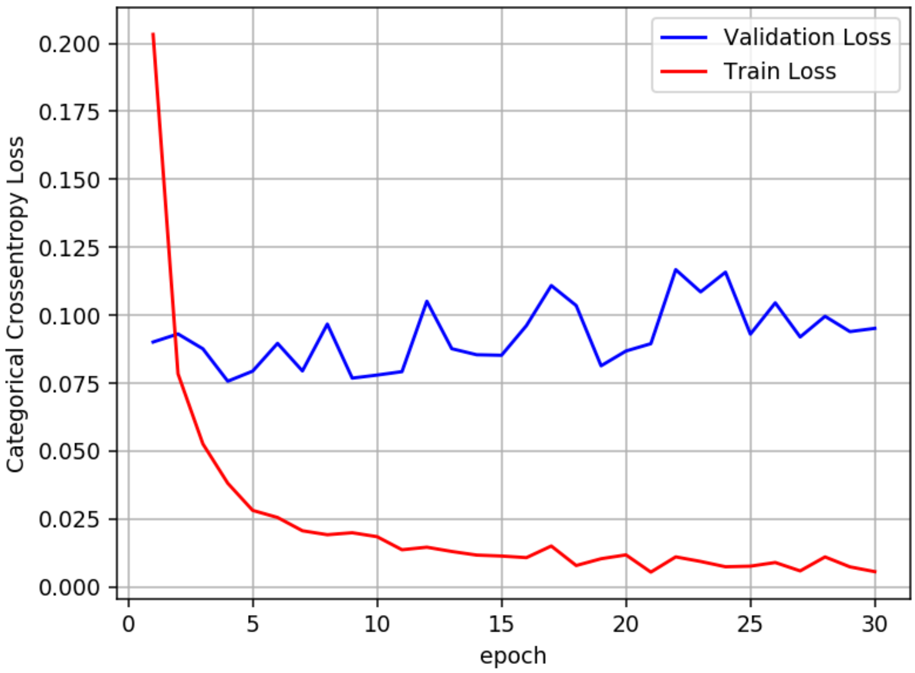
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 11s 188us/step - loss: 0.2033 - acc: 0.9383 - val_l
oss: 0.0901 - val_acc: 0.9723
Epoch 2/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0784 - acc: 0.9760 - val_l
oss: 0.0931 - val_acc: 0.9683
Epoch 3/30
60000/60000 [==============================] - 10s 160us/step - loss: 0.0526 - acc: 0.9834 - val_l
oss: 0.0876 - val_acc: 0.9718
Epoch 4/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0381 - acc: 0.9877 - val_l
oss: 0.0757 - val_acc: 0.9782
Epoch 5/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0282 - acc: 0.9908 - val_l
oss: 0.0794 - val_acc: 0.9781
Epoch 6/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0255 - acc: 0.9922 - val_l
oss: 0.0896 - val_acc: 0.9774
Epoch 7/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0207 - acc: 0.9928 - val_l
oss: 0.0795 - val_acc: 0.9794
Epoch 8/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.0192 - acc: 0.9937 - val_l
oss: 0.0967 - val_acc: 0.9755
Epoch 9/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0199 - acc: 0.9936 - val_l
oss: 0.0768 - val_acc: 0.9807
Epoch 10/30
60000/60000 [==============================] - 10s 163us/step - loss: 0.0185 - acc: 0.9939 - val_l
oss: 0.0780 - val_acc: 0.9798
Epoch 11/30
```

```
60000/60000 [==============================] - 10s 164us/step - loss: 0.0137 - acc: 0.9953 - val_l
oss: 0.0792 - val_acc: 0.9804
Epoch 12/30
60000/60000 [==============================] - 10s 163us/step - loss: 0.0146 - acc: 0.9957 - val_l
oss: 0.1051 - val_acc: 0.9760
Epoch 13/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0130 - acc: 0.9957 - val_l
oss: 0.0877 - val_acc: 0.9805
Epoch 14/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.0117 - acc: 0.9963 - val_l
oss: 0.0854 - val_acc: 0.9808
Epoch 15/30
60000/60000 [==============================] - 10s 163us/step - loss: 0.0114 - acc: 0.9965 - val_l
oss: 0.0852 - val_acc: 0.9824
Epoch 16/30
60000/60000 [==============================] - 10s 172us/step - loss: 0.0108 - acc: 0.9967 - val_l
oss: 0.0961 - val_acc: 0.9800
Epoch 17/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.0150 - acc: 0.9955 - val_l
oss: 0.1109 - val_acc: 0.9759
Epoch 18/30
60000/60000 [==============================] - 10s 168us/step - loss: 0.0079 - acc: 0.9975 - val_l
oss: 0.1035 - val_acc: 0.9774
Epoch 19/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0104 - acc: 0.9967 - val_l
oss: 0.0814 - val_acc: 0.9827
Epoch 20/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.0118 - acc: 0.9961 - val_l
oss: 0.0868 - val_acc: 0.9811
Epoch 21/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0054 - acc: 0.9983 - val_l
oss: 0.0895 - val_acc: 0.9827
Epoch 22/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0110 - acc: 0.9968 - val_l
oss: 0.1167 - val_acc: 0.9769
Epoch 23/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0094 - acc: 0.9971 - val_l
oss: 0.1086 - val_acc: 0.9783
Epoch 24/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0074 - acc: 0.9977 - val_l
oss: 0.1158 - val_acc: 0.9793
Epoch 25/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0076 - acc: 0.9979 - val_l
oss: 0.0930 - val_acc: 0.9823
Epoch 26/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.0090 - acc: 0.9976 - val_l
oss: 0.1045 - val_acc: 0.9810
Epoch 27/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0059 - acc: 0.9982 - val_l
oss: 0.0920 - val_acc: 0.9821
Epoch 28/30
60000/60000 [==============================] - 10s 162us/step - loss: 0.0110 - acc: 0.9971 - val_l
oss: 0.0996 - val_acc: 0.9795
Epoch 29/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.0074 - acc: 0.9981 - val_l
oss: 0.0939 - val_acc: 0.9814
Epoch 30/30
60000/60000 [==============================] - 10s 164us/step - loss: 0.0056 - acc: 0.9984 - val_l
oss: 0.0951 - val_acc: 0.9816
```

In [42]:

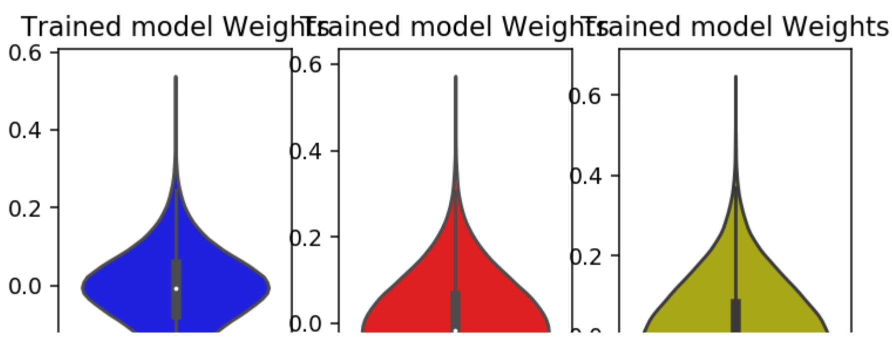
```
score_relu_3 = model_relu_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_3[0])
print('Test accuracy:', score_relu_3[1])

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

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

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

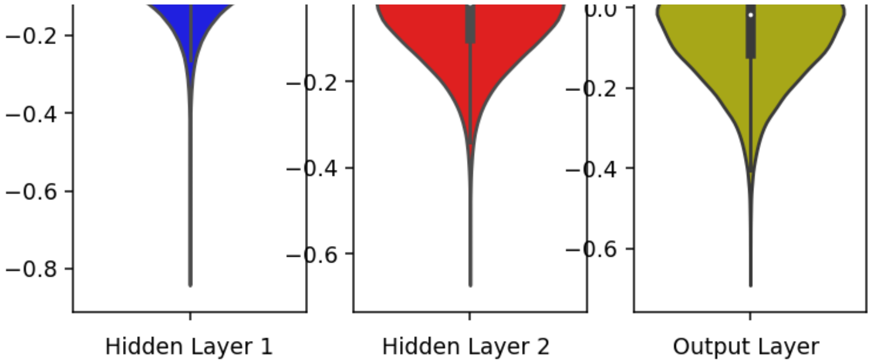
```
Test score: 0.09514001322890968
Test accuracy: 0.9816
```



In [43]:

```
w_after = model_relu_3.get_weights()

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


fig = plt.figure()
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()
```





Batch normalization on hidden layer + Relu + Adam

In [45]:

```
model_batch_3 = Sequential()

model_batch_3.add(Dense(589, activation='relu', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.058, seed=None)))
model_batch_3.add(BatchNormalization())

model_batch_3.add(Dense(423, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
068, seed=None)) )
model_batch_3.add(BatchNormalization())

model_batch_3.add(Dense(272, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
085, seed=None)) )
model_batch_3.add(BatchNormalization())

model_batch_3.add(Dense(output_dim, activation='softmax'))


model_batch_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_27 (Dense)             (None, 589)               462365    
_________________________________________________________________
batch_normalization_14 (Batc (None, 589)               2356      
_________________________________________________________________
dense_28 (Dense)             (None, 423)               249570    
_________________________________________________________________
batch_normalization_15 (Batc (None, 423)               1692      
_________________________________________________________________
dense_29 (Dense)             (None, 272)               115328    
_________________________________________________________________
batch_normalization_16 (Batc (None, 272)               1088      
_________________________________________________________________
dense_30 (Dense)             (None, 10)                2730      
=================================================================
Total params: 835,129
Trainable params: 832,561
Non-trainable params: 2,568
_________________________________________________________________
```

In [46]:

```
model_batch_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 16s 259us/step - loss: 0.1742 - acc: 0.9471 - val_l
oss: 0.0962 - val_acc: 0.9699
Epoch 2/30
```

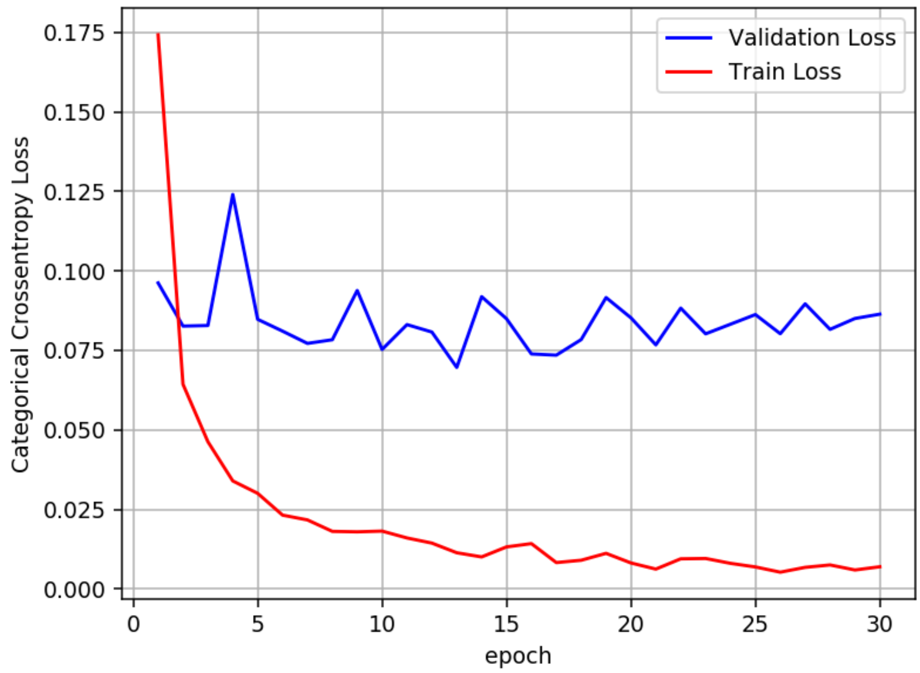
```
60000/60000 [==============================] - 14s 232us/step - loss: 0.0643 - acc: 0.9801 - val_l
oss: 0.0826 - val_acc: 0.9744
Epoch 3/30
60000/60000 [==============================] - 13s 209us/step - loss: 0.0463 - acc: 0.9851 - val_l
oss: 0.0828 - val_acc: 0.9734
Epoch 4/30
60000/60000 [==============================] - 13s 212us/step - loss: 0.0340 - acc: 0.9891 - val_l
oss: 0.1240 - val_acc: 0.9660
Epoch 5/30
60000/60000 [==============================] - 12s 207us/step - loss: 0.0301 - acc: 0.9894 - val_l
oss: 0.0848 - val_acc: 0.9748
Epoch 6/30
60000/60000 [==============================] - 12s 207us/step - loss: 0.0232 - acc: 0.9922 - val_l
oss: 0.0811 - val_acc: 0.9769
Epoch 7/30
60000/60000 [==============================] - 14s 229us/step - loss: 0.0217 - acc: 0.9924 - val_l
oss: 0.0772 - val_acc: 0.9780
Epoch 8/30
60000/60000 [==============================] - 13s 219us/step - loss: 0.0181 - acc: 0.9937 - val_l
oss: 0.0784 - val_acc: 0.9796
Epoch 9/30
60000/60000 [==============================] - 13s 219us/step - loss: 0.0180 - acc: 0.9936 - val_l
oss: 0.0938 - val_acc: 0.9760
Epoch 10/30
60000/60000 [==============================] - 13s 209us/step - loss: 0.0182 - acc: 0.9943 - val_l
oss: 0.0753 - val_acc: 0.9783
Epoch 11/30
60000/60000 [==============================] - 13s 222us/step - loss: 0.0160 - acc: 0.9946 - val_l
oss: 0.0831 - val_acc: 0.9792
Epoch 12/30
60000/60000 [==============================] - 13s 218us/step - loss: 0.0144 - acc: 0.9950 - val_l
oss: 0.0808 - val_acc: 0.9794
Epoch 13/30
60000/60000 [==============================] - 13s 218us/step - loss: 0.0114 - acc: 0.9965 - val_l
oss: 0.0697 - val_acc: 0.9823
Epoch 14/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0101 - acc: 0.9964 - val_l
oss: 0.0919 - val_acc: 0.9772
Epoch 15/30
60000/60000 [==============================] - 13s 218us/step - loss: 0.0132 - acc: 0.9956 - val_l
oss: 0.0850 - val_acc: 0.9783
Epoch 16/30
60000/60000 [==============================] - 14s 232us/step - loss: 0.0142 - acc: 0.9954 - val_l
oss: 0.0739 - val_acc: 0.9828
Epoch 17/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0083 - acc: 0.9970 - val_l
oss: 0.0735 - val_acc: 0.9814
Epoch 18/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0090 - acc: 0.9972 - val_l
oss: 0.0784 - val_acc: 0.9825
Epoch 19/30
60000/60000 [==============================] - 13s 224us/step - loss: 0.0112 - acc: 0.9963 - val_l
oss: 0.0916 - val_acc: 0.9792
Epoch 20/30
60000/60000 [==============================] - 14s 233us/step - loss: 0.0082 - acc: 0.9972 - val_l
oss: 0.0852 - val_acc: 0.9801
Epoch 21/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0062 - acc: 0.9979 - val_l
oss: 0.0767 - val_acc: 0.9827
Epoch 22/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0095 - acc: 0.9969 - val_l
oss: 0.0883 - val_acc: 0.9801
Epoch 23/30
60000/60000 [==============================] - 13s 222us/step - loss: 0.0096 - acc: 0.9970 - val_l
oss: 0.0802 - val_acc: 0.9805
Epoch 24/30
60000/60000 [==============================] - 14s 226us/step - loss: 0.0080 - acc: 0.9975 - val_l
oss: 0.0832 - val_acc: 0.9815
Epoch 25/30
60000/60000 [==============================] - 13s 223us/step - loss: 0.0069 - acc: 0.9978 - val_l
oss: 0.0863 - val_acc: 0.9811
Epoch 26/30
60000/60000 [==============================] - 13s 215us/step - loss: 0.0053 - acc: 0.9982 - val_l
oss: 0.0802 - val_acc: 0.9815
Epoch 27/30
60000/60000 [==============================] - 13s 225us/step - loss: 0.0068 - acc: 0.9978 - val_l
oss: 0.0896 - val_acc: 0.9805
```

```
Epoch 28/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0075 - acc: 0.9975 - val_l
oss: 0.0816 - val_acc: 0.9811
Epoch 29/30
60000/60000 [==============================] - 14s 236us/step - loss: 0.0060 - acc: 0.9978 - val_l
oss: 0.0850 - val_acc: 0.9834
Epoch 30/30
60000/60000 [==============================] - 13s 220us/step - loss: 0.0070 - acc: 0.9976 - val_l
oss: 0.0864 - val_acc: 0.9817
```

In [47]:

```python
score_batch_3 = model_batch_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_batch_3[0])
print('Test accuracy:', score_batch_3[1])

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

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

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

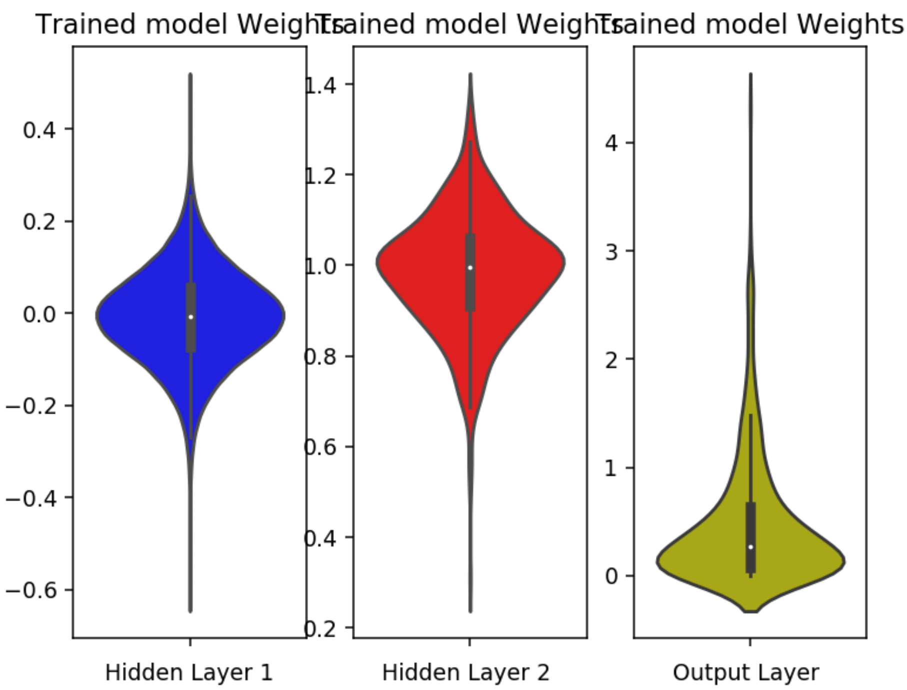
```
Test score: 0.08639855843480118
Test accuracy: 0.9817
```



In [48]:

```python
w_after = model_batch_3.get_weights()

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


fig = plt.figure()
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()
```



Drop + Relu + Adam

In [49]:

```
model_drop_3 = Sequential()

model_drop_3.add(Dense(401, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.070, seed=None)))
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.5))

model_drop_3.add(Dense(219, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
95, seed=None)) )
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.5))

model_drop_3.add(Dense(121, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
28, seed=None)) )
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.5))

model_drop_3.add(Dense(output_dim, activation='softmax'))


model_drop_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_31 (Dense)             (None, 401)               314785    
_________________________________________________________________
batch_normalization_17 (Batc (None, 401)               1604      
```

```
batch_normalization_17 (Batc (None, 401)               1604
_________________________________________________________________
dropout_9 (Dropout)          (None, 401)               0
_________________________________________________________________
dense_32 (Dense)             (None, 219)               88038
_________________________________________________________________
batch_normalization_18 (Batc (None, 219)               876
_________________________________________________________________
dropout_10 (Dropout)         (None, 219)               0
_________________________________________________________________
dense_33 (Dense)             (None, 121)               26620
_________________________________________________________________
batch_normalization_19 (Batc (None, 121)               484
_________________________________________________________________
dropout_11 (Dropout)         (None, 121)               0
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1220
=================================================================
Total params: 433,627
Trainable params: 432,145
Non-trainable params: 1,482
_________________________________________________________________
```

In [50]:

```python
model_drop_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

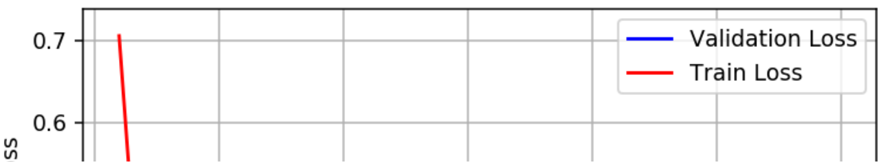
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 14s 227us/step - loss: 0.7051 - acc: 0.7825 - val_l
oss: 0.1937 - val_acc: 0.9398
Epoch 2/30
60000/60000 [==============================] - 11s 190us/step - loss: 0.2958 - acc: 0.9122 - val_l
oss: 0.1423 - val_acc: 0.9571
Epoch 3/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.2296 - acc: 0.9335 - val_l
oss: 0.1228 - val_acc: 0.9622
Epoch 4/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.1852 - acc: 0.9450 - val_l
oss: 0.1040 - val_acc: 0.9688
Epoch 5/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.1623 - acc: 0.9525 - val_l
oss: 0.0963 - val_acc: 0.9722
Epoch 6/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.1507 - acc: 0.9566 - val_l
oss: 0.0854 - val_acc: 0.9741
Epoch 7/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.1365 - acc: 0.9603 - val_l
oss: 0.0850 - val_acc: 0.9747
Epoch 8/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.1241 - acc: 0.9631 - val_l
oss: 0.0837 - val_acc: 0.9755
Epoch 9/30
60000/60000 [==============================] - 11s 175us/step - loss: 0.1148 - acc: 0.9662 - val_l
oss: 0.0748 - val_acc: 0.9773
Epoch 10/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.1088 - acc: 0.9678 - val_l
oss: 0.0690 - val_acc: 0.9804
Epoch 11/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.1023 - acc: 0.9699 - val_l
oss: 0.0676 - val_acc: 0.9799
Epoch 12/30
60000/60000 [==============================] - 11s 186us/step - loss: 0.1000 - acc: 0.9695 - val_l
oss: 0.0669 - val_acc: 0.9815
Epoch 13/30
60000/60000 [==============================] - 11s 190us/step - loss: 0.0952 - acc: 0.9709 - val_l
oss: 0.0672 - val_acc: 0.9805
Epoch 14/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0869 - acc: 0.9739 - val_l
oss: 0.0658 - val_acc: 0.9814
Epoch 15/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.0851 - acc: 0.9742 - val_l
oss: 0.0635 - val_acc: 0.9813
```

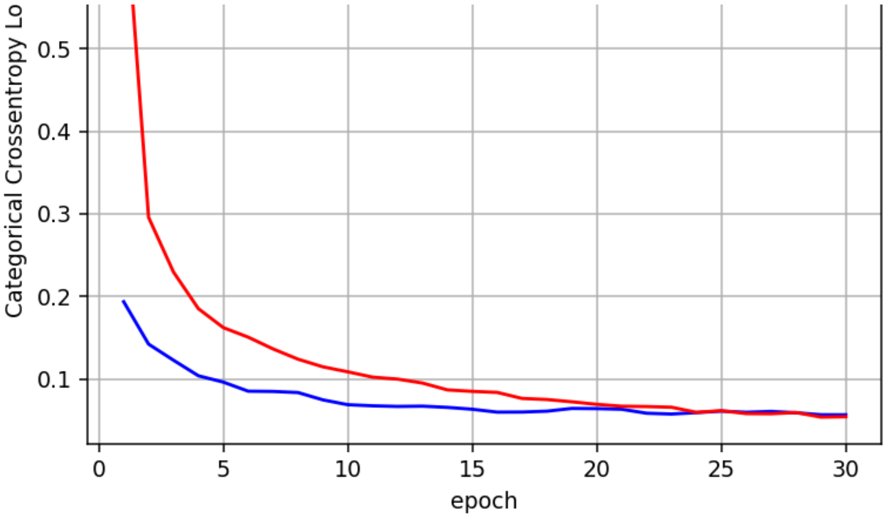
```
Epoch 16/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0839 - acc: 0.9744 - val_l
oss: 0.0600 - val_acc: 0.9824
Epoch 17/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0766 - acc: 0.9762 - val_l
oss: 0.0601 - val_acc: 0.9830
Epoch 18/30
60000/60000 [==============================] - 12s 192us/step - loss: 0.0753 - acc: 0.9770 - val_l
oss: 0.0612 - val_acc: 0.9824
Epoch 19/30
60000/60000 [==============================] - 11s 187us/step - loss: 0.0725 - acc: 0.9784 - val_l
oss: 0.0645 - val_acc: 0.9820
Epoch 20/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0694 - acc: 0.9793 - val_l
oss: 0.0642 - val_acc: 0.9827
Epoch 21/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0671 - acc: 0.9801 - val_l
oss: 0.0635 - val_acc: 0.9824
Epoch 22/30
60000/60000 [==============================] - 11s 186us/step - loss: 0.0668 - acc: 0.9801 - val_l
oss: 0.0587 - val_acc: 0.9833
Epoch 23/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0659 - acc: 0.9804 - val_l
oss: 0.0577 - val_acc: 0.9849
Epoch 24/30
60000/60000 [==============================] - 12s 193us/step - loss: 0.0597 - acc: 0.9821 - val_l
oss: 0.0594 - val_acc: 0.9840
Epoch 25/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0617 - acc: 0.9809 - val_l
oss: 0.0612 - val_acc: 0.9832
Epoch 26/30
60000/60000 [==============================] - 11s 188us/step - loss: 0.0583 - acc: 0.9824 - val_l
oss: 0.0598 - val_acc: 0.9841
Epoch 27/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.0581 - acc: 0.9819 - val_l
oss: 0.0609 - val_acc: 0.9839
Epoch 28/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.0595 - acc: 0.9824 - val_l
oss: 0.0593 - val_acc: 0.9839
Epoch 29/30
60000/60000 [==============================] - 12s 192us/step - loss: 0.0540 - acc: 0.9836 - val_l
oss: 0.0568 - val_acc: 0.9848
Epoch 30/30
60000/60000 [==============================] - 11s 187us/step - loss: 0.0544 - acc: 0.9832 - val_l
oss: 0.0568 - val_acc: 0.9858
```

In [51]:

```python
score_drop_3 = model_drop_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_drop_3[0])
print('Test accuracy:', score_drop_3[1])

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

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

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

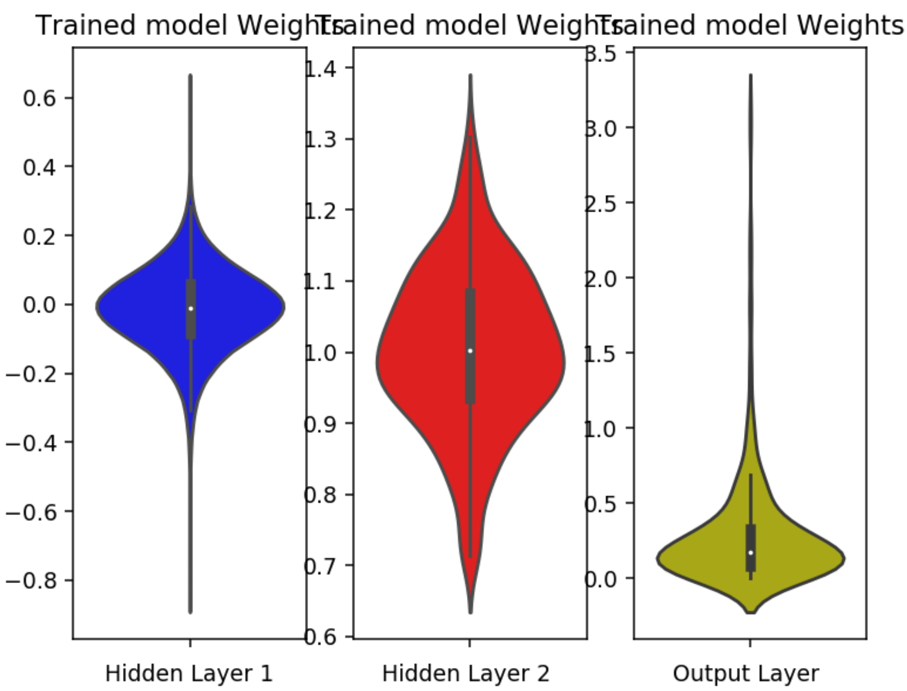
```
Test score: 0.05677158319847949
Test accuracy: 0.9858
```





In [52]:

```
w_after = model_drop_3.get_weights()

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


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

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

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



5 Layered Architecture

Relu + Adam

In [53]:

```
# start building a model
model_relu_5 = Sequential()
model_relu_5.add(Dense(697, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.053, seed=None)))
model_relu_5.add(Dense(550, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
60, seed=None)) )
model_relu_5.add(Dense(462, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
65, seed=None)) )
model_relu_5.add(Dense(322, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
78, seed=None)) )
model_relu_5.add(Dense(199, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
00, seed=None)) )
model_relu_5.add(Dense(output_dim, activation='softmax'))

print(model_relu_5.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 697)               547145    
_________________________________________________________________
dense_36 (Dense)             (None, 550)               383900    
_________________________________________________________________
dense_37 (Dense)             (None, 462)               254562    
_________________________________________________________________
dense_38 (Dense)             (None, 322)               149086    
_________________________________________________________________
dense_39 (Dense)             (None, 199)               64277     
_________________________________________________________________
dense_40 (Dense)             (None, 10)                2000      
=================================================================
Total params: 1,400,970
Trainable params: 1,400,970
Non-trainable params: 0
_________________________________________________________________
None
```

In [54]:

```
model_relu_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 20s 336us/step - loss: 0.2144 - acc: 0.9343 - val_l
oss: 0.1240 - val_acc: 0.9648
Epoch 2/30
60000/60000 [==============================] - 18s 302us/step - loss: 0.0875 - acc: 0.9726 - val_l
oss: 0.0997 - val_acc: 0.9691
Epoch 3/30
60000/60000 [==============================] - 18s 295us/step - loss: 0.0587 - acc: 0.9813 - val_l
oss: 0.0940 - val_acc: 0.9722
Epoch 4/30
60000/60000 [==============================] - 17s 288us/step - loss: 0.0467 - acc: 0.9851 - val_l
oss: 0.0828 - val_acc: 0.9745
Epoch 5/30
60000/60000 [==============================] - 19s 324us/step - loss: 0.0387 - acc: 0.9877 - val_l
oss: 0.1106 - val_acc: 0.9712
Epoch 6/30
60000/60000 [==============================] - 18s 300us/step - loss: 0.0337 - acc: 0.9900 - val_l
oss: 0.0836 - val_acc: 0.9748
```

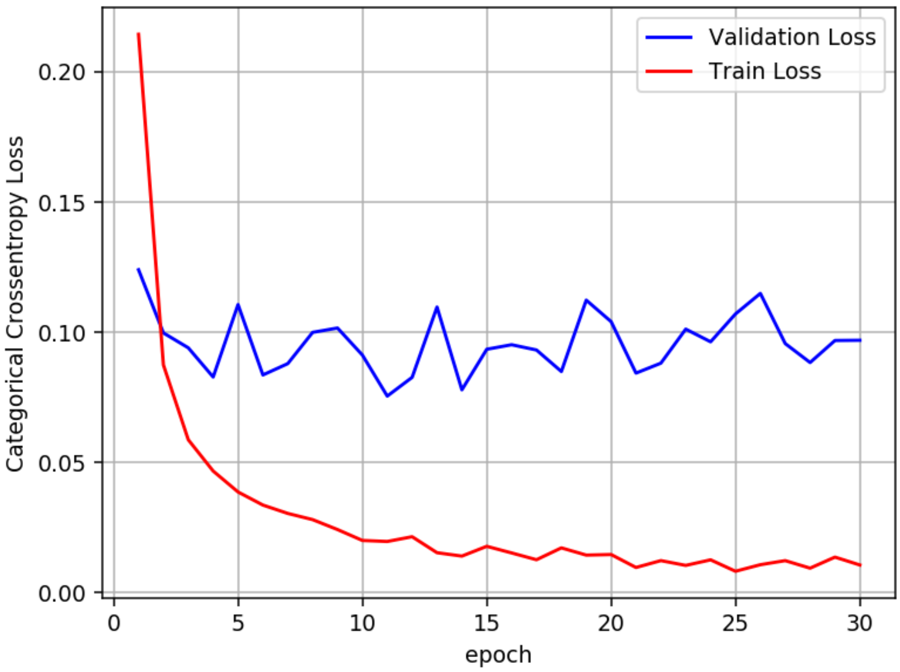
```
Epoch 7/30
60000/60000 [==============================] - 18s 292us/step - loss: 0.0305 - acc: 0.9906 - val_l
oss: 0.0880 - val_acc: 0.9793
Epoch 8/30
60000/60000 [==============================] - 18s 308us/step - loss: 0.0281 - acc: 0.9917 - val_l
oss: 0.1000 - val_acc: 0.9753
Epoch 9/30
60000/60000 [==============================] - 18s 297us/step - loss: 0.0243 - acc: 0.9925 - val_l
oss: 0.1016 - val_acc: 0.9757
Epoch 10/30
60000/60000 [==============================] - 17s 287us/step - loss: 0.0201 - acc: 0.9939 - val_l
oss: 0.0913 - val_acc: 0.9784
Epoch 11/30
60000/60000 [==============================] - 17s 290us/step - loss: 0.0197 - acc: 0.9939 - val_l
oss: 0.0755 - val_acc: 0.9827
Epoch 12/30
60000/60000 [==============================] - 18s 302us/step - loss: 0.0215 - acc: 0.9936 - val_l
oss: 0.0827 - val_acc: 0.9807
Epoch 13/30
60000/60000 [==============================] - 17s 286us/step - loss: 0.0154 - acc: 0.9951 - val_l
oss: 0.1097 - val_acc: 0.9790
Epoch 14/30
60000/60000 [==============================] - 18s 292us/step - loss: 0.0141 - acc: 0.9962 - val_l
oss: 0.0779 - val_acc: 0.9821
Epoch 15/30
60000/60000 [==============================] - 19s 310us/step - loss: 0.0178 - acc: 0.9950 - val_l
oss: 0.0935 - val_acc: 0.9790
Epoch 16/30
60000/60000 [==============================] - 18s 292us/step - loss: 0.0153 - acc: 0.9956 - val_l
oss: 0.0952 - val_acc: 0.9795
Epoch 17/30
60000/60000 [==============================] - 18s 306us/step - loss: 0.0127 - acc: 0.9964 - val_l
oss: 0.0932 - val_acc: 0.9799
Epoch 18/30
60000/60000 [==============================] - 18s 306us/step - loss: 0.0172 - acc: 0.9954 - val_l
oss: 0.0849 - val_acc: 0.9815
Epoch 19/30
60000/60000 [==============================] - 18s 298us/step - loss: 0.0145 - acc: 0.9960 - val_l
oss: 0.1123 - val_acc: 0.9778
Epoch 20/30
60000/60000 [==============================] - 18s 296us/step - loss: 0.0147 - acc: 0.9967 - val_l
oss: 0.1041 - val_acc: 0.9806
Epoch 21/30
60000/60000 [==============================] - 18s 293us/step - loss: 0.0097 - acc: 0.9972 - val_l
oss: 0.0843 - val_acc: 0.9835
Epoch 22/30
60000/60000 [==============================] - 18s 302us/step - loss: 0.0124 - acc: 0.9968 - val_l
oss: 0.0881 - val_acc: 0.9824
Epoch 23/30
60000/60000 [==============================] - 17s 290us/step - loss: 0.0105 - acc: 0.9970 - val_l
oss: 0.1012 - val_acc: 0.9819
Epoch 24/30
60000/60000 [==============================] - 18s 292us/step - loss: 0.0127 - acc: 0.9967 - val_l
oss: 0.0963 - val_acc: 0.9805
Epoch 25/30
60000/60000 [==============================] - 19s 313us/step - loss: 0.0083 - acc: 0.9977 - val_l
oss: 0.1070 - val_acc: 0.9827
Epoch 26/30
60000/60000 [==============================] - 17s 287us/step - loss: 0.0108 - acc: 0.9970 - val_l
oss: 0.1149 - val_acc: 0.9806
Epoch 27/30
60000/60000 [==============================] - 17s 288us/step - loss: 0.0124 - acc: 0.9970 - val_l
oss: 0.0957 - val_acc: 0.9822
Epoch 28/30
60000/60000 [==============================] - 17s 288us/step - loss: 0.0095 - acc: 0.9976 - val_l
oss: 0.0884 - val_acc: 0.9837
Epoch 29/30
60000/60000 [==============================] - 18s 304us/step - loss: 0.0137 - acc: 0.9965 - val_l
oss: 0.0969 - val_acc: 0.9824
Epoch 30/30
60000/60000 [==============================] - 18s 293us/step - loss: 0.0107 - acc: 0.9974 - val_l
oss: 0.0969 - val_acc: 0.9801
```

In [55]:

```
score_relu_5 = model_relu_5.evaluate(X_test, Y_test, verbose=0)
```

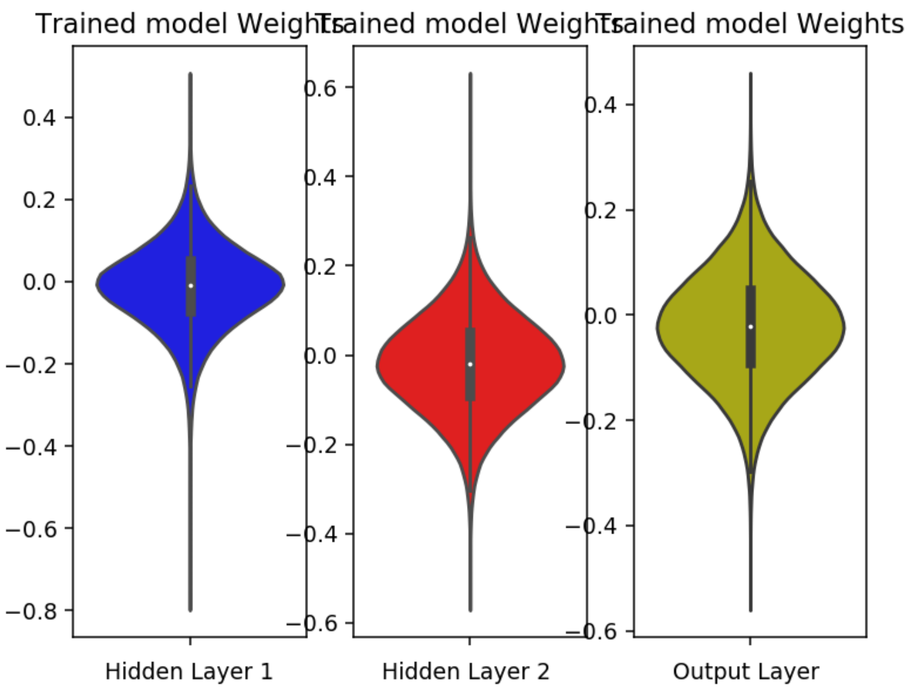
```
print('Test score:', score_relu_5[0])
print('Test accuracy:', score_relu_5[1])

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

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

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

```
Test score: 0.09693762213855957
Test accuracy: 0.9801
```



In [56]:

```
w_after = model_relu_5.get_weights()

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


fig = plt.figure()
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()
```



Batch Normalization on Hidden Layers + Relu + Adam

In [57]:

```
model_batch_5 = Sequential()

model_batch_5.add(Dense(667, activation='relu', input_shape=(input_dim,), kernel_initializer=Random
Normal(mean=0.0, stddev=0.054, seed=None)))
model_batch_5.add(BatchNormalization())

model_batch_5.add(Dense(579, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
058, seed=None)) )
model_batch_5.add(BatchNormalization())

model_batch_5.add(Dense(499, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
063, seed=None)) )
model_batch_5.add(BatchNormalization())

model_batch_5.add(Dense(349, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
075, seed=None)) )
model_batch_5.add(BatchNormalization())

model_batch_5.add(Dense(205, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.
098, seed=None)) )
model_batch_5.add(BatchNormalization())

model_batch_5.add(Dense(output_dim, activation='softmax'))


model_batch_5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 667)               523595
_________________________________________________________________
batch_normalization_20 (Batc (None, 667)               2668
_________________________________________________________________
dense_42 (Dense)             (None, 579)               386772
_________________________________________________________________
batch_normalization_21 (Batc (None, 579)               2316
_________________________________________________________________
dense_43 (Dense)             (None, 499)               289420
_________________________________________________________________
batch_normalization_22 (Batc (None, 499)               1996
```

```
_________________________________________________________________
dense_44 (Dense)             (None, 349)               174500
_________________________________________________________________
batch_normalization_23 (Batc (None, 349)               1396
_________________________________________________________________
dense_45 (Dense)             (None, 205)               71750
_________________________________________________________________
batch_normalization_24 (Batc (None, 205)               820
_________________________________________________________________
dense_46 (Dense)             (None, 10)                2060
=================================================================
Total params: 1,457,293
Trainable params: 1,452,695
Non-trainable params: 4,598
_________________________________________________________________
```

In [58]:

```
model_batch_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))
```

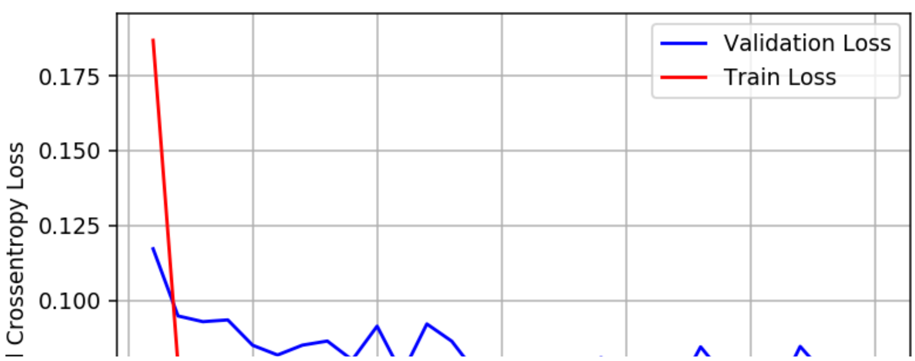
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 28s 466us/step - loss: 0.1868 - acc: 0.9430 - val_l
oss: 0.1172 - val_acc: 0.9652
Epoch 2/30
60000/60000 [==============================] - 24s 401us/step - loss: 0.0786 - acc: 0.9756 - val_l
oss: 0.0948 - val_acc: 0.9700
Epoch 3/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0562 - acc: 0.9816 - val_l
oss: 0.0929 - val_acc: 0.9730
Epoch 4/30
60000/60000 [==============================] - 24s 401us/step - loss: 0.0463 - acc: 0.9853 - val_l
oss: 0.0935 - val_acc: 0.9725
Epoch 5/30
60000/60000 [==============================] - 26s 427us/step - loss: 0.0406 - acc: 0.9869 - val_l
oss: 0.0850 - val_acc: 0.9742
Epoch 6/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.0339 - acc: 0.9886 - val_l
oss: 0.0818 - val_acc: 0.9769
Epoch 7/30
60000/60000 [==============================] - 24s 403us/step - loss: 0.0326 - acc: 0.9893 - val_l
oss: 0.0851 - val_acc: 0.9754
Epoch 8/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0264 - acc: 0.9912 - val_l
oss: 0.0864 - val_acc: 0.9775
Epoch 9/30
60000/60000 [==============================] - 24s 399us/step - loss: 0.0253 - acc: 0.9919 - val_l
oss: 0.0801 - val_acc: 0.9782
Epoch 10/30
60000/60000 [==============================] - 25s 414us/step - loss: 0.0233 - acc: 0.9920 - val_l
oss: 0.0914 - val_acc: 0.9743
Epoch 11/30
60000/60000 [==============================] - 24s 400us/step - loss: 0.0239 - acc: 0.9923 - val_l
oss: 0.0754 - val_acc: 0.9794
Epoch 12/30
60000/60000 [==============================] - 24s 404us/step - loss: 0.0201 - acc: 0.9932 - val_l
oss: 0.0921 - val_acc: 0.9759
Epoch 13/30
60000/60000 [==============================] - 25s 415us/step - loss: 0.0224 - acc: 0.9924 - val_l
oss: 0.0864 - val_acc: 0.9785
Epoch 14/30
60000/60000 [==============================] - 24s 400us/step - loss: 0.0158 - acc: 0.9947 - val_l
oss: 0.0756 - val_acc: 0.9798
Epoch 15/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0183 - acc: 0.9938 - val_l
oss: 0.0750 - val_acc: 0.9809
Epoch 16/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.0169 - acc: 0.9947 - val_l
oss: 0.0672 - val_acc: 0.9831
Epoch 17/30
60000/60000 [==============================] - 24s 405us/step - loss: 0.0139 - acc: 0.9957 - val_l
oss: 0.0648 - val_acc: 0.9832
Epoch 18/30
```

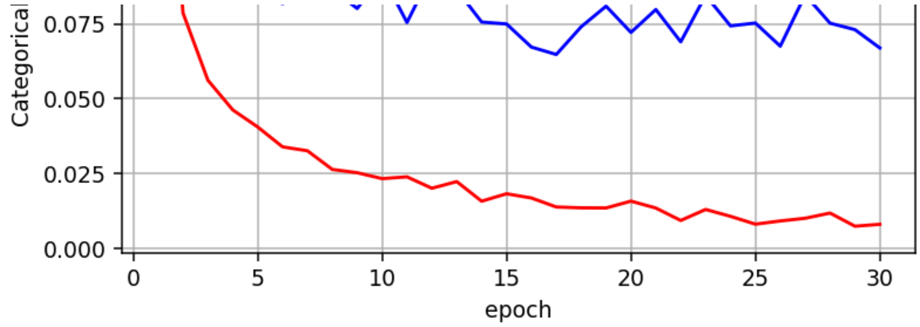
```
60000/60000 [==============================] - 25s 418us/step - loss: 0.0136 - acc: 0.9954 - val_l
oss: 0.0740 - val_acc: 0.9828
Epoch 19/30
60000/60000 [==============================] - 24s 406us/step - loss: 0.0135 - acc: 0.9957 - val_l
oss: 0.0809 - val_acc: 0.9819
Epoch 20/30
60000/60000 [==============================] - 25s 416us/step - loss: 0.0158 - acc: 0.9952 - val_l
oss: 0.0721 - val_acc: 0.9813
Epoch 21/30
60000/60000 [==============================] - 24s 406us/step - loss: 0.0135 - acc: 0.9955 - val_l
oss: 0.0798 - val_acc: 0.9808
Epoch 22/30
60000/60000 [==============================] - 25s 412us/step - loss: 0.0094 - acc: 0.9969 - val_l
oss: 0.0690 - val_acc: 0.9835
Epoch 23/30
60000/60000 [==============================] - 25s 409us/step - loss: 0.0130 - acc: 0.9956 - val_l
oss: 0.0845 - val_acc: 0.9805
Epoch 24/30
60000/60000 [==============================] - 24s 400us/step - loss: 0.0108 - acc: 0.9964 - val_l
oss: 0.0743 - val_acc: 0.9826
Epoch 25/30
60000/60000 [==============================] - 25s 416us/step - loss: 0.0081 - acc: 0.9973 - val_l
oss: 0.0753 - val_acc: 0.9825
Epoch 26/30
60000/60000 [==============================] - 24s 404us/step - loss: 0.0092 - acc: 0.9970 - val_l
oss: 0.0675 - val_acc: 0.9826
Epoch 27/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0101 - acc: 0.9966 - val_l
oss: 0.0846 - val_acc: 0.9821
Epoch 28/30
60000/60000 [==============================] - 24s 404us/step - loss: 0.0118 - acc: 0.9962 - val_l
oss: 0.0752 - val_acc: 0.9821
Epoch 29/30
60000/60000 [==============================] - 25s 408us/step - loss: 0.0075 - acc: 0.9978 - val_l
oss: 0.0731 - val_acc: 0.9828
Epoch 30/30
60000/60000 [==============================] - 25s 421us/step - loss: 0.0081 - acc: 0.9976 - val_l
oss: 0.0669 - val_acc: 0.9851
```

In [59]:

```python
score_batch_5 = model_batch_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_batch_5[0])
print('Test accuracy:', score_batch_5[1])

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

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

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

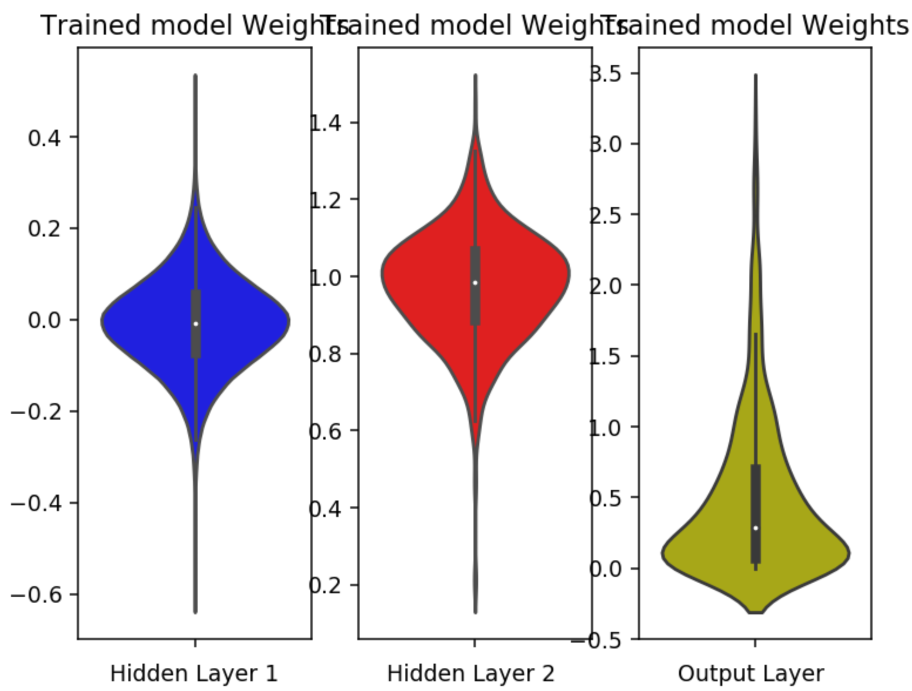
```
Test score: 0.06694545334176073
Test accuracy: 0.9851
```





In [60]:

```
w_after = model_batch_5.get_weights()

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


fig = plt.figure()
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()
```



Dropout + Relu + Adam

In [61]:

```
model_drop_5 = Sequential()

model_drop_5.add(Dense(609, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.057, seed=None)))
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(599, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
57, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(410, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
69, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(232, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0
92, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(144, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
17, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(output_dim, activation='softmax'))


model_drop_5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 609)               478065
_________________________________________________________________
batch_normalization_25 (Batc (None, 609)               2436
_________________________________________________________________
dropout_12 (Dropout)         (None, 609)               0
_________________________________________________________________
dense_48 (Dense)             (None, 599)               365390
_________________________________________________________________
batch_normalization_26 (Batc (None, 599)               2396
_________________________________________________________________
dropout_13 (Dropout)         (None, 599)               0
_________________________________________________________________
dense_49 (Dense)             (None, 410)               246000
_________________________________________________________________
batch_normalization_27 (Batc (None, 410)               1640
_________________________________________________________________
dropout_14 (Dropout)         (None, 410)               0
_________________________________________________________________
dense_50 (Dense)             (None, 232)               95352
_________________________________________________________________
batch_normalization_28 (Batc (None, 232)               928
_________________________________________________________________
dropout_15 (Dropout)         (None, 232)               0
_________________________________________________________________
dense_51 (Dense)             (None, 144)               33552
_________________________________________________________________
batch_normalization_29 (Batc (None, 144)               576
_________________________________________________________________
dropout_16 (Dropout)         (None, 144)               0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1450
=================================================================
Total params: 1,227,785
Trainable params: 1,223,797
Non-trainable params: 3,988
_________________________________________________________________
```

In [62]:

```
model_drop_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

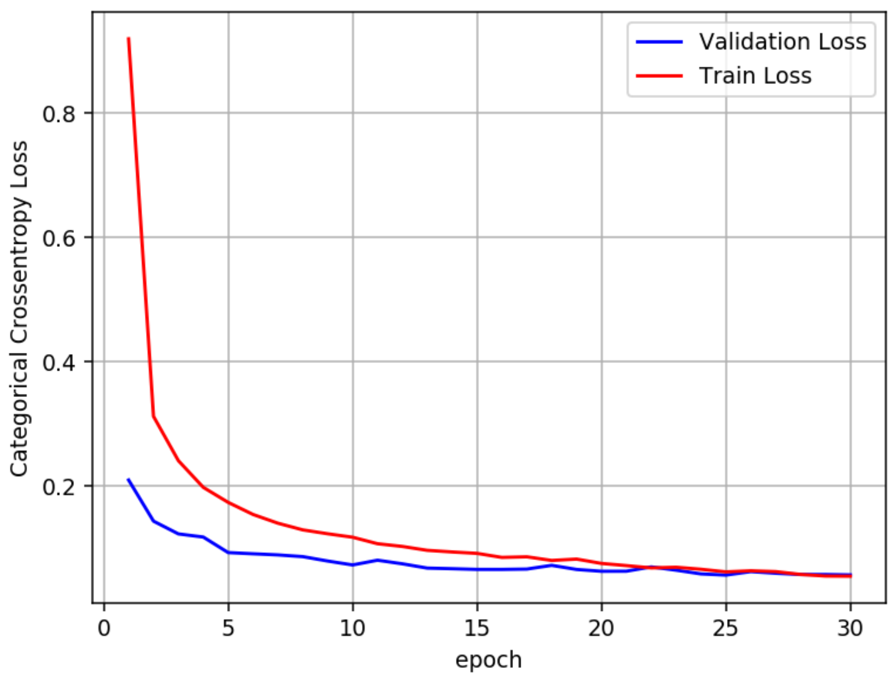
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 29s 475us/step - loss: 0.9197 - acc: 0.7174 - val_l
oss: 0.2094 - val_acc: 0.9404
Epoch 2/30
60000/60000 [==============================] - 25s 423us/step - loss: 0.3119 - acc: 0.9107 - val_l
oss: 0.1432 - val_acc: 0.9593
Epoch 3/30
60000/60000 [==============================] - 24s 398us/step - loss: 0.2406 - acc: 0.9318 - val_l
oss: 0.1227 - val_acc: 0.9645
Epoch 4/30
60000/60000 [==============================] - 24s 400us/step - loss: 0.1976 - acc: 0.9450 - val_l
oss: 0.1176 - val_acc: 0.9667
Epoch 5/30
60000/60000 [==============================] - 25s 417us/step - loss: 0.1734 - acc: 0.9512 - val_l
oss: 0.0926 - val_acc: 0.9736
Epoch 6/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.1540 - acc: 0.9561 - val_l
oss: 0.0907 - val_acc: 0.9744
Epoch 7/30
60000/60000 [==============================] - 25s 414us/step - loss: 0.1399 - acc: 0.9607 - val_l
oss: 0.0889 - val_acc: 0.9750
Epoch 8/30
60000/60000 [==============================] - 24s 400us/step - loss: 0.1291 - acc: 0.9629 - val_l
oss: 0.0861 - val_acc: 0.9766
Epoch 9/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.1229 - acc: 0.9653 - val_l
oss: 0.0790 - val_acc: 0.9770
Epoch 10/30
60000/60000 [==============================] - 25s 410us/step - loss: 0.1173 - acc: 0.9662 - val_l
oss: 0.0727 - val_acc: 0.9806
Epoch 11/30
60000/60000 [==============================] - 24s 401us/step - loss: 0.1069 - acc: 0.9700 - val_l
oss: 0.0803 - val_acc: 0.9777s
Epoch 12/30
60000/60000 [==============================] - 25s 416us/step - loss: 0.1025 - acc: 0.9717 - val_l
oss: 0.0746 - val_acc: 0.9803
Epoch 13/30
60000/60000 [==============================] - 24s 407us/step - loss: 0.0962 - acc: 0.9730 - val_l
oss: 0.0676 - val_acc: 0.9809
Epoch 14/30
60000/60000 [==============================] - 24s 403us/step - loss: 0.0936 - acc: 0.9731 - val_l
oss: 0.0667 - val_acc: 0.9830
Epoch 15/30
60000/60000 [==============================] - 25s 415us/step - loss: 0.0913 - acc: 0.9737 - val_l
oss: 0.0655 - val_acc: 0.9817
Epoch 16/30
60000/60000 [==============================] - 24s 403us/step - loss: 0.0848 - acc: 0.9761 - val_l
oss: 0.0655 - val_acc: 0.9823
Epoch 17/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0858 - acc: 0.9747 - val_l
oss: 0.0661 - val_acc: 0.9813
Epoch 18/30
60000/60000 [==============================] - 24s 407us/step - loss: 0.0800 - acc: 0.9772 - val_l
oss: 0.0721 - val_acc: 0.9821
Epoch 19/30
60000/60000 [==============================] - 25s 421us/step - loss: 0.0822 - acc: 0.9763 - val_l
oss: 0.0655 - val_acc: 0.9823
Epoch 20/30
60000/60000 [==============================] - 24s 408us/step - loss: 0.0752 - acc: 0.9781 - val_l
oss: 0.0625 - val_acc: 0.9839
Epoch 21/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.0717 - acc: 0.9797 - val_l
oss: 0.0626 - val_acc: 0.9828
Epoch 22/30
60000/60000 [==============================] - 25s 417us/step - loss: 0.0680 - acc: 0.9800 - val_l
oss: 0.0696 - val_acc: 0.9824
Epoch 23/30
60000/60000 [==============================] - 24s 402us/step - loss: 0.0690 - acc: 0.9800 - val_l
oss: 0.0643 - val_acc: 0.9835
Epoch 24/30
60000/60000 [==============================] - 25s 418us/step - loss: 0.0658 - acc: 0.9806 - val_l
```

```
oss: 0.0583 - val_acc: 0.9839
Epoch 25/30
60000/60000 [==============================] - 24s 403us/step - loss: 0.0616 - acc: 0.9829 - val_l
oss: 0.0564 - val_acc: 0.9848
Epoch 26/30
60000/60000 [==============================] - 25s 408us/step - loss: 0.0633 - acc: 0.9815 - val_l
oss: 0.0620 - val_acc: 0.9840
Epoch 27/30
60000/60000 [==============================] - 25s 416us/step - loss: 0.0620 - acc: 0.9823 - val_l
oss: 0.0594 - val_acc: 0.9843
Epoch 28/30
60000/60000 [==============================] - 24s 405us/step - loss: 0.0574 - acc: 0.9836 - val_l
oss: 0.0573 - val_acc: 0.9852
Epoch 29/30
60000/60000 [==============================] - 25s 417us/step - loss: 0.0549 - acc: 0.9839 - val_l
oss: 0.0573 - val_acc: 0.9844
Epoch 30/30
60000/60000 [==============================] - 24s 405us/step - loss: 0.0546 - acc: 0.9848 - val_l
oss: 0.0567 - val_acc: 0.9855
```

In [63]:

```
score_drop_5 = model_drop_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_drop_5[0])
print('Test accuracy:', score_drop_5[1])

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

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

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

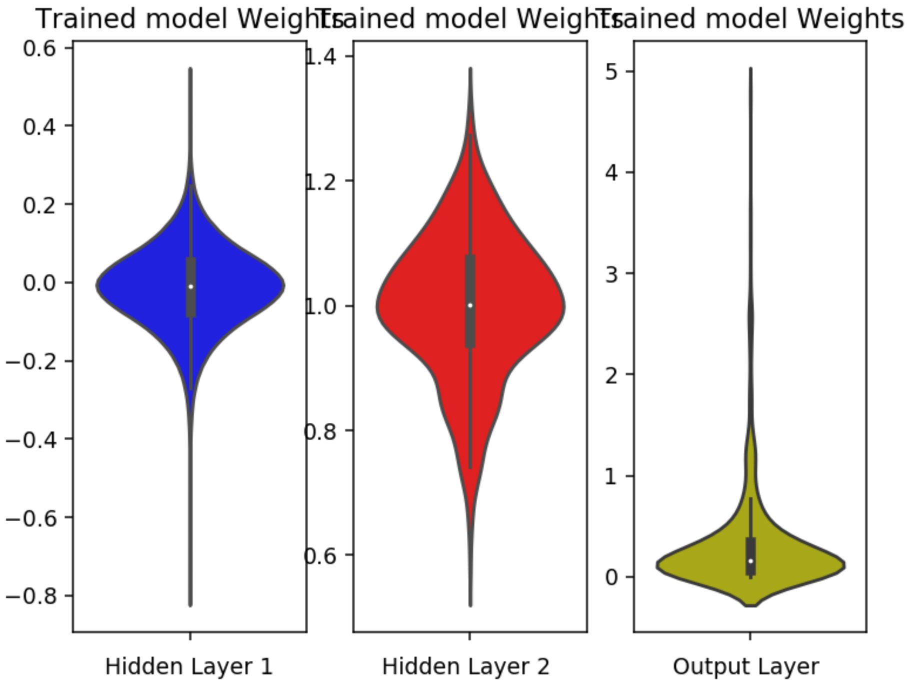
```
Test score: 0.056678825084562415
Test accuracy: 0.9855
```



In [64]:

```
w_after = model_drop_5.get_weights()

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

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


fig = plt.figure()
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()
```



Prettytable

In [74]:

```
number= [1,2,3,4,5,6,7,8,9]
name= ["Relu layer 2","Relu layer 3","Relu layer 5"," Batch layer 2"," Batch layer 3"," Batch layer
5","Drop layer 2","Drop layer 3","Drop layer 5"]
score=
[score_relu_2[0],score_relu_3[0],score_relu_5[0],score_batch_2[0],score_batch_3[0],score_batch_5[0
],score_drop_2[0],score_drop_3[0],score_drop_5[0]]
acc=
[score_relu_2[1],score_relu_3[1],score_relu_5[1],score_batch_2[1],score_batch_3[1],score_batch_5[1
],score_drop_2[1],score_drop_3[1],score_drop_5[1]]

#Initialize Prettytable
ptable = PrettyTable()
ptable.add_column("Index", number)
ptable.add_column("Architecture", name)
ptable.add_column("Test Score", score)
ptable.add_column("Test Accuracy", acc)

print(ptable)
```

Index	Architecture	Test Score	Test Accuracy
1	Relu layer 2	0.09564156731741441	0.9817
2	Relu layer 3	0.09514001322890968	0.9816
3	Relu layer 5	0.09693762213855957	0.9801
4	Batch layer 2	0.09071999986333348	0.9804
5	Batch layer 3	0.08639855843480118	0.9817
6	Batch layer 5	0.06694545334176073	0.9851
7	Drop layer 2	0.052775146638258594	0.9848
8	Drop layer 3	0.05677158319847949	0.9858
9	Drop layer 5	0.056678825084562415	0.9855

Conclusions

1. We implement Keras on MNIST dataset.
2. Different layered architectures like 2, 3, 5 hidden layers are used to build Neural networks.
3. Activation method RELU is used and for optimization ADAM is used.
4. In each layered network, Batch normalization and dropout layer is also added to check the performance of model.
5. Test accuracy is same for all architectures.
6. Test score is same in case of Relu, in case of dropout test score is low than Relu.

```
In [ ]:
```