1. Keras - MLPs on MNIST

2. Import Libraries

In [1]:

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use th
from keras.utils import np_utils
from keras.datasets import mnist
from keras.optimizers import Adam,RMSprop,SGD
from keras.layers import Dropout
from keras.layers import Dense, Activation
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential
from keras.initializers import RandomNormal
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import time
```

```
Using TensorFlow backend.
```

3. Plot Function

In [2]:

```
# 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()


```

4. Data

In [3]:

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

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

# 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])

# 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 (
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%


```

```
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)
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 [4]:

```
# 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]
```

5. Normalise the data

In [5]:

```
# 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

# example data point after normlizing
print(X_train[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.
```

6. Encode Label

In [6]:

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

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

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

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

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

7. Model (MLP)

7.1 Parameters

In [7]:

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

batch_size = 128
nb_epoch = 10


```

7.2.1 MLP (2 hidden layers) + Batch-Norm + Dropout + AdamOptimizer

In [8]:

```

model_1 = Sequential()

model_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer
model_1.add(BatchNormalization())
model_1.add(Dropout(0.5))

model_1.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model_1.add(BatchNormalization())
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

```
WARNING:tensorflow:From C:\Users\Byron\Applications\PythonMaster\lib\site-pa
ckages\tensorflow\python\framework\op_def_library.py:263: colocate_with (fro
m tensorflow.python.framework.ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\Byron\Applications\PythonMaster\lib\site-pa
ckages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tenso
rflow.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_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [9]:

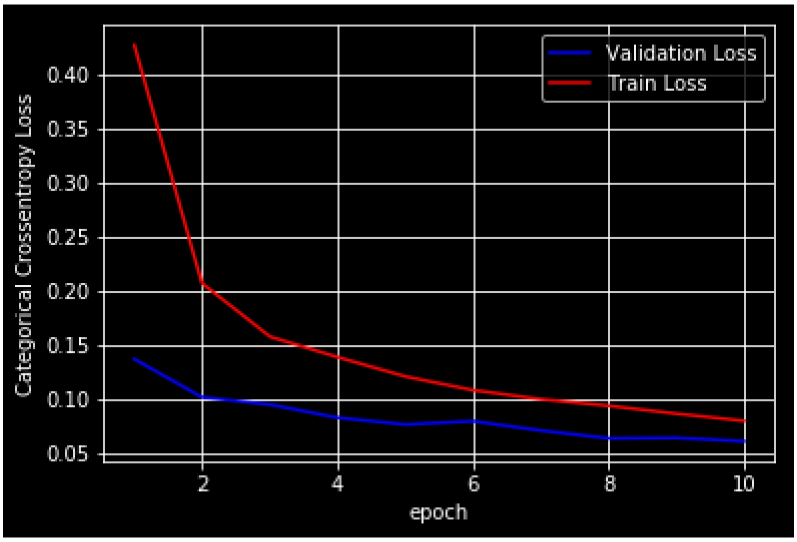
```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
WARNING:tensorflow:From C:\Users\Byron\Applications\PythonMaster\lib\site-pa
ckages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.pyt
hon.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 8s 136us/step - loss: 0.4271
- acc: 0.8700 - val_loss: 0.1373 - val_acc: 0.9563
Epoch 2/10
60000/60000 [==============================] - 7s 124us/step - loss: 0.2069
- acc: 0.9383 - val_loss: 0.1020 - val_acc: 0.9682
Epoch 3/10
60000/60000 [==============================] - 7s 121us/step - loss: 0.1579
- acc: 0.9527 - val_loss: 0.0952 - val_acc: 0.9699
Epoch 4/10
60000/60000 [==============================] - 7s 120us/step - loss: 0.1390
- acc: 0.9589 - val_loss: 0.0831 - val_acc: 0.9735
Epoch 5/10
60000/60000 [==============================] - 7s 121us/step - loss: 0.1212
- acc: 0.9623 - val_loss: 0.0769 - val_acc: 0.9758
Epoch 6/10
60000/60000 [==============================] - 7s 121us/step - loss: 0.1085
- acc: 0.9658 - val_loss: 0.0799 - val_acc: 0.9760
Epoch 7/10
60000/60000 [==============================] - 7s 122us/step - loss: 0.1002
- acc: 0.9689 - val_loss: 0.0713 - val_acc: 0.9779
Epoch 8/10
60000/60000 [==============================] - 7s 122us/step - loss: 0.0941
- acc: 0.9710 - val_loss: 0.0641 - val_acc: 0.9804
Epoch 9/10
60000/60000 [==============================] - 7s 122us/step - loss: 0.0868
- acc: 0.9737 - val_loss: 0.0645 - val_acc: 0.9804
Epoch 10/10
60000/60000 [==============================] - 7s 124us/step - loss: 0.0802
- acc: 0.9746 - val_loss: 0.0614 - val_acc: 0.9819
```

In [10]:

```
score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

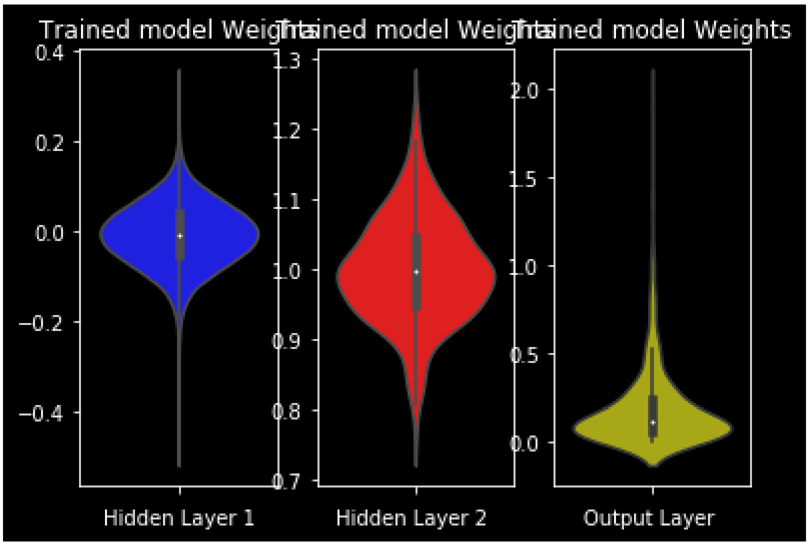
```
Test score: 0.061443126168311574
Test accuracy: 0.9819
```



In [11]:

```
w_after = model_1.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()

del(model_1)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



7.2.2 MLP (3 hidden layers) + Batch-Norm + Dropout + AdamOptimizer

In [12]:

```

model_2 = Sequential()

model_2.add(Dense(500, activation='relu', input_shape=(input_dim,), kernel_initializer
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(300, activation='relu', kernel_initializer='he_normal'))
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(output_dim, activation='softmax'))

model_2.summary()

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 500)               392500
_________________________________________________________________
batch_normalization_3 (Batch (None, 500)               2000
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_5 (Dense)              (None, 300)               150300
_________________________________________________________________
batch_normalization_4 (Batch (None, 300)               1200
_________________________________________________________________
dropout_4 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               30100
_________________________________________________________________
batch_normalization_5 (Batch (None, 100)               400
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010
=================================================================
Total params: 577,510
Trainable params: 575,710
Non-trainable params: 1,800
_________________________________________________________________
```

In [13]:

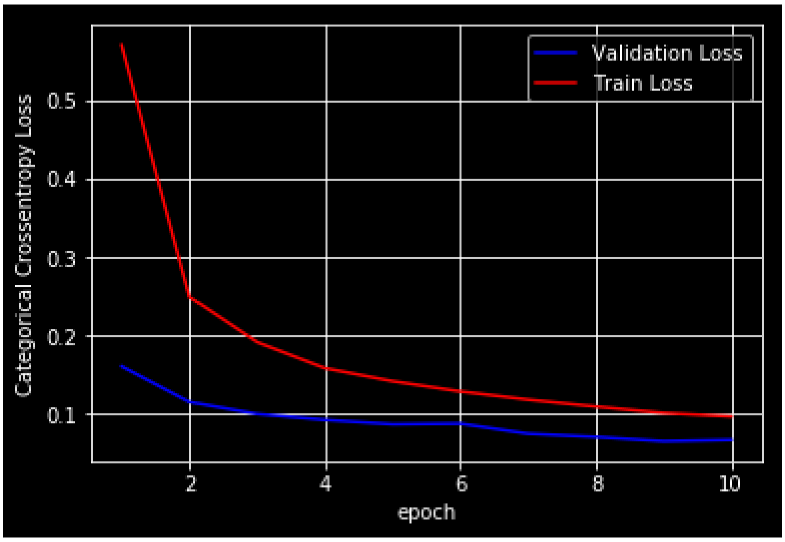
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 10s 162us/step - loss: 0.5704
- acc: 0.8269 - val_loss: 0.1611 - val_acc: 0.9505
Epoch 2/10
60000/60000 [==============================] - 9s 146us/step - loss: 0.2497
- acc: 0.9281 - val_loss: 0.1158 - val_acc: 0.9645
Epoch 3/10
60000/60000 [==============================] - 9s 147us/step - loss: 0.1918
- acc: 0.9439 - val_loss: 0.1005 - val_acc: 0.9705
Epoch 4/10
60000/60000 [==============================] - 9s 145us/step - loss: 0.1587
- acc: 0.9531 - val_loss: 0.0930 - val_acc: 0.9725
Epoch 5/10
60000/60000 [==============================] - 9s 142us/step - loss: 0.1421
- acc: 0.9579 - val_loss: 0.0873 - val_acc: 0.9732
Epoch 6/10
60000/60000 [==============================] - 8s 134us/step - loss: 0.1291
- acc: 0.9620 - val_loss: 0.0882 - val_acc: 0.9746
Epoch 7/10
60000/60000 [==============================] - 8s 134us/step - loss: 0.1187
- acc: 0.9644 - val_loss: 0.0754 - val_acc: 0.9783
Epoch 8/10
60000/60000 [==============================] - 8s 137us/step - loss: 0.1099
- acc: 0.9679 - val_loss: 0.0712 - val_acc: 0.9790
Epoch 9/10
60000/60000 [==============================] - 8s 137us/step - loss: 0.1017
- acc: 0.9699 - val_loss: 0.0657 - val_acc: 0.9809
Epoch 10/10
60000/60000 [==============================] - 8s 140us/step - loss: 0.0973
- acc: 0.9709 - val_loss: 0.0675 - val_acc: 0.9790
```

In [14]:

```
score = model_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

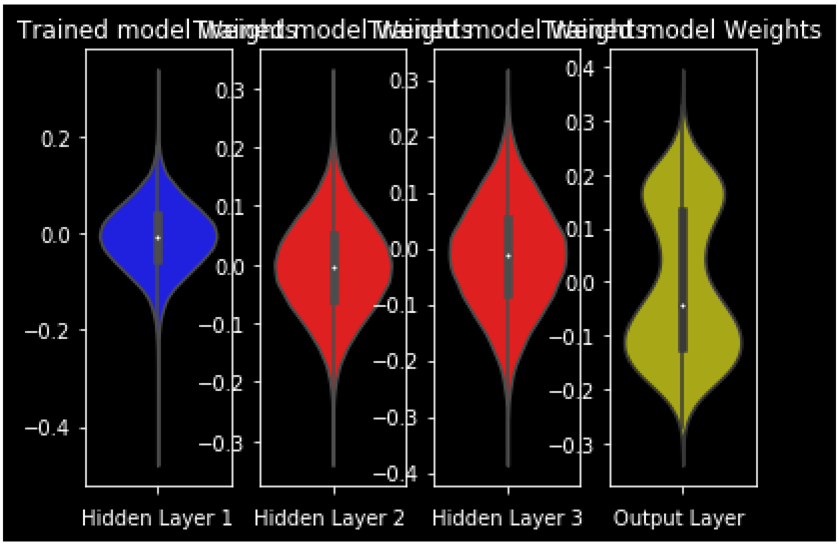
```
Test score: 0.06748124550754438
Test accuracy: 0.979
```



In [16]:

```
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
out_w = w_after[18].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()

del(model_2)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



7.2.3 MLP (25 hidden layers) + Batch-Norm + Dropout + AdamOptimizer

In [17]:

```

model_3 = Sequential()

model_3.add(Dense(200, activation='relu', input_shape=(input_dim,), kernel_initializer
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
```

```
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(output_dim, activation='softmax'))

model_3.summary()


```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 200)               157000    
_________________________________________________________________
batch_normalization_6 (Batch (None, 200)               800       
_________________________________________________________________
dropout_6 (Dropout)          (None, 200)               0         
```

```
dense_9 (Dense)              (None, 100)               20100
_________________________________________________________________
batch_normalization_7 (Batch (None, 100)               400
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               10100
_________________________________________________________________
batch_normalization_8 (Batch (None, 100)               400
```

In [18]:

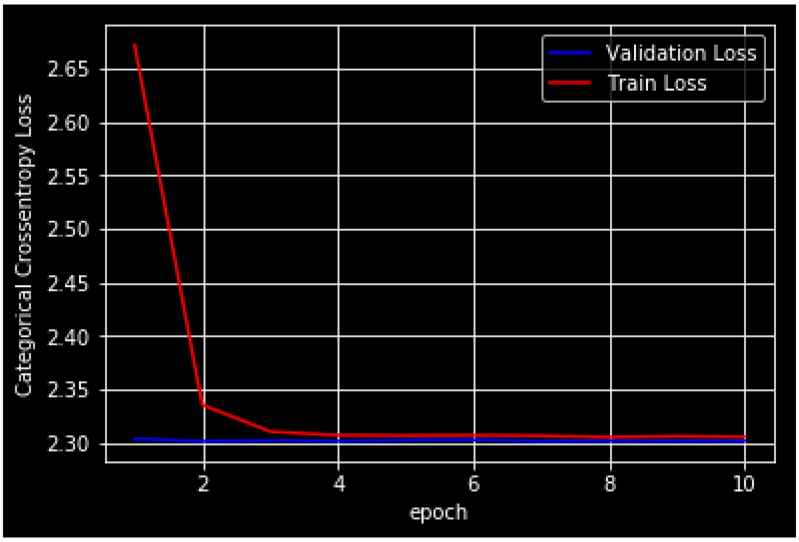
```
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 29s 477us/step - loss: 2.6718
- acc: 0.0996 - val_loss: 2.3042 - val_acc: 0.1135
Epoch 2/10
60000/60000 [==============================] - 19s 318us/step - loss: 2.3356
- acc: 0.1038 - val_loss: 2.3017 - val_acc: 0.1135
Epoch 3/10
60000/60000 [==============================] - 19s 321us/step - loss: 2.3108
- acc: 0.1067 - val_loss: 2.3025 - val_acc: 0.1135
Epoch 4/10
60000/60000 [==============================] - 19s 321us/step - loss: 2.3074
- acc: 0.1067 - val_loss: 2.3019 - val_acc: 0.1135
Epoch 5/10
60000/60000 [==============================] - 19s 320us/step - loss: 2.3071
- acc: 0.1086 - val_loss: 2.3027 - val_acc: 0.1135
Epoch 6/10
60000/60000 [==============================] - 19s 320us/step - loss: 2.3073
- acc: 0.1057 - val_loss: 2.3031 - val_acc: 0.1135
Epoch 7/10
60000/60000 [==============================] - 19s 322us/step - loss: 2.3069
- acc: 0.1084 - val_loss: 2.3018 - val_acc: 0.1135
Epoch 8/10
60000/60000 [==============================] - 19s 324us/step - loss: 2.3055
- acc: 0.1082 - val_loss: 2.3021 - val_acc: 0.1135
Epoch 9/10
60000/60000 [==============================] - 22s 369us/step - loss: 2.3064
- acc: 0.1053 - val_loss: 2.3022 - val_acc: 0.1135
Epoch 10/10
60000/60000 [==============================] - 20s 337us/step - loss: 2.3058
- acc: 0.1056 - val_loss: 2.3020 - val_acc: 0.1135
```

In [19]:

```
score = model_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

del(model_3)
```

```
Test score: 2.302019982147217
Test accuracy: 0.1135
```



7.2.4 MLP (4 hidden layers) + Batch-Norm + Dropout + AdamOptimizer

In [20]:

```
model_4 = Sequential()

model_4.add(Dense(100, activation='relu', input_shape=(input_dim,), kernel_initializer=
model_4.add(BatchNormalization())
model_4.add(Dropout(0.5))

model_4.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_4.add(BatchNormalization())
model_4.add(Dropout(0.5))

model_4.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_4.add(BatchNormalization())
model_4.add(Dropout(0.5))

model_4.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model_4.add(BatchNormalization())
model_4.add(Dropout(0.5))

model_4.add(Dense(output_dim, activation='softmax'))

model_4.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_34 (Dense)             (None, 100)               78500     
_________________________________________________________________
batch_normalization_31 (Batc (None, 100)               400       
_________________________________________________________________
dropout_31 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 100)               10100     
_________________________________________________________________
batch_normalization_32 (Batc (None, 100)               400       
_________________________________________________________________
dropout_32 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_36 (Dense)             (None, 100)               10100     
_________________________________________________________________
batch_normalization_33 (Batc (None, 100)               400       
_________________________________________________________________
dropout_33 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_37 (Dense)             (None, 100)               10100     
_________________________________________________________________
batch_normalization_34 (Batc (None, 100)               400       
_________________________________________________________________
dropout_34 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_38 (Dense)             (None, 10)                1010      
=================================================================
Total params: 111,410
Trainable params: 110,610
Non-trainable params: 800
_________________________________________________________________
```

In [21]:

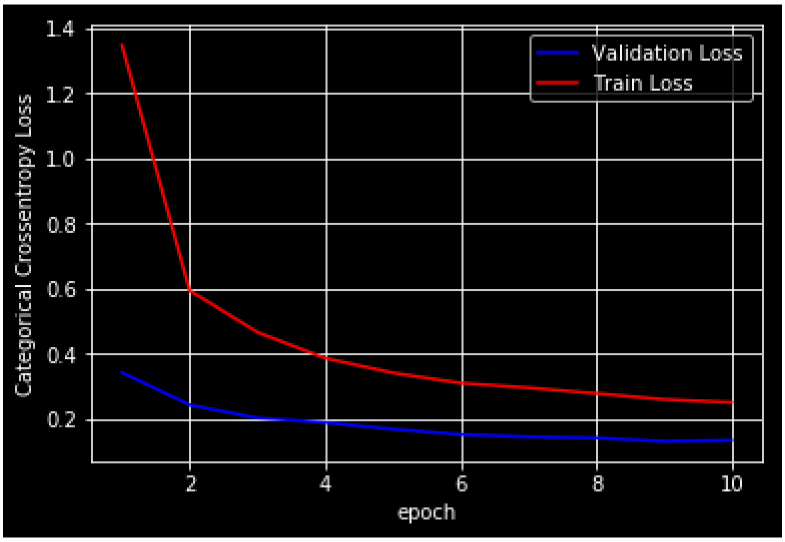
```
model_4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_4.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 8s 132us/step - loss: 1.3473
- acc: 0.5643 - val_loss: 0.3420 - val_acc: 0.9041
Epoch 2/10
60000/60000 [==============================] - 5s 81us/step - loss: 0.5937 -
acc: 0.8200 - val_loss: 0.2426 - val_acc: 0.9299
Epoch 3/10
60000/60000 [==============================] - 5s 82us/step - loss: 0.4655 -
acc: 0.8671 - val_loss: 0.2026 - val_acc: 0.9411
Epoch 4/10
60000/60000 [==============================] - 5s 77us/step - loss: 0.3861 -
acc: 0.8914 - val_loss: 0.1887 - val_acc: 0.9482
Epoch 5/10
60000/60000 [==============================] - 5s 77us/step - loss: 0.3411 -
acc: 0.9063 - val_loss: 0.1684 - val_acc: 0.9521
Epoch 6/10
60000/60000 [==============================] - 5s 76us/step - loss: 0.3097 -
acc: 0.9158 - val_loss: 0.1519 - val_acc: 0.9575
Epoch 7/10
60000/60000 [==============================] - 5s 78us/step - loss: 0.2953 -
acc: 0.9196 - val_loss: 0.1446 - val_acc: 0.9595
Epoch 8/10
60000/60000 [==============================] - 5s 80us/step - loss: 0.2778 -
acc: 0.9250 - val_loss: 0.1413 - val_acc: 0.9611
Epoch 9/10
60000/60000 [==============================] - 5s 78us/step - loss: 0.2596 -
acc: 0.9294 - val_loss: 0.1308 - val_acc: 0.9654
Epoch 10/10
60000/60000 [==============================] - 5s 79us/step - loss: 0.2502 -
acc: 0.9319 - val_loss: 0.1341 - val_acc: 0.9650
```

In [22]:

```
score = model_4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

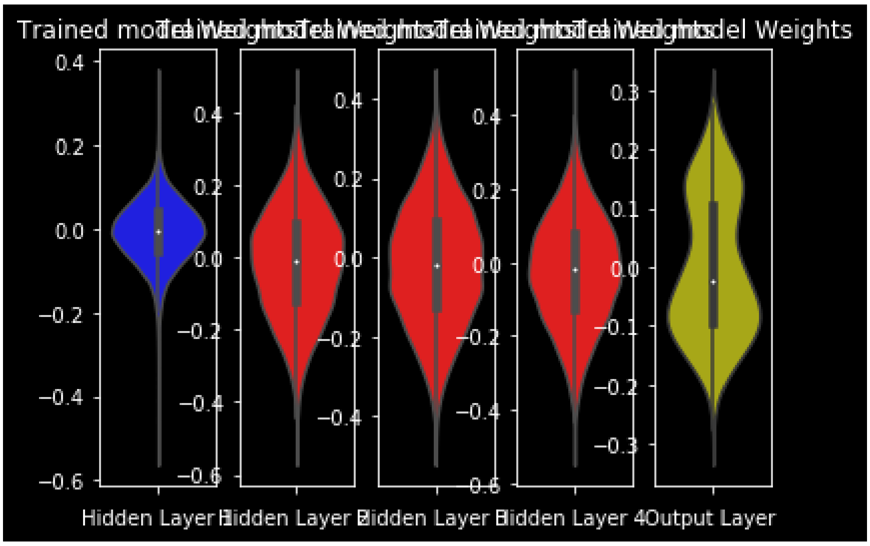
```
Test score: 0.13409332928974182
Test accuracy: 0.965
```



In [24]:

```
h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
h4_w = w_after[18].flatten().reshape(-1,1)
out_w = w_after[24].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 5, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 5, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 5, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 5, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 5, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()

del(model_4)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



7.2.5 MLP (1 hidden layer) + Batch-Norm + Dropout + AdamOptimizer

In [25]:

```
model_5 = Sequential()

model_5.add(Dense(1000, activation='relu', input_shape=(input_dim,), kernel_initialize
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(output_dim, activation='softmax'))

model_5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_39 (Dense)             (None, 1000)              785000
_________________________________________________________________
batch_normalization_35 (Batc (None, 1000)              4000
_________________________________________________________________
dropout_35 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_40 (Dense)             (None, 10)                10010
=================================================================
Total params: 799,010
Trainable params: 797,010
Non-trainable params: 2,000
_________________________________________________________________
```

In [26]:

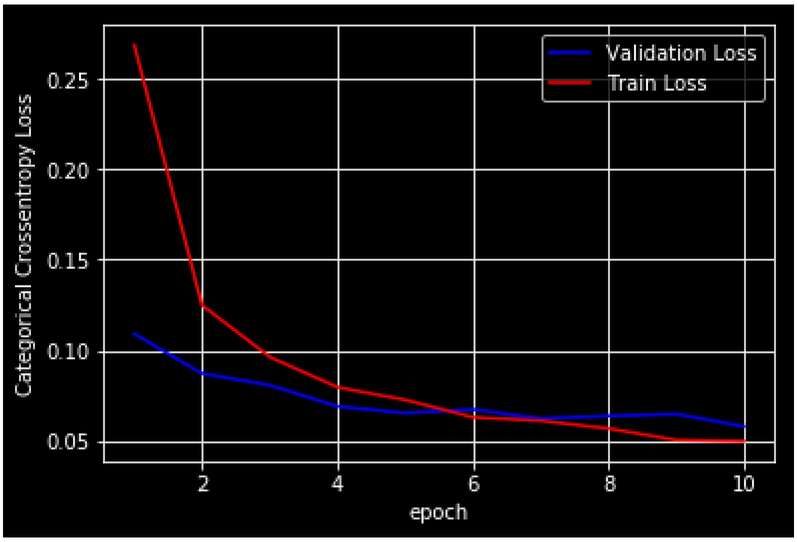
```
model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 12s 206us/step - loss: 0.2689
- acc: 0.9215 - val_loss: 0.1091 - val_acc: 0.9662
Epoch 2/10
60000/60000 [==============================] - 10s 164us/step - loss: 0.1248
- acc: 0.9616 - val_loss: 0.0872 - val_acc: 0.9743
Epoch 3/10
60000/60000 [==============================] - 10s 165us/step - loss: 0.0962
- acc: 0.9700 - val_loss: 0.0806 - val_acc: 0.9748
Epoch 4/10
60000/60000 [==============================] - 10s 166us/step - loss: 0.0795
- acc: 0.9747 - val_loss: 0.0689 - val_acc: 0.9783
Epoch 5/10
60000/60000 [==============================] - 10s 169us/step - loss: 0.0725
- acc: 0.9763 - val_loss: 0.0653 - val_acc: 0.9806
Epoch 6/10
60000/60000 [==============================] - 10s 167us/step - loss: 0.0629
- acc: 0.9800 - val_loss: 0.0673 - val_acc: 0.9797
Epoch 7/10
60000/60000 [==============================] - 10s 172us/step - loss: 0.0610
- acc: 0.9798 - val_loss: 0.0622 - val_acc: 0.9808
Epoch 8/10
60000/60000 [==============================] - 10s 172us/step - loss: 0.0567
- acc: 0.9806 - val_loss: 0.0637 - val_acc: 0.9819
Epoch 9/10
60000/60000 [==============================] - 10s 175us/step - loss: 0.0503
- acc: 0.9831 - val_loss: 0.0647 - val_acc: 0.9816
Epoch 10/10
60000/60000 [==============================] - 10s 175us/step - loss: 0.0496
- acc: 0.9836 - val_loss: 0.0579 - val_acc: 0.9822
```

In [27]:

```
score = model_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

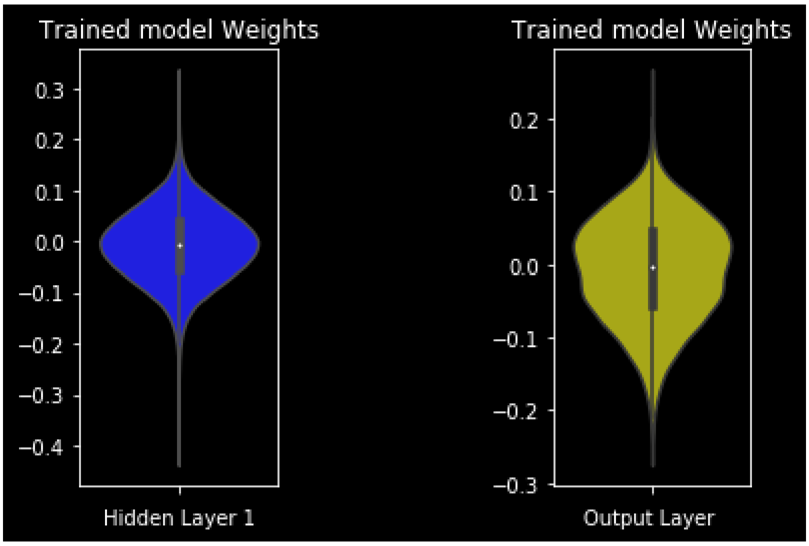
```
Test score: 0.05786827448041295
Test accuracy: 0.9822
```



In [29]:

```
w_after = model_5.get_weights()
h1_w = w_after[0].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, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

del(model_5)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



7.2.6 MLP (1 hidden layer) + Batch-Norm + Dropout (low) + AdamOptimizer

In [30]:

```
model_6 = Sequential()

model_6.add(Dense(300, activation='relu', input_shape=(input_dim,), kernel_initializer
model_6.add(BatchNormalization())
model_6.add(Dropout(0.1))

model_6.add(Dense(output_dim, activation='softmax'))

model_6.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_41 (Dense)             (None, 300)               235500    
_________________________________________________________________
batch_normalization_36 (Batc (None, 300)               1200      
_________________________________________________________________
dropout_36 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 10)                3010      
=================================================================
Total params: 239,710
Trainable params: 239,110
Non-trainable params: 600
_________________________________________________________________
```

In [31]:

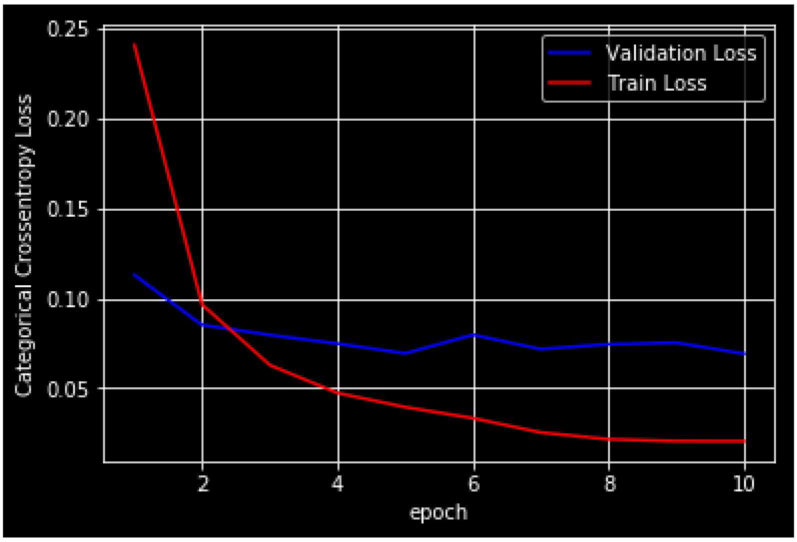
```
model_6.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_6.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 98us/step - loss: 0.2407 -
acc: 0.9280 - val_loss: 0.1131 - val_acc: 0.9678
Epoch 2/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0964 -
acc: 0.9715 - val_loss: 0.0853 - val_acc: 0.9743
Epoch 3/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0629 -
acc: 0.9813 - val_loss: 0.0797 - val_acc: 0.9759
Epoch 4/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0474 -
acc: 0.9863 - val_loss: 0.0749 - val_acc: 0.9767
Epoch 5/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0396 -
acc: 0.9877 - val_loss: 0.0695 - val_acc: 0.9793
Epoch 6/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0334 -
acc: 0.9897 - val_loss: 0.0798 - val_acc: 0.9771
Epoch 7/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0255 -
acc: 0.9922 - val_loss: 0.0719 - val_acc: 0.9782
Epoch 8/10
60000/60000 [==============================] - 4s 67us/step - loss: 0.0217 -
acc: 0.9933 - val_loss: 0.0746 - val_acc: 0.9768
Epoch 9/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.0207 -
acc: 0.9934 - val_loss: 0.0754 - val_acc: 0.9790
Epoch 10/10
60000/60000 [==============================] - 4s 67us/step - loss: 0.0206 -
acc: 0.9934 - val_loss: 0.0693 - val_acc: 0.9800
```

In [32]:

```
score = model_6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

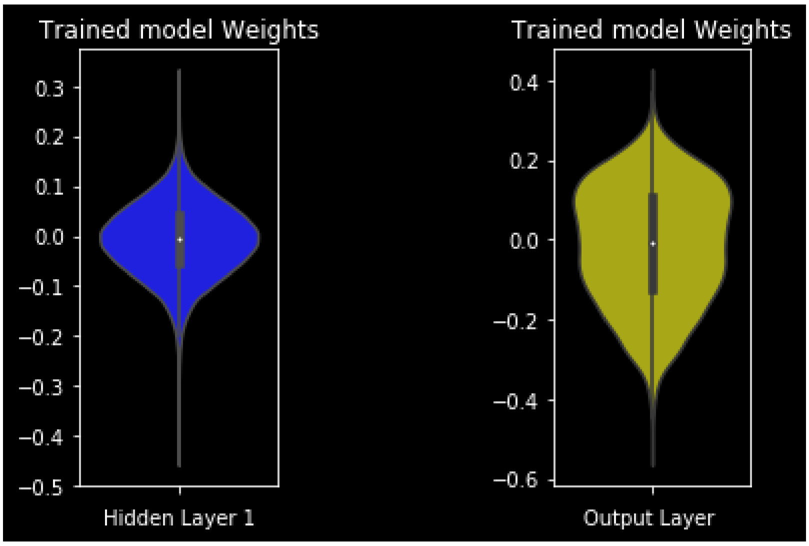
```
Test score: 0.06930483953608782
Test accuracy: 0.98
```



In [34]:

```
w_after = model_6.get_weights()
h1_w = w_after[0].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, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

del(model_6)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



7.2.7 MLP (1 hidden layer) + Batch-Norm + Dropout (high) + AdamOptimizer

In [35]:

```
model_7 = Sequential()

model_7.add(Dense(300, activation='relu', input_shape=(input_dim,), kernel_initializer
model_7.add(BatchNormalization())
model_7.add(Dropout(0.9))

model_7.add(Dense(output_dim, activation='softmax'))

model_7.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_43 (Dense)             (None, 300)               235500    
_________________________________________________________________
batch_normalization_37 (Batc (None, 300)               1200      
_________________________________________________________________
dropout_37 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_44 (Dense)             (None, 10)                3010      
=================================================================
Total params: 239,710
Trainable params: 239,110
Non-trainable params: 600
_________________________________________________________________
```

In [36]:

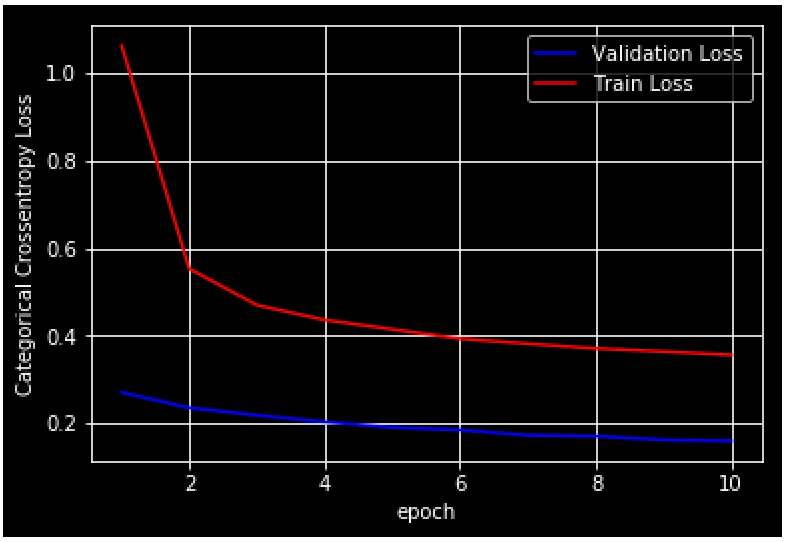
```
model_7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

history = model_7.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 104us/step - loss: 1.0630
- acc: 0.7194 - val_loss: 0.2704 - val_acc: 0.9217
Epoch 2/10
60000/60000 [==============================] - 4s 68us/step - loss: 0.5531 -
acc: 0.8324 - val_loss: 0.2353 - val_acc: 0.9301
Epoch 3/10
60000/60000 [==============================] - 4s 68us/step - loss: 0.4699 -
acc: 0.8575 - val_loss: 0.2184 - val_acc: 0.9344
Epoch 4/10
60000/60000 [==============================] - 4s 68us/step - loss: 0.4361 -
acc: 0.8693 - val_loss: 0.2031 - val_acc: 0.9395
Epoch 5/10
60000/60000 [==============================] - 4s 69us/step - loss: 0.4146 -
acc: 0.8765 - val_loss: 0.1903 - val_acc: 0.9399
Epoch 6/10
60000/60000 [==============================] - 4s 69us/step - loss: 0.3932 -
acc: 0.8820 - val_loss: 0.1848 - val_acc: 0.9442
Epoch 7/10
60000/60000 [==============================] - 4s 69us/step - loss: 0.3817 -
acc: 0.8842 - val_loss: 0.1725 - val_acc: 0.9470
Epoch 8/10
60000/60000 [==============================] - 4s 70us/step - loss: 0.3709 -
acc: 0.8880 - val_loss: 0.1705 - val_acc: 0.9483
Epoch 9/10
60000/60000 [==============================] - 4s 69us/step - loss: 0.3637 -
acc: 0.8915 - val_loss: 0.1619 - val_acc: 0.9510
Epoch 10/10
60000/60000 [==============================] - 4s 69us/step - loss: 0.3562 -
acc: 0.8944 - val_loss: 0.1598 - val_acc: 0.9525
```

In [37]:

```python
score = model_7.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, ve

# 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 ep

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

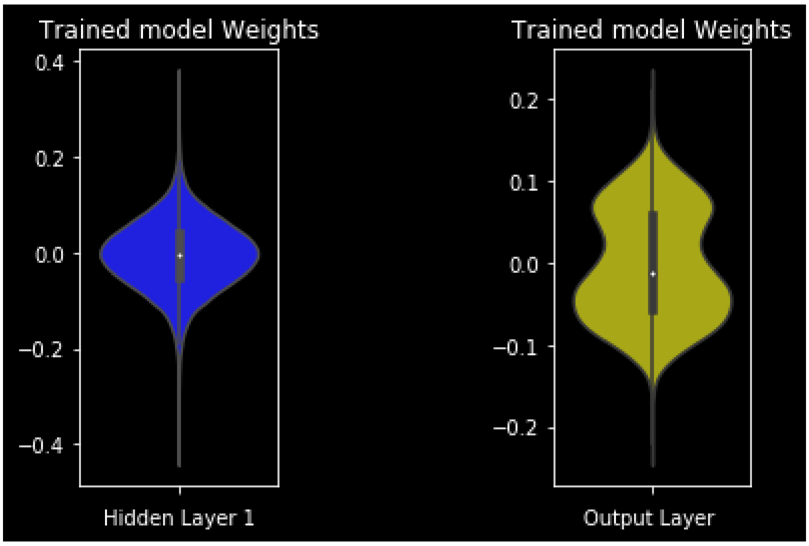
```
Test score: 0.15979730788581073
Test accuracy: 0.9525
```



In [39]:

```
w_after = model_7.get_weights()
h1_w = w_after[0].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, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

del(model_7)
```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\scipy\stats\stat
s.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional in
dexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the fu
ture this will be interpreted as an array index, `arr[np.array(seq)]`, which
will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Type *Markdown* and LaTeX: α^2

In [2]:

```
import prettytable

table = prettytable.PrettyTable()
table.add_row(['Model','Parameters','Training loss','Validation loss','Accuracy','epoc
table.add_row(['2 HL,relu,adam,dropout,batch norm','471434','0.08','0.06','98','10'])
table.add_row(['3 HL,relu,adam,dropout,batch norm','577520','0.09','0.07','97','10'])
table.add_row(['25 HL,relu,adam,dropout,batch norm,100 neurons','420810','2.3','2.3','
table.add_row(['4 HL,relu,adam,dropout,batch norm,100 neurons','111410','0.17','0.09',
table.add_row(['1 HL,relu,adam,dropout,batch norm,1000 neurons','799010','0.05','0.06'
table.add_row(['1 HL,relu,adam,batch norm,low dropout,300 neurons','239710','0.01','0.
table.add_row(['1 HL,relu,adam,batch norm,high dropout,300 neurons','239710','0.36','0

print(table)
```

```
+-----------------------------------------------------+------------+---------
------+-----------------+----------+----------+
|                      Field 1                        |  Field 2   |    Field
3     |     Field 4     | Field 5  | Field 6  |
+-----------------------------------------------------+------------+---------
------+-----------------+----------+----------+
|                       Model                         | Parameters | Training
loss | Validation loss | Accuracy |  epochs  |
|          2 HL,relu,adam,dropout,batch norm          |   471434   |      0.0
8     |       0.06      |    98    |    10    |
|          3 HL,relu,adam,dropout,batch norm          |   577520   |      0.0
9     |       0.07      |    97    |    10    |
|    25 HL,relu,adam,dropout,batch norm,100 neurons   |   420810   |      2.3
|       2.3       |    11    |    10    |
|    4 HL,relu,adam,dropout,batch norm,100 neurons    |   111410   |      0.1
7     |       0.09      |    97    |    10    |
|    1 HL,relu,adam,dropout,batch norm,1000 neurons   |   799010   |      0.0
5     |       0.06      |    98    |    10    |
|  1 HL,relu,adam,batch norm,low dropout,300 neurons  |   239710   |      0.0
1     |       0.06      |    98    |    10    |
| 1 HL,relu,adam,batch norm,high dropout,300 neurons  |   239710   |      0.3
6     |       0.16      |    95    |    10    |
+-----------------------------------------------------+------------+---------
------+-----------------+----------+----------+
```