Keras -- MLPs on MNIST

In [71]:

```
from keras.datasets import mnist
import tensorflow as tf
from keras import backend
import seaborn as sns
import numpy as np
from keras.initializers import RandomNormal
```

In [72]:

```
%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 [73]:

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

In [74]:

```
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)
```

In [75]:

```
# 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 [76]:

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

In [77]:

```
# 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 [78]:

```
# 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 [79]:

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

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

In [80]:

```
# here we are having a class number for each image
from keras import utils as np_utils
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.]
```

Adam classifier

In [81]:

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

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the activatio

# from keras.layers import Activation, Dense

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

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

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


from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization

```

In [82]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 50
```

MLP + Relu activation + Adam + 2 layers

In [106]:

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


model2 = Sequential()

model2.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

model2.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stdd
model2.add(BatchNormalization())
model2.add(Dropout(0.2))

model2.add(Dense(output_dim, activation='softmax'))


model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_48 (Dense)             (None, 128)               100480    
_________________________________________________________________
batch_normalization_35 (Batc (None, 128)               512       
_________________________________________________________________
dropout_35 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 32)                4128      
_________________________________________________________________
batch_normalization_36 (Batc (None, 32)                128       
_________________________________________________________________
dropout_36 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_50 (Dense)             (None, 10)                330       
=================================================================
Total params: 105,578
Trainable params: 105,258
Non-trainable params: 320
_________________________________________________________________
```

In [107]:

```
model2.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 7s 117us/step - loss: 1.0096
- acc: 0.6958 - val_loss: 0.4400 - val_acc: 0.8911
Epoch 2/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.5755 -
acc: 0.8389 - val_loss: 0.3203 - val_acc: 0.9180
Epoch 3/50
60000/60000 [==============================] - 4s 72us/step - loss: 0.4760 -
acc: 0.8664 - val_loss: 0.2707 - val_acc: 0.9260
Epoch 4/50
60000/60000 [==============================] - 7s 111us/step - loss: 0.4207
- acc: 0.8807 - val_loss: 0.2375 - val_acc: 0.9330
Epoch 5/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.3776 -
acc: 0.8949 - val_loss: 0.2209 - val_acc: 0.9358
Epoch 6/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.3584 -
acc: 0.8974 - val_loss: 0.2041 - val_acc: 0.9408
Epoch 7/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.3345 -
acc: 0.9043 - val_loss: 0.1899 - val_acc: 0.9429
Epoch 8/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.3183 -
acc: 0.9078 - val_loss: 0.1803 - val_acc: 0.9473
Epoch 9/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.3034 -
acc: 0.9123 - val_loss: 0.1712 - val_acc: 0.9497
Epoch 10/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2945 -
acc: 0.9150 - val_loss: 0.1625 - val_acc: 0.9529
Epoch 11/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.2842 -
acc: 0.9181 - val_loss: 0.1591 - val_acc: 0.9552
Epoch 12/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2707 -
acc: 0.9211 - val_loss: 0.1535 - val_acc: 0.9546
Epoch 13/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2617 -
acc: 0.9249 - val_loss: 0.1484 - val_acc: 0.9549
Epoch 14/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.2602 -
acc: 0.9244 - val_loss: 0.1436 - val_acc: 0.9573
Epoch 15/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2479 -
acc: 0.9283 - val_loss: 0.1402 - val_acc: 0.9588
Epoch 16/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.2432 -
acc: 0.9299 - val_loss: 0.1381 - val_acc: 0.9579
Epoch 17/50
60000/60000 [==============================] - 6s 99us/step - loss: 0.2379 -
acc: 0.9310 - val_loss: 0.1349 - val_acc: 0.9601
Epoch 18/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2346 -
```

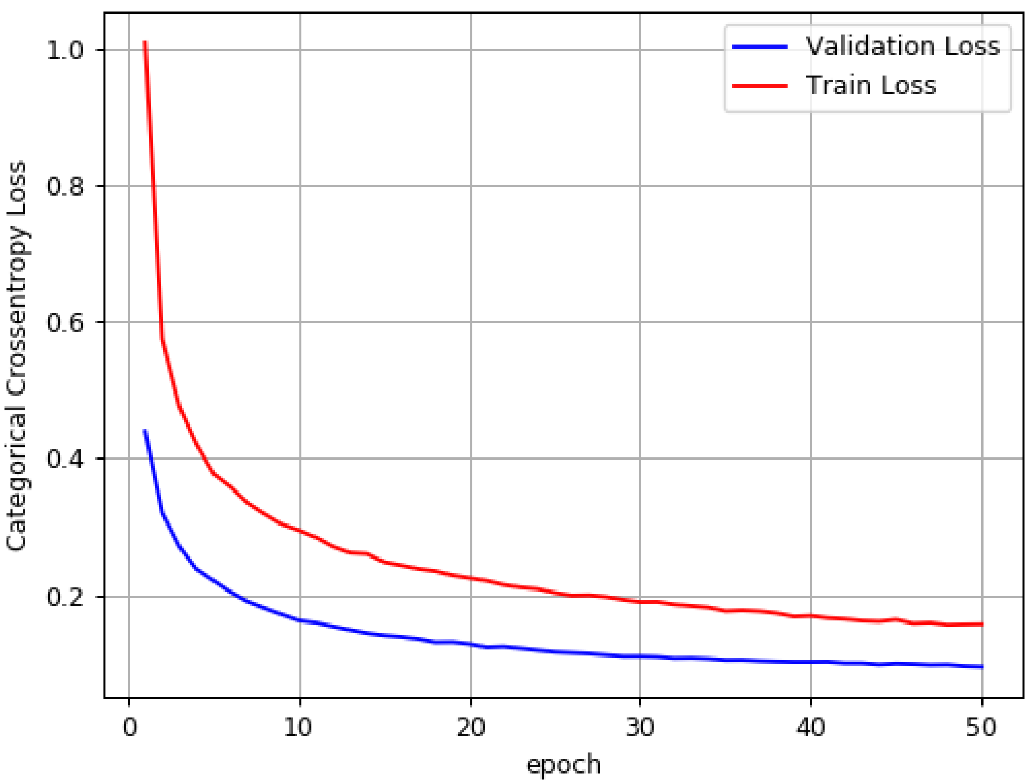
```
acc: 0.9310 - val_loss: 0.1302 - val_acc: 0.9606
Epoch 19/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2286 -
acc: 0.9332 - val_loss: 0.1305 - val_acc: 0.9608
Epoch 20/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2246 -
acc: 0.9342 - val_loss: 0.1278 - val_acc: 0.9615
Epoch 21/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.2207 -
acc: 0.9352 - val_loss: 0.1229 - val_acc: 0.9630
Epoch 22/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2149 -
acc: 0.9379 - val_loss: 0.1239 - val_acc: 0.9624
Epoch 23/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2110 -
acc: 0.9390 - val_loss: 0.1213 - val_acc: 0.9635
Epoch 24/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2087 -
acc: 0.9383 - val_loss: 0.1188 - val_acc: 0.9640
Epoch 25/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.2028 -
acc: 0.9408 - val_loss: 0.1162 - val_acc: 0.9653
Epoch 26/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1988 -
acc: 0.9419 - val_loss: 0.1151 - val_acc: 0.9657
Epoch 27/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.1993
- acc: 0.9415 - val_loss: 0.1139 - val_acc: 0.9654
Epoch 28/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1969 -
acc: 0.9418 - val_loss: 0.1119 - val_acc: 0.9668
Epoch 29/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1926 -
acc: 0.9430 - val_loss: 0.1098 - val_acc: 0.9667
Epoch 30/50
60000/60000 [==============================] - 6s 98us/step - loss: 0.1894 -
acc: 0.9440 - val_loss: 0.1099 - val_acc: 0.9679
Epoch 31/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1896 -
acc: 0.9438 - val_loss: 0.1091 - val_acc: 0.9678
Epoch 32/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1857 -
acc: 0.9451 - val_loss: 0.1069 - val_acc: 0.9696
Epoch 33/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1836 -
acc: 0.9458 - val_loss: 0.1073 - val_acc: 0.9682
Epoch 34/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.1814
- acc: 0.9475 - val_loss: 0.1063 - val_acc: 0.9679
Epoch 35/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1762 -
acc: 0.9480 - val_loss: 0.1042 - val_acc: 0.9683
Epoch 36/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1773 -
acc: 0.9472 - val_loss: 0.1043 - val_acc: 0.9699
Epoch 37/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1760 -
acc: 0.9477 - val_loss: 0.1029 - val_acc: 0.9696
Epoch 38/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.1731 -
acc: 0.9499 - val_loss: 0.1021 - val_acc: 0.9703
```

```
Epoch 39/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.1682 -
acc: 0.9494 - val_loss: 0.1016 - val_acc: 0.9700
Epoch 40/50
60000/60000 [==============================] - 6s 94us/step - loss: 0.1693 -
acc: 0.9505 - val_loss: 0.1016 - val_acc: 0.9704
Epoch 41/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1661 -
acc: 0.9515 - val_loss: 0.1018 - val_acc: 0.9692
Epoch 42/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1647 -
acc: 0.9507 - val_loss: 0.0998 - val_acc: 0.9705
Epoch 43/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1621 -
acc: 0.9522 - val_loss: 0.0997 - val_acc: 0.9706
Epoch 44/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1611 -
acc: 0.9536 - val_loss: 0.0978 - val_acc: 0.9716
Epoch 45/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1638 -
acc: 0.9514 - val_loss: 0.0991 - val_acc: 0.9712
Epoch 46/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.1581 -
acc: 0.9534 - val_loss: 0.0985 - val_acc: 0.9714
Epoch 47/50
60000/60000 [==============================] - 6s 96us/step - loss: 0.1591 -
acc: 0.9526 - val_loss: 0.0973 - val_acc: 0.9716
Epoch 48/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.1559 -
acc: 0.9528 - val_loss: 0.0976 - val_acc: 0.9711
Epoch 49/50
60000/60000 [==============================] - 6s 95us/step - loss: 0.1565 -
acc: 0.9530 - val_loss: 0.0954 - val_acc: 0.9713
Epoch 50/50
60000/60000 [==============================] - 6s 97us/step - loss: 0.1568 -
acc: 0.9526 - val_loss: 0.0946 - val_acc: 0.9726
```

In [108]:

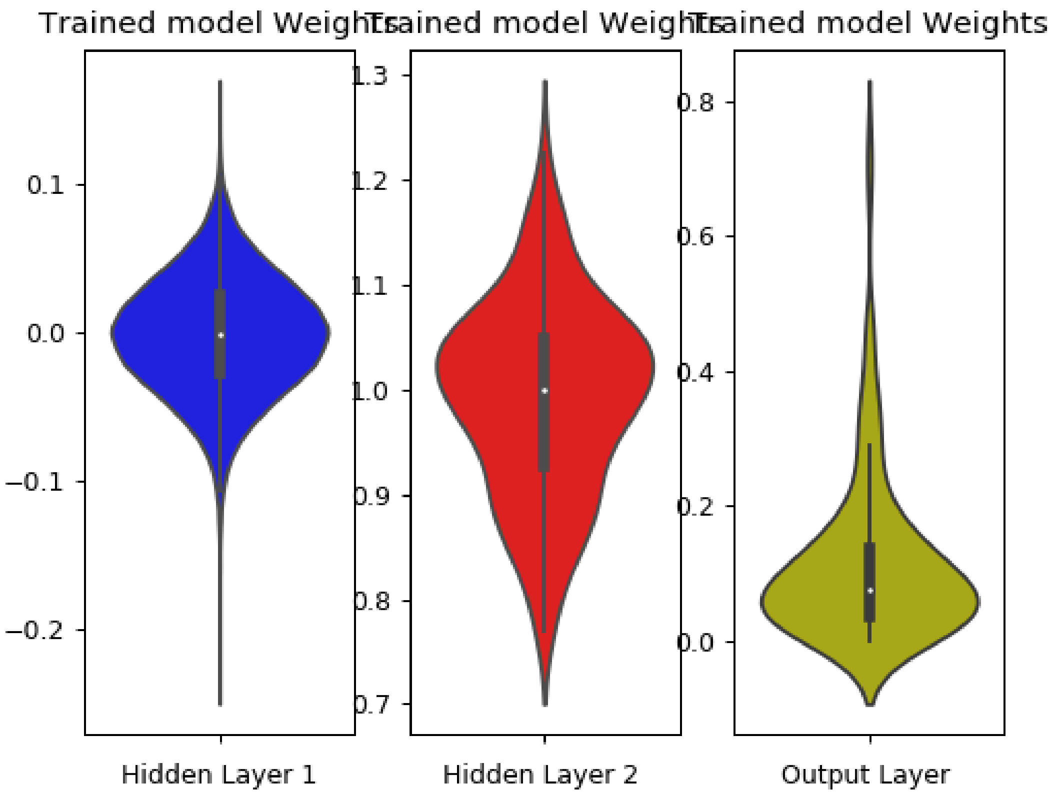
```
score = model2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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: 0.09460396405863576
Test accuracy: 0.9726
```



In [109]:

```
w_after = model2.get_weights()

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


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

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

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



MLP + Relu activation + Adam + 3 layers

In [102]:

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


model1 = Sequential()

model1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=
model1.add(BatchNormalization())
model1.add(Dropout(0.7))

model1.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

model1.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stdd
model1.add(BatchNormalization())
model1.add(Dropout(0.2))

model1.add(Dense(output_dim, activation='softmax'))


model1.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropou
t() uses dropout rate instead of keep_prob. Please ensure that this is inten
ded.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_44 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_32 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_32 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_33 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_33 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_46 (Dense)             (None, 32)                8224      
_________________________________________________________________
batch_normalization_34 (Batc (None, 32)                128       
_________________________________________________________________
dropout_34 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_47 (Dense)             (None, 10)                330       
=================================================================
Total params: 545,002
Trainable params: 543,402
Non-trainable params: 1,600
_________________________________________________________________
```

In [103]:

```
model1.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 9s 154us/step - loss: 1.5934
- acc: 0.4778 - val_loss: 0.7093 - val_acc: 0.8354
Epoch 2/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.9885
- acc: 0.6906 - val_loss: 0.5044 - val_acc: 0.8758
Epoch 3/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.8015
- acc: 0.7528 - val_loss: 0.4044 - val_acc: 0.8948
Epoch 4/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.7103
- acc: 0.7832 - val_loss: 0.3492 - val_acc: 0.9040
Epoch 5/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.6460
- acc: 0.8008 - val_loss: 0.3120 - val_acc: 0.9120
Epoch 6/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.5992
- acc: 0.8148 - val_loss: 0.2890 - val_acc: 0.9174
Epoch 7/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.5651
- acc: 0.8265 - val_loss: 0.2710 - val_acc: 0.9223
Epoch 8/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.5347
- acc: 0.8352 - val_loss: 0.2541 - val_acc: 0.9253
Epoch 9/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.5192
- acc: 0.8385 - val_loss: 0.2457 - val_acc: 0.9269
Epoch 10/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.5020
- acc: 0.8462 - val_loss: 0.2358 - val_acc: 0.9294
Epoch 11/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.4801
- acc: 0.8541 - val_loss: 0.2262 - val_acc: 0.9330
Epoch 12/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.4693
- acc: 0.8577 - val_loss: 0.2201 - val_acc: 0.9333
Epoch 13/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.4620
- acc: 0.8586 - val_loss: 0.2140 - val_acc: 0.9374
Epoch 14/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.4427
- acc: 0.8672 - val_loss: 0.2089 - val_acc: 0.9374
Epoch 15/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.4374
- acc: 0.8659 - val_loss: 0.2046 - val_acc: 0.9384
Epoch 16/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.4266
- acc: 0.8704 - val_loss: 0.1991 - val_acc: 0.9395
Epoch 17/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.4191
- acc: 0.8717 - val_loss: 0.1946 - val_acc: 0.9424
Epoch 18/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.4073
- acc: 0.8777 - val_loss: 0.1909 - val_acc: 0.9431
```

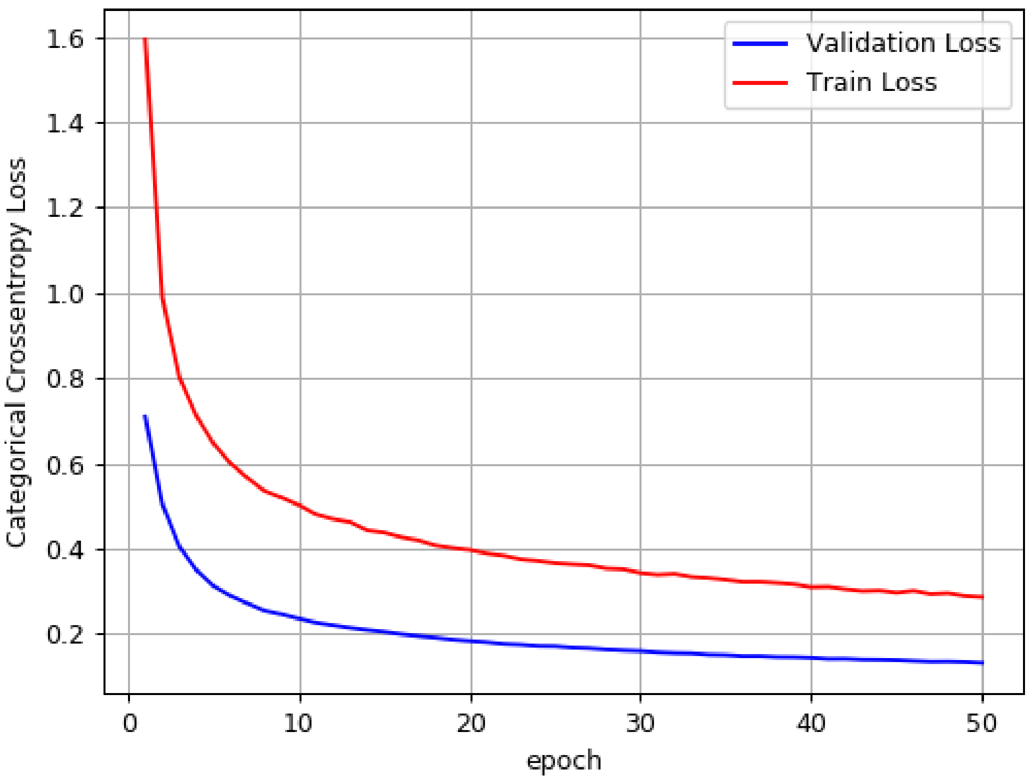
```
Epoch 19/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.4015
- acc: 0.8799 - val_loss: 0.1864 - val_acc: 0.9441
Epoch 20/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3969
- acc: 0.8815 - val_loss: 0.1833 - val_acc: 0.9446
Epoch 21/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3886
- acc: 0.8838 - val_loss: 0.1803 - val_acc: 0.9462
Epoch 22/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3832
- acc: 0.8851 - val_loss: 0.1763 - val_acc: 0.9470
Epoch 23/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.3749
- acc: 0.8862 - val_loss: 0.1746 - val_acc: 0.9465
Epoch 24/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3710
- acc: 0.8874 - val_loss: 0.1714 - val_acc: 0.9476
Epoch 25/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3658
- acc: 0.8898 - val_loss: 0.1709 - val_acc: 0.9482
Epoch 26/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3631
- acc: 0.8907 - val_loss: 0.1676 - val_acc: 0.9495
Epoch 27/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3609
- acc: 0.8906 - val_loss: 0.1661 - val_acc: 0.9497
Epoch 28/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3533
- acc: 0.8941 - val_loss: 0.1631 - val_acc: 0.9512
Epoch 29/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3515
- acc: 0.8934 - val_loss: 0.1613 - val_acc: 0.9515
Epoch 30/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3428
- acc: 0.8972 - val_loss: 0.1598 - val_acc: 0.9513
Epoch 31/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3387
- acc: 0.8973 - val_loss: 0.1567 - val_acc: 0.9528
Epoch 32/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.3408
- acc: 0.8973 - val_loss: 0.1550 - val_acc: 0.9529
Epoch 33/50
60000/60000 [==============================] - 7s 121us/step - loss: 0.3336
- acc: 0.8990 - val_loss: 0.1542 - val_acc: 0.9529
Epoch 34/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3310
- acc: 0.9017 - val_loss: 0.1512 - val_acc: 0.9536
Epoch 35/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3269
- acc: 0.9015 - val_loss: 0.1504 - val_acc: 0.9539
Epoch 36/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3218
- acc: 0.9037 - val_loss: 0.1477 - val_acc: 0.9546
Epoch 37/50
60000/60000 [==============================] - 8s 126us/step - loss: 0.3218
- acc: 0.9039 - val_loss: 0.1478 - val_acc: 0.9548
Epoch 38/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3194
- acc: 0.9042 - val_loss: 0.1458 - val_acc: 0.9556
Epoch 39/50
```

```
60000/60000 [==============================] - 7s 122us/step - loss: 0.3163
- acc: 0.9053 - val_loss: 0.1455 - val_acc: 0.9556
Epoch 40/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3094
- acc: 0.9076 - val_loss: 0.1441 - val_acc: 0.9562
Epoch 41/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3102
- acc: 0.9072 - val_loss: 0.1414 - val_acc: 0.9559
Epoch 42/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3047
- acc: 0.9085 - val_loss: 0.1416 - val_acc: 0.9560
Epoch 43/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.3006
- acc: 0.9101 - val_loss: 0.1397 - val_acc: 0.9571
Epoch 44/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3017
- acc: 0.9098 - val_loss: 0.1393 - val_acc: 0.9572
Epoch 45/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.2966
- acc: 0.9112 - val_loss: 0.1384 - val_acc: 0.9578
Epoch 46/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.3011
- acc: 0.9097 - val_loss: 0.1366 - val_acc: 0.9589
Epoch 47/50
60000/60000 [==============================] - 7s 122us/step - loss: 0.2935
- acc: 0.9116 - val_loss: 0.1348 - val_acc: 0.9590
Epoch 48/50
60000/60000 [==============================] - 7s 123us/step - loss: 0.2954
- acc: 0.9124 - val_loss: 0.1351 - val_acc: 0.9591
Epoch 49/50
60000/60000 [==============================] - 8s 131us/step - loss: 0.2888
- acc: 0.9134 - val_loss: 0.1340 - val_acc: 0.9601
Epoch 50/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.2864
- acc: 0.9137 - val_loss: 0.1320 - val_acc: 0.9613
```

In [104]:

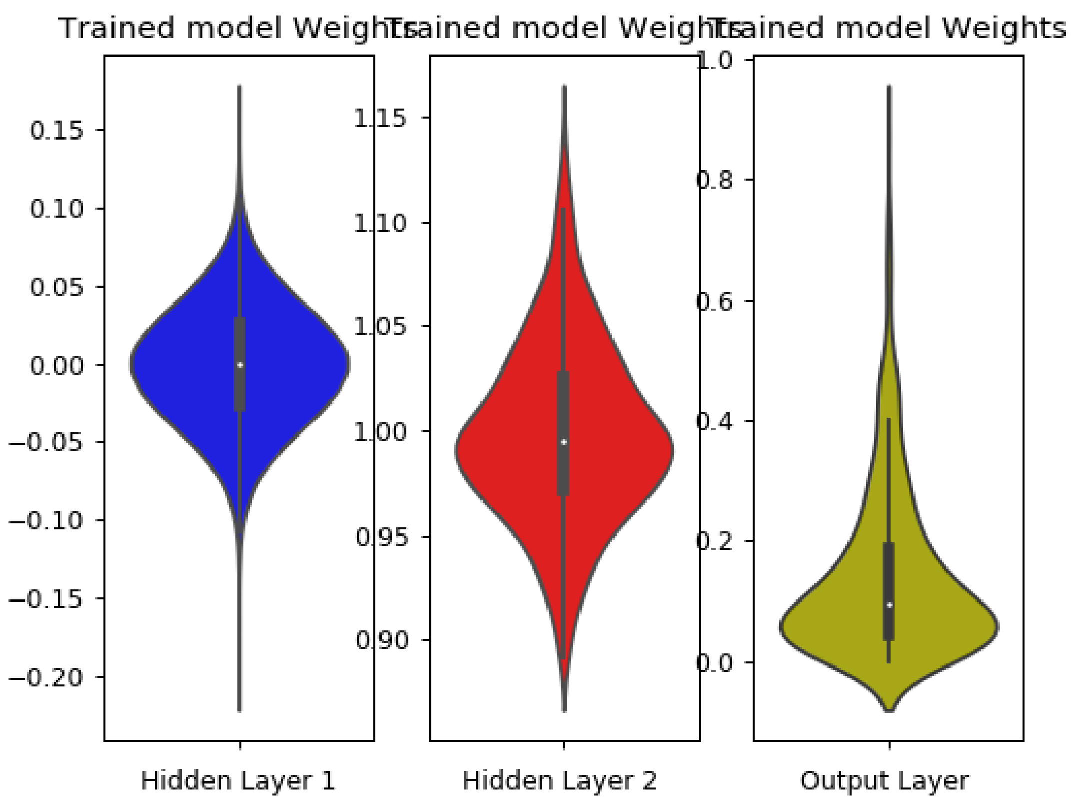
```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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: 0.13201109928507357
Test accuracy: 0.9613
```



In [105]:

```
w_after = model1.get_weights()

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


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

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

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



MLP + Relu activation + Adam + 5 layers

In [91]:

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


model3 = Sequential()

model3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
model3.add(BatchNormalization())
model3.add(Dropout(0.3))

model3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stdd
model3.add(BatchNormalization())
model3.add(Dropout(0.2))

model3.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, std
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

model3.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stdd
model3.add(BatchNormalization())
model3.add(Dropout(0.2))

model3.add(Dense(output_dim, activation='softmax'))


model3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_21 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_31 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_22 (Batc (None, 128)               512       
_________________________________________________________________
dropout_22 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_32 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_23 (Batc (None, 64)                256       
_________________________________________________________________
dropout_23 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_33 (Dense)             (None, 128)               8320      
_________________________________________________________________
batch_normalization_24 (Batc (None, 128)               512       
_________________________________________________________________
dropout_24 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 32)                4128      
```

```
_________________________________________________________________
batch_normalization_25 (Batc (None, 32)                128       
_________________________________________________________________
dropout_25 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_35 (Dense)             (None, 10)                330       
=================================================================
Total params: 492,074
Trainable params: 490,346
Non-trainable params: 1,728
_________________________________________________________________
```

In [92]:

```
model3.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 14s 228us/step - loss: 1.9873
- acc: 0.3325 - val_loss: 1.0039 - val_acc: 0.7655
Epoch 2/50
60000/60000 [==============================] - 8s 131us/step - loss: 1.3469
- acc: 0.5551 - val_loss: 0.7163 - val_acc: 0.8395
Epoch 3/50
60000/60000 [==============================] - 10s 162us/step - loss: 1.1051
- acc: 0.6443 - val_loss: 0.5478 - val_acc: 0.8675
Epoch 4/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.9521
- acc: 0.6964 - val_loss: 0.4420 - val_acc: 0.8865
Epoch 5/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.8474
- acc: 0.7340 - val_loss: 0.3786 - val_acc: 0.8999
Epoch 6/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.7693
- acc: 0.7592 - val_loss: 0.3360 - val_acc: 0.9077
Epoch 7/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.7177
- acc: 0.7770 - val_loss: 0.3047 - val_acc: 0.9161
Epoch 8/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.6677
- acc: 0.7957 - val_loss: 0.2803 - val_acc: 0.9232
Epoch 9/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.6282
- acc: 0.8078 - val_loss: 0.2612 - val_acc: 0.9268
Epoch 10/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.5995
- acc: 0.8176 - val_loss: 0.2474 - val_acc: 0.9298
Epoch 11/50
60000/60000 [==============================] - 10s 164us/step - loss: 0.5748
- acc: 0.8268 - val_loss: 0.2388 - val_acc: 0.9319
Epoch 12/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.5469
- acc: 0.8348 - val_loss: 0.2251 - val_acc: 0.9357
Epoch 13/50
60000/60000 [==============================] - 10s 164us/step - loss: 0.5277
- acc: 0.8425 - val_loss: 0.2190 - val_acc: 0.9362
Epoch 14/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.5097
- acc: 0.8488 - val_loss: 0.2111 - val_acc: 0.9383
Epoch 15/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.4911
- acc: 0.8545 - val_loss: 0.2019 - val_acc: 0.9421
Epoch 16/50
60000/60000 [==============================] - 10s 164us/step - loss: 0.4771
- acc: 0.8572 - val_loss: 0.1988 - val_acc: 0.9424
Epoch 17/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.4644
- acc: 0.8617 - val_loss: 0.1903 - val_acc: 0.9454
Epoch 18/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.4511
- acc: 0.8669 - val_loss: 0.1841 - val_acc: 0.9477
```

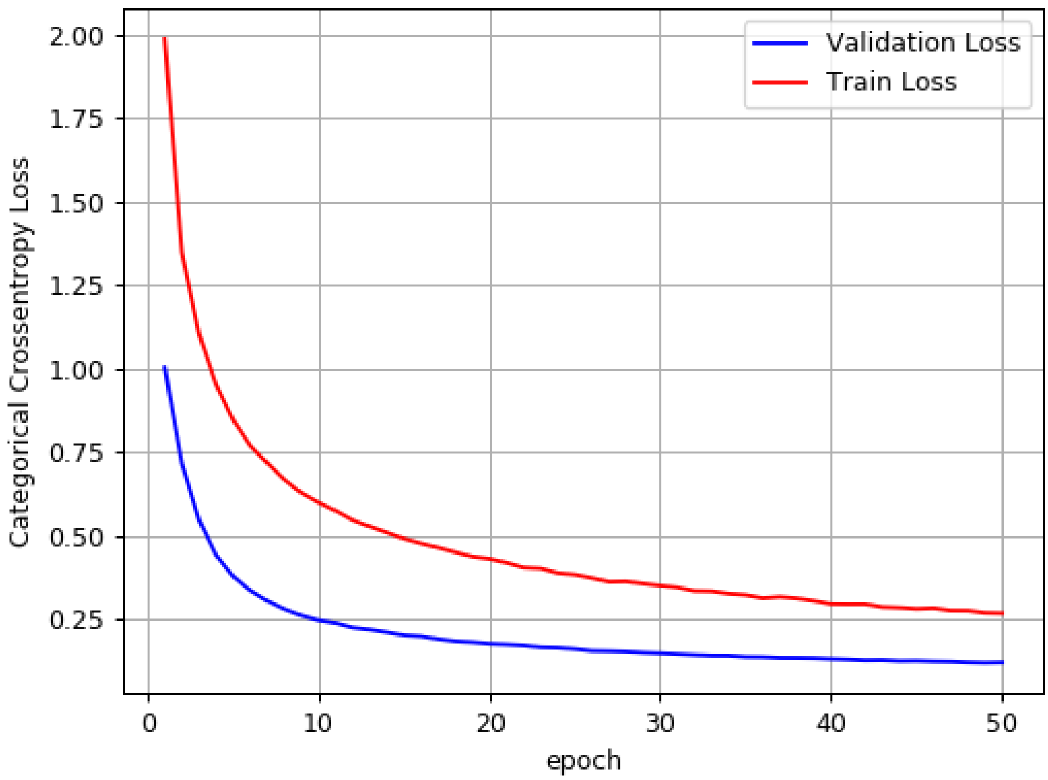
```
Epoch 19/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.4375
- acc: 0.8717 - val_loss: 0.1810 - val_acc: 0.9478
Epoch 20/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.4309
- acc: 0.8735 - val_loss: 0.1766 - val_acc: 0.9492
Epoch 21/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.4192
- acc: 0.8777 - val_loss: 0.1743 - val_acc: 0.9503
Epoch 22/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.4048
- acc: 0.8822 - val_loss: 0.1716 - val_acc: 0.9504
Epoch 23/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.4019
- acc: 0.8827 - val_loss: 0.1662 - val_acc: 0.9519
Epoch 24/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.3884
- acc: 0.8860 - val_loss: 0.1649 - val_acc: 0.9521
Epoch 25/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.3835
- acc: 0.8889 - val_loss: 0.1611 - val_acc: 0.9544
Epoch 26/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.3736
- acc: 0.8914 - val_loss: 0.1557 - val_acc: 0.9563
Epoch 27/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3623
- acc: 0.8948 - val_loss: 0.1547 - val_acc: 0.9553
Epoch 28/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.3635
- acc: 0.8948 - val_loss: 0.1532 - val_acc: 0.9553
Epoch 29/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.3569
- acc: 0.8963 - val_loss: 0.1501 - val_acc: 0.9566
Epoch 30/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3509
- acc: 0.8993 - val_loss: 0.1483 - val_acc: 0.9567
Epoch 31/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3454
- acc: 0.9012 - val_loss: 0.1459 - val_acc: 0.9573
Epoch 32/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3349
- acc: 0.9049 - val_loss: 0.1434 - val_acc: 0.9569
Epoch 33/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3338
- acc: 0.9044 - val_loss: 0.1411 - val_acc: 0.9584
Epoch 34/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3266
- acc: 0.9065 - val_loss: 0.1409 - val_acc: 0.9596
Epoch 35/50
60000/60000 [==============================] - 10s 161us/step - loss: 0.3226
- acc: 0.9085 - val_loss: 0.1373 - val_acc: 0.9598
Epoch 36/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3132
- acc: 0.9098 - val_loss: 0.1368 - val_acc: 0.9601
Epoch 37/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.3172
- acc: 0.9103 - val_loss: 0.1344 - val_acc: 0.9602
Epoch 38/50
60000/60000 [==============================] - 10s 167us/step - loss: 0.3127
- acc: 0.9109 - val_loss: 0.1341 - val_acc: 0.9617
Epoch 39/50
```

```
60000/60000 [==============================] - 10s 163us/step - loss: 0.3049
- acc: 0.9142 - val_loss: 0.1334 - val_acc: 0.9603
Epoch 40/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.2957
- acc: 0.9165 - val_loss: 0.1313 - val_acc: 0.9620
Epoch 41/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2951
- acc: 0.9157 - val_loss: 0.1296 - val_acc: 0.9627
Epoch 42/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2952
- acc: 0.9164 - val_loss: 0.1272 - val_acc: 0.9629
Epoch 43/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2864
- acc: 0.9183 - val_loss: 0.1278 - val_acc: 0.9625
Epoch 44/50
60000/60000 [==============================] - 10s 169us/step - loss: 0.2846
- acc: 0.9192 - val_loss: 0.1252 - val_acc: 0.9641
Epoch 45/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2811
- acc: 0.9202 - val_loss: 0.1257 - val_acc: 0.9639
Epoch 46/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2830
- acc: 0.9210 - val_loss: 0.1240 - val_acc: 0.9645
Epoch 47/50
60000/60000 [==============================] - 10s 162us/step - loss: 0.2760
- acc: 0.9215 - val_loss: 0.1232 - val_acc: 0.9656
Epoch 48/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2757
- acc: 0.9219 - val_loss: 0.1210 - val_acc: 0.9661
Epoch 49/50
60000/60000 [==============================] - 10s 164us/step - loss: 0.2691
- acc: 0.9237 - val_loss: 0.1198 - val_acc: 0.9654
Epoch 50/50
60000/60000 [==============================] - 10s 163us/step - loss: 0.2680
- acc: 0.9255 - val_loss: 0.1207 - val_acc: 0.9653
```

In [93]:

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

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

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

# 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: 0.24941171036660673
Test accuracy: 0.9267
```



In [95]:

```
w_after = model3.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()
```