Keras -- MLPs on MNIST

In [0]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

In [0]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal, he_normal
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
```

In [0]:
```
%matplotlib inline
#%matplotlib
#%matplotlib nbagg
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, ty, 'r', label="Train Loss")
    ax.plot(x, vy, 'b', label="Validation Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
    plt.show()
```

In [0]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [45]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)"%(X_train.shape[1], X_train.shape
print("Number of test examples :", X_test.shape[0], "and each image is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

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

In [0]:

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

In [47]:

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

```
Number of training examples : 60000 and each image is of shape (784)
Number of test examples : 10000 and each image is of shape (784)
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   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 [48]:

```python
X_train = X_train/255
X_test = X_test/255
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 [49]:

```python
print("Class label of first image :", y_train[0])
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print("After converting the output into a vector : ",Y_train[0])
```

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

Softmax classifier

In [0]:

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

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

In [0]:

```
# some model parameters
output_dim = 10
input_dim = X_train.shape[1]
batch_size = 128
nb_epoch = 22
```

2 hidden layer - MLP with Batch normalization and Dropout(0.5)

In [52]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(430, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(322, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_40 (Dense)             (None, 430)               337550    
_________________________________________________________________
batch_normalization_31 (Batc (None, 430)               1720      
_________________________________________________________________
dropout_31 (Dropout)         (None, 430)               0         
_________________________________________________________________
dense_41 (Dense)             (None, 322)               138782    
_________________________________________________________________
batch_normalization_32 (Batc (None, 322)               1288      
_________________________________________________________________
dropout_32 (Dropout)         (None, 322)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 10)                3230      
=================================================================
Total params: 482,570
Trainable params: 481,066
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 6s 96us/step - loss: 0.4084 - acc: 0.8767 - val_loss: 0.1316 - val_acc: 0.9579
Epoch 2/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.1966 - acc: 0.9399 - val_loss: 0.1040 - val_acc: 0.9679
Epoch 3/22
```

```
60000/60000 [==============================] - 4s 59us/step - loss: 0.1504 - acc: 0.9532 - val_loss: 0.0897 - val_acc: 0.9723
Epoch 4/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.1307 - acc: 0.9600 - val_loss: 0.0799 - val_acc: 0.9748
Epoch 5/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.1170 - acc: 0.9634 - val_loss: 0.0798 - val_acc: 0.9751
Epoch 6/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.1062 - acc: 0.9666 - val_loss: 0.0697 - val_acc: 0.9769
Epoch 7/22
60000/60000 [==============================] - 3s 58us/step - loss: 0.0952 - acc: 0.9703 - val_loss: 0.0700 - val_acc: 0.9786
Epoch 8/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0891 - acc: 0.9716 - val_loss: 0.0617 - val_acc: 0.9804
Epoch 9/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0845 - acc: 0.9738 - val_loss: 0.0672 - val_acc: 0.9792
Epoch 10/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0780 - acc: 0.9751 - val_loss: 0.0610 - val_acc: 0.9818
Epoch 11/22
60000/60000 [==============================] - 3s 58us/step - loss: 0.0725 - acc: 0.9769 - val_loss: 0.0599 - val_acc: 0.9820
Epoch 12/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0708 - acc: 0.9777 - val_loss: 0.0585 - val_acc: 0.9826
Epoch 13/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0660 - acc: 0.9776 - val_loss: 0.0625 - val_acc: 0.9809
Epoch 14/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0661 - acc: 0.9785 - val_loss: 0.0622 - val_acc: 0.9816
Epoch 15/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0604 - acc: 0.9804 - val_loss: 0.0586 - val_acc: 0.9837
Epoch 16/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0591 - acc: 0.9804 - val_loss: 0.0580 - val_acc: 0.9837
Epoch 17/22
60000/60000 [==============================] - 3s 58us/step - loss: 0.0611 - acc: 0.9801 - val_loss: 0.0552 - val_acc: 0.9837
Epoch 18/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0541 - acc: 0.9824 - val_loss: 0.0563 - val_acc: 0.9842
Epoch 19/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0552 - acc: 0.9827 - val_loss: 0.0574 - val_acc: 0.9843
Epoch 20/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0502 - acc: 0.9838 - val_loss: 0.0569 - val_acc: 0.9839
Epoch 21/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0506 - acc: 0.9835 - val_loss: 0.0528 - val_acc: 0.9841
Epoch 22/22
60000/60000 [==============================] - 4s 58us/step - loss: 0.0498 - acc: 0.9838 - val_loss: 0.0515 - val_acc: 0.9856
CPU times: user 1min 37s, sys: 7.86 s, total: 1min 45s
Wall time: 1min 19s
```

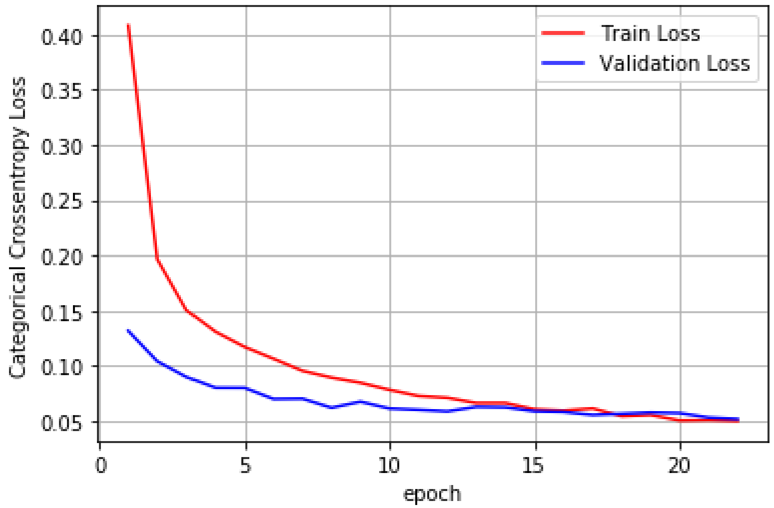
In [53]:

```
score = model_relu.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))

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

```
Test score: 0.051465931210145935
Test accuracy: 0.9856
```



In [54]:

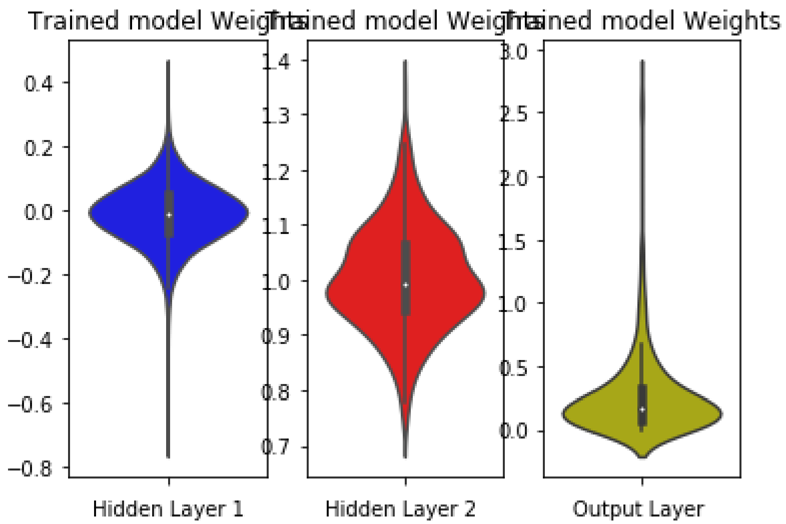
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



3 hidden layer - MLP with Batch normalization and Dropout(0.5)

In [55]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(270, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(162, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_43 (Dense)             (None, 320)               251200
_________________________________________________________________
batch_normalization_33 (Batc (None, 320)               1280
_________________________________________________________________
dropout_33 (Dropout)         (None, 320)               0
_________________________________________________________________
dense_44 (Dense)             (None, 270)               86670
_________________________________________________________________
batch_normalization_34 (Batc (None, 270)               1080
_________________________________________________________________
dropout_34 (Dropout)         (None, 270)               0
_________________________________________________________________
dense_45 (Dense)             (None, 162)               43902
_________________________________________________________________
batch_normalization_35 (Batc (None, 162)               648
_________________________________________________________________
dropout_35 (Dropout)         (None, 162)               0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1630
=================================================================
Total params: 386,410
Trainable params: 384,906
```

```
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 7s 115us/step - loss: 0.6269 - acc: 0.8051 - val_loss: 0.1738 - val_acc: 0.9466
Epoch 2/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.2746 - acc: 0.9184 - val_loss: 0.1327 - val_acc: 0.9583
Epoch 3/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.2140 - acc: 0.9364 - val_loss: 0.1081 - val_acc: 0.9651
Epoch 4/22
60000/60000 [==============================] - 4s 71us/step - loss: 0.1834 - acc: 0.9457 - val_loss: 0.0949 - val_acc: 0.9705
Epoch 5/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.1573 - acc: 0.9532 - val_loss: 0.0908 - val_acc: 0.9708
Epoch 6/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.1418 - acc: 0.9573 - val_loss: 0.0824 - val_acc: 0.9752
Epoch 7/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.1276 - acc: 0.9618 - val_loss: 0.0787 - val_acc: 0.9757
Epoch 8/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.1192 - acc: 0.9641 - val_loss: 0.0758 - val_acc: 0.9768
Epoch 9/22
60000/60000 [==============================] - 4s 71us/step - loss: 0.1159 - acc: 0.9652 - val_loss: 0.0706 - val_acc: 0.9784
Epoch 10/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.1067 - acc: 0.9674 - val_loss: 0.0740 - val_acc: 0.9768
Epoch 11/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.1051 - acc: 0.9670 - val_loss: 0.0675 - val_acc: 0.9792
Epoch 12/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.0980 - acc: 0.9701 - val_loss: 0.0737 - val_acc: 0.9776
Epoch 13/22
60000/60000 [==============================] - 4s 71us/step - loss: 0.0965 - acc: 0.9708 - val_loss: 0.0619 - val_acc: 0.9807
Epoch 14/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.0909 - acc: 0.9724 - val_loss: 0.0616 - val_acc: 0.9826
Epoch 15/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0856 - acc: 0.9738 - val_loss: 0.0665 - val_acc: 0.9795
Epoch 16/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0866 - acc: 0.9736 - val_loss: 0.0630 - val_acc: 0.9816
Epoch 17/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0814 - acc: 0.9750 - val_loss: 0.0680 - val_acc: 0.9806
Epoch 18/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0797 - acc: 0.9750 - val_loss: 0.0661 - val_acc: 0.9813
Epoch 19/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0778 - acc: 0.9760 - val_loss: 0.0618 - val_acc: 0.9818
Epoch 20/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0745 - acc: 0.9768 - val_loss: 0.0621 - val_acc: 0.9822
Epoch 21/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.0711 - acc: 0.9779 - val_loss: 0.0622 - val_acc: 0.9827
```

```
Epoch 22/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0700 - acc: 0.9790 - val_loss: 0.0603 - val_acc: 0.9820
CPU times: user 1min 59s, sys: 9.78 s, total: 2min 9s
Wall time: 1min 34s
```

In [56]:

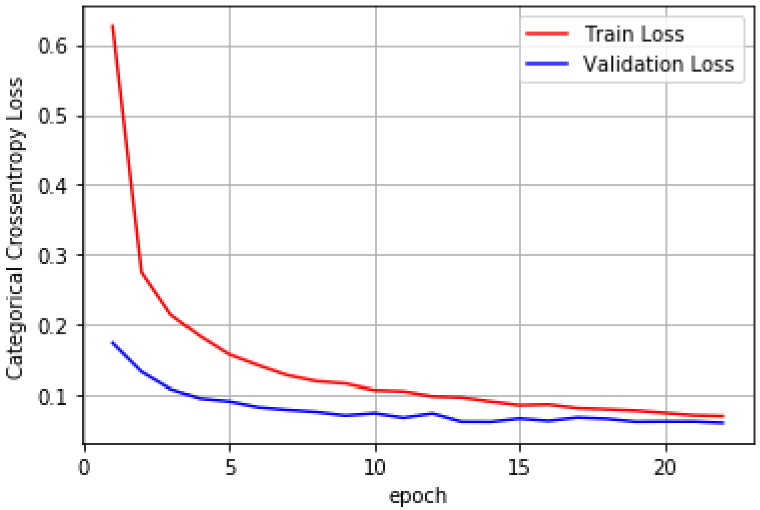
```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

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

```
Test score: 0.06025076956005068
Test accuracy: 0.982
```



In [57]:

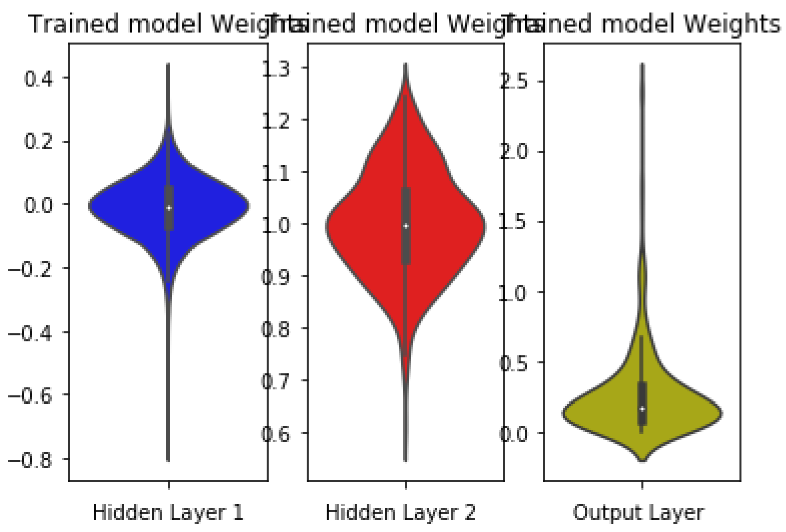
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



5 hidden layer - MLP with Batch normalization and Dropout(0.5)

In [58]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(340, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(190, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(112, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(72, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 340)               266900
_________________________________________________________________
batch_normalization_36 (Batc (None, 340)               1360
_________________________________________________________________
dropout_36 (Dropout)         (None, 340)               0
_________________________________________________________________
dense_48 (Dense)             (None, 190)               64790
_________________________________________________________________
batch_normalization_37 (Batc (None, 190)               760
_________________________________________________________________
dropout_37 (Dropout)         (None, 190)               0
_________________________________________________________________
dense_49 (Dense)             (None, 112)               21392
_________________________________________________________________
batch_normalization_38 (Batc (None, 112)               448
_________________________________________________________________
```

```
dropout_38 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_50 (Dense)             (None, 72)                8136
_________________________________________________________________
batch_normalization_39 (Batc (None, 72)                288
_________________________________________________________________
dropout_39 (Dropout)         (None, 72)                0
_________________________________________________________________
dense_51 (Dense)             (None, 38)                2774
_________________________________________________________________
batch_normalization_40 (Batc (None, 38)                152
_________________________________________________________________
dropout_40 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                390
=================================================================
Total params: 367,390
Trainable params: 365,886
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 9s 151us/step - loss: 1.5619 - acc: 0.4815 - val_loss: 0.4524 - val_acc: 0.8852
Epoch 2/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.6659 - acc: 0.7948 - val_loss: 0.2386 - val_acc: 0.9382
Epoch 3/22
60000/60000 [==============================] - 6s 95us/step - loss: 0.4515 - acc: 0.8809 - val_loss: 0.1770 - val_acc: 0.9544
Epoch 4/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.3605 - acc: 0.9086 - val_loss: 0.1603 - val_acc: 0.9598
Epoch 5/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.3098 - acc: 0.9238 - val_loss: 0.1407 - val_acc: 0.9652
Epoch 6/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.2733 - acc: 0.9344 - val_loss: 0.1306 - val_acc: 0.9678
Epoch 7/22
60000/60000 [==============================] - 6s 95us/step - loss: 0.2510 - acc: 0.9403 - val_loss: 0.1234 - val_acc: 0.9706
Epoch 8/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.2338 - acc: 0.9445 - val_loss: 0.1102 - val_acc: 0.9721
Epoch 9/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.2155 - acc: 0.9492 - val_loss: 0.1160 - val_acc: 0.9721
Epoch 10/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.2071 - acc: 0.9517 - val_loss: 0.1066 - val_acc: 0.9742
Epoch 11/22
60000/60000 [==============================] - 6s 94us/step - loss: 0.1990 - acc: 0.9537 - val_loss: 0.0999 - val_acc: 0.9749
Epoch 12/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.1885 - acc: 0.9569 - val_loss: 0.1000 - val_acc: 0.9759
```

```
Epoch 13/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1792 - acc: 0.9588 - val_loss: 0.0948 - val_acc: 0.9777
Epoch 14/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1725 - acc: 0.9602 - val_loss: 0.1010 - val_acc: 0.9768
Epoch 15/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.1693 - acc: 0.9605 - val_loss: 0.0946 - val_acc: 0.9776
Epoch 16/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.1565 - acc: 0.9633 - val_loss: 0.0939 - val_acc: 0.9773
Epoch 17/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1565 - acc: 0.9644 - val_loss: 0.0961 - val_acc: 0.9779
Epoch 18/22
60000/60000 [==============================] - 6s 94us/step - loss: 0.1485 - acc: 0.9666 - val_loss: 0.0900 - val_acc: 0.9786
Epoch 19/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1456 - acc: 0.9673 - val_loss: 0.0898 - val_acc: 0.9789
Epoch 20/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.1453 - acc: 0.9669 - val_loss: 0.0906 - val_acc: 0.9798
Epoch 21/22
60000/60000 [==============================] - 6s 95us/step - loss: 0.1363 - acc: 0.9686 - val_loss: 0.0883 - val_acc: 0.9808
Epoch 22/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1417 - acc: 0.9680 - val_loss: 0.0792 - val_acc: 0.9828
CPU times: user 2min 41s, sys: 13 s, total: 2min 54s
Wall time: 2min 7s
```

In [59]:

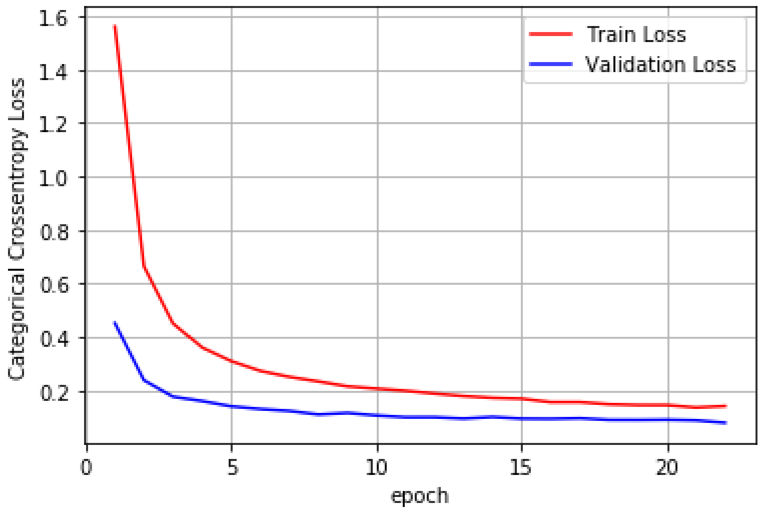
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

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

```
Test score: 0.07917401027954184
Test accuracy: 0.9828
```



In [60]:

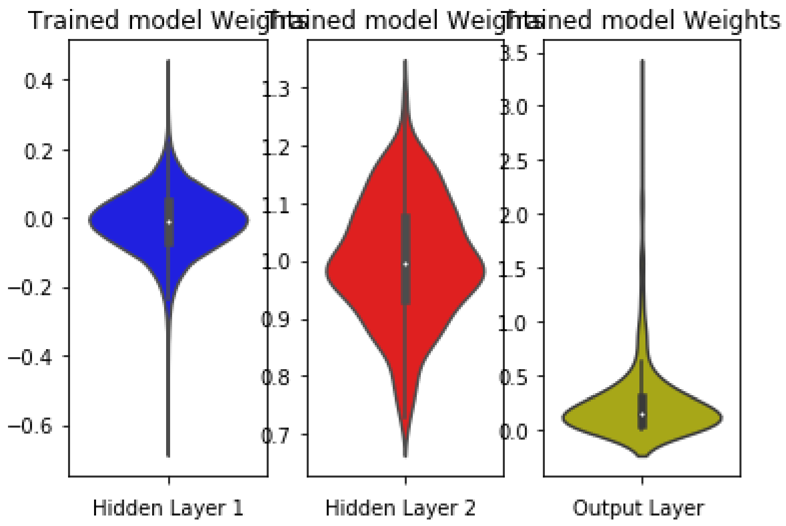
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



2 hidden layer - MLP with Batch normalization and Dropout(0.25)

In [61]:

```python
%%time
model_relu = Sequential()
model_relu.add(Dense(430, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(322, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 430)               337550    
_________________________________________________________________
batch_normalization_41 (Batc (None, 430)               1720      
_________________________________________________________________
dropout_41 (Dropout)         (None, 430)               0         
_________________________________________________________________
dense_54 (Dense)             (None, 322)               138782    
_________________________________________________________________
batch_normalization_42 (Batc (None, 322)               1288      
_________________________________________________________________
dropout_42 (Dropout)         (None, 322)               0         
_________________________________________________________________
dense_55 (Dense)             (None, 10)                3230      
=================================================================
Total params: 482,570
Trainable params: 481,066
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 6s 108us/step - loss: 0.2631 - acc: 0.9192 - val_loss: 0.1142 - val_acc: 0.9656
Epoch 2/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.1189 - acc: 0.9634 - val_loss: 0.0845 - val_acc: 0.9738
Epoch 3/22
```

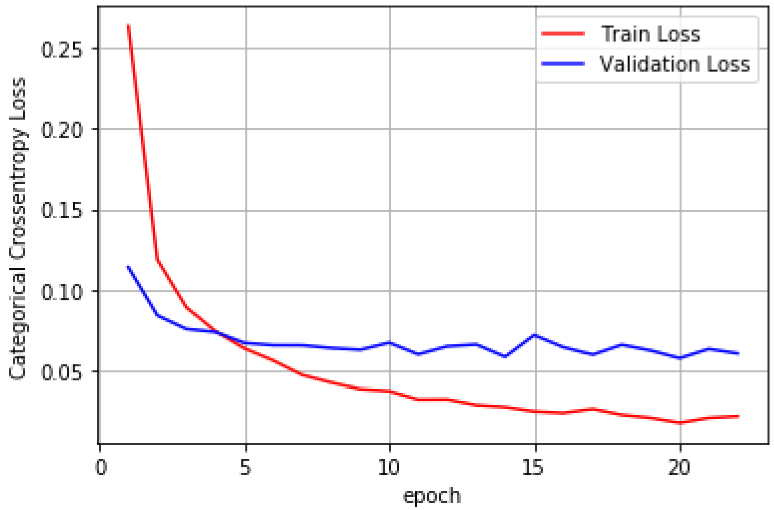
```
60000/60000 [==============================] - 3s 57us/step - loss: 0.0894 - acc: 0.9722 - val_loss: 0.0761 - val_acc: 0.9779
Epoch 4/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0747 - acc: 0.9760 - val_loss: 0.0742 - val_acc: 0.9782
Epoch 5/22
60000/60000 [==============================] - 3s 53us/step - loss: 0.0642 - acc: 0.9796 - val_loss: 0.0675 - val_acc: 0.9792
Epoch 6/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0567 - acc: 0.9813 - val_loss: 0.0661 - val_acc: 0.9808
Epoch 7/22
60000/60000 [==============================] - 3s 54us/step - loss: 0.0478 - acc: 0.9845 - val_loss: 0.0661 - val_acc: 0.9808
Epoch 8/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0432 - acc: 0.9859 - val_loss: 0.0643 - val_acc: 0.9803
Epoch 9/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0392 - acc: 0.9867 - val_loss: 0.0633 - val_acc: 0.9816
Epoch 10/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0380 - acc: 0.9871 - val_loss: 0.0677 - val_acc: 0.9821
Epoch 11/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0328 - acc: 0.9889 - val_loss: 0.0605 - val_acc: 0.9847
Epoch 12/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0329 - acc: 0.9891 - val_loss: 0.0654 - val_acc: 0.9829
Epoch 13/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0294 - acc: 0.9899 - val_loss: 0.0666 - val_acc: 0.9817
Epoch 14/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0282 - acc: 0.9903 - val_loss: 0.0590 - val_acc: 0.9839
Epoch 15/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0255 - acc: 0.9911 - val_loss: 0.0723 - val_acc: 0.9812
Epoch 16/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0246 - acc: 0.9916 - val_loss: 0.0649 - val_acc: 0.9831
Epoch 17/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0270 - acc: 0.9908 - val_loss: 0.0603 - val_acc: 0.9828
Epoch 18/22
60000/60000 [==============================] - 3s 54us/step - loss: 0.0234 - acc: 0.9922 - val_loss: 0.0664 - val_acc: 0.9819
Epoch 19/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0215 - acc: 0.9928 - val_loss: 0.0628 - val_acc: 0.9829
Epoch 20/22
60000/60000 [==============================] - 3s 58us/step - loss: 0.0185 - acc: 0.9932 - val_loss: 0.0581 - val_acc: 0.9845
Epoch 21/22
60000/60000 [==============================] - 3s 54us/step - loss: 0.0215 - acc: 0.9928 - val_loss: 0.0638 - val_acc: 0.9846
Epoch 22/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0226 - acc: 0.9922 - val_loss: 0.0610 - val_acc: 0.9838
CPU times: user 1min 35s, sys: 7.73 s, total: 1min 43s
Wall time: 1min 17s
```

In [62]:
```
score = model_relu.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))

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

```
Test score: 0.061047554688937 45
Test accuracy: 0.9838
```



In [63]:

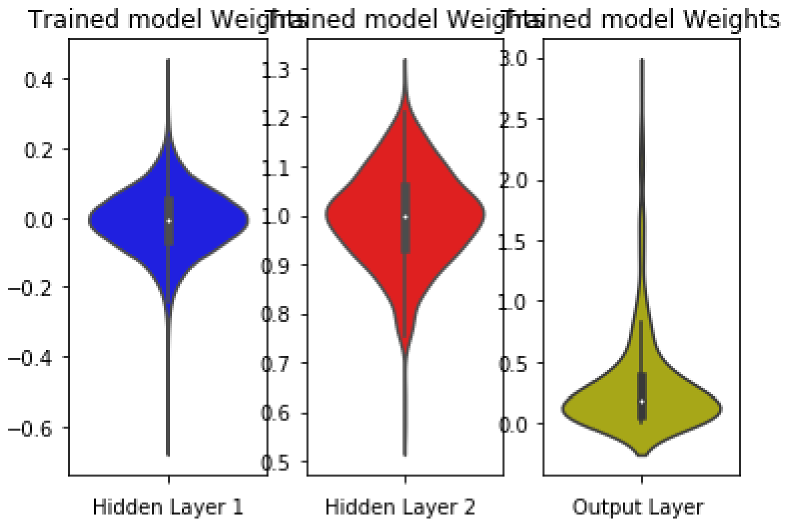
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



3 hidden layer - MLP with Batch normalization and Dropout(0.25)

In [64]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(270, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(162, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 320)               251200    
_________________________________________________________________
batch_normalization_43 (Batc (None, 320)               1280      
_________________________________________________________________
dropout_43 (Dropout)         (None, 320)               0         
_________________________________________________________________
dense_57 (Dense)             (None, 270)               86670     
_________________________________________________________________
batch_normalization_44 (Batc (None, 270)               1080      
_________________________________________________________________
dropout_44 (Dropout)         (None, 270)               0         
_________________________________________________________________
dense_58 (Dense)             (None, 162)               43902     
_________________________________________________________________
batch_normalization_45 (Batc (None, 162)               648       
_________________________________________________________________
dropout_45 (Dropout)         (None, 162)               0         
_________________________________________________________________
dense_59 (Dense)             (None, 10)                1630      
=================================================================
Total params: 386,410
Trainable params: 384,906
```

```
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 7s 123us/step - loss: 0.3332 - acc: 0.8979 - val_loss: 0.1211 - val_acc: 0.9635
Epoch 2/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.1476 - acc: 0.9552 - val_loss: 0.0920 - val_acc: 0.9712
Epoch 3/22
60000/60000 [==============================] - 4s 72us/step - loss: 0.1152 - acc: 0.9647 - val_loss: 0.0842 - val_acc: 0.9732
Epoch 4/22
60000/60000 [==============================] - 4s 70us/step - loss: 0.0954 - acc: 0.9700 - val_loss: 0.0704 - val_acc: 0.9777
Epoch 5/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0809 - acc: 0.9744 - val_loss: 0.0716 - val_acc: 0.9786
Epoch 6/22
60000/60000 [==============================] - 4s 70us/step - loss: 0.0708 - acc: 0.9769 - val_loss: 0.0657 - val_acc: 0.9790
Epoch 7/22
60000/60000 [==============================] - 4s 72us/step - loss: 0.0665 - acc: 0.9784 - val_loss: 0.0660 - val_acc: 0.9802
Epoch 8/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0598 - acc: 0.9805 - val_loss: 0.0615 - val_acc: 0.9805
Epoch 9/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0561 - acc: 0.9819 - val_loss: 0.0637 - val_acc: 0.9798
Epoch 10/22
60000/60000 [==============================] - 4s 69us/step - loss: 0.0523 - acc: 0.9826 - val_loss: 0.0670 - val_acc: 0.9806
Epoch 11/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0480 - acc: 0.9847 - val_loss: 0.0637 - val_acc: 0.9819
Epoch 12/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0437 - acc: 0.9859 - val_loss: 0.0608 - val_acc: 0.9824
Epoch 13/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0428 - acc: 0.9863 - val_loss: 0.0608 - val_acc: 0.9836
Epoch 14/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0425 - acc: 0.9862 - val_loss: 0.0617 - val_acc: 0.9818
Epoch 15/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0379 - acc: 0.9874 - val_loss: 0.0643 - val_acc: 0.9822
Epoch 16/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0359 - acc: 0.9883 - val_loss: 0.0690 - val_acc: 0.9809
Epoch 17/22
60000/60000 [==============================] - 4s 70us/step - loss: 0.0363 - acc: 0.9878 - val_loss: 0.0632 - val_acc: 0.9831
Epoch 18/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0320 - acc: 0.9893 - val_loss: 0.0596 - val_acc: 0.9828
Epoch 19/22
60000/60000 [==============================] - 4s 70us/step - loss: 0.0316 - acc: 0.9893 - val_loss: 0.0570 - val_acc: 0.9839
Epoch 20/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0310 - acc: 0.9899 - val_loss: 0.0565 - val_acc: 0.9841
Epoch 21/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0290 - acc: 0.9901 - val_loss: 0.0604 - val_acc: 0.9837
```

```
Epoch 22/22
60000/60000 [==============================] - 4s 70us/step - loss: 0.0255 - acc: 0.9913 - val_loss: 0.0632 - val_acc: 0.9823
CPU times: user 2min, sys: 9.8 s, total: 2min 9s
Wall time: 1min 34s
```

In [65]:

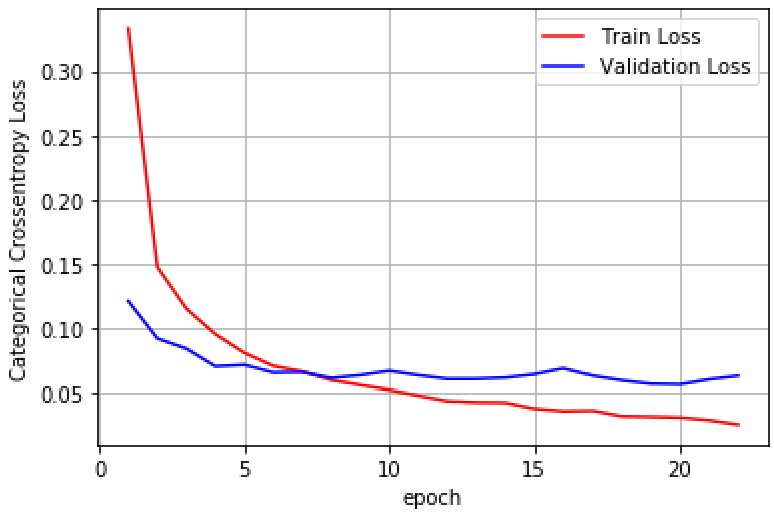
```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

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

```
Test score: 0.06317311227827886
Test accuracy: 0.9823
```



In [66]:

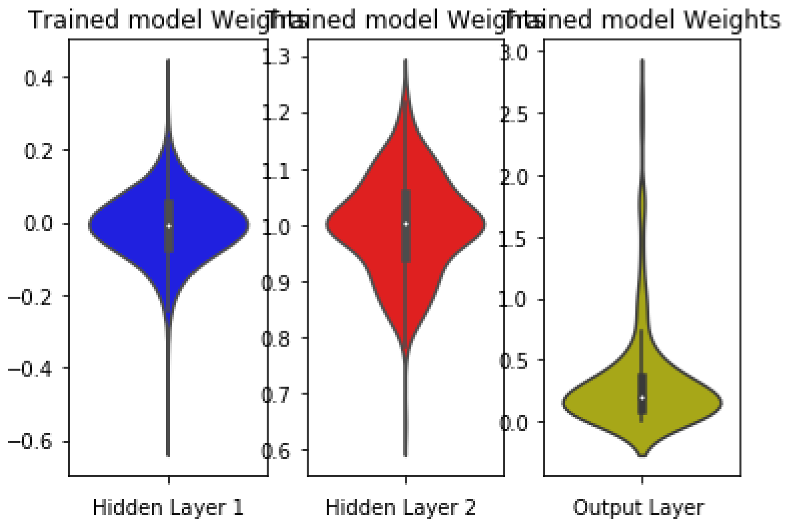
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



5 hidden layer - MLP with Batch normalization and Dropout(0.25)

In [67]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(340, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(190, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(112, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(72, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.25))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_60 (Dense)             (None, 340)               266900
_________________________________________________________________
batch_normalization_46 (Batc (None, 340)               1360
_________________________________________________________________
dropout_46 (Dropout)         (None, 340)               0
_________________________________________________________________
dense_61 (Dense)             (None, 190)               64790
_________________________________________________________________
batch_normalization_47 (Batc (None, 190)               760
_________________________________________________________________
dropout_47 (Dropout)         (None, 190)               0
_________________________________________________________________
dense_62 (Dense)             (None, 112)               21392
_________________________________________________________________
batch_normalization_48 (Batc (None, 112)               448
_________________________________________________________________
```

```
dropout_48 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_63 (Dense)             (None, 72)                8136
_________________________________________________________________
batch_normalization_49 (Batc (None, 72)                288
_________________________________________________________________
dropout_49 (Dropout)         (None, 72)                0
_________________________________________________________________
dense_64 (Dense)             (None, 38)                2774
_________________________________________________________________
batch_normalization_50 (Batc (None, 38)                152
_________________________________________________________________
dropout_50 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_65 (Dense)             (None, 10)                390
=================================================================
Total params: 367,390
Trainable params: 365,886
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 9s 158us/step - loss: 0.6399 - acc: 0.8115 - val_loss: 0.1626 - val_acc: 0.9524
Epoch 2/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.2371 - acc: 0.9354 - val_loss: 0.1139 - val_acc: 0.9656
Epoch 3/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.1787 - acc: 0.9519 - val_loss: 0.1129 - val_acc: 0.9670
Epoch 4/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.1463 - acc: 0.9602 - val_loss: 0.0871 - val_acc: 0.9752
Epoch 5/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1288 - acc: 0.9641 - val_loss: 0.0904 - val_acc: 0.9751
Epoch 6/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1159 - acc: 0.9678 - val_loss: 0.0823 - val_acc: 0.9773
Epoch 7/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.1024 - acc: 0.9720 - val_loss: 0.0843 - val_acc: 0.9769
Epoch 8/22
60000/60000 [==============================] - 6s 94us/step - loss: 0.0970 - acc: 0.9727 - val_loss: 0.0812 - val_acc: 0.9780
Epoch 9/22
60000/60000 [==============================] - 6s 96us/step - loss: 0.0920 - acc: 0.9744 - val_loss: 0.0740 - val_acc: 0.9801
Epoch 10/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0849 - acc: 0.9762 - val_loss: 0.0728 - val_acc: 0.9798
Epoch 11/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.0778 - acc: 0.9781 - val_loss: 0.0756 - val_acc: 0.9801
Epoch 12/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0724 - acc: 0.9793 - val_loss: 0.0770 - val_acc: 0.9799
```

```
Epoch 13/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0678 - acc: 0.9810 - val_loss: 0.0694 - val_acc: 0.9820
Epoch 14/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0669 - acc: 0.9815 - val_loss: 0.0633 - val_acc: 0.9827
Epoch 15/22
60000/60000 [==============================] - 5s 92us/step - loss: 0.0621 - acc: 0.9828 - val_loss: 0.0725 - val_acc: 0.9822
Epoch 16/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0589 - acc: 0.9833 - val_loss: 0.0686 - val_acc: 0.9823
Epoch 17/22
60000/60000 [==============================] - 6s 94us/step - loss: 0.0569 - acc: 0.9838 - val_loss: 0.0626 - val_acc: 0.9828
Epoch 18/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0536 - acc: 0.9852 - val_loss: 0.0676 - val_acc: 0.9825
Epoch 19/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0529 - acc: 0.9847 - val_loss: 0.0763 - val_acc: 0.9805
Epoch 20/22
60000/60000 [==============================] - 6s 95us/step - loss: 0.0524 - acc: 0.9849 - val_loss: 0.0668 - val_acc: 0.9833
Epoch 21/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0480 - acc: 0.9864 - val_loss: 0.0664 - val_acc: 0.9841
Epoch 22/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0445 - acc: 0.9867 - val_loss: 0.0704 - val_acc: 0.9838
CPU times: user 2min 40s, sys: 13 s, total: 2min 53s
Wall time: 2min 7s
```

In [68]:

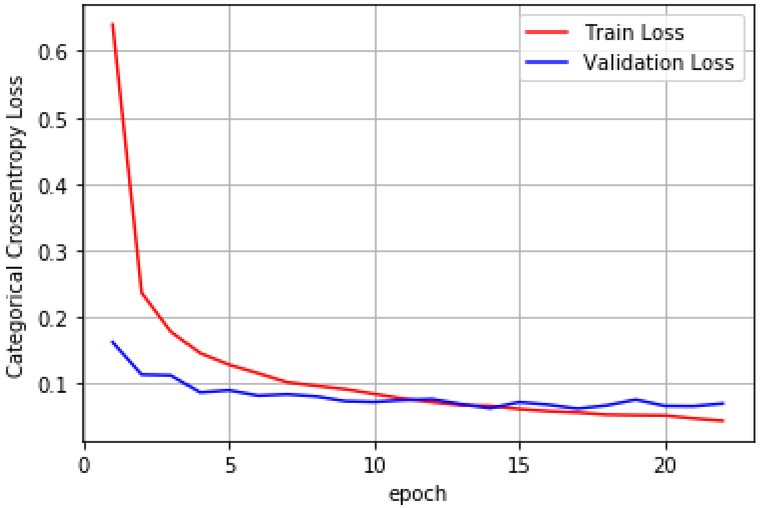
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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))

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

```
Test score: 0.07041437144926749
Test accuracy: 0.9838
```



In [69]:

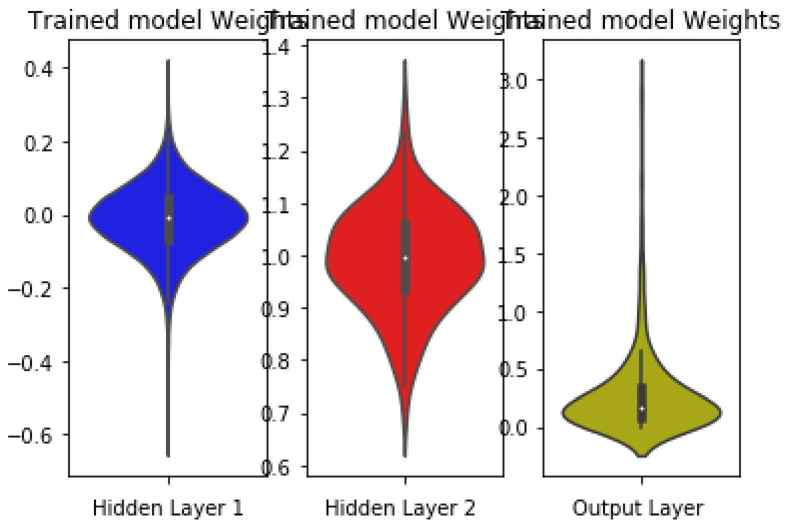
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



2 hidden layer - MLP with Batch normalization,Dropout(0.5) and Mean Squared Error

In [70]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(430, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(322, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_66 (Dense)             (None, 430)               337550    
_________________________________________________________________
batch_normalization_51 (Batc (None, 430)               1720      
_________________________________________________________________
dropout_51 (Dropout)         (None, 430)               0         
_________________________________________________________________
dense_67 (Dense)             (None, 322)               138782    
_________________________________________________________________
batch_normalization_52 (Batc (None, 322)               1288      
_________________________________________________________________
dropout_52 (Dropout)         (None, 322)               0         
_________________________________________________________________
dense_68 (Dense)             (None, 10)                3230      
=================================================================
Total params: 482,570
Trainable params: 481,066
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 7s 120us/step - loss: 0.0200 - acc: 0.8633 - val_loss: 0.0079 - val_acc: 0.9483
Epoch 2/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0103 - acc: 0.9329 - val_loss: 0.0060 - val_acc: 0.9611
Epoch 3/22
```

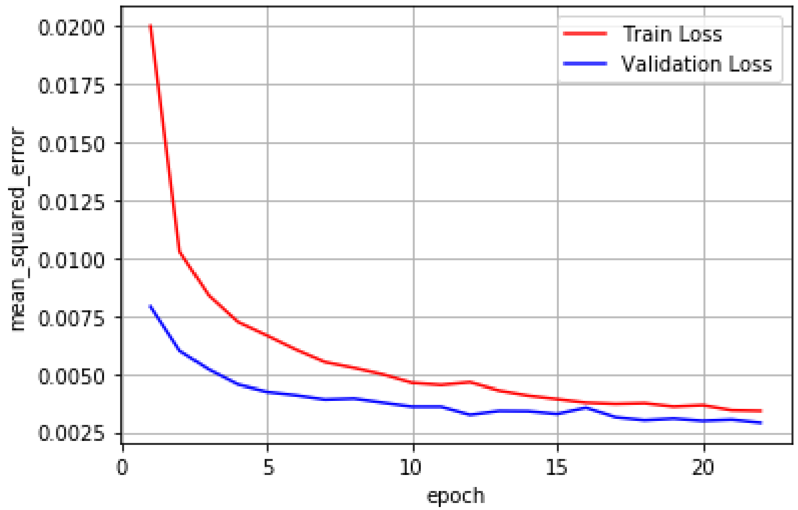
```
60000/60000 [==============================] - 3s 54us/step - loss: 0.0084 - acc: 0.9455 - val_loss: 0.0052 - val_acc: 0.9666
Epoch 4/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0072 - acc: 0.9534 - val_loss: 0.0045 - val_acc: 0.9705
Epoch 5/22
60000/60000 [==============================] - 3s 56us/step - loss: 0.0067 - acc: 0.9568 - val_loss: 0.0042 - val_acc: 0.9730
Epoch 6/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0060 - acc: 0.9612 - val_loss: 0.0041 - val_acc: 0.9733
Epoch 7/22
60000/60000 [==============================] - 4s 58us/step - loss: 0.0055 - acc: 0.9649 - val_loss: 0.0039 - val_acc: 0.9749
Epoch 8/22
60000/60000 [==============================] - 3s 58us/step - loss: 0.0053 - acc: 0.9664 - val_loss: 0.0039 - val_acc: 0.9743
Epoch 9/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0050 - acc: 0.9684 - val_loss: 0.0038 - val_acc: 0.9759
Epoch 10/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0046 - acc: 0.9704 - val_loss: 0.0036 - val_acc: 0.9759
Epoch 11/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0045 - acc: 0.9710 - val_loss: 0.0036 - val_acc: 0.9771
Epoch 12/22
60000/60000 [==============================] - 4s 59us/step - loss: 0.0046 - acc: 0.9699 - val_loss: 0.0032 - val_acc: 0.9792
Epoch 13/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0043 - acc: 0.9723 - val_loss: 0.0034 - val_acc: 0.9780
Epoch 14/22
60000/60000 [==============================] - 3s 54us/step - loss: 0.0041 - acc: 0.9736 - val_loss: 0.0034 - val_acc: 0.9780
Epoch 15/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0039 - acc: 0.9750 - val_loss: 0.0033 - val_acc: 0.9782
Epoch 16/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0038 - acc: 0.9759 - val_loss: 0.0036 - val_acc: 0.9771
Epoch 17/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0037 - acc: 0.9764 - val_loss: 0.0031 - val_acc: 0.9798
Epoch 18/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0038 - acc: 0.9762 - val_loss: 0.0030 - val_acc: 0.9808
Epoch 19/22
60000/60000 [==============================] - 3s 53us/step - loss: 0.0036 - acc: 0.9770 - val_loss: 0.0031 - val_acc: 0.9801
Epoch 20/22
60000/60000 [==============================] - 3s 54us/step - loss: 0.0037 - acc: 0.9768 - val_loss: 0.0030 - val_acc: 0.9811
Epoch 21/22
60000/60000 [==============================] - 3s 57us/step - loss: 0.0034 - acc: 0.9783 - val_loss: 0.0030 - val_acc: 0.9813
Epoch 22/22
60000/60000 [==============================] - 3s 55us/step - loss: 0.0034 - acc: 0.9781 - val_loss: 0.0029 - val_acc: 0.9819
CPU times: user 1min 35s, sys: 8.25 s, total: 1min 44s
Wall time: 1min 18s
```

In [71]:
```
score = model_relu.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('mean_squared_error')

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

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

```
Test score: 0.002905523286913556
Test accuracy: 0.9819
```



In [72]:

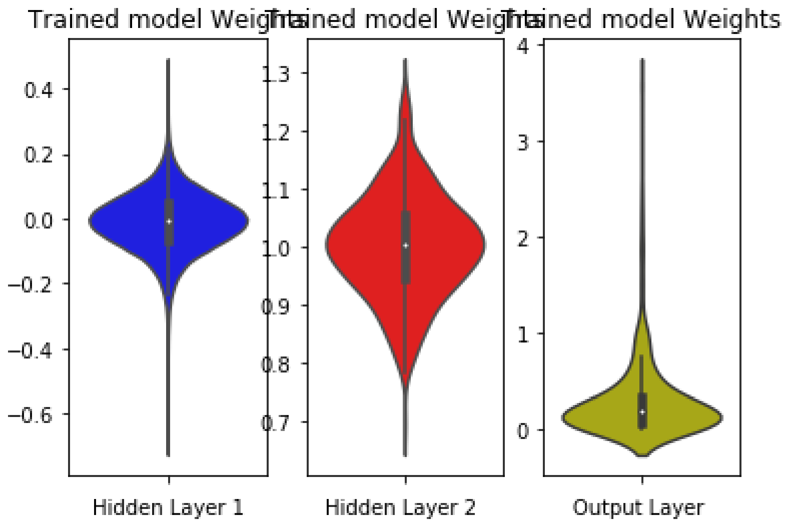
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



3 hidden layer - MLP with Batch normalization,Dropout(0.5) and Mean Squared Error

In [73]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(270, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(162, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_69 (Dense)             (None, 320)               251200    
_________________________________________________________________
batch_normalization_53 (Batc (None, 320)               1280      
_________________________________________________________________
dropout_53 (Dropout)         (None, 320)               0         
_________________________________________________________________
dense_70 (Dense)             (None, 270)               86670     
_________________________________________________________________
batch_normalization_54 (Batc (None, 270)               1080      
_________________________________________________________________
dropout_54 (Dropout)         (None, 270)               0         
_________________________________________________________________
dense_71 (Dense)             (None, 162)               43902     
_________________________________________________________________
batch_normalization_55 (Batc (None, 162)               648       
_________________________________________________________________
dropout_55 (Dropout)         (None, 162)               0         
_________________________________________________________________
dense_72 (Dense)             (None, 10)                1630      
=================================================================
Total params: 386,410
Trainable params: 384,906
```

```
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 8s 137us/step - loss: 0.0285 - acc: 0.7985 - val_loss: 0.0101 - val_acc: 0.9338
Epoch 2/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0131 - acc: 0.9137 - val_loss: 0.0071 - val_acc: 0.9550
Epoch 3/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0106 - acc: 0.9309 - val_loss: 0.0060 - val_acc: 0.9626
Epoch 4/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0094 - acc: 0.9389 - val_loss: 0.0052 - val_acc: 0.9672
Epoch 5/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0083 - acc: 0.9462 - val_loss: 0.0052 - val_acc: 0.9664
Epoch 6/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0077 - acc: 0.9508 - val_loss: 0.0047 - val_acc: 0.9694
Epoch 7/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0071 - acc: 0.9540 - val_loss: 0.0045 - val_acc: 0.9708
Epoch 8/22
60000/60000 [==============================] - 4s 63us/step - loss: 0.0066 - acc: 0.9574 - val_loss: 0.0042 - val_acc: 0.9734
Epoch 9/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0062 - acc: 0.9598 - val_loss: 0.0039 - val_acc: 0.9748
Epoch 10/22
60000/60000 [==============================] - 4s 67us/step - loss: 0.0060 - acc: 0.9617 - val_loss: 0.0040 - val_acc: 0.9743
Epoch 11/22
60000/60000 [==============================] - 4s 64us/step - loss: 0.0058 - acc: 0.9627 - val_loss: 0.0037 - val_acc: 0.9764
Epoch 12/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0055 - acc: 0.9647 - val_loss: 0.0038 - val_acc: 0.9764
Epoch 13/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0053 - acc: 0.9666 - val_loss: 0.0037 - val_acc: 0.9760
Epoch 14/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0049 - acc: 0.9687 - val_loss: 0.0035 - val_acc: 0.9783
Epoch 15/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0050 - acc: 0.9685 - val_loss: 0.0036 - val_acc: 0.9770
Epoch 16/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0048 - acc: 0.9692 - val_loss: 0.0037 - val_acc: 0.9771
Epoch 17/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0045 - acc: 0.9707 - val_loss: 0.0035 - val_acc: 0.9773
Epoch 18/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0046 - acc: 0.9703 - val_loss: 0.0035 - val_acc: 0.9783
Epoch 19/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0045 - acc: 0.9719 - val_loss: 0.0036 - val_acc: 0.9762
Epoch 20/22
60000/60000 [==============================] - 4s 68us/step - loss: 0.0042 - acc: 0.9730 - val_loss: 0.0034 - val_acc: 0.9796
Epoch 21/22
60000/60000 [==============================] - 4s 66us/step - loss: 0.0041 - acc: 0.9741 - val_loss: 0.0032 - val_acc: 0.9790
```

```
Epoch 22/22
60000/60000 [==============================] - 4s 65us/step - loss: 0.0041 - acc: 0.9735 - val_loss: 0.0032 - val_acc: 0.9800
CPU times: user 1min 54s, sys: 8.76 s, total: 2min 3s
Wall time: 1min 32s
```

In [74]:

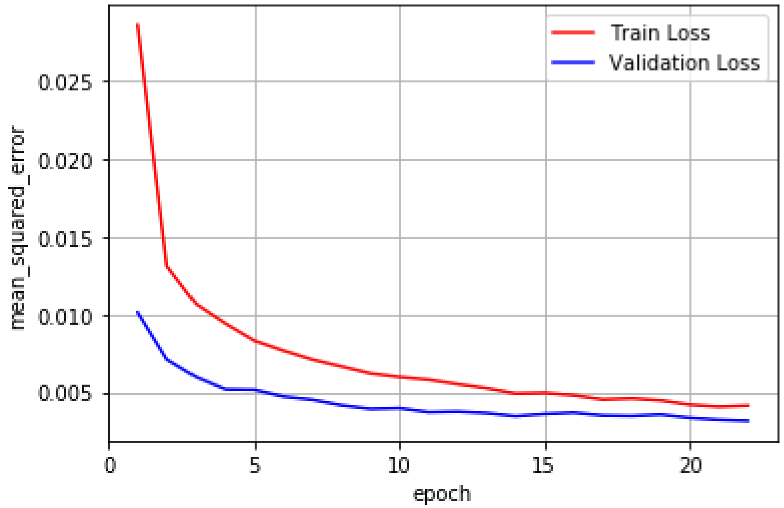
```python
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('mean_squared_error')

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

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

```
Test score: 0.0031677277905113464
Test accuracy: 0.98
```



In [75]:

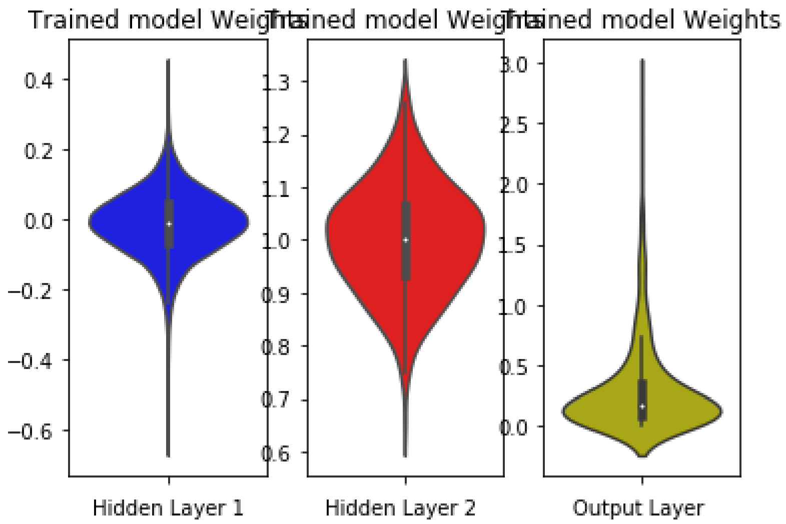
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



5 hidden layer - MLP with Batch normalization,Dropout(0.5) and Mean Squared Error

In [76]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(340, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(190, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(112, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(72, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(38, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_73 (Dense)             (None, 340)               266900
_________________________________________________________________
batch_normalization_56 (Batc (None, 340)               1360
_________________________________________________________________
dropout_56 (Dropout)         (None, 340)               0
_________________________________________________________________
dense_74 (Dense)             (None, 190)               64790
_________________________________________________________________
batch_normalization_57 (Batc (None, 190)               760
_________________________________________________________________
dropout_57 (Dropout)         (None, 190)               0
_________________________________________________________________
dense_75 (Dense)             (None, 112)               21392
_________________________________________________________________
batch_normalization_58 (Batc (None, 112)               448
_________________________________________________________________
```

```
dropout_58 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_76 (Dense)             (None, 72)                8136
_________________________________________________________________
batch_normalization_59 (Batc (None, 72)                288
_________________________________________________________________
dropout_59 (Dropout)         (None, 72)                0
_________________________________________________________________
dense_77 (Dense)             (None, 38)                2774
_________________________________________________________________
batch_normalization_60 (Batc (None, 38)                152
_________________________________________________________________
dropout_60 (Dropout)         (None, 38)                0
_________________________________________________________________
dense_78 (Dense)             (None, 10)                390
=================================================================
Total params: 367,390
Trainable params: 365,886
Non-trainable params: 1,504
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/22
60000/60000 [==============================] - 10s 171us/step - loss: 0.0692 - acc: 0.4424 - val_loss: 0.0222 - val_acc: 0.8628
Epoch 2/22
60000/60000 [==============================] - 5s 89us/step - loss: 0.0287 - acc: 0.8108 - val_loss: 0.0099 - val_acc: 0.9366
Epoch 3/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0177 - acc: 0.8909 - val_loss: 0.0081 - val_acc: 0.9509
Epoch 4/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0142 - acc: 0.9137 - val_loss: 0.0075 - val_acc: 0.9533
Epoch 5/22
60000/60000 [==============================] - 6s 94us/step - loss: 0.0124 - acc: 0.9252 - val_loss: 0.0065 - val_acc: 0.9617
Epoch 6/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.0115 - acc: 0.9311 - val_loss: 0.0063 - val_acc: 0.9633
Epoch 7/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0104 - acc: 0.9376 - val_loss: 0.0066 - val_acc: 0.9621
Epoch 8/22
60000/60000 [==============================] - 6s 93us/step - loss: 0.0098 - acc: 0.9415 - val_loss: 0.0059 - val_acc: 0.9661
Epoch 9/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0095 - acc: 0.9434 - val_loss: 0.0057 - val_acc: 0.9671
Epoch 10/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0091 - acc: 0.9448 - val_loss: 0.0054 - val_acc: 0.9683
Epoch 11/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0087 - acc: 0.9480 - val_loss: 0.0056 - val_acc: 0.9672
Epoch 12/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0083 - acc: 0.9506 - val_loss: 0.0050 - val_acc: 0.9713
```

```
Epoch 13/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0079 - acc: 0.9530 - val_loss: 0.0046 - val_acc: 0.9738
Epoch 14/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0077 - acc: 0.9547 - val_loss: 0.0050 - val_acc: 0.9706
Epoch 15/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0075 - acc: 0.9560 - val_loss: 0.0048 - val_acc: 0.9733
Epoch 16/22
60000/60000 [==============================] - 6s 95us/step - loss: 0.0074 - acc: 0.9558 - val_loss: 0.0048 - val_acc: 0.9724
Epoch 17/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0070 - acc: 0.9581 - val_loss: 0.0043 - val_acc: 0.9752
Epoch 18/22
60000/60000 [==============================] - 6s 92us/step - loss: 0.0072 - acc: 0.9569 - val_loss: 0.0044 - val_acc: 0.9749
Epoch 19/22
60000/60000 [==============================] - 5s 92us/step - loss: 0.0069 - acc: 0.9593 - val_loss: 0.0044 - val_acc: 0.9744
Epoch 20/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0069 - acc: 0.9590 - val_loss: 0.0043 - val_acc: 0.9749
Epoch 21/22
60000/60000 [==============================] - 5s 90us/step - loss: 0.0066 - acc: 0.9609 - val_loss: 0.0045 - val_acc: 0.9737
Epoch 22/22
60000/60000 [==============================] - 5s 91us/step - loss: 0.0063 - acc: 0.9624 - val_loss: 0.0043 - val_acc: 0.9756
CPU times: user 2min 39s, sys: 12.7 s, total: 2min 52s
Wall time: 2min 7s
```

In [77]:

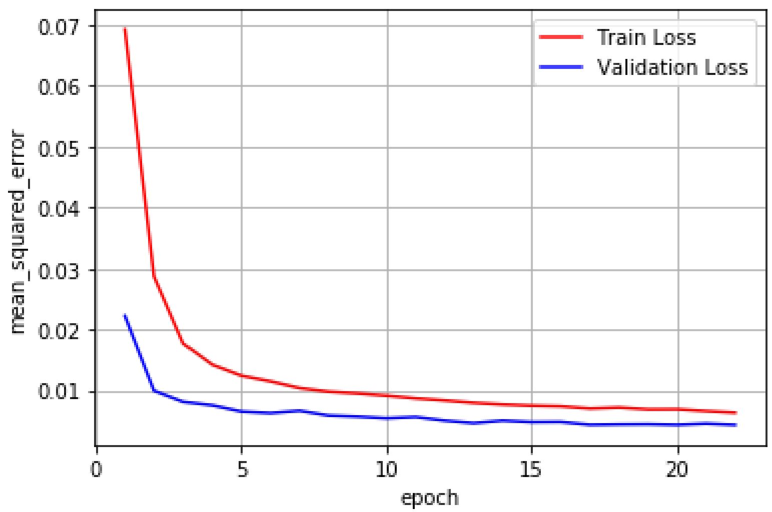
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('mean_squared_error')

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

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

```
Test score: 0.004310477123901086
Test accuracy: 0.9756
```



In [78]:

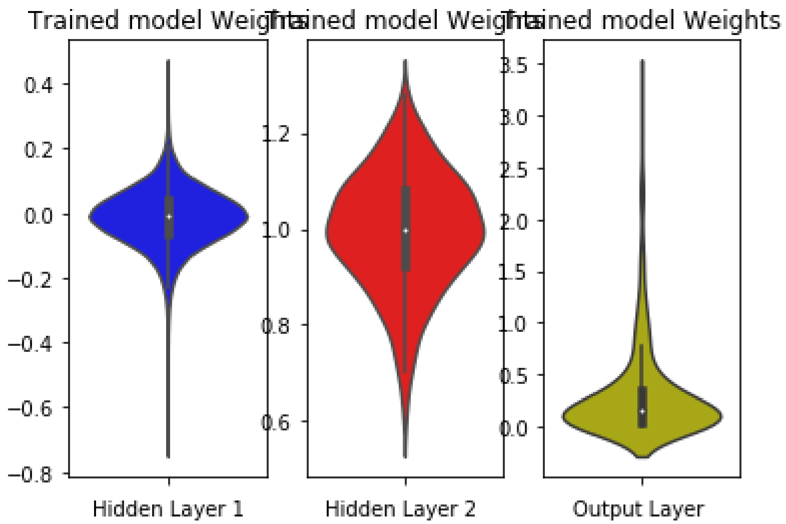
```
%%time
w_after = model_relu.get_weights()

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

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

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

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



Summary:

In [81]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Activation", "optimizer", "BatchNormalization","LossFunction","Dropout","Hidden Layers","Test score","Test 
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.5", "2-(HiddenLayer)[430,322]","0.051","0.99"])
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.5", "3-(HiddenLayer)[320,270,162]","0.061","0.98"])
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.5", "5-(HiddenLayer)[340,190,112,72,38]","0.079","0.98"
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.25", "2-(HiddenLayer)[430,322]","0.061","0.98"])
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.25", "3-(HiddenLayer)[320,270,162]","0.063","0.98"])
pt.add_row(["relu", "adam", "Yes", "Categorical Crossentropy Loss", "0.25", "5-(HiddenLayer)[340,190,112,72,38]","0.070","0.98
pt.add_row(["relu", "adam", "Yes", "mean_squared_error", "0.5", "2-(HiddenLayer)[430,322]","0.003","0.98"])
pt.add_row(["relu", "adam", "Yes", "mean_squared_error", "0.5", "3-(HiddenLayer)[320,270,162]","0.003","0.98"])
pt.add_row(["relu", "adam", "Yes", "mean_squared_error", "0.5", "5-(HiddenLayer)[340,190,112,72,38]","0.004","0.98"])
print(pt)
```

```
+------------+-----------+--------------------+------------------------------+---------+------------------------------------+---
---------+---------------+
| Activation | optimizer | BatchNormalization |         LossFunction         | Dropout |           Hidden Layers            | Te
st score | Test accuracy |
+------------+-----------+--------------------+------------------------------+---------+------------------------------------+---
---------+---------------+
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.5   |      2-(HiddenLayer)[430,322]      |
0.051    |      0.99     |
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.5   |    3-(HiddenLayer)[320,270,162]    |
0.061    |      0.98     |
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.5   | 5-(HiddenLayer)[340,190,112,72,38] |
0.079    |      0.98     |
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.25  |      2-(HiddenLayer)[430,322]      |
0.061    |      0.98     |
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.25  |    3-(HiddenLayer)[320,270,162]    |
0.063    |      0.98     |
|    relu    |    adam   |        Yes         | Categorical Crossentropy Loss |   0.25  | 5-(HiddenLayer)[340,190,112,72,38] |
0.070    |      0.98     |
|    relu    |    adam   |        Yes         |      mean_squared_error      |   0.5   |      2-(HiddenLayer)[430,322]      |
0.003    |      0.98     |
|    relu    |    adam   |        Yes         |      mean_squared_error      |   0.5   |    3-(HiddenLayer)[320,270,162]    |
0.003    |      0.98     |
|    relu    |    adam   |        Yes         |      mean_squared_error      |   0.5   | 5-(HiddenLayer)[340,190,112,72,38] |
0.004    |      0.98     |
+------------+-----------+--------------------+------------------------------+---------+------------------------------------+---
---------+---------------+
```

Conclusion:

1. From the above model we could see the Dropout (0.5) workes well when compare to the Dropout(0.25).

2. Model test score and accuracy are good when we used the 0.5 as Dropout and Categorical Crossentropy as loss function.
3. Overall model perform well with Dropout 0.5 and with Epoch 20.For graphical representation i have increased the epoch value to 22 and also we see model get overfitting when epoch value incresed more than 35.