In [1]:

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

```
Using TensorFlow backend.
```

In [2]:

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

In [3]:

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

In [4]:

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

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

In [5]:

```
# 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 * 78

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

In [6]:

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

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

```
Number of training examples : 60000 and each image is of shape
(784)
Number of test examples : 10000 and each image is of shape (78
4)
```

In [7]:

```
# 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   3  18  18  18 126 136 17
5  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  3
0  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 25
3 251  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  1
1   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 25
3 190   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  2
5   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  9
3 252 253 187
   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  4
6 130 183 253
 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 22
1 253 253 253
 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 25
3 253 253 195
  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 21
2 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:

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

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

In [9]:

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

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

In [10]:

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

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

In [11]:

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

In [12]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [13]:

```
X_train
```

Out[13]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Model 1 : 2 Hidden Layers with ReLU Activation and Adam Optimizer

In [13]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
```

```
Model: "sequential_1"
_________________________________________________________________
_____
Layer (type)                 Output Shape              Para
m #
=================================================================
=====
dense_1 (Dense)              (None, 512)               4019
20
_________________________________________________________________
_____
dense_2 (Dense)              (None, 256)               1313
28
_________________________________________________________________
_____
dense_3 (Dense)              (None, 10)                2570
=================================================================
=====
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
_____
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 28us/step
- loss: 0.2222 - accuracy: 0.9332 - val_loss: 0.1094 - val_
accuracy: 0.9663
Epoch 2/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0817 - accuracy: 0.9753 - val_loss: 0.0779 - val_
accuracy: 0.9762
Epoch 3/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0501 - accuracy: 0.9847 - val_loss: 0.0806 - val_
accuracy: 0.9745
Epoch 4/20
```

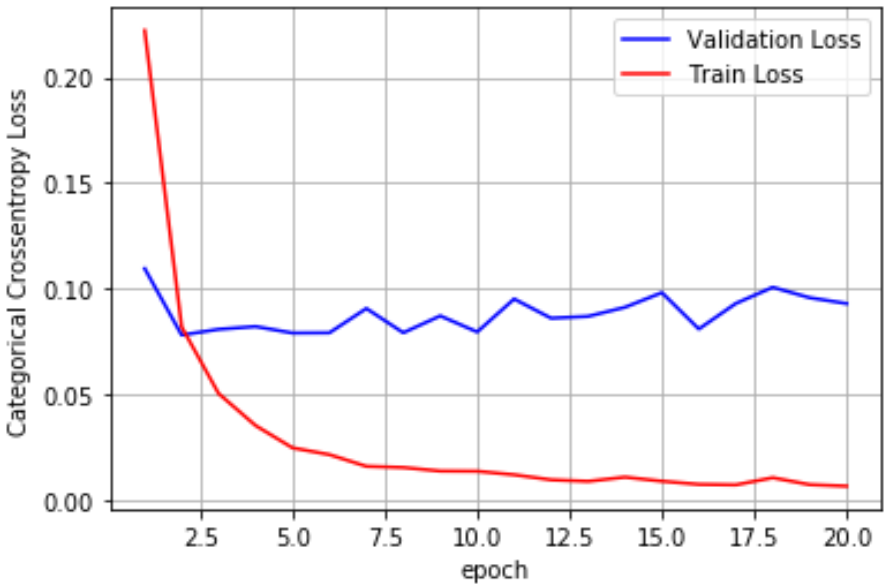
```
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0350 - accuracy: 0.9888 - val_loss: 0.0819 - val_
accuracy: 0.9774
Epoch 5/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0244 - accuracy: 0.9923 - val_loss: 0.0788 - val_
accuracy: 0.9783
Epoch 6/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0212 - accuracy: 0.9931 - val_loss: 0.0789 - val_
accuracy: 0.9774
Epoch 7/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0157 - accuracy: 0.9950 - val_loss: 0.0904 - val_
accuracy: 0.9755
Epoch 8/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0151 - accuracy: 0.9951 - val_loss: 0.0789 - val_
accuracy: 0.9794
Epoch 9/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0134 - accuracy: 0.9955 - val_loss: 0.0869 - val_
accuracy: 0.9785
Epoch 10/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0133 - accuracy: 0.9955 - val_loss: 0.0793 - val_
accuracy: 0.9814
Epoch 11/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0117 - accuracy: 0.9962 - val_loss: 0.0950 - val_
accuracy: 0.9779
Epoch 12/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0093 - accuracy: 0.9967 - val_loss: 0.0858 - val_
accuracy: 0.9806
Epoch 13/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0086 - accuracy: 0.9972 - val_loss: 0.0867 - val_
accuracy: 0.9803
Epoch 14/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0105 - accuracy: 0.9965 - val_loss: 0.0910 - val_
accuracy: 0.9808
Epoch 15/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0086 - accuracy: 0.9971 - val_loss: 0.0979 - val_
accuracy: 0.9803
Epoch 16/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0071 - accuracy: 0.9975 - val_loss: 0.0807 - val_
accuracy: 0.9818
Epoch 17/20
60000/60000 [==============================] - 1s 23us/step
```

```
- loss: 0.0069 - accuracy: 0.9976 - val_loss: 0.0928 - val_
accuracy: 0.9823
Epoch 18/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0103 - accuracy: 0.9969 - val_loss: 0.1005 - val_
accuracy: 0.9811
Epoch 19/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0070 - accuracy: 0.9980 - val_loss: 0.0955 - val_
accuracy: 0.9820
Epoch 20/20
60000/60000 [==============================] - 1s 23us/step
- loss: 0.0063 - accuracy: 0.9980 - val_loss: 0.0928 - val_
accuracy: 0.9815
```

In [14]:

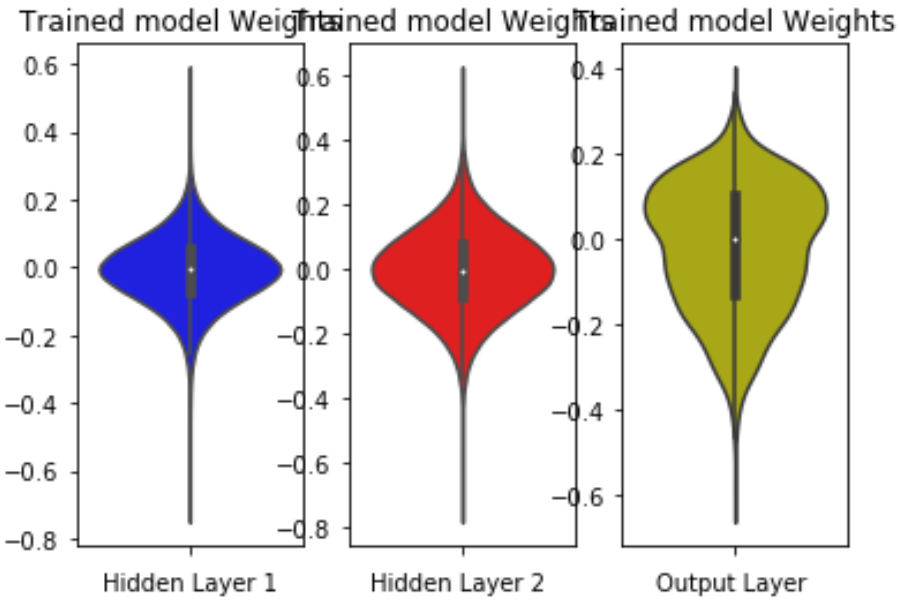
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score1 = score[0]
accuracy1 = score[1]
print('Test score:', score1)
print('Test accuracy:', accuracy1)

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

# we will get val_loss and val_acc only when you pass the paramter validation
# 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 nu


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

```
Test score: 0.0927800526930005
Test accuracy: 0.9815000295639038
```



In [15]:

```
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()
```



Model 2 : 3 Hidden Layers with ReLU Activation and Adam Optimizer

In [16]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
```

```
Model: "sequential_2"
_________________________________________________________________
_____
Layer (type)                 Output Shape              Para
m #
=================================================================
======
dense_4 (Dense)              (None, 512)               4019
20
_________________________________________________________________
_____
dense_5 (Dense)              (None, 256)               1313
28
_________________________________________________________________
_____
dense_6 (Dense)              (None, 128)               3289
6
_________________________________________________________________
_____
dense_7 (Dense)              (None, 10)                1290
=================================================================
======
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
_____
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 27us/step
- loss: 0.2082 - accuracy: 0.9378 - val_loss: 0.1011 - val_
accuracy: 0.9691
Epoch 2/20
60000/60000 [==============================] - 2s 25us/step
- loss: 0.0819 - accuracy: 0.9749 - val_loss: 0.0932 - val_
```

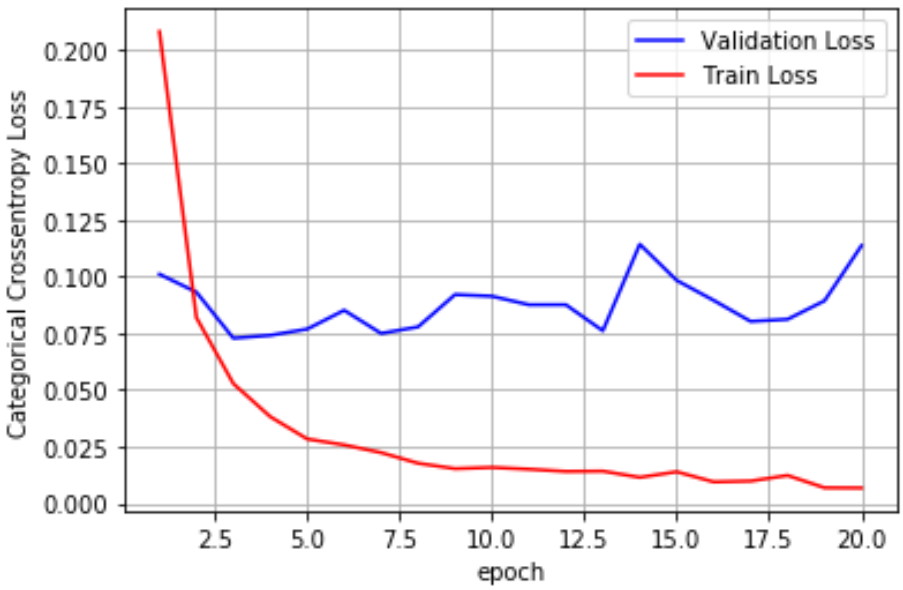
```
accuracy: 0.9710
Epoch 3/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0528 - accuracy: 0.9829 - val_loss: 0.0729 - val_
accuracy: 0.9757
Epoch 4/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0382 - accuracy: 0.9873 - val_loss: 0.0741 - val_
accuracy: 0.9782
Epoch 5/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0284 - accuracy: 0.9906 - val_loss: 0.0769 - val_
accuracy: 0.9768
Epoch 6/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0258 - accuracy: 0.9916 - val_loss: 0.0852 - val_
accuracy: 0.9769
Epoch 7/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0224 - accuracy: 0.9927 - val_loss: 0.0750 - val_
accuracy: 0.9792
Epoch 8/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0177 - accuracy: 0.9942 - val_loss: 0.0778 - val_
accuracy: 0.9795
Epoch 9/20
60000/60000 [==============================] - 1s 25us/step
- loss: 0.0153 - accuracy: 0.9951 - val_loss: 0.0921 - val_
accuracy: 0.9762
Epoch 10/20
60000/60000 [==============================] - 1s 25us/step
- loss: 0.0158 - accuracy: 0.9945 - val_loss: 0.0914 - val_
accuracy: 0.9770
Epoch 11/20
60000/60000 [==============================] - 2s 25us/step
- loss: 0.0150 - accuracy: 0.9948 - val_loss: 0.0876 - val_
accuracy: 0.9796
Epoch 12/20
60000/60000 [==============================] - 2s 26us/step
- loss: 0.0141 - accuracy: 0.9953 - val_loss: 0.0876 - val_
accuracy: 0.9811
Epoch 13/20
60000/60000 [==============================] - 1s 25us/step
- loss: 0.0142 - accuracy: 0.9955 - val_loss: 0.0761 - val_
accuracy: 0.9832
Epoch 14/20
60000/60000 [==============================] - 2s 26us/step
- loss: 0.0114 - accuracy: 0.9964 - val_loss: 0.1143 - val_
accuracy: 0.9760
Epoch 15/20
60000/60000 [==============================] - 1s 24us/step
- loss: 0.0139 - accuracy: 0.9954 - val_loss: 0.0984 - val_
accuracy: 0.9802
```

```
Epoch 16/20
60000/60000 [==============================] - 1s 25us/step
- loss: 0.0095 - accuracy: 0.9967 - val_loss: 0.0895 - val_
accuracy: 0.9825
Epoch 17/20
60000/60000 [==============================] - 2s 26us/step
- loss: 0.0098 - accuracy: 0.9973 - val_loss: 0.0803 - val_
accuracy: 0.9837
Epoch 18/20
60000/60000 [==============================] - 2s 26us/step
- loss: 0.0123 - accuracy: 0.9965 - val_loss: 0.0812 - val_
accuracy: 0.9823
Epoch 19/20
60000/60000 [==============================] - 2s 25us/step
- loss: 0.0069 - accuracy: 0.9980 - val_loss: 0.0894 - val_
accuracy: 0.9821
Epoch 20/20
60000/60000 [==============================] - 1s 25us/step
- loss: 0.0068 - accuracy: 0.9977 - val_loss: 0.1137 - val_
accuracy: 0.9787
```

In [17]:

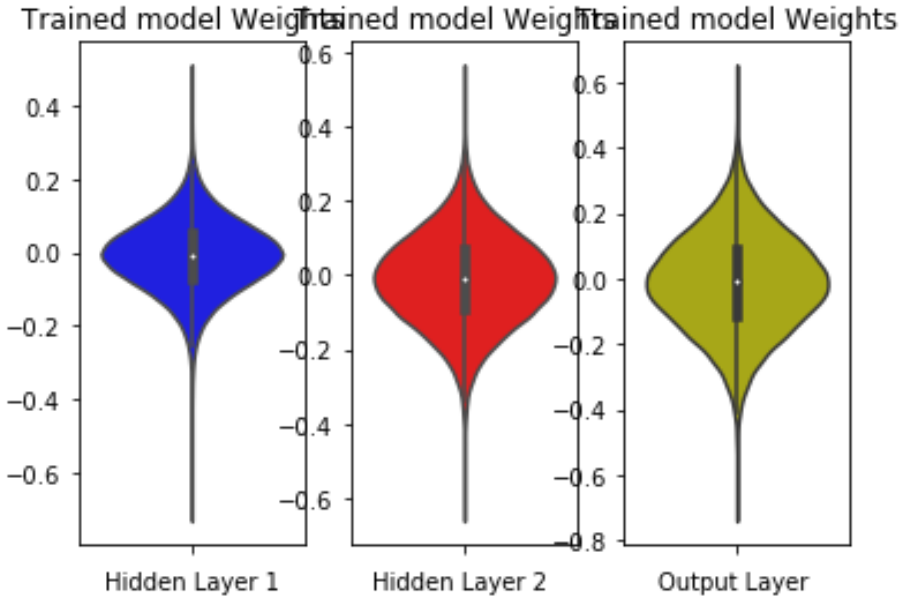
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score2 = score[0]
accuracy2 = score[1]
print('Test score:', score2)
print('Test accuracy:', accuracy2)

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

x = list(range(1,nb_epoch+1))

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

```
Test score: 0.11367766052724149
Test accuracy: 0.9786999821662903
```



In [18]:

```
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()
```



Model 3 : 5 Hidden Layers with ReLU Activation and Adam Optimizer

In [19]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(r
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(r
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_e
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_9 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_10 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_12 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 31us/step -
loss: 0.2693 - accuracy: 0.9196 - val_loss: 0.1208 - val_accur
acy: 0.9629
```

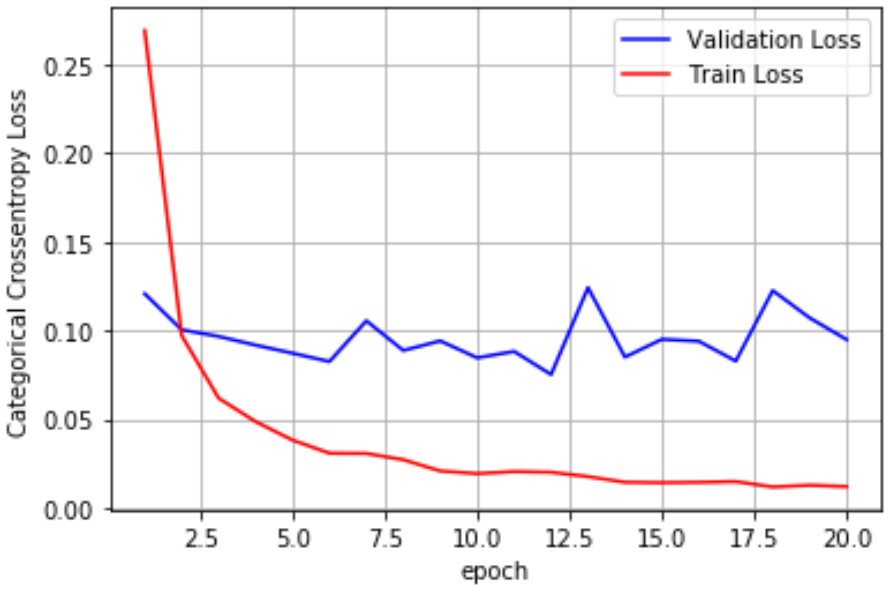
```
Epoch 2/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0971 - accuracy: 0.9702 - val_loss: 0.1004 - val_accur
acy: 0.9688
Epoch 3/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0618 - accuracy: 0.9812 - val_loss: 0.0966 - val_accur
acy: 0.9717
Epoch 4/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0486 - accuracy: 0.9838 - val_loss: 0.0918 - val_accur
acy: 0.9723
Epoch 5/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0381 - accuracy: 0.9878 - val_loss: 0.0872 - val_accur
acy: 0.9737
Epoch 6/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0307 - accuracy: 0.9903 - val_loss: 0.0825 - val_accur
acy: 0.9787
Epoch 7/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0306 - accuracy: 0.9905 - val_loss: 0.1055 - val_accur
acy: 0.9725
Epoch 8/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0271 - accuracy: 0.9913 - val_loss: 0.0887 - val_accur
acy: 0.9759
Epoch 9/20
60000/60000 [==============================] - 2s 31us/step -
loss: 0.0207 - accuracy: 0.9932 - val_loss: 0.0941 - val_accur
acy: 0.9771
Epoch 10/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0192 - accuracy: 0.9939 - val_loss: 0.0846 - val_accur
acy: 0.9803
Epoch 11/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0204 - accuracy: 0.9936 - val_loss: 0.0882 - val_accur
acy: 0.9793
Epoch 12/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0200 - accuracy: 0.9935 - val_loss: 0.0752 - val_accur
acy: 0.9814
Epoch 13/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0175 - accuracy: 0.9947 - val_loss: 0.1242 - val_accur
acy: 0.9725
Epoch 14/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0143 - accuracy: 0.9954 - val_loss: 0.0850 - val_accur
acy: 0.9826
Epoch 15/20
```

```
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0141 - accuracy: 0.9955 - val_loss: 0.0951 - val_accur
acy: 0.9770
Epoch 16/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0143 - accuracy: 0.9958 - val_loss: 0.0940 - val_accur
acy: 0.9797
Epoch 17/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0147 - accuracy: 0.9954 - val_loss: 0.0828 - val_accur
acy: 0.9807
Epoch 18/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0117 - accuracy: 0.9962 - val_loss: 0.1226 - val_accur
acy: 0.9735
Epoch 19/20
60000/60000 [==============================] - 2s 28us/step -
loss: 0.0127 - accuracy: 0.9963 - val_loss: 0.1071 - val_accur
acy: 0.9783
Epoch 20/20
60000/60000 [==============================] - 2s 29us/step -
loss: 0.0119 - accuracy: 0.9962 - val_loss: 0.0949 - val_accur
acy: 0.9805
```

In [20]:

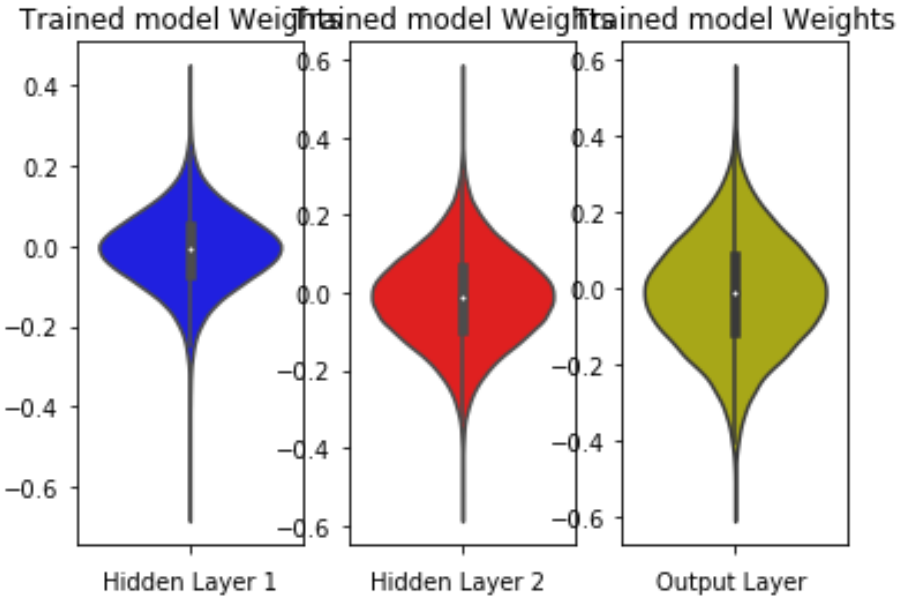
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
score3 = score[0]
accuracy3 = score[1]
print('Test score:', score3)
print('Test accuracy:', accuracy3)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,nb_epoch+1))

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

```
Test score: 0.09488223801337149
Test accuracy: 0.9804999828338623
```



In [21]:

```
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()
```



Model 4 : 2 Hidden Layers with ReLU Activation, Adam Optimizer, Dropout & Batch Normalization

In [22]:

```
from keras.layers import Dropout
from keras.layers import BatchNormalization

model_relu_dor = Sequential()

#Layer 1
model_relu_dor.add(Dense(512, activation='relu', input_shape=(input_dim,), ke
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 2
model_relu_dor.add(Dense(256, activation='relu', kernel_initializer=RandomNor
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))

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

print(model_relu_dor.summary())

model_relu_dor.compile(optimizer='adam', loss='categorical_crossentropy', met

history = model_relu_dor.fit(X_train, Y_train, batch_size=batch_size, epochs=
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                2570
=================================================================
```

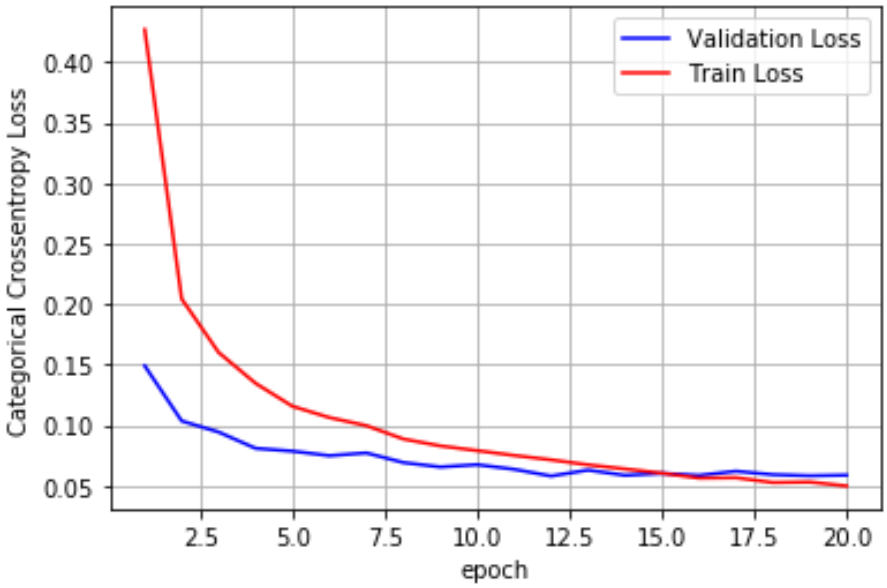
```
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
___
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 48us/step -
loss: 0.4269 - accuracy: 0.8707 - val_loss: 0.1491 - val_accur
acy: 0.9530
Epoch 2/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.2044 - accuracy: 0.9386 - val_loss: 0.1035 - val_accur
acy: 0.9677
Epoch 3/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.1602 - accuracy: 0.9509 - val_loss: 0.0944 - val_accur
acy: 0.9716
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.1347 - accuracy: 0.9578 - val_loss: 0.0809 - val_accur
acy: 0.9756
Epoch 5/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.1157 - accuracy: 0.9636 - val_loss: 0.0786 - val_accur
acy: 0.9772
Epoch 6/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.1063 - accuracy: 0.9667 - val_loss: 0.0749 - val_accur
acy: 0.9779
Epoch 7/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.0997 - accuracy: 0.9702 - val_loss: 0.0771 - val_accur
acy: 0.9759
Epoch 8/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0886 - accuracy: 0.9723 - val_loss: 0.0692 - val_accur
acy: 0.9789
Epoch 9/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0829 - accuracy: 0.9737 - val_loss: 0.0655 - val_accur
acy: 0.9812
Epoch 10/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.0790 - accuracy: 0.9748 - val_loss: 0.0674 - val_accur
acy: 0.9795
Epoch 11/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0751 - accuracy: 0.9764 - val_loss: 0.0636 - val_accur
acy: 0.9810
Epoch 12/20
60000/60000 [==============================] - 2s 40us/step -
```

```
loss: 0.0714 - accuracy: 0.9772 - val_loss: 0.0580 - val_accur
acy: 0.9829
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.0673 - accuracy: 0.9788 - val_loss: 0.0629 - val_accur
acy: 0.9810
Epoch 14/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0639 - accuracy: 0.9792 - val_loss: 0.0586 - val_accur
acy: 0.9829
Epoch 15/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0604 - accuracy: 0.9803 - val_loss: 0.0603 - val_accur
acy: 0.9807
Epoch 16/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.0563 - accuracy: 0.9816 - val_loss: 0.0585 - val_accur
acy: 0.9827
Epoch 17/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0567 - accuracy: 0.9817 - val_loss: 0.0620 - val_accur
acy: 0.9825
Epoch 18/20
60000/60000 [==============================] - 2s 41us/step -
loss: 0.0527 - accuracy: 0.9829 - val_loss: 0.0592 - val_accur
acy: 0.9833
Epoch 19/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0531 - accuracy: 0.9823 - val_loss: 0.0583 - val_accur
acy: 0.9829
Epoch 20/20
60000/60000 [==============================] - 2s 40us/step -
loss: 0.0498 - accuracy: 0.9837 - val_loss: 0.0588 - val_accur
acy: 0.9839
```

In [23]:

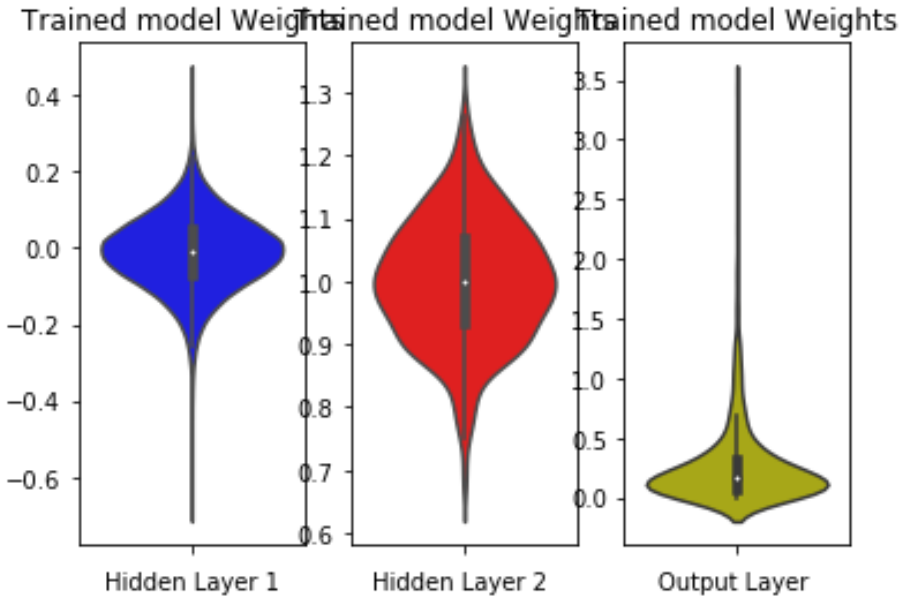
```
score = model_relu_dor.evaluate(X_test, Y_test, verbose=0)
score11 = score[0]
accuracy11 = score[1]
print('Test score:', score11)
print('Test accuracy:', accuracy11)

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

x = list(range(1,nb_epoch+1))

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

```
Test score: 0.05877195674158866
Test accuracy: 0.9839000105857849
```



In [24]:

```
w_after = model_relu_dor.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()
```



Model 5 : 3 Hidden Layers with ReLU Activation, Adam Optimizer, Dropout & Batch Normalization

In [25]:

```
model_relu_dor = Sequential()
#Layer 1
model_relu_dor.add(Dense(512, activation='relu', input_shape=(input_dim,), ke
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 2
model_relu_dor.add(Dense(256, activation='relu', kernel_initializer=RandomNor
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 3
model_relu_dor.add(Dense(128, activation='relu', kernel_initializer=RandomNor
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))

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

print(model_relu_dor.summary())

model_relu_dor.compile(optimizer='adam', loss='categorical_crossentropy', met

history = model_relu_dor.fit(X_train, Y_train, batch_size=batch_size, epochs=
```

```
Model: "sequential_5"
_________________________________________________________________
_______
Layer (type)                 Output Shape              Para
m #
=================================================================
======
dense_17 (Dense)             (None, 512)               4019
20
_________________________________________________________________
_______
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
_______
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
_______
dense_18 (Dense)             (None, 256)               1313
28
_________________________________________________________________
_______
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
_______
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
```

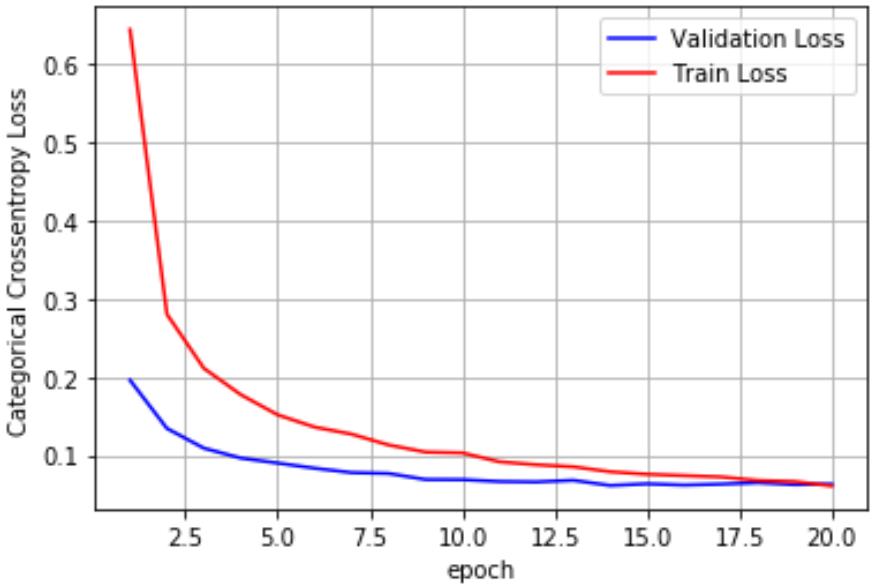
```
_______
dense_19 (Dense)             (None, 128)               3289
6
_________________________________________________________
_______
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________
_______
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________
_______
dense_20 (Dense)             (None, 10)                1290
=========================================================
======
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________
_______
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 61us/step
- loss: 0.6434 - accuracy: 0.8034 - val_loss: 0.1959 - val_
accuracy: 0.9407
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.2801 - accuracy: 0.9179 - val_loss: 0.1344 - val_
accuracy: 0.9579
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.2110 - accuracy: 0.9384 - val_loss: 0.1091 - val_
accuracy: 0.9662
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1774 - accuracy: 0.9482 - val_loss: 0.0965 - val_
accuracy: 0.9706
Epoch 5/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1516 - accuracy: 0.9552 - val_loss: 0.0901 - val_
accuracy: 0.9726
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1360 - accuracy: 0.9593 - val_loss: 0.0835 - val_
accuracy: 0.9739
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1270 - accuracy: 0.9624 - val_loss: 0.0780 - val_
accuracy: 0.9765
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1132 - accuracy: 0.9664 - val_loss: 0.0768 - val_
accuracy: 0.9779
```

```
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.1042 - accuracy: 0.9683 - val_loss: 0.0692 - val_
accuracy: 0.9802
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.1031 - accuracy: 0.9689 - val_loss: 0.0690 - val_
accuracy: 0.9788
Epoch 11/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0918 - accuracy: 0.9722 - val_loss: 0.0666 - val_
accuracy: 0.9804
Epoch 12/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.0880 - accuracy: 0.9729 - val_loss: 0.0661 - val_
accuracy: 0.9799
Epoch 13/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0854 - accuracy: 0.9738 - val_loss: 0.0681 - val_
accuracy: 0.9813
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0791 - accuracy: 0.9754 - val_loss: 0.0615 - val_
accuracy: 0.9823
Epoch 15/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.0759 - accuracy: 0.9770 - val_loss: 0.0636 - val_
accuracy: 0.9810
Epoch 16/20
60000/60000 [==============================] - 3s 51us/step
- loss: 0.0741 - accuracy: 0.9772 - val_loss: 0.0621 - val_
accuracy: 0.9827
Epoch 17/20
60000/60000 [==============================] - 3s 53us/step
- loss: 0.0724 - accuracy: 0.9781 - val_loss: 0.0632 - val_
accuracy: 0.9829
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0679 - accuracy: 0.9793 - val_loss: 0.0650 - val_
accuracy: 0.9815
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step
- loss: 0.0665 - accuracy: 0.9803 - val_loss: 0.0626 - val_
accuracy: 0.9831
Epoch 20/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0609 - accuracy: 0.9815 - val_loss: 0.0632 - val_
accuracy: 0.9828
```

In [26]:

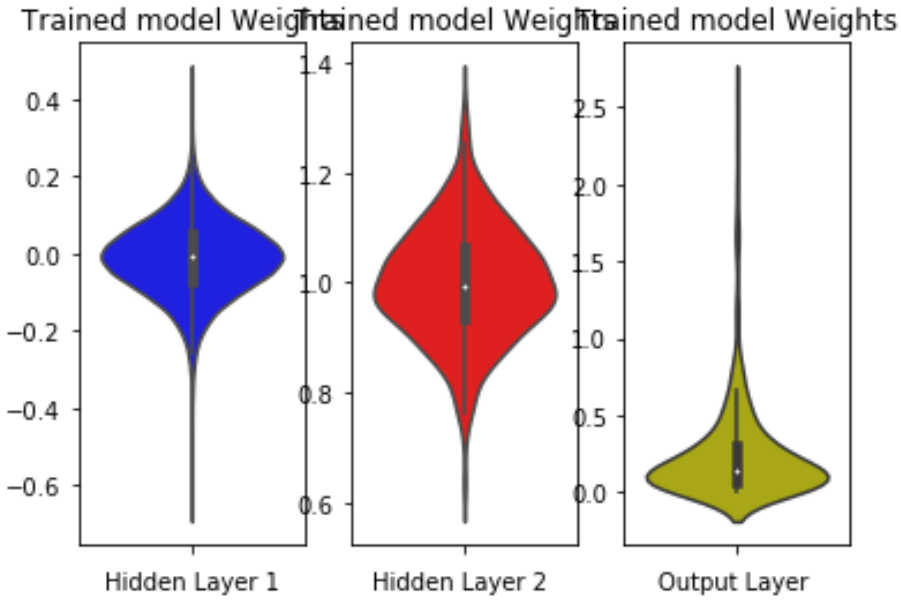
```
score = model_relu_dor.evaluate(X_test, Y_test, verbose=0)
score22 = score[0]
accuracy22 = score[1]
print('Test score:', score22)
print('Test accuracy:', accuracy22)

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

x = list(range(1,nb_epoch+1))

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

```
Test score: 0.06324021853187588
Test accuracy: 0.9828000068664551
```



In [27]:

```
w_after = model_relu_dor.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()
```



Model 6 : 5 Hidden Layers with ReLU Activation, Adam Optimizer, Dropout & Batch Normalization

In [28]:

```
model_relu_dor = Sequential()
#Layer 1
model_relu_dor.add(Dense(512, activation='relu', input_shape=(input_dim,), ke
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 2
model_relu_dor.add(Dense(256, activation='relu', kernel_initializer=RandomNor
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 3
model_relu_dor.add(Dense(128, activation='relu', kernel_initializer=RandomNor
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 4
model_relu_dor.add(Dense(64, activation='relu', kernel_initializer=RandomNorm
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
#Layer 5
model_relu_dor.add(Dense(32, activation='relu', kernel_initializer=RandomNorm
model_relu_dor.add(BatchNormalization())
model_relu_dor.add(Dropout(0.5))
model_relu_dor.add(Dense(output_dim, activation='softmax'))

print(model_relu_dor.summary())

model_relu_dor.compile(optimizer='adam', loss='categorical_crossentropy', met

history = model_relu_dor.fit(X_train, Y_train, batch_size=batch_size, epochs=
```

```
Model: "sequential_6"
_________________________________________________________________
_______
Layer (type)                 Output Shape              Para
m #
=================================================================
======
dense_21 (Dense)             (None, 512)               4019
20
_________________________________________________________________
_______
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
_______
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
_______
dense_22 (Dense)             (None, 256)               1313
28
_________________________________________________________________
```

```
_______
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
_______
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
_______
dense_23 (Dense)             (None, 128)               3289
6
_________________________________________________________________
_______
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
_______
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
_______
dense_24 (Dense)             (None, 64)                8256
_________________________________________________________________
_______
batch_normalization_9 (Batch (None, 64)                256
_________________________________________________________________
_______
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
_______
dense_25 (Dense)             (None, 32)                2080
_________________________________________________________________
_______
batch_normalization_10 (Batc (None, 32)                128
_________________________________________________________________
_______
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
_______
dense_26 (Dense)             (None, 10)                330
=================================================================
======
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
_______
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 89us/step
- loss: 1.6273 - accuracy: 0.4693 - val_loss: 0.4213 - val_
accuracy: 0.8927
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.6966 - accuracy: 0.7912 - val_loss: 0.2432 - val_
accuracy: 0.9333
```

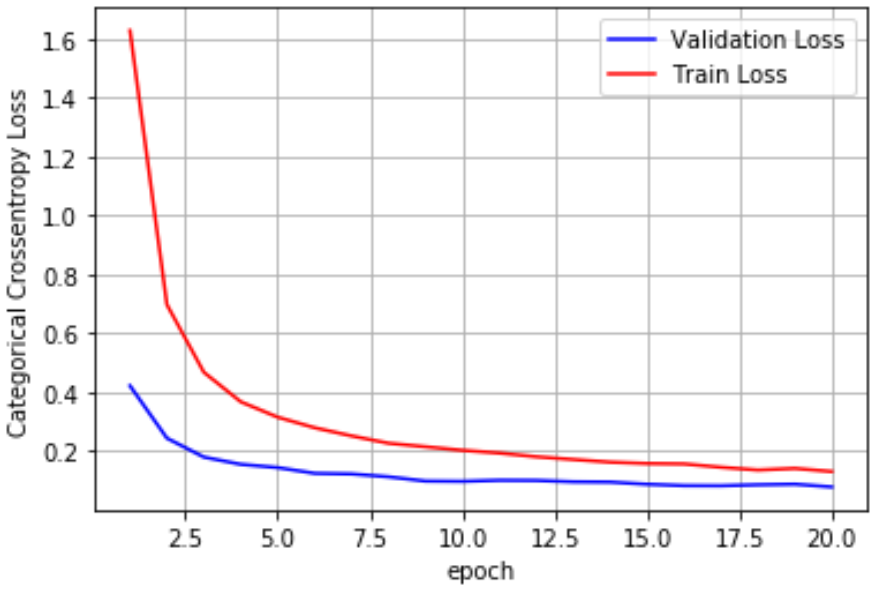
```
Epoch 3/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.4670 - accuracy: 0.8733 - val_loss: 0.1790 - val_
accuracy: 0.9527
Epoch 4/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.3667 - accuracy: 0.9060 - val_loss: 0.1544 - val_
accuracy: 0.9605
Epoch 5/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.3145 - accuracy: 0.9248 - val_loss: 0.1433 - val_
accuracy: 0.9627
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.2786 - accuracy: 0.9340 - val_loss: 0.1239 - val_
accuracy: 0.9686
Epoch 7/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.2509 - accuracy: 0.9416 - val_loss: 0.1218 - val_
accuracy: 0.9704
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.2260 - accuracy: 0.9463 - val_loss: 0.1120 - val_
accuracy: 0.9734
Epoch 9/20
60000/60000 [==============================] - 4s 75us/step
- loss: 0.2141 - accuracy: 0.9512 - val_loss: 0.0975 - val_
accuracy: 0.9764
Epoch 10/20
60000/60000 [==============================] - 4s 74us/step
- loss: 0.2020 - accuracy: 0.9541 - val_loss: 0.0966 - val_
accuracy: 0.9764
Epoch 11/20
60000/60000 [==============================] - 5s 75us/step
- loss: 0.1927 - accuracy: 0.9563 - val_loss: 0.1003 - val_
accuracy: 0.9761
Epoch 12/20
60000/60000 [==============================] - 4s 75us/step
- loss: 0.1801 - accuracy: 0.9585 - val_loss: 0.0997 - val_
accuracy: 0.9769
Epoch 13/20
60000/60000 [==============================] - 5s 76us/step
- loss: 0.1709 - accuracy: 0.9610 - val_loss: 0.0951 - val_
accuracy: 0.9771
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1618 - accuracy: 0.9636 - val_loss: 0.0942 - val_
accuracy: 0.9777
Epoch 15/20
60000/60000 [==============================] - 4s 74us/step
- loss: 0.1573 - accuracy: 0.9651 - val_loss: 0.0868 - val_
accuracy: 0.9799
Epoch 16/20
```

```
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1558 - accuracy: 0.9649 - val_loss: 0.0821 - val_
accuracy: 0.9803
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1444 - accuracy: 0.9675 - val_loss: 0.0817 - val_
accuracy: 0.9813
Epoch 18/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1351 - accuracy: 0.9700 - val_loss: 0.0849 - val_
accuracy: 0.9804
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1405 - accuracy: 0.9683 - val_loss: 0.0871 - val_
accuracy: 0.9799
Epoch 20/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.1301 - accuracy: 0.9710 - val_loss: 0.0776 - val_
accuracy: 0.9815
```

In [29]:

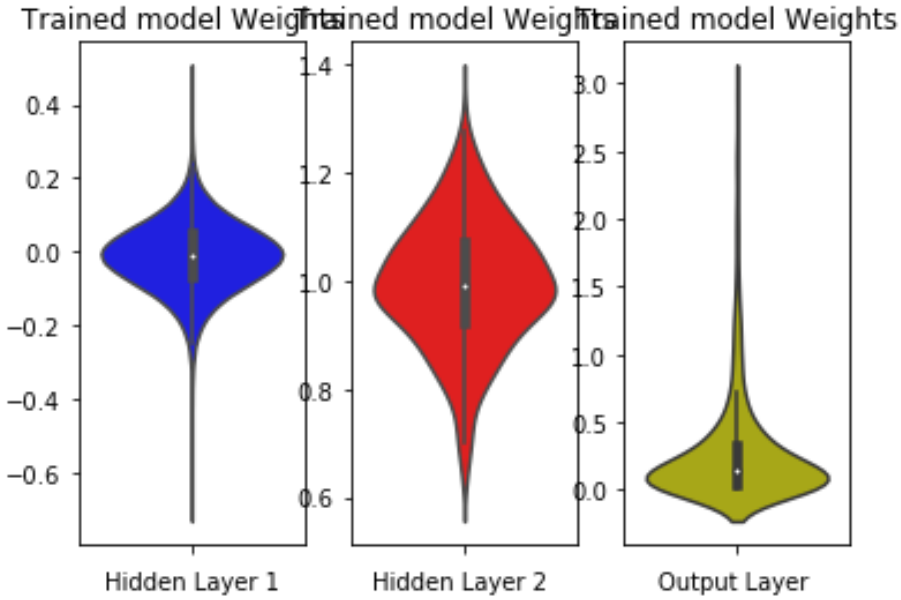
```
score = model_relu_dor.evaluate(X_test, Y_test, verbose=0)
score33 = score[0]
accuracy33 = score[1]
print('Test score:', score33)
print('Test accuracy:', accuracy33)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,nb_epoch+1))

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

```
Test score: 0.0776194792546332
Test accuracy: 0.9815000295639038
```



In [30]:

```
w_after = model_relu_dor.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()
```



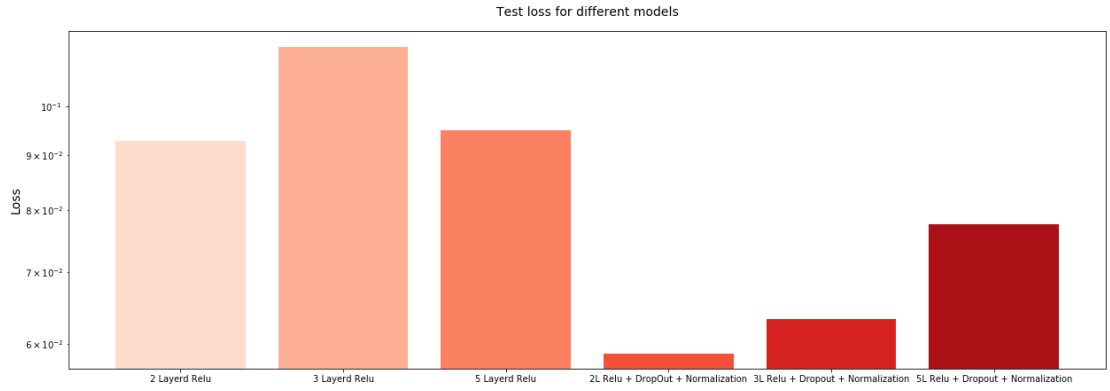
In [31]:

```
import seaborn as sns
models = ['2 Layerd Relu', '3 Layerd Relu', '5 Layerd Relu', '2L Relu + Drop(
scores = [score1, score2, score3, score11, score22, score33]
accuracies = [accuracy1, accuracy2, accuracy3, accuracy11, accuracy22, accura
```

Vizualizing Results

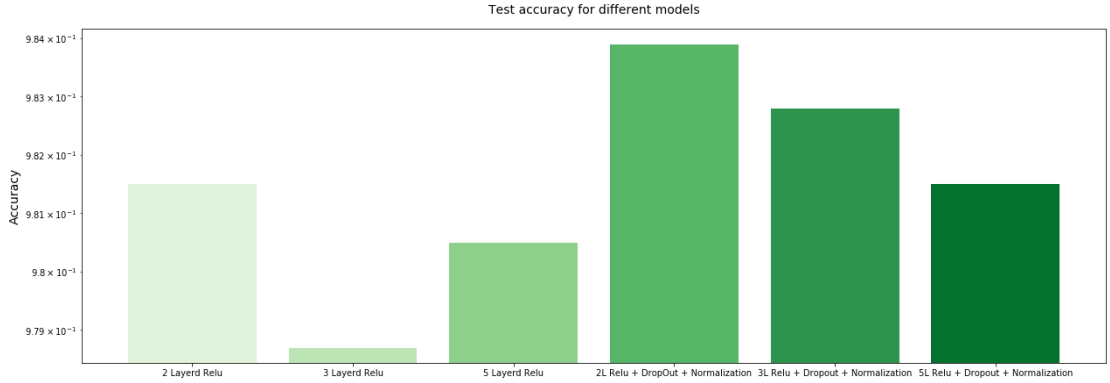
In [32]:

```
plt.figure(figsize=(21,7))
fig = plt.bar(models, scores, color=sns.color_palette("Reds",6))
plt.ylabel('Loss', size = 14)
plt.yscale('log')
plt.title('Test loss for different models', size = 14, y = 1.03)
plt.show()
```



In [33]:

```
plt.figure(figsize=(21,7))
fig = plt.bar(models, accuracies, color=sns.color_palette("Greens",6))
plt.ylabel('Accuracy', size = 14)
plt.yscale('log')
plt.title('Test accuracy for different models', size = 14, y = 1.03)
plt.show()
```



Conclusion

In [34]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model", "Loss", "Accuracy"]

table.add_row([models[0], score1, accuracy1])
table.add_row([models[1], score2, accuracy2])
table.add_row([models[2], score3, accuracy3])
table.add_row([models[3], score11, accuracy11])
table.add_row([models[4], score22, accuracy22])
table.add_row([models[5], score33, accuracy33])
print(table)
```

```
+-----------------------------------+---------------------+---
-----------------+
|               Model               |         Loss        |
Accuracy      |
+-----------------------------------+---------------------+---
-----------------+
|           2 Layerd Relu           |  0.0927800526930005 | 0.
9815000295639038 |
|           3 Layerd Relu           | 0.11367766052724149 | 0.
9786999821662903 |
|           5 Layerd Relu           | 0.09488223801337149 | 0.
9804999828338623 |
| 2L Relu + DropOut + Normalization | 0.05877195674158866 | 0.
9839000105857849 |
| 3L Relu + Dropout + Normalization | 0.06324021853187588 | 0.
9828000068664551 |
| 5L Relu + Dropout + Normalization |  0.0776194792546332 | 0.
9815000295639038 |
+-----------------------------------+---------------------+---
-----------------+
```

Summary

- 2 Hidden Layered ReLU Activation with Adam Optimizer, Dropout & Batch Normalization has the best Accuracy among all 6 Architectures
- Architectures with Dropout & Batch Normalization has less test loss when compared with architectures without Dropout & Batch Normalization