Implementing differrent MLP architectures

1. 2-Layer MLP architecture
2. 3-Layer MLP architecture
3. 5-Layer MLP architecture

- All above architectures are implemented with Dropout and Batch-Normalization layer ini between

In [2]:
```
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

# For plotting purposes
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.utils import to_categorical
from keras.models import Sequential
from keras.initializers import he_normal
from keras.layers import BatchNormalization, Dense, Dropout
from keras.utils import np_utils
from keras.initializers import RandomNormal
# Import MNIST Dataset
from keras.datasets import mnist
```

Loading the MNIST data and printing the input shape

In [3]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

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

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

Transforming the input data into 1*784

In [4]:
```
#Converting the input into a One dimensional vector (1*784)

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

#Printing the values of each image shape.
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d)"%(X_train.sh
ape[1]))
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_test.shap
e[1]))

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

```
Number of training examples : 60000 and each image is of shape (784)
Number of training 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   0 139 253 190   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 [5]:

```
#Normalizing the vector space using min-max normalization

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

In [6]:
```
print("Class label of first image :", y_train[0])

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

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


# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20

# Plot train and cross validation loss
def plot_train_cv_loss(trained_model, epochs, colors=['b']):
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x_axis_values = list(range(1,epochs+1))

    validation_loss = trained_model.history['val_loss']
    train_loss = trained_model.history['loss']

    ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
    ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()


# Plot weight distribution using violin plot
def plot_weights(model):
    w_after = model.get_weights()

    o1_w = w_after[0].flatten().reshape(-1,1)
    o2_w = w_after[2].flatten().reshape(-1,1)
    out_w = w_after[4].flatten().reshape(-1,1)


    fig = plt.figure(figsize=(10,7))
    plt.title("Weight matrices after model trained\n")
    plt.subplot(1, 3, 1)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o1_w,color='b')
    plt.xlabel('Hidden Layer 1')

    plt.subplot(1, 3, 2)
    plt.title("Trained model\n Weights")
    ax = sns.violinplot(y=o2_w, color='r')
    plt.xlabel('Hidden Layer 2 ')

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

```
Class label of first image : 5
```

Implementing 2-layer MLP architecture with relu as an activation function

In [12]:
```
#2 Layer architecture using relu activation function

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal
(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, s
eed=None)) )
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, validati
on_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.2370 - acc: 0.9291 - val_loss: 0.1132 - val_acc: 0.9632
Epoch 2/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0865 - acc: 0.9738 - val_loss: 0.0906 - val_acc: 0.9712
Epoch 3/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0542 - acc: 0.9834 - val_loss: 0.0834 - val_acc: 0.9723
Epoch 4/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0362 - acc: 0.9888 - val_loss: 0.0755 - val_acc: 0.9761
Epoch 5/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0266 - acc: 0.9916 - val_loss: 0.0682 - val_acc: 0.9795
Epoch 6/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0202 - acc: 0.9934 - val_loss: 0.0671 - val_acc: 0.9798
Epoch 7/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0172 - acc: 0.9945 - val_loss: 0.0800 - val_acc: 0.9777
Epoch 8/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0196 - acc: 0.9935 - val_loss: 0.0731 - val_acc: 0.9797
Epoch 9/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0109 - acc: 0.9964 - val_loss: 0.0920 - val_acc: 0.9766
Epoch 10/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0130 - acc: 0.9959 - val_loss: 0.0961 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0108 - acc: 0.9964 - val_loss: 0.0830 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0117 - acc: 0.9957 - val_loss: 0.0900 - val_acc: 0.9775
Epoch 13/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0099 - acc: 0.9968 - val_loss: 0.0896 - val_acc: 0.9786
Epoch 14/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0097 - acc: 0.9966 - val_loss: 0.0862 - val_acc: 0.9794
Epoch 15/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0089 - acc: 0.9973 - val_loss: 0.0956 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0076 - acc: 0.9975 - val_loss: 0.1075 - val_acc: 0.9770
Epoch 17/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0053 - acc: 0.9985 - val_loss: 0.0911 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0084 - acc: 0.9972 - val_loss: 0.0954 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0071 - acc: 0.9976 - val_loss: 0.1070 - val_acc: 0.9778
Epoch 20/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0094 - acc: 0.9967 - val_loss: 0.0978 - val_acc: 0.9800
```

In [13]:

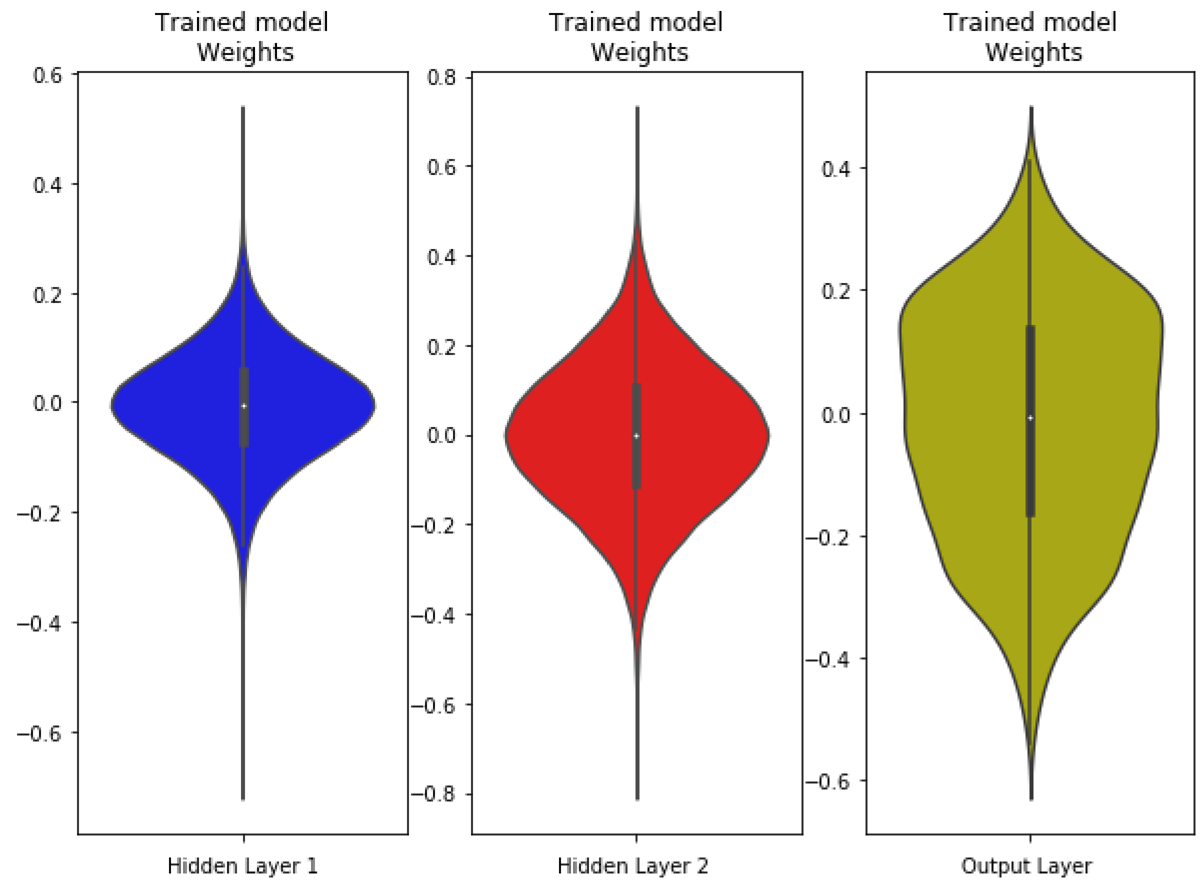
```
#Printing the score of the above model
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

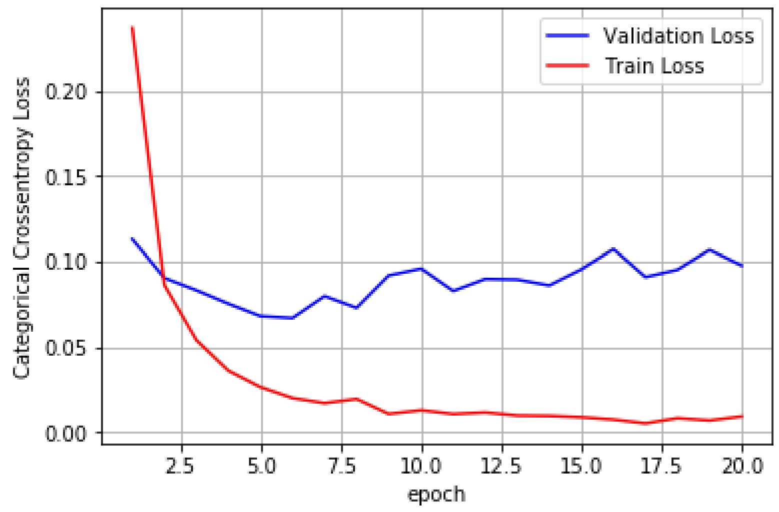
```
Test score: 0.0978071436640309
Test accuracy: 0.98
```

Visualizing the performance of the model

In [18]:
```
# Plot weight distribution using violin plot
plot_weights(model_relu)

warnings.filterwarnings(action='ignore', category=DataConversionWarning)

print()
print()

# Plot train and cross validation error
plot_train_cv_loss(history, nb_epoch)
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





Observations:-

- In the above diagram we can see that there is a huge gap between the Train-loss and the Validation loss in 20 epochs which is clear sign that the model is overfitting.
- The above 2-layer MLP model is a simple Neural-network in which no normalization and dropout layers are added.
- Lets see how the model behaves with the inclusion of batch-normalization.

2-layer MLP with Batch-normalization

In [19]:
```
from keras.layers.normalization import BatchNormalization
from keras import initializers
model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,),  kernel_initializer=initialize
rs.he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(128, activation='relu',kernel_initializer=initializers. he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_5 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

Compiling and fitting the above model

In [22]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

History = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
ion_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0083 - acc: 0.9973 - val_loss:
0.0861 - val_acc: 0.9799
Epoch 2/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0055 - acc: 0.9982 - val_loss:
0.0828 - val_acc: 0.9810
Epoch 3/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0072 - acc: 0.9976 - val_loss:
0.0934 - val_acc: 0.9788
Epoch 4/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0084 - acc: 0.9972 - val_loss:
0.0850 - val_acc: 0.9808
Epoch 5/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0060 - acc: 0.9980 - val_loss:
0.0856 - val_acc: 0.9806
Epoch 6/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0046 - acc: 0.9986 - val_loss:
0.0838 - val_acc: 0.9811
Epoch 7/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0044 - acc: 0.9985 - val_loss:
0.0806 - val_acc: 0.9823
Epoch 8/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0046 - acc: 0.9986 - val_loss:
0.0939 - val_acc: 0.9815
Epoch 9/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0075 - acc: 0.9977 - val_loss:
0.0993 - val_acc: 0.9777
Epoch 10/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0056 - acc: 0.9983 - val_loss:
0.0827 - val_acc: 0.9820
Epoch 11/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0036 - acc: 0.9990 - val_loss:
0.0863 - val_acc: 0.9823
Epoch 12/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0032 - acc: 0.9990 - val_loss:
0.0794 - val_acc: 0.9832
Epoch 13/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0054 - acc: 0.9986 - val_loss:
0.0731 - val_acc: 0.9848
Epoch 14/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0029 - acc: 0.9992 - val_loss:
0.0765 - val_acc: 0.9843
Epoch 15/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0040 - acc: 0.9987 - val_loss:
0.0909 - val_acc: 0.9815
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0041 - acc: 0.9987 - val_loss:
0.0835 - val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0056 - acc: 0.9980 - val_loss:
0.0953 - val_acc: 0.9813
Epoch 18/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0042 - acc: 0.9987 - val_loss:
0.0842 - val_acc: 0.9839
Epoch 19/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0033 - acc: 0.9989 - val_loss:
0.0818 - val_acc: 0.9837
Epoch 20/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0038 - acc: 0.9988 - val_loss:
0.0777 - val_acc: 0.9837
```

In [21]:

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

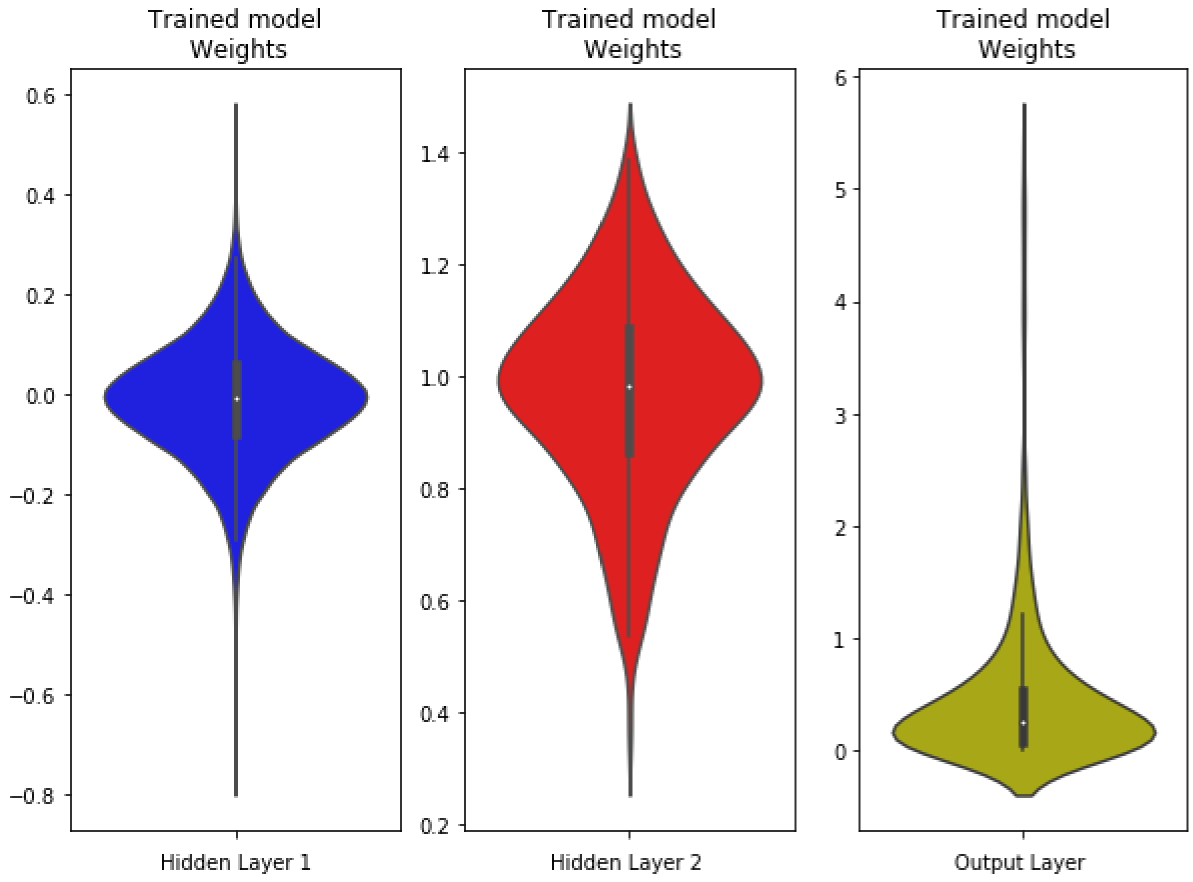
```
Test score: 0.07969029123196178
Test accuracy: 0.9808
```

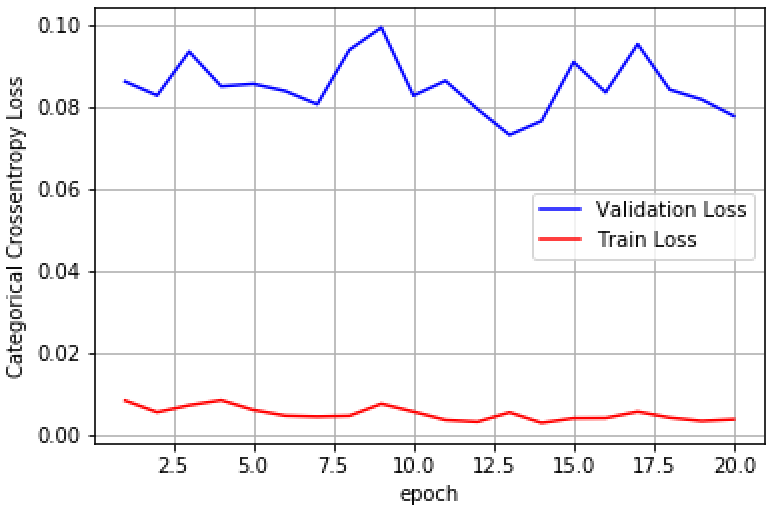
Visualizing the performance of the above model

In [23]:

```python
# Plot weight distribution using violin plot
plot_weights(model_batch)

warnings.filterwarnings(action='ignore', category=DataConversionWarning)

print()
print()

# Plot train and cross validation error
plot_train_cv_loss(History, nb_epoch)
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





OBSERVATIONS:-

- With the inclusion of batch-normalization the performance of the 2-layer MLP model decreased even further as the gap between the Train and validation loss is more than the previous model so the model has affected by high variance.
- The model's performance can be improved if some regularization is introduced which can be done by adding dropout layers.

Implementing the dropout layer into the above model

In [24]:
```
from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),  kernel_initializer=initializers.he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation="relu",  kernel_initializer=initializers.he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [25]:

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

drop_his = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
ion_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.4358 - acc: 0.8680 - val_loss:
0.1491 - val_acc: 0.9531
Epoch 2/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.2131 - acc: 0.9359 - val_loss:
0.1089 - val_acc: 0.9653
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1622 - acc: 0.9504 - val_loss:
0.0924 - val_acc: 0.9705
Epoch 4/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1387 - acc: 0.9581 - val_loss:
0.0825 - val_acc: 0.9744
Epoch 5/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1239 - acc: 0.9629 - val_loss:
0.0749 - val_acc: 0.9762
Epoch 6/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1095 - acc: 0.9671 - val_loss:
0.0750 - val_acc: 0.9767
Epoch 7/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1046 - acc: 0.9684 - val_loss:
0.0634 - val_acc: 0.9802
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0913 - acc: 0.9714 - val_loss:
0.0636 - val_acc: 0.9794
Epoch 9/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0870 - acc: 0.9726 - val_loss:
0.0584 - val_acc: 0.9814
Epoch 10/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0813 - acc: 0.9748 - val_loss:
0.0637 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0780 - acc: 0.9760 - val_loss:
0.0595 - val_acc: 0.9817
Epoch 12/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0716 - acc: 0.9778 - val_loss:
0.0568 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0718 - acc: 0.9771 - val_loss:
0.0612 - val_acc: 0.9808
Epoch 14/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0662 - acc: 0.9787 - val_loss:
0.0595 - val_acc: 0.9828
Epoch 15/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0645 - acc: 0.9788 - val_loss:
0.0551 - val_acc: 0.9838
Epoch 16/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0616 - acc: 0.9802 - val_loss:
0.0611 - val_acc: 0.9830
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0600 - acc: 0.9811 - val_loss:
0.0571 - val_acc: 0.9824
Epoch 18/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0569 - acc: 0.9818 - val_loss:
0.0519 - val_acc: 0.9840
Epoch 19/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0544 - acc: 0.9824 - val_loss:
0.0549 - val_acc: 0.9827
Epoch 20/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0540 - acc: 0.9830 - val_loss:
0.0572 - val_acc: 0.9844
```

In [27]:

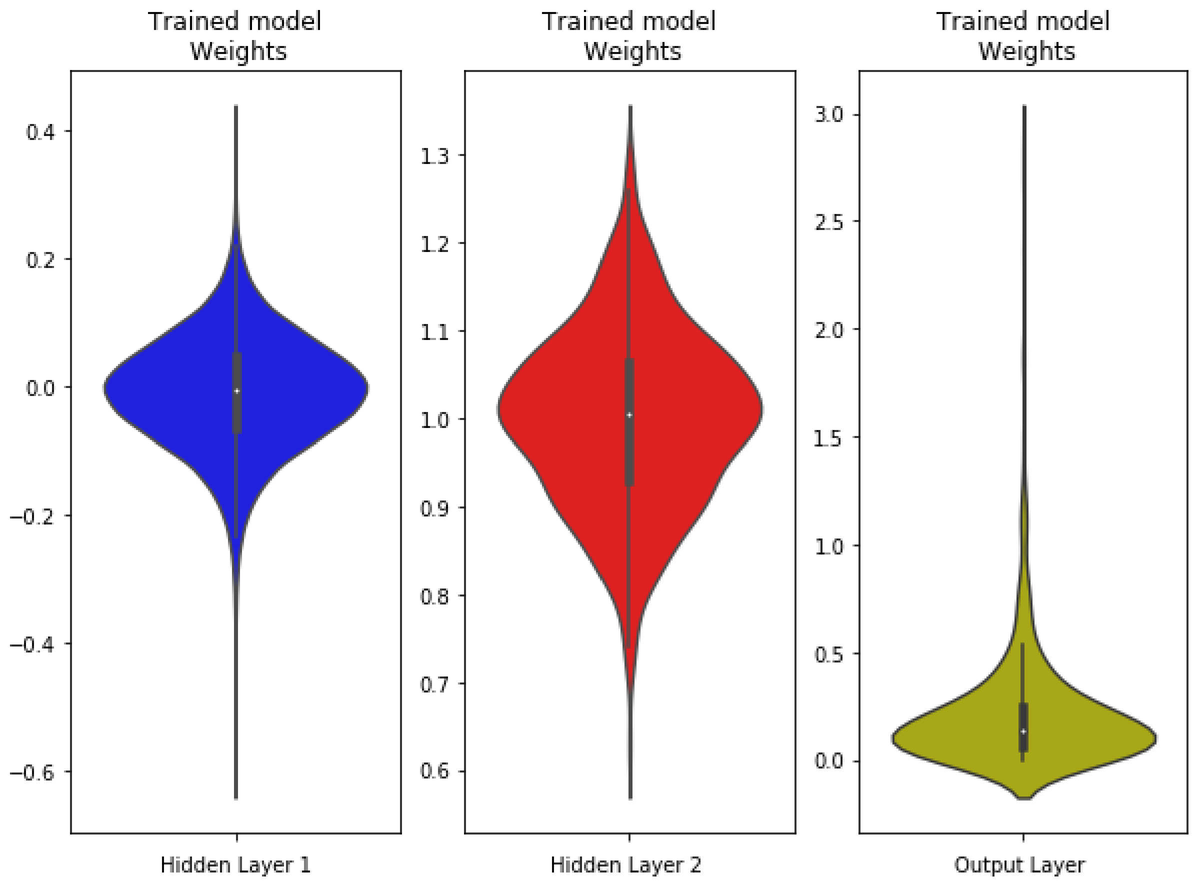
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

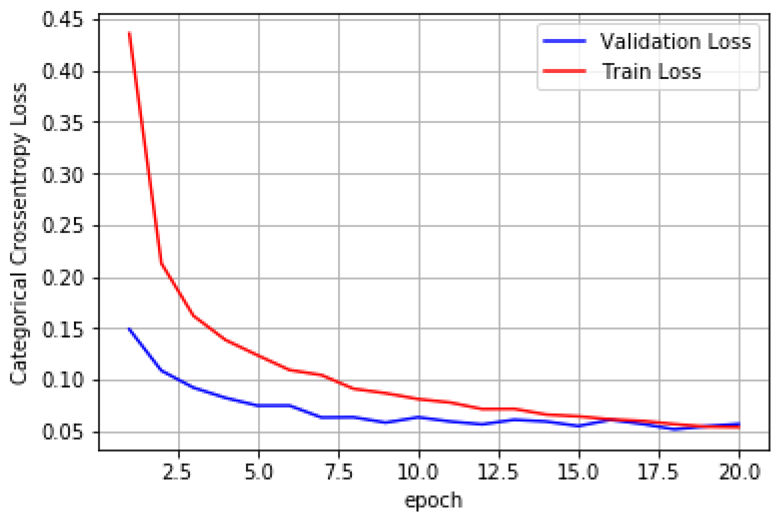
```
Test score: 0.05715570239143854
Test accuracy: 0.9844
```

In [26]:

```
# Plot weight distribution using violin plot
plot_weights(model_drop)

warnings.filterwarnings(action='ignore', category=DataConversionWarning)

print()
print()

# Plot train and cross validation error
plot_train_cv_loss(drop_his, nb_epoch)
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:1713: Fut
ureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(se
q)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(se
q)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





Observations:

- After studying the above plot we can see a clear improvement in the model as the gap between both the losses is very small which means the model is less overfiiting as compared to the previous models.
- The model starts performing well between the 13-20 epochs.
- The categorical cross-entropy loss is around 0.057 with 98.44 % accuracy.

Implementing 3-layer MLP architecture with Batch-Normalization

In [28]:
```
model_3l_batch = Sequential()

model_3l_batch.add(Dense(1024, activation='relu', input_shape=(input_dim,),  kernel_initializer=initia
lizers.he_normal(seed=None)))
model_3l_batch.add(BatchNormalization())

model_3l_batch.add(Dense(512, activation='relu',kernel_initializer=initializers. he_normal(seed=None))
 )
model_3l_batch.add(BatchNormalization())

model_3l_batch.add(Dense(128, activation='relu',kernel_initializer=initializers. he_normal(seed=None))
 )
model_3l_batch.add(BatchNormalization())

model_3l_batch.add(Dense(output_dim, activation='softmax'))


model_3l_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_5 (Batch (None, 1024)              4096      
_________________________________________________________________
dense_11 (Dense)             (None, 512)               524800    
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dense_12 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512       
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,402,250
Trainable params: 1,398,922
Non-trainable params: 3,328
_________________________________________________________________
```

In [29]:

```
model_3l_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
final_model_batch =model_3l_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=25, verbose=1, v
alidation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 12s 203us/step - loss: 0.1639 - acc: 0.9501 - val_loss:
0.0845 - val_acc: 0.9732
Epoch 2/25
60000/60000 [==============================] - 11s 183us/step - loss: 0.0647 - acc: 0.9797 - val_loss:
0.0732 - val_acc: 0.9766
Epoch 3/25
60000/60000 [==============================] - 11s 183us/step - loss: 0.0444 - acc: 0.9858 - val_loss:
0.0805 - val_acc: 0.9753
Epoch 4/25
60000/60000 [==============================] - 11s 190us/step - loss: 0.0328 - acc: 0.9896 - val_loss:
0.0702 - val_acc: 0.9786
Epoch 5/25
60000/60000 [==============================] - 11s 188us/step - loss: 0.0278 - acc: 0.9911 - val_loss:
0.0653 - val_acc: 0.9788
Epoch 6/25
60000/60000 [==============================] - 11s 191us/step - loss: 0.0262 - acc: 0.9915 - val_loss:
0.0728 - val_acc: 0.9789
Epoch 7/25
60000/60000 [==============================] - 11s 190us/step - loss: 0.0198 - acc: 0.9935 - val_loss:
0.0807 - val_acc: 0.9777
Epoch 8/25
60000/60000 [==============================] - 11s 189us/step - loss: 0.0194 - acc: 0.9935 - val_loss:
0.0891 - val_acc: 0.9745
Epoch 9/25
60000/60000 [==============================] - 11s 186us/step - loss: 0.0196 - acc: 0.9933 - val_loss:
0.0749 - val_acc: 0.9801
Epoch 10/25
60000/60000 [==============================] - 11s 184us/step - loss: 0.0151 - acc: 0.9948 - val_loss:
0.0816 - val_acc: 0.9793
Epoch 11/25
60000/60000 [==============================] - 11s 188us/step - loss: 0.0147 - acc: 0.9949 - val_loss:
0.0776 - val_acc: 0.9805
Epoch 12/25
60000/60000 [==============================] - 11s 185us/step - loss: 0.0131 - acc: 0.9958 - val_loss:
0.0742 - val_acc: 0.9796
Epoch 13/25
60000/60000 [==============================] - 11s 183us/step - loss: 0.0129 - acc: 0.9956 - val_loss:
0.0737 - val_acc: 0.9789
Epoch 14/25
60000/60000 [==============================] - 12s 194us/step - loss: 0.0089 - acc: 0.9970 - val_loss:
0.0813 - val_acc: 0.9807
Epoch 15/25
60000/60000 [==============================] - 11s 189us/step - loss: 0.0122 - acc: 0.9959 - val_loss:
0.0677 - val_acc: 0.9820
Epoch 16/25
60000/60000 [==============================] - 11s 185us/step - loss: 0.0114 - acc: 0.9962 - val_loss:
0.0734 - val_acc: 0.9822
Epoch 17/25
60000/60000 [==============================] - 11s 183us/step - loss: 0.0090 - acc: 0.9970 - val_loss:
0.0869 - val_acc: 0.9792
Epoch 18/25
60000/60000 [==============================] - 11s 182us/step - loss: 0.0086 - acc: 0.9974 - val_loss:
0.0780 - val_acc: 0.9834
Epoch 19/25
60000/60000 [==============================] - 11s 183us/step - loss: 0.0086 - acc: 0.9971 - val_loss:
0.0859 - val_acc: 0.9797
Epoch 20/25
60000/60000 [==============================] - 11s 185us/step - loss: 0.0111 - acc: 0.9964 - val_loss:
0.0837 - val_acc: 0.9811
Epoch 21/25
60000/60000 [==============================] - 11s 187us/step - loss: 0.0067 - acc: 0.9979 - val_loss:
0.0798 - val_acc: 0.9827
Epoch 22/25
60000/60000 [==============================] - 11s 191us/step - loss: 0.0091 - acc: 0.9969 - val_loss:
0.0860 - val_acc: 0.9784
Epoch 23/25
60000/60000 [==============================] - 11s 186us/step - loss: 0.0064 - acc: 0.9977 - val_loss:
0.0684 - val_acc: 0.9842
Epoch 24/25
60000/60000 [==============================] - 11s 184us/step - loss: 0.0049 - acc: 0.9984 - val_loss:
0.0750 - val_acc: 0.9842
Epoch 25/25
60000/60000 [==============================] - 11s 182us/step - loss: 0.0079 - acc: 0.9972 - val_loss:
0.0649 - val_acc: 0.9840
```

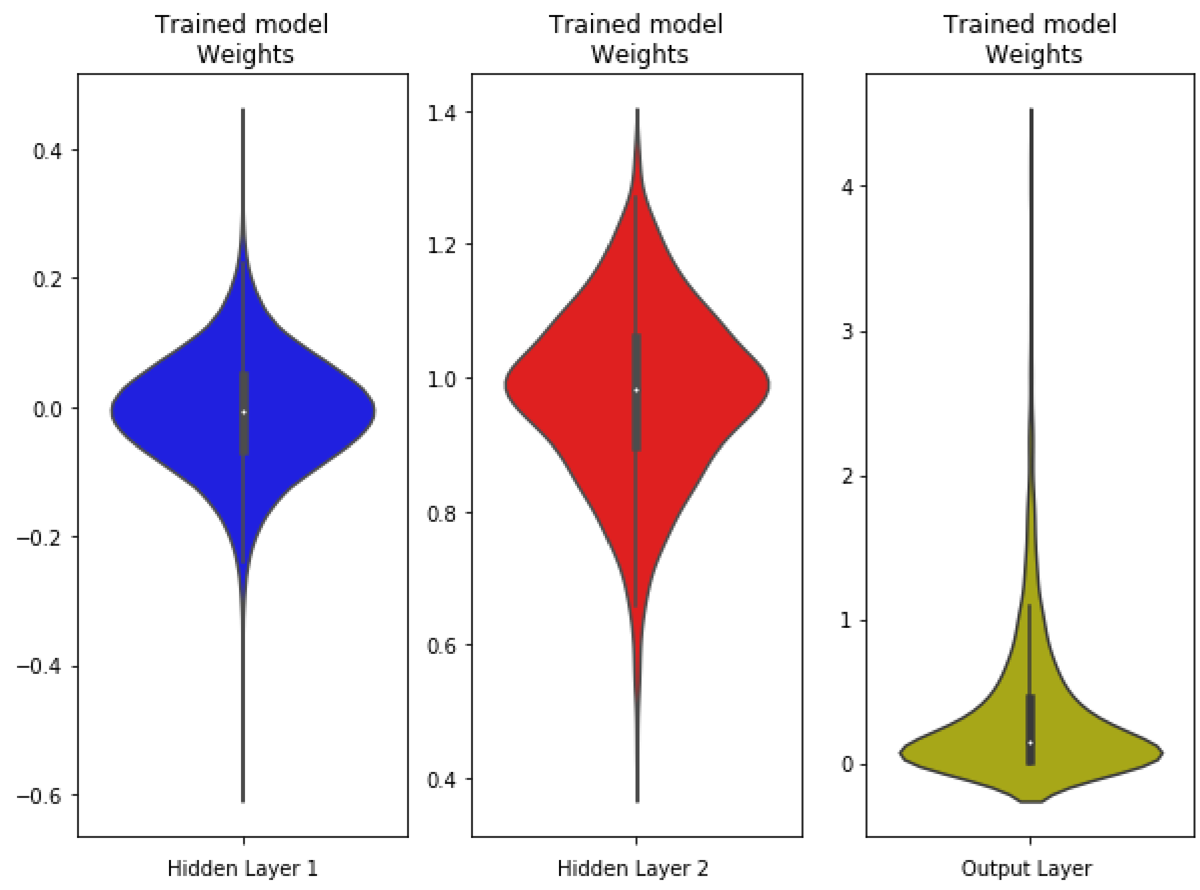
Scores of the above 3-layer MLP model

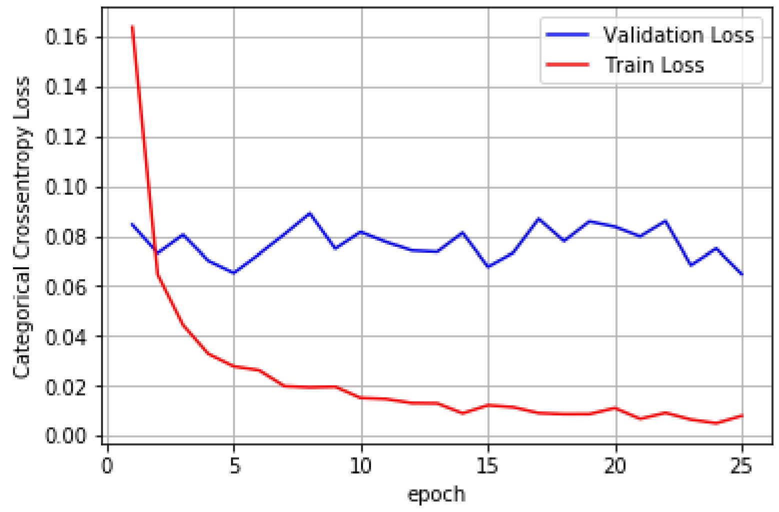
In [30]:
```
score_3l = model_3l_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_3l[0])
print('Test accuracy:', score_3l[1])
```

```
Test score: 0.06487380075010915
Test accuracy: 0.984
```

In [31]:
```
# Plot weight distribution using violin plot
plot_weights(model_31_batch)

epoch=25
# Plot train and cross validation error
plot_train_cv_loss(final_model_batch, epoch)
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





Observation:

- In the above plot the 3-layer MLP model is clearly overfitting and need to be regularized with the help of drop-out layers.

Implementing Drop-out layers into the 3-layer MLP model

In [32]:

```
model_3l_drop = Sequential()
#First Layer
model_3l_drop.add(Dense(1024, activation='relu', input_shape=(input_dim,),  kernel_initializer=initial
izers.he_normal(seed=None)))
model_3l_drop.add(BatchNormalization())
model_3l_drop.add(Dropout(0.3))

#Second Layer
model_3l_drop.add(Dense(512, activation="relu",  kernel_initializer=initializers.he_normal(seed=None))
 )
model_3l_drop.add(BatchNormalization())
model_3l_drop.add(Dropout(0.3))

#Third Layer
model_3l_drop.add(Dense(128, activation="relu",  kernel_initializer=initializers.he_normal(seed=None))
 )
model_3l_drop.add(BatchNormalization())
model_3l_drop.add(Dropout(0.3))

#Final Dense Layer
model_3l_drop.add(Dense(output_dim, activation='softmax'))


model_3l_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_8 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_15 (Dense)             (None, 512)               524800    
_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,402,250
Trainable params: 1,398,922
Non-trainable params: 3,328
_________________________________________________________________
```

In [33]:

```
model_3l_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
final_model_drop =model_3l_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=25, verbose=1, val
idation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 13s 213us/step - loss: 0.2677 - acc: 0.9185 - val_loss:
0.1099 - val_acc: 0.9636
Epoch 2/25
60000/60000 [==============================] - 11s 191us/step - loss: 0.1260 - acc: 0.9611 - val_loss:
0.0813 - val_acc: 0.9738
Epoch 3/25
60000/60000 [==============================] - 12s 194us/step - loss: 0.0930 - acc: 0.9716 - val_loss:
0.0756 - val_acc: 0.9766
Epoch 4/25
60000/60000 [==============================] - 12s 193us/step - loss: 0.0782 - acc: 0.9754 - val_loss:
0.0642 - val_acc: 0.9784
Epoch 5/25
60000/60000 [==============================] - 12s 198us/step - loss: 0.0663 - acc: 0.9799 - val_loss:
0.0616 - val_acc: 0.9807
Epoch 6/25
60000/60000 [==============================] - 12s 201us/step - loss: 0.0579 - acc: 0.9817 - val_loss:
0.0609 - val_acc: 0.9805
Epoch 7/25
60000/60000 [==============================] - 12s 204us/step - loss: 0.0547 - acc: 0.9822 - val_loss:
0.0648 - val_acc: 0.9804
Epoch 8/25
60000/60000 [==============================] - 12s 206us/step - loss: 0.0468 - acc: 0.9850 - val_loss:
0.0621 - val_acc: 0.9826
Epoch 9/25
60000/60000 [==============================] - 12s 201us/step - loss: 0.0431 - acc: 0.9862 - val_loss:
0.0545 - val_acc: 0.9839
Epoch 10/25
60000/60000 [==============================] - 12s 205us/step - loss: 0.0401 - acc: 0.9866 - val_loss:
0.0597 - val_acc: 0.9835
Epoch 11/25
60000/60000 [==============================] - 12s 204us/step - loss: 0.0399 - acc: 0.9872 - val_loss:
0.0589 - val_acc: 0.9807
Epoch 12/25
60000/60000 [==============================] - 12s 205us/step - loss: 0.0356 - acc: 0.9886 - val_loss:
0.0553 - val_acc: 0.9833
Epoch 13/25
60000/60000 [==============================] - 12s 195us/step - loss: 0.0346 - acc: 0.9885 - val_loss:
0.0543 - val_acc: 0.9845
Epoch 14/25
60000/60000 [==============================] - 12s 193us/step - loss: 0.0315 - acc: 0.9896 - val_loss:
0.0568 - val_acc: 0.9845
Epoch 15/25
60000/60000 [==============================] - 12s 200us/step - loss: 0.0292 - acc: 0.9906 - val_loss:
0.0642 - val_acc: 0.9830
Epoch 16/25
60000/60000 [==============================] - 12s 200us/step - loss: 0.0275 - acc: 0.9908 - val_loss:
0.0651 - val_acc: 0.9837
Epoch 17/25
60000/60000 [==============================] - 12s 197us/step - loss: 0.0265 - acc: 0.9914 - val_loss:
0.0577 - val_acc: 0.9834
Epoch 18/25
60000/60000 [==============================] - 12s 195us/step - loss: 0.0278 - acc: 0.9907 - val_loss:
0.0594 - val_acc: 0.9837
Epoch 19/25
60000/60000 [==============================] - 12s 195us/step - loss: 0.0256 - acc: 0.9913 - val_loss:
0.0537 - val_acc: 0.9843
Epoch 20/25
60000/60000 [==============================] - 12s 199us/step - loss: 0.0235 - acc: 0.9921 - val_loss:
0.0543 - val_acc: 0.9860
Epoch 21/25
60000/60000 [==============================] - 12s 197us/step - loss: 0.0241 - acc: 0.9918 - val_loss:
0.0550 - val_acc: 0.9848
Epoch 22/25
60000/60000 [==============================] - 12s 193us/step - loss: 0.0193 - acc: 0.9937 - val_loss:
0.0581 - val_acc: 0.9851
Epoch 23/25
60000/60000 [==============================] - 12s 194us/step - loss: 0.0206 - acc: 0.9928 - val_loss:
0.0614 - val_acc: 0.9839
Epoch 24/25
60000/60000 [==============================] - 12s 194us/step - loss: 0.0186 - acc: 0.9938 - val_loss:
0.0522 - val_acc: 0.9864
Epoch 25/25
60000/60000 [==============================] - 12s 193us/step - loss: 0.0177 - acc: 0.9940 - val_loss:
0.0629 - val_acc: 0.9837
```

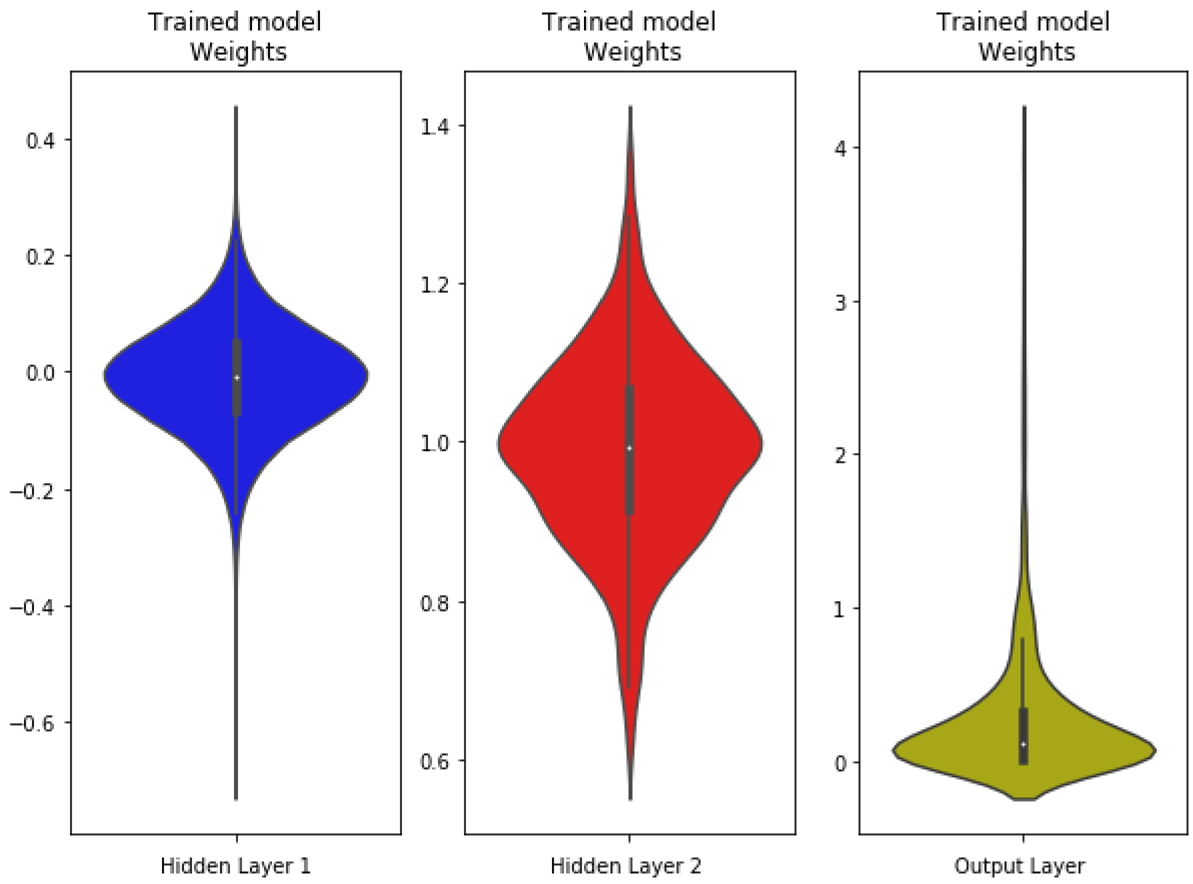
In [34]:

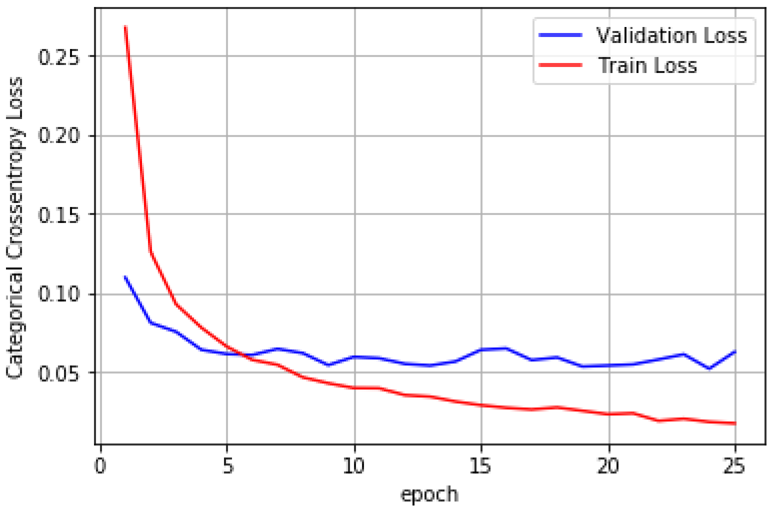
```
score_3l_drop = model_3l_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_3l_drop[0])
print('Test accuracy:', score_3l_drop[1])
```

```
Test score: 0.06286698746977418
Test accuracy: 0.9837
```

In [35]:
```
# Plot weight distribution using violin plot
plot_weights(model_3l_drop)

epoch=25
# Plot train and cross validation error
plot_train_cv_loss(final_model_drop, epoch)
```

```
c:\users\admin\appdata\local\programs\python\python36\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





Observations:

- After studying the above plot we can see a clear improvement in the model as the gap between both the losses is small which means the model is less overfiiting as compared to the previous models.
- The model ran for a total of 25 epochs.
- The categorical cross-entropy loss is around 0.062 with 98.37 % accuracy.

Implementing the 5-Layer MLP model

In [51]:

```
model_5l_drop = Sequential()

model_5l_drop.add(Dense(4096, activation='relu', input_shape=(input_dim,),  kernel_initializer=initial
izers.he_normal(seed=None)))
model_5l_drop.add(BatchNormalization())
model_5l_drop.add(Dropout(0.38))

model_5l_drop.add(Dense(2048, activation="relu",  kernel_initializer=initializers.he_normal(seed=None
)) )
model_5l_drop.add(BatchNormalization())
model_5l_drop.add(Dropout(0.38))

model_5l_drop.add(Dense(1024, activation="relu",  kernel_initializer=initializers.he_normal(seed=None
)) )
model_5l_drop.add(BatchNormalization())
model_5l_drop.add(Dropout(0.38))

model_5l_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),  kernel_initializer=initiali
zers.he_normal(seed=None)))
model_5l_drop.add(BatchNormalization())
model_5l_drop.add(Dropout(0.38))

model_5l_drop.add(Dense(128, activation='relu', input_shape=(input_dim,),  kernel_initializer=initiali
zers.he_normal(seed=None)))
model_5l_drop.add(BatchNormalization())
model_5l_drop.add(Dropout(0.38))



model_5l_drop.add(Dense(output_dim, activation='softmax'))


model_5l_drop.summary()


Model_5l=model_5l_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
])
final_5l_drop =model_5l_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=25, verbose=1, valida
tion_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 4096)              3215360
_________________________________________________________________
batch_normalization_18 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_13 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_29 (Dense)             (None, 2048)              8390656
_________________________________________________________________
batch_normalization_19 (Batc (None, 2048)              8192
_________________________________________________________________
dropout_14 (Dropout)         (None, 2048)              0
_________________________________________________________________
dense_30 (Dense)             (None, 1024)              2098176
_________________________________________________________________
batch_normalization_20 (Batc (None, 1024)              4096
_________________________________________________________________
dropout_15 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
batch_normalization_21 (Batc (None, 512)               2048
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_32 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_22 (Batc (None, 128)               512
_________________________________________________________________
dropout_17 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_33 (Dense)             (None, 10)                1290
=================================================================
Total params: 14,327,178
Trainable params: 14,311,562
Non-trainable params: 15,616
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 110s 2ms/step - loss: 0.2825 - acc: 0.9145 - val_loss:
0.1056 - val_acc: 0.9668
Epoch 2/25
60000/60000 [==============================] - 107s 2ms/step - loss: 0.1298 - acc: 0.9621 - val_loss:
0.0818 - val_acc: 0.9754
Epoch 3/25
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0997 - acc: 0.9702 - val_loss:
0.0786 - val_acc: 0.9767
Epoch 4/25
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0882 - acc: 0.9743 - val_loss:
0.0751 - val_acc: 0.9781
Epoch 5/25
60000/60000 [==============================] - 115s 2ms/step - loss: 0.0730 - acc: 0.9780 - val_loss:
0.0820 - val_acc: 0.9749
Epoch 6/25
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0666 - acc: 0.9793 - val_loss:
0.0718 - val_acc: 0.9793
Epoch 7/25
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0658 - acc: 0.9802 - val_loss:
0.0717 - val_acc: 0.9796
Epoch 8/25
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0588 - acc: 0.9821 - val_loss:
0.0676 - val_acc: 0.9812
Epoch 9/25
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0530 - acc: 0.9839 - val_loss:
0.0704 - val_acc: 0.9800
Epoch 10/25
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0469 - acc: 0.9859 - val_loss:
0.0715 - val_acc: 0.9790
Epoch 11/25
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0454 - acc: 0.9859 - val_loss:
0.0628 - val_acc: 0.9826
Epoch 12/25
60000/60000 [==============================] - 110s 2ms/step - loss: 0.0429 - acc: 0.9868 - val_loss:
0.0749 - val_acc: 0.9798
Epoch 13/25
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0409 - acc: 0.9875 - val_loss:
0.0696 - val_acc: 0.9801
Epoch 14/25
60000/60000 [==============================] - 111s 2ms/step - loss: 0.0372 - acc: 0.9886 - val_loss:
0.0663 - val_acc: 0.9814
Epoch 15/25
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0361 - acc: 0.9892 - val_loss:
0.0622 - val_acc: 0.9842
Epoch 16/25
60000/60000 [==============================] - 132s 2ms/step - loss: 0.0379 - acc: 0.9888 - val_loss:
```

```
0.0789 - val_acc: 0.9810
Epoch 17/25
60000/60000 [==============================] - 120s 2ms/step - loss: 0.0338 - acc: 0.9895 - val_loss:
0.0655 - val_acc: 0.9829
Epoch 18/25
60000/60000 [==============================] - 112s 2ms/step - loss: 0.0319 - acc: 0.9902 - val_loss:
0.0659 - val_acc: 0.9833
Epoch 19/25
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0301 - acc: 0.9907 - val_loss:
0.0571 - val_acc: 0.9845
Epoch 20/25
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0279 - acc: 0.9914 - val_loss:
0.0524 - val_acc: 0.9863
Epoch 21/25
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0259 - acc: 0.9920 - val_loss:
0.0569 - val_acc: 0.9847
Epoch 22/25
60000/60000 [==============================] - 114s 2ms/step - loss: 0.0243 - acc: 0.9926 - val_loss:
0.0581 - val_acc: 0.9839
Epoch 23/25
60000/60000 [==============================] - 115s 2ms/step - loss: 0.0243 - acc: 0.9927 - val_loss:
0.0616 - val_acc: 0.9839
Epoch 24/25
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0225 - acc: 0.9931 - val_loss:
0.0577 - val_acc: 0.9858
Epoch 25/25
60000/60000 [==============================] - 114s 2ms/step - loss: 0.0218 - acc: 0.9931 - val_loss:
0.0667 - val_acc: 0.9849
```

Scores of the above model

In [52]:

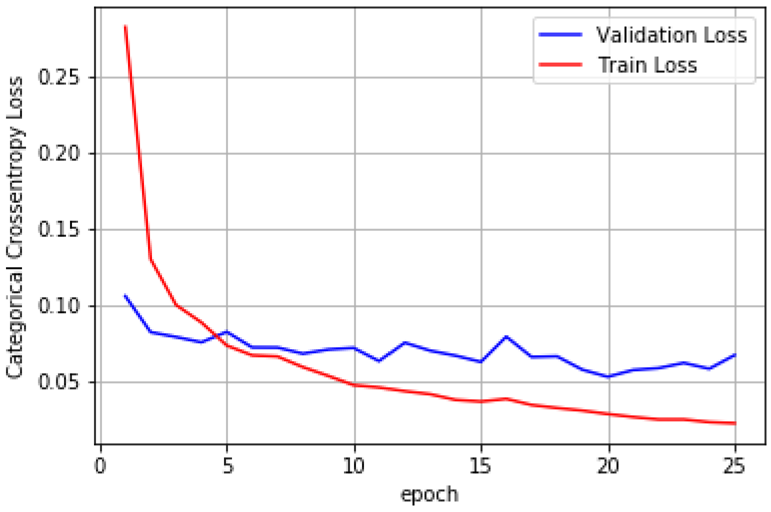
```python
Model_51_drop = model_51_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', Model_51_drop[0])
print('Test accuracy:', Model_51_drop[1])
```

```
Test score: 0.06666945320260712
Test accuracy: 0.9849
```

Plotting the train and validation loss of the above model

In [54]:

```python
epoch=25
# Plot train and cross validation error
plot_train_cv_loss(final_51_drop,epoch)
```



Conclusion

1. After implementing all the above MLP architectures I can conclude that this models tends to overfitts easily as the number of layers are increased.
2. The techniques like Batch-Normalization and Drop-outs proved very useful in improving the model's performance and solves these problems efficiently.

- Internal Covariance-shift.
- Overfitting.

In [ ]: