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 [0]:

```
import matplotlib.pyplot as plt
import numpy as np
import time
import pandas as pd

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


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

In [0]:

```
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 [6]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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])
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

In [0]:

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

In [0]:

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

In [0]:

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

batch_size = 128
nb_epoch = 20
```

2 Layers

In [28]:

```
model_drop = Sequential()
```

```
model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(
mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(80, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, see
d=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_12 (Dense)             (None, 364)               285740
_________________________________________________________________
batch_normalization_9 (Batch (None, 364)               1456
_________________________________________________________________
dropout_9 (Dropout)          (None, 364)               0
_________________________________________________________________
dense_13 (Dense)             (None, 80)                29200
_________________________________________________________________
batch_normalization_10 (Batc (None, 80)                320
_________________________________________________________________
dropout_10 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                810
=================================================================
Total params: 317,526
Trainable params: 316,638
Non-trainable params: 888
_________________________________________________________________
```

In [29]:

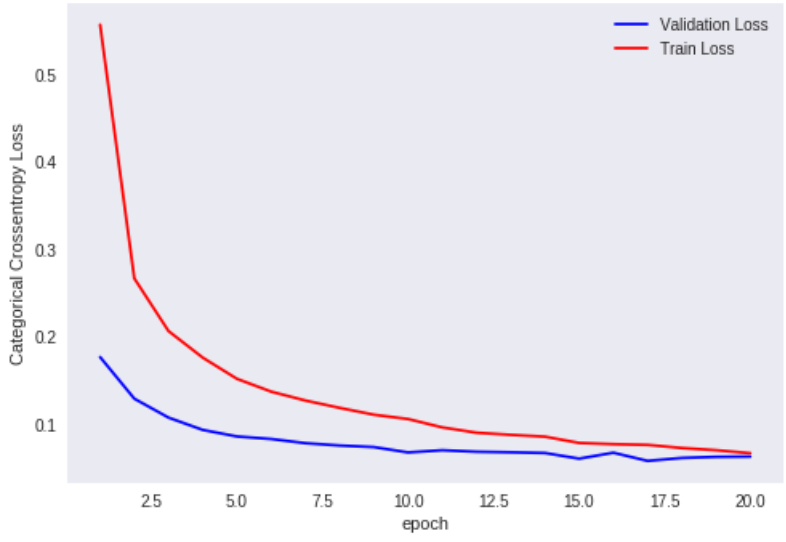
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, Y_test))

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.5568 - acc: 0.8326 - val_loss: 0
.1769 - val_acc: 0.9444
Epoch 2/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.2668 - acc: 0.9217 - val_loss: 0
.1293 - val_acc: 0.9592
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2066 - acc: 0.9394 - val_loss: 0
.1076 - val_acc: 0.9659
Epoch 4/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1762 - acc: 0.9473 - val_loss: 0
.0936 - val_acc: 0.9715
Epoch 5/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1520 - acc: 0.9543 - val_loss: 0
.0861 - val_acc: 0.9743
Epoch 6/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1373 - acc: 0.9595 - val_loss: 0
.0832 - val_acc: 0.9748
Epoch 7/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1272 - acc: 0.9623 - val_loss: 0
.0784 - val_acc: 0.9750
Epoch 8/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1188 - acc: 0.9640 - val_loss: 0
.0757 - val_acc: 0.9768
Epoch 9/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1110 - acc: 0.9666 - val_loss: 0
```

```
.0740 - val_acc: 0.9784
Epoch 10/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1060 - acc: 0.9685 - val_loss: 0
.0678 - val_acc: 0.9789
Epoch 11/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0964 - acc: 0.9710 - val_loss: 0
.0703 - val_acc: 0.9786
Epoch 12/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0904 - acc: 0.9725 - val_loss: 0
.0686 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0879 - acc: 0.9732 - val_loss: 0
.0680 - val_acc: 0.9791
Epoch 14/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0859 - acc: 0.9739 - val_loss: 0
.0672 - val_acc: 0.9790
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0787 - acc: 0.9760 - val_loss: 0
.0607 - val_acc: 0.9815
Epoch 16/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0773 - acc: 0.9765 - val_loss: 0
.0675 - val_acc: 0.9803
Epoch 17/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0765 - acc: 0.9759 - val_loss: 0
.0581 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0729 - acc: 0.9774 - val_loss: 0
.0615 - val_acc: 0.9819
Epoch 19/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0704 - acc: 0.9784 - val_loss: 0
.0628 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0669 - acc: 0.9795 - val_loss: 0
.0631 - val_acc: 0.9810
Test score: 0.06307864126095082
Test accuracy: 0.981
```

In [30]:

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

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

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

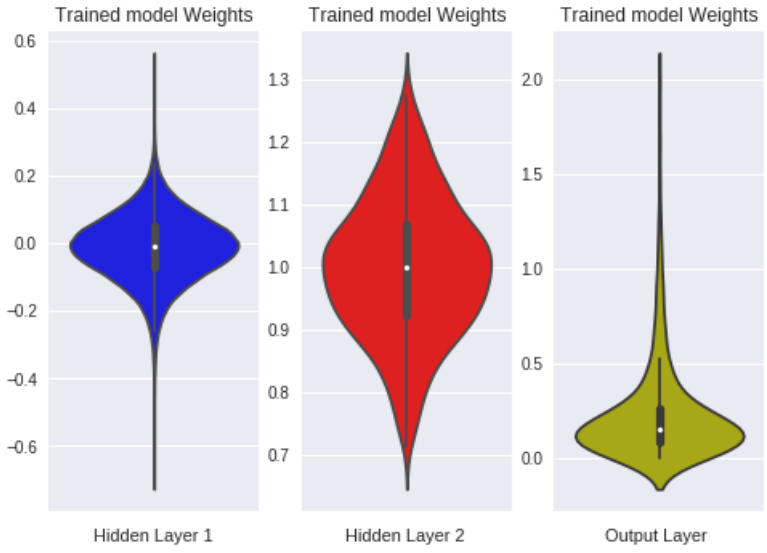


In [31]:

```
w_after = model_drop.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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



3 Layers

In [32]:

```
model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(
mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, se
ed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, se
ed=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_15 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_12 (Batc (None, 256)               1024
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

In [34]:

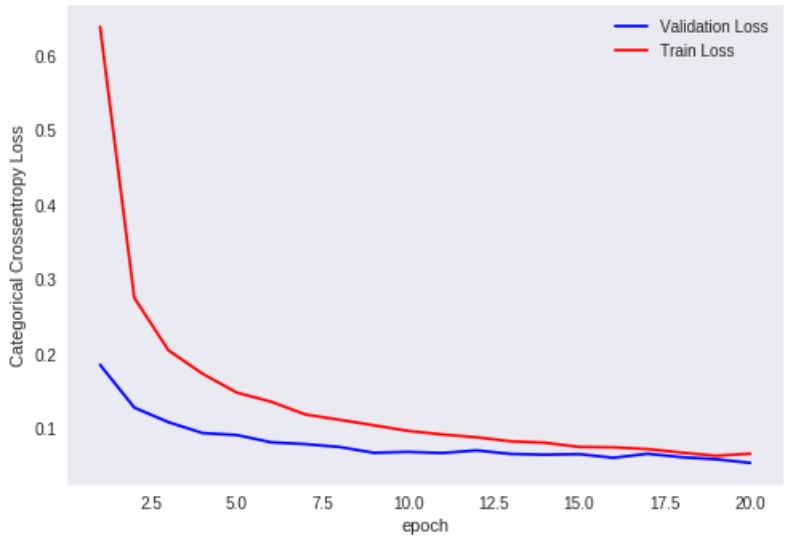
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, Y_test))

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.6390 - acc: 0.8050 - val_loss:
0.1855 - val_acc: 0.9417
Epoch 2/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.2754 - acc: 0.9183 - val_loss:
0.1281 - val_acc: 0.9615
Epoch 3/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.2049 - acc: 0.9396 - val_loss: 0
.1085 - val_acc: 0.9662
Epoch 4/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.1734 - acc: 0.9488 - val_loss: 0
.0939 - val_acc: 0.9721
Epoch 5/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1482 - acc: 0.9563 - val_loss: 0
.0913 - val_acc: 0.9719
Epoch 6/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.1360 - acc: 0.9596 - val_loss: 0
.0815 - val_acc: 0.9766
Epoch 7/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1189 - acc: 0.9641 - val_loss: 0
.0791 - val_acc: 0.9766
Epoch 8/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.1117 - acc: 0.9663 - val_loss: 0
.0752 - val_acc: 0.9761
Epoch 9/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.1044 - acc: 0.9686 - val_loss: 0
.0675 - val_acc: 0.9802
Epoch 10/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0969 - acc: 0.9709 - val_loss: 0
.0687 - val_acc: 0.9797
Epoch 11/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0921 - acc: 0.9724 - val_loss:
0.0672 - val_acc: 0.9807
```

```
0.0672 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0883 - acc: 0.9741 - val_loss: 0
.0707 - val_acc: 0.9804
Epoch 13/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0829 - acc: 0.9743 - val_loss:
0.0660 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0810 - acc: 0.9755 - val_loss:
0.0650 - val_acc: 0.9802
Epoch 15/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0754 - acc: 0.9772 - val_loss:
0.0657 - val_acc: 0.9831
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0748 - acc: 0.9772 - val_loss:
0.0608 - val_acc: 0.9812
Epoch 17/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0725 - acc: 0.9773 - val_loss:
0.0662 - val_acc: 0.9813
Epoch 18/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0678 - acc: 0.9797 - val_loss:
0.0615 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.0635 - acc: 0.9805 - val_loss:
0.0588 - val_acc: 0.9837
Epoch 20/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0663 - acc: 0.9803 - val_loss:
0.0539 - val_acc: 0.9849
Test score: 0.053948233158088985
Test accuracy: 0.9849
```

In [35]:

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

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

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

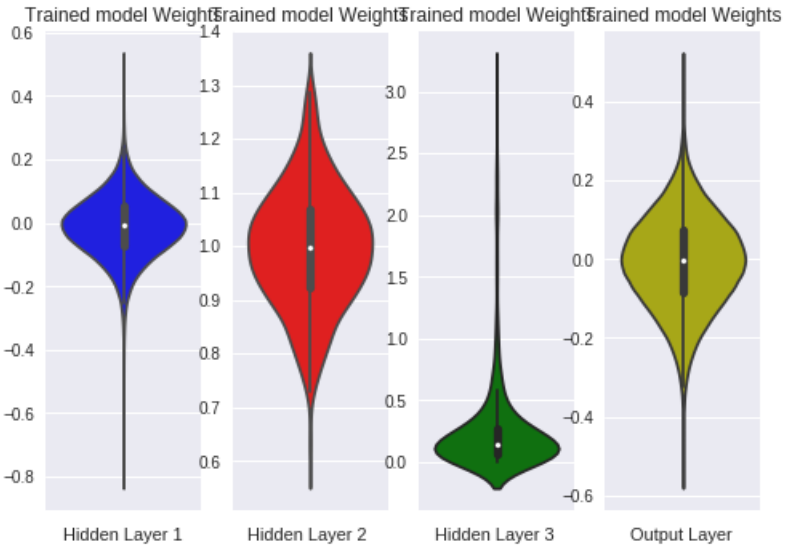


In [36]:

```
w_after = model_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
out_w = w_after[6].flatten().reshape(-1,1)


fig = plt.figure()
```

```
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  violin_data = remove_na(group_data)
```



5 Layers

In [37]:

```
model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(
mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, se
ed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, se
ed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.176, see
d=None)) )
model_drop.add(BatchNormalization())
```

```
model_drop.add(Dropout(0.5))

model_drop.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, 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_19 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_20 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_15 (Batc (None, 256)               1024
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_16 (Batc (None, 128)               512
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_22 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_17 (Batc (None, 64)                256
_________________________________________________________________
dropout_17 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_23 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_18 (Batc (None, 32)                128
_________________________________________________________________
dropout_18 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_24 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
```

In [38]:

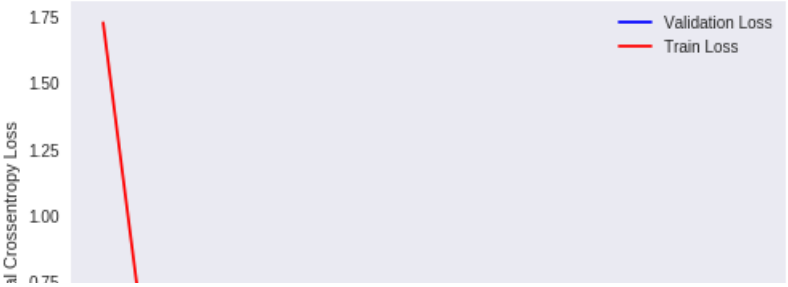
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
n_data=(X_test, Y_test))

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

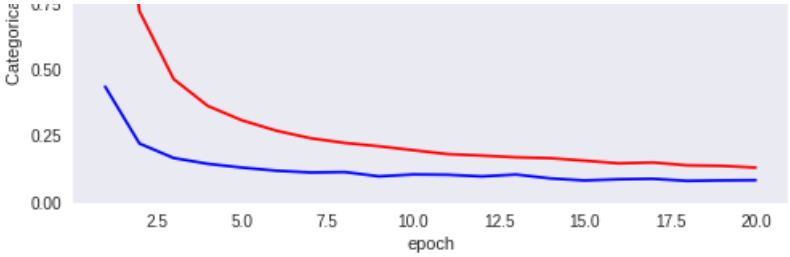
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 276us/step - loss: 1.7274 - acc: 0.4389 - val_loss:
0.4358 - val_acc: 0.8914
Epoch 2/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.7210 - acc: 0.7873 - val_loss:
0.2211 - val_acc: 0.9370
Epoch 3/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.4643 - acc: 0.8757 - val_loss:
0.1661 - val_acc: 0.9541
Epoch 4/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.3627 - acc: 0.9081 - val_loss:
```

```
0.1441 - val_acc: 0.9617
Epoch 5/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.3083 - acc: 0.9257 - val_loss:
0.1298 - val_acc: 0.9660
Epoch 6/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.2693 - acc: 0.9365 - val_loss:
0.1182 - val_acc: 0.9704
Epoch 7/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.2408 - acc: 0.9446 - val_loss:
0.1114 - val_acc: 0.9727
Epoch 8/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.2231 - acc: 0.9479 - val_loss:
0.1131 - val_acc: 0.9718
Epoch 9/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.2105 - acc: 0.9513 - val_loss:
0.0968 - val_acc: 0.9766
Epoch 10/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.1956 - acc: 0.9556 - val_loss:
0.1040 - val_acc: 0.9752
Epoch 11/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.1809 - acc: 0.9590 - val_loss:
0.1028 - val_acc: 0.9751
Epoch 12/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1753 - acc: 0.9599 - val_loss:
0.0966 - val_acc: 0.9765
Epoch 13/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1688 - acc: 0.9613 - val_loss:
0.1037 - val_acc: 0.9763
Epoch 14/20
60000/60000 [==============================] - 14s 225us/step - loss: 0.1655 - acc: 0.9627 - val_loss:
0.0891 - val_acc: 0.9779
Epoch 15/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.1561 - acc: 0.9643 - val_loss:
0.0812 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.1459 - acc: 0.9664 - val_loss:
0.0858 - val_acc: 0.9804
Epoch 17/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.1492 - acc: 0.9665 - val_loss:
0.0876 - val_acc: 0.9797
Epoch 18/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.1385 - acc: 0.9684 - val_loss:
0.0797 - val_acc: 0.9799
Epoch 19/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1365 - acc: 0.9700 - val_loss:
0.0815 - val_acc: 0.9804
Epoch 20/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.1300 - acc: 0.9714 - val_loss:
0.0824 - val_acc: 0.9810
Test score: 0.08238632999660912
Test accuracy: 0.981
```

In [39]:

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

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

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

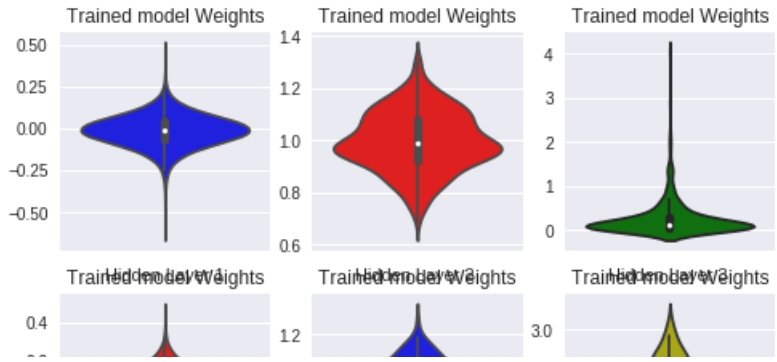


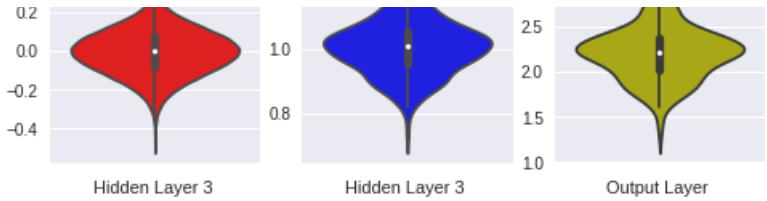


In [41]:

```
w_after = model_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
h3_w = w_after[4].flatten().reshape(-1,1)
h4_w = w_after[6].flatten().reshape(-1,1)
h5_w = w_after[8].flatten().reshape(-1,1)
out_w = w_after[10].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(2, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(2, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 3, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 3, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='b')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 3, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is deprecat
ed and is a private function. Do not use.
  violin_data = remove_na(group_data)
```




0.2
0.0
−0.2
−0.4
Hidden Layer 3
1.0
0.8
Hidden Layer 3
25
20
15
10
Output Layer