Keras -- MLPs on MNIST

In [1]:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/compat/v2_compat.py:68: disable_resource_variables (from
tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

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

```
Using TensorFlow backend.
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, ty, 'r', label="Train Loss")
    ax.plot(x, vy, 'b', label="Validation Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
    plt.show()
```

In [4]:

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
```

In [5]:

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

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

In [0]:

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

In [7]:

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

```
Number of training examples : 60000 and each image is of shape (784)
Number of test examples : 10000 and each image is of shape (784)
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:

```
X_train = X_train/255
X_test = X_test/255
print(X_train[0])
```

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

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.31764706 0.94117647 0.99215686
 0.99215686 0.46666667 0.09803922 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
 0.58823529 0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
 0.         0.         0.         0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         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 [9]:

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

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

Softmax classifier

In [0]:

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

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

In [0]:

```
# some model parameters
```

```
output_dim = 10
input_dim = X_train.shape[1]
batch_size = 128
nb_epoch = 20
```

2 hidden layer - MLP + ReLU activation + ADAM optimizer + Batch normalization + Dropout

In [12]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(480, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Plea
se use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please us
e tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4479: The name tf.truncated_normal is deprecated. Ple
ase use tf.random.truncated_normal instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:148: The name tf.placeholder_with_default is
deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 480)               376800
_________________________________________________________________
batch_normalization_1 (Batch (None, 480)               1920
_________________________________________________________________
dropout_1 (Dropout)          (None, 480)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               123136
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 505,450
```

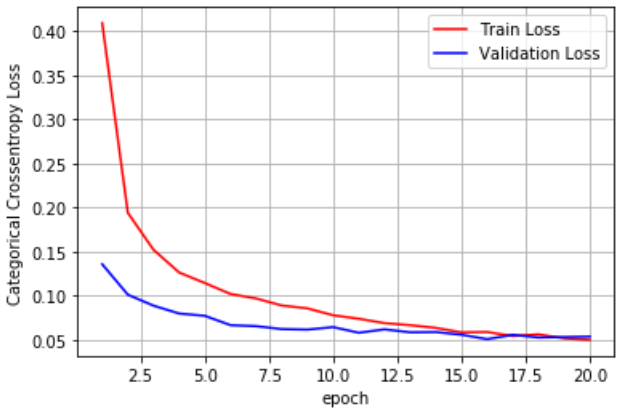
```
Trainable params: 503,978
Non-trainable params: 1,472
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name t
f.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3576: The name tf.log is deprecated. Please use tf.ma
th.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/ops/math_grad.py:1424: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please us
e tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf
.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use t
f.compat.v1.Session instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated. P
lease use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please us
e tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is
deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecated.
Please use tf.compat.v1.variables_initializer instead.

60000/60000 [==============================] - 13s 217us/step - loss: 0.4090 - acc: 0.8755 - val_l
oss: 0.1359 - val_acc: 0.9569
Epoch 2/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.1940 - acc: 0.9407 -
val_loss: 0.1013 - val_acc: 0.9682
Epoch 3/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1522 - acc: 0.9540 -
val_loss: 0.0889 - val_acc: 0.9731
Epoch 4/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1264 - acc: 0.9608 -
val_loss: 0.0800 - val_acc: 0.9758
Epoch 5/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1145 - acc: 0.9645 -
val_loss: 0.0774 - val_acc: 0.9760
Epoch 6/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1021 - acc: 0.9682 -
val_loss: 0.0668 - val_acc: 0.9776
Epoch 7/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0970 - acc: 0.9695 -
val_loss: 0.0656 - val_acc: 0.9807
Epoch 8/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0890 - acc: 0.9717 -
val_loss: 0.0623 - val_acc: 0.9806
Epoch 9/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0858 - acc: 0.9731 -
val_loss: 0.0618 - val_acc: 0.9809
Epoch 10/20
```

```
60000/60000 [==============================] - 3s 55us/step - loss: 0.0780 - acc: 0.9750 -
val_loss: 0.0646 - val_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0740 - acc: 0.9766 -
val_loss: 0.0582 - val_acc: 0.9829
Epoch 12/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0691 - acc: 0.9780 -
val_loss: 0.0620 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0668 - acc: 0.9789 -
val_loss: 0.0587 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0637 - acc: 0.9794 -
val_loss: 0.0590 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0586 - acc: 0.9806 -
val_loss: 0.0560 - val_acc: 0.9829
Epoch 16/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0590 - acc: 0.9808 -
val_loss: 0.0509 - val_acc: 0.9849
Epoch 17/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0544 - acc: 0.9821 -
val_loss: 0.0558 - val_acc: 0.9839
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0564 - acc: 0.9818 -
val_loss: 0.0528 - val_acc: 0.9846
Epoch 19/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0518 - acc: 0.9824 -
val_loss: 0.0533 - val_acc: 0.9843
Epoch 20/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0501 - acc: 0.9837 -
val_loss: 0.0538 - val_acc: 0.9839
CPU times: user 1min 25s, sys: 7.62 s, total: 1min 33s
Wall time: 1min 16s
```

In [13]:

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

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

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

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

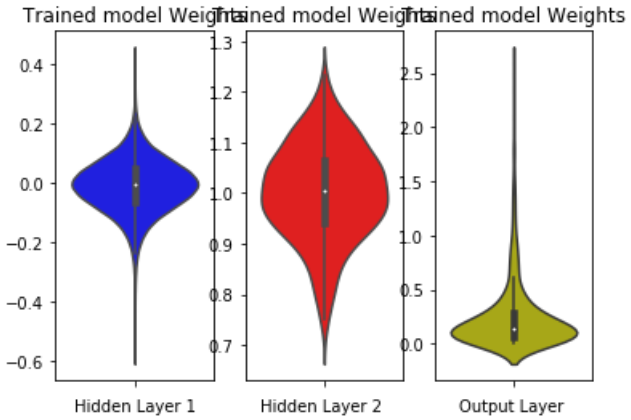
```
Test score: 0.05383474605759839
Test accuracy: 0.9839
```



In [14]:

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

```
w_after = model_relu.get_weights()

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

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

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

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



3 hidden layer - MLP + ReLU activation + ADAM optimizer + Batch normalization + Dropout

In [15]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(360, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(120, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 360)               282600
```

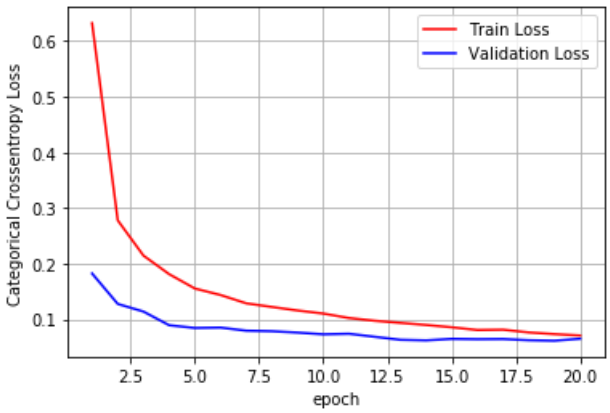
```
_________________________________________________________________
batch_normalization_3 (Batch (None, 360)               1440
_________________________________________________________________
dropout_3 (Dropout)          (None, 360)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               92416
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               30840
_________________________________________________________________
batch_normalization_5 (Batch (None, 120)               480
_________________________________________________________________
dropout_5 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1210
=================================================================
Total params: 410,010
Trainable params: 408,538
Non-trainable params: 1,472
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.6322 - acc: 0.8065 -
val_loss: 0.1825 - val_acc: 0.9428
Epoch 2/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.2779 - acc: 0.9175 -
val_loss: 0.1278 - val_acc: 0.9611
Epoch 3/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.2146 - acc: 0.9373 -
val_loss: 0.1141 - val_acc: 0.9657
Epoch 4/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.1813 - acc: 0.9468 -
val_loss: 0.0896 - val_acc: 0.9723
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1555 - acc: 0.9532 -
val_loss: 0.0846 - val_acc: 0.9740
Epoch 6/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.1438 - acc: 0.9575 -
val_loss: 0.0853 - val_acc: 0.9751
Epoch 7/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1289 - acc: 0.9622 -
val_loss: 0.0797 - val_acc: 0.9748
Epoch 8/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1224 - acc: 0.9635 -
val_loss: 0.0788 - val_acc: 0.9764
Epoch 9/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1160 - acc: 0.9662 -
val_loss: 0.0763 - val_acc: 0.9775
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1105 - acc: 0.9663 -
val_loss: 0.0733 - val_acc: 0.9792
Epoch 11/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1025 - acc: 0.9695 -
val_loss: 0.0742 - val_acc: 0.9773
Epoch 12/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0976 - acc: 0.9708 -
val_loss: 0.0688 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0938 - acc: 0.9726 -
val_loss: 0.0634 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0900 - acc: 0.9729 -
val_loss: 0.0623 - val_acc: 0.9819
Epoch 15/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0859 - acc: 0.9738 -
val_loss: 0.0650 - val_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0810 - acc: 0.9760 -
val_loss: 0.0644 - val_acc: 0.9805
Epoch 17/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0815 - acc: 0.9752 -
val_loss: 0.0647 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0766 - acc: 0.9765 -
```

```
val_loss: 0.0626 - val_acc: 0.9816
Epoch 19/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0736 - acc: 0.9776 -
val_loss: 0.0617 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0711 - acc: 0.9780 -
val_loss: 0.0653 - val_acc: 0.9820
CPU times: user 1min 44s, sys: 8.53 s, total: 1min 52s
Wall time: 1min 21s
```

In [16]:

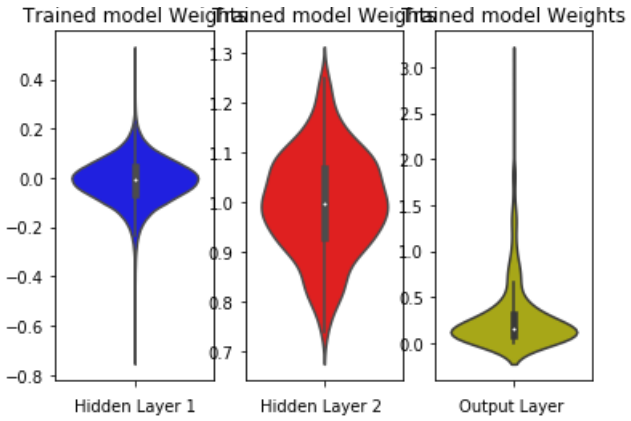
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
Test score: 0.06532301284242421
Test accuracy: 0.982
```



In [17]:

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

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

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

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

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



5 hidden layer - MLP + ReLU activation + ADAM optimizer + Batch normalization + Dropout

In [18]:

```
%%time
model_relu = Sequential()
model_relu.add(Dense(300, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(180, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(56, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 300)               235500
_________________________________________________________________
batch_normalization_6 (Batch (None, 300)               1200
_________________________________________________________________
dropout_6 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_9 (Dense)              (None, 180)               54180
_________________________________________________________________
batch_normalization_7 (Batch (None, 180)               720
_________________________________________________________________
dropout_7 (Dropout)          (None, 180)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               23168
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
```

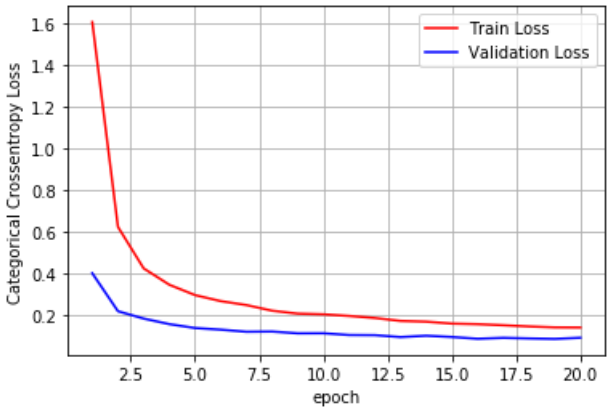
```
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 56)                3640
_________________________________________________________________
batch_normalization_10 (Batc (None, 56)                224
_________________________________________________________________
dropout_10 (Dropout)         (None, 56)                0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                570
=================================================================
Total params: 328,226
Trainable params: 326,770
Non-trainable params: 1,456
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 117us/step - loss: 1.6076 - acc: 0.4736 -
val_loss: 0.4020 - val_acc: 0.8917
Epoch 2/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.6244 - acc: 0.8169 -
val_loss: 0.2197 - val_acc: 0.9374
Epoch 3/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.4253 - acc: 0.8873 -
val_loss: 0.1843 - val_acc: 0.9505
Epoch 4/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.3466 - acc: 0.9116 -
val_loss: 0.1577 - val_acc: 0.9588
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.2970 - acc: 0.9258 -
val_loss: 0.1390 - val_acc: 0.9640
Epoch 6/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.2681 - acc: 0.9333 -
val_loss: 0.1312 - val_acc: 0.9665
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.2493 - acc: 0.9396 -
val_loss: 0.1211 - val_acc: 0.9699
Epoch 8/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.2222 - acc: 0.9461 -
val_loss: 0.1221 - val_acc: 0.9702
Epoch 9/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.2080 - acc: 0.9493 -
val_loss: 0.1135 - val_acc: 0.9724
Epoch 10/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.2048 - acc: 0.9503 -
val_loss: 0.1138 - val_acc: 0.9712
Epoch 11/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1970 - acc: 0.9525 -
val_loss: 0.1055 - val_acc: 0.9742
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1871 - acc: 0.9548 -
val_loss: 0.1047 - val_acc: 0.9744
Epoch 13/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1731 - acc: 0.9592 -
val_loss: 0.0955 - val_acc: 0.9775
Epoch 14/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1695 - acc: 0.9591 -
val_loss: 0.1020 - val_acc: 0.9754
Epoch 15/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1603 - acc: 0.9605 -
val_loss: 0.0960 - val_acc: 0.9771
Epoch 16/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.1575 - acc: 0.9613 -
val_loss: 0.0874 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1523 - acc: 0.9630 -
val_loss: 0.0919 - val_acc: 0.9789
Epoch 18/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1466 - acc: 0.9646 -
val_loss: 0.0889 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1421 - acc: 0.9661 -
val_loss: 0.0867 - val_acc: 0.9791
Epoch 20/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1415 - acc: 0.9659 -
val_loss: 0.0923 - val_acc: 0.9781
```

```
CPU times: user 2min 23s, sys: 11.8 s, total: 2min 35s
Wall time: 1min 52s
```

In [19]:

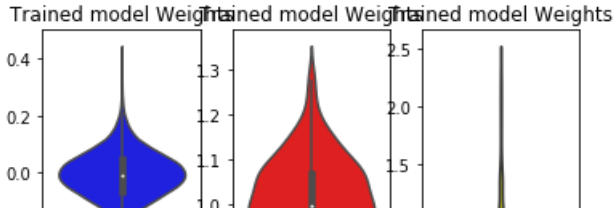
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

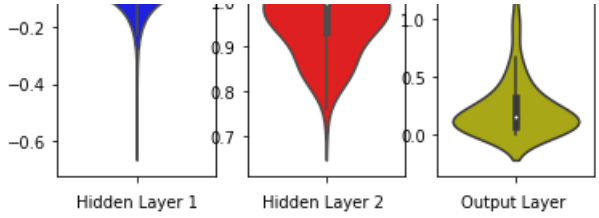
```
Test score: 0.09230942500107922
Test accuracy: 0.9781
```



In [20]:

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

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

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

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

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





Summary

In [22]:

```
from prettytable import PrettyTable
pt = PrettyTable()
pt.field_names = ["Activation", "optimizer", "BatchNormalization", "Dropout","Layers","Test score",
"Test accuracy"]
pt.add_row(["relu", "adam", "Yes", "0.5", "784(Input)-480(L1)-256(L2)-
10(Output)","0.054","0.983"])
pt.add_row(["relu", "adam", "Yes", "0.5", "784(Input)-360(L1)-256(L2)-120(L3)-10(Output)","0.065","
0.982"])
pt.add_row(["relu", "adam", "Yes", "0.5", "784(Input)-360(L1)-180(L2)-128(L3)-64(L4)-56(L5)-
10(Output)","0.092","0.978"])
print(pt)
```

```
+------------+-----------+--------------------+---------+-----------------------------------------
---------------+------------+---------------+
| Activation | optimizer | BatchNormalization | Dropout |                                Layers
| Test score | Test accuracy |
+------------+-----------+--------------------+---------+-----------------------------------------
---------------+------------+---------------+
|    relu    |    adam   |        Yes         |   0.5   |             784(Input)-480(L1)-256(L2)-10(
utput)             |   0.054    |     0.983     |
|    relu    |    adam   |        Yes         |   0.5   |         784(Input)-360(L1)-256(L2)-120(L3)
10(Output)         |   0.065    |     0.982     |
|    relu    |    adam   |        Yes         |   0.5   | 784(Input)-360(L1)-180(L2)-128(L3)-
64(L4)-56(L5)-10(Output) |   0.092    |     0.978     |
+------------+-----------+--------------------+---------+-----------------------------------------
---------------+------------+---------------+
```