49_14_deep_neural_keras_minst_50epoch

August 19, 2018

1 Keras -- MLPs on MNIST

1.1 Objective

-Try with 2,3,5 hidden layer
-for each cases try dropout batch normalize use RELU activation and adam optimizer for all
-For every model plot train/test against epoch

```
In [0]: # if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use t
        from keras.utils import np_utils
        from keras.datasets import mnist
        import seaborn as sns
        from keras.initializers import RandomNormal
```

```
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, vy, 'b', label="Validation Loss")
            ax.plot(x, ty, 'r', label="Train Loss")
            plt.legend()
            plt.grid()
            fig.canvas.draw()
```

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

```
In [32]: #smuk take sample
         #X_train=X_train[0:500]
         #y_train=y_train[0:500]
         #X_test=X_test[501:600]
         #y_test=y_test[501:600]
         nb_epoch = 50
         print("Number of training examples :", X_train.shape[0], "and each image is of shape
         print("Number of training examples :", X_test.shape[0], "and each image is of shape (
```

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

```
In [0]: # if you observe the input shape its 3 dimensional vector
        # for each image we have a (28*28) vector
        # we will convert the (28*28) vector into single dimensional vector of 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])
```

```
In [34]: # after converting the input images from 3d to 2d vectors

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

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

```
In [35]: # An example data point
         print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
```

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

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

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

```
In [37]: # example data point after normlizing
         print(X_train[0])
```

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         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.         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 [38]: # here we are having a class number for each image
         print("Class label of first image :", y_train[0])

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

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

         print("After converting the output into a vector : ",Y_train[0])
```

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

Softmax classifier

```
In [0]: # https://keras.io/getting-started/sequential-model-guide/

        # The Sequential model is a linear stack of layers.
        # you can create a Sequential model by passing a list of layer instances to the constr

        # model = Sequential([
        #     Dense(32, input_shape=(784,)),
        #     Activation('relu'),
        #     Dense(10),
        #     Activation('softmax'),
        # ])
```

```
# You can also simply add layers via the .add() method:

# model = Sequential()
# model.add(Dense(32, input_dim=784))
# model.add(Activation('relu'))

###

# https://keras.io/layers/core/

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_re
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bias) where
# activation is the element-wise activation function passed as the activation argument
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bias is True).

# output = activation(dot(input, kernel) + bias)  => y = activation(WT. X + b)

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the activatio

# from keras.layers import Activation, Dense

# model.add(Dense(64))
# model.add(Activation('tanh'))

# This is equivalent to:
# model.add(Dense(64, activation='tanh'))

# there are many activation functions ar available ex: tanh, relu, softmax


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
```

MLP + ReLU + ADAM

```
In [41]: model_relu = Sequential()
         model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
         model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
         model_relu.add(Dense(output_dim, activation='softmax'))

         print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_16 (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 1/50
60000/60000 [==============================] - 4s 73us/step - loss: 0.2321 - acc: 0.9305 - val_
Epoch 2/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0855 - acc: 0.9740 - val_
Epoch 3/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0559 - acc: 0.9824 - val_
Epoch 4/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0357 - acc: 0.9892 - val_
Epoch 5/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0293 - acc: 0.9906 - val_
Epoch 6/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0226 - acc: 0.9931 - val_
Epoch 7/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0163 - acc: 0.9945 - val_
Epoch 8/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0149 - acc: 0.9952 - val_
Epoch 9/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0118 - acc: 0.9959 - val_
Epoch 10/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0122 - acc: 0.9960 - val_
Epoch 11/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0128 - acc: 0.9958 - val_
```

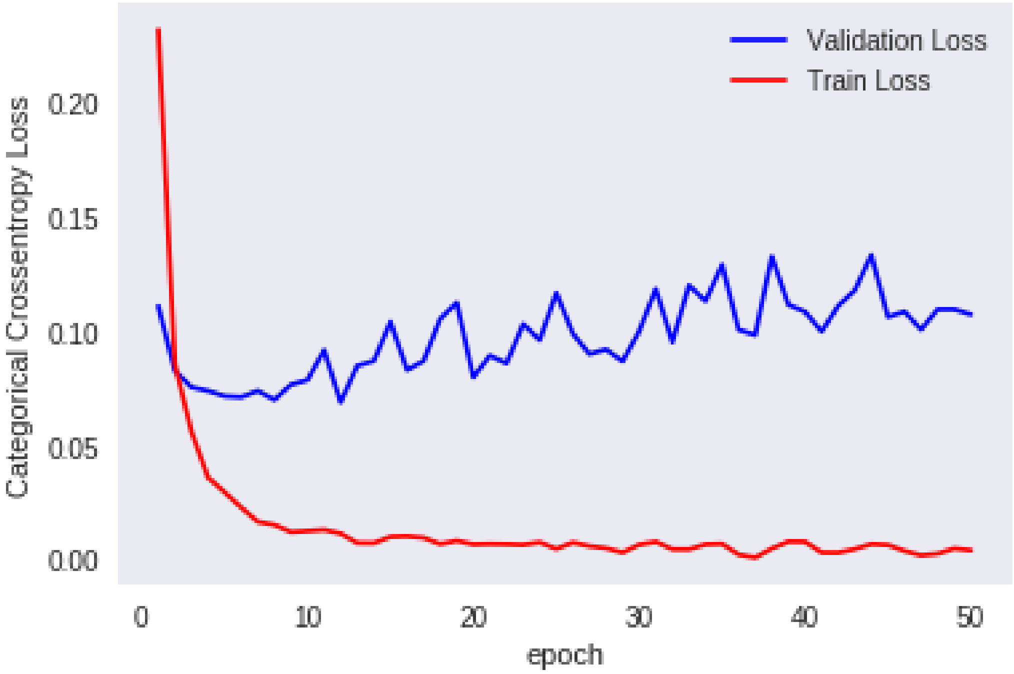
```
Epoch 12/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0111 - acc: 0.9961 - val_
Epoch 13/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0070 - acc: 0.9978 - val_
Epoch 14/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0069 - acc: 0.9977 - val_
Epoch 15/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0098 - acc: 0.9967 - val_
Epoch 16/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0100 - acc: 0.9968 - val_
Epoch 17/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0093 - acc: 0.9966 - val_
Epoch 18/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0065 - acc: 0.9978 - val_
Epoch 19/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0080 - acc: 0.9972 - val_
Epoch 20/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0064 - acc: 0.9980 - val_
Epoch 21/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0066 - acc: 0.9979 - val_
Epoch 22/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0065 - acc: 0.9980 - val_
Epoch 23/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0062 - acc: 0.9982 - val_
Epoch 24/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0073 - acc: 0.9978 - val_
Epoch 25/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0044 - acc: 0.9985 - val_
Epoch 26/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0072 - acc: 0.9977 - val_
Epoch 27/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0056 - acc: 0.9983 - val_
Epoch 28/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0047 - acc: 0.9985 - val_
Epoch 29/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0028 - acc: 0.9991 - val_
Epoch 30/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0062 - acc: 0.9981 - val_
Epoch 31/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0075 - acc: 0.9977 - val_
Epoch 32/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0042 - acc: 0.9988 - val_
Epoch 33/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0042 - acc: 0.9987 - val_
Epoch 34/50
60000/60000 [==============================] - 3s 58us/step - loss: 0.0064 - acc: 0.9981 - val_
Epoch 35/50
60000/60000 [==============================] - 3s 58us/step - loss: 0.0066 - acc: 0.9978 - val_
```

```
Epoch 36/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0019 - acc: 0.9993 - val_
Epoch 37/50
60000/60000 [==============================] - 4s 63us/step - loss: 5.6054e-04 - acc: 0.9999 -
Epoch 38/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0046 - acc: 0.9987 - val_
Epoch 39/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0076 - acc: 0.9979 - val_
Epoch 40/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0075 - acc: 0.9978 - val_
Epoch 41/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0028 - acc: 0.9991 - val_
Epoch 42/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0027 - acc: 0.9992 - val_
Epoch 43/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0043 - acc: 0.9986 - val_
Epoch 44/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0065 - acc: 0.9982 - val_
Epoch 45/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0061 - acc: 0.9981 - val_
Epoch 46/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0035 - acc: 0.9991 - val_
Epoch 47/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0015 - acc: 0.9995 - val_
Epoch 48/50
60000/60000 [==============================] - 4s 59us/step - loss: 0.0023 - acc: 0.9994 - val_
Epoch 49/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0046 - acc: 0.9987 - val_
Epoch 50/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0038 - acc: 0.9989 - val_
```

```
In [42]: score = model_relu.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])

         import pandas as pd
         scoretrain = model_relu.evaluate(X_train, Y_train, verbose=0)
         aa=pd.DataFrame()
         bb=pd.DataFrame({'type':['3 Layer '],'Test Score':[score[0]],'Test Accuracy':[score[1
                            'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
         print('Score',bb)
         aa=aa.append(bb)

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

         # list of epoch numbers
```

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of


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

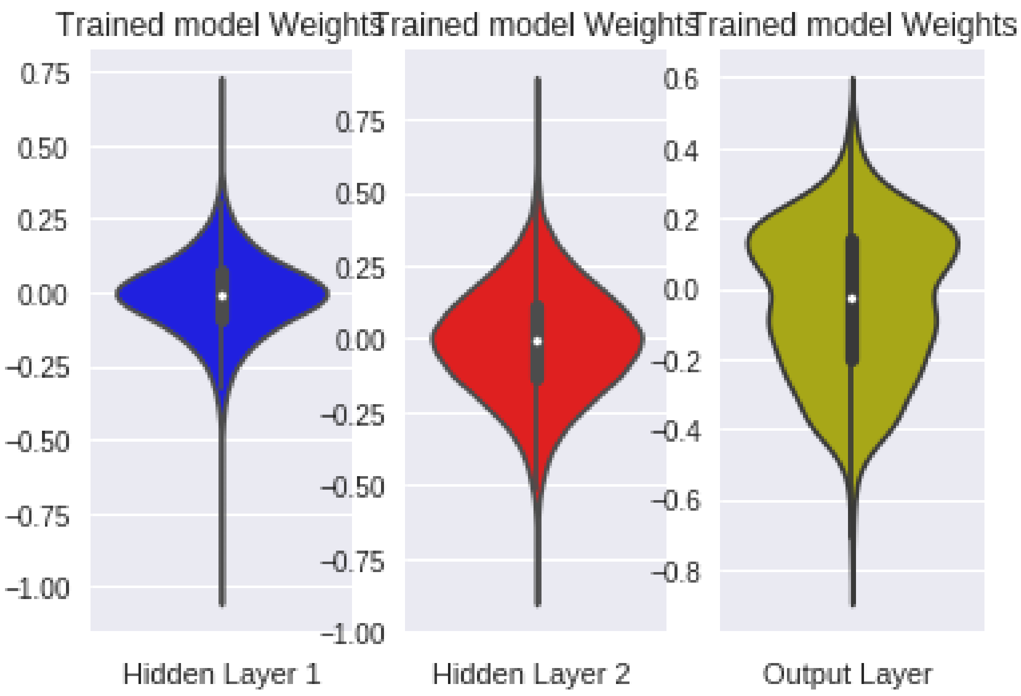
```
Test score: 0.107209661454717
Test accuracy: 0.9826
Score     Test Accuracy  Test Score  Train Accuracy  Train Score      type
0          0.9826     0.10721        0.999683     0.000822  3 Layer
```



```
In [43]: print(aa)
         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()
```

```
   Test Accuracy  Test Score  Train Accuracy  Train Score     type
0         0.9826     0.10721        0.999683     0.000822  3 Layer
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

In [44]:
```
# Multilayer perceptron

# https://intoli.com/blog/neural-network-initialization/
# If we sample weights from a normal distribution N(0,) we satisfy this condition wit
# h1 =>  =(2/(ni+ni+1) = 0.039  => N(0,) = N(0,0.039)
# h2 =>  =(2/(ni+ni+1) = 0.055  => N(0,) = N(0,0.055)
# h1 =>  =(2/(ni+ni+1) = 0.120  => N(0,) = N(0,0.120)

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
model_batch.add(BatchNormalization())

model_batch.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

```
In [45]: model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

         history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.1909 - acc: 0.9435 - val
Epoch 2/50
60000/60000 [==============================] - 7s 112us/step - loss: 0.0707 - acc: 0.9794 - val
Epoch 3/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.0473 - acc: 0.9859 - val
Epoch 4/50
60000/60000 [==============================] - 7s 111us/step - loss: 0.0335 - acc: 0.9899 - val
Epoch 5/50
60000/60000 [==============================] - 7s 108us/step - loss: 0.0230 - acc: 0.9931 - val
Epoch 6/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0190 - acc: 0.9944 - val
Epoch 7/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0179 - acc: 0.9944 - val
Epoch 8/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.0164 - acc: 0.9951 - val
Epoch 9/50
60000/60000 [==============================] - 6s 101us/step - loss: 0.0136 - acc: 0.9959 - val
Epoch 10/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0112 - acc: 0.9965 - val
Epoch 11/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0117 - acc: 0.9964 - val
Epoch 12/50
60000/60000 [==============================] - 6s 103us/step - loss: 0.0101 - acc: 0.9968 - val
Epoch 13/50
```

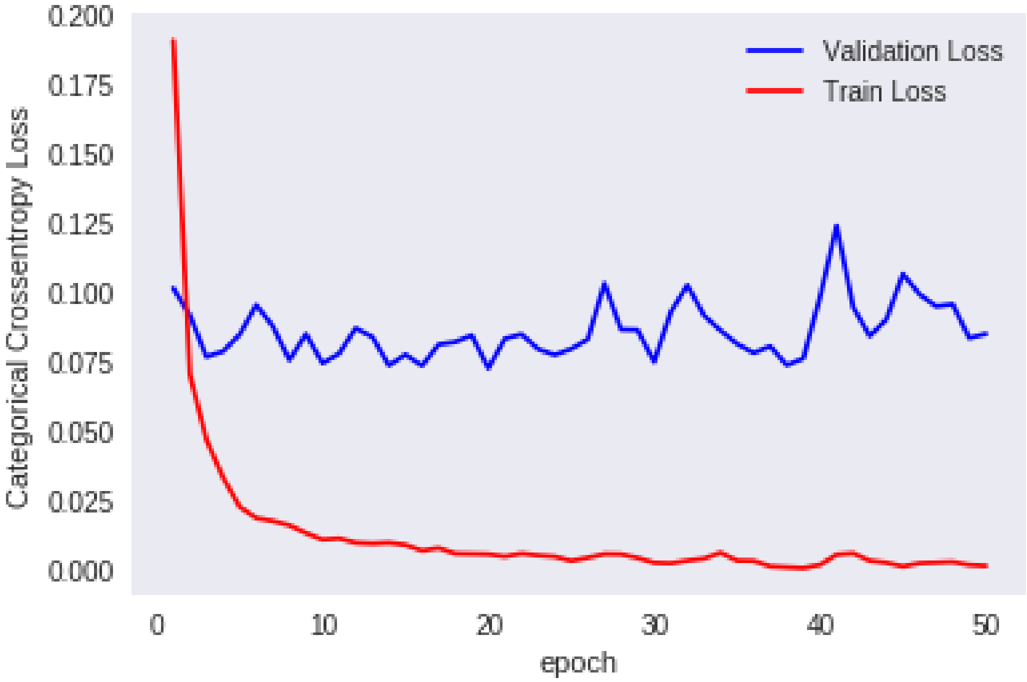
```
60000/60000 [==============================] - 7s 112us/step - loss: 0.0098 - acc: 0.9967 - val
Epoch 14/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0101 - acc: 0.9966 - val
Epoch 15/50
60000/60000 [==============================] - 7s 108us/step - loss: 0.0093 - acc: 0.9970 - val
Epoch 16/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0073 - acc: 0.9980 - val
Epoch 17/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0082 - acc: 0.9972 - val
Epoch 18/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.0062 - acc: 0.9981 - val
Epoch 19/50
60000/60000 [==============================] - 6s 101us/step - loss: 0.0061 - acc: 0.9979 - val
Epoch 20/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.0060 - acc: 0.9979 - val
Epoch 21/50
60000/60000 [==============================] - 6s 100us/step - loss: 0.0052 - acc: 0.9983 - val
Epoch 22/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0062 - acc: 0.9980 - val
Epoch 23/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.0056 - acc: 0.9982 - val
Epoch 24/50
60000/60000 [==============================] - 6s 103us/step - loss: 0.0051 - acc: 0.9984 - val
Epoch 25/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0036 - acc: 0.9990 - val
Epoch 26/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0047 - acc: 0.9986 - val
Epoch 27/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0061 - acc: 0.9981 - val
Epoch 28/50
60000/60000 [==============================] - 6s 103us/step - loss: 0.0060 - acc: 0.9979 - val
Epoch 29/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0047 - acc: 0.9982 - val
Epoch 30/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0029 - acc: 0.9991 - val
Epoch 31/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0027 - acc: 0.9992 - val
Epoch 32/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0037 - acc: 0.9988 - val
Epoch 33/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0044 - acc: 0.9985 - val
Epoch 34/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0066 - acc: 0.9977 - val
Epoch 35/50
60000/60000 [==============================] - 6s 101us/step - loss: 0.0037 - acc: 0.9988 - val
Epoch 36/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0036 - acc: 0.9989 - val
Epoch 37/50
```

```
60000/60000 [==============================] - 6s 102us/step - loss: 0.0016 - acc: 0.9995 - val
Epoch 38/50
60000/60000 [==============================] - 6s 101us/step - loss: 0.0013 - acc: 0.9997 - val
Epoch 39/50
60000/60000 [==============================] - 6s 104us/step - loss: 9.6881e-04 - acc: 0.9998 -
Epoch 40/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0022 - acc: 0.9994 - val
Epoch 41/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0059 - acc: 0.9979 - val
Epoch 42/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0063 - acc: 0.9979 - val
Epoch 43/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0036 - acc: 0.9987 - val
Epoch 44/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0029 - acc: 0.9991 - val
Epoch 45/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0016 - acc: 0.9995 - val
Epoch 46/50
60000/60000 [==============================] - 6s 103us/step - loss: 0.0028 - acc: 0.9990 - val
Epoch 47/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0030 - acc: 0.9992 - val
Epoch 48/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.0032 - acc: 0.9989 - val
Epoch 49/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0021 - acc: 0.9993 - val
Epoch 50/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0018 - acc: 0.9994 - val
```

```
In [46]: score = model_batch.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])


         scoretrain = model_batch.evaluate(X_train, Y_train, verbose=0)
         bb=pd.DataFrame({'type':['3 Layer batch norm'],'Test Score':[score[0]],'Test Accuracy
                             'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
         aa=aa.append(bb)

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

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

         # print(history.history.keys())
         # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
         # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,
```

```
# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of

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

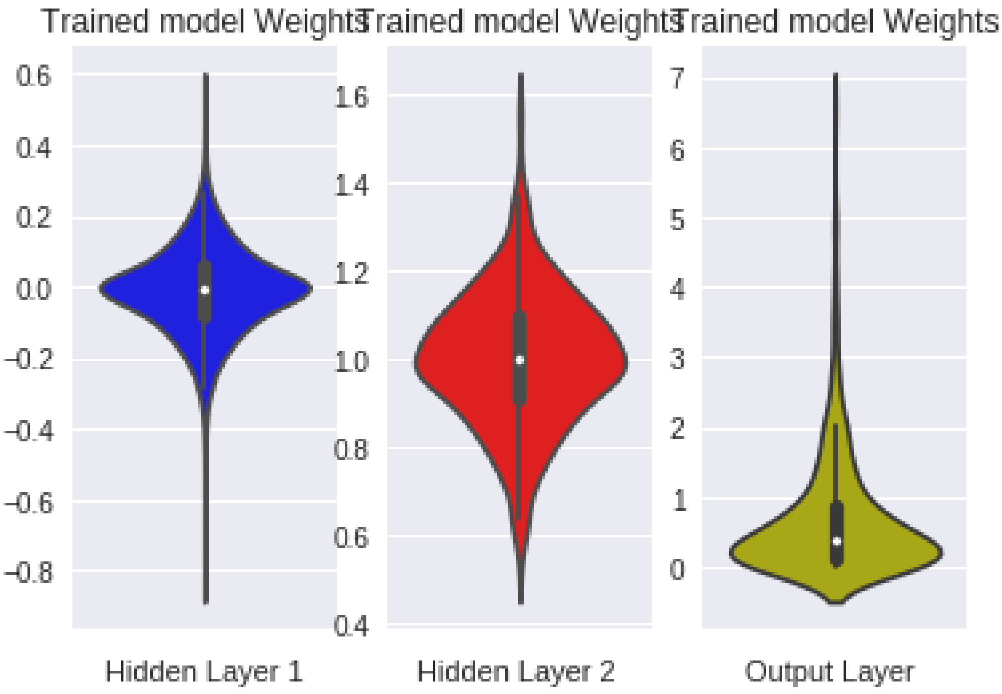
```
Test score: 0.08531817659379323
Test accuracy: 0.9826
```



```
In [47]: w_after = model_batch.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
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



5. MLP + Dropout + AdamOptimizer

In [48]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
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_20 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [49]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
```

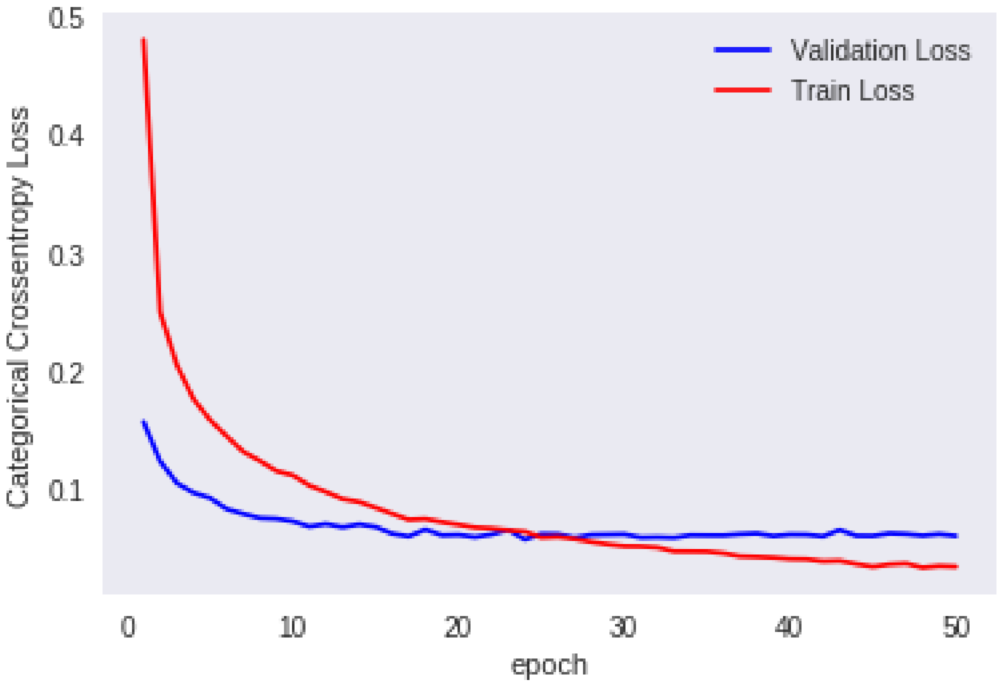
```
60000/60000 [==============================] - 8s 126us/step - loss: 0.4800 - acc: 0.8543 - val
Epoch 2/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.2490 - acc: 0.9253 - val
Epoch 3/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.2043 - acc: 0.9382 - val
Epoch 4/50
60000/60000 [==============================] - 7s 108us/step - loss: 0.1756 - acc: 0.9467 - val
Epoch 5/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.1578 - acc: 0.9524 - val
Epoch 6/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.1441 - acc: 0.9567 - val
Epoch 7/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.1309 - acc: 0.9603 - val
Epoch 8/50
60000/60000 [==============================] - 7s 110us/step - loss: 0.1232 - acc: 0.9635 - val
Epoch 9/50
60000/60000 [==============================] - 7s 111us/step - loss: 0.1145 - acc: 0.9646 - val
Epoch 10/50
60000/60000 [==============================] - 7s 114us/step - loss: 0.1109 - acc: 0.9659 - val
Epoch 11/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.1022 - acc: 0.9685 - val
Epoch 12/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0968 - acc: 0.9695 - val
Epoch 13/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.0909 - acc: 0.9724 - val
Epoch 14/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0884 - acc: 0.9726 - val
Epoch 15/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0835 - acc: 0.9739 - val
Epoch 16/50
60000/60000 [==============================] - 7s 111us/step - loss: 0.0783 - acc: 0.9754 - val
Epoch 17/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0733 - acc: 0.9770 - val
Epoch 18/50
60000/60000 [==============================] - 7s 116us/step - loss: 0.0739 - acc: 0.9765 - val
Epoch 19/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0712 - acc: 0.9778 - val
Epoch 20/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0691 - acc: 0.9780 - val
Epoch 21/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0666 - acc: 0.9786 - val
Epoch 22/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0659 - acc: 0.9785 - val
Epoch 23/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0643 - acc: 0.9797 - val
Epoch 24/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0631 - acc: 0.9795 - val
Epoch 25/50
```

```
60000/60000 [==============================] - 7s 110us/step - loss: 0.0582 - acc: 0.9818 - val
Epoch 26/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0587 - acc: 0.9810 - val
Epoch 27/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.0570 - acc: 0.9815 - val
Epoch 28/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0544 - acc: 0.9826 - val
Epoch 29/50
60000/60000 [==============================] - 7s 110us/step - loss: 0.0524 - acc: 0.9836 - val
Epoch 30/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0508 - acc: 0.9836 - val
Epoch 31/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0506 - acc: 0.9840 - val
Epoch 32/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0497 - acc: 0.9838 - val
Epoch 33/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0464 - acc: 0.9851 - val
Epoch 34/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0463 - acc: 0.9849 - val
Epoch 35/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0462 - acc: 0.9851 - val
Epoch 36/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0448 - acc: 0.9856 - val
Epoch 37/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0421 - acc: 0.9862 - val
Epoch 38/50
60000/60000 [==============================] - 7s 109us/step - loss: 0.0417 - acc: 0.9861 - val
Epoch 39/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0411 - acc: 0.9867 - val
Epoch 40/50
60000/60000 [==============================] - 6s 108us/step - loss: 0.0399 - acc: 0.9871 - val
Epoch 41/50
60000/60000 [==============================] - 6s 104us/step - loss: 0.0397 - acc: 0.9871 - val
Epoch 42/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0380 - acc: 0.9874 - val
Epoch 43/50
60000/60000 [==============================] - 6s 106us/step - loss: 0.0384 - acc: 0.9881 - val
Epoch 44/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0357 - acc: 0.9883 - val
Epoch 45/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0334 - acc: 0.9888 - val
Epoch 46/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0354 - acc: 0.9886 - val
Epoch 47/50
60000/60000 [==============================] - 6s 107us/step - loss: 0.0361 - acc: 0.9884 - val
Epoch 48/50
60000/60000 [==============================] - 7s 110us/step - loss: 0.0326 - acc: 0.9889 - val
Epoch 49/50
```

```
60000/60000 [==============================] - 6s 103us/step - loss: 0.0340 - acc: 0.9891 - va
Epoch 50/50
60000/60000 [==============================] - 6s 105us/step - loss: 0.0334 - acc: 0.9893 - va
```

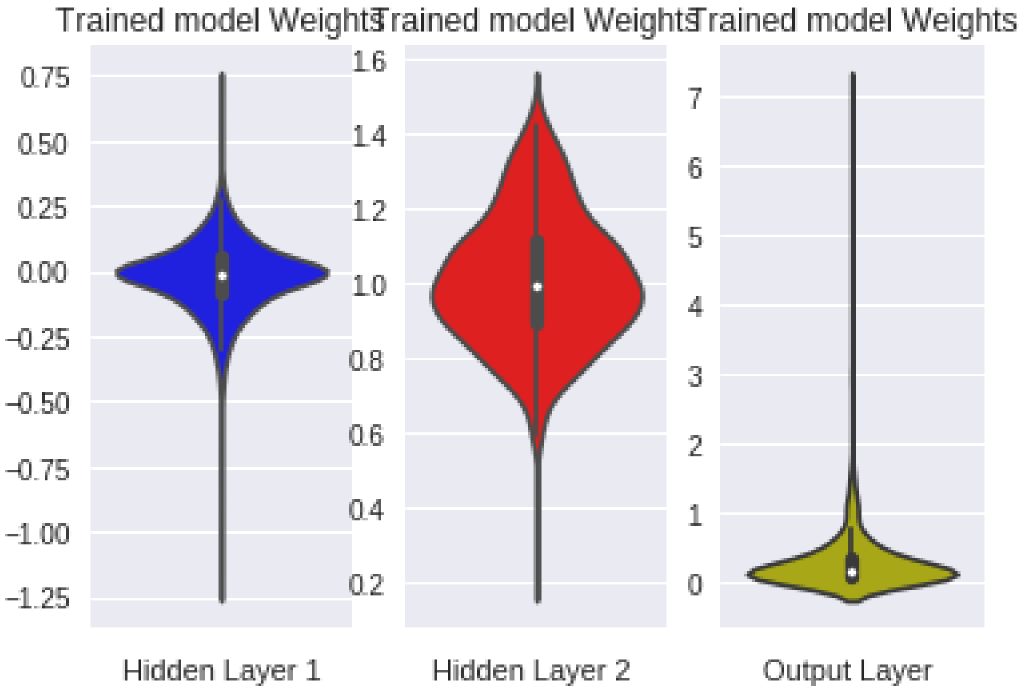
```
In [50]: score = model_drop.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])
         scoretrain = model_drop.evaluate(X_train, Y_train, verbose=0)
         bb=pd.DataFrame({'type':['3 Layer batch norm dropout'],'Test Score':[score[0]],'Test
                            'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
         aa=aa.append(bb)
         fig,ax = plt.subplots(1,1)
         ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

         # print(history.history.keys())
         # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
         # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

         # we will get val_loss and val_acc only when you pass the paramter validation_data
         # val_loss : validation loss
         # val_acc : validation accuracy

         # loss : training loss
         # acc : train accuracy
         # for each key in histrory.histrory we will have a list of length equal to number of

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

```
Test score: 0.05944320905764616
Test accuracy: 0.9851
```



```
In [51]: 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
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



Hyper-parameter tuning of Keras models using Sklearn

```
In [0]: #SMUK last
        from keras.optimizers import Adam,RMSprop,SGD
        def best_hyperparameters(activ,drop,norm):


            model = Sequential()
            model.add(Dense(512, activation=activ, input_shape=(input_dim,), kernel_initialize
            if norm=='True':
                model.add(BatchNormalization())
            model.add(Dropout(drop))

            model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(mean=0.0, s
            if norm=='True':
                model.add(BatchNormalization())
            model.add(Dropout(drop))

            model.add(Dense(output_dim, activation='softmax'))
```

```
        model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='ada

        return model
```

In [0]:
```
#SMUK at last
# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-

activ = ['sigmoid','relu']
drop=[.5,1]
norm=['True','False']

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

#model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch, batch_size=ba
param_grid = dict(activ=activ,drop=drop,norm=norm)

# if you are using CPU
# grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
# if you are using GPU dont use the n_jobs parameter

#grid = GridSearchCV(estimator=model, param_grid=param_grid)
#grid_result = grid.fit(X_train, Y_train)
```

In [0]:
```
#print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
#means = grid_result.cv_results_['mean_test_score']
#stds = grid_result.cv_results_['std_test_score']
#params = grid_result.cv_results_['params']
#for mean, stdev, param in zip(means, stds, params):
#    print("%f (%f) with: %r" % (mean, stdev, param))
```

In [54]:
```
#model with best parameter
#score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
#scoretrain = model_drop.evaluate(X_train, Y_train, verbose=0)
#bb=pd.DataFrame({'type':['3 Layer batch norm dropout'],'Test Score':[score[0]],'Test
#                    'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
#aa=aa.append(bb)
```

```
Test score: 0.05944320905764616
Test accuracy: 0.9851
```

2 Try with 3 hidden layer

In [55]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
```

```
#smuk
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve



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

import pandas as pd
scoretrain = model_relu.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['4 Layer '],'Test Score':[score[0]],'Test Accuracy':[score[1
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
print('Score',bb)
aa=aa.append(bb)

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)


print(aa)
w_after = model_relu.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()
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='r')
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()


from keras.layers.normalization import BatchNormalization
# h1 =>  =(2/(ni+ni+1) = 0.039  => N(0,) = N(0,0.039)
# h2 =>  =(2/(ni+ni+1) = 0.055  => N(0,) = N(0,0.055)
# h1 =>  =(2/(ni+ni+1) = 0.120  => N(0,) = N(0,0.120)

model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
model_batch.add(BatchNormalization())

#smuk
model_batch.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.
model_batch.add(BatchNormalization())
#smuk
model_batch.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.
model_batch.add(BatchNormalization())

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


model_batch.summary()

model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, v
```

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


scoretrain = model_batch.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['4 Layer batch norm'],'Test Score':[score[0]],'Test Accuracy
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of

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


w_after = model_batch.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()
plt.title("Weight matrices after model trained batch norm")
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='r')
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()

#

# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initial
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()

model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve


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

```
print('Test accuracy:', score[1])
scoretrain = model_drop.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['4 Layer batch norm dropout'],'Test Score':[score[0]],'Test 
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
aa=aa.append(bb)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of 

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


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()
plt.title("Weight matrices after model trained batch norm and dropout")
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 ')
```

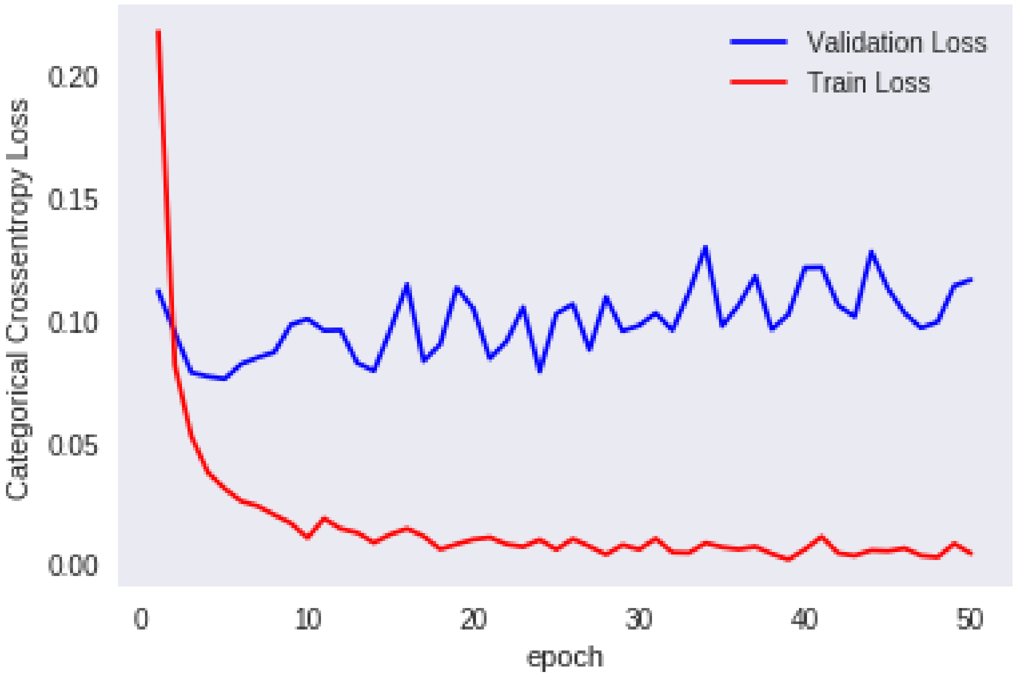
```
plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

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

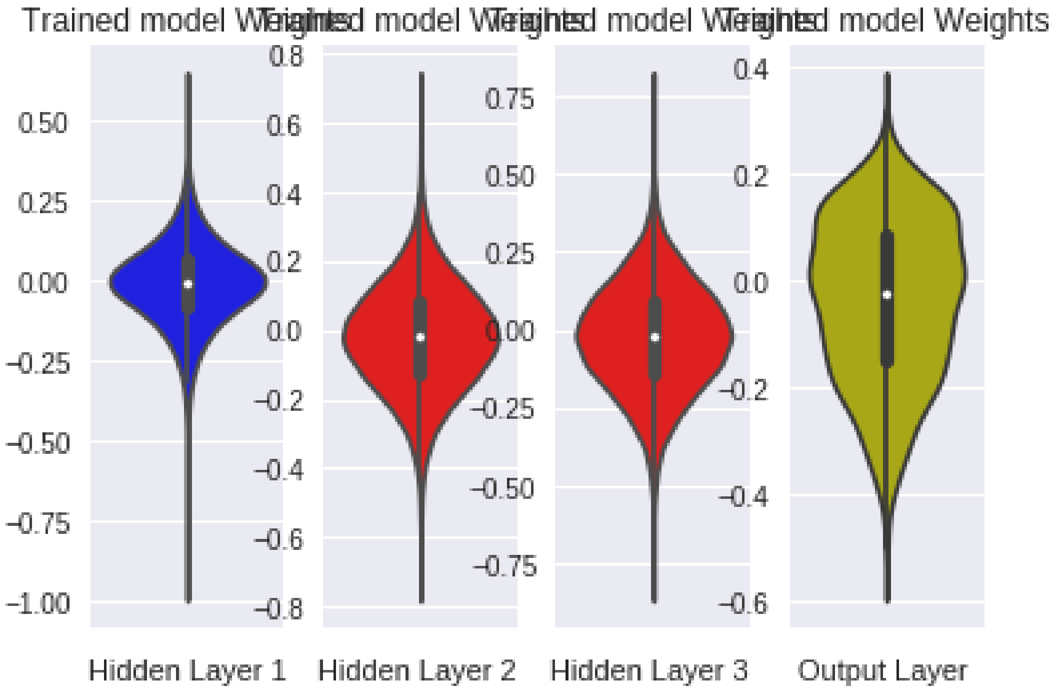
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_24 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_25 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_26 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 5s 83us/step - loss: 0.2178 - acc: 0.9335 - val_
Epoch 2/50
60000/60000 [==============================] - 4s 68us/step - loss: 0.0813 - acc: 0.9753 - val_
Epoch 3/50
60000/60000 [==============================] - 4s 69us/step - loss: 0.0521 - acc: 0.9835 - val_
Epoch 4/50
60000/60000 [==============================] - 4s 67us/step - loss: 0.0373 - acc: 0.9881 - val_
Epoch 5/50
60000/60000 [==============================] - 4s 68us/step - loss: 0.0307 - acc: 0.9899 - val_
Epoch 6/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0256 - acc: 0.9919 - val_
Epoch 7/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0236 - acc: 0.9920 - val_
Epoch 8/50
60000/60000 [==============================] - 4s 67us/step - loss: 0.0200 - acc: 0.9934 - val_
Epoch 9/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0165 - acc: 0.9945 - val_
Epoch 10/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0105 - acc: 0.9964 - val_
```

```
Epoch 11/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0185 - acc: 0.9940 - val_
Epoch 12/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0143 - acc: 0.9952 - val_
Epoch 13/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0127 - acc: 0.9960 - val_
Epoch 14/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0085 - acc: 0.9973 - val_
Epoch 15/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0120 - acc: 0.9962 - val_
Epoch 16/50
60000/60000 [==============================] - 4s 73us/step - loss: 0.0143 - acc: 0.9956 - val_
Epoch 17/50
60000/60000 [==============================] - 4s 68us/step - loss: 0.0112 - acc: 0.9964 - val_
Epoch 18/50
60000/60000 [==============================] - 4s 67us/step - loss: 0.0059 - acc: 0.9980 - val_
Epoch 19/50
60000/60000 [==============================] - 4s 68us/step - loss: 0.0080 - acc: 0.9977 - val_
Epoch 20/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0099 - acc: 0.9967 - val_
Epoch 21/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0107 - acc: 0.9965 - val_
Epoch 22/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0079 - acc: 0.9979 - val_
Epoch 23/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0069 - acc: 0.9980 - val_
Epoch 24/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0098 - acc: 0.9970 - val_
Epoch 25/50
60000/60000 [==============================] - 4s 63us/step - loss: 0.0057 - acc: 0.9983 - val_
Epoch 26/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0101 - acc: 0.9969 - val_
Epoch 27/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0071 - acc: 0.9977 - val_
Epoch 28/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0037 - acc: 0.9989 - val_
Epoch 29/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0077 - acc: 0.9978 - val_
Epoch 30/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0058 - acc: 0.9984 - val_
Epoch 31/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0102 - acc: 0.9968 - val_
Epoch 32/50
60000/60000 [==============================] - 4s 69us/step - loss: 0.0046 - acc: 0.9987 - val_
Epoch 33/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0044 - acc: 0.9987 - val_
Epoch 34/50
60000/60000 [==============================] - 4s 68us/step - loss: 0.0084 - acc: 0.9977 - val_
```

```
Epoch 35/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0067 - acc: 0.9983 - val_
Epoch 36/50
60000/60000 [==============================] - 4s 67us/step - loss: 0.0059 - acc: 0.9984 - val_
Epoch 37/50
60000/60000 [==============================] - 4s 67us/step - loss: 0.0070 - acc: 0.9980 - val_
Epoch 38/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0040 - acc: 0.9988 - val_
Epoch 39/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0016 - acc: 0.9995 - val_
Epoch 40/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0058 - acc: 0.9984 - val_
Epoch 41/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0110 - acc: 0.9970 - val_
Epoch 42/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0042 - acc: 0.9986 - val_
Epoch 43/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0033 - acc: 0.9991 - val_
Epoch 44/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0054 - acc: 0.9986 - val_
Epoch 45/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0051 - acc: 0.9984 - val_
Epoch 46/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0063 - acc: 0.9982 - val_
Epoch 47/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0033 - acc: 0.9991 - val_
Epoch 48/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0026 - acc: 0.9993 - val_
Epoch 49/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0082 - acc: 0.9981 - val_
Epoch 50/50
60000/60000 [==============================] - 4s 65us/step - loss: 0.0041 - acc: 0.9987 - val_
Test score: 0.11627201831419585
Test accuracy: 0.9828
Score    Test Accuracy  Test Score  Train Accuracy  Train Score      type
0         0.9828    0.116272          0.9993     0.002566  4 Layer
   Test Accuracy  Test Score  Train Accuracy  Train Score  \
0         0.9826    0.107210        0.999683     0.000822
0         0.9826    0.085318        0.999283     0.002036
0         0.9851    0.059443        0.999300     0.002966
0         0.9828    0.116272        0.999300     0.002566

                         type
0                     3 Layer
0          3 Layer batch norm
0  3 Layer batch norm dropout
0                     4 Layer
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```

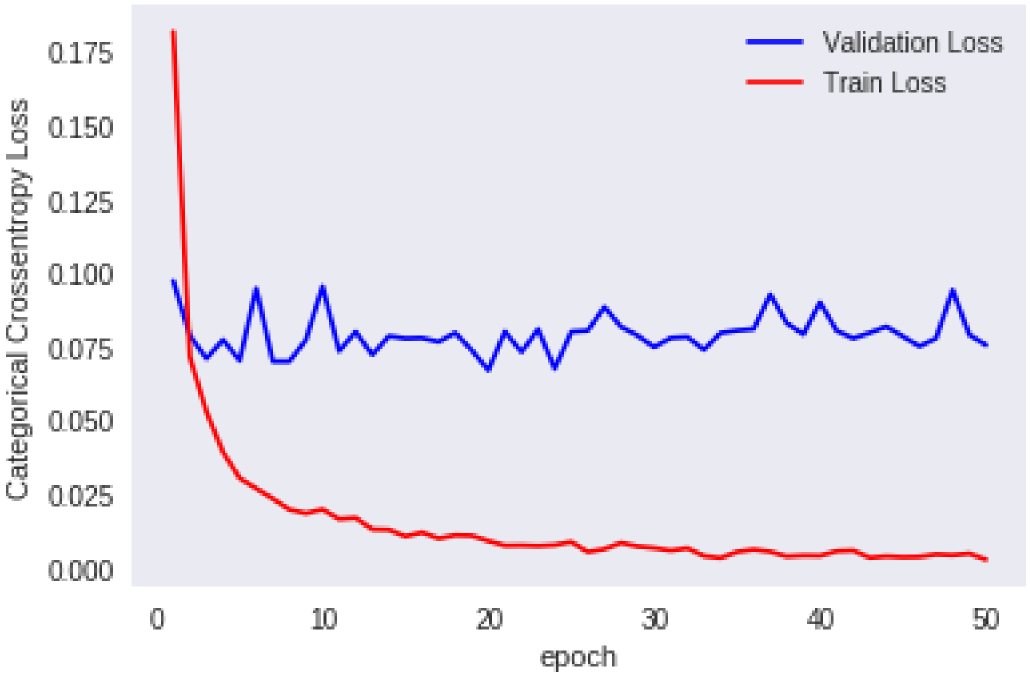


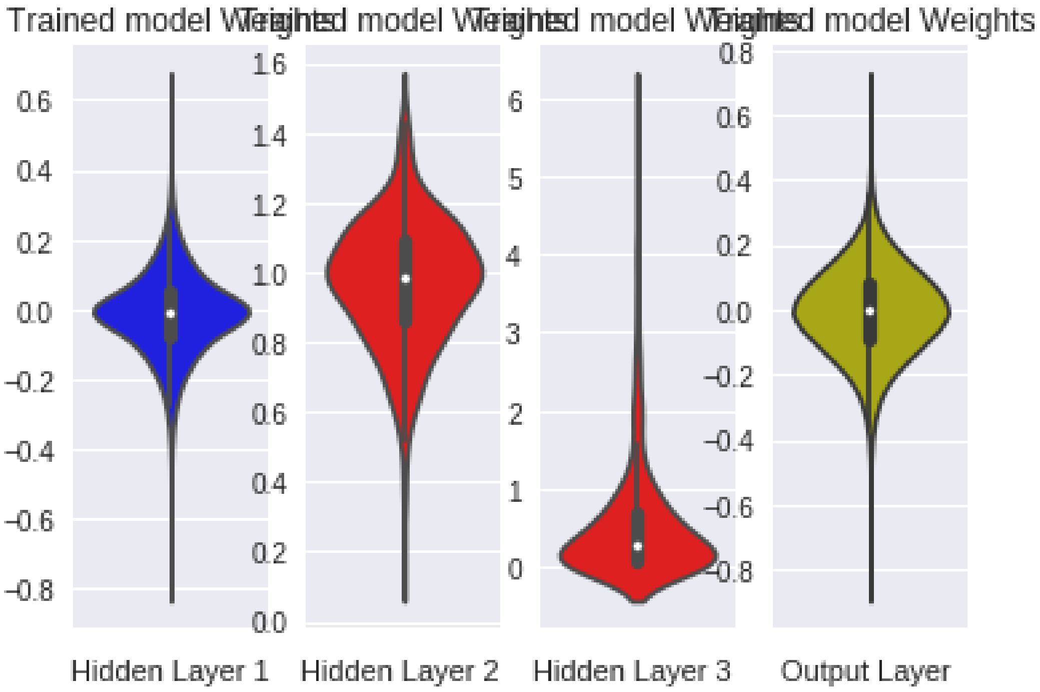


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048
_________________________________________________________________
dense_28 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dense_29 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_11 (Batc (None, 128)               512
_________________________________________________________________
dense_30 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 9s 151us/step - loss: 0.1816 - acc: 0.9447 - val
Epoch 2/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0714 - acc: 0.9781 - val
Epoch 3/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0529 - acc: 0.9826 - val
Epoch 4/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0390 - acc: 0.9876 - val
Epoch 5/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.0303 - acc: 0.9900 - val
Epoch 6/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0268 - acc: 0.9911 - val
Epoch 7/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.0234 - acc: 0.9917 - val
Epoch 8/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0197 - acc: 0.9934 - val
Epoch 9/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0184 - acc: 0.9939 - val
Epoch 10/50
60000/60000 [==============================] - 8s 131us/step - loss: 0.0197 - acc: 0.9934 - val
Epoch 11/50
60000/60000 [==============================] - 8s 131us/step - loss: 0.0164 - acc: 0.9944 - val
Epoch 12/50
```

```
60000/60000 [==============================] - 8s 129us/step - loss: 0.0169 - acc: 0.9942 - va
Epoch 13/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0129 - acc: 0.9955 - va
Epoch 14/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0128 - acc: 0.9957 - va
Epoch 15/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.0105 - acc: 0.9967 - va
Epoch 16/50
60000/60000 [==============================] - 8s 131us/step - loss: 0.0119 - acc: 0.9956 - va
Epoch 17/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0098 - acc: 0.9968 - va
Epoch 18/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.0109 - acc: 0.9961 - va
Epoch 19/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.0108 - acc: 0.9961 - va
Epoch 20/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0090 - acc: 0.9970 - va
Epoch 21/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0074 - acc: 0.9978 - va
Epoch 22/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0075 - acc: 0.9974 - va
Epoch 23/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0073 - acc: 0.9977 - va
Epoch 24/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0076 - acc: 0.9974 - va
Epoch 25/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0087 - acc: 0.9972 - va
Epoch 26/50
60000/60000 [==============================] - 8s 131us/step - loss: 0.0053 - acc: 0.9980 - va
Epoch 27/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.0063 - acc: 0.9980 - va
Epoch 28/50
60000/60000 [==============================] - 8s 126us/step - loss: 0.0084 - acc: 0.9972 - va
Epoch 29/50
60000/60000 [==============================] - 8s 133us/step - loss: 0.0072 - acc: 0.9978 - va
Epoch 30/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0066 - acc: 0.9975 - va
Epoch 31/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0058 - acc: 0.9980 - va
Epoch 32/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0065 - acc: 0.9979 - va
Epoch 33/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0039 - acc: 0.9987 - va
Epoch 34/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0032 - acc: 0.9989 - va
Epoch 35/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0053 - acc: 0.9982 - va
Epoch 36/50
```

```
60000/60000 [==============================] - 8s 130us/step - loss: 0.0061 - acc: 0.9981 - val
Epoch 37/50
60000/60000 [==============================] - 8s 127us/step - loss: 0.0054 - acc: 0.9981 - val
Epoch 38/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0038 - acc: 0.9989 - val
Epoch 39/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0041 - acc: 0.9989 - val
Epoch 40/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0040 - acc: 0.9986 - val
Epoch 41/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0055 - acc: 0.9980 - val
Epoch 42/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.0057 - acc: 0.9981 - val
Epoch 43/50
60000/60000 [==============================] - 8s 132us/step - loss: 0.0033 - acc: 0.9989 - val
Epoch 44/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0038 - acc: 0.9988 - val
Epoch 45/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0035 - acc: 0.9990 - val
Epoch 46/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0036 - acc: 0.9989 - val
Epoch 47/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0045 - acc: 0.9984 - val
Epoch 48/50
60000/60000 [==============================] - 8s 128us/step - loss: 0.0042 - acc: 0.9986 - val
Epoch 49/50
60000/60000 [==============================] - 8s 129us/step - loss: 0.0047 - acc: 0.9985 - val
Epoch 50/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0027 - acc: 0.9992 - val
Test score: 0.07546096074496722
Test accuracy: 0.9857
```



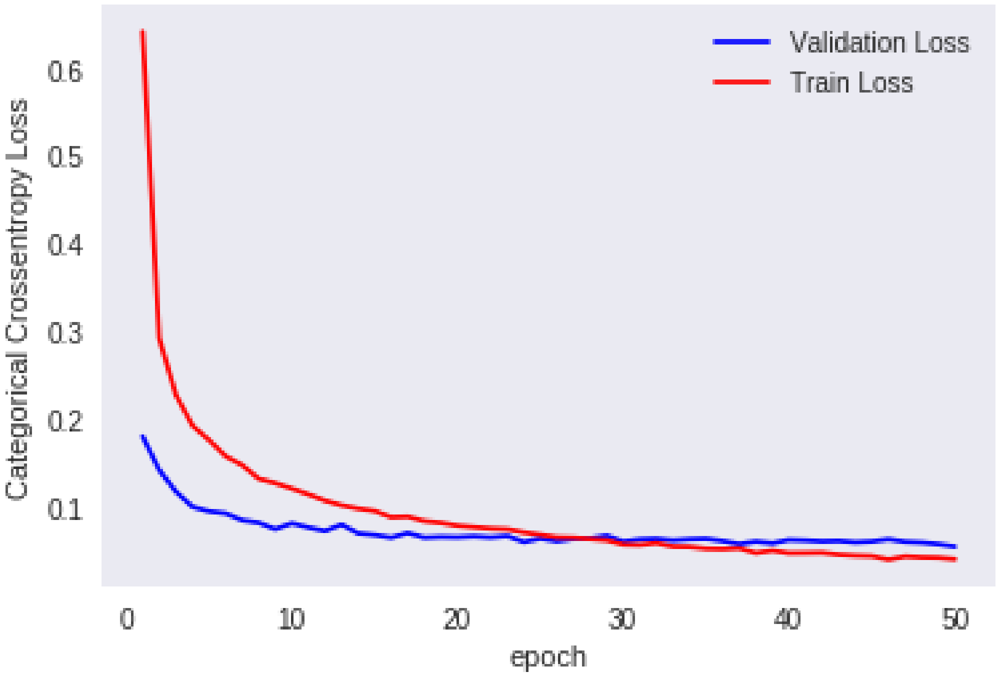


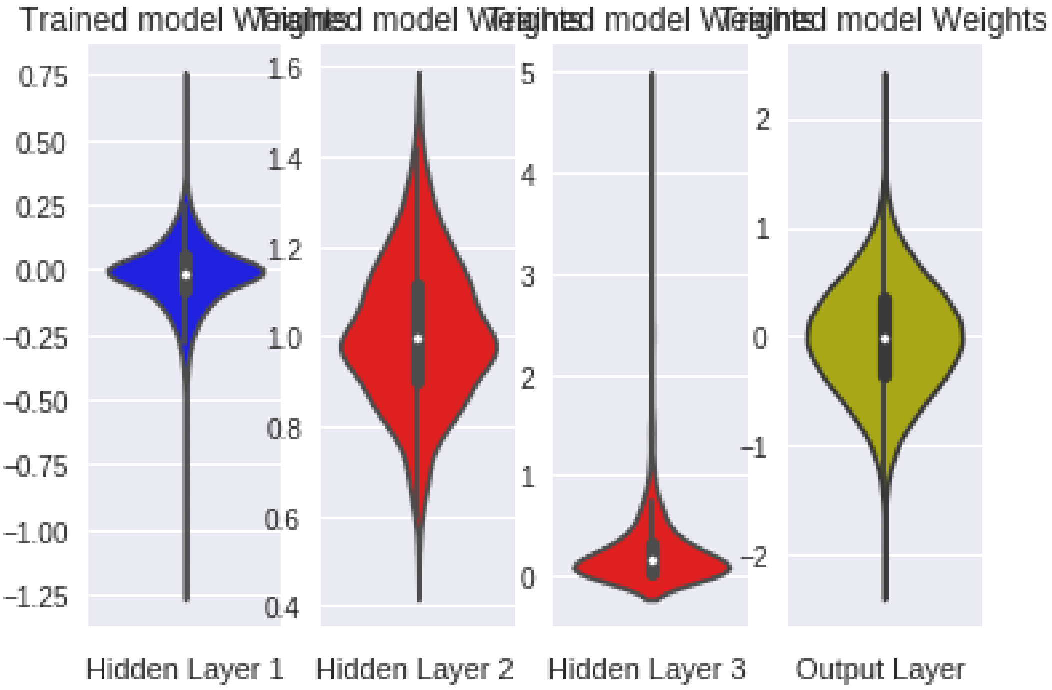
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_31 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_32 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_13 (Batc (None, 256)               1024
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_33 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_14 (Batc (None, 128)               512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 10s 169us/step - loss: 0.6409 - acc: 0.8022 - va
Epoch 2/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.2919 - acc: 0.9123 - val
Epoch 3/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.2271 - acc: 0.9331 - val
Epoch 4/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.1921 - acc: 0.9441 - val
Epoch 5/50
60000/60000 [==============================] - 8s 133us/step - loss: 0.1756 - acc: 0.9484 - val
Epoch 6/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.1572 - acc: 0.9533 - val
Epoch 7/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.1470 - acc: 0.9561 - val
Epoch 8/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.1316 - acc: 0.9602 - val
Epoch 9/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.1267 - acc: 0.9614 - val
Epoch 10/50
60000/60000 [==============================] - 8s 133us/step - loss: 0.1204 - acc: 0.9645 - val
Epoch 11/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.1138 - acc: 0.9663 - val
```

```
Epoch 12/50
60000/60000 [==============================] - 8s 136us/step - loss: 0.1067 - acc: 0.9681 - va
Epoch 13/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.1013 - acc: 0.9691 - va
Epoch 14/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.0976 - acc: 0.9706 - va
Epoch 15/50
60000/60000 [==============================] - 8s 137us/step - loss: 0.0949 - acc: 0.9713 - va
Epoch 16/50
60000/60000 [==============================] - 9s 144us/step - loss: 0.0878 - acc: 0.9738 - va
Epoch 17/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0884 - acc: 0.9732 - va
Epoch 18/50
60000/60000 [==============================] - 8s 136us/step - loss: 0.0833 - acc: 0.9745 - va
Epoch 19/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.0814 - acc: 0.9749 - va
Epoch 20/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0781 - acc: 0.9763 - va
Epoch 21/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0764 - acc: 0.9768 - va
Epoch 22/50
60000/60000 [==============================] - 8s 139us/step - loss: 0.0748 - acc: 0.9770 - va
Epoch 23/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0741 - acc: 0.9770 - va
Epoch 24/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.0704 - acc: 0.9781 - va
Epoch 25/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0677 - acc: 0.9791 - va
Epoch 26/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.0648 - acc: 0.9800 - va
Epoch 27/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.0644 - acc: 0.9802 - va
Epoch 28/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.0630 - acc: 0.9807 - va
Epoch 29/50
60000/60000 [==============================] - 8s 134us/step - loss: 0.0615 - acc: 0.9809 - va
Epoch 30/50
60000/60000 [==============================] - 8s 139us/step - loss: 0.0566 - acc: 0.9830 - va
Epoch 31/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.0561 - acc: 0.9829 - va
Epoch 32/50
60000/60000 [==============================] - 8s 135us/step - loss: 0.0583 - acc: 0.9820 - va
Epoch 33/50
60000/60000 [==============================] - 8s 136us/step - loss: 0.0542 - acc: 0.9836 - va
Epoch 34/50
60000/60000 [==============================] - 8s 136us/step - loss: 0.0539 - acc: 0.9832 - va
Epoch 35/50
60000/60000 [==============================] - 8s 133us/step - loss: 0.0519 - acc: 0.9841 - va
```

```
Epoch 36/50
60000/60000 [==============================] - 8s 130us/step - loss: 0.0515 - acc: 0.9838 - val
Epoch 37/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0524 - acc: 0.9839 - val
Epoch 38/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.0469 - acc: 0.9850 - val
Epoch 39/50
60000/60000 [==============================] - 8s 139us/step - loss: 0.0496 - acc: 0.9852 - val
Epoch 40/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0468 - acc: 0.9855 - val
Epoch 41/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0469 - acc: 0.9853 - val
Epoch 42/50
60000/60000 [==============================] - 8s 138us/step - loss: 0.0472 - acc: 0.9853 - val
Epoch 43/50
60000/60000 [==============================] - 8s 140us/step - loss: 0.0450 - acc: 0.9858 - val
Epoch 44/50
60000/60000 [==============================] - 8s 139us/step - loss: 0.0439 - acc: 0.9865 - val
Epoch 45/50
60000/60000 [==============================] - 9s 142us/step - loss: 0.0434 - acc: 0.9863 - val
Epoch 46/50
60000/60000 [==============================] - 8s 139us/step - loss: 0.0394 - acc: 0.9875 - val
Epoch 47/50
60000/60000 [==============================] - 8s 133us/step - loss: 0.0430 - acc: 0.9866 - val
Epoch 48/50
60000/60000 [==============================] - 8s 137us/step - loss: 0.0420 - acc: 0.9871 - val
Epoch 49/50
60000/60000 [==============================] - 8s 136us/step - loss: 0.0416 - acc: 0.9870 - val
Epoch 50/50
60000/60000 [==============================] - 8s 141us/step - loss: 0.0398 - acc: 0.9875 - val
Test score: 0.05416695562318346
Test accuracy: 0.9862
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50
epoch


Trained model Weights
Trained model Weights
Trained model Weights
Trained model Weights
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
1.6
1.4
1.2
1.0
0.8
0.6
0.4
5
4
3
2
1
0
2
1
0
−1
−2
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Output Layer

3 Try 5 hidden layer

In [56]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
#smuk
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_relu.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

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

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve



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

import pandas as pd
scoretrain = model_relu.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['6 Layer '],'Test Score':[score[0]],'Test Accuracy':[score[1]
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
print('Score',bb)
aa=aa.append(bb)

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)


print(aa)
w_after = model_relu.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(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')


plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()


from keras.layers.normalization import BatchNormalization
# h1 =>  =(2/(ni+ni+1) = 0.039  => N(0,) = N(0,0.039)
# h2 =>  =(2/(ni+ni+1) = 0.055  => N(0,) = N(0,0.055)
# h1 =>  =(2/(ni+ni+1) = 0.120  => N(0,) = N(0,0.120)

model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
```

```
model_batch.add(BatchNormalization())

#smuk
model_batch.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.
model_batch.add(BatchNormalization())
#smuk
model_batch.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.
model_batch.add(BatchNormalization())

model_batch.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_batch.add(BatchNormalization())

model_batch.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_batch.add(BatchNormalization())

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


model_batch.summary()

model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, v


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


scoretrain = model_batch.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['6 Layer batch norm'],'Test Score':[score[0]],'Test Accuracy
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
```

```
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of

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


w_after = model_batch.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 batch norm")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')
```

```
plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()



#

# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))


model_drop.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
#smuk
model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

model_drop.summary()

model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve


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

```
print('Test accuracy:', score[1])
scoretrain = model_drop.evaluate(X_train, Y_train, verbose=0)
bb=pd.DataFrame({'type':['6 Layer batch norm dropout'],'Test Score':[score[0]],'Test
                   'Train Score':[scoretrain[0]],'Train Accuracy':[scoretrain[1]]})
aa=aa.append(bb)
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of

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


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(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

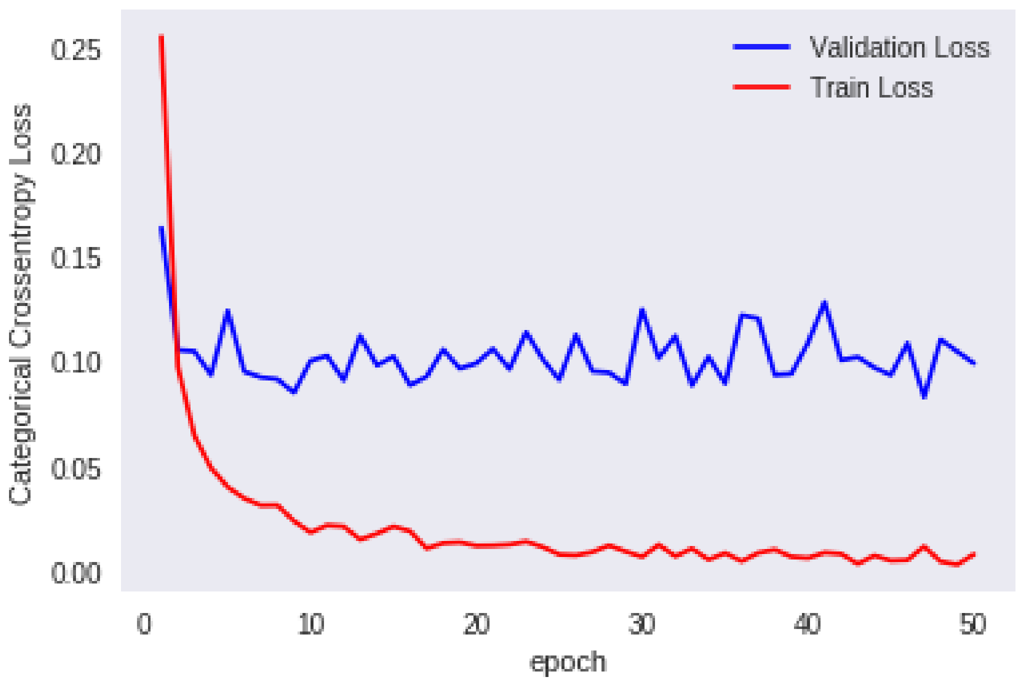
```
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')


plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

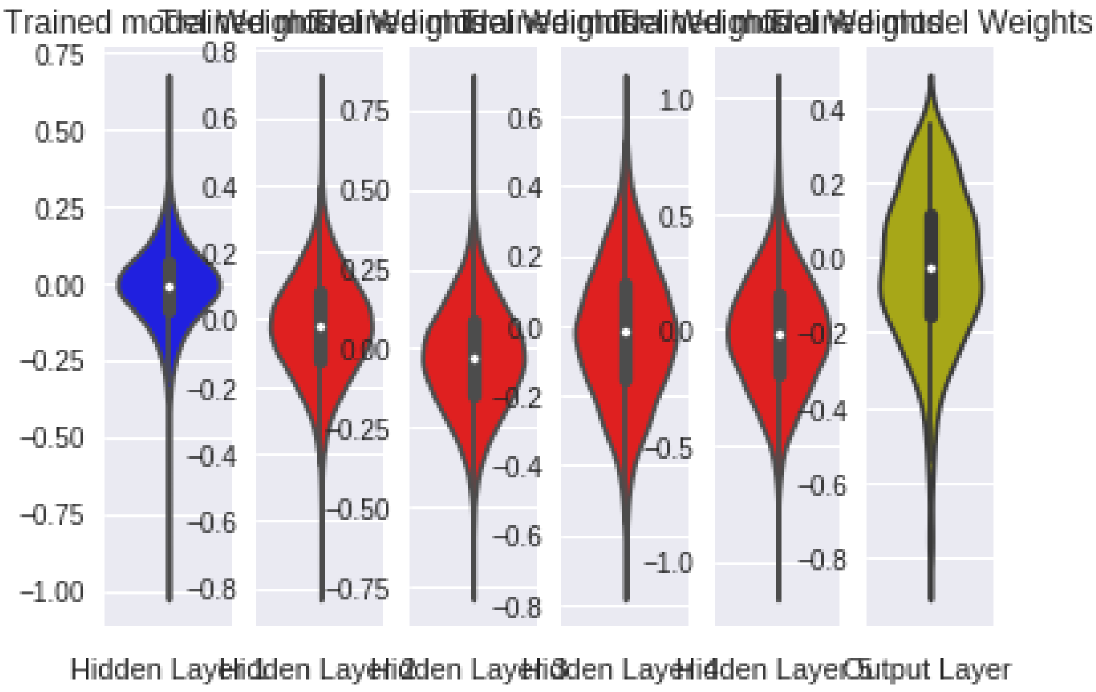
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_36 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_37 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_38 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_39 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_40 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 6s 102us/step - loss: 0.2553 - acc: 0.9222 - va
Epoch 2/50
```

```
60000/60000 [==============================] - 5s 78us/step - loss: 0.0973 - acc: 0.9706 - val_
Epoch 3/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0642 - acc: 0.9806 - val_
Epoch 4/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0487 - acc: 0.9846 - val_
Epoch 5/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0396 - acc: 0.9868 - val_
Epoch 6/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0342 - acc: 0.9892 - val_
Epoch 7/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0308 - acc: 0.9902 - val_
Epoch 8/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0309 - acc: 0.9902 - val_
Epoch 9/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0233 - acc: 0.9923 - val_
Epoch 10/50
60000/60000 [==============================] - 5s 81us/step - loss: 0.0178 - acc: 0.9945 - val_
Epoch 11/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0212 - acc: 0.9932 - val_
Epoch 12/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0207 - acc: 0.9934 - val_
Epoch 13/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0144 - acc: 0.9956 - val_
Epoch 14/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0173 - acc: 0.9946 - val_
Epoch 15/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0207 - acc: 0.9936 - val_
Epoch 16/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0184 - acc: 0.9942 - val_
Epoch 17/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0102 - acc: 0.9969 - val_
Epoch 18/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0127 - acc: 0.9960 - val_
Epoch 19/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0131 - acc: 0.9963 - val_
Epoch 20/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0114 - acc: 0.9966 - val_
Epoch 21/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0115 - acc: 0.9962 - val_
Epoch 22/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0120 - acc: 0.9964 - val_
Epoch 23/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0134 - acc: 0.9961 - val_
Epoch 24/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0108 - acc: 0.9970 - val_
Epoch 25/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0072 - acc: 0.9978 - val_
Epoch 26/50
```

```
60000/60000 [==============================] - 5s 79us/step - loss: 0.0069 - acc: 0.9983 - val_
Epoch 27/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0084 - acc: 0.9975 - val_
Epoch 28/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0116 - acc: 0.9967 - val_
Epoch 29/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0087 - acc: 0.9976 - val_
Epoch 30/50
60000/60000 [==============================] - 5s 75us/step - loss: 0.0061 - acc: 0.9981 - val_
Epoch 31/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0119 - acc: 0.9968 - val_
Epoch 32/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0064 - acc: 0.9981 - val_
Epoch 33/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0101 - acc: 0.9972 - val_
Epoch 34/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0047 - acc: 0.9986 - val_
Epoch 35/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0079 - acc: 0.9977 - val_
Epoch 36/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0040 - acc: 0.9987 - val_
Epoch 37/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0080 - acc: 0.9975 - val_
Epoch 38/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0096 - acc: 0.9972 - val_
Epoch 39/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0061 - acc: 0.9985 - val_
Epoch 40/50
60000/60000 [==============================] - 5s 78us/step - loss: 0.0055 - acc: 0.9985 - val_
Epoch 41/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0082 - acc: 0.9979 - val_
Epoch 42/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0075 - acc: 0.9981 - val_
Epoch 43/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0028 - acc: 0.9993 - val_
Epoch 44/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0067 - acc: 0.9982 - val_
Epoch 45/50
60000/60000 [==============================] - 5s 80us/step - loss: 0.0043 - acc: 0.9989 - val_
Epoch 46/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0046 - acc: 0.9989 - val_
Epoch 47/50
60000/60000 [==============================] - 5s 77us/step - loss: 0.0111 - acc: 0.9969 - val_
Epoch 48/50
60000/60000 [==============================] - 5s 79us/step - loss: 0.0039 - acc: 0.9989 - val_
Epoch 49/50
60000/60000 [==============================] - 5s 76us/step - loss: 0.0024 - acc: 0.9992 - val_
Epoch 50/50
```

```
60000/60000 [==============================] - 5s 76us/step - loss: 0.0073 - acc: 0.9980 - val
Test score: 0.09885513828289336
Test accuracy: 0.9835
Score    Test Accuracy  Test Score  Train Accuracy  Train Score      type
0         0.9835    0.098855        0.998933     0.003696  6 Layer
   Test Accuracy  Test Score  Train Accuracy  Train Score  \
0         0.9826    0.107210        0.999683     0.000822
0         0.9826    0.085318        0.999283     0.002036
0         0.9851    0.059443        0.999300     0.002966
0         0.9828    0.116272        0.999300     0.002566
0         0.9857    0.075461        0.999583     0.001542
0         0.9862    0.054167        0.998650     0.004089
0         0.9835    0.098855        0.998933     0.003696

                         type
0                     3 Layer
0           3 Layer batch norm
0   3 Layer batch norm dropout
0                     4 Layer
0           4 Layer batch norm
0   4 Layer batch norm dropout
0                     6 Layer


/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



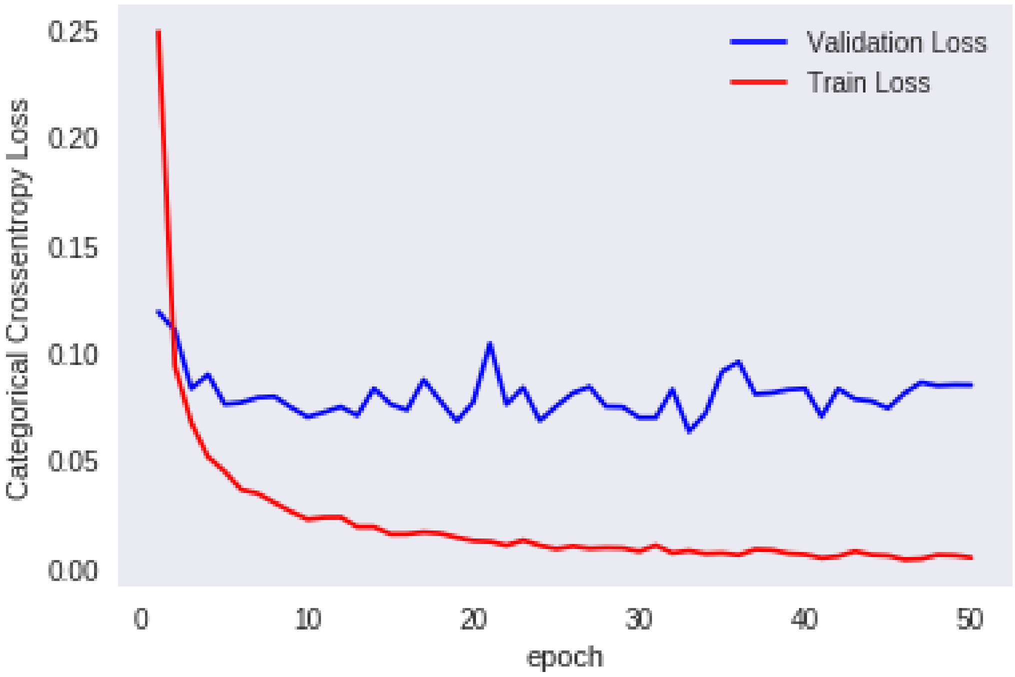


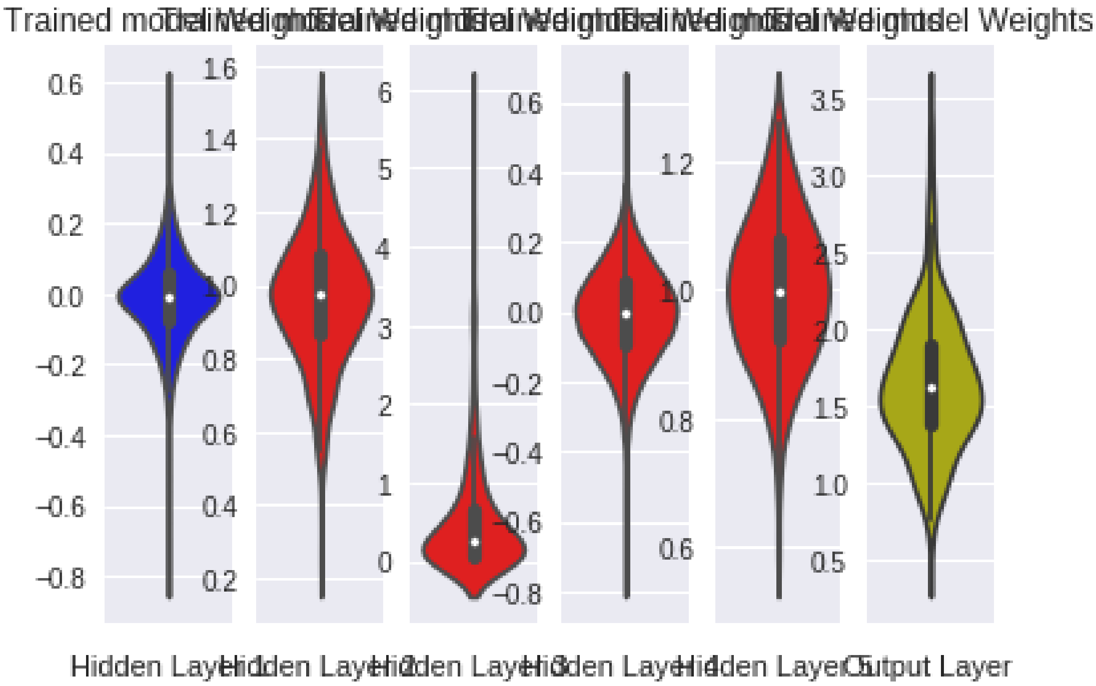
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_41 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_15 (Batc (None, 512)               2048
_________________________________________________________________
dense_42 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_16 (Batc (None, 256)               1024
_________________________________________________________________
dense_43 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_17 (Batc (None, 128)               512
_________________________________________________________________
dense_44 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_18 (Batc (None, 64)                256
_________________________________________________________________
dense_45 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_19 (Batc (None, 32)                128
_________________________________________________________________
dense_46 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 13s 218us/step - loss: 0.2488 - acc: 0.9314 - va
Epoch 2/50
60000/60000 [==============================] - 11s 178us/step - loss: 0.0936 - acc: 0.9719 - va
Epoch 3/50
60000/60000 [==============================] - 11s 176us/step - loss: 0.0670 - acc: 0.9792 - va
Epoch 4/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0513 - acc: 0.9838 - va
Epoch 5/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0445 - acc: 0.9864 - va
Epoch 6/50
60000/60000 [==============================] - 11s 184us/step - loss: 0.0361 - acc: 0.9884 - va
Epoch 7/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0343 - acc: 0.9890 - va
Epoch 8/50
60000/60000 [==============================] - 11s 178us/step - loss: 0.0300 - acc: 0.9908 - va
Epoch 9/50
60000/60000 [==============================] - 11s 183us/step - loss: 0.0259 - acc: 0.9920 - va
Epoch 10/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0223 - acc: 0.9931 - va
```

```
Epoch 11/50
60000/60000 [==============================] - 11s 184us/step - loss: 0.0232 - acc: 0.9926 - va
Epoch 12/50
60000/60000 [==============================] - 11s 182us/step - loss: 0.0233 - acc: 0.9925 - va
Epoch 13/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0188 - acc: 0.9939 - va
Epoch 14/50
60000/60000 [==============================] - 11s 178us/step - loss: 0.0188 - acc: 0.9937 - va
Epoch 15/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0154 - acc: 0.9947 - va
Epoch 16/50
60000/60000 [==============================] - 11s 185us/step - loss: 0.0154 - acc: 0.9951 - va
Epoch 17/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0163 - acc: 0.9950 - va
Epoch 18/50
60000/60000 [==============================] - 11s 182us/step - loss: 0.0158 - acc: 0.9947 - va
Epoch 19/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0138 - acc: 0.9955 - va
Epoch 20/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0123 - acc: 0.9959 - va
Epoch 21/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0120 - acc: 0.9961 - va
Epoch 22/50
60000/60000 [==============================] - 11s 183us/step - loss: 0.0102 - acc: 0.9966 - va
Epoch 23/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0126 - acc: 0.9958 - va
Epoch 24/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0101 - acc: 0.9966 - va
Epoch 25/50
60000/60000 [==============================] - 11s 184us/step - loss: 0.0085 - acc: 0.9973 - va
Epoch 26/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0098 - acc: 0.9966 - va
Epoch 27/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0087 - acc: 0.9973 - va
Epoch 28/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0091 - acc: 0.9971 - va
Epoch 29/50
60000/60000 [==============================] - 11s 178us/step - loss: 0.0089 - acc: 0.9972 - va
Epoch 30/50
60000/60000 [==============================] - 11s 177us/step - loss: 0.0074 - acc: 0.9976 - va
Epoch 31/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0102 - acc: 0.9967 - va
Epoch 32/50
60000/60000 [==============================] - 11s 182us/step - loss: 0.0067 - acc: 0.9975 - va
Epoch 33/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0077 - acc: 0.9976 - va
Epoch 34/50
60000/60000 [==============================] - 11s 183us/step - loss: 0.0063 - acc: 0.9981 - va
```

```
Epoch 35/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0066 - acc: 0.9977 - va
Epoch 36/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0057 - acc: 0.9982 - va
Epoch 37/50
60000/60000 [==============================] - 11s 184us/step - loss: 0.0083 - acc: 0.9972 - va
Epoch 38/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0080 - acc: 0.9975 - va
Epoch 39/50
60000/60000 [==============================] - 11s 185us/step - loss: 0.0064 - acc: 0.9979 - va
Epoch 40/50
60000/60000 [==============================] - 11s 185us/step - loss: 0.0059 - acc: 0.9980 - va
Epoch 41/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0043 - acc: 0.9987 - va
Epoch 42/50
60000/60000 [==============================] - 11s 182us/step - loss: 0.0051 - acc: 0.9983 - va
Epoch 43/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0074 - acc: 0.9977 - va
Epoch 44/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0058 - acc: 0.9983 - va
Epoch 45/50
60000/60000 [==============================] - 11s 180us/step - loss: 0.0054 - acc: 0.9983 - va
Epoch 46/50
60000/60000 [==============================] - 11s 177us/step - loss: 0.0035 - acc: 0.9991 - va
Epoch 47/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0039 - acc: 0.9987 - va
Epoch 48/50
60000/60000 [==============================] - 11s 179us/step - loss: 0.0058 - acc: 0.9981 - va
Epoch 49/50
60000/60000 [==============================] - 11s 182us/step - loss: 0.0055 - acc: 0.9983 - va
Epoch 50/50
60000/60000 [==============================] - 11s 181us/step - loss: 0.0045 - acc: 0.9986 - va
Test score: 0.0846944084940711
Test accuracy: 0.9824
```



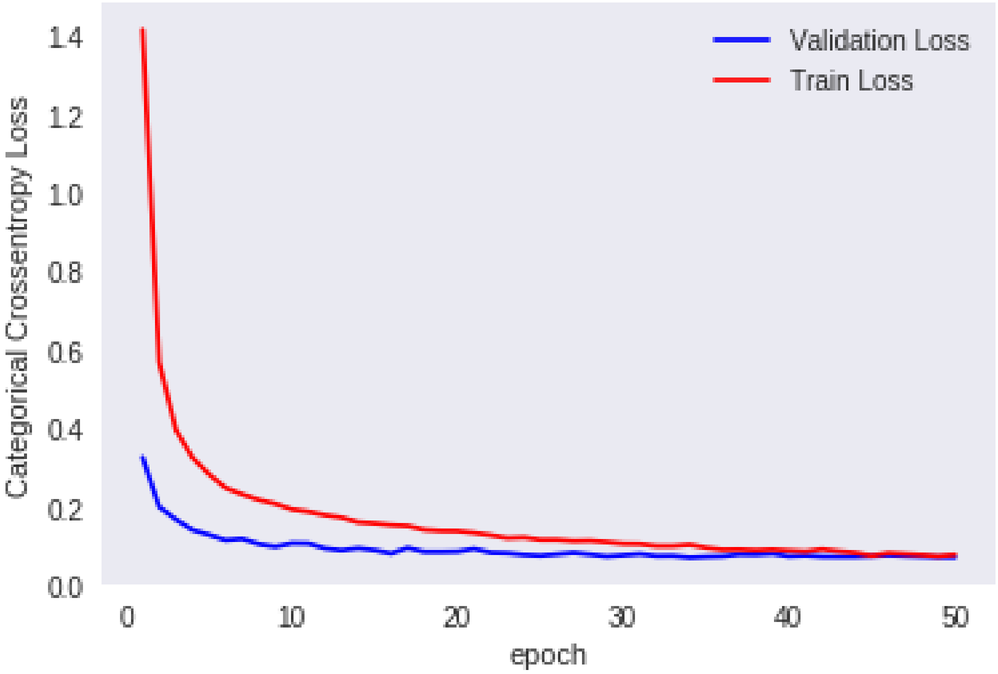


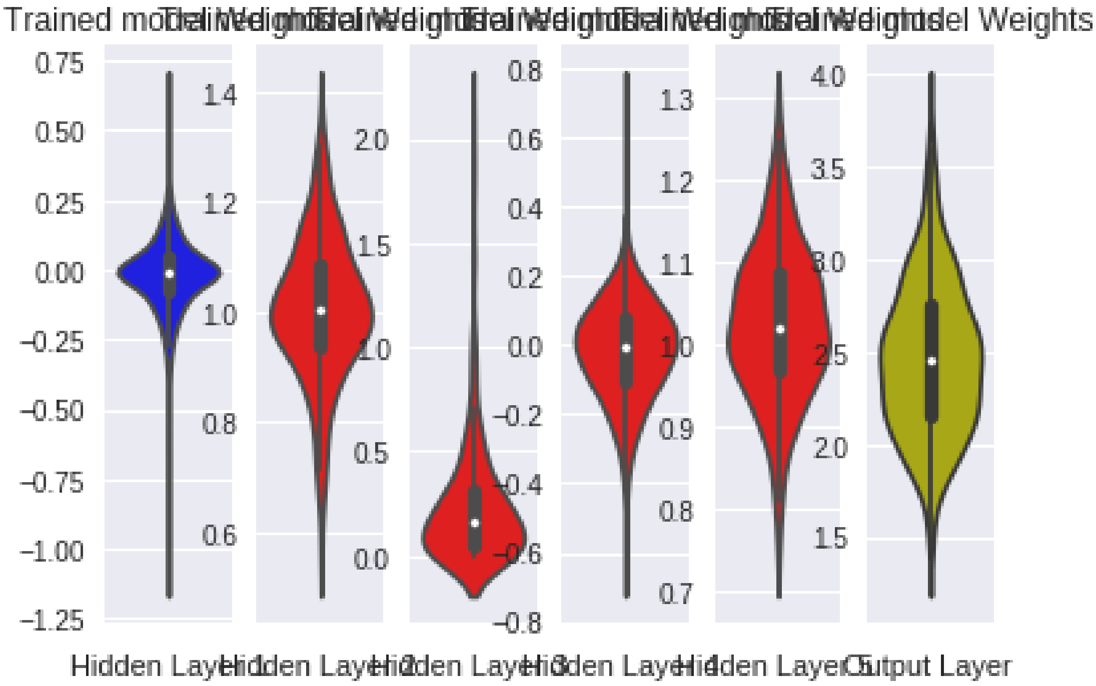
```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_47 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_20 (Batc (None, 512)               2048
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_48 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_21 (Batc (None, 256)               1024
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_49 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_22 (Batc (None, 128)               512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_50 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_23 (Batc (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_51 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_24 (Batc (None, 32)                128
_________________________________________________________________
dropout_12 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 15s 254us/step - loss: 1.4138 - acc: 0.5471 - va
Epoch 2/50
60000/60000 [==============================] - 12s 195us/step - loss: 0.5675 - acc: 0.8446 - va
Epoch 3/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.3949 - acc: 0.9011 - va
Epoch 4/50
60000/60000 [==============================] - 12s 197us/step - loss: 0.3242 - acc: 0.9220 - va
Epoch 5/50
60000/60000 [==============================] - 11s 190us/step - loss: 0.2824 - acc: 0.9341 - va
```

```
Epoch 6/50
60000/60000 [==============================] - 11s 192us/step - loss: 0.2476 - acc: 0.9429 - va
Epoch 7/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.2313 - acc: 0.9481 - va
Epoch 8/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.2173 - acc: 0.9505 - va
Epoch 9/50
60000/60000 [==============================] - 12s 195us/step - loss: 0.2072 - acc: 0.9531 - va
Epoch 10/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1930 - acc: 0.9559 - va
Epoch 11/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.1869 - acc: 0.9577 - va
Epoch 12/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.1782 - acc: 0.9594 - va
Epoch 13/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.1721 - acc: 0.9621 - va
Epoch 14/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.1601 - acc: 0.9645 - va
Epoch 15/50
60000/60000 [==============================] - 12s 196us/step - loss: 0.1571 - acc: 0.9640 - va
Epoch 16/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.1528 - acc: 0.9653 - va
Epoch 17/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1506 - acc: 0.9662 - va
Epoch 18/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.1408 - acc: 0.9687 - va
Epoch 19/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1381 - acc: 0.9689 - va
Epoch 20/50
60000/60000 [==============================] - 12s 200us/step - loss: 0.1376 - acc: 0.9686 - va
Epoch 21/50
60000/60000 [==============================] - 12s 194us/step - loss: 0.1330 - acc: 0.9702 - va
Epoch 22/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.1270 - acc: 0.9718 - va
Epoch 23/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.1198 - acc: 0.9723 - va
Epoch 24/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1214 - acc: 0.9727 - va
Epoch 25/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.1153 - acc: 0.9740 - va
Epoch 26/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1153 - acc: 0.9738 - va
Epoch 27/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.1122 - acc: 0.9750 - va
Epoch 28/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1129 - acc: 0.9753 - va
Epoch 29/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.1096 - acc: 0.9746 - va
```

```
Epoch 30/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1054 - acc: 0.9754 - va
Epoch 31/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.1049 - acc: 0.9767 - va
Epoch 32/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.0995 - acc: 0.9782 - va
Epoch 33/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.0995 - acc: 0.9777 - va
Epoch 34/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.1031 - acc: 0.9771 - va
Epoch 35/50
60000/60000 [==============================] - 11s 190us/step - loss: 0.0951 - acc: 0.9788 - va
Epoch 36/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.0900 - acc: 0.9797 - va
Epoch 37/50
60000/60000 [==============================] - 11s 188us/step - loss: 0.0907 - acc: 0.9792 - va
Epoch 38/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.0887 - acc: 0.9802 - va
Epoch 39/50
60000/60000 [==============================] - 11s 187us/step - loss: 0.0905 - acc: 0.9794 - va
Epoch 40/50
60000/60000 [==============================] - 12s 194us/step - loss: 0.0878 - acc: 0.9810 - va
Epoch 41/50
60000/60000 [==============================] - 12s 193us/step - loss: 0.0854 - acc: 0.9808 - va
Epoch 42/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.0915 - acc: 0.9796 - va
Epoch 43/50
60000/60000 [==============================] - 11s 190us/step - loss: 0.0861 - acc: 0.9810 - va
Epoch 44/50
60000/60000 [==============================] - 11s 187us/step - loss: 0.0822 - acc: 0.9817 - va
Epoch 45/50
60000/60000 [==============================] - 11s 191us/step - loss: 0.0745 - acc: 0.9830 - va
Epoch 46/50
60000/60000 [==============================] - 12s 192us/step - loss: 0.0817 - acc: 0.9810 - va
Epoch 47/50
60000/60000 [==============================] - 12s 195us/step - loss: 0.0791 - acc: 0.9823 - va
Epoch 48/50
60000/60000 [==============================] - 11s 190us/step - loss: 0.0767 - acc: 0.9831 - va
Epoch 49/50
60000/60000 [==============================] - 11s 186us/step - loss: 0.0734 - acc: 0.9833 - va
Epoch 50/50
60000/60000 [==============================] - 11s 189us/step - loss: 0.0775 - acc: 0.9831 - va
Test score: 0.07074182197086484
Test accuracy: 0.9858
```


Validation Loss
Train Loss
Categorical Crossentropy Loss
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
epoch


Trained model Weights
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
1.4
1.2
1.0
0.8
0.6
2.0
1.5
1.0
0.5
0.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1.3
1.2
1.1
1.0
0.9
0.8
0.7
4.0
3.5
3.0
2.5
2.0
1.5
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Hidden Layer 4
Hidden Layer 5
Output Layer

```
In [57]: aa
```

```
Out[57]:    Test Accuracy  Test Score  Train Accuracy  Train Score  \
         0         0.9826    0.107210        0.999683     0.000822
         0         0.9826    0.085318        0.999283     0.002036
         0         0.9851    0.059443        0.999300     0.002966
         0         0.9828    0.116272        0.999300     0.002566
         0         0.9857    0.075461        0.999583     0.001542
         0         0.9862    0.054167        0.998650     0.004089
         0         0.9835    0.098855        0.998933     0.003696
         0         0.9824    0.084694        0.998617     0.004070
         0         0.9858    0.070742        0.997333     0.009903

                                type
         0                   3 Layer
         0           3 Layer batch norm
         0   3 Layer batch norm dropout
         0                   4 Layer
         0           4 Layer batch norm
         0   4 Layer batch norm dropout
         0                   6 Layer
         0           6 Layer batch norm
         0   6 Layer batch norm dropout
```