Keras -- MLPs on MNIST (10 models trained)

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this commanc
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

```
%matplotlib notebook
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()
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

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

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

```
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)
```

```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 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])
```

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

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

```
# 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   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]
```

```
# 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
```

```
# 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.
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
 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.         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.        ]
```

```
# 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


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

# 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_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=
# 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 activation argument

# 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
```

```
# some model parameters
output_dim = 10
input_dim = X_train.shape[1]
batch_size = 128
nb_epoch = 20
```

* 2 Hidden layers MLP + ReLU + ADAM without batch normalization and d

```
model1_1 = Sequential()
model1_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
model1_1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
history = model1_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valid
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_29 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_30 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2223 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0851 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0511 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0368 - acc: 0.9
Epoch 5/20
.9
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
.9
Epoch 7/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0168 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0153 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0152 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0115 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0105 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0104 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0108 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0102 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0056 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0085 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0098 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0054 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0090 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0081 - acc: 0.9
```

```
score = model1_1.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# 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, verbose=1, v
# 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 epochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.09645916890637461
Test accuracy: 0.9819
```

```
score = model1_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

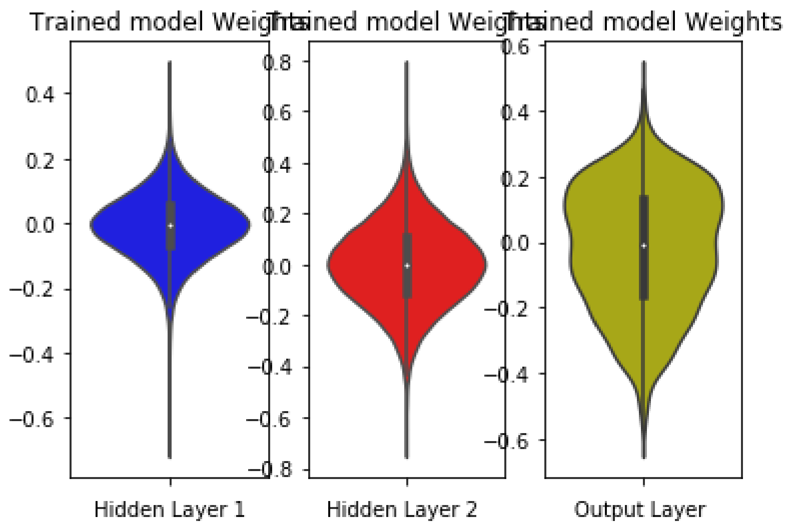
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
# 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, verbose=1, v

# 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 epochs

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

```
Test score: 0.09645916890637461
Test accuracy: 0.9819
```

```
%matplotlib inline
w_after = model1_1.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()
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
model1 = Sequential()
model1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorma
model1.add(BatchNormalization())
model1.add(Dropout(0.5))
model1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125,
model1.add(BatchNormalization())
model1.add(Dropout(0.5))
model1.add(Dense(output_dim, activation='softmax'))
print(model1.summary())
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_31 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_32 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_15 (Batc (None, 128)               512
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_33 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

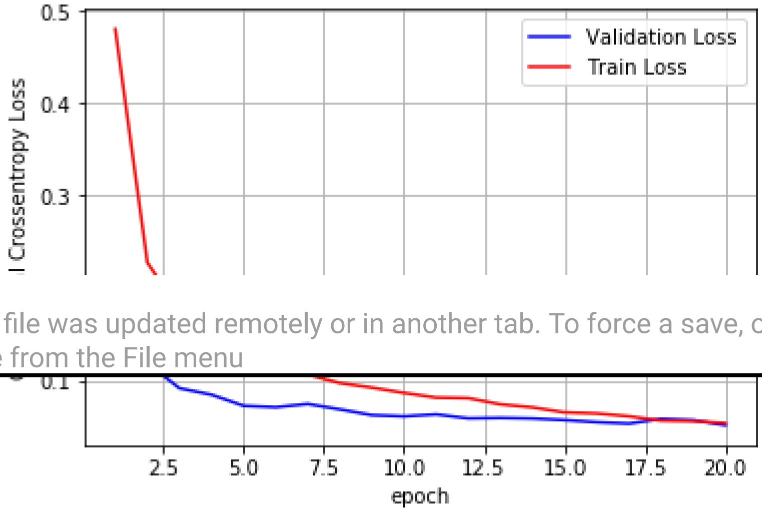


```
Epoch 3/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1773 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1522 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1322 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1156 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1066 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0977 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0927 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0870 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0819 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0813 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0746 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0713 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0659 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0647 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0616 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0570 - acc: 0.9
Epoch 19/20
```

```
60000/60000 [==============================] - 5s 91us/step - loss: 0.0564 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0543 - acc: 0.9
```

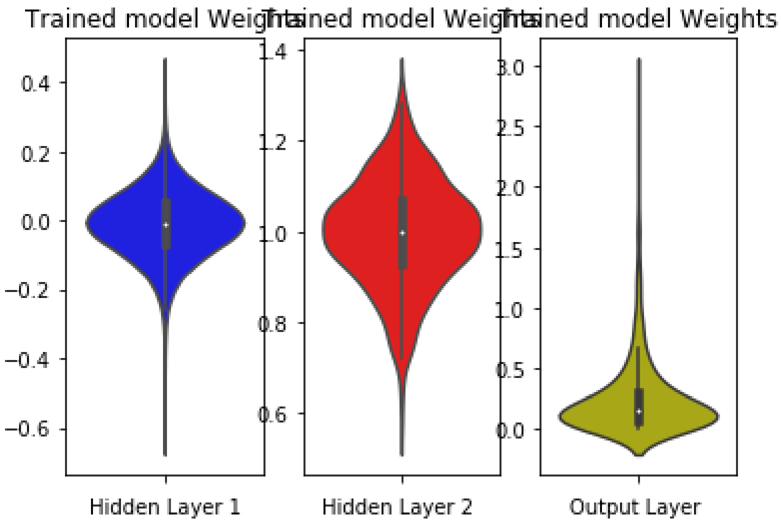
```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.052021935435730846
Test accuracy: 0.9836
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
%matplotlib inline
w_after = model1.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()
```



* 2 Hidden layers MLP + SIGMOID + SGD + Batch Normalization + Dropout

```
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
model1 = Sequential()
model1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNc
model1.add(BatchNormalization())
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
12
model1.add(Dropout(0.4))
model1.add(Dense(output_dim, activation='softmax'))
print(model1.summary())
model1.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 149us/step - loss: 1.0443 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.5988 - acc: 0.8
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.5188 - acc: 0.8
Epoch 4/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.4803 - acc: 0.8
Epoch 5/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.4539 - acc: 0.8
Epoch 6/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.4333 - acc: 0.8
Epoch 7/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.4198 - acc: 0.8
Epoch 8/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.4062 - acc: 0.8
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3955 - acc: 0.8
Epoch 10/20
```

```
60000/60000 [==============================] - 4s 73us/step - loss: 0.3863 - acc: 0.8
Epoch 11/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3762 - acc: 0.8
Epoch 12/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3687 - acc: 0.8
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.3649 - acc: 0.8
Epoch 14/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3617 - acc: 0.8
Epoch 15/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3541 - acc: 0.8
Epoch 16/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3461 - acc: 0.8
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3451 - acc: 0.8
Epoch 18/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3389 - acc: 0.8
Epoch 19/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.3329 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.3312 - acc: 0.9
```

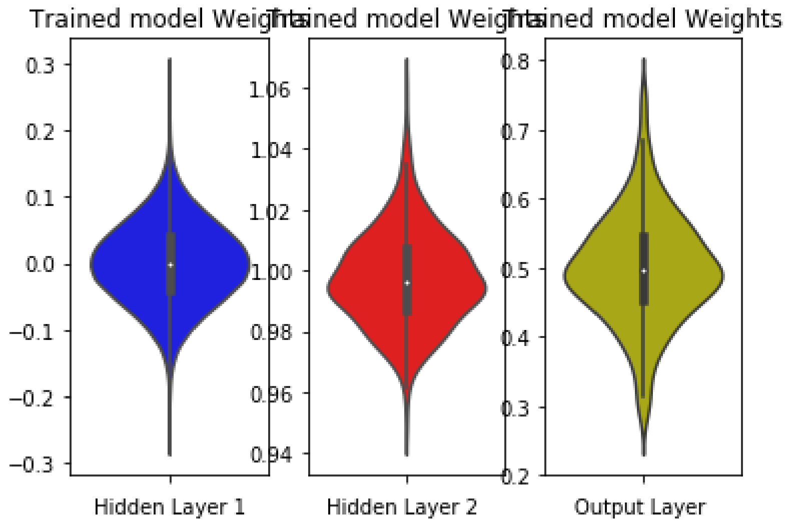
```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.23021167929247022
Test accuracy: 0.9338
```

```
%matplotlib inline
w_after = model1.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()
```



* 2 Hidden layers MLP + tanh + SGD + Batch Normalization + Dropout

```
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
model1 = Sequential()
model1.add(Dense(512, activation='tanh', input_shape=(input_dim,), kernel_initializer=RandomNorma
model1.add(BatchNormalization())
model1.add(Dropout(0.4))
                                                                                          ,
model1.add(Dense(output_dim, activation='softmax'))
print(model1.summary())
model1.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
                                                                    .6
```

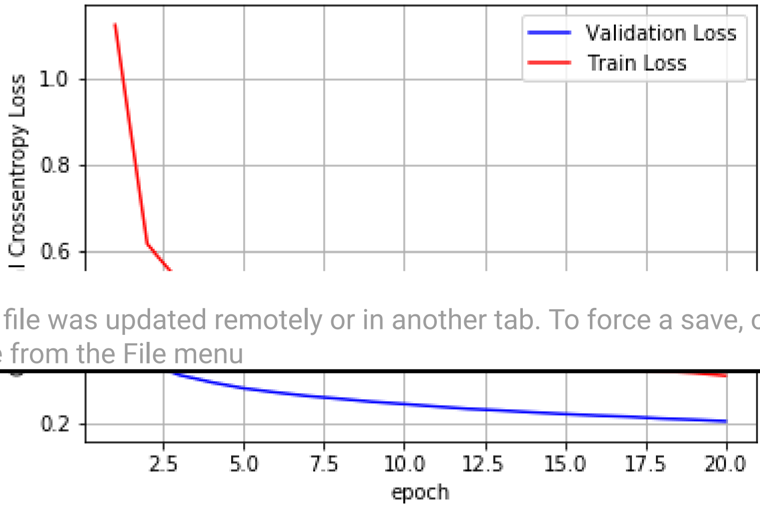
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
                                                                    .8
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.5301 - acc: 0.8
Epoch 4/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.4834 - acc: 0.8
Epoch 5/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.4525 - acc: 0.8
Epoch 6/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.4300 - acc: 0.8
Epoch 7/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.4120 - acc: 0.8
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3959 - acc: 0.8
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3889 - acc: 0.8
Epoch 10/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3743 - acc: 0.8
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3671 - acc: 0.8
Epoch 12/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3602 - acc: 0.8
Epoch 13/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3506 - acc: 0.8
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3443 - acc: 0.8
Epoch 15/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3377 - acc: 0.8
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3312 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3258 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3224 - acc: 0.9
Epoch 19/20
```

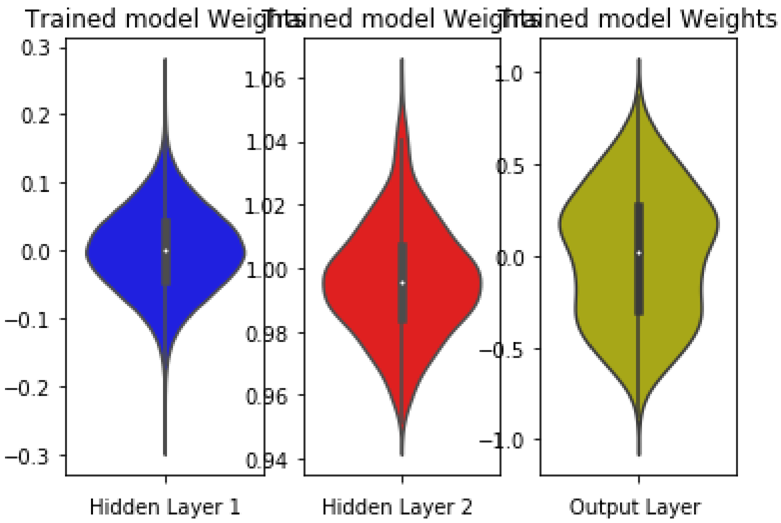
```
60000/60000 [==============================] - 4s 73us/step - loss: 0.3180 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3115 - acc: 0.9
```

```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.20571986180469393
Test accuracy: 0.9393
```




```
%matplotlib inline
w_after = model1.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()
```




```
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
model1 = Sequential()
model1.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNc
model1.add(BatchNormalization())
model1.add(Dropout(0.4))
model1.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0.12
model1.add(BatchNormalization())
model1.add(Dropout(0.4))
model1.add(Dense(output_dim, activation='softmax'))
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

at

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               65664
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
.8
```

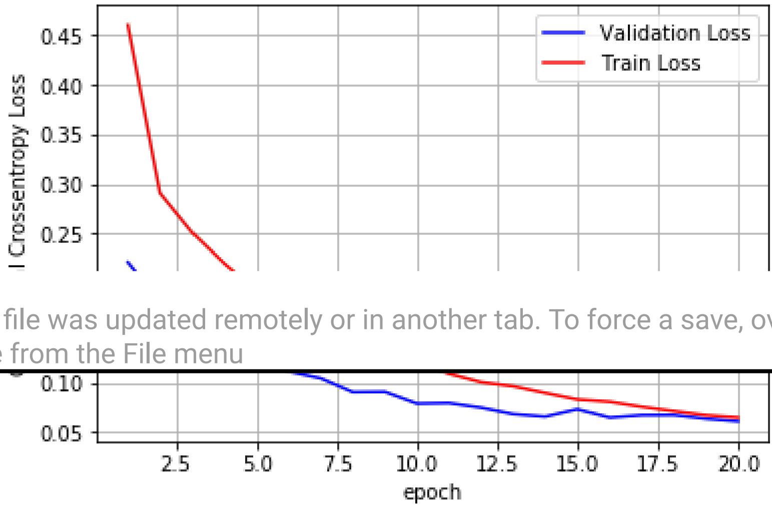
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
.9
Epoch 3/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2510 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2198 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1926 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1729 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1572 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1438 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1276 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1197 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1088 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1003 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0962 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0894 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0829 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0805 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0754 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0711 - acc: 0.9
Epoch 19/20
```

```
60000/60000 [==============================] - 5s 82us/step - loss: 0.0669 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0646 - acc: 0.9
```

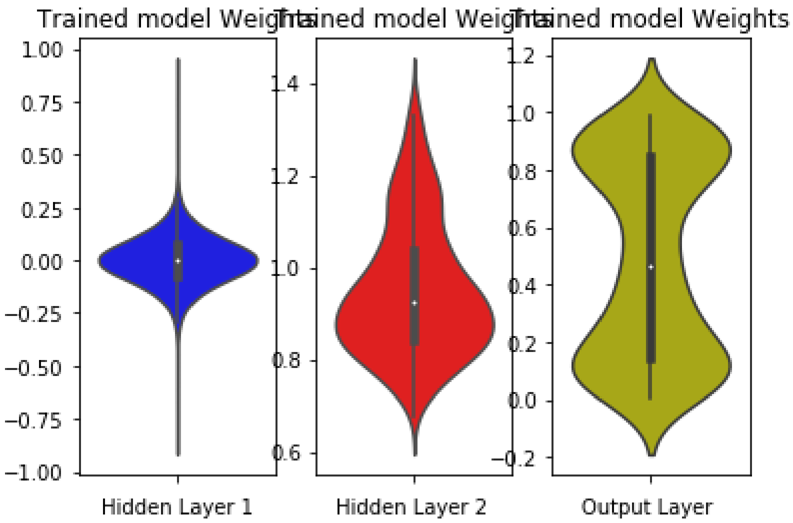
```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06078589027107228
Test accuracy: 0.9818
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
%matplotlib inline
w_after = model1.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()
```



2 layer + ADAM + DROPOUT + BATCH NORMALIZATION + TANH

```
from keras.layers import Dense, Dropout, Flatten, Activation, BatchNormalization
model1 = Sequential()
model1.add(Dense(512, activation='tanh', input_shape=(input_dim,), kernel_initializer=RandomNorma
model1.add(BatchNormalization())
model1.add(Dropout(0.4))
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
model1.add(Dense(output_dim, activation='softmax'))
print(model1.summary())
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validat
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512       
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

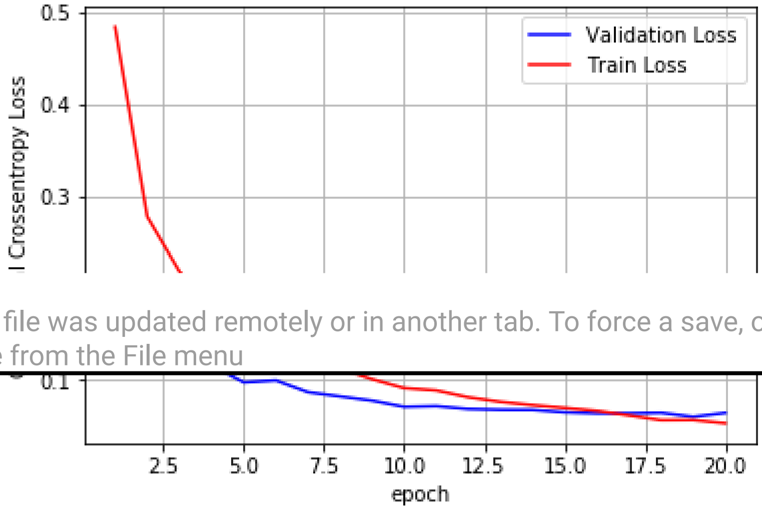
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
Epoch 3/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2183 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1825 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1561 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1358 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1208 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1121 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1008 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0911 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0885 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0811 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0761 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0725 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0695 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0663 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0613 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0561 - acc: 0.9
Epoch 19/20
```

```
60000/60000 [==============================] - 5s 84us/step - loss: 0.0563 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0527 - acc: 0.9
```

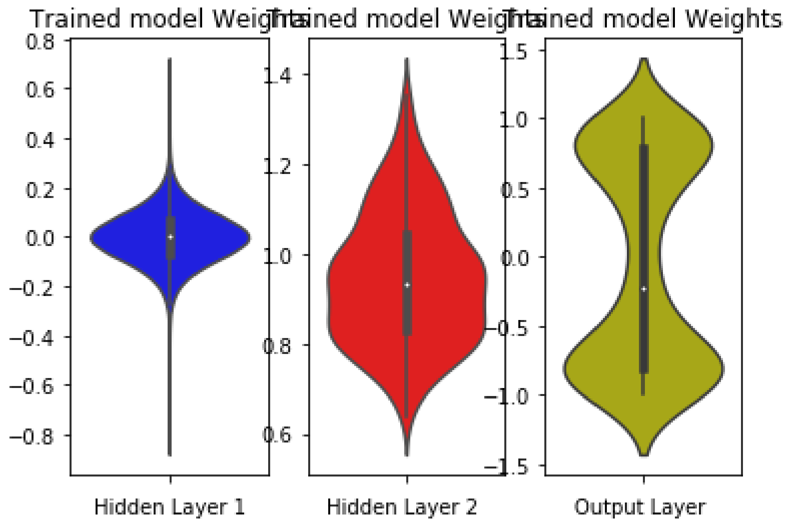
```
score = model1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06402380225082l6
Test accuracy: 0.9822
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
%matplotlib inline
w_after = model1.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)
fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')
plt.subplot(1,3,2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')
plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



3 hidden layer MLP + Batch-Norm on hidden Layers + AdamOptimizer but

```
# Multilayer perceptron

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

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
from keras.layers.normalization import BatchNormalization
model_2 = Sequential()
model_2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorm
model_2.add(BatchNormalization())
model_2.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55,
model_2.add(BatchNormalization())
model_2.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55,
model_2.add(BatchNormalization())
model_2.add(Dense(output_dim, activation='softmax'))
model_2.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_34 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_16 (Batc (None, 512)               2048      
_________________________________________________________________
dense_35 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_17 (Batc (None, 256)               1024      
_________________________________________________________________
dense_36 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_18 (Batc (None, 128)               512       
_________________________________________________________________
dense_37 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

```
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.2075 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0741 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0487 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0341 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0285 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0205 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0180 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0148 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0159 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0154 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0132 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0115 - acc: 0.
```

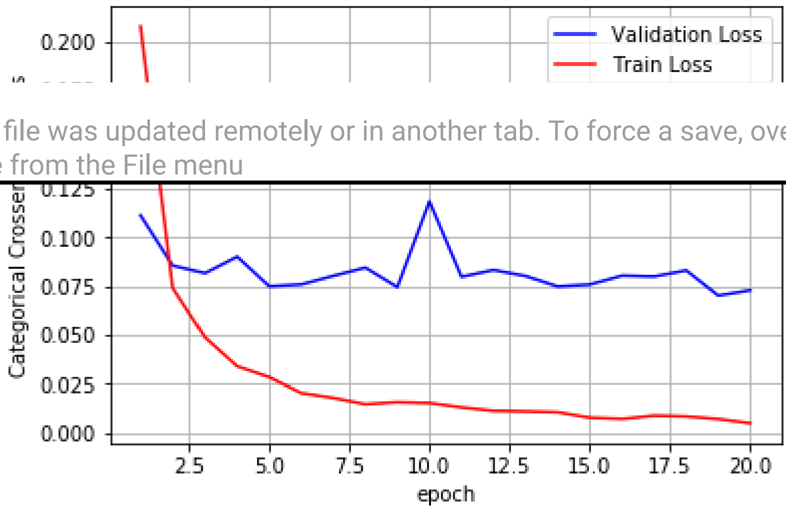
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
60000/60000 [==============================] - 7s 116us/step - loss: 0.0107 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0080 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0073 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0090 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0086 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0073 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0051 - acc: 0.
```

```
score = model_2.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, verbose=1, v

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

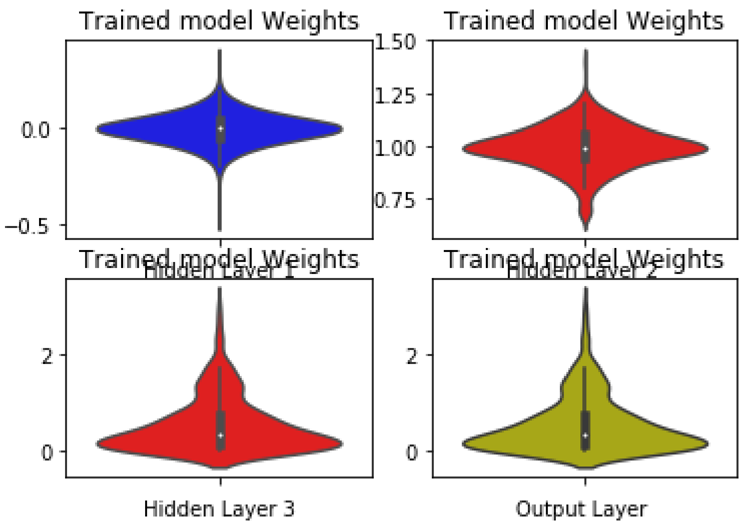
```
Test score: 0.07286385818796771
Test accuracy: 0.9821
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
w_after = model_2.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[4].flatten().reshape(-1,1)

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

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

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

plt.subplot(2, 2, 4)
```

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



3 HIDDEN LAYER AND RELU WITH DROPOUT AND BATCH NORMALIZATION

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

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
Nc
model_2_1.add(Dropout(0.5))
model_2_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNc
model_2_1.add(BatchNormalization())
model_2_1.add(Dropout(0.5))
model_2_1.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNc
model_2_1.add(BatchNormalization())
model_2_1.add(Dropout(0.5))
model_2_1.add(Dense(output_dim, activation='softmax'))
model_2_1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_38 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_19 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_39 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_20 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_40 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_21 (Batc (None, 128)               512       
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_41 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
model_2_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model_2_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.4808 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.2249 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1794 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1506 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1379 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.1205 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.1148 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.1046 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0998 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0949 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0939 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0830 - acc: 0.
```

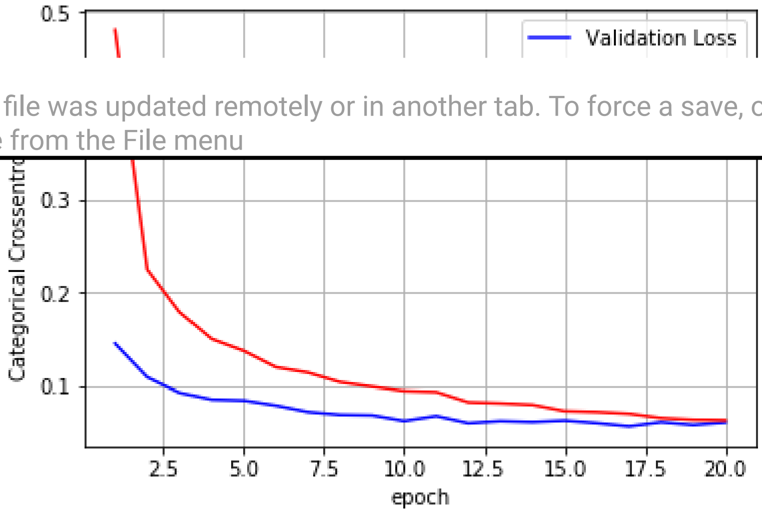
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
60000/60000 [==============================] - 7s 120us/step - loss: 0.0803 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0736 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0726 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0708 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0662 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0644 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0639 - acc: 0.
```

```
score = model_2_1.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, verbose=1, v

# 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 epochs

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

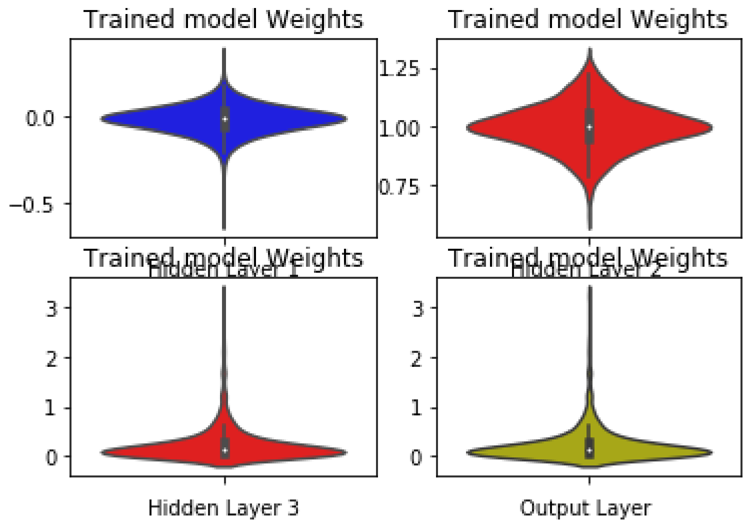
```
Test score: 0.061630533440283033
Test accuracy: 0.9821
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
w_after = model_2_1.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[4].flatten().reshape(-1,1)

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

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

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

```
plt.subplot(2, 2, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



* 5 Hidden layer MLP + ReLu + AdamOptimizer

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
model_3_1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNc
model_3_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNc
model_3_1.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNc
model_3_1.add(Dense(64, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
model_3_1.add(Dense(32, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNor
model_3_1.add(Dense(output_dim, activation='softmax'))
model_3_1.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_42 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_43 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_44 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_45 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_46 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_47 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

```
model_3_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
```

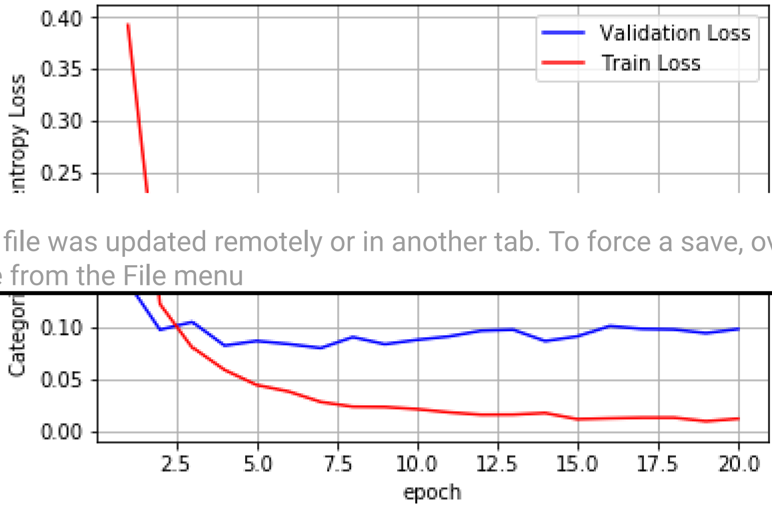
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 108us/step - loss: 0.3922 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1226 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0807 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0589 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0443 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0380 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0279 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0233 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0230 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0209 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0178 - acc: 0.9
Epoch 12/20
[illegible].9
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
[illegible].9
Epoch 14/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0172 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0112 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0121 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0127 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0127 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0092 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0117 - acc: 0.9
```

```
score = model_3_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,nb_epoch+1))
# 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, verbose=1, v
# 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 epochs
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

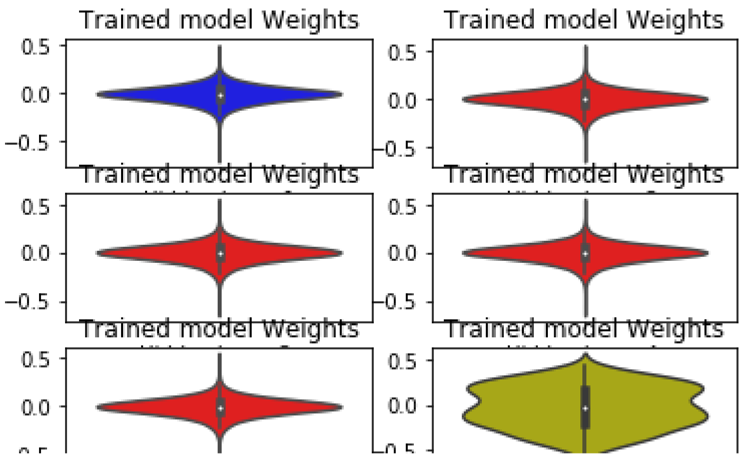
```
Test score: 0.09843842560593985
Test accuracy: 0.9782
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
w_after = model_3_1.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(3,2 , 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

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

plt.subplot(3,2, 4)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(3, 2, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 5 ')

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



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

* 5 Hidden layer MLP + Dropout + Batch Normalization + ReLu + AdamOpt

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

from keras.layers import Dropout
model_3 = Sequential()
model_3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorm
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorm
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorm
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(64, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorma
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(32, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNorma
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(output_dim, activation='softmax'))
model_3.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_48 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_22 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_23 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_50 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_24 (Batc (None, 128)               512       
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_51 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_25 (Batc (None, 64)                256       
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
dense_52 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_26 (Batc (None, 32)                128       
_________________________________________________________________
dropout_20 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_53 (Dense)             (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
```

```
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 225us/step - loss: 1.1314 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.4195 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.3209 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.2763 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.2495 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.2268 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.2138 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1929 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1868 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1752 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1716 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.1643 - acc: 0
```

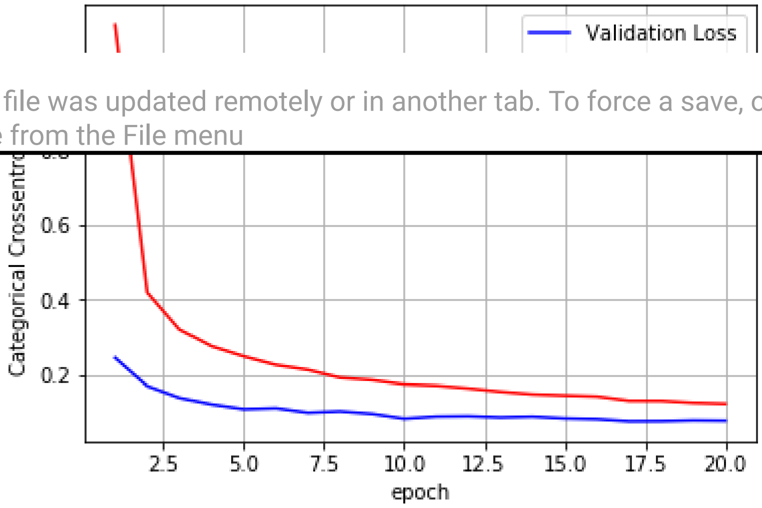
This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
0
60000/60000 [==============================] - 9s 157us/step - loss: 0.1486 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 10s 158us/step - loss: 0.1457 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1430 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1317 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1314 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1263 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1238 - acc: 0
```

```
score = model_3.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, verbose=1, v

# 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 epochs

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

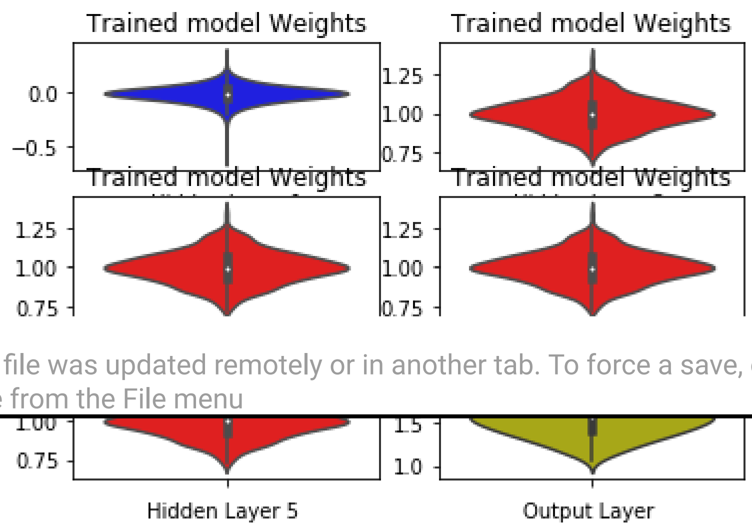
```
Test score: 0.07888613095330074
Test accuracy: 0.9813
```



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
w_after = model_3.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(3,2 , 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(3, 2, 3)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(3,2, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(3, 2, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 5 ')

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



This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Hiddden layers", "batch normalization","activation function","optimizer" ,"drop

x.add_row([2,'no',"relu",'adam','no',98.19])
x.add_row([2, 'yes','relu',"adam",'yes', 98.36])
x.add_row([2,'yes',"sigmoid",'SGD','yes',93.38])
x.add_row([2, 'yes','tanh',"SGD",'yes', 93.93])
x.add_row([2,'yes',"sigmoid",'adam','yes',98.18])
x.add_row([2, 'yes','tanh',"adam",'yes', 98.22])
x.add_row([3, 'no','relu','adam','no', 98.2])
x.add_row([3,'yes','relu','adam','yes', 98.35])
x.add_row([5,'no','relu','adam','no', 97.6])
x.add_row([5, 'yes','relu','adam','yes', 98.16])
print(x)
```

Hiddden layers	batch normalization	activation function	optimizer	dropout
2	no	relu	adam	no
2	yes	relu	adam	yes
2	yes	sigmoid	SGD	yes
2	yes	tanh	SGD	yes
2	yes	sigmoid	adam	yes
2	yes	tanh	adam	yes
3	no	relu	adam	no
3	yes	relu	adam	yes
5	no	relu	adam	no
5	yes	relu	adam	yes

Conclusion: 1.We can see that 2 hidden layer with batch normalization and dropout gives higher accuracy. 2.If th models tend to overfit. 3.Without doing batch normalization the convergennce rate is slow,so batch normalizatic without dropout and only batch normlization we can see it performs better than a MLP which doent have both ba weak regulizer. 5.We can observe Adam as best optimizer as compared to SGD. 6.Tanh activation function with S with SGD optimizer. 7.ReLu is best activation function. 8.So for MNIST dataset,a MLP works well when it has drc and 2 hidden layers

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu