In [1]:
```
from keras.utils import np_utils
from keras.datasets import mnist
from keras.initializers import RandomNormal
import seaborn as sns

%matplotlib inline
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [0]:
```
#Loading the training and the test data
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

In [3]:
```
#finding the shape of training and test data
print('Shape of training data is',x_train.shape[0],'and each image is of size {} x {}'.format(x_train.shape[
print('Shape of test dats is ',x_test.shape[0],'and each image is of size {} x {}'.format(x_test.shape[1],x_
```

```
Shape of training data is 60000 and each image is of size 28 x 28
Shape of test dats is  10000 and each image is of size 28 x 28
```

In [0]:
```
#as the input image is of size 28*28 thats why we will convert each one of them to a 1*784 vector
#i.e each pixel reperesents a dimension of the image
#we will reshape the matrix

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 [5]:
```
#shape of the data after converting from 3d to 2d

print('SHAPE OF TRAINING DATA IS: ',x_train.shape[0],'AND EACH IMAGE IS OF SIZE: ',x_train.shape[1])
print('SHAPE OF TEST DATA IS: ',x_test.shape[0],'EACH IMAGE IS OF SIZE: ',x_test.shape[1])
```

```
SHAPE OF TRAINING DATA IS:  60000 AND EACH IMAGE IS OF SIZE:  784
SHAPE OF TEST DATA IS:  10000 EACH IMAGE IS OF SIZE:  784
```

In [6]:
```python
#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]:

```

x_train = x_train/255
x_test = x_test/255
```

In [8]:

```
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.
```

In [9]:
```
#printing the class labels of some of the images
print('class label of first image',y_train[0])
print('class label of 11th image',y_train[10])
print('class label of 100th image',y_train[99])


#also we are converting here 10 class output to binary using to_categorical function of keras
#in a way we are performing one hot encoding on the output data

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

print('AFTER encoding')
print('output for first image is ',Y_train[0])
print('output for 11th image is',Y_train[10])
print('output for 100th image is ',Y_train[99])
```

```
class label of first image 5
class label of 11th image 3
class label of 100th image 1
AFTER encoding
output for first image is  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
output for 11th image is [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
output for 100th image is  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
```

WE will use three different architectures for model implementation

- Model with 2 hidden layers
- Model with 3 hidden layers
- Model with 5 hidden layers

  In Each Architecture We will implement 4 models:
  - MLP + Relu + Adam
  - MLP + Relu + Adam + Dropout
  - MLP + Relu + Adam + Batch Normalization
  - MLP + Relu + Adam + Dropout + Batch Normalization

In each of the models we will perform hypereparameter tuning using GridSearch CV and Randomized cv

In [0]:
```
import warnings
warnings.filterwarnings('ignore')
from keras.models import Sequential
from keras.layers import Dense,Activation
#here we are importing the sequential and the dense,activation to specify about the fully connected MLP and


#some model parameters

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

```

1. Architecture1: Model with 2 hidden layers:

input(784)-Relu(512)-Relu(256)-Output(10)

1.1 MLP + Relu + Adamoptimizer

In [11]:

```
""" for weight initialization we wii initailize using He normalization
"""
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.062)
# h2 =>  σ=√(2/(fan_in) = 0.088  => N(0,σ) = N(0,0.088)
# out =>  σ=√(2/(fan_in+1) = 0.120  => N(0,σ) = N(0,0.120))

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.088, seed=No
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(x_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat
```

```
WARNING: Logging before flag parsing goes to stderr.
W0827 14:22:42.262042 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
eras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get
_default_graph instead.

W0827 14:22:42.285037 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
eras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please use tf.compat.v1.placehol
der instead.

W0827 14:22:42.289400 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
eras/backend/tensorflow_backend.py:4115: The name tf.random_normal is deprecated. Please use tf.random.normal i
nstead.

W0827 14:22:42.337172 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
eras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please use tf.random.uniform
instead.

W0827 14:22:42.357798 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
eras/optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer inst
ead.

W0827 14:22:42.384944 139961174951808 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/k
```

```
eras/backend/tensorflow_backend.py:3295: The name tf.log is deprecated. Please use tf.math.log instead.

W0827 14:22:42.495736 139961174951808 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflo
w/python/ops/math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.2184 - acc: 0.9336 - val_loss: 0.1089
- val_acc: 0.9656
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0821 - acc: 0.9752 - val_loss: 0.0842
- val_acc: 0.9735
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0503 - acc: 0.9843 - val_loss: 0.0760
- val_acc: 0.9763
Epoch 4/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0347 - acc: 0.9891 - val_loss: 0.0828
- val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0259 - acc: 0.9919 - val_loss: 0.0734
- val_acc: 0.9780
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0199 - acc: 0.9937 - val_loss: 0.0757
- val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0180 - acc: 0.9942 - val_loss: 0.0632
```

```
- val_acc: 0.9815
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0128 - acc: 0.9961 - val_loss: 0.0935
- val_acc: 0.9752
Epoch 9/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0184 - acc: 0.9937 - val_loss: 0.0893
- val_acc: 0.9789
Epoch 10/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0125 - acc: 0.9958 - val_loss: 0.0877
- val_acc: 0.9784
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0113 - acc: 0.9965 - val_loss: 0.0981
- val_acc: 0.9779
Epoch 12/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0102 - acc: 0.9967 - val_loss: 0.0787
- val_acc: 0.9815
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0104 - acc: 0.9967 - val_loss: 0.0783
- val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0062 - acc: 0.9978 - val_loss: 0.0900
- val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0113 - acc: 0.9962 - val_loss: 0.0954
- val_acc: 0.9787
Epoch 16/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0089 - acc: 0.9971 - val_loss: 0.0846
- val_acc: 0.9823
Epoch 17/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0089 - acc: 0.9972 - val_loss: 0.0924
- val_acc: 0.9794
Epoch 18/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0070 - acc: 0.9978 - val_loss: 0.0989
- val_acc: 0.9795
Epoch 19/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0101 - acc: 0.9969 - val_loss: 0.0958
- val_acc: 0.9804
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0071 - acc: 0.9978 - val_loss: 0.0965
- val_acc: 0.9811
```

In [12]:

```
#evaluation on test data

score = model_relu.evaluate(x_test,Y_test,verbose = 1)
print('Loss on test data is: ',score[0])
print('accuracy on test data is: ',score[1])
```

```
10000/10000 [==============================] - 1s 53us/step
Loss on test data is:  0.09651037384942601
accuracy on test data is:  0.9811
```

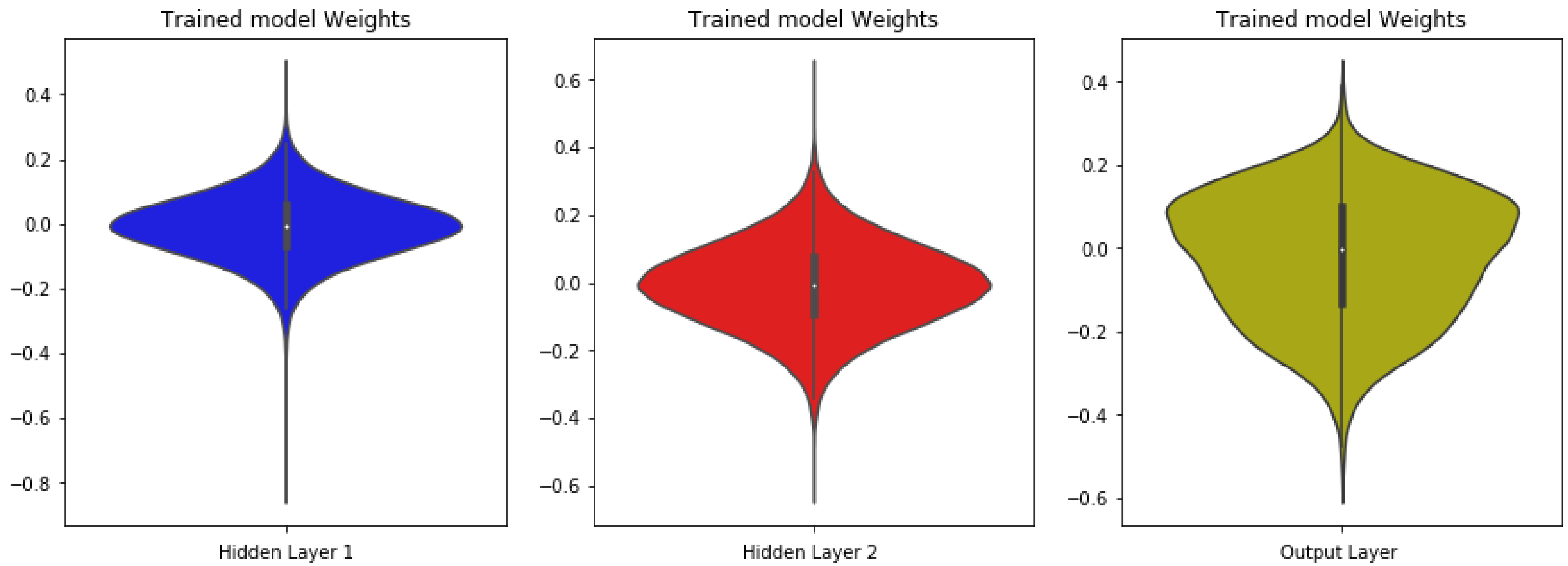
In [0]:

```
import pickle
def savetofile(obj,filename):
  pickle.dump(obj,open(filename+".p",'wb'))

def openfromfile(filename):
  temp = pickle.load(open(filename+".p",'rb'))
  return temp
```

In [0]:

```
def violin_plot(dl_model):
  w_after= dl_model.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(figsize = (15,5))
  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()
```

In [16]:
```
violin_plot(model_relu)
```



In [0]:
```
arch_1_model_1 = savetofile(history,'arch_1_model_1')
```

1.2 MLP + Relu + Adamoptimizer + dropout

In [19]:
```
from keras.layers import Dropout
model_relu_drop = Sequential()

model_relu_drop.add(Dense(512,activation = 'relu',input_shape = (input_dim,), kernel_initializer = RandomNor
model_relu_drop.add(Dropout(0.5))
#adding the dropout layer for each layer

model_relu_drop.add(Dense(256,activation = 'relu',kernel_initializer = RandomNormal(mean = 0.0,stddev = 0.08
model_relu_drop.add(Dropout(0.5))


model_relu_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_drop.summary())

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

```
W0827 14:28:45.499598 139961174951808 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/keras/bac
kend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecat
ed and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
```

```
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.4535 - acc: 0.8571 - val_loss: 0.1509 - v
al_acc: 0.9526
Epoch 2/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.2033 - acc: 0.9391 - val_loss: 0.1025 - v
al_acc: 0.9680
Epoch 3/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1537 - acc: 0.9540 - val_loss: 0.0862 - v
al_acc: 0.9721
Epoch 4/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1307 - acc: 0.9603 - val_loss: 0.0781 - v
al_acc: 0.9761
Epoch 5/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1137 - acc: 0.9660 - val_loss: 0.0754 - v
al_acc: 0.9763
Epoch 6/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1003 - acc: 0.9696 - val_loss: 0.0711 - v
al_acc: 0.9772
Epoch 7/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0957 - acc: 0.9706 - val_loss: 0.0658 - v
al_acc: 0.9801
Epoch 8/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0860 - acc: 0.9743 - val_loss: 0.0654 - v
al_acc: 0.9791
Epoch 9/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0782 - acc: 0.9752 - val_loss: 0.0686 - v
al_acc: 0.9793
Epoch 10/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0756 - acc: 0.9766 - val_loss: 0.0648 - v
al_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0709 - acc: 0.9780 - val_loss: 0.0611 - v
al_acc: 0.9827
Epoch 12/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0662 - acc: 0.9795 - val_loss: 0.0585 - v
al_acc: 0.9828
Epoch 13/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0611 - acc: 0.9810 - val_loss: 0.0616 - v
al_acc: 0.9827
Epoch 14/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0610 - acc: 0.9803 - val_loss: 0.0604 - v
```

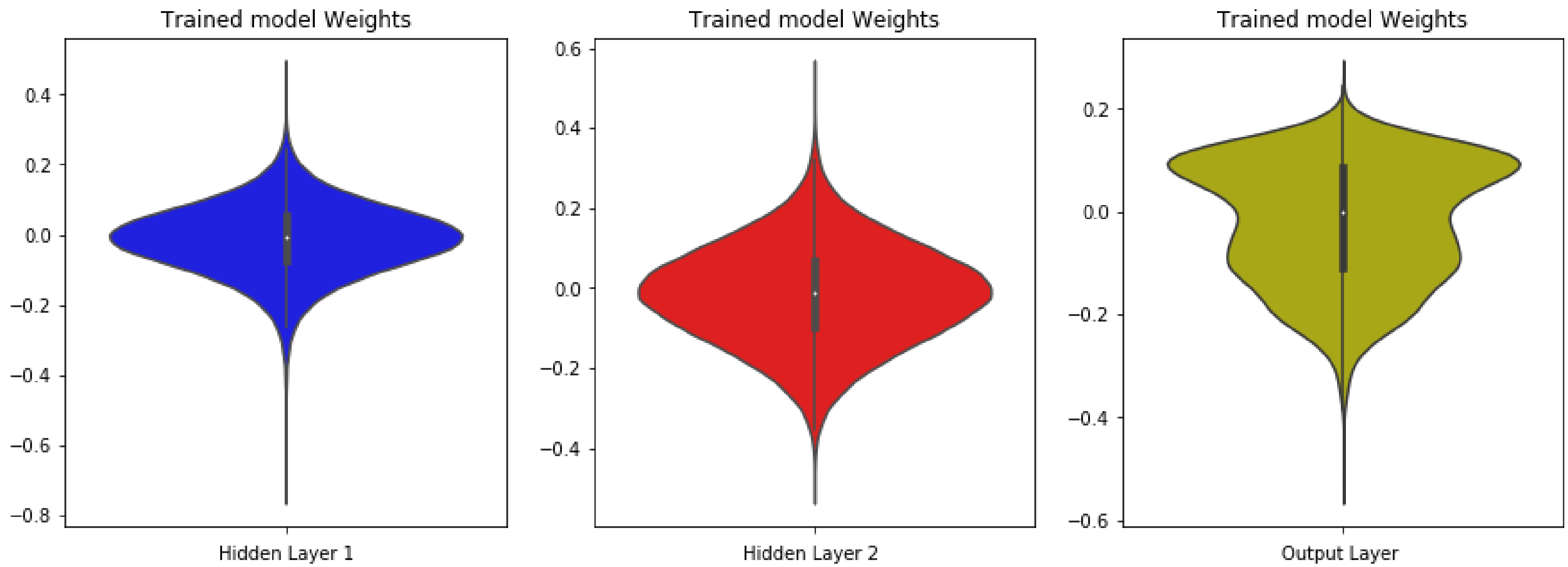
```
al_acc: 0.9821
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0569 - acc: 0.9823 - val_loss: 0.0633 - v
al_acc: 0.9829
Epoch 16/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0573 - acc: 0.9820 - val_loss: 0.0608 - v
al_acc: 0.9837
Epoch 17/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0543 - acc: 0.9827 - val_loss: 0.0571 - v
al_acc: 0.9842
Epoch 18/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0502 - acc: 0.9835 - val_loss: 0.0579 - v
al_acc: 0.9843
Epoch 19/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0527 - acc: 0.9835 - val_loss: 0.0584 - v
al_acc: 0.9840
Epoch 20/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0475 - acc: 0.9850 - val_loss: 0.0589 - v
al_acc: 0.9847
```

In [20]:

```
score = model_relu_drop.evaluate(x_test,Y_test,verbose = 1)
print('Loss on test data is: ',score[0])
print('accuracy on test data is: ',score[1])
```

```
10000/10000 [==============================] - 1s 53us/step
Loss on test data is:  0.058890673811543455
accuracy on test data is:  0.9847
```

In [21]:
```
violin_plot(model_relu_drop)
```



In [0]:
```
arch_1_model_2 = savetofile(history,'arch_1_model_2')
```

1.3 MLP + Relu + Adamoptimizer + BatchNormalization

In [23]:

```
from keras.layers import BatchNormalization

model_relu_batch = Sequential()
model_relu_batch.add(Dense(512,activation = 'relu',input_shape = (input_dim,),kernel_initializer = RandomNor
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(256,activation = 'relu',kernel_initializer = RandomNormal(mean = 0.0,stddev = 0.0
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch.summary())


model_relu_batch.compile(optimizer = 'adam',loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_relu_batch.fit(x_train,Y_train,batch_size = batch_size,epochs = nb_epoch,verbose = 1,validat

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328    
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024      
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2570      
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1855 - acc: 0.9443 - val_loss: 0.0889 - v
al_acc: 0.9716
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0671 - acc: 0.9799 - val_loss: 0.0789 - v
```

```
al_acc: 0.9741
Epoch 3/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0414 - acc: 0.9875 - val_loss: 0.0822 - v
al_acc: 0.9726
Epoch 4/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0305 - acc: 0.9901 - val_loss: 0.0771 - v
al_acc: 0.9767
Epoch 5/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0228 - acc: 0.9929 - val_loss: 0.0761 - v
al_acc: 0.9761
Epoch 6/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0198 - acc: 0.9936 - val_loss: 0.0837 - v
al_acc: 0.9759
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0213 - acc: 0.9930 - val_loss: 0.0885 - v
al_acc: 0.9739
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0168 - acc: 0.9946 - val_loss: 0.0790 - v
al_acc: 0.9778
Epoch 9/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0128 - acc: 0.9959 - val_loss: 0.0789 - v
al_acc: 0.9779
Epoch 10/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0126 - acc: 0.9961 - val_loss: 0.0750 - v
al_acc: 0.9800
Epoch 11/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0113 - acc: 0.9963 - val_loss: 0.0772 - v
al_acc: 0.9792
Epoch 12/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0122 - acc: 0.9960 - val_loss: 0.0862 - v
al_acc: 0.9782
Epoch 13/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0099 - acc: 0.9968 - val_loss: 0.0818 - v
al_acc: 0.9784
Epoch 14/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0091 - acc: 0.9971 - val_loss: 0.0866 - v
al_acc: 0.9800
Epoch 15/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0085 - acc: 0.9974 - val_loss: 0.0882 - v
al_acc: 0.9785
Epoch 16/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0071 - acc: 0.9975 - val_loss: 0.0876 - v
al_acc: 0.9781
```

```
Epoch 17/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0080 - acc: 0.9972 - val_loss: 0.0940 - v
al_acc: 0.9783
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0118 - acc: 0.9960 - val_loss: 0.0801 - v
al_acc: 0.9813
Epoch 19/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0070 - acc: 0.9979 - val_loss: 0.0739 - v
al_acc: 0.9830
Epoch 20/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0087 - acc: 0.9974 - val_loss: 0.0840 - v
al_acc: 0.9798
```

In [24]:

```
score = model_relu_batch.evaluate(x_test,Y_test,verbose = 1)
print('Loss on the test data is: ',score[0])
print('Accuracy on the test data is:',score[1])

```

```
10000/10000 [==============================] - 1s 64us/step
Loss on the test data is:  0.0839915322766581
Accuracy on the test data is: 0.9798
```

In [25]:
```
violin_plot(model_relu_batch)
```

In [0]:
```
arch_1_mode_3 = savetofile(history,'arch_1_model_3')
```

1.4 MLP + Relu + Adamoptimizer + BatchNormalization +Dropout

In [27]:

```
model_relu_batch_drop = Sequential()
model_relu_batch_drop.add(Dense(512,activation = 'relu',input_shape = (input_dim,),kernel_initializer = Rand
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(256,activation = 'relu',kernel_initializer = RandomNormal(mean = 0.0,stddev
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch_drop.summary())

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                2570      
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

```
60000/60000 [==============================] - 6s 100us/step - loss: 0.4389 - acc: 0.8669 - val_loss: 0.1409 -
val_acc: 0.9577
Epoch 2/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.2087 - acc: 0.9364 - val_loss: 0.1097 - v
al_acc: 0.9651
Epoch 3/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1626 - acc: 0.9494 - val_loss: 0.0911 - v
al_acc: 0.9713
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1358 - acc: 0.9578 - val_loss: 0.0831 - v
al_acc: 0.9743
Epoch 5/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.1191 - acc: 0.9632 - val_loss: 0.0717 - v
al_acc: 0.9764
Epoch 6/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.1086 - acc: 0.9653 - val_loss: 0.0715 - v
al_acc: 0.9774
Epoch 7/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0967 - acc: 0.9699 - val_loss: 0.0666 - v
al_acc: 0.9784
Epoch 8/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0905 - acc: 0.9710 - val_loss: 0.0629 - v
al_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0853 - acc: 0.9730 - val_loss: 0.0667 - v
al_acc: 0.9804
Epoch 10/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0817 - acc: 0.9743 - val_loss: 0.0570 - v
al_acc: 0.9820
Epoch 11/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0735 - acc: 0.9763 - val_loss: 0.0621 - v
al_acc: 0.9823
Epoch 12/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0695 - acc: 0.9775 - val_loss: 0.0576 - v
al_acc: 0.9831
Epoch 13/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0649 - acc: 0.9787 - val_loss: 0.0604 - v
al_acc: 0.9834
Epoch 14/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0621 - acc: 0.9804 - val_loss: 0.0583 - v
al_acc: 0.9828
Epoch 15/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0614 - acc: 0.9798 - val_loss: 0.0581 - v
```

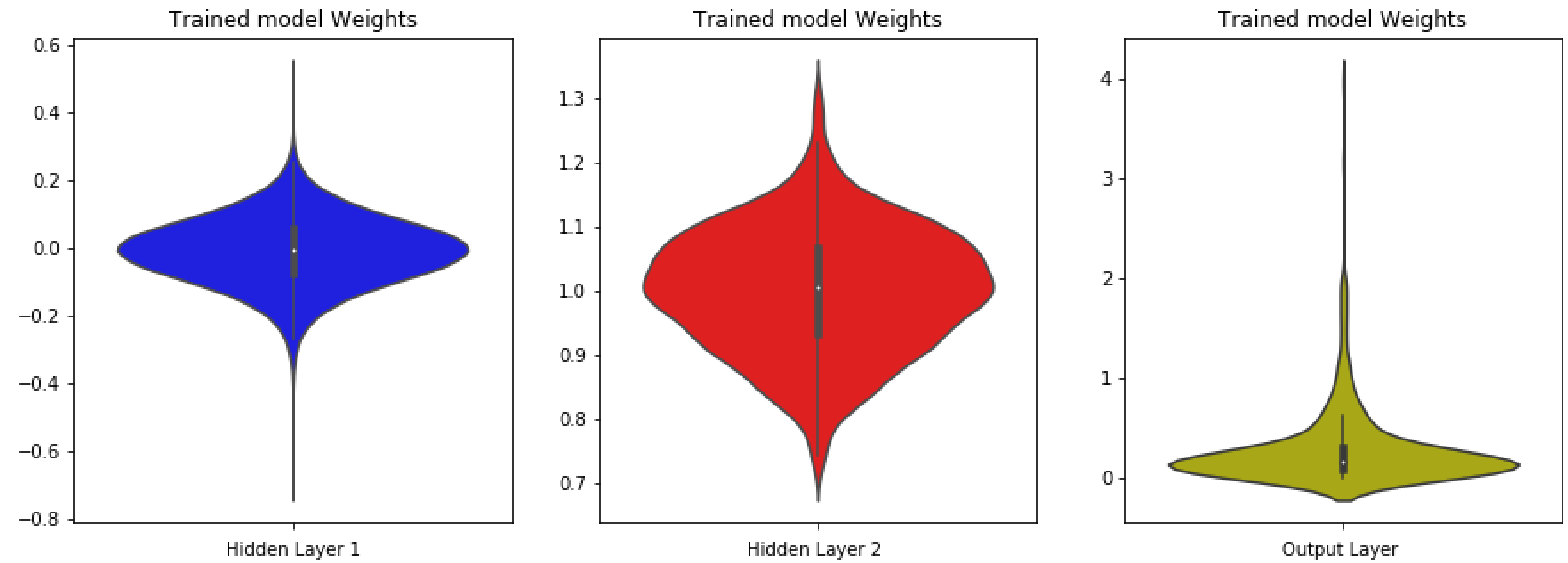
```
al_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0584 - acc: 0.9813 - val_loss: 0.0557 - v
al_acc: 0.9841
Epoch 17/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0563 - acc: 0.9818 - val_loss: 0.0537 - v
al_acc: 0.9840
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0566 - acc: 0.9815 - val_loss: 0.0601 - v
al_acc: 0.9825
Epoch 19/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0505 - acc: 0.9837 - val_loss: 0.0553 - v
al_acc: 0.9838
Epoch 20/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0469 - acc: 0.9848 - val_loss: 0.0551 - v
al_acc: 0.9846
```

In [28]:

```
score = model_relu_batch_drop.evaluate(x_test,Y_test)
print('loss on test data is:',score[0])
print('accuracy on test data is',score[1])
```

```
10000/10000 [==============================] - 1s 68us/step
loss on test data is: 0.05505812452407554
accuracy on test data is 0.9846
```

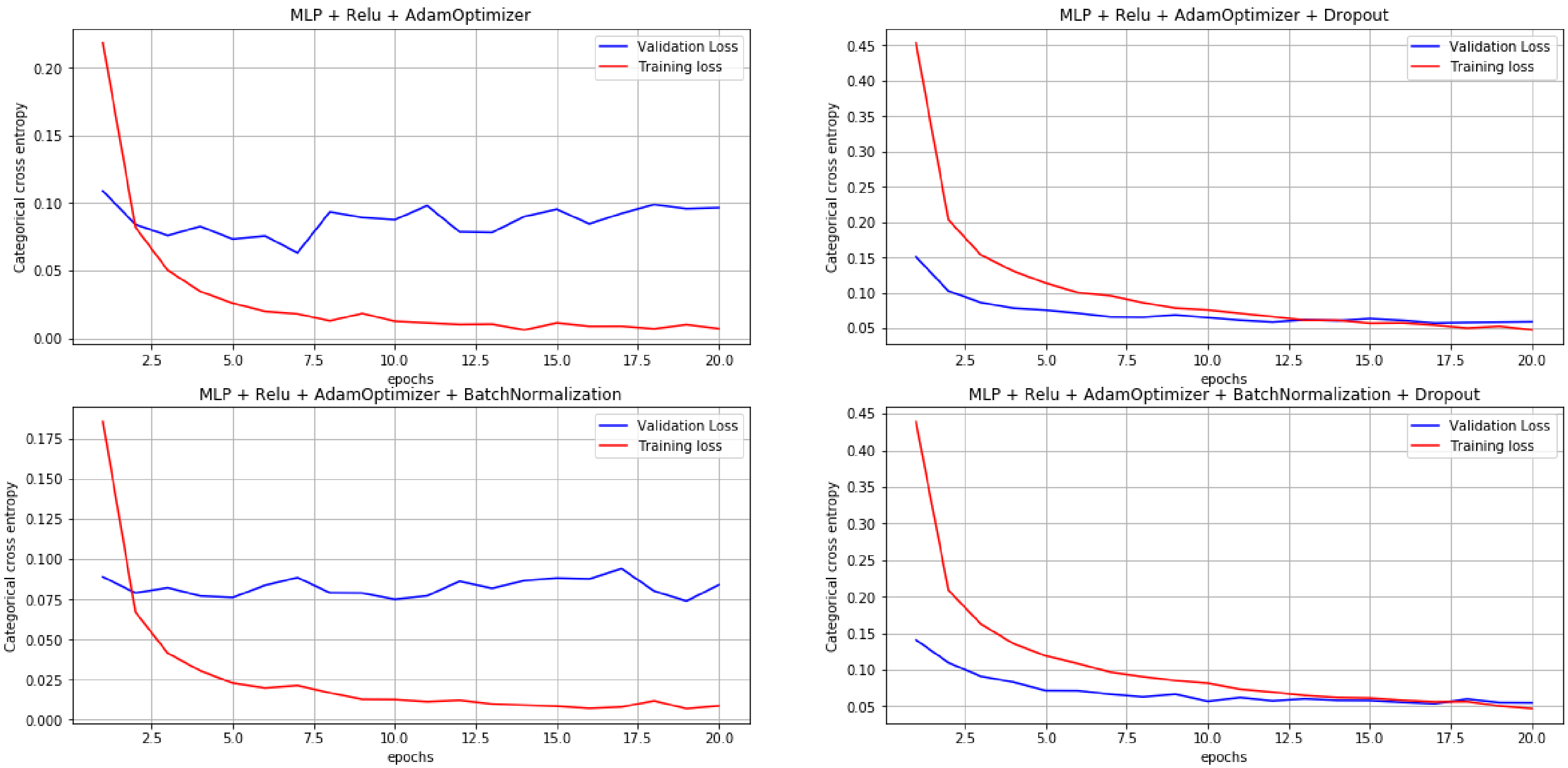
In [29]:
```
violin_plot(model_relu_batch_drop)
```



In [0]:
```
arch_1_model_4 = savetofile(history,'arch_1_model_4')
```

In [31]:
```
#plotting for all 4 models

plt.figure(figsize = (20,20))
plt.grid()
x = list(range(1,nb_epoch+1))

"""MODEL 1"""
plt.subplot(4,2,1)
plt.title('MLP + Relu + AdamOptimizer')
plt.grid()
plt.plot(x,openfromfile('arch_1_model_1').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_1_model_1').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 2"""

plt.subplot(4,2,2)
plt.title('MLP + Relu + AdamOptimizer + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_1_model_2').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_1_model_2').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()



"""MODEL 3"""
plt.subplot(4,2,3)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization')
plt.grid()
plt.plot(x,openfromfile('arch_1_model_3').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_1_model_3').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 4"""
```

```python
plt.subplot(4,2,4)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_1_model_4').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_1_model_4').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()
plt.show()
```



2. ARCHITECTURE 2: Model with 3 hidden layers

Input(786) - relu(1000) - relu(500)-relu(250)-softmax(10)

In [0]:
```
### Model 1: MLP + Relu + Adamoptimizer
```

In [33]:

```python


# for relu layers
from keras.initializers import he_normal

model_relu = Sequential()
model_relu.add(Dense(1000, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed =
model_relu.add(Dense(500, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(250, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(x_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 1000)              785000    
_________________________________________________________________
dense_14 (Dense)             (None, 500)               500500    
_________________________________________________________________
dense_15 (Dense)             (None, 250)               125250    
_________________________________________________________________
dense_16 (Dense)             (None, 10)                2510      
=================================================================
Total params: 1,413,260
Trainable params: 1,413,260
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1955 - acc: 0.9404 - val_loss: 0.0872 - v
al_acc: 0.9739
Epoch 2/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0765 - acc: 0.9768 - val_loss: 0.0833 - v
al_acc: 0.9733
```

```
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0501 - acc: 0.9835 - val_loss: 0.0751 - v
al_acc: 0.9767
Epoch 4/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0380 - acc: 0.9879 - val_loss: 0.0693 - v
al_acc: 0.9806
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0297 - acc: 0.9906 - val_loss: 0.0716 - v
al_acc: 0.9803
Epoch 6/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0258 - acc: 0.9916 - val_loss: 0.0795 - v
al_acc: 0.9777
Epoch 7/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0213 - acc: 0.9930 - val_loss: 0.0797 - v
al_acc: 0.9797
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0167 - acc: 0.9947 - val_loss: 0.0774 - v
al_acc: 0.9798
Epoch 9/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0193 - acc: 0.9935 - val_loss: 0.0741 - v
al_acc: 0.9818
Epoch 10/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0167 - acc: 0.9947 - val_loss: 0.0844 - v
al_acc: 0.9785
Epoch 11/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0141 - acc: 0.9956 - val_loss: 0.0713 - v
al_acc: 0.9823
Epoch 12/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0123 - acc: 0.9962 - val_loss: 0.0977 - v
al_acc: 0.9771
Epoch 13/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0123 - acc: 0.9964 - val_loss: 0.0915 - v
al_acc: 0.9809
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0138 - acc: 0.9957 - val_loss: 0.0800 - v
al_acc: 0.9821
Epoch 15/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0059 - acc: 0.9983 - val_loss: 0.0838 - v
al_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0167 - acc: 0.9952 - val_loss: 0.0834 - v
al_acc: 0.9831
Epoch 17/20
```

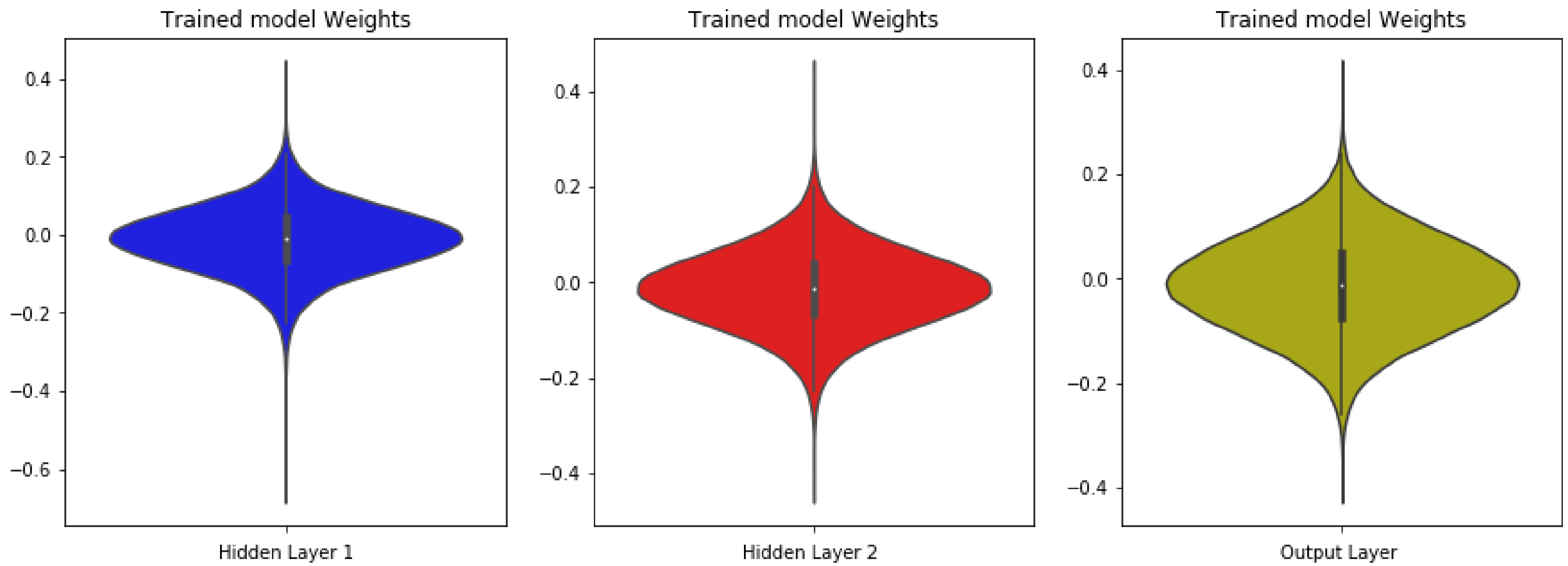
```
60000/60000 [==============================] - 4s 70us/step - loss: 0.0077 - acc: 0.9975 - val_loss: 0.1016 - v
al_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0140 - acc: 0.9958 - val_loss: 0.1018 - v
al_acc: 0.9777
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0067 - acc: 0.9981 - val_loss: 0.0938 - v
al_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0085 - acc: 0.9975 - val_loss: 0.0882 - v
al_acc: 0.9820
```

In [34]:

```
#evaluation on test data

score = model_relu.evaluate(x_test,Y_test,verbose = 1)
print('Loss on test data is: ',score[0])
print('accuracy on test data is: ',score[1])
```

```
10000/10000 [==============================] - 1s 71us/step
Loss on test data is:  0.08816329310913239
accuracy on test data is:  0.982
```

In [37]:
```
violin_plot(model_relu)
```



In [0]:
```
arch_2_model_1 = savetofile(history,'arch_2_model_1')
```

Model 2: MLP + Relu + AdamOptimizer + Dropout

In [39]:

```
from keras.layers import Dropout
model_relu_drop = Sequential()

model_relu_drop.add(Dense(1000,activation = 'relu',input_shape = (input_dim,), kernel_initializer = he_norma
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(250,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))



model_relu_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_drop.summary())

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 1000)              785000
_________________________________________________________________
dropout_5 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_18 (Dense)             (None, 500)               500500
_________________________________________________________________
dropout_6 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_19 (Dense)             (None, 250)               125250
_________________________________________________________________
dropout_7 (Dropout)          (None, 250)               0
_________________________________________________________________
dense_20 (Dense)             (None, 10)                2510
```

```
=================================================================
Total params: 1,413,260
Trainable params: 1,413,260
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.4537 - acc: 0.8580 - val_loss: 0.1325 -
val_acc: 0.9586
Epoch 2/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1918 - acc: 0.9444 - val_loss: 0.1045 - v
al_acc: 0.9677
Epoch 3/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1493 - acc: 0.9565 - val_loss: 0.0926 - v
al_acc: 0.9727
Epoch 4/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1252 - acc: 0.9640 - val_loss: 0.0826 - v
al_acc: 0.9759
Epoch 5/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1088 - acc: 0.9691 - val_loss: 0.0775 - v
al_acc: 0.9762
Epoch 6/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1008 - acc: 0.9706 - val_loss: 0.0667 - v
al_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0888 - acc: 0.9741 - val_loss: 0.0665 - v
al_acc: 0.9787
Epoch 8/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0840 - acc: 0.9753 - val_loss: 0.0698 - v
al_acc: 0.9789
Epoch 9/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0818 - acc: 0.9760 - val_loss: 0.0636 - v
al_acc: 0.9819
Epoch 10/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0736 - acc: 0.9788 - val_loss: 0.0615 - v
al_acc: 0.9820
Epoch 11/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0704 - acc: 0.9791 - val_loss: 0.0594 - v
al_acc: 0.9816
Epoch 12/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0676 - acc: 0.9798 - val_loss: 0.0598 - v
al_acc: 0.9831
```

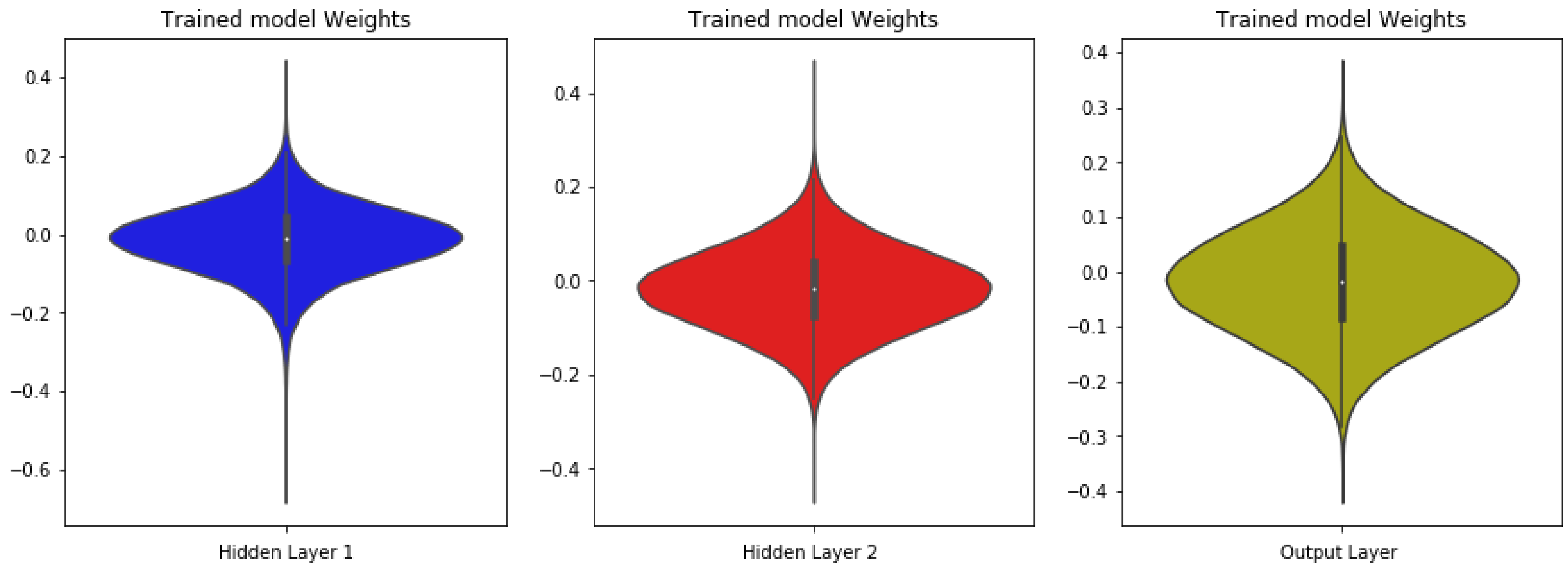
```
Epoch 13/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0634 - acc: 0.9816 - val_loss: 0.0677 - v
al_acc: 0.9809
Epoch 14/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0616 - acc: 0.9816 - val_loss: 0.0600 - v
al_acc: 0.9830
Epoch 15/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0598 - acc: 0.9825 - val_loss: 0.0607 - v
al_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0558 - acc: 0.9833 - val_loss: 0.0638 - v
al_acc: 0.9833
Epoch 17/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0565 - acc: 0.9831 - val_loss: 0.0693 - v
al_acc: 0.9824
Epoch 18/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0524 - acc: 0.9847 - val_loss: 0.0585 - v
al_acc: 0.9849
Epoch 19/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0526 - acc: 0.9845 - val_loss: 0.0642 - v
al_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0504 - acc: 0.9851 - val_loss: 0.0612 - v
al_acc: 0.9836
```

In [40]:

```
score = model_relu_drop.evaluate(x_test,Y_test,verbose = 1)
print('Loss on test data is: ',score[0])
print('accuracy on test data is: ',score[1])
```

```
10000/10000 [==============================] - 1s 69us/step
Loss on test data is:  0.061200255445798345
accuracy on test data is:  0.9836
```

In [41]:
```
violin_plot(model_relu_drop)
```



In [0]:
```
arch_2_model_2 = savetofile(history,'arch_2_model_2')
```

Model3 : MLP + Relu + Adamoptimizer + BatchNormalization

In [43]:

```
## Model 3
model_relu_batch = Sequential()
model_relu_batch.add(Dense(1000,activation = 'relu',input_shape = (input_dim,),kernel_initializer = he_norma
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(250,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch.summary())


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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_21 (Dense)             (None, 1000)              785000    
_________________________________________________________________
batch_normalization_5 (Batch (None, 1000)              4000      
_________________________________________________________________
dense_22 (Dense)             (None, 500)               500500    
_________________________________________________________________
batch_normalization_6 (Batch (None, 500)               2000      
_________________________________________________________________
dense_23 (Dense)             (None, 250)               125250    
_________________________________________________________________
batch_normalization_7 (Batch (None, 250)               1000      
_________________________________________________________________
dense_24 (Dense)             (None, 10)                2510      
=================================================================
Total params: 1,420,260
Trainable params: 1,416,760
Non-trainable params: 3,500
_________________________________________________________________
```

```
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1613 - acc: 0.9502 - val_loss: 0.0893 -
val_acc: 0.9721
Epoch 2/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0654 - acc: 0.9794 - val_loss: 0.0841 -
val_acc: 0.9728
Epoch 3/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0441 - acc: 0.9862 - val_loss: 0.0803 -
val_acc: 0.9755
Epoch 4/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0356 - acc: 0.9882 - val_loss: 0.0733 -
val_acc: 0.9767
Epoch 5/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0283 - acc: 0.9907 - val_loss: 0.0960 -
val_acc: 0.9718
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0272 - acc: 0.9906 - val_loss: 0.0766 -
val_acc: 0.9778
Epoch 7/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0212 - acc: 0.9927 - val_loss: 0.0784 -
val_acc: 0.9765
Epoch 8/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0202 - acc: 0.9933 - val_loss: 0.0773 -
val_acc: 0.9794
Epoch 9/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0188 - acc: 0.9939 - val_loss: 0.0950 -
val_acc: 0.9755
Epoch 10/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0167 - acc: 0.9943 - val_loss: 0.0744 -
val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0164 - acc: 0.9945 - val_loss: 0.0881 -
val_acc: 0.9785
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0164 - acc: 0.9944 - val_loss: 0.0763 -
val_acc: 0.9805
Epoch 13/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0127 - acc: 0.9957 - val_loss: 0.0662 -
val_acc: 0.9827
Epoch 14/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0106 - acc: 0.9964 - val_loss: 0.0812 -
```

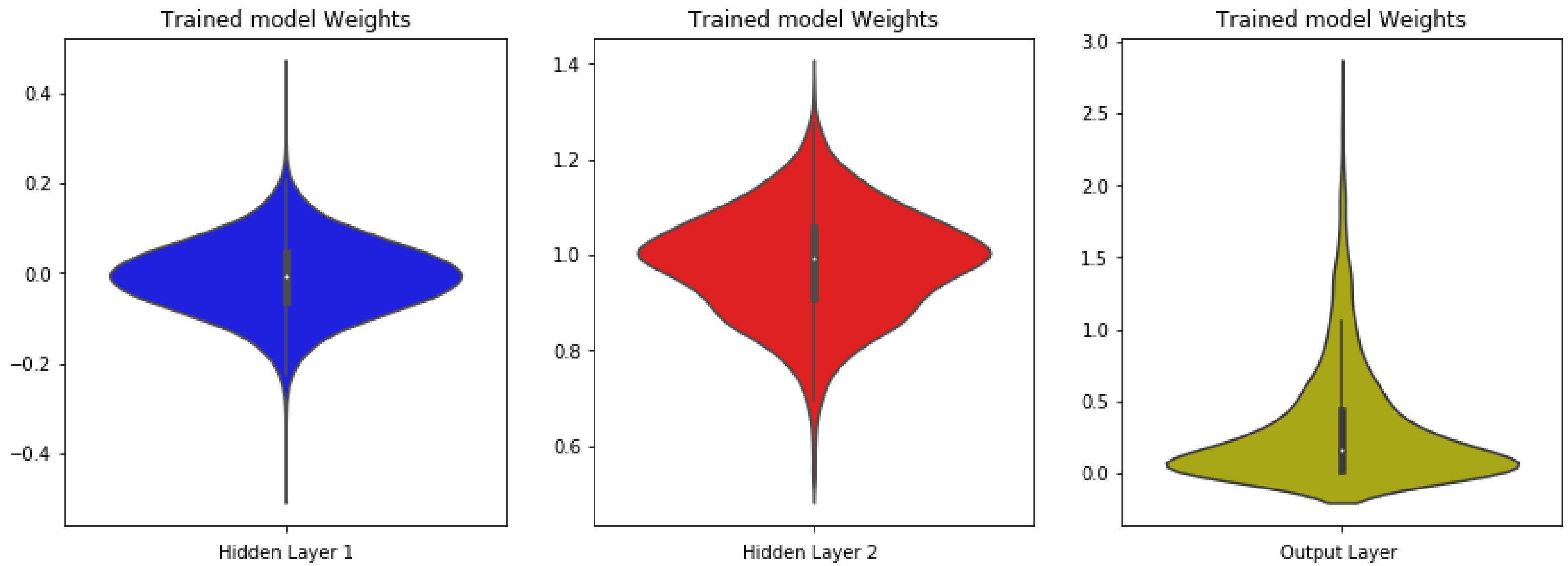
```
val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0120 - acc: 0.9961 - val_loss: 0.1095 -
val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0121 - acc: 0.9958 - val_loss: 0.0758 -
val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0085 - acc: 0.9972 - val_loss: 0.0858 -
val_acc: 0.9786
Epoch 18/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0094 - acc: 0.9965 - val_loss: 0.0894 -
val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0113 - acc: 0.9963 - val_loss: 0.0856 -
val_acc: 0.9792
Epoch 20/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0071 - acc: 0.9979 - val_loss: 0.0750 -
val_acc: 0.9837
```

In [44]:

```
score = model_relu_batch.evaluate(x_test,Y_test,verbose = 1)
print('Loss on the test data is: ',score[0])
print('Accuracy on the test data is:',score[1])
```

```
10000/10000 [==============================] - 1s 100us/step
Loss on the test data is:  0.07504327980409116
Accuracy on the test data is: 0.9837
```

In [45]:
```
violin_plot(model_relu_batch)
```



In [0]:
```
arch_2_model_3 = savetofile(history,'arch_2_model_3')
```

Model4: MLP + Relu + Adamoptimizer + BatchNormalization + Dropout

In [47]:

```
## Model 4

model_relu_batch_drop = Sequential()
model_relu_batch_drop.add(Dense(1000,activation = 'relu',input_shape = (input_dim,),kernel_initializer = he_
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(250,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch_drop.summary())

model_relu_batch_drop.compile(optimizer = 'adam',loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_relu_batch_drop.fit(x_train,Y_train,batch_size = batch_size,epochs = nb_epoch,verbose = 1,va

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 1000)              785000    
_________________________________________________________________
batch_normalization_8 (Batch (None, 1000)              4000      
_________________________________________________________________
dropout_8 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_26 (Dense)             (None, 500)               500500    
_________________________________________________________________
batch_normalization_9 (Batch (None, 500)               2000      
_________________________________________________________________
dropout_9 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 250)               125250    
```

```
_________________________________________________________________
batch_normalization_10 (Batc (None, 250)               1000
_________________________________________________________________
dropout_10 (Dropout)         (None, 250)               0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                2510
=================================================================
Total params: 1,420,260
Trainable params: 1,416,760
Non-trainable params: 3,500
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.4173 - acc: 0.8741 - val_loss: 0.1345 -
val_acc: 0.9598
Epoch 2/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.1879 - acc: 0.9437 - val_loss: 0.1048 -
val_acc: 0.9661
Epoch 3/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1495 - acc: 0.9543 - val_loss: 0.0883 -
val_acc: 0.9721
Epoch 4/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1230 - acc: 0.9624 - val_loss: 0.0769 -
val_acc: 0.9770
Epoch 5/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1073 - acc: 0.9668 - val_loss: 0.0709 -
val_acc: 0.9775
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0976 - acc: 0.9700 - val_loss: 0.0645 -
val_acc: 0.9791
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0920 - acc: 0.9710 - val_loss: 0.0658 -
val_acc: 0.9796
Epoch 8/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0809 - acc: 0.9750 - val_loss: 0.0582 -
val_acc: 0.9818
Epoch 9/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0771 - acc: 0.9749 - val_loss: 0.0539 -
val_acc: 0.9835
Epoch 10/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0719 - acc: 0.9772 - val_loss: 0.0629 -
val_acc: 0.9829
```

```
Epoch 11/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0666 - acc: 0.9790 - val_loss: 0.0610 -
val_acc: 0.9820
Epoch 12/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0657 - acc: 0.9789 - val_loss: 0.0617 -
val_acc: 0.9822
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0604 - acc: 0.9814 - val_loss: 0.0591 -
val_acc: 0.9829
Epoch 14/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0596 - acc: 0.9809 - val_loss: 0.0558 -
val_acc: 0.9826
Epoch 15/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0573 - acc: 0.9815 - val_loss: 0.0581 -
val_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0569 - acc: 0.9819 - val_loss: 0.0594 -
val_acc: 0.9824
Epoch 17/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0504 - acc: 0.9837 - val_loss: 0.0548 -
val_acc: 0.9843
Epoch 18/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0498 - acc: 0.9842 - val_loss: 0.0566 -
val_acc: 0.9835
Epoch 19/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0461 - acc: 0.9854 - val_loss: 0.0540 -
val_acc: 0.9843
Epoch 20/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0447 - acc: 0.9855 - val_loss: 0.0521 -
val_acc: 0.9849
```

In [48]:

```
score = model_relu_batch_drop.evaluate(x_test,Y_test)
print('loss on test data is:',score[0])
print('accuracy on test data is',score[1])
```

```
10000/10000 [==============================] - 1s 95us/step
loss on test data is: 0.05211034208216879
accuracy on test data is 0.9849
```

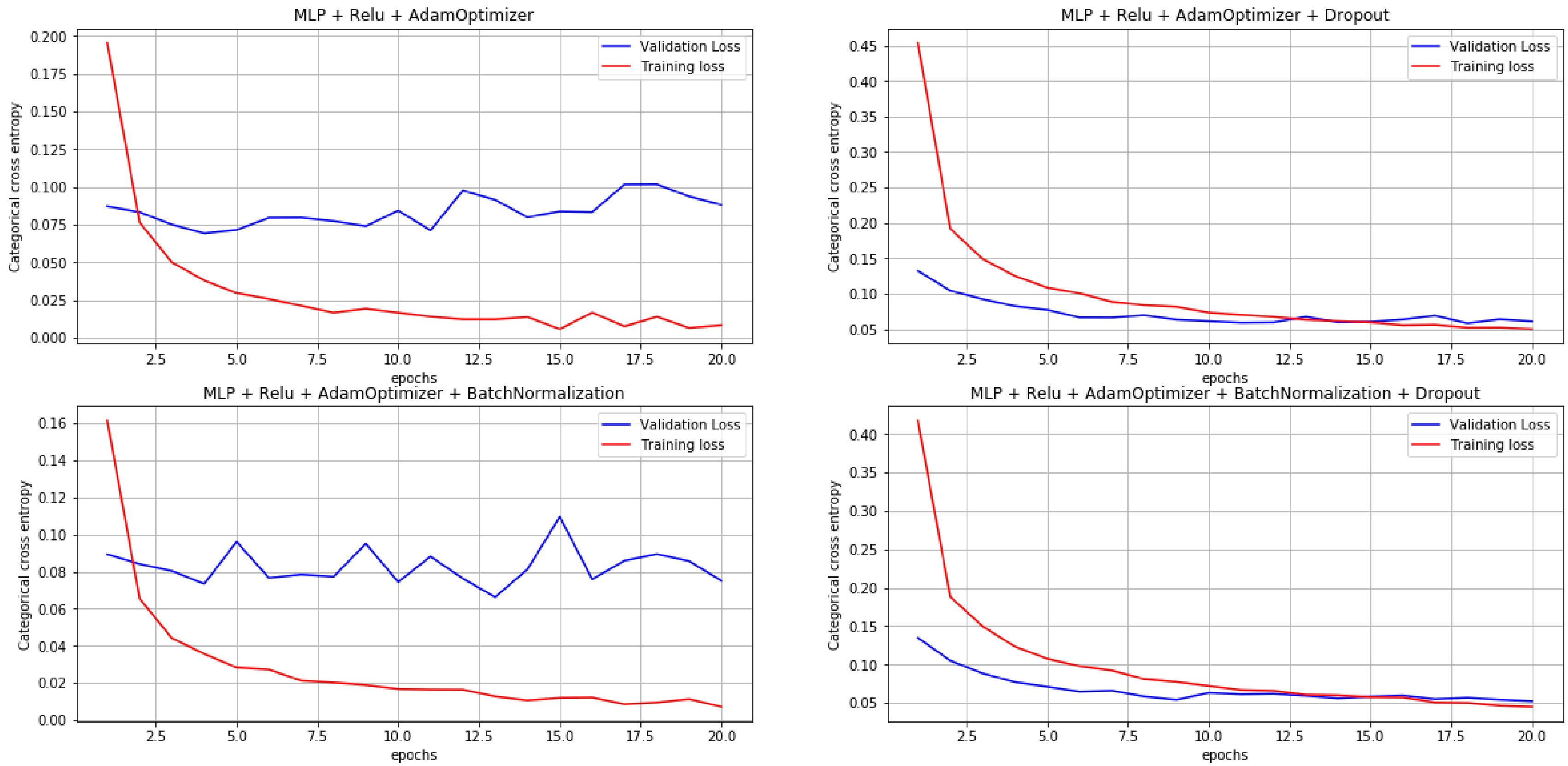
In [49]:
```
violin_plot(model_relu_batch_drop)
```

In [0]:
```
arch_2_model_4 = savetofile(history,'arch_2_model_4')
```

In [54]:

```
#plotting for all four models

plt.figure(figsize = (20,20))
#plt.grid()
x = list(range(1,nb_epoch+1))

"""MODEL 1"""
plt.subplot(4,2,1)
plt.title('MLP + Relu + AdamOptimizer')
plt.grid()
plt.plot(x,openfromfile('arch_2_model_1').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_2_model_1').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 2"""

plt.subplot(4,2,2)
plt.title('MLP + Relu + AdamOptimizer + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_2_model_2').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_2_model_2').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()



"""MODEL 3"""
plt.subplot(4,2,3)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization')
plt.grid()
plt.plot(x,openfromfile('arch_2_model_3').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_2_model_3').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 4"""
```

```python
plt.subplot(4,2,4)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_2_model_4').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_2_model_4').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()
plt.show()
```



Architecture 3: Model with 5 hidden layers

Input(786) - relu(200) - relu(300) - relu (400) - relu(500) - relu(600) - softmax(10)

Model1: MLP + Relu + Adamoptimizer

In [66]:
```
from keras.initializers import he_normal

model_relu = Sequential()
model_relu.add(Dense(200, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed = N
model_relu.add(Dense(300, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(400, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(500, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(600, activation='relu', kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(x_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_dat

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_59 (Dense)             (None, 200)               157000    
_________________________________________________________________
dense_60 (Dense)             (None, 300)               60300     
_________________________________________________________________
dense_61 (Dense)             (None, 400)               120400    
_________________________________________________________________
dense_62 (Dense)             (None, 500)               200500    
_________________________________________________________________
dense_63 (Dense)             (None, 600)               300600    
_________________________________________________________________
dense_64 (Dense)             (None, 10)                6010      
=================================================================
Total params: 844,810
Trainable params: 844,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.2383 - acc: 0.9274 - val_loss: 0.1470 - 
val_acc: 0.9552
```

```
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1032 - acc: 0.9681 - val_loss: 0.1007 - v
al_acc: 0.9709
Epoch 3/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0711 - acc: 0.9781 - val_loss: 0.0999 - v
al_acc: 0.9700
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0573 - acc: 0.9820 - val_loss: 0.0823 - v
al_acc: 0.9765
Epoch 5/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0449 - acc: 0.9854 - val_loss: 0.0920 - v
al_acc: 0.9745
Epoch 6/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0400 - acc: 0.9878 - val_loss: 0.1036 - v
al_acc: 0.9732
Epoch 7/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0338 - acc: 0.9893 - val_loss: 0.1101 - v
al_acc: 0.9754
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0324 - acc: 0.9893 - val_loss: 0.1165 - v
al_acc: 0.9704
Epoch 9/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0297 - acc: 0.9911 - val_loss: 0.0886 - v
al_acc: 0.9767
Epoch 10/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0257 - acc: 0.9919 - val_loss: 0.0942 - v
al_acc: 0.9791
Epoch 11/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0221 - acc: 0.9934 - val_loss: 0.1040 - v
al_acc: 0.9755
Epoch 12/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0266 - acc: 0.9919 - val_loss: 0.1057 - v
al_acc: 0.9772
Epoch 13/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0183 - acc: 0.9945 - val_loss: 0.1061 - v
al_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0198 - acc: 0.9943 - val_loss: 0.0918 - v
al_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0176 - acc: 0.9946 - val_loss: 0.0898 - v
al_acc: 0.9796
Epoch 16/20
```

```
60000/60000 [==============================] - 5s 80us/step - loss: 0.0225 - acc: 0.9938 - val_loss: 0.0900 - v
al_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0173 - acc: 0.9950 - val_loss: 0.0850 - v
al_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0173 - acc: 0.9953 - val_loss: 0.0968 - v
al_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0143 - acc: 0.9961 - val_loss: 0.1019 - v
al_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0137 - acc: 0.9961 - val_loss: 0.0912 - v
al_acc: 0.9830
```

In [67]:
```
violin_plot(model_relu)
```

In [0]:
```
arch_3_model_1 = savetofile(history,'arch_3_model_1')
```

Model2: MLP + Relu + Adamoptimizer + Dropout

In [56]:

```
## Model 2: mlp_relu+adam_dropout

model_relu_drop = Sequential()

model_relu_drop.add(Dense(200,activation = 'relu',input_shape = (input_dim,), kernel_initializer = he_normal
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(300,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(400,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))

model_relu_drop.add(Dense(600,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_drop.add(Dropout(0.5))




model_relu_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_drop.summary())

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

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 200)               157000
_________________________________________________________________
dropout_11 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_42 (Dense)             (None, 300)               60300
_________________________________________________________________
dropout_12 (Dropout)         (None, 300)               0
_________________________________________________________________
```

```
dense_43 (Dense)             (None, 400)               120400
_________________________________________________________________
dropout_13 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_44 (Dense)             (None, 500)               200500
_________________________________________________________________
dropout_14 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_45 (Dense)             (None, 600)               300600
_________________________________________________________________
dropout_15 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                6010
=================================================================
Total params: 844,810
Trainable params: 844,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 177us/step - loss: 1.0938 - acc: 0.6332 - val_loss: 0.2784 - 
val_acc: 0.9222
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.4213 - acc: 0.8857 - val_loss: 0.2062 - v
al_acc: 0.9431
Epoch 3/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.3402 - acc: 0.9121 - val_loss: 0.1707 - v
al_acc: 0.9515
Epoch 4/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2892 - acc: 0.9238 - val_loss: 0.1533 - v
al_acc: 0.9591
Epoch 5/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2694 - acc: 0.9303 - val_loss: 0.1406 - v
al_acc: 0.9625
Epoch 6/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2491 - acc: 0.9364 - val_loss: 0.1439 - v
al_acc: 0.9627
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2335 - acc: 0.9420 - val_loss: 0.1402 - v
al_acc: 0.9624
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.2214 - acc: 0.9427 - val_loss: 0.1378 - v
```

```
al_acc: 0.9641
Epoch 9/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2143 - acc: 0.9452 - val_loss: 0.1324 - v
al_acc: 0.9649
Epoch 10/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2069 - acc: 0.9467 - val_loss: 0.1254 - v
al_acc: 0.9673
Epoch 11/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1988 - acc: 0.9491 - val_loss: 0.1189 - v
al_acc: 0.9684
Epoch 12/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1914 - acc: 0.9505 - val_loss: 0.1133 - v
al_acc: 0.9696
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1846 - acc: 0.9533 - val_loss: 0.1143 - v
al_acc: 0.9687
Epoch 14/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.1805 - acc: 0.9531 - val_loss: 0.1128 - v
al_acc: 0.9688
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1797 - acc: 0.9543 - val_loss: 0.1076 - v
al_acc: 0.9703
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1776 - acc: 0.9555 - val_loss: 0.1144 - v
al_acc: 0.9707
Epoch 17/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1689 - acc: 0.9573 - val_loss: 0.1003 - v
al_acc: 0.9738
Epoch 18/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1666 - acc: 0.9579 - val_loss: 0.1136 - v
al_acc: 0.9692
Epoch 19/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1636 - acc: 0.9586 - val_loss: 0.1026 - v
al_acc: 0.9730
Epoch 20/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1618 - acc: 0.9583 - val_loss: 0.1061 - v
al_acc: 0.9728
```

In [57]:
```
violin_plot(model_relu_drop)
```

In [0]:
```
arch_3_model_2 = savetofile(history,'arch_3_model_2')
```

Model3: MLP + Relu + Adamoptimizer + BatchNormalization

In [59]:

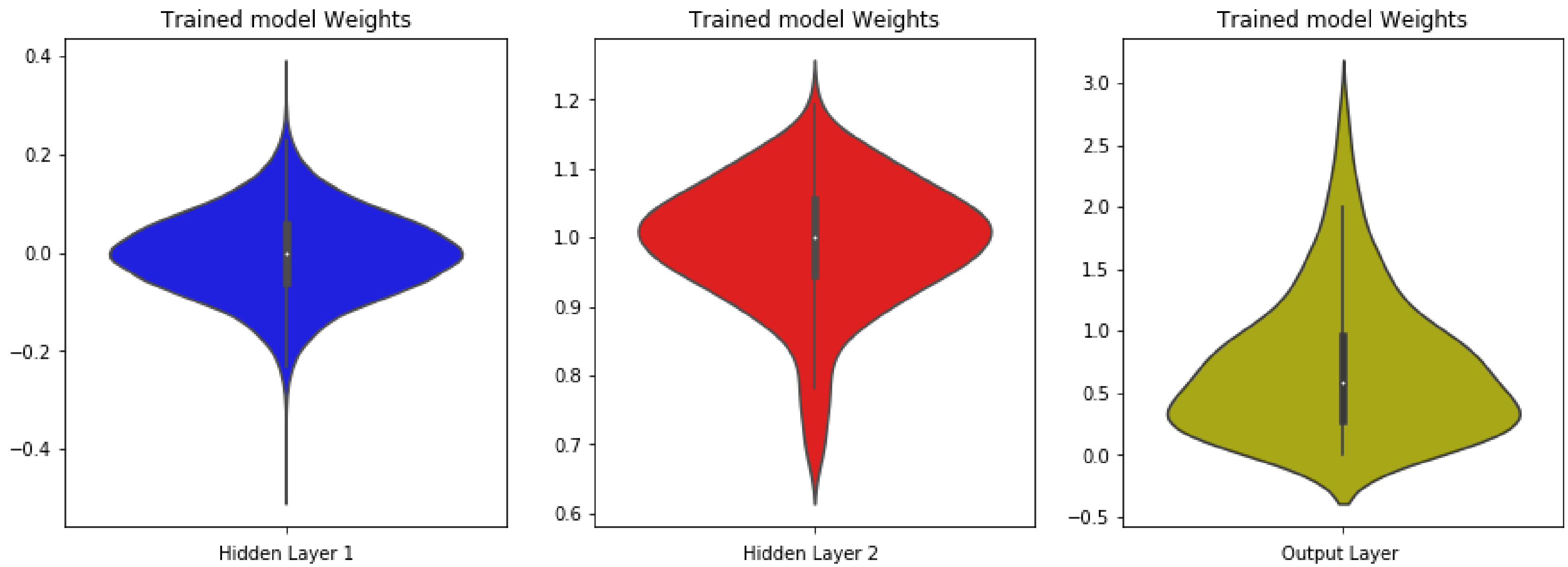
```
## Model 3: Mlp+relu+adam+batchnormalization
model_relu_batch = Sequential()

model_relu_batch.add(Dense(200,activation = 'relu',input_shape = (input_dim,),kernel_initializer = he_normal
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(300,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(400,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())

#model_relu_batch = Sequential()
model_relu_batch.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())

model_relu_batch.add(Dense(600,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch.add(BatchNormalization())



model_relu_batch.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch.summary())

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_47 (Dense)             (None, 200)               157000    
_________________________________________________________________
batch_normalization_11 (Batc (None, 200)               800       
_________________________________________________________________
dense_48 (Dense)             (None, 300)               60300     
_________________________________________________________________
batch_normalization_12 (Batc (None, 300)               1200      
_________________________________________________________________
dense_49 (Dense)             (None, 400)               120400    
```

```
_________________________________________________________________
batch_normalization_13 (Batc (None, 400)               1600      
_________________________________________________________________
dense_50 (Dense)             (None, 500)               200500    
_________________________________________________________________
batch_normalization_14 (Batc (None, 500)               2000      
_________________________________________________________________
dense_51 (Dense)             (None, 600)               300600    
_________________________________________________________________
batch_normalization_15 (Batc (None, 600)               2400      
_________________________________________________________________
dense_52 (Dense)             (None, 10)                6010      
=================================================================
Total params: 852,810
Trainable params: 848,810
Non-trainable params: 4,000
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 276us/step - loss: 0.2225 - acc: 0.9317 - val_loss: 0.1142 - val_acc: 0.9664
Epoch 2/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0886 - acc: 0.9721 - val_loss: 0.1059 - val_acc: 0.9676
Epoch 3/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0662 - acc: 0.9789 - val_loss: 0.1109 - val_acc: 0.9679
Epoch 4/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0518 - acc: 0.9835 - val_loss: 0.0925 - val_acc: 0.9715
Epoch 5/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0441 - acc: 0.9855 - val_loss: 0.0967 - val_acc: 0.9730
Epoch 6/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0401 - acc: 0.9872 - val_loss: 0.1137 - val_acc: 0.9689
Epoch 7/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0348 - acc: 0.9885 - val_loss: 0.0910 - val_acc: 0.9771
Epoch 8/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0325 - acc: 0.9890 - val_loss: 0.0939 - val_acc: 0.9747
```

```
Epoch 9/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0305 - acc: 0.9896 - val_loss: 0.0874 -
val_acc: 0.9782
Epoch 10/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0256 - acc: 0.9917 - val_loss: 0.1018 -
val_acc: 0.9758
Epoch 11/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0306 - acc: 0.9899 - val_loss: 0.0858 -
val_acc: 0.9781
Epoch 12/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0188 - acc: 0.9939 - val_loss: 0.0884 -
val_acc: 0.9793
Epoch 13/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0227 - acc: 0.9926 - val_loss: 0.0938 -
val_acc: 0.9776
Epoch 14/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0208 - acc: 0.9929 - val_loss: 0.1239 -
val_acc: 0.9699
Epoch 15/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0198 - acc: 0.9933 - val_loss: 0.0988 -
val_acc: 0.9774
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0229 - acc: 0.9927 - val_loss: 0.1024 -
val_acc: 0.9754
Epoch 17/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0148 - acc: 0.9952 - val_loss: 0.1036 -
val_acc: 0.9753
Epoch 18/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0145 - acc: 0.9954 - val_loss: 0.1119 -
val_acc: 0.9759
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0200 - acc: 0.9934 - val_loss: 0.0937 -
val_acc: 0.9763
Epoch 20/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0151 - acc: 0.9950 - val_loss: 0.0845 -
val_acc: 0.9808
```

In [60]:
```
violin_plot(model_relu_batch)
```



In [0]:
```
arch_3_model_3 = savetofile(history,'arch_3_modell_3')
```

Model4: MLP + Relu + Adamoptimizer + BtachNormalization + Dropout

In [62]:
```
## Model 4: mlp+ relu + adam+ batchNormalization + Dropout

model_relu_batch_drop = Sequential()
model_relu_batch_drop.add(Dense(200,activation = 'relu',input_shape = (input_dim,),kernel_initializer = he_n
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(300,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(400,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(500,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))

model_relu_batch_drop.add(Dense(600,activation = 'relu',kernel_initializer = he_normal(seed = None)))
model_relu_batch_drop.add(BatchNormalization())
model_relu_batch_drop.add(Dropout(0.5))


model_relu_batch_drop.add(Dense(output_dim,activation = 'softmax'))
print(model_relu_batch_drop.summary())

model_relu_batch_drop.compile(optimizer = 'adam',loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_relu_batch_drop.fit(x_train,Y_train,batch_size = batch_size,epochs = nb_epoch,verbose = 1,va

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 200)               157000    
_________________________________________________________________
batch_normalization_16 (Batc (None, 200)               800       
_________________________________________________________________
```

```
dropout_16 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_54 (Dense)             (None, 300)               60300
_________________________________________________________________
batch_normalization_17 (Batc (None, 300)               1200
_________________________________________________________________
dropout_17 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_55 (Dense)             (None, 400)               120400
_________________________________________________________________
batch_normalization_18 (Batc (None, 400)               1600
_________________________________________________________________
dropout_18 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_56 (Dense)             (None, 500)               200500
_________________________________________________________________
batch_normalization_19 (Batc (None, 500)               2000
_________________________________________________________________
dropout_19 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_57 (Dense)             (None, 600)               300600
_________________________________________________________________
batch_normalization_20 (Batc (None, 600)               2400
_________________________________________________________________
dropout_20 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_58 (Dense)             (None, 10)                6010
=================================================================
Total params: 852,810
Trainable params: 848,810
Non-trainable params: 4,000
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.9661 - acc: 0.7062 - val_loss: 0.2848 -
val_acc: 0.9172
Epoch 2/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.3900 - acc: 0.8819 - val_loss: 0.1996 -
val_acc: 0.9399
Epoch 3/20
60000/60000 [==============================] - 10s 175us/step - loss: 0.3009 - acc: 0.9115 - val_loss: 0.1443 -
val_acc: 0.9566
```

```
Epoch 4/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.2568 - acc: 0.9240 - val_loss: 0.1305 -
val_acc: 0.9607
Epoch 5/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.2274 - acc: 0.9333 - val_loss: 0.1209 -
val_acc: 0.9632
Epoch 6/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.2071 - acc: 0.9405 - val_loss: 0.1047 -
val_acc: 0.9690
Epoch 7/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1903 - acc: 0.9444 - val_loss: 0.1015 -
val_acc: 0.9707
Epoch 8/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1802 - acc: 0.9468 - val_loss: 0.1017 -
val_acc: 0.9687
Epoch 9/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1705 - acc: 0.9505 - val_loss: 0.0984 -
val_acc: 0.9729
Epoch 10/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1591 - acc: 0.9536 - val_loss: 0.0927 -
val_acc: 0.9739
Epoch 11/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1496 - acc: 0.9563 - val_loss: 0.0866 -
val_acc: 0.9741
Epoch 12/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.1500 - acc: 0.9566 - val_loss: 0.0890 -
val_acc: 0.9745
Epoch 13/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.1451 - acc: 0.9584 - val_loss: 0.0843 -
val_acc: 0.9755
Epoch 14/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.1392 - acc: 0.9593 - val_loss: 0.0841 -
val_acc: 0.9754
Epoch 15/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1321 - acc: 0.9614 - val_loss: 0.0821 -
val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1278 - acc: 0.9624 - val_loss: 0.0751 -
val_acc: 0.9784
Epoch 17/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1238 - acc: 0.9637 - val_loss: 0.0764 -
val_acc: 0.9776
Epoch 18/20
```

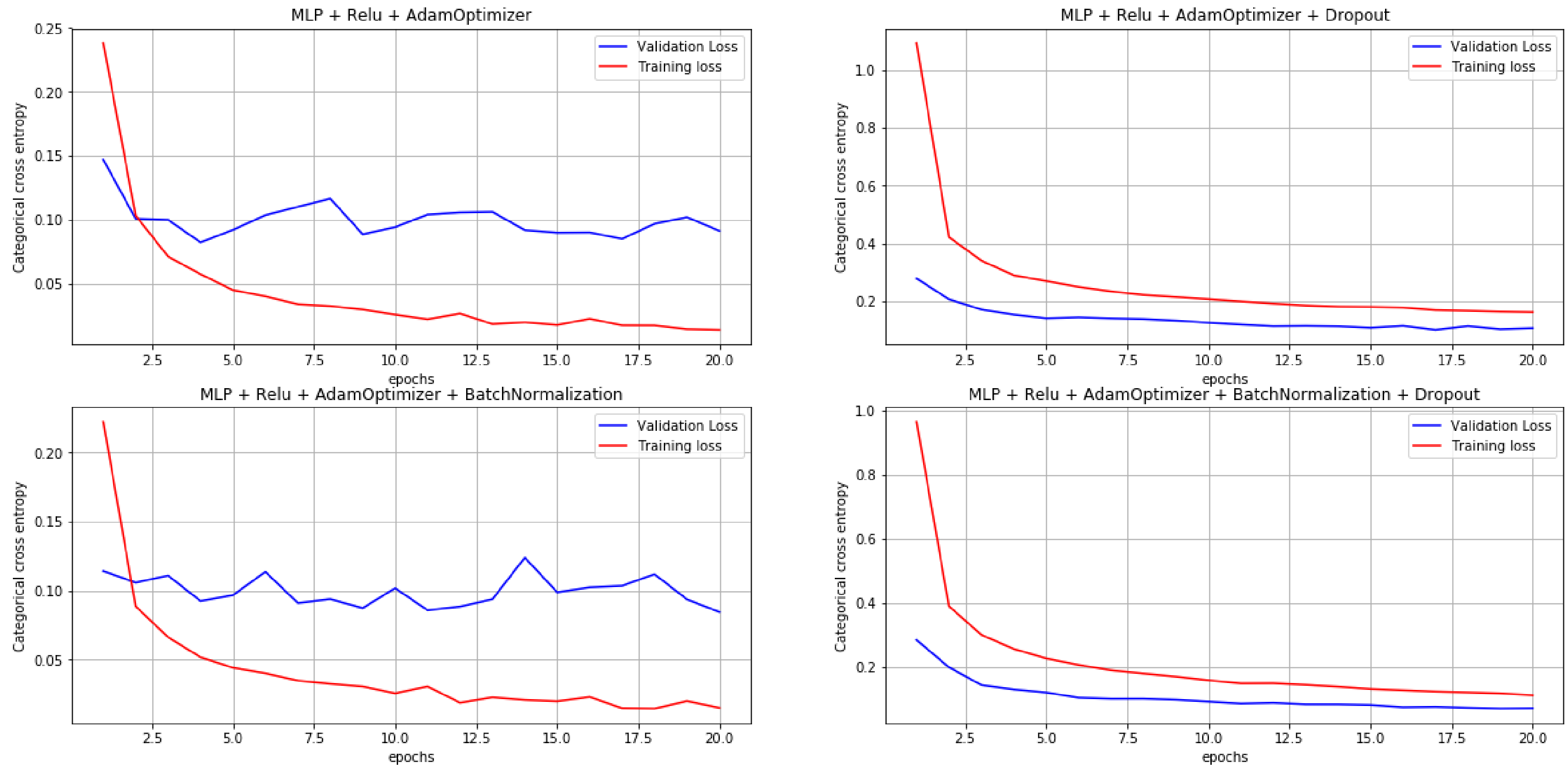
```
60000/60000 [==============================] - 10s 171us/step - loss: 0.1211 - acc: 0.9650 - val_loss: 0.0733 - 
val_acc: 0.9793
Epoch 19/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.1182 - acc: 0.9658 - val_loss: 0.0707 - 
val_acc: 0.9802
Epoch 20/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.1124 - acc: 0.9674 - val_loss: 0.0717 - 
val_acc: 0.9792
```

In [63]:
```
violin_plot(model_relu_batch_drop)
```

In [0]:
```
arch_3_model_4 = savetofile(history,'arch_3_model_4')
```

In [69]:
```
#plotting all the models

plt.figure(figsize = (20,20))
#plt.grid()
x = list(range(1,nb_epoch+1))

"""MODEL 1"""
plt.subplot(4,2,1)
plt.title('MLP + Relu + AdamOptimizer')
plt.grid()
plt.plot(x,openfromfile('arch_3_model_1').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_3_model_1').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 2"""

plt.subplot(4,2,2)
plt.title('MLP + Relu + AdamOptimizer + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_3_model_2').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_3_model_2').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()



"""MODEL 3"""
plt.subplot(4,2,3)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization')
plt.grid()
plt.plot(x,openfromfile('arch_3_modell_3').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_3_modell_3').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()


"""MODEL 4"""
```

```
plt.subplot(4,2,4)
plt.title('MLP + Relu + AdamOptimizer + BatchNormalization + Dropout')
plt.grid()
plt.plot(x,openfromfile('arch_3_model_4').history['val_loss'],color = 'b',label = 'Validation Loss')
plt.plot(x,openfromfile('arch_3_model_4').history['loss'],color = 'r',label = 'Training loss')
plt.xlabel('epochs')
plt.ylabel('Categorical cross entropy')
plt.legend()
plt.show()
```



PROCEDURE:

Following steps were followed:

- Data is imported and one hot encoded for each of the classes.

- Primarily we are considering 3 different neural network architecture here:
  - MLP with 2 hidden layers
  - MLP with 3 hidden layers
  - MLP with 5 hidden layers

- In each of the architecture we have tried 4 different for understanding the intricate working of the model and how to avoid overfitting and underfitting of the data.
  - Simple MLP
  - MLP + dropout with dropout rate = 0.5
  - MLP with Batch Normalization
  - MLP with Batch Normalization and Dropout(dropout rate = 0.5)
- After implementing each of the 4 techniques in each architecture we plot the violin plots to see the distribution of weights that we get after the implementation of optimization Algorithm.
- Finally we plot the graphs of loss vs epochs in each architecture to see how regularization is affected by adding different layers in the model.

Conclusion

In [70]:

```


from prettytable import PrettyTable

table_arch1 = PrettyTable()
models = ['MLP + relu + adamoptimizer','MLP + relu + adamoptimizer + dropout','MLP + relu + adamoptimizer +
tr_loss = ['0.005','0.0480','0.0074','0.0491']
tr_acc = ['99.82','98.47','99.72','98.38']
te_loss = ['0.101','0.0634','0.0768','0.054']
te_acc = ['98.02','98.28','98.16','98.31']

table_arch1.add_column('Model',models)
table_arch1.add_column('trainig loss',tr_loss)
table_arch1.add_column('Training Accuracy(%)',tr_acc)
table_arch1.add_column('Test loss',te_loss)
table_arch1.add_column('Test_Accuracy(%)',te_acc)
print('\t\t\t\t Architecture: Input(784)-Relu(512)-Relu(256)-SoftMax(10)')
print(table_arch1)
print('\n\n\n')

table_arch2 = PrettyTable()
models = ['MLP + relu + adamoptimizer','MLP + relu + adamoptimizer + dropout','MLP + relu + adamoptimizer +
tr_loss = ['0.0052','0.0507','0.0002','0.0462']
tr_acc = ['99.84','98.48','99.72','98.53']
te_loss = ['0.0935','0.0676','0.0885','0.0542']
te_acc = ['98.01','98.36','98.03','98.47']

table_arch2.add_column('Model',models)
table_arch2.add_column('trainig loss',tr_loss)
table_arch2.add_column('Training Accuracy(%)',tr_acc)
table_arch2.add_column('Test loss',te_loss)
table_arch2.add_column('Test_Accuracy(%)',te_acc)
print('\t\t\t\t Architecture: Input(784)-Relu(1000)-Relu(500)-Relu(250)-SoftMax(10)')
print(table_arch2)
print('\n\n\n')



table_arch3 = PrettyTable()
models = ['MLP + relu + adamoptimizer','MLP + relu + adamoptimizer + dropout','MLP + relu + adamoptimizer +
tr_loss = ['0.0137','0.1654','0.0141','0.114']
tr_acc = ['99.60','95.75','99.57','96.64']
```

```
te_loss = ['0.0935','0.1105','0.0847','0.0784']
te_acc = ['98.15','97.21','98.05','97.83']

table_arch3.add_column('Model',models)
table_arch3.add_column('trainig loss',tr_loss)
table_arch3.add_column('Training Accuracy(%)',tr_acc)
table_arch3.add_column('Test loss',te_loss)
table_arch3.add_column('Test_Accuracy(%)',te_acc)
print('\t\t\t\t Architecture: Input(784)-Relu(200)-Relu(300)-Relu(400)-Relu(500)-Relu(600)-SoftMax(10)')
print(table_arch3)
```

```
                                 Architecture: Input(784)-Relu(512)-Relu(256)-SoftMax(10)
+------------------------------------------------------------+--------------+----------------------+-----------+
------------------+
|                           Model                            | trainig loss | Training Accuracy(%) | Test loss |
Test_Accuracy(%) |
+------------------------------------------------------------+--------------+----------------------+-----------+
------------------+
|                 MLP + relu + adamoptimizer                 |    0.005     |        99.82         |   0.101   |
98.02        |
|            MLP + relu + adamoptimizer + dropout            |    0.0480    |        98.47         |   0.0634  |
98.28        |
|      MLP + relu + adamoptimizer + BatchNormalization       |    0.0074    |        99.72         |   0.0768  |
98.16        |
| MLP + relu + adamoptimizer + BatchNormalization + Dropout |    0.0491    |        98.38         |   0.054   |
98.31        |
+------------------------------------------------------------+--------------+----------------------+-----------+
------------------+


                                 Architecture: Input(784)-Relu(1000)-Relu(500)-Relu(250)-SoftMax(10)
+------------------------------------------------------------+--------------+----------------------+-----------+
------------------+
|                           Model                            | trainig loss | Training Accuracy(%) | Test loss |
Test_Accuracy(%) |
+------------------------------------------------------------+--------------+----------------------+-----------+
------------------+
|                 MLP + relu + adamoptimizer                 |    0.0052    |        99.84         |   0.0935  |
98.01        |
|            MLP + relu + adamoptimizer + dropout            |    0.0507    |        98.48         |   0.0676  |
```

				98.36
MLP + relu + adamoptimizer + BatchNormalization	0.0002	99.72	0.0885	98.03
MLP + relu + adamoptimizer + BatchNormalization + Dropout	0.0462	98.53	0.0542	98.47

Architecture: Input(784)-Relu(200)-Relu(300)-Relu(400)-Relu(500)-Relu(600)-SoftMax(10)

Model	trainig loss	Training Accuracy(%)	Test loss	Test_Accuracy(%)
MLP + relu + adamoptimizer	0.0137	99.60	0.0935	98.15
MLP + relu + adamoptimizer + dropout	0.1654	95.75	0.1105	97.21
MLP + relu + adamoptimizer + BatchNormalization	0.0141	99.57	0.0847	98.05
MLP + relu + adamoptimizer + BatchNormalization + Dropout	0.114	96.64	0.0784	97.83