MLP - Architecture using MNIST Dataset

1. Objective :

To build a different type of MLP Architecture.

2. Dataset :

In [0]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# References
# https://keras.io/datasets/

from keras.datasets import mnist
```

```
Using TensorFlow backend.
```

In [0]:

```
# mnist dataset for train and test

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [7]:

```
# Shape of the data

print("training features shape")
print("="*100)
print(x_train.shape)
print('training label shape')
print("="*100)
print(y_train.shape)
print("testing features shape")
print("="*100)
print(x_test.shape)
print("testing label shape")
print("="*100)
print(y_test.shape)
```

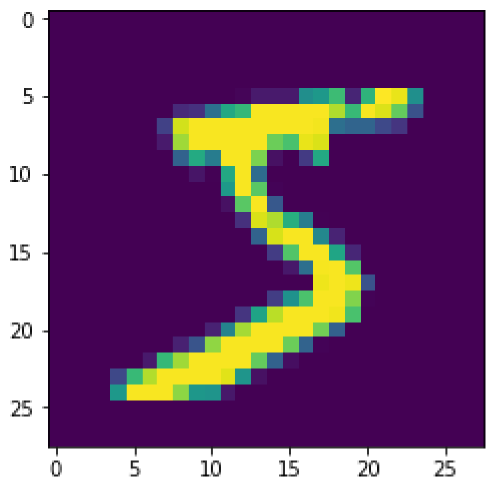
```
training features shape
=====================================================================
===============================
(60000, 28, 28)
training label shape
=====================================================================
===============================
(60000,)
testing features shape
=====================================================================
===============================
(10000, 28, 28)
testing label shape
=====================================================================
===============================
(10000,)
```

In [8]:

```
# Image data

plt.close()
plt.imshow(x_train[0])
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x7f1fed0d2668>
```



In [9]:

```
print("The Number is",y_train[0])
```

```
The Number is 5
```

In [10]:

```
# Reshaping Image data from 28*28 to 784 Vector

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])

print("training features shape")
print("="*100)
print(x_train.shape)

print("testing features shape")
print("="*100)
print(x_test.shape)
```

```
training features shape
====================================================================
================================
(60000, 784)
testing features shape
====================================================================
================================
(10000, 784)
```

In [11]:

```
# Labels in Mnist dataset
labels=set(y_train)

print("Mnist dataset labels")
print("="*100)
print(labels)
```

```
Mnist dataset labels
======================================================================
==============================
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [0]:

```
# Converting categorical Feature into one- hot encoding

# References
# https://keras.io/utils/

from keras.utils import np_utils
```

In [13]:

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

print("Before one-hot encoding")
print("="*100)
print(y_train.shape)
print("After one-hot encoding")
print("="*100)
print(Y_train.shape)
```

```
Before one-hot encoding
======================================================================
==============================
(60000,)
After one-hot encoding
======================================================================
==============================
(60000, 10)
```

3. Data Normalization :

In this Mnist Image dataset only contains pixels values. The pixel range is 0 to 255. So we apply the Data Normalization.

Data Normalization= $(X - X_a)/(X_b - X_a)$ Where X_a=minimum value of the data X_b=Maximum value of the data

In [0]:

```
X_train = x_train/255
X_test = x_test/255
```

In [15]:

```
print("Before Data Normalization")
print("="*100)
print(x_train[0][180:200])
print("After Data Normalization")
print("="*100)
print(X_train[0][180:200])
```

```
Before Data Normalization
=====================================================================
===============================
[170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0
0
   0   0]
After Data Normalization
=====================================================================
===============================
[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.        ]
```

4. MLP Architectures:

- 2 layers (Hidden) without BN and Dropout
- 3 layers (Hidden) without BN and Dropout
- 5 layers (Hidden) without BN and Dropout
- 3 layers (Hidden) with Batch Normalization(BN)
- 5 layers (Hidden) With Dropout
- 2 layers (Hidden) with BN and Dropout
- 3 layers (Hidden) with BN and Dropout
- 5 layers (Hidden) With BN and Dropout

4.1. Two Hidden Layer Architecture without BN and Dropout:

In [16]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
op = 10

print("Input layer size")
print("="*125)
print(ip)
print("Layer 1 size")
print("="*125)
print(layer_1)
print("Layer 2 size")
print("="*125)
print(layer_2)
print("Output layer size")
print("="*125)
print(op)
```

```
Input layer size
=====================================================================
========================================================
784
Layer 1 size
=====================================================================
========================================================
512
Layer 2 size
=====================================================================
========================================================
256
Output layer size
=====================================================================
========================================================
10
```

4.1.1 Sequence Model :

In [0]:

```
# References
# https://keras.io/getting-started/sequential-model-guide/
# https://keras.io/initializers/
# http://www.malinc.se/math/latex/basiccodeen.php
# https://keras.io/layers/core/#dense
# https://keras.io/layers/normalization/
# https://keras.io/layers/core/#dropout

from keras.models import Sequential
from keras.layers import Dense,Activation,BatchNormalization,Dropout
from keras.initializers import RandomNormal
```

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# https://github.com/h5py/h5py/issues/961

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [21]:

```
# Model Summary

model.summary()
```

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

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.1.2 Model Training:

In [23]:

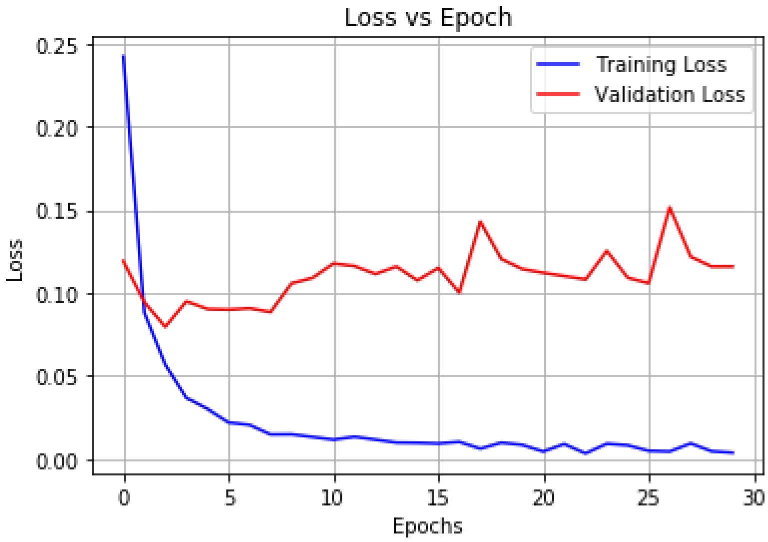
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensor
flow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.op
s.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 8s 157us/step - loss:
0.2422 - acc: 0.9282 - val_loss: 0.1192 - val_acc: 0.9647
Epoch 2/30
48000/48000 [==============================] - 7s 149us/step - loss:
0.0882 - acc: 0.9725 - val_loss: 0.0944 - val_acc: 0.9713
Epoch 3/30
48000/48000 [==============================] - 7s 151us/step - loss:
0.0567 - acc: 0.9821 - val_loss: 0.0795 - val_acc: 0.9773
Epoch 4/30
48000/48000 [==============================] - 7s 152us/step - loss:
0.0367 - acc: 0.9879 - val_loss: 0.0949 - val_acc: 0.9732
Epoch 5/30
48000/48000 [==============================] - 7s 151us/step - loss:
0.0300 - acc: 0.9903 - val_loss: 0.0903 - val_acc: 0.9752
Epoch 6/30
48000/48000 [==============================] - 7s 153us/step - loss:
0.0220 - acc: 0.9927 - val_loss: 0.0899 - val_acc: 0.9772
Epoch 7/30
48000/48000 [==============================] - 7s 153us/step - loss:
0.0205 - acc: 0.9930 - val_loss: 0.0907 - val_acc: 0.9772
Epoch 8/30
48000/48000 [==============================] - 7s 151us/step - loss:
0.0148 - acc: 0.9949 - val_loss: 0.0884 - val_acc: 0.9780
Epoch 9/30
48000/48000 [==============================] - 7s 151us/step - loss:
0.0149 - acc: 0.9950 - val_loss: 0.1058 - val_acc: 0.9756
Epoch 10/30
48000/48000 [==============================] - 7s 152us/step - loss:
0.0133 - acc: 0.9954 - val_loss: 0.1091 - val_acc: 0.9753
Epoch 11/30
48000/48000 [==============================] - 7s 154us/step - loss:
0.0117 - acc: 0.9963 - val_loss: 0.1177 - val_acc: 0.9758
Epoch 12/30
48000/48000 [==============================] - 7s 155us/step - loss:
0.0134 - acc: 0.9956 - val_loss: 0.1162 - val_acc: 0.9760
Epoch 13/30
48000/48000 [==============================] - 7s 153us/step - loss:
0.0117 - acc: 0.9959 - val_loss: 0.1114 - val_acc: 0.9759
Epoch 14/30
48000/48000 [==============================] - 7s 153us/step - loss:
0.0099 - acc: 0.9966 - val_loss: 0.1159 - val_acc: 0.9766
Epoch 15/30
48000/48000 [==============================] - 7s 152us/step - loss:
0.0097 - acc: 0.9973 - val_loss: 0.1076 - val_acc: 0.9789
Epoch 16/30
48000/48000 [==============================] - 7s 154us/step - loss:
0.0093 - acc: 0.9971 - val_loss: 0.1150 - val_acc: 0.9777
Epoch 17/30
```

```
48000/48000 [==============================] - 7s 154us/step - loss: 
0.0103 - acc: 0.9964 - val_loss: 0.1001 - val_acc: 0.9810
Epoch 18/30
48000/48000 [==============================] - 7s 152us/step - loss: 
0.0062 - acc: 0.9984 - val_loss: 0.1429 - val_acc: 0.9715
Epoch 19/30
48000/48000 [==============================] - 7s 152us/step - loss: 
0.0098 - acc: 0.9967 - val_loss: 0.1205 - val_acc: 0.9755
Epoch 20/30
48000/48000 [==============================] - 7s 153us/step - loss: 
0.0085 - acc: 0.9972 - val_loss: 0.1144 - val_acc: 0.9771
Epoch 21/30
48000/48000 [==============================] - 7s 154us/step - loss: 
0.0045 - acc: 0.9987 - val_loss: 0.1121 - val_acc: 0.9797
Epoch 22/30
48000/48000 [==============================] - 8s 157us/step - loss: 
0.0090 - acc: 0.9972 - val_loss: 0.1102 - val_acc: 0.9806
Epoch 23/30
48000/48000 [==============================] - 7s 154us/step - loss: 
0.0035 - acc: 0.9989 - val_loss: 0.1082 - val_acc: 0.9822
Epoch 24/30
48000/48000 [==============================] - 7s 150us/step - loss: 
0.0092 - acc: 0.9972 - val_loss: 0.1254 - val_acc: 0.9778
Epoch 25/30
48000/48000 [==============================] - 7s 151us/step - loss: 
0.0083 - acc: 0.9976 - val_loss: 0.1092 - val_acc: 0.9803
Epoch 26/30
48000/48000 [==============================] - 7s 155us/step - loss: 
0.0049 - acc: 0.9985 - val_loss: 0.1058 - val_acc: 0.9814
Epoch 27/30
48000/48000 [==============================] - 7s 153us/step - loss: 
0.0045 - acc: 0.9984 - val_loss: 0.1516 - val_acc: 0.9761
Epoch 28/30
48000/48000 [==============================] - 7s 155us/step - loss: 
0.0094 - acc: 0.9971 - val_loss: 0.1219 - val_acc: 0.9790
Epoch 29/30
48000/48000 [==============================] - 7s 152us/step - loss: 
0.0047 - acc: 0.9986 - val_loss: 0.1159 - val_acc: 0.9804
Epoch 30/30
48000/48000 [==============================] - 7s 150us/step - loss: 
0.0038 - acc: 0.9989 - val_loss: 0.1159 - val_acc: 0.9813
```

In [24]:

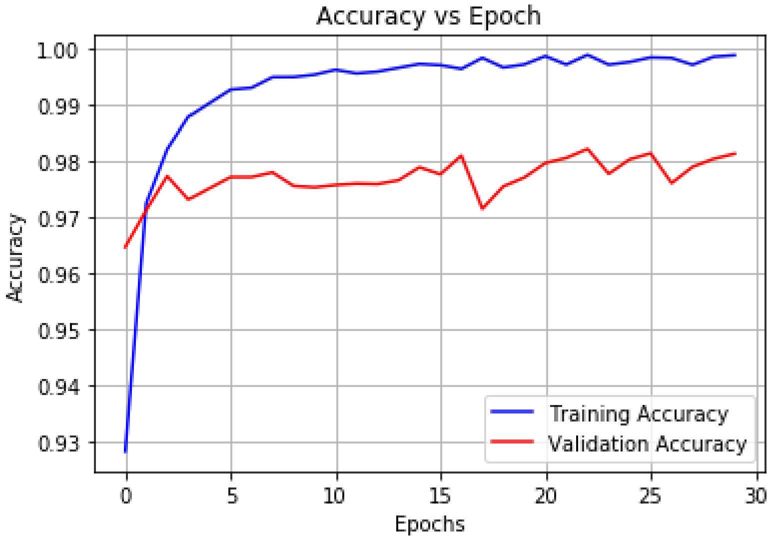
```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



Observation: Model becomes overfitting.

In [25]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [26]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_4', 'trainable': True, 'batch_input_shape': (None, 78
4), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'dense_5', 'trainable': True, 'units': 256, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bias
_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regular
izer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_6', 'trainable': True, 'units': 10, 'activation': 'so
ftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.09, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

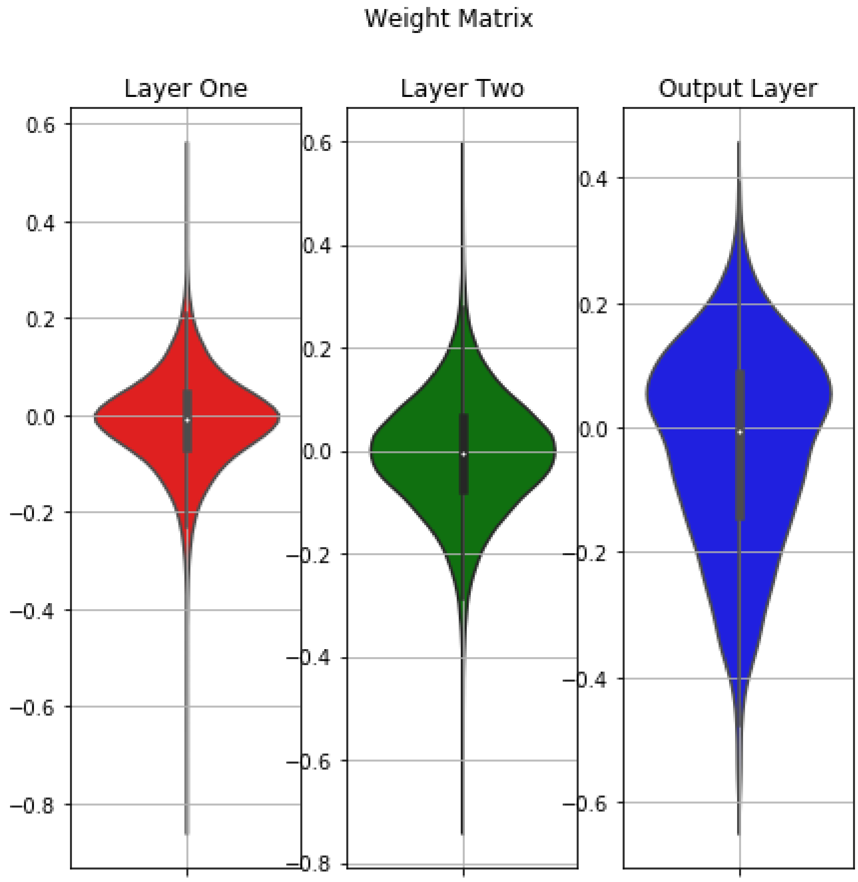
In [28]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[1].get_weights()[0]
output_weights = model.layers[2].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*125)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*125)
print(layer2_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*125)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
========================================================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
========================================================
(512, 256)
 Output Layer Weight Matrix Shape
=====================================================================
========================================================
(256, 10)
```

In [29]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 3, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,3,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,3,3)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.1.3Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [31]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
=====================================================================
===============================
0.10787136471846893
 Evaluate Accuracy
=====================================================================
===============================
0.9813
```

4.1.4 Model Predict:

In [32]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[0].reshape(1,784)
test_data.shape
```

Out[32]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [34]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[0].argmax())

print("The Actual Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
==============================
7
The Actual Label of the predicted data
======================================================================
==============================
7
```

4.1.5 Observation:

In [1]:

```
from prettytable import PrettyTable
```

In [2]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Batch Size","Epochs"]
b.add_row(["Xavier",0.5,100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([2,0.0038,0.1159 ,0.1078,0.9989,0.9813,0.9813])
print(b)
print(a)
```

```
+----------------+-----------+------------+--------+
| Initialization | Optimizer | Batch Size | Epochs |
+----------------+-----------+------------+--------+
|     Xavier     |    0.5    |    100     |   30   |
+----------------+-----------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   2   |   0.0038   |  0.1159  |   0.1078  |   0.9989  |  0.9813 |
0.9813  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.2. Three Hidden Layer Architecture without BN and Dropout:

In [39]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Layer 3 size
=====================================================================
===============================
128
Output layer size
=====================================================================
===============================
10
```

4.2.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [42]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_12 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_13 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290      
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.2.2 Model Training

In [44]:

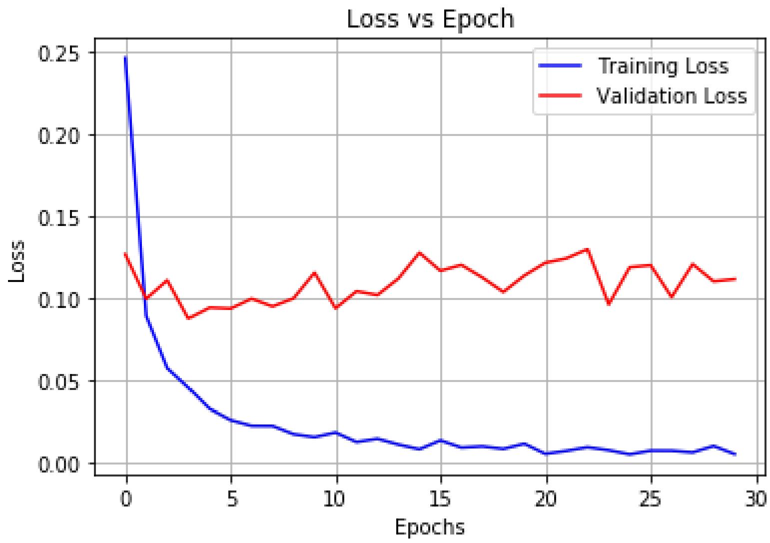
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.2466 - acc: 0.9271 - val_loss: 0.1271 - val_acc: 0.9603
Epoch 2/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0895 - acc: 0.9727 - val_loss: 0.1000 - val_acc: 0.9713
Epoch 3/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0577 - acc: 0.9817 - val_loss: 0.1113 - val_acc: 0.9686
Epoch 4/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0458 - acc: 0.9850 - val_loss: 0.0880 - val_acc: 0.9756
Epoch 5/30
48000/48000 [==============================] - 8s 160us/step - loss:
0.0330 - acc: 0.9895 - val_loss: 0.0946 - val_acc: 0.9721
Epoch 6/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0262 - acc: 0.9916 - val_loss: 0.0940 - val_acc: 0.9753
Epoch 7/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0227 - acc: 0.9925 - val_loss: 0.1001 - val_acc: 0.9756
Epoch 8/30
48000/48000 [==============================] - 8s 157us/step - loss:
0.0226 - acc: 0.9929 - val_loss: 0.0953 - val_acc: 0.9752
Epoch 9/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0176 - acc: 0.9942 - val_loss: 0.1003 - val_acc: 0.9759
Epoch 10/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0158 - acc: 0.9945 - val_loss: 0.1159 - val_acc: 0.9744
Epoch 11/30
48000/48000 [==============================] - 8s 160us/step - loss:
0.0186 - acc: 0.9937 - val_loss: 0.0942 - val_acc: 0.9781
Epoch 12/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0129 - acc: 0.9958 - val_loss: 0.1046 - val_acc: 0.9777
Epoch 13/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0149 - acc: 0.9948 - val_loss: 0.1023 - val_acc: 0.9778
Epoch 14/30
48000/48000 [==============================] - 8s 157us/step - loss:
0.0113 - acc: 0.9964 - val_loss: 0.1124 - val_acc: 0.9767
Epoch 15/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0085 - acc: 0.9970 - val_loss: 0.1281 - val_acc: 0.9742
Epoch 16/30
48000/48000 [==============================] - 8s 160us/step - loss:
0.0139 - acc: 0.9953 - val_loss: 0.1171 - val_acc: 0.9772
Epoch 17/30
48000/48000 [==============================] - 8s 157us/step - loss:
0.0095 - acc: 0.9970 - val_loss: 0.1207 - val_acc: 0.9773
Epoch 18/30
48000/48000 [==============================] - 7s 156us/step - loss:
0.0102 - acc: 0.9969 - val_loss: 0.1130 - val_acc: 0.9786
```

```
Epoch 19/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0087 - acc: 0.9974 - val_loss: 0.1041 - val_acc: 0.9789
Epoch 20/30
48000/48000 [==============================] - 8s 161us/step - loss:
0.0118 - acc: 0.9966 - val_loss: 0.1141 - val_acc: 0.9781
Epoch 21/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0057 - acc: 0.9984 - val_loss: 0.1220 - val_acc: 0.9767
Epoch 22/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0075 - acc: 0.9980 - val_loss: 0.1246 - val_acc: 0.9753
Epoch 23/30
48000/48000 [==============================] - 8s 160us/step - loss:
0.0097 - acc: 0.9969 - val_loss: 0.1302 - val_acc: 0.9756
Epoch 24/30
48000/48000 [==============================] - 8s 157us/step - loss:
0.0079 - acc: 0.9975 - val_loss: 0.0965 - val_acc: 0.9816
Epoch 25/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0054 - acc: 0.9984 - val_loss: 0.1192 - val_acc: 0.9788
Epoch 26/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0076 - acc: 0.9978 - val_loss: 0.1205 - val_acc: 0.9768
Epoch 27/30
48000/48000 [==============================] - 8s 160us/step - loss:
0.0076 - acc: 0.9977 - val_loss: 0.1010 - val_acc: 0.9806
Epoch 28/30
48000/48000 [==============================] - 8s 159us/step - loss:
0.0065 - acc: 0.9980 - val_loss: 0.1212 - val_acc: 0.9790
Epoch 29/30
48000/48000 [==============================] - 8s 164us/step - loss:
0.0105 - acc: 0.9973 - val_loss: 0.1106 - val_acc: 0.9788
Epoch 30/30
48000/48000 [==============================] - 8s 158us/step - loss:
0.0055 - acc: 0.9985 - val_loss: 0.1120 - val_acc: 0.9787
```

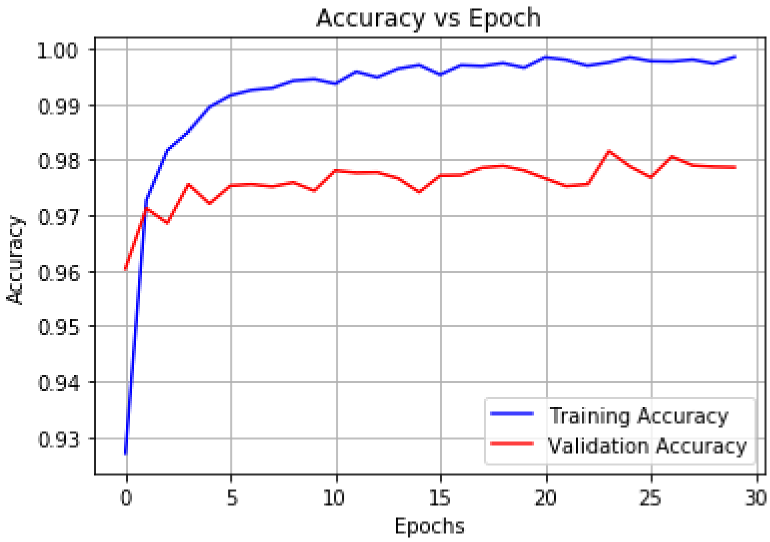
In [45]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [46]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [47]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_11', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'dense_12', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_13', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_14', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.12, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

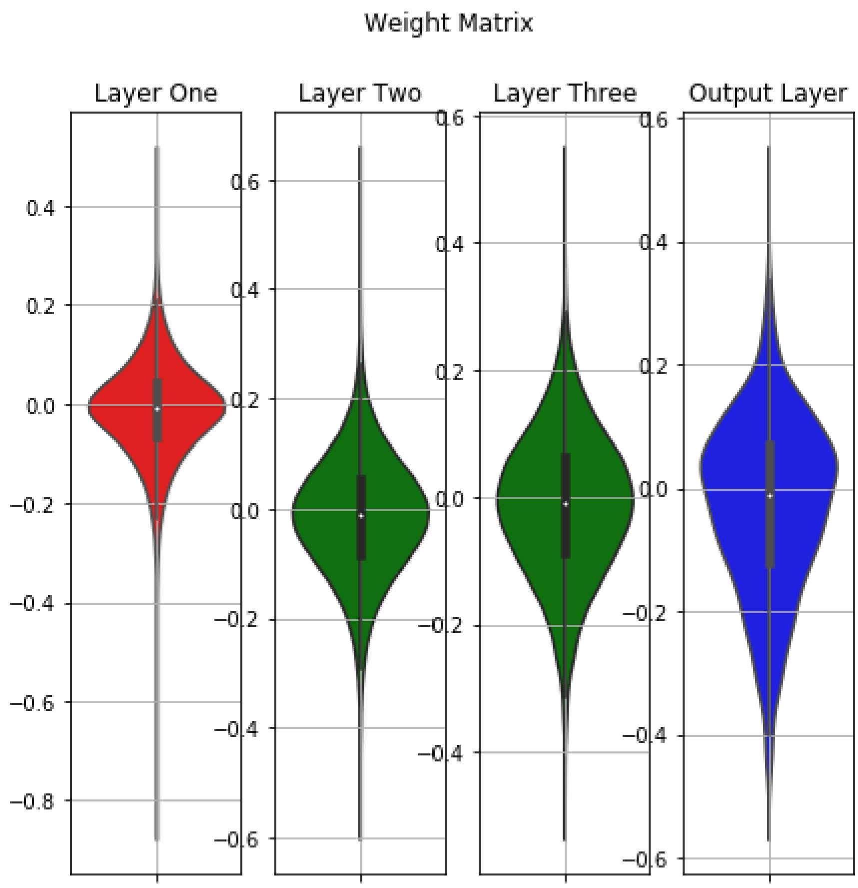
In [49]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[1].get_weights()[0]
layer3_weights = model.layers[2].get_weights()[0]
output_weights = model.layers[3].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
======================================================================
==============================
(784, 512)
 Layer Two Weight Matrix Shape
======================================================================
==============================
(512, 256)
 Layer Three Weight Matrix Shape
======================================================================
==============================
(256, 128)
 Output Layer Weight Matrix Shape
======================================================================
==============================
(128, 10)
```

In [50]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 4, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,4,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,4,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='g')
plt.grid()

plt.subplot(1,4,4)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.2.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [52]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
=====================================================================
===============================
0.08808950365521413
 Evaluate Accuracy
=====================================================================
===============================
0.9828
```

4.2.4 Model Predict:

In [0]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[50].reshape(1,784)
test_data.shape
```

Out[56]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [0]:

```
print("The Actual Label of the predicted data")
print("="*125)
print(Y_test[50].argmax())

print("The Actual Label of the predicted data")
print("="*125)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
=====================================================================
========================================================
6
The Actual Label of the predicted data
=====================================================================
========================================================
6
```

4.2.5 Observation:

In [3]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([3,0.0055 ,0.1120 ,0.0880,0.9985,0.9787,0.9828])
print(b)
print(a)
```

```
+----------------+-----------+------------+--------+
| Initialization | Optimizer | Batch Size | Epochs |
+----------------+-----------+------------+--------+
|     Xavier     |    Adam   |    100     |   30   |
+----------------+-----------+------------+--------+
+-------+------------+----------+-----------+-----------+----------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+----------+--
--------+
|   3   |   0.0055   |  0.112   |   0.088   |   0.9985  |  0.9787 |
0.9828  |
+-------+------------+----------+-----------+-----------+----------+--
--------+
```

4.3. Five Hidden Layer Architecture without BN and Dropout:

In [54]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
layer_4 = 64
layer_5 = 32
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Layer 4 size")
print("="*100)
print(layer_4)
print("Layer 5 size")
print("="*100)
print(layer_5)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Layer 3 size
=====================================================================
===============================
128
Layer 4 size
=====================================================================
===============================
64
Layer 5 size
=====================================================================
===============================
32
Output layer size
=====================================================================
```

```
================================
10
```

4.3.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Layer 4

model.add(Dense(layer_4,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Layer 5

model.add(Dense(layer_5,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [57]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_21 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_22 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_23 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_24 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_25 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_26 (Dense)             (None, 10)                330       
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.3.2 Model Training:

In [59]:

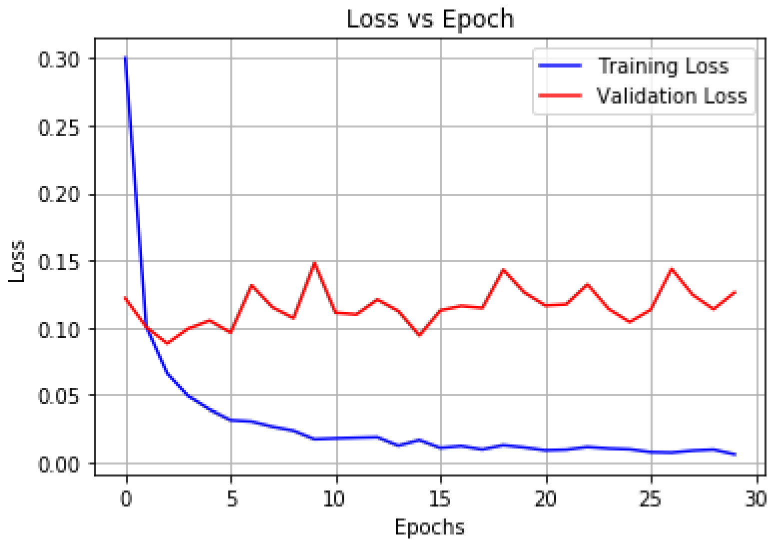
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 9s 181us/step - loss:
0.3001 - acc: 0.9098 - val_loss: 0.1219 - val_acc: 0.9632
Epoch 2/30
48000/48000 [==============================] - 8s 163us/step - loss:
0.1008 - acc: 0.9698 - val_loss: 0.1005 - val_acc: 0.9700
Epoch 3/30
48000/48000 [==============================] - 8s 164us/step - loss:
0.0660 - acc: 0.9798 - val_loss: 0.0882 - val_acc: 0.9722
Epoch 4/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0491 - acc: 0.9851 - val_loss: 0.0993 - val_acc: 0.9728
Epoch 5/30
48000/48000 [==============================] - 8s 166us/step - loss:
0.0393 - acc: 0.9875 - val_loss: 0.1053 - val_acc: 0.9723
Epoch 6/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0315 - acc: 0.9899 - val_loss: 0.0963 - val_acc: 0.9743
Epoch 7/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0305 - acc: 0.9901 - val_loss: 0.1315 - val_acc: 0.9693
Epoch 8/30
48000/48000 [==============================] - 8s 163us/step - loss:
0.0266 - acc: 0.9911 - val_loss: 0.1151 - val_acc: 0.9710
Epoch 9/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0237 - acc: 0.9925 - val_loss: 0.1070 - val_acc: 0.9740
Epoch 10/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.0174 - acc: 0.9943 - val_loss: 0.1481 - val_acc: 0.9690
Epoch 11/30
48000/48000 [==============================] - 8s 164us/step - loss:
0.0180 - acc: 0.9945 - val_loss: 0.1111 - val_acc: 0.9752
Epoch 12/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0185 - acc: 0.9942 - val_loss: 0.1098 - val_acc: 0.9759
Epoch 13/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.0189 - acc: 0.9941 - val_loss: 0.1208 - val_acc: 0.9751
Epoch 14/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0126 - acc: 0.9961 - val_loss: 0.1123 - val_acc: 0.9766
Epoch 15/30
48000/48000 [==============================] - 8s 166us/step - loss:
0.0167 - acc: 0.9954 - val_loss: 0.0942 - val_acc: 0.9787
Epoch 16/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.0109 - acc: 0.9965 - val_loss: 0.1128 - val_acc: 0.9759
Epoch 17/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0123 - acc: 0.9963 - val_loss: 0.1161 - val_acc: 0.9755
Epoch 18/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0098 - acc: 0.9969 - val_loss: 0.1146 - val_acc: 0.9766
```

```
Epoch 19/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.0130 - acc: 0.9963 - val_loss: 0.1430 - val_acc: 0.9713
Epoch 20/30
48000/48000 [==============================] - 8s 172us/step - loss:
0.0113 - acc: 0.9967 - val_loss: 0.1262 - val_acc: 0.9752
Epoch 21/30
48000/48000 [==============================] - 8s 175us/step - loss:
0.0091 - acc: 0.9973 - val_loss: 0.1164 - val_acc: 0.9773
Epoch 22/30
48000/48000 [==============================] - 8s 170us/step - loss:
0.0096 - acc: 0.9972 - val_loss: 0.1173 - val_acc: 0.9774
Epoch 23/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0116 - acc: 0.9965 - val_loss: 0.1320 - val_acc: 0.9741
Epoch 24/30
48000/48000 [==============================] - 8s 164us/step - loss:
0.0105 - acc: 0.9971 - val_loss: 0.1138 - val_acc: 0.9773
Epoch 25/30
48000/48000 [==============================] - 8s 165us/step - loss:
0.0099 - acc: 0.9973 - val_loss: 0.1041 - val_acc: 0.9802
Epoch 26/30
48000/48000 [==============================] - 8s 166us/step - loss:
0.0078 - acc: 0.9980 - val_loss: 0.1132 - val_acc: 0.9795
Epoch 27/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0074 - acc: 0.9980 - val_loss: 0.1437 - val_acc: 0.9734
Epoch 28/30
48000/48000 [==============================] - 8s 170us/step - loss:
0.0089 - acc: 0.9974 - val_loss: 0.1245 - val_acc: 0.9790
Epoch 29/30
48000/48000 [==============================] - 8s 168us/step - loss:
0.0097 - acc: 0.9972 - val_loss: 0.1136 - val_acc: 0.9794
Epoch 30/30
48000/48000 [==============================] - 8s 167us/step - loss:
0.0061 - acc: 0.9983 - val_loss: 0.1261 - val_acc: 0.9772
```

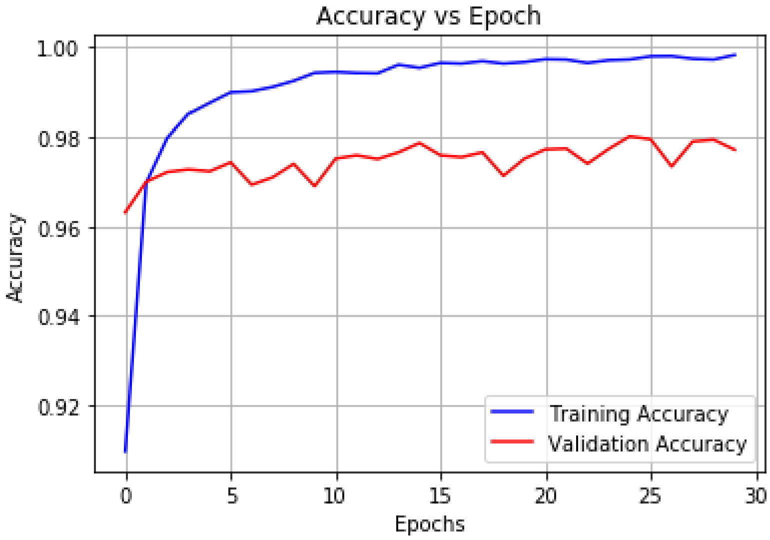
In [60]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [61]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [62]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_21', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'dense_22', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_23', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_24', 'trainable': True, 'units': 64, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.1, 'seed': None}}, 'bias_
initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regulari
zer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_25', 'trainable': True, 'units': 32, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.14, 'seed': None}}, 'bias
_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regular
izer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dense_26', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.22, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

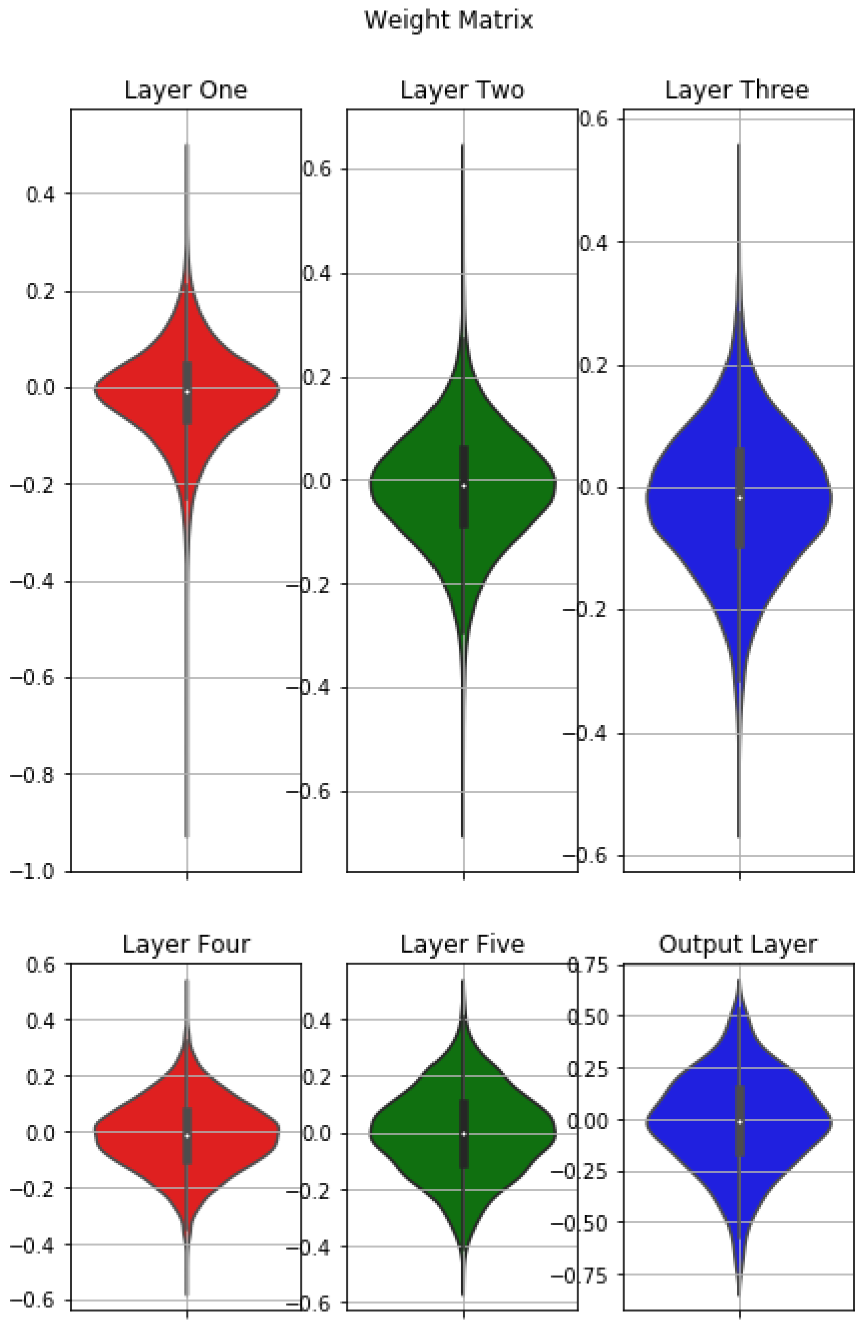
In [64]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[1].get_weights()[0]
layer3_weights = model.layers[2].get_weights()[0]
layer4_weights = model.layers[3].get_weights()[0]
layer5_weights = model.layers[4].get_weights()[0]
output_weights = model.layers[5].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Layer Four Weight Matrix Shape")
print("="*100)
print(layer4_weights.shape)
print(" Layer Five Weight Matrix Shape")
print("="*100)
print(layer5_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Layer Three Weight Matrix Shape
=====================================================================
===============================
(256, 128)
 Layer Four Weight Matrix Shape
=====================================================================
===============================
(128, 64)
 Layer Five Weight Matrix Shape
=====================================================================
===============================
(64, 32)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(32, 10)
```

In [65]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(1,figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 3, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,3,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,3,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='b')
plt.grid()

fig = plt.figure(2,figsize=(7,7))

plt.subplot(2,3,1)
plt.title("Layer Four")
sns.violinplot(y=layer4_weights,color='r')
plt.grid()

plt.subplot(2,3,2)
plt.title("Layer Five")
sns.violinplot(y=layer5_weights,color='g')
plt.grid()

plt.subplot(2,3,3)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.3.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [67]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
======================================================================
===============================
0.10328517330627829
 Evaluate Accuracy
======================================================================
===============================
0.9816
```

4.3.4 Model Predict:

In [68]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[500].reshape(1,784)
test_data.shape
```

Out[68]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [70]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[500].argmax())

print("The Actual Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
3
The Actual Label of the predicted data
======================================================================
===============================
3
```

4.3.5 Observation:

In [4]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([5,0.0061 ,0.1261 ,0.1032,0.9983,0.9772,0.9816])
print(b)
print(a)
```

```
+----------------+-----------+------------+--------+
| Initialization | Optimizer | Batch Size | Epochs |
+----------------+-----------+------------+--------+
|     Xavier     |    Adam   |    100     |   30   |
+----------------+-----------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   5   |   0.0061   |  0.1261  |   0.1032  |   0.9983  |  0.9772 |
0.9816  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.4. Three Hidden Layer Architecture with BN:

In [72]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Layer 3 size
=====================================================================
===============================
128
Output layer size
=====================================================================
===============================
10
```

4.4.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Batch Normalization Layer

model.add(BatchNormalization())

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [74]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_27 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048      
_________________________________________________________________
dense_28 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024      
_________________________________________________________________
dense_29 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_11 (Batc (None, 128)               512       
_________________________________________________________________
dense_30 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.4.2 Model Training

In [76]:

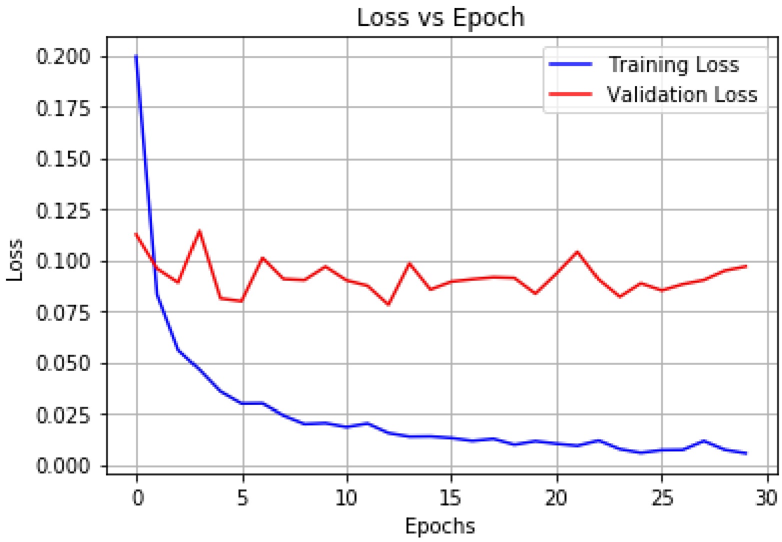
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 11s 233us/step - loss:
0.1993 - acc: 0.9394 - val_loss: 0.1124 - val_acc: 0.9656
Epoch 2/30
48000/48000 [==============================] - 10s 203us/step - loss:
0.0830 - acc: 0.9745 - val_loss: 0.0957 - val_acc: 0.9719
Epoch 3/30
48000/48000 [==============================] - 10s 204us/step - loss:
0.0558 - acc: 0.9823 - val_loss: 0.0889 - val_acc: 0.9726
Epoch 4/30
48000/48000 [==============================] - 10s 204us/step - loss:
0.0463 - acc: 0.9854 - val_loss: 0.1142 - val_acc: 0.9664
Epoch 5/30
48000/48000 [==============================] - 10s 207us/step - loss:
0.0356 - acc: 0.9885 - val_loss: 0.0811 - val_acc: 0.9768
Epoch 6/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0297 - acc: 0.9903 - val_loss: 0.0798 - val_acc: 0.9769
Epoch 7/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0298 - acc: 0.9899 - val_loss: 0.1010 - val_acc: 0.9713
Epoch 8/30
48000/48000 [==============================] - 10s 203us/step - loss:
0.0240 - acc: 0.9923 - val_loss: 0.0906 - val_acc: 0.9773
Epoch 9/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.0198 - acc: 0.9934 - val_loss: 0.0901 - val_acc: 0.9761
Epoch 10/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0203 - acc: 0.9927 - val_loss: 0.0967 - val_acc: 0.9758
Epoch 11/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0183 - acc: 0.9941 - val_loss: 0.0900 - val_acc: 0.9766
Epoch 12/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.0201 - acc: 0.9928 - val_loss: 0.0873 - val_acc: 0.9768
Epoch 13/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.0154 - acc: 0.9948 - val_loss: 0.0780 - val_acc: 0.9795
Epoch 14/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0137 - acc: 0.9952 - val_loss: 0.0982 - val_acc: 0.9782
Epoch 15/30
48000/48000 [==============================] - 10s 209us/step - loss:
0.0138 - acc: 0.9953 - val_loss: 0.0854 - val_acc: 0.9797
Epoch 16/30
48000/48000 [==============================] - 10s 208us/step - loss:
0.0130 - acc: 0.9959 - val_loss: 0.0893 - val_acc: 0.9786
Epoch 17/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0116 - acc: 0.9960 - val_loss: 0.0905 - val_acc: 0.9783
Epoch 18/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0126 - acc: 0.9959 - val_loss: 0.0915 - val_acc: 0.9782
```

```
Epoch 19/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0097 - acc: 0.9966 - val_loss: 0.0911 - val_acc: 0.9804
Epoch 20/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.0115 - acc: 0.9962 - val_loss: 0.0834 - val_acc: 0.9810
Epoch 21/30
48000/48000 [==============================] - 10s 204us/step - loss:
0.0102 - acc: 0.9966 - val_loss: 0.0932 - val_acc: 0.9800
Epoch 22/30
48000/48000 [==============================] - 10s 205us/step - loss:
0.0092 - acc: 0.9970 - val_loss: 0.1039 - val_acc: 0.9775
Epoch 23/30
48000/48000 [==============================] - 10s 203us/step - loss:
0.0118 - acc: 0.9961 - val_loss: 0.0904 - val_acc: 0.9788
Epoch 24/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0076 - acc: 0.9977 - val_loss: 0.0819 - val_acc: 0.9807
Epoch 25/30
48000/48000 [==============================] - 10s 199us/step - loss:
0.0058 - acc: 0.9983 - val_loss: 0.0885 - val_acc: 0.9821
Epoch 26/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0071 - acc: 0.9978 - val_loss: 0.0850 - val_acc: 0.9823
Epoch 27/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0072 - acc: 0.9976 - val_loss: 0.0881 - val_acc: 0.9809
Epoch 28/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.0116 - acc: 0.9963 - val_loss: 0.0901 - val_acc: 0.9794
Epoch 29/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.0073 - acc: 0.9977 - val_loss: 0.0947 - val_acc: 0.9788
Epoch 30/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.0055 - acc: 0.9982 - val_loss: 0.0968 - val_acc: 0.9806
```

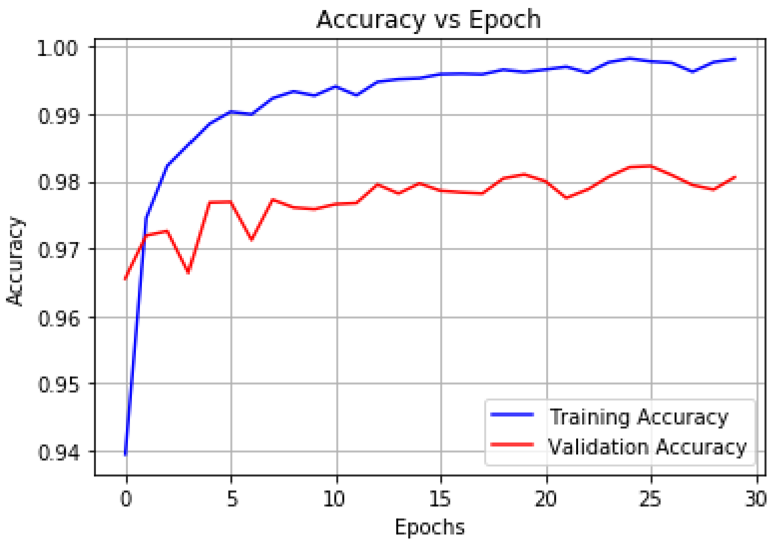
In [77]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [78]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [79]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_27', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'batch_normalization_9', 'trainable': True, 'axis': -1, 'mom
entum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta_
initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initializ
er': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initializer':
{'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializer':
{'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None, 'gamm
a_regularizer': None, 'beta_constraint': None, 'gamma_constraint': No
ne}
{'name': 'dense_28', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_10', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dense_29', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_11', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dense_30', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.12, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

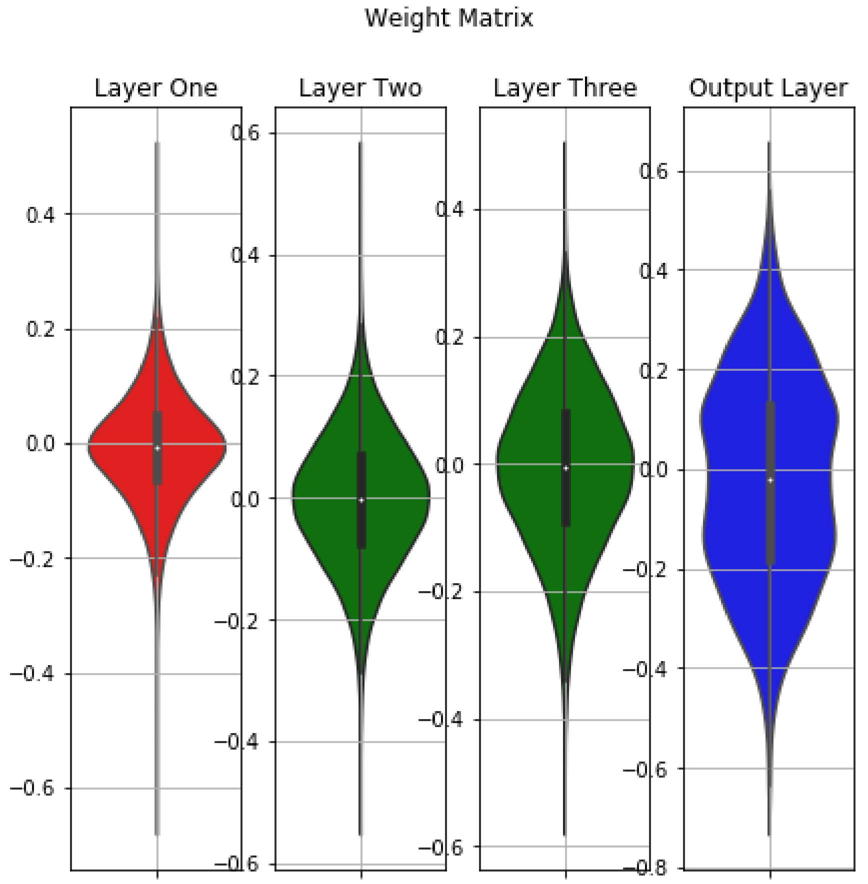
In [81]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[2].get_weights()[0]
layer3_weights = model.layers[4].get_weights()[0]
output_weights = model.layers[6].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Layer Three Weight Matrix Shape
=====================================================================
===============================
(256, 128)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(128, 10)
```

In [82]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 4, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,4,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,4,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='g')
plt.grid()

plt.subplot(1,4,4)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.4.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [84]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
=====================================================================
===============================
0.087774215674301
 Evaluate Accuracy
=====================================================================
===============================
0.9805
```

4.4.4 Model Predict:

In [85]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[50].reshape(1,784)
test_data.shape
```

Out[85]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [87]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[50].argmax())

print("The Actual Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
6
The Actual Label of the predicted data
======================================================================
===============================
6
```

4.4.5 Observation:

In [5]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([3,0.0055 ,0.0968  ,0.0877,0.9982,0.9806,0.9805])
print(b)
print(a)
```

```
+----------------+-----------+------------+--------+
| Initialization | Optimizer | Batch Size | Epochs |
+----------------+-----------+------------+--------+
|     Xavier     |    Adam   |    100     |   30   |
+----------------+-----------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   3   |   0.0055   |  0.0968  |   0.0877  |   0.9982  |  0.9806 |
0.9805  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.5. Five Hidden Layer Architecture with Dropout:

In [89]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
layer_4 = 64
layer_5 = 32
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Layer 4 size")
print("="*100)
print(layer_4)
print("Layer 5 size")
print("="*100)
print(layer_5)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Layer 3 size
=====================================================================
===============================
128
Layer 4 size
=====================================================================
===============================
64
Layer 5 size
=====================================================================
===============================
32
Output layer size
=====================================================================
```

```
================================
10
```

4.5.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Dropout Layer

model.add(Dropout(0.5))

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Dropout Layer

model.add(Dropout(0.5))

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Dropout Layer

model.add(Dropout(0.5))

# Layer 4

model.add(Dense(layer_4,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Dropout Layer

model.add(Dropout(0.5))

# Layer 5

model.add(Dense(layer_5,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Dropout Layer

model.add(Dropout(0.5))


# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [91]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_31 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_32 (Dense)             (None, 256)               131328    
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_33 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_35 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_13 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_36 (Dense)             (None, 10)                330       
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.5.2 Model Training:

In [93]:

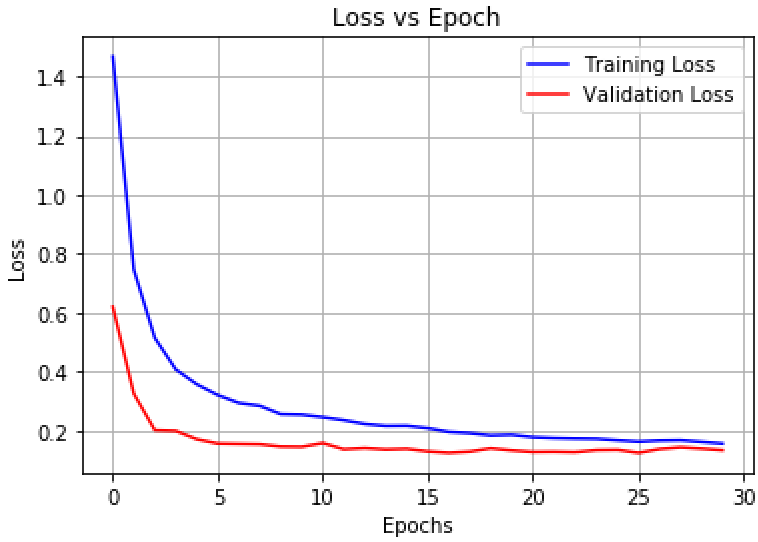
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 11s 224us/step - loss:
1.4668 - acc: 0.4502 - val_loss: 0.6198 - val_acc: 0.7566
Epoch 2/30
48000/48000 [==============================] - 10s 199us/step - loss:
0.7464 - acc: 0.7584 - val_loss: 0.3248 - val_acc: 0.9353
Epoch 3/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.5142 - acc: 0.8629 - val_loss: 0.2009 - val_acc: 0.9542
Epoch 4/30
48000/48000 [==============================] - 10s 198us/step - loss:
0.4072 - acc: 0.8979 - val_loss: 0.1984 - val_acc: 0.9518
Epoch 5/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.3567 - acc: 0.9103 - val_loss: 0.1701 - val_acc: 0.9605
Epoch 6/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.3201 - acc: 0.9210 - val_loss: 0.1553 - val_acc: 0.9656
Epoch 7/30
48000/48000 [==============================] - 10s 198us/step - loss:
0.2933 - acc: 0.9292 - val_loss: 0.1542 - val_acc: 0.9660
Epoch 8/30
48000/48000 [==============================] - 10s 199us/step - loss:
0.2837 - acc: 0.9336 - val_loss: 0.1531 - val_acc: 0.9663
Epoch 9/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.2553 - acc: 0.9387 - val_loss: 0.1451 - val_acc: 0.9692
Epoch 10/30
48000/48000 [==============================] - 10s 202us/step - loss:
0.2535 - acc: 0.9395 - val_loss: 0.1442 - val_acc: 0.9694
Epoch 11/30
48000/48000 [==============================] - 10s 203us/step - loss:
0.2451 - acc: 0.9424 - val_loss: 0.1571 - val_acc: 0.9687
Epoch 12/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.2348 - acc: 0.9439 - val_loss: 0.1360 - val_acc: 0.9709
Epoch 13/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.2219 - acc: 0.9484 - val_loss: 0.1399 - val_acc: 0.9703
Epoch 14/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.2153 - acc: 0.9487 - val_loss: 0.1354 - val_acc: 0.9732
Epoch 15/30
48000/48000 [==============================] - 10s 198us/step - loss:
0.2158 - acc: 0.9490 - val_loss: 0.1375 - val_acc: 0.9721
Epoch 16/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.2077 - acc: 0.9494 - val_loss: 0.1287 - val_acc: 0.9734
Epoch 17/30
48000/48000 [==============================] - 10s 199us/step - loss:
0.1950 - acc: 0.9525 - val_loss: 0.1239 - val_acc: 0.9746
Epoch 18/30
48000/48000 [==============================] - 9s 198us/step - loss:
0.1909 - acc: 0.9543 - val_loss: 0.1280 - val_acc: 0.9750
```

```
Epoch 19/30
48000/48000 [==============================] - 9s 198us/step - loss:
0.1832 - acc: 0.9562 - val_loss: 0.1392 - val_acc: 0.9721
Epoch 20/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.1853 - acc: 0.9548 - val_loss: 0.1321 - val_acc: 0.9761
Epoch 21/30
48000/48000 [==============================] - 9s 193us/step - loss:
0.1771 - acc: 0.9567 - val_loss: 0.1269 - val_acc: 0.9740
Epoch 22/30
48000/48000 [==============================] - 9s 193us/step - loss:
0.1740 - acc: 0.9596 - val_loss: 0.1275 - val_acc: 0.9743
Epoch 23/30
48000/48000 [==============================] - 10s 203us/step - loss:
0.1724 - acc: 0.9590 - val_loss: 0.1262 - val_acc: 0.9763
Epoch 24/30
48000/48000 [==============================] - 10s 200us/step - loss:
0.1714 - acc: 0.9583 - val_loss: 0.1332 - val_acc: 0.9753
Epoch 25/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.1664 - acc: 0.9591 - val_loss: 0.1340 - val_acc: 0.9756
Epoch 26/30
48000/48000 [==============================] - 9s 192us/step - loss:
0.1620 - acc: 0.9602 - val_loss: 0.1238 - val_acc: 0.9764
Epoch 27/30
48000/48000 [==============================] - 10s 198us/step - loss:
0.1657 - acc: 0.9605 - val_loss: 0.1368 - val_acc: 0.9760
Epoch 28/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.1667 - acc: 0.9599 - val_loss: 0.1427 - val_acc: 0.9760
Epoch 29/30
48000/48000 [==============================] - 9s 193us/step - loss:
0.1611 - acc: 0.9615 - val_loss: 0.1381 - val_acc: 0.9756
Epoch 30/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.1554 - acc: 0.9617 - val_loss: 0.1328 - val_acc: 0.9758
```

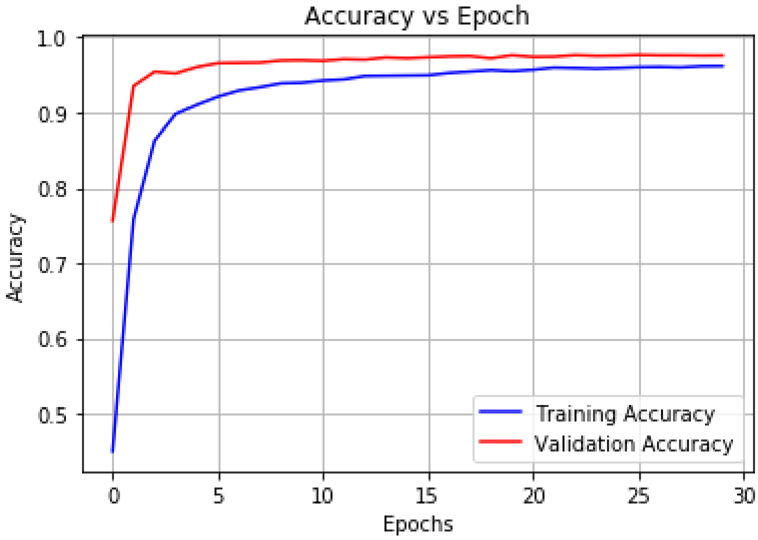
In [94]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [95]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [96]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_31', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'dropout_9', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_32', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dropout_10', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_33', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dropout_11', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_34', 'trainable': True, 'units': 64, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.1, 'seed': None}}, 'bias_
initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regulari
zer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dropout_12', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_35', 'trainable': True, 'units': 32, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.14, 'seed': None}}, 'bias
_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regular
izer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'dropout_13', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_36', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.22, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

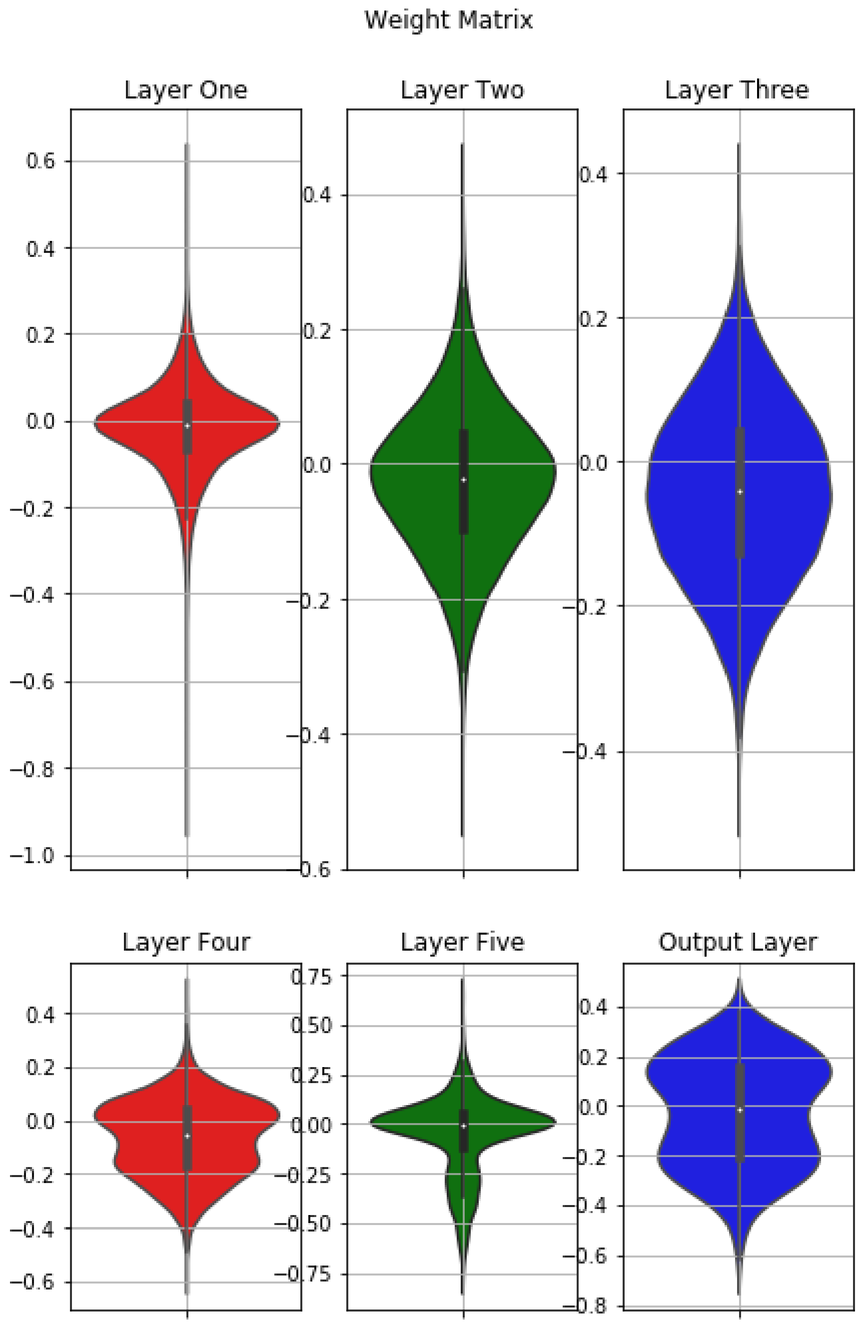
In [98]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[2].get_weights()[0]
layer3_weights = model.layers[4].get_weights()[0]
layer4_weights = model.layers[6].get_weights()[0]
layer5_weights = model.layers[8].get_weights()[0]
output_weights = model.layers[10].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Layer Four Weight Matrix Shape")
print("="*100)
print(layer4_weights.shape)
print(" Layer Five Weight Matrix Shape")
print("="*100)
print(layer5_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Layer Three Weight Matrix Shape
=====================================================================
===============================
(256, 128)
 Layer Four Weight Matrix Shape
=====================================================================
===============================
(128, 64)
 Layer Five Weight Matrix Shape
=====================================================================
===============================
(64, 32)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(32, 10)
```

In [99]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(1,figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 3, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,3,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,3,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='b')
plt.grid()

fig = plt.figure(2,figsize=(7,7))

plt.subplot(2,3,1)
plt.title("Layer Four")
sns.violinplot(y=layer4_weights,color='r')
plt.grid()

plt.subplot(2,3,2)
plt.title("Layer Five")
sns.violinplot(y=layer5_weights,color='g')
plt.grid()

plt.subplot(2,3,3)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.5.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [102]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
======================================================================
================================
0.13766393155145343
 Evaluate Accuracy
======================================================================
================================
0.9762
```

4.5.4 Model Predict:

In [103]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[5000].reshape(1,784)
test_data.shape
```

Out[103]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [107]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[5000].argmax())

print("The predicted Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
3
The predicted Label of the predicted data
======================================================================
===============================
3
```

4.5.5 Observation:

In [133]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Dropout rate","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",0.5,100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([5,0.1554 ,0.1328  ,0.1376,0.9617,0.9758,0.9762])
print(b)
print(a)
```

```
+----------------+-----------+--------------+------------+--------+
| Initialization | Optimizer | Dropout rate | Batch Size | Epochs |
+----------------+-----------+--------------+------------+--------+
|     Xavier     |    Adam   |     0.5      |    100     |   30   |
+----------------+-----------+--------------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   5   |   0.1554   |  0.1328  |   0.1376  |   0.9617  |  0.9758 |
0.9762  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.6. Two Hidden Layer Architecture with BN and Dropout:

In [109]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Output layer size
=====================================================================
===============================
10
```

4.6.1 Sequence Model :

In [0]:

```
# References
# https://keras.io/getting-started/sequential-model-guide/
# https://keras.io/initializers/
# http://www.malinc.se/math/latex/basiccodeen.php
# https://keras.io/layers/core/#dense
# https://keras.io/layers/normalization/
# https://keras.io/layers/core/#dropout

from keras.models import Sequential
from keras.layers import Dense,Activation,BatchNormalization,Dropout
from keras.initializers import RandomNormal
```

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [112]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_37 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_14 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_38 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_13 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_39 (Dense)             (None, 10)                2570      
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.6.2 Model Training:

In [114]:

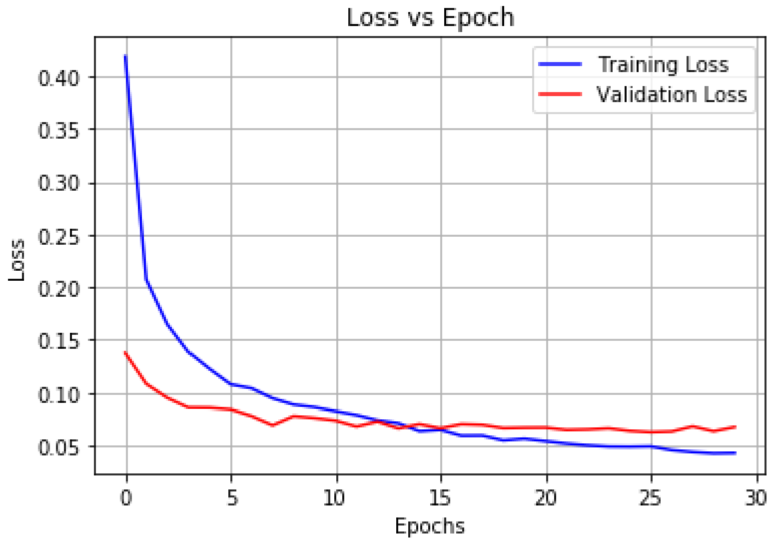
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.4187 - acc: 0.8743 - val_loss: 0.1374 - val_acc: 0.9593
Epoch 2/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.2074 - acc: 0.9369 - val_loss: 0.1082 - val_acc: 0.9678
Epoch 3/30
48000/48000 [==============================] - 9s 192us/step - loss:
0.1647 - acc: 0.9493 - val_loss: 0.0950 - val_acc: 0.9703
Epoch 4/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.1384 - acc: 0.9569 - val_loss: 0.0858 - val_acc: 0.9746
Epoch 5/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.1224 - acc: 0.9620 - val_loss: 0.0856 - val_acc: 0.9729
Epoch 6/30
48000/48000 [==============================] - 9s 193us/step - loss:
0.1077 - acc: 0.9665 - val_loss: 0.0840 - val_acc: 0.9744
Epoch 7/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.1040 - acc: 0.9673 - val_loss: 0.0777 - val_acc: 0.9765
Epoch 8/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.0945 - acc: 0.9708 - val_loss: 0.0690 - val_acc: 0.9794
Epoch 9/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.0885 - acc: 0.9718 - val_loss: 0.0776 - val_acc: 0.9761
Epoch 10/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.0861 - acc: 0.9723 - val_loss: 0.0758 - val_acc: 0.9783
Epoch 11/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.0822 - acc: 0.9740 - val_loss: 0.0735 - val_acc: 0.9784
Epoch 12/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.0786 - acc: 0.9750 - val_loss: 0.0679 - val_acc: 0.9808
Epoch 13/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.0738 - acc: 0.9767 - val_loss: 0.0726 - val_acc: 0.9790
Epoch 14/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.0710 - acc: 0.9766 - val_loss: 0.0662 - val_acc: 0.9812
Epoch 15/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.0636 - acc: 0.9786 - val_loss: 0.0702 - val_acc: 0.9801
Epoch 16/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.0648 - acc: 0.9795 - val_loss: 0.0663 - val_acc: 0.9811
Epoch 17/30
48000/48000 [==============================] - 9s 192us/step - loss:
0.0593 - acc: 0.9807 - val_loss: 0.0702 - val_acc: 0.9805
Epoch 18/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.0594 - acc: 0.9804 - val_loss: 0.0696 - val_acc: 0.9815
```

```
Epoch 19/30
48000/48000 [==============================] - 9s 194us/step - loss:
0.0548 - acc: 0.9824 - val_loss: 0.0665 - val_acc: 0.9822
Epoch 20/30
48000/48000 [==============================] - 10s 201us/step - loss:
0.0564 - acc: 0.9815 - val_loss: 0.0669 - val_acc: 0.9823
Epoch 21/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.0541 - acc: 0.9820 - val_loss: 0.0670 - val_acc: 0.9827
Epoch 22/30
48000/48000 [==============================] - 9s 192us/step - loss:
0.0518 - acc: 0.9829 - val_loss: 0.0648 - val_acc: 0.9815
Epoch 23/30
48000/48000 [==============================] - 9s 193us/step - loss:
0.0501 - acc: 0.9834 - val_loss: 0.0652 - val_acc: 0.9818
Epoch 24/30
48000/48000 [==============================] - 9s 192us/step - loss:
0.0490 - acc: 0.9843 - val_loss: 0.0663 - val_acc: 0.9816
Epoch 25/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.0488 - acc: 0.9840 - val_loss: 0.0638 - val_acc: 0.9831
Epoch 26/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.0491 - acc: 0.9841 - val_loss: 0.0626 - val_acc: 0.9836
Epoch 27/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.0456 - acc: 0.9851 - val_loss: 0.0633 - val_acc: 0.9828
Epoch 28/30
48000/48000 [==============================] - 9s 196us/step - loss:
0.0437 - acc: 0.9853 - val_loss: 0.0681 - val_acc: 0.9810
Epoch 29/30
48000/48000 [==============================] - 9s 197us/step - loss:
0.0426 - acc: 0.9859 - val_loss: 0.0635 - val_acc: 0.9833
Epoch 30/30
48000/48000 [==============================] - 9s 195us/step - loss:
0.0428 - acc: 0.9850 - val_loss: 0.0675 - val_acc: 0.9816
```

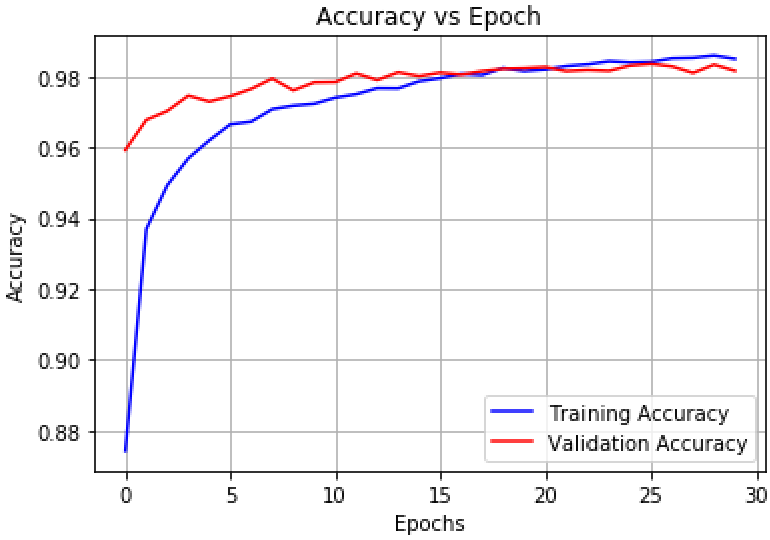
In [115]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [116]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [117]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_37', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'batch_normalization_12', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_14', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_38', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_13', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_15', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_39', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.09, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

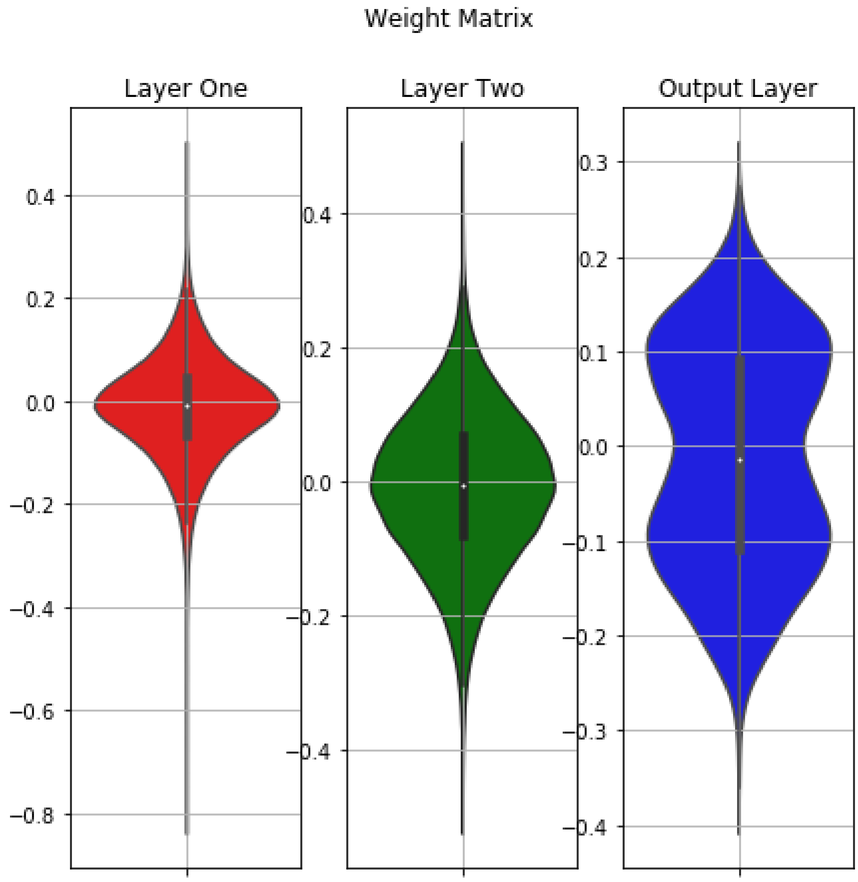
In [118]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[3].get_weights()[0]
output_weights = model.layers[6].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(256, 10)
```

In [119]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 3, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,3,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,3,3)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.6.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [121]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
=====================================================================
===============================
0.05716658136337064
 Evaluate Accuracy
=====================================================================
===============================
0.9841
```

4.6.4 Model Predict:

In [122]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[0].reshape(1,784)
test_data.shape
```

Out[122]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [124]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[0].argmax())

print("The Predicted Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
7
The Predicted Label of the predicted data
======================================================================
===============================
7
```

4.6.5 Observation:

In [126]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Dropout rate","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",0.5,100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([2,0.0428 ,0.0675  ,0.0571,0.9850,0.9816,0.9841])
print(b)
print(a)
```

```
+----------------+-----------+--------------+------------+--------+
| Initialization | Optimizer | Dropout rate | Batch Size | Epochs |
+----------------+-----------+--------------+------------+--------+
|     Xavier     |    Adam   |     0.5      |    100     |   30   |
+----------------+-----------+--------------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   2   |   0.0428   |  0.0675  |   0.0571  |   0.985   |  0.9816 |
0.9841  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.7. Three Hidden Layer Architecture with BN and Dropout:

In [127]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
=====================================================================
===============================
784
Layer 1 size
=====================================================================
===============================
512
Layer 2 size
=====================================================================
===============================
256
Layer 3 size
=====================================================================
===============================
128
Output layer size
=====================================================================
===============================
10
```

4.7.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))


# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [129]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_40 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_41 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_15 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_16 (Batc (None, 128)               512       
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_43 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.7.2 Model Training

In [131]:

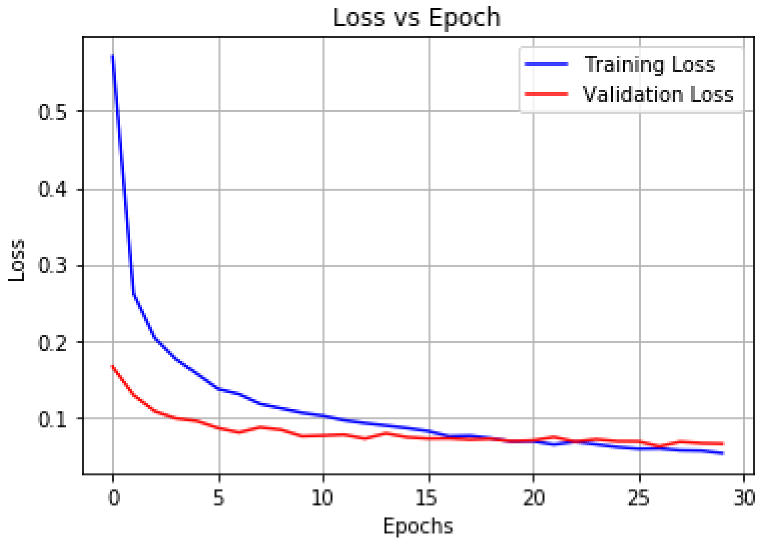
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 13s 264us/step - loss:
0.5715 - acc: 0.8273 - val_loss: 0.1654 - val_acc: 0.9502
Epoch 2/30
48000/48000 [==============================] - 10s 216us/step - loss:
0.2613 - acc: 0.9239 - val_loss: 0.1282 - val_acc: 0.9619
Epoch 3/30
48000/48000 [==============================] - 10s 213us/step - loss:
0.2034 - acc: 0.9383 - val_loss: 0.1072 - val_acc: 0.9680
Epoch 4/30
48000/48000 [==============================] - 10s 211us/step - loss:
0.1757 - acc: 0.9477 - val_loss: 0.0981 - val_acc: 0.9719
Epoch 5/30
48000/48000 [==============================] - 10s 210us/step - loss:
0.1561 - acc: 0.9544 - val_loss: 0.0945 - val_acc: 0.9726
Epoch 6/30
48000/48000 [==============================] - 10s 215us/step - loss:
0.1363 - acc: 0.9598 - val_loss: 0.0853 - val_acc: 0.9751
Epoch 7/30
48000/48000 [==============================] - 10s 211us/step - loss:
0.1294 - acc: 0.9625 - val_loss: 0.0794 - val_acc: 0.9767
Epoch 8/30
48000/48000 [==============================] - 10s 210us/step - loss:
0.1167 - acc: 0.9648 - val_loss: 0.0862 - val_acc: 0.9753
Epoch 9/30
48000/48000 [==============================] - 10s 214us/step - loss:
0.1108 - acc: 0.9666 - val_loss: 0.0830 - val_acc: 0.9759
Epoch 10/30
48000/48000 [==============================] - 10s 215us/step - loss:
0.1052 - acc: 0.9682 - val_loss: 0.0745 - val_acc: 0.9783
Epoch 11/30
48000/48000 [==============================] - 10s 213us/step - loss:
0.1013 - acc: 0.9691 - val_loss: 0.0754 - val_acc: 0.9778
Epoch 12/30
48000/48000 [==============================] - 10s 212us/step - loss:
0.0955 - acc: 0.9726 - val_loss: 0.0765 - val_acc: 0.9773
Epoch 13/30
48000/48000 [==============================] - 10s 213us/step - loss:
0.0917 - acc: 0.9728 - val_loss: 0.0713 - val_acc: 0.9789
Epoch 14/30
48000/48000 [==============================] - 11s 222us/step - loss:
0.0885 - acc: 0.9735 - val_loss: 0.0783 - val_acc: 0.9798
Epoch 15/30
48000/48000 [==============================] - 10s 218us/step - loss:
0.0852 - acc: 0.9751 - val_loss: 0.0732 - val_acc: 0.9790
Epoch 16/30
48000/48000 [==============================] - 10s 215us/step - loss:
0.0813 - acc: 0.9757 - val_loss: 0.0715 - val_acc: 0.9813
Epoch 17/30
48000/48000 [==============================] - 10s 215us/step - loss:
0.0745 - acc: 0.9771 - val_loss: 0.0717 - val_acc: 0.9803
Epoch 18/30
48000/48000 [==============================] - 10s 217us/step - loss:
0.0751 - acc: 0.9769 - val_loss: 0.0701 - val_acc: 0.9813
```

```
Epoch 19/30
48000/48000 [==============================] - 10s 214us/step - loss:
0.0718 - acc: 0.9775 - val_loss: 0.0710 - val_acc: 0.9805
Epoch 20/30
48000/48000 [==============================] - 10s 214us/step - loss:
0.0675 - acc: 0.9788 - val_loss: 0.0679 - val_acc: 0.9824
Epoch 21/30
48000/48000 [==============================] - 10s 214us/step - loss:
0.0680 - acc: 0.9796 - val_loss: 0.0690 - val_acc: 0.9808
Epoch 22/30
48000/48000 [==============================] - 10s 216us/step - loss:
0.0636 - acc: 0.9803 - val_loss: 0.0734 - val_acc: 0.9803
Epoch 23/30
48000/48000 [==============================] - 10s 215us/step - loss:
0.0672 - acc: 0.9792 - val_loss: 0.0675 - val_acc: 0.9806
Epoch 24/30
48000/48000 [==============================] - 10s 214us/step - loss:
0.0638 - acc: 0.9799 - val_loss: 0.0705 - val_acc: 0.9818
Epoch 25/30
48000/48000 [==============================] - 10s 213us/step - loss:
0.0602 - acc: 0.9811 - val_loss: 0.0679 - val_acc: 0.9822
Epoch 26/30
48000/48000 [==============================] - 10s 216us/step - loss:
0.0581 - acc: 0.9819 - val_loss: 0.0677 - val_acc: 0.9815
Epoch 27/30
48000/48000 [==============================] - 10s 216us/step - loss:
0.0584 - acc: 0.9813 - val_loss: 0.0613 - val_acc: 0.9827
Epoch 28/30
48000/48000 [==============================] - 10s 213us/step - loss:
0.0560 - acc: 0.9831 - val_loss: 0.0673 - val_acc: 0.9814
Epoch 29/30
48000/48000 [==============================] - 10s 212us/step - loss:
0.0556 - acc: 0.9827 - val_loss: 0.0653 - val_acc: 0.9823
Epoch 30/30
48000/48000 [==============================] - 11s 220us/step - loss:
0.0522 - acc: 0.9839 - val_loss: 0.0647 - val_acc: 0.9824
```

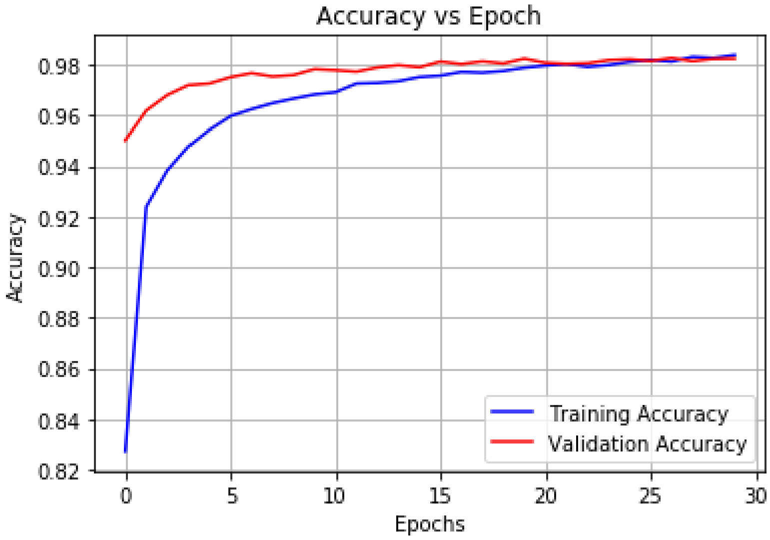
In [134]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [135]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [136]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_40', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'batch_normalization_14', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_16', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_41', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_15', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_17', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_42', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_16', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_18', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_43', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.12, 'seed': None}}, 'b
```

```
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

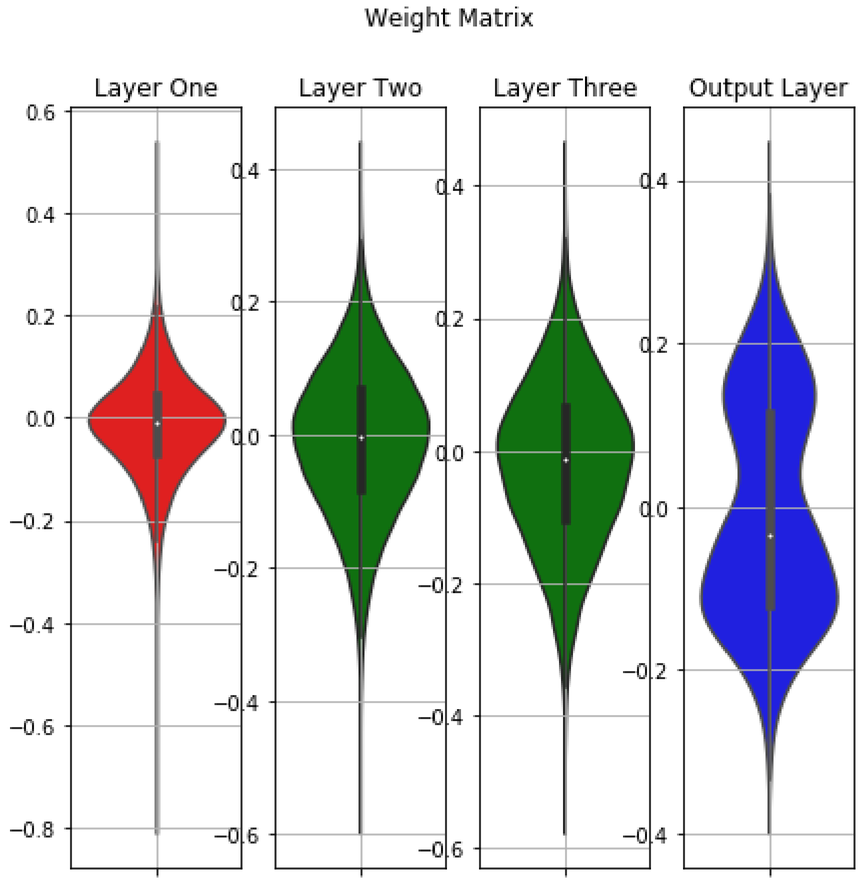
In [137]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[3].get_weights()[0]
layer3_weights = model.layers[6].get_weights()[0]
output_weights = model.layers[9].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Layer Three Weight Matrix Shape
=====================================================================
===============================
(256, 128)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(128, 10)
```

In [138]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 4, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,4,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,4,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='g')
plt.grid()

plt.subplot(1,4,4)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.7.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [140]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
======================================================================
==============================
0.05935220344241825
 Evaluate Accuracy
======================================================================
==============================
0.9838
```

4.7.4 Model Predict:

In [141]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[50].reshape(1,784)
test_data.shape
```

Out[141]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [143]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[50].argmax())

print("The Actual Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
6
The Actual Label of the predicted data
======================================================================
===============================
6
```

4.7.5 Observation:

In [144]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Dropout rate","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",0.5,100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([3,0.0522 ,0.0647  ,0.0593,0.9839,0.9824,0.9838])
print(b)
print(a)
```

```
+----------------+-----------+--------------+------------+--------+
| Initialization | Optimizer | Dropout rate | Batch Size | Epochs |
+----------------+-----------+--------------+------------+--------+
|     Xavier     |    Adam   |     0.5      |    100     |   30   |
+----------------+-----------+--------------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   3   |   0.0522   |  0.0647  |   0.0593  |   0.9839  |  0.9824 |
0.9838  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

4.8. Five Hidden Layer Architecture with BN and Dropout:

In [145]:

```
# Parameters

ip = X_train.shape[1]
layer_1 = 512
layer_2 = 256
layer_3 = 128
layer_4 = 64
layer_5 = 32
op = 10

print("Input layer size")
print("="*100)
print(ip)
print("Layer 1 size")
print("="*100)
print(layer_1)
print("Layer 2 size")
print("="*100)
print(layer_2)
print("Layer 3 size")
print("="*100)
print(layer_3)
print("Layer 4 size")
print("="*100)
print(layer_4)
print("Layer 5 size")
print("="*100)
print(layer_5)
print("Output layer size")
print("="*100)
print(op)
```

```
Input layer size
======================================================================
==============================
784
Layer 1 size
======================================================================
==============================
512
Layer 2 size
======================================================================
==============================
256
Layer 3 size
======================================================================
==============================
128
Layer 4 size
======================================================================
==============================
64
Layer 5 size
======================================================================
==============================
32
Output layer size
======================================================================
```

```
================================
10
```

4.8.1 Sequence Model :

Xavier/ Glorot Initialization :

$$sigma = \sqrt{\frac{2}{fan_i + fan_o}}$$

In [0]:

```
# Model Creation

model = Sequential()

# Layer 1

model.add(Dense(layer_1,input_dim=ip,activation="relu",kernel_initializer=RandomNor

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 2

model.add(Dense(layer_2,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 3

model.add(Dense(layer_3,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 4

model.add(Dense(layer_4,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())

# Dropout Layer

model.add(Dropout(0.5))

# Layer 5

model.add(Dense(layer_5,activation="relu",kernel_initializer=RandomNormal(mean=0.0,

# Batch Normalization Layer

model.add(BatchNormalization())
```

```
# Dropout Layer

model.add(Dropout(0.5))


# Output layer

model.add(Dense(op,activation="softmax",kernel_initializer=RandomNormal(mean=0.0,st
```

In [147]:

```
# Model Summary

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_44 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_17 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_19 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_18 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_20 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_46 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_19 (Batc (None, 128)               512       
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_47 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_20 (Batc (None, 64)                256       
_________________________________________________________________
dropout_22 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_48 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_21 (Batc (None, 32)                128       
_________________________________________________________________
dropout_23 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_49 (Dense)             (None, 10)                330       
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
```

In [0]:

```
# Model compilation

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"]
```

4.8.2 Model Training:

In [149]:

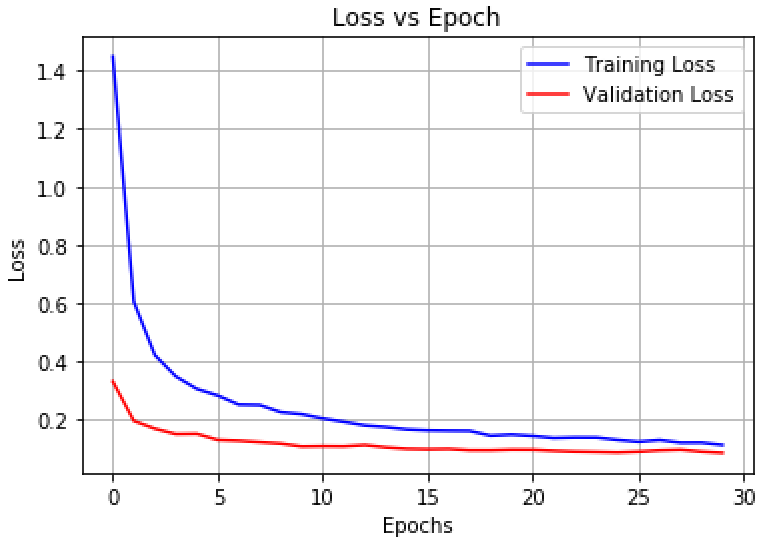
```
# model training

History=model.fit(X_train,Y_train,batch_size=100,epochs=30,verbose=1,validation_spl
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 15s 311us/step - loss:
1.4479 - acc: 0.5345 - val_loss: 0.3308 - val_acc: 0.9136
Epoch 2/30
48000/48000 [==============================] - 11s 238us/step - loss:
0.6059 - acc: 0.8328 - val_loss: 0.1950 - val_acc: 0.9474
Epoch 3/30
48000/48000 [==============================] - 11s 239us/step - loss:
0.4220 - acc: 0.8950 - val_loss: 0.1681 - val_acc: 0.9558
Epoch 4/30
48000/48000 [==============================] - 11s 240us/step - loss:
0.3484 - acc: 0.9143 - val_loss: 0.1494 - val_acc: 0.9622
Epoch 5/30
48000/48000 [==============================] - 12s 240us/step - loss:
0.3052 - acc: 0.9273 - val_loss: 0.1503 - val_acc: 0.9618
Epoch 6/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.2824 - acc: 0.9331 - val_loss: 0.1292 - val_acc: 0.9669
Epoch 7/30
48000/48000 [==============================] - 11s 239us/step - loss:
0.2510 - acc: 0.9411 - val_loss: 0.1264 - val_acc: 0.9680
Epoch 8/30
48000/48000 [==============================] - 11s 239us/step - loss:
0.2495 - acc: 0.9426 - val_loss: 0.1216 - val_acc: 0.9691
Epoch 9/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.2250 - acc: 0.9488 - val_loss: 0.1167 - val_acc: 0.9719
Epoch 10/30
48000/48000 [==============================] - 12s 241us/step - loss:
0.2180 - acc: 0.9492 - val_loss: 0.1066 - val_acc: 0.9731
Epoch 11/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.2033 - acc: 0.9533 - val_loss: 0.1076 - val_acc: 0.9731
Epoch 12/30
48000/48000 [==============================] - 12s 245us/step - loss:
0.1919 - acc: 0.9562 - val_loss: 0.1070 - val_acc: 0.9742
Epoch 13/30
48000/48000 [==============================] - 12s 248us/step - loss:
0.1793 - acc: 0.9586 - val_loss: 0.1116 - val_acc: 0.9747
Epoch 14/30
48000/48000 [==============================] - 12s 241us/step - loss:
0.1736 - acc: 0.9597 - val_loss: 0.1036 - val_acc: 0.9764
Epoch 15/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.1656 - acc: 0.9626 - val_loss: 0.0985 - val_acc: 0.9756
Epoch 16/30
48000/48000 [==============================] - 11s 239us/step - loss:
0.1622 - acc: 0.9627 - val_loss: 0.0972 - val_acc: 0.9765
Epoch 17/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.1610 - acc: 0.9633 - val_loss: 0.0982 - val_acc: 0.9777
Epoch 18/30
48000/48000 [==============================] - 12s 246us/step - loss:
0.1604 - acc: 0.9635 - val_loss: 0.0936 - val_acc: 0.9782
```

```
Epoch 19/30
48000/48000 [==============================] - 12s 243us/step - loss:
0.1438 - acc: 0.9676 - val_loss: 0.0936 - val_acc: 0.9788
Epoch 20/30
48000/48000 [==============================] - 12s 244us/step - loss:
0.1475 - acc: 0.9673 - val_loss: 0.0957 - val_acc: 0.9773
Epoch 21/30
48000/48000 [==============================] - 12s 243us/step - loss:
0.1429 - acc: 0.9669 - val_loss: 0.0954 - val_acc: 0.9787
Epoch 22/30
48000/48000 [==============================] - 12s 247us/step - loss:
0.1359 - acc: 0.9693 - val_loss: 0.0917 - val_acc: 0.9793
Epoch 23/30
48000/48000 [==============================] - 12s 245us/step - loss:
0.1378 - acc: 0.9679 - val_loss: 0.0890 - val_acc: 0.9798
Epoch 24/30
48000/48000 [==============================] - 12s 245us/step - loss:
0.1375 - acc: 0.9692 - val_loss: 0.0878 - val_acc: 0.9800
Epoch 25/30
48000/48000 [==============================] - 12s 247us/step - loss:
0.1292 - acc: 0.9707 - val_loss: 0.0862 - val_acc: 0.9805
Epoch 26/30
48000/48000 [==============================] - 12s 249us/step - loss:
0.1229 - acc: 0.9726 - val_loss: 0.0887 - val_acc: 0.9803
Epoch 27/30
48000/48000 [==============================] - 12s 255us/step - loss:
0.1289 - acc: 0.9715 - val_loss: 0.0932 - val_acc: 0.9803
Epoch 28/30
48000/48000 [==============================] - 12s 244us/step - loss:
0.1197 - acc: 0.9728 - val_loss: 0.0955 - val_acc: 0.9799
Epoch 29/30
48000/48000 [==============================] - 12s 244us/step - loss:
0.1199 - acc: 0.9722 - val_loss: 0.0888 - val_acc: 0.9815
Epoch 30/30
48000/48000 [==============================] - 12s 242us/step - loss:
0.1117 - acc: 0.9747 - val_loss: 0.0847 - val_acc: 0.9814
```

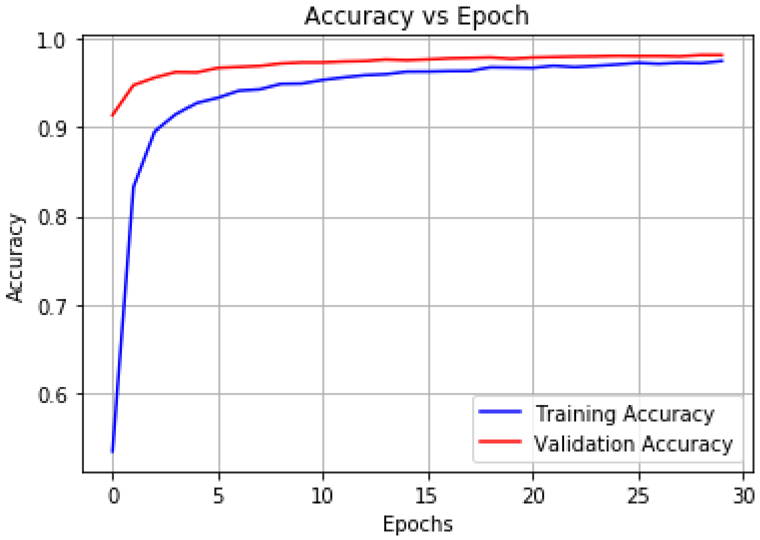
In [150]:

```
# References
# https://machinelearningmastery.com/display-deep-learning-model-training-history-i
# https://keras.io/models/sequential/

# Plotting Loss vs Epoch

plt.close()
plt.plot(History.history['loss'],'b',label="Training Loss")
plt.plot(History.history['val_loss'],'r',label="Validation Loss")
plt.title("Loss vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()
```



In [151]:

```
# Plotting Accuracy vs Epoch

plt.close()
plt.plot(History.history['acc'],'b',label="Training Accuracy")
plt.plot(History.history['val_acc'],'r',label="Validation Accuracy")
plt.title("Accuracy vs Epoch")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



In [152]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer


for layer in model.layers:
  print(layer.get_config())
```

```
{'name': 'dense_44', 'trainable': True, 'batch_input_shape': (None, 7
84), 'dtype': 'float32', 'units': 512, 'activation': 'relu', 'use_bia
s': True, 'kernel_initializer': {'class_name': 'RandomNormal', 'confi
g': {'mean': 0.0, 'stddev': 0.04, 'seed': None}}, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'kernel_regularizer': None, 'b
ias_regularizer': None, 'activity_regularizer': None, 'kernel_constra
int': None, 'bias_constraint': None}
{'name': 'batch_normalization_17', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_19', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_45', 'trainable': True, 'units': 256, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.05, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_18', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_20', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_46', 'trainable': True, 'units': 128, 'activation':
'relu', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rando
mNormal', 'config': {'mean': 0.0, 'stddev': 0.07, 'seed': None}}, 'bi
as_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regul
arizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_19', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_21', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_47', 'trainable': True, 'units': 64, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.1, 'seed': None}}, 'bias_
```

```
initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regulari
zer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_20', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_22', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_48', 'trainable': True, 'units': 32, 'activation': 'r
elu', 'use_bias': True, 'kernel_initializer': {'class_name': 'RandomN
ormal', 'config': {'mean': 0.0, 'stddev': 0.14, 'seed': None}}, 'bias
_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regular
izer': None, 'bias_regularizer': None, 'activity_regularizer': None,
'kernel_constraint': None, 'bias_constraint': None}
{'name': 'batch_normalization_21', 'trainable': True, 'axis': -1, 'mo
mentum': 0.99, 'epsilon': 0.001, 'center': True, 'scale': True, 'beta
_initializer': {'class_name': 'Zeros', 'config': {}}, 'gamma_initiali
zer': {'class_name': 'Ones', 'config': {}}, 'moving_mean_initialize
r': {'class_name': 'Zeros', 'config': {}}, 'moving_variance_initializ
er': {'class_name': 'Ones', 'config': {}}, 'beta_regularizer': None,
'gamma_regularizer': None, 'beta_constraint': None, 'gamma_constrain
t': None}
{'name': 'dropout_23', 'trainable': True, 'rate': 0.5, 'noise_shape':
None, 'seed': None}
{'name': 'dense_49', 'trainable': True, 'units': 10, 'activation': 's
oftmax', 'use_bias': True, 'kernel_initializer': {'class_name': 'Rand
omNormal', 'config': {'mean': 0.0, 'stddev': 0.22, 'seed': None}}, 'b
ias_initializer': {'class_name': 'Zeros', 'config': {}}, 'kernel_regu
larizer': None, 'bias_regularizer': None, 'activity_regularizer': Non
e, 'kernel_constraint': None, 'bias_constraint': None}
```

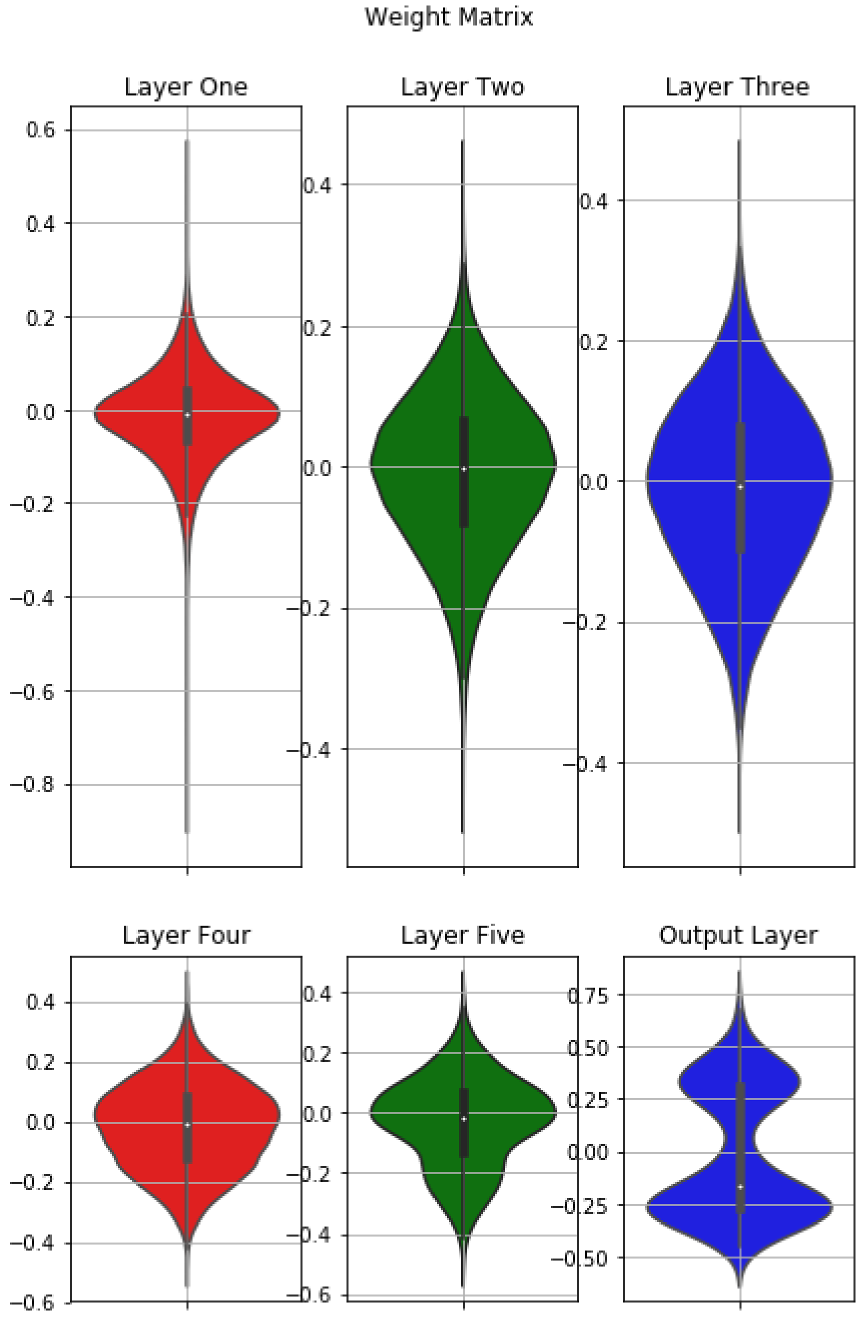
In [154]:

```
# References
# https://stackoverflow.com/questions/43715047/keras-2-x-get-weights-of-layer

layer1_weights = model.layers[0].get_weights()[0]
layer2_weights = model.layers[3].get_weights()[0]
layer3_weights = model.layers[6].get_weights()[0]
layer4_weights = model.layers[9].get_weights()[0]
layer5_weights = model.layers[12].get_weights()[0]
output_weights = model.layers[15].get_weights()[0]

print(" Layer One Weight Matrix Shape")
print("="*100)
print(layer1_weights.shape)
print(" Layer Two Weight Matrix Shape")
print("="*100)
print(layer2_weights.shape)
print(" Layer Three Weight Matrix Shape")
print("="*100)
print(layer3_weights.shape)
print(" Layer Four Weight Matrix Shape")
print("="*100)
print(layer4_weights.shape)
print(" Layer Five Weight Matrix Shape")
print("="*100)
print(layer5_weights.shape)
print(" Output Layer Weight Matrix Shape")
print("="*100)
print(output_weights.shape)
```

```
 Layer One Weight Matrix Shape
=====================================================================
===============================
(784, 512)
 Layer Two Weight Matrix Shape
=====================================================================
===============================
(512, 256)
 Layer Three Weight Matrix Shape
=====================================================================
===============================
(256, 128)
 Layer Four Weight Matrix Shape
=====================================================================
===============================
(128, 64)
 Layer Five Weight Matrix Shape
=====================================================================
===============================
(64, 32)
 Output Layer Weight Matrix Shape
=====================================================================
===============================
(32, 10)
```

In [155]:

```
# References
# https://seaborn.pydata.org/generated/seaborn.violinplot.html
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html

fig = plt.figure(1,figsize=(7,7))
fig.suptitle("Weight Matrix")
plt.subplot(1, 3, 1)
plt.title("Layer One")
sns.violinplot(y=layer1_weights,color='r')
plt.grid()

plt.subplot(1,3,2)
plt.title("Layer Two")
sns.violinplot(y=layer2_weights,color='g')
plt.grid()

plt.subplot(1,3,3)
plt.title("Layer Three")
sns.violinplot(y=layer3_weights,color='b')
plt.grid()

fig = plt.figure(2,figsize=(7,7))

plt.subplot(2,3,1)
plt.title("Layer Four")
sns.violinplot(y=layer4_weights,color='r')
plt.grid()

plt.subplot(2,3,2)
plt.title("Layer Five")
sns.violinplot(y=layer5_weights,color='g')
plt.grid()

plt.subplot(2,3,3)
plt.title("Output Layer")
sns.violinplot(y=output_weights,color='b')
plt.grid()
```



Observation: The weights are not too large and not too small. The weights are normally distributed.

4.8.3 Model Evaluation:

In [0]:

```
# Refernces
# https://keras.io/models/model/#evaluate

evaluate_data=model.evaluate(X_test,Y_test,verbose=0)
```

In [157]:

```
evaluate_loss = evaluate_data[0]
evaluate_acc  = evaluate_data[1]

print(" Evaluate Loss")
print("="*100)
print(evaluate_loss)

print(" Evaluate Accuracy")
print("="*100)
print(evaluate_acc)
```

```
 Evaluate Loss
======================================================================
===============================
0.08053734491495415
 Evaluate Accuracy
======================================================================
===============================
0.9817
```

4.8.4 Model Predict:

In [158]:

```
# References
# https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-data
# https://keras.io/models/model/#predict

test_data=X_test[500].reshape(1,784)
test_data.shape
```

Out[158]:

```
(1, 784)
```

In [0]:

```
pred_label = model.predict(test_data)
```

In [160]:

```
print("The Actual Label of the predicted data")
print("="*100)
print(Y_test[500].argmax())

print("The Actual Label of the predicted data")
print("="*100)
print(pred_label.argmax())
```

```
The Actual Label of the predicted data
======================================================================
===============================
3
The Actual Label of the predicted data
======================================================================
===============================
3
```

4.8.5 Observation:

In [163]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Dropout rate","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",0.5,100,30])
a.field_names = ["Layer","Train loss","Val_loss","Test_loss","Train_acc","Val_acc",
a.add_row([5,0.1117,0.0847 ,0.0805,0.9747,0.9814,0.9817])

print(b)
print(a)
```

```
+----------------+-----------+--------------+------------+--------+
| Initialization | Optimizer | Dropout rate | Batch Size | Epochs |
+----------------+-----------+--------------+------------+--------+
|     Xavier     |    Adam   |     0.5      |    100     |   30   |
+----------------+-----------+--------------+------------+--------+
+-------+------------+----------+-----------+-----------+---------+--
--------+
| Layer | Train loss | Val_loss | Test_loss | Train_acc | Val_acc | T
est_acc |
+-------+------------+----------+-----------+-----------+---------+--
--------+
|   5   |   0.1117   |  0.0847  |   0.0805  |   0.9747  |  0.9814 |
0.9817  |
+-------+------------+----------+-----------+-----------+---------+--
--------+
```

5. Conclusion:

In [164]:

```
a=PrettyTable()
b=PrettyTable()
b.field_names = ["Initialization","Optimizer","Dropout rate","Batch Size","Epochs"]
b.add_row(["Xavier","Adam",0.5,100,30])
a.field_names = ["Layer","BN","Dropout","Train loss","Val_loss","Test_loss","Train_
a.add_row([2,"NO","NO",0.0038,0.1159 ,0.1078,0.9989,0.9813,0.9813])
a.add_row([3,"NO","NO",0.0055 ,0.1120 ,0.0880,0.9985,0.9787,0.9828])
a.add_row([5,"NO","NO",0.0061 ,0.1261 ,0.1032,0.9983,0.9772,0.9816])
a.add_row([3,"YES","NO",0.0055 ,0.0968  ,0.0877,0.9982,0.9806,0.9805])
a.add_row([5,"NO","YES",0.1554 ,0.1328  ,0.1376,0.9617,0.9758,0.9762])
a.add_row([2,"YES","YES",0.0428 ,0.0675  ,0.0571,0.9850,0.9816,0.9841])
a.add_row([3,"YES","YES",0.0522 ,0.0647  ,0.0593,0.9839,0.9824,0.9838])
a.add_row([5,"YES","YES",0.1117,0.0847 ,0.0805,0.9747,0.9814,0.9817])
print("The common factors for all architecture")
print("="*100)
print(b)
print("Architecture loss and accuracy comparision")
print("="*100)
print(a)
```

```
The common factors for all architecture
=====================================================================
===============================
+----------------+-----------+--------------+------------+--------+
| Initialization | Optimizer | Dropout rate | Batch Size | Epochs |
+----------------+-----------+--------------+------------+--------+
|     Xavier     |    Adam   |     0.5      |    100     |   30   |
+----------------+-----------+--------------+------------+--------+
Architecture loss and accuracy comparision
=====================================================================
===============================
+-------+-----+---------+------------+----------+-----------+--------
---+---------+----------+
| Layer |  BN | Dropout | Train loss | Val_loss | Test_loss | Train_a
cc | Val_acc | Test_acc |
+-------+-----+---------+------------+----------+-----------+--------
---+---------+----------+
|   2   |  NO |    NO   |   0.0038   |  0.1159  |   0.1078  |   0.998
9  |  0.9813 |  0.9813  |
|   3   |  NO |    NO   |   0.0055   |  0.112   |   0.088   |   0.998
5  |  0.9787 |  0.9828  |
|   5   |  NO |    NO   |   0.0061   |  0.1261  |   0.1032  |   0.998
3  |  0.9772 |  0.9816  |
|   3   | YES |    NO   |   0.0055   |  0.0968  |   0.0877  |   0.998
2  |  0.9806 |  0.9805  |
|   5   |  NO |   YES   |   0.1554   |  0.1328  |   0.1376  |   0.961
7  |  0.9758 |  0.9762  |
|   2   | YES |   YES   |   0.0428   |  0.0675  |   0.0571  |   0.985
|  0.9816 |  0.9841  |
|   3   | YES |   YES   |   0.0522   |  0.0647  |   0.0593  |   0.983
9  |  0.9824 |  0.9838  |
|   5   | YES |   YES   |   0.1117   |  0.0847  |   0.0805  |   0.974
7  |  0.9814 |  0.9817  |
+-------+-----+---------+------------+----------+-----------+--------
---+---------+----------+
```

Data Preparation:

The Minist Data was gathered by using Keras library and it was splitted into Train and Test data.The train and Test data have (28,28) dimension. So the (28,28) has been converted into (784,). And the 10 class labeled data was converted into categorical labeled data using One Hot Encoding.

Data Normalization:

Before the data's are going into the model, the data's are Normalized using Data Normalization formula.

Deep Neural Network:

The Different architectures (2 layer, 3 layer, 5 layers) have been implemented.The Xavier Initialization was implemented for this assignment.The Batch Normalization layer is used to avoid the Internal Covariance shift and the Dropout layer is used to avoid the Model Overfitting. Adam Optimizer has been implemented.

Loss vs Epoch plot:

Loss(Train and Validation) vs Epoch graph has been plotted sucessfully for each of the Architecture.

Weight Plot:

Weight (Weight matrix got After the Optimization) graph has been plotted successfully for each of the Architecture.