Keras -- MLPs on MNIST

In [1]:

```
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam
from keras.layers import Dense, Activation
import seaborn as sns
import numpy as np
import keras
```

```
Using TensorFlow backend.
```

In [2]:

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

In [3]:

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

In [4]:

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

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

In [5]:

```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

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

In [6]:

```
# after converting the input images from 3d to 2d vectors

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

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

In [7]:

```python
numofclasses = 10
y_train = keras.utils.to_categorical(y_train, numofclasses)
y_test = keras.utils.to_categorical(y_test, numofclasses)
y_train[:5]
```

Out[7]:

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

In [8]:

```python
# An example data point
print(X_train[0])
```

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

In [9]:

```python
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
```

```python
# before we move to apply machine learning algorithms lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [10]:

```python
# example data point after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.31764706  0.94117647  0.99215686
 0.99215686  0.46666667  0.09803922  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.17647059  0.72941176  0.99215686  0.99215686
 0.58823529  0.10588235  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.0627451   0.36470588  0.98823529  0.99215686  0.73333333
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.97647059  0.99215686  0.97647059  0.25098039  0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.18039216  0.50980392  0.71764706  0.99215686
 0.99215686  0.81176471  0.00784314  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.15294118  0.58039216
 0.89803922  0.99215686  0.99215686  0.99215686  0.98039216  0.71372549
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.09411765  0.44705882  0.86666667  0.99215686  0.99215686  0.99215686
 0.99215686  0.78823529  0.30588235  0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.09019608  0.25882353  0.83529412  0.99215686
 0.99215686  0.99215686  0.99215686  0.77647059  0.31764706  0.00784314
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.07058824  0.67058824
 0.85882353  0.99215686  0.99215686  0.99215686  0.99215686  0.76470588
 0.31372549  0.03529412  0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.21568627  0.6745098   0.88627451  0.99215686  0.99215686  0.99215686
 0.99215686  0.95686275  0.52156863  0.04313725  0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.53333333  0.99215686
 0.99215686  0.99215686  0.83137255  0.52941176  0.51764706  0.0627451
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

MODEL 1 WITHOUT BN AND DROPOUT 2 LAYERS :

In [11]:

```
# some model parameters

output_dim = 10
```

```
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [12]:

```
model1 = Sequential()
model1.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model1.add(Dense(120, activation='relu', kernel_initializer='random_uniform'))
model1.add(Dense(output_dim, activation='softmax'))
print(model1.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 364)               285740
_________________________________________________________________
dense_2 (Dense)              (None, 120)               43800
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1210
=================================================================
Total params: 330,750
Trainable params: 330,750
Non-trainable params: 0
_________________________________________________________________
None
```

In [13]:

```
model1.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [14]:

```
# Training the model
results = model1.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch,validation_data=(X_
test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.2798 - accuracy: 0.9214 - va
l_loss: 0.1341 - val_accuracy: 0.9596
Epoch 2/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1042 - accuracy: 0.9686 - va
l_loss: 0.0871 - val_accuracy: 0.9723
Epoch 3/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0670 - accuracy: 0.9790 - va
l_loss: 0.0798 - val_accuracy: 0.9752
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0471 - accuracy: 0.9855 - va
l_loss: 0.0749 - val_accuracy: 0.9760
Epoch 5/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0337 - accuracy: 0.9891 - va
l_loss: 0.0765 - val_accuracy: 0.9769
Epoch 6/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0270 - accuracy: 0.9914 - va
l_loss: 0.0701 - val_accuracy: 0.9789
Epoch 7/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0202 - accuracy: 0.9939 - va
l_loss: 0.0808 - val_accuracy: 0.9762
Epoch 8/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0163 - accuracy: 0.9946 - va
l_loss: 0.0774 - val_accuracy: 0.9784
Epoch 9/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0170 - accuracy: 0.9944 - va
l_loss: 0.0937 - val_accuracy: 0.9762
Epoch 10/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0137 - accuracy: 0.9953 - va
l_loss: 0.0723 - val_accuracy: 0.9812
Epoch 11/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0093 - accuracy: 0.9970 - va
```

```
l_loss: 0.0795 - val_accuracy: 0.9809
Epoch 12/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0104 - accuracy: 0.9967 - va
l_loss: 0.0960 - val_accuracy: 0.9773
Epoch 13/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0119 - accuracy: 0.9961 - va
l_loss: 0.1018 - val_accuracy: 0.9759
Epoch 14/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0097 - accuracy: 0.9969 - va
l_loss: 0.0787 - val_accuracy: 0.9809
Epoch 15/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0077 - accuracy: 0.9977 - va
l_loss: 0.0833 - val_accuracy: 0.9812
Epoch 16/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.0087 - accuracy: 0.9967 - va
l_loss: 0.0940 - val_accuracy: 0.9796
Epoch 17/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0064 - accuracy: 0.9977 - va
l_loss: 0.1040 - val_accuracy: 0.9794
Epoch 18/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0073 - accuracy: 0.9975 - va
l_loss: 0.1002 - val_accuracy: 0.9800
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0095 - accuracy: 0.9970 - va
l_loss: 0.0984 - val_accuracy: 0.9804
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0053 - accuracy: 0.9984 - va
l_loss: 0.0927 - val_accuracy: 0.9830
```

In [15]:

```
score = model1.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 72us/step
```

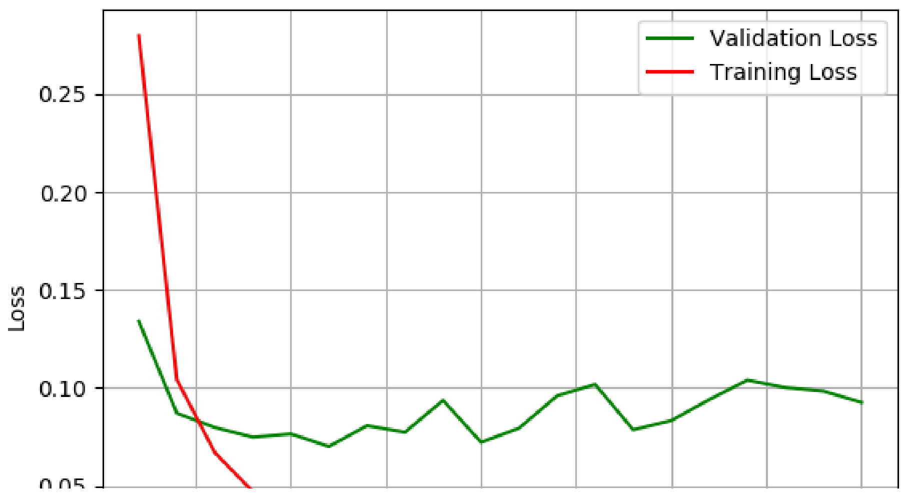
In [16]:

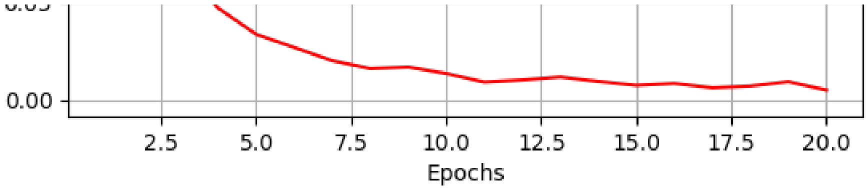
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```





In [17]:

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

```
Test accuracy: 0.9829999804496765
```

2 LAYERS WITH BN AND DROPOUT :

In [18]:

```
model2 = Sequential()

model2.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model2.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))
model2.add(Dense(output_dim, activation='softmax'))
print(model2.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 364)               285740    
_________________________________________________________________
dense_5 (Dense)              (None, 128)               46720     
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512       
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290      
=================================================================
Total params: 334,262
Trainable params: 334,006
Non-trainable params: 256
_________________________________________________________________
None
```

In [19]:

```
model2.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

In [20]:

```
results = model2.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 30, valid
ation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
Epoch 2/20
Epoch 3/20
Epoch 4/20
Epoch 5/20
Epoch 6/20
Epoch 7/20
Epoch 8/20
Epoch 9/20
Epoch 10/20
Epoch 11/20
```

```
Epoch 12/20
Epoch 13/20
Epoch 14/20
Epoch 15/20
Epoch 16/20
Epoch 17/20
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

In [21]:

```
score = model2.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 0s 44us/step
```

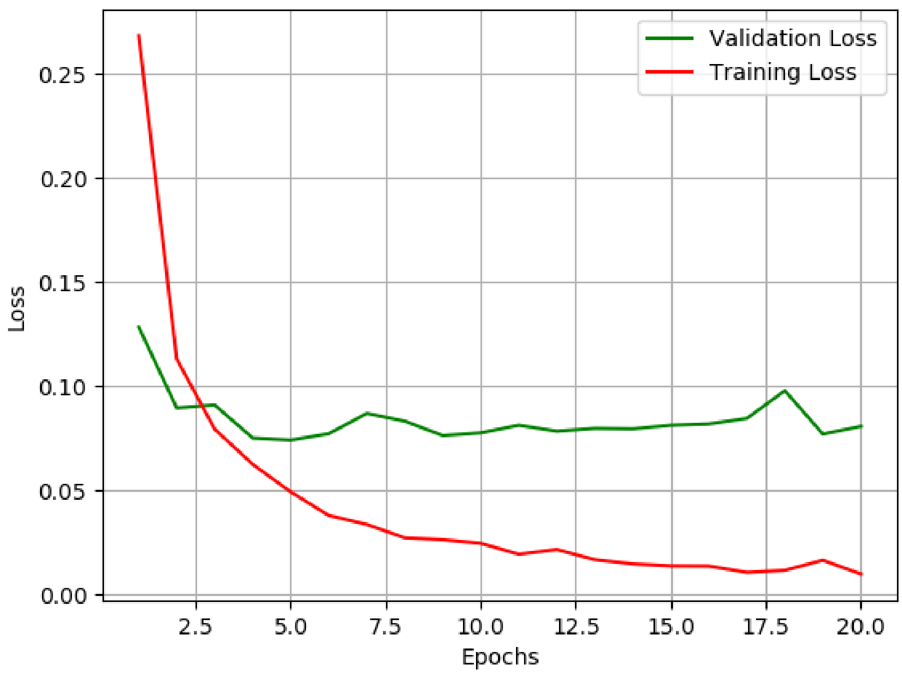
In [22]:

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

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [23]:

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

```
Test accuracy: 0.9825000166893005
```

Changing Dropout to 0.8

In [24]:

```
model2 = Sequential()
model2.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model2.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model2.add(BatchNormalization())
model2.add(Dropout(0.8))
model2.add(Dense(output_dim, activation='softmax'))
print(model2.summary())
```

```
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 364)               285740
_________________________________________________________________
dense_8 (Dense)              (None, 128)               46720
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 334,262
Trainable params: 334,006
Non-trainable params: 256
_________________________________________________________________
None
```

In [25]:

```
model2.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [26]:

```
results = model2.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.4992 - accuracy: 0.8541 - va
l_loss: 0.1452 - val_accuracy: 0.9573
Epoch 2/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1964 - accuracy: 0.9478 - va
l_loss: 0.1035 - val_accuracy: 0.9687
Epoch 3/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1378 - accuracy: 0.9629 - va
l_loss: 0.0976 - val_accuracy: 0.9713
Epoch 4/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.1093 - accuracy: 0.9704 - va
l_loss: 0.0900 - val_accuracy: 0.9725
Epoch 5/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0933 - accuracy: 0.9739 - va
l_loss: 0.0865 - val_accuracy: 0.9770
Epoch 6/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0737 - accuracy: 0.9795 - va
l_loss: 0.0785 - val_accuracy: 0.9791
Epoch 7/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0670 - accuracy: 0.9811 - va
l_loss: 0.0735 - val_accuracy: 0.9802
Epoch 8/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0579 - accuracy: 0.9838 - va
l_loss: 0.0868 - val_accuracy: 0.9752
Epoch 9/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0513 - accuracy: 0.9852 - va
l_loss: 0.0798 - val_accuracy: 0.9786
Epoch 10/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0457 - accuracy: 0.9865 - va
l_loss: 0.0820 - val_accuracy: 0.9792
```

```
l_loss: 0.0820 - val_accuracy: 0.9792
Epoch 11/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0431 - accuracy: 0.9877 - va
l_loss: 0.0830 - val_accuracy: 0.9769
Epoch 12/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0395 - accuracy: 0.9879 - va
l_loss: 0.0916 - val_accuracy: 0.9783
Epoch 13/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0376 - accuracy: 0.9887 - va
l_loss: 0.0997 - val_accuracy: 0.9765
Epoch 14/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0311 - accuracy: 0.9911 - va
l_loss: 0.1028 - val_accuracy: 0.9774
Epoch 15/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0301 - accuracy: 0.9911 - va
l_loss: 0.0919 - val_accuracy: 0.9795
Epoch 16/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0261 - accuracy: 0.9922 - va
l_loss: 0.0871 - val_accuracy: 0.9811
Epoch 17/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0280 - accuracy: 0.9916 - va
l_loss: 0.1085 - val_accuracy: 0.9779
Epoch 18/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0239 - accuracy: 0.9927 - va
l_loss: 0.0917 - val_accuracy: 0.9800
Epoch 19/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0249 - accuracy: 0.9928 - va
l_loss: 0.0943 - val_accuracy: 0.9820
Epoch 20/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0210 - accuracy: 0.9937 - va
l_loss: 0.0877 - val_accuracy: 0.9826
```

In [27]:

```
score = model2.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 0s 44us/step
```

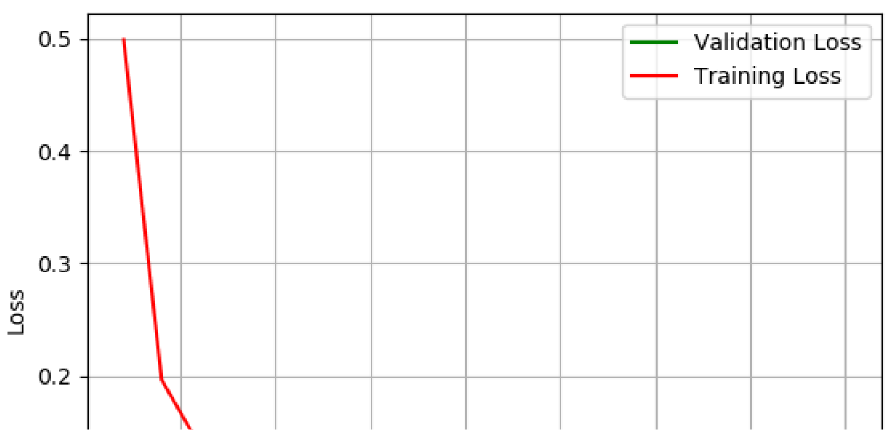
In [28]:

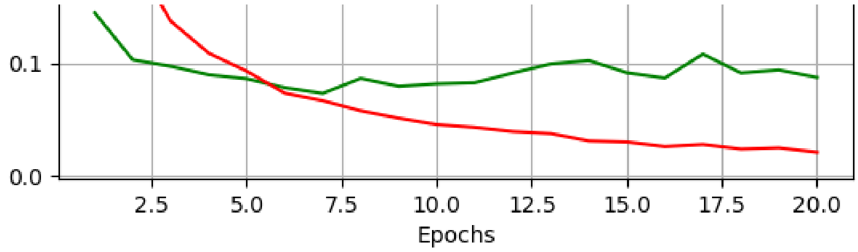
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```





In [29]:

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

```
Test accuracy: 0.9825999736785889
```

Changing DropOut Rate To 0.2

In [30]:

```
model2 = Sequential()
model2.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model2.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model2.add(BatchNormalization())
model2.add(Dropout(0.2))
model2.add(Dense(output_dim, activation='softmax'))
print(model2.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_11 (Dense)             (None, 128)               46720     
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512       
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 334,262
Trainable params: 334,006
Non-trainable params: 256
_________________________________________________________________
None
```

In [31]:

```
model2.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [32]:

```
results = model2.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.2127 - accuracy: 0.9363 - va
l_loss: 0.1067 - val_accuracy: 0.9690
Epoch 2/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0897 - accuracy: 0.9721 - va
l_loss: 0.0833 - val_accuracy: 0.9741
Epoch 3/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0615 - accuracy: 0.9812 - va
l_loss: 0.0780 - val_accuracy: 0.9762
Epoch 4/20
```

```
60000/60000 [==============================] - 3s 55us/step - loss: 0.0446 - accuracy: 0.9855 - va
l_loss: 0.0775 - val_accuracy: 0.9768
Epoch 5/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0361 - accuracy: 0.9886 - va
l_loss: 0.0817 - val_accuracy: 0.9771
Epoch 6/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0288 - accuracy: 0.9906 - va
l_loss: 0.0872 - val_accuracy: 0.9760
Epoch 7/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0243 - accuracy: 0.9920 - va
l_loss: 0.0730 - val_accuracy: 0.9805
Epoch 8/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0232 - accuracy: 0.9927 - va
l_loss: 0.0688 - val_accuracy: 0.9805
Epoch 9/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0205 - accuracy: 0.9932 - va
l_loss: 0.0810 - val_accuracy: 0.9773
Epoch 10/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0158 - accuracy: 0.9949 - va
l_loss: 0.0778 - val_accuracy: 0.9802
Epoch 11/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0143 - accuracy: 0.9952 - va
l_loss: 0.0795 - val_accuracy: 0.9788
Epoch 12/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0151 - accuracy: 0.9949 - va
l_loss: 0.0922 - val_accuracy: 0.9781
Epoch 13/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0132 - accuracy: 0.9956 - va
l_loss: 0.0788 - val_accuracy: 0.9798
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0107 - accuracy: 0.9962 - va
l_loss: 0.0929 - val_accuracy: 0.9759
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0148 - accuracy: 0.9949 - va
l_loss: 0.0911 - val_accuracy: 0.9792
Epoch 16/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0107 - accuracy: 0.9964 - va
l_loss: 0.0846 - val_accuracy: 0.9805
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0068 - accuracy: 0.9979 - va
l_loss: 0.0890 - val_accuracy: 0.9790
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0087 - accuracy: 0.9970 - va
l_loss: 0.0911 - val_accuracy: 0.9795
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0111 - accuracy: 0.9963 - va
l_loss: 0.0888 - val_accuracy: 0.9774
Epoch 20/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0099 - accuracy: 0.9966 - va
l_loss: 0.0831 - val_accuracy: 0.9816
```

In [33]:

```python
score = model2.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 0s 46us/step
```

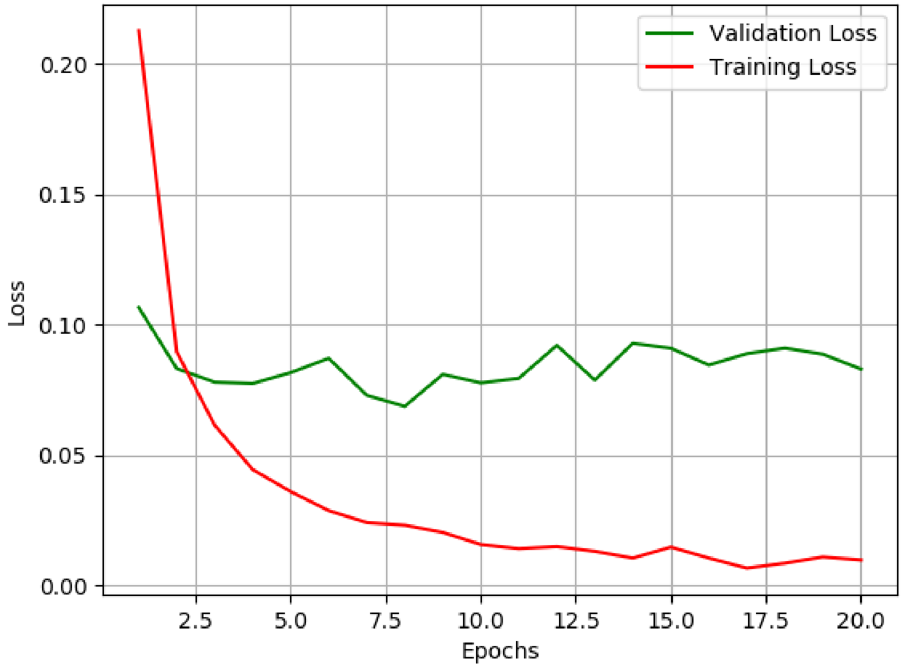
In [34]:

```python
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [35]:

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

```
Test accuracy: 0.9815999865531921
```

Model With 3 Hidden Layers :

In [36]:

```
model3 = Sequential()
model3.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model3.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model3.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model3.add(Dense(output_dim, activation='softmax'))
print(model3.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_14 (Dense)             (None, 128)               46720     
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650       
=================================================================
Total params: 341,366
Trainable params: 341,366
Non-trainable params: 0
_________________________________________________________________
None
```

In [37]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [38]:

```
results = model3.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 30, valid
ation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
Epoch 2/20
Epoch 3/20
Epoch 4/20
Epoch 5/20
Epoch 6/20
Epoch 7/20
Epoch 8/20
Epoch 9/20
Epoch 10/20
Epoch 11/20
Epoch 12/20
Epoch 13/20
Epoch 14/20
Epoch 15/20
Epoch 16/20
Epoch 17/20
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

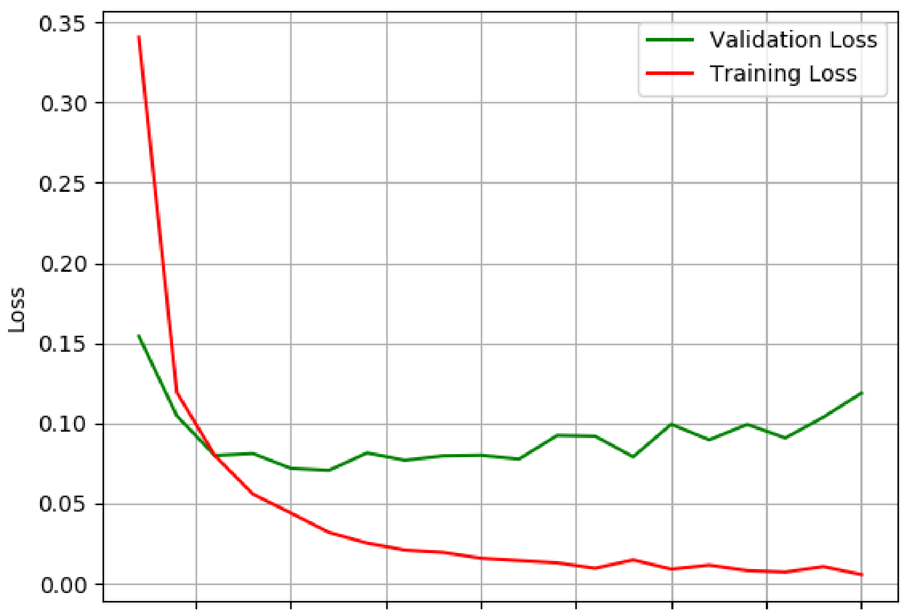
In [39]:

```
score = model3.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 0s 50us/step
```

In [40]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')
# list of epoch numbers
list_of_epoch = list(range(1,nb_epoch+1))
train_loss = results.history['loss']
val_loss = results.history['val_loss']
ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [41]:

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

```
Test accuracy: 0.9761999845504761
```

Model with 3 hidden layers with BN & DropOut :

In [86]:

```
model3 = Sequential()
model3.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model3.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model3.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(Dense(output_dim, activation='softmax'))
print(model3.summary())
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_53 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_54 (Dense)             (None, 128)               46720     
_________________________________________________________________
dense_55 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_10 (Batc (None, 64)                256       
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_56 (Dense)             (None, 10)                650       
=================================================================
Total params: 341,622
Trainable params: 341,494
Non-trainable params: 128
_________________________________________________________________
None
```

In [87]:

```
model3.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [88]:

```
results = model3.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 30, valid
ation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
Epoch 2/20
Epoch 3/20
Epoch 4/20
Epoch 5/20
Epoch 6/20
Epoch 7/20
Epoch 8/20
Epoch 9/20
Epoch 10/20
Epoch 11/20
Epoch 12/20
Epoch 13/20
Epoch 14/20
```

```
Epoch 15/20
Epoch 16/20
Epoch 17/20
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

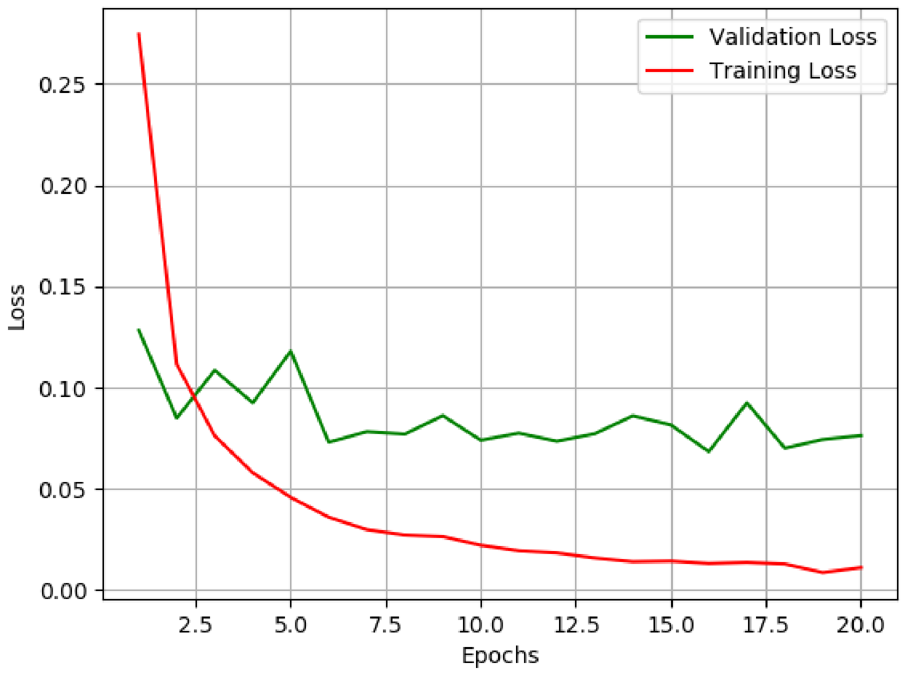
In [89]:

```
score = model3.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 54us/step
```

In [90]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')
# list of epoch numbers
list_of_epoch = list(range(1,nb_epoch+1))
train_loss = results.history['loss']
val_loss = results.history['val_loss']
ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [91]:

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

```
Test accuracy: 0.9818000197410583
```

DropOut with 0.8

In [92]:

```
model4 = Sequential()
model4.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
```

```
m'))
model4.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model4.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model4.add(BatchNormalization())
model4.add(Dropout(0.8))
model4.add(Dense(output_dim, activation='softmax'))
print(model4.summary())
```

```
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_57 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_58 (Dense)             (None, 128)               46720
_________________________________________________________________
dense_59 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_11 (Batc (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_60 (Dense)             (None, 10)                650
=================================================================
Total params: 341,622
Trainable params: 341,494
Non-trainable params: 128
_________________________________________________________________
None
```

In [93]:

```
model4.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [94]:

```
results = model4.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.5065 - accuracy: 0.8498 - va
l_loss: 0.1633 - val_accuracy: 0.9533
Epoch 2/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.2083 - accuracy: 0.9464 - va
l_loss: 0.1047 - val_accuracy: 0.9675
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1507 - accuracy: 0.9619 - va
l_loss: 0.1190 - val_accuracy: 0.9679
Epoch 4/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.1111 - accuracy: 0.9704 - va
l_loss: 0.0968 - val_accuracy: 0.9742
Epoch 5/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0969 - accuracy: 0.9743 - va
l_loss: 0.1000 - val_accuracy: 0.9737
Epoch 6/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0843 - accuracy: 0.9776 - va
l_loss: 0.1257 - val_accuracy: 0.9717
Epoch 7/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0712 - accuracy: 0.9806 - va
l_loss: 0.0896 - val_accuracy: 0.9783
Epoch 8/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0629 - accuracy: 0.9821 - va
l_loss: 0.1210 - val_accuracy: 0.9756
Epoch 9/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0592 - accuracy: 0.9829 - va
l_loss: 0.0984 - val_accuracy: 0.9795
Epoch 10/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0490 - accuracy: 0.9848 - va
l_loss: 0.1216 - val_accuracy: 0.9752
Epoch 11/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0461 - accuracy: 0.9860 - va
```

```
l_loss: 0.1012 - val_accuracy: 0.9801
Epoch 12/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0478 - accuracy: 0.9847 - va
l_loss: 0.1217 - val_accuracy: 0.9784
Epoch 13/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0389 - accuracy: 0.9877 - va
l_loss: 0.1106 - val_accuracy: 0.9817
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0416 - accuracy: 0.9879 - va
l_loss: 0.1341 - val_accuracy: 0.9790
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0392 - accuracy: 0.9873 - va
l_loss: 0.1147 - val_accuracy: 0.9820
Epoch 16/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0382 - accuracy: 0.9869 - va
l_loss: 0.1445 - val_accuracy: 0.9782
Epoch 17/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0303 - accuracy: 0.9893 - va
l_loss: 0.1444 - val_accuracy: 0.9773
Epoch 18/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0385 - accuracy: 0.9869 - va
l_loss: 0.1094 - val_accuracy: 0.9820
Epoch 19/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0268 - accuracy: 0.9903 - va
l_loss: 0.1085 - val_accuracy: 0.9825
Epoch 20/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0267 - accuracy: 0.9906 - va
l_loss: 0.1142 - val_accuracy: 0.9813
```

In [95]:

```
score = model4.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 56us/step
```

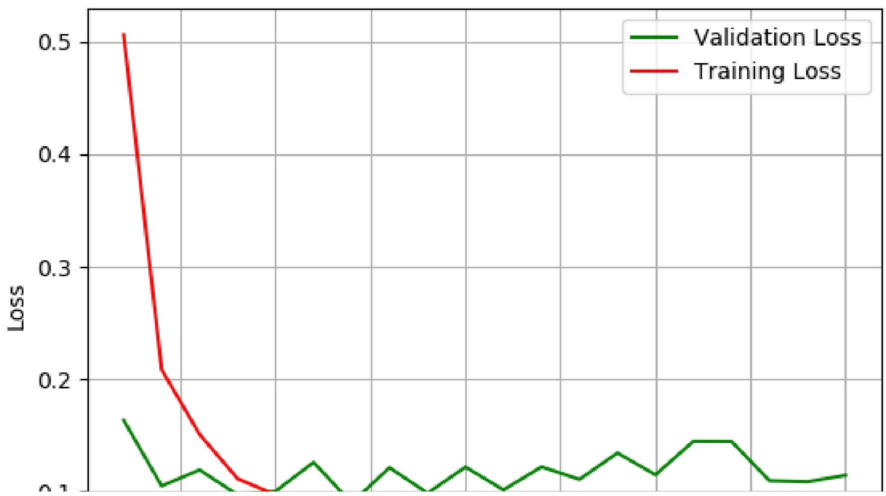
In [96]:

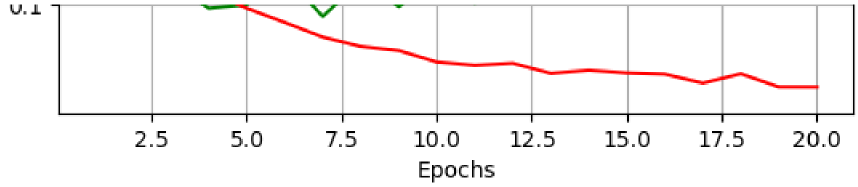
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```





In [97]:

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

```
Test accuracy: 0.9812999963760376
```

changing dropout rate to 0.2

In [55]:

```
model4 = Sequential()
model4.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model4.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model4.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model4.add(BatchNormalization())
model4.add(Dropout(0.2))
model4.add(Dense(output_dim, activation='softmax'))
print(model4.summary())
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_26 (Dense)             (None, 128)               46720     
_________________________________________________________________
dense_27 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256       
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_28 (Dense)             (None, 10)                650       
=================================================================
Total params: 341,622
Trainable params: 341,494
Non-trainable params: 128
_________________________________________________________________
None
```

In [56]:

```
model4.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [57]:

```
results = model4.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2278 - accuracy: 0.9344 - va
l_loss: 0.1169 - val_accuracy: 0.9688
Epoch 2/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0891 - accuracy: 0.9729 - va
l_loss: 0.1046 - val_accuracy: 0.9671
Epoch 3/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0589 - accuracy: 0.9817 - va
l_loss: 0.1284 - val_accuracy: 0.9610
```

```
Epoch 4/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0467 - accuracy: 0.9857 - va
l_loss: 0.0777 - val_accuracy: 0.9768
Epoch 5/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0368 - accuracy: 0.9892 - va
l_loss: 0.0656 - val_accuracy: 0.9808
Epoch 6/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0267 - accuracy: 0.9916 - va
l_loss: 0.0695 - val_accuracy: 0.9808
Epoch 7/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0252 - accuracy: 0.9920 - va
l_loss: 0.0647 - val_accuracy: 0.9817
Epoch 8/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0198 - accuracy: 0.9937 - va
l_loss: 0.0855 - val_accuracy: 0.9775
Epoch 9/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0182 - accuracy: 0.9942 - va
l_loss: 0.0880 - val_accuracy: 0.9767
Epoch 10/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0188 - accuracy: 0.9941 - va
l_loss: 0.0762 - val_accuracy: 0.9806
Epoch 11/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0134 - accuracy: 0.9957 - va
l_loss: 0.0832 - val_accuracy: 0.9790
Epoch 12/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0137 - accuracy: 0.9958 - va
l_loss: 0.0663 - val_accuracy: 0.9829
Epoch 13/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0092 - accuracy: 0.9972 - va
l_loss: 0.0887 - val_accuracy: 0.9787
Epoch 14/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0156 - accuracy: 0.9951 - va
l_loss: 0.0791 - val_accuracy: 0.9816
Epoch 15/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0101 - accuracy: 0.9969 - va
l_loss: 0.0879 - val_accuracy: 0.9806
Epoch 16/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0112 - accuracy: 0.9965 - va
l_loss: 0.0699 - val_accuracy: 0.9821
Epoch 17/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0095 - accuracy: 0.9970 - va
l_loss: 0.0826 - val_accuracy: 0.9814
Epoch 18/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0063 - accuracy: 0.9979 - va
l_loss: 0.0756 - val_accuracy: 0.9843
Epoch 19/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0093 - accuracy: 0.9973 - va
l_loss: 0.0848 - val_accuracy: 0.9791
Epoch 20/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0067 - accuracy: 0.9979 - va
l_loss: 0.0775 - val_accuracy: 0.9838
```

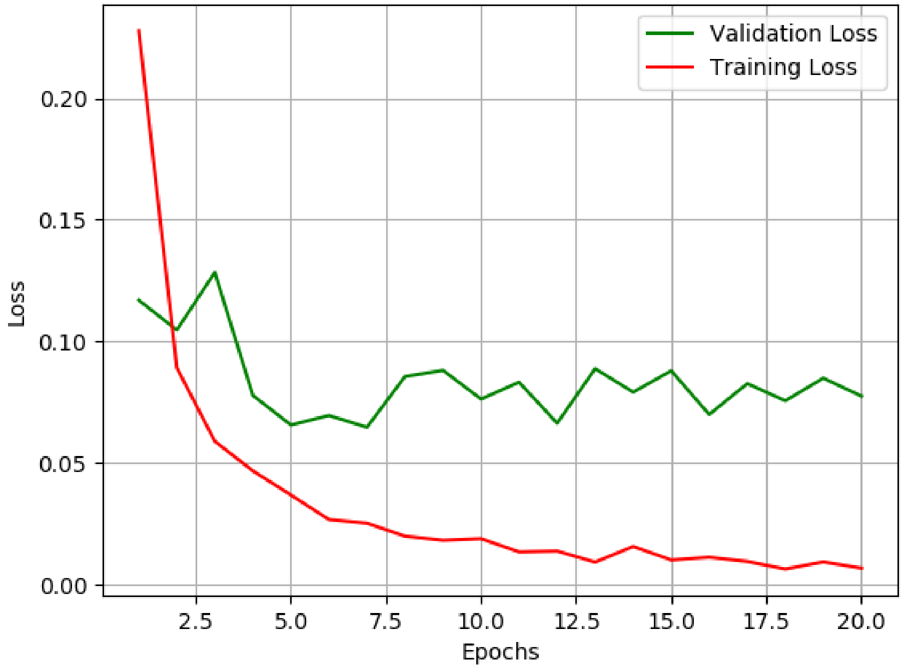
In [58]:

```
score = model4.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 55us/step
```

In [59]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')
list_of_epoch = list(range(1,nb_epoch+1))
train_loss = results.history['loss']
val_loss = results.history['val_loss']
ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [60]:

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

```
Test accuracy: 0.9837999939918518
```

Model with 5 hidden layers :

In [61]:

```
model5 = Sequential()
model5.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model5.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model5.add(Dense(96, activation='relu', kernel_initializer='random_uniform'))
model5.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model5.add(Dense(32, activation='relu', kernel_initializer='random_uniform'))
model5.add(Dense(output_dim, activation='softmax'))
print(model5.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_30 (Dense)             (None, 128)               46720
_________________________________________________________________
dense_31 (Dense)             (None, 96)                12384
_________________________________________________________________
dense_32 (Dense)             (None, 64)                6208
_________________________________________________________________
dense_33 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_34 (Dense)             (None, 10)                330
=================================================================
Total params: 353,462
Trainable params: 353,462
Non-trainable params: 0
_________________________________________________________________
None
```

In [62]:

```
model5.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model5.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

In [63]:

```
results = model5.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 30, valid
ation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
Epoch 2/20
Epoch 3/20
Epoch 4/20
Epoch 5/20
Epoch 6/20
Epoch 7/20
Epoch 8/20
Epoch 9/20
Epoch 10/20
Epoch 11/20
Epoch 12/20
Epoch 13/20
Epoch 14/20
Epoch 15/20
Epoch 16/20
Epoch 17/20
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

In [64]:

```
score = model5.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 63us/step
```

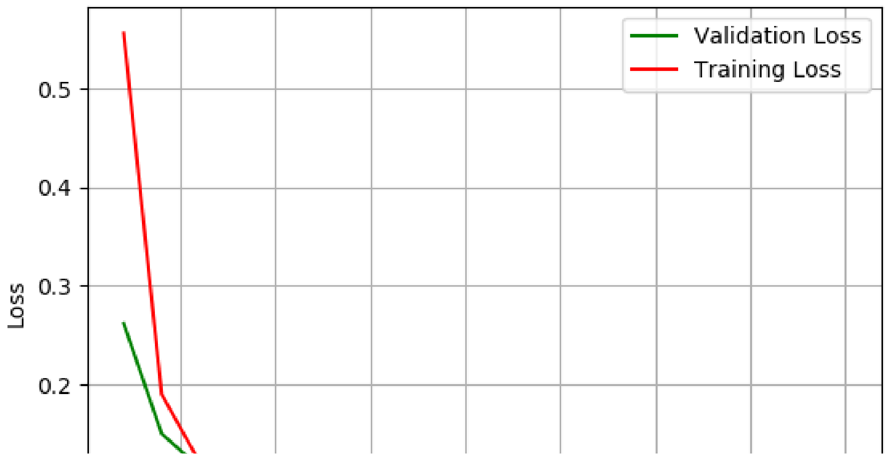
In [65]:

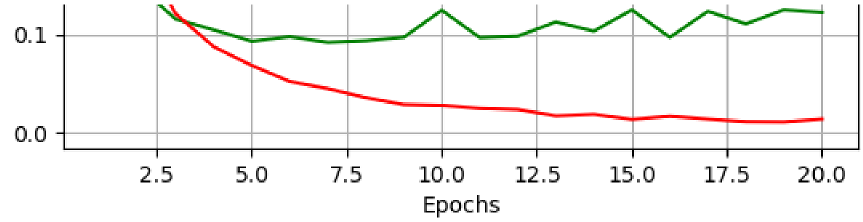
```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('Epochs') ; ax.set_ylabel('Loss')

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```





In [66]:

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

```
Test accuracy: 0.9768999814987183
```

Model With 5 hidden layers along with BN and Dropout

In [67]:

```
model6 = Sequential()
model6.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model6.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model6.add(Dense(96, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(32, activation='relu', kernel_initializer='random_uniform'))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))
model6.add(Dense(output_dim, activation='softmax'))
print(model6.summary())
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 364)               285740    
_________________________________________________________________
dense_36 (Dense)             (None, 128)               46720     
_________________________________________________________________
dense_37 (Dense)             (None, 96)                12384     
_________________________________________________________________
dense_38 (Dense)             (None, 64)                6208      
_________________________________________________________________
dense_39 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_7 (Batch (None, 32)                128       
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_40 (Dense)             (None, 10)                330       
=================================================================
Total params: 353,590
Trainable params: 353,526
Non-trainable params: 64
_________________________________________________________________
None
```

In [68]:

```
model6.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [69]:

```
results = model6.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 30, valid
ation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
Epoch 2/20
```

```
Epoch 3/20
Epoch 4/20
Epoch 5/20
Epoch 6/20
Epoch 7/20
Epoch 8/20
Epoch 9/20
Epoch 10/20
Epoch 11/20
Epoch 12/20
Epoch 13/20
Epoch 14/20
Epoch 15/20
Epoch 16/20
Epoch 17/20
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

In [70]:

```
score = model6.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 56us/step
```

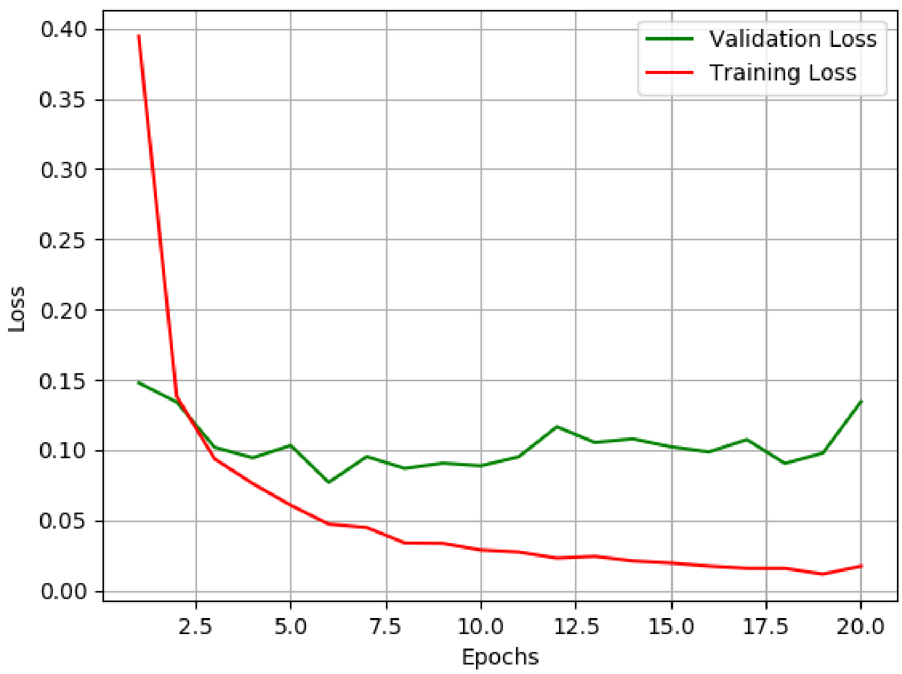
In [71]:

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

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [72]:

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

```
Test accuracy: 0.9768000245094299
```

Dropout to 0.8

In [73]:

```
model6 = Sequential()
model6.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model6.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model6.add(Dense(96, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(32, activation='relu', kernel_initializer='random_uniform'))
model6.add(BatchNormalization())
model6.add(Dropout(0.8))
model6.add(Dense(output_dim, activation='softmax'))
print(model6.summary())
```

```
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_42 (Dense)             (None, 128)               46720
_________________________________________________________________
dense_43 (Dense)             (None, 96)                12384
_________________________________________________________________
dense_44 (Dense)             (None, 64)                6208
_________________________________________________________________
dense_45 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_8 (Batch (None, 32)                128
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                330
=================================================================
Total params: 353,590
Trainable params: 353,526
Non-trainable params: 64
_________________________________________________________________
None
```

In [74]:

```
model6.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [75]:

```
results = model6.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.8172 - accuracy: 0.7139 - va
l_loss: 0.2154 - val_accuracy: 0.9497
Epoch 2/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.4396 - accuracy: 0.8383 - va
l_loss: 0.1347 - val_accuracy: 0.9660
Epoch 3/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.3578 - accuracy: 0.8597 - va
l_loss: 0.1160 - val_accuracy: 0.9722
Epoch 4/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.3197 - accuracy: 0.8700 - va
```

```
l_loss: 0.0984 - val_accuracy: 0.9756
Epoch 5/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.3013 - accuracy: 0.8733 - va
l_loss: 0.0975 - val_accuracy: 0.9775
Epoch 6/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2850 - accuracy: 0.8792 - va
l_loss: 0.1093 - val_accuracy: 0.9766
Epoch 7/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.2698 - accuracy: 0.8864 - va
l_loss: 0.1413 - val_accuracy: 0.9700
Epoch 8/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2617 - accuracy: 0.8892 - va
l_loss: 0.1271 - val_accuracy: 0.9759
Epoch 9/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.2481 - accuracy: 0.8956 - va
l_loss: 0.1172 - val_accuracy: 0.9761
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.2380 - accuracy: 0.8997 - va
l_loss: 0.1037 - val_accuracy: 0.9792
Epoch 11/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2433 - accuracy: 0.8986 - va
l_loss: 0.1231 - val_accuracy: 0.9787
Epoch 12/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.2241 - accuracy: 0.9064 - va
l_loss: 0.1119 - val_accuracy: 0.9817
Epoch 13/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.2055 - accuracy: 0.9132 - va
l_loss: 0.1059 - val_accuracy: 0.9821
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.2039 - accuracy: 0.9183 - va
l_loss: 0.1331 - val_accuracy: 0.9791
Epoch 15/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1944 - accuracy: 0.9209 - va
l_loss: 0.1355 - val_accuracy: 0.9786
Epoch 16/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1909 - accuracy: 0.9203 - va
l_loss: 0.1303 - val_accuracy: 0.9818
Epoch 17/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.1806 - accuracy: 0.9239 - va
l_loss: 0.1263 - val_accuracy: 0.9824
Epoch 18/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1823 - accuracy: 0.9261 - va
l_loss: 0.1418 - val_accuracy: 0.9819
Epoch 19/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.1870 - accuracy: 0.9252 - va
l_loss: 0.1468 - val_accuracy: 0.9812
Epoch 20/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.1823 - accuracy: 0.9261 - va
l_loss: 0.1471 - val_accuracy: 0.9811
```

In [76]:

```
score = model6.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 56us/step
```

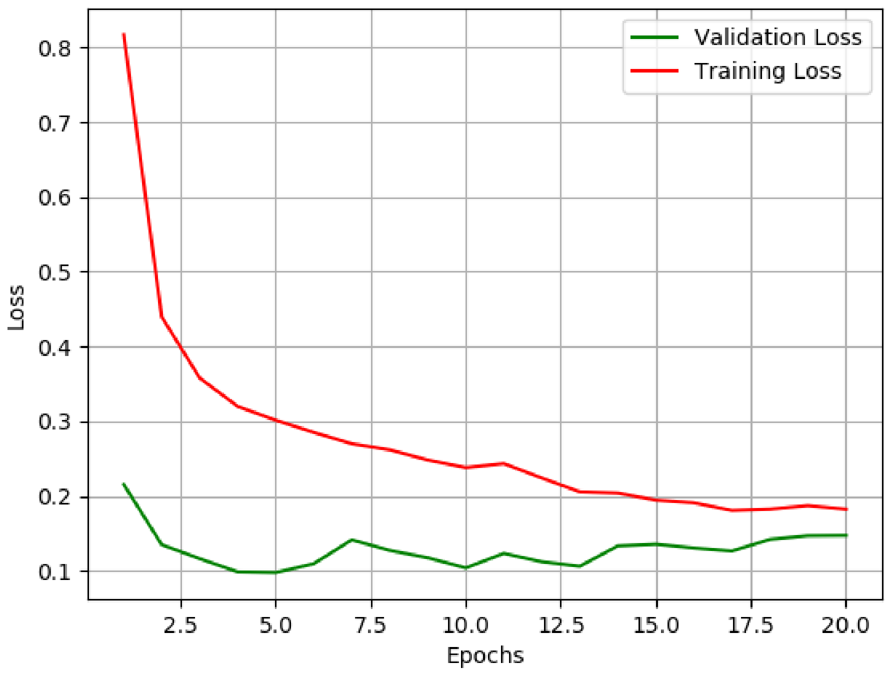
In [77]:

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

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [78]:

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

```
Test accuracy: 0.9811000227928162
```

DropOut to 0.2

In [79]:

```
model6 = Sequential()
model6.add(Dense(364, activation='relu', input_shape=(input_dim,),kernel_initializer='random_unifor
m'))
model6.add(Dense(128, activation='relu',kernel_initializer='random_uniform'))
model6.add(Dense(96, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(64, activation='relu', kernel_initializer='random_uniform'))
model6.add(Dense(32, activation='relu', kernel_initializer='random_uniform'))
model6.add(BatchNormalization())
model6.add(Dropout(0.2))
model6.add(Dense(output_dim, activation='softmax'))
print(model6.summary())
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 364)               285740
_________________________________________________________________
dense_48 (Dense)             (None, 128)               46720
_________________________________________________________________
dense_49 (Dense)             (None, 96)                12384
_________________________________________________________________
dense_50 (Dense)             (None, 64)                6208
_________________________________________________________________
dense_51 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_9 (Batch (None, 32)                128
_________________________________________________________________
dropout_9 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                330
=================================================================
Total params: 353,590
```

```
Trainable params: 353,526
Non-trainable params: 64
_________________________________________________________________
None
```

In [80]:

```
model6.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
```

In [81]:

```
results = model6.fit(X_train, y_train, batch_size = batch_size, epochs=nb_epoch, verbose= 1, valida
tion_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.3328 - accuracy: 0.9092 - va
l_loss: 0.1275 - val_accuracy: 0.9672
Epoch 2/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.1045 - accuracy: 0.9708 - va
l_loss: 0.0985 - val_accuracy: 0.9708
Epoch 3/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0696 - accuracy: 0.9801 - va
l_loss: 0.1003 - val_accuracy: 0.9707
Epoch 4/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0529 - accuracy: 0.9850 - va
l_loss: 0.0821 - val_accuracy: 0.9755
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0427 - accuracy: 0.9882 - va
l_loss: 0.0860 - val_accuracy: 0.9768
Epoch 6/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0341 - accuracy: 0.9898 - va
l_loss: 0.0892 - val_accuracy: 0.9767
Epoch 7/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0269 - accuracy: 0.9923 - va
l_loss: 0.0939 - val_accuracy: 0.9757
Epoch 8/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0264 - accuracy: 0.9919 - va
l_loss: 0.0809 - val_accuracy: 0.9806
Epoch 9/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0229 - accuracy: 0.9932 - va
l_loss: 0.0822 - val_accuracy: 0.9785
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0192 - accuracy: 0.9944 - va
l_loss: 0.0980 - val_accuracy: 0.9785
Epoch 11/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0191 - accuracy: 0.9942 - va
l_loss: 0.0891 - val_accuracy: 0.9797
Epoch 12/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0164 - accuracy: 0.9954 - va
l_loss: 0.0855 - val_accuracy: 0.9800
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0143 - accuracy: 0.9957 - va
l_loss: 0.0841 - val_accuracy: 0.9807
Epoch 14/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0161 - accuracy: 0.9954 - va
l_loss: 0.0692 - val_accuracy: 0.9836
Epoch 15/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0127 - accuracy: 0.9959 - va
l_loss: 0.0786 - val_accuracy: 0.9819
Epoch 16/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0136 - accuracy: 0.9958 - va
l_loss: 0.0804 - val_accuracy: 0.9813
Epoch 17/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0119 - accuracy: 0.9965 - va
l_loss: 0.0800 - val_accuracy: 0.9823
Epoch 18/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0112 - accuracy: 0.9968 - va
l_loss: 0.0887 - val_accuracy: 0.9810
Epoch 19/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0074 - accuracy: 0.9978 - va
l_loss: 0.1007 - val_accuracy: 0.9787
Epoch 20/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0137 - accuracy: 0.9961 - va
l_loss: 0.0786 - val_accuracy: 0.9826
```

In [82]:

```
score = model6.evaluate(X_test, y_test)
```

```
10000/10000 [==============================] - 1s 55us/step
```

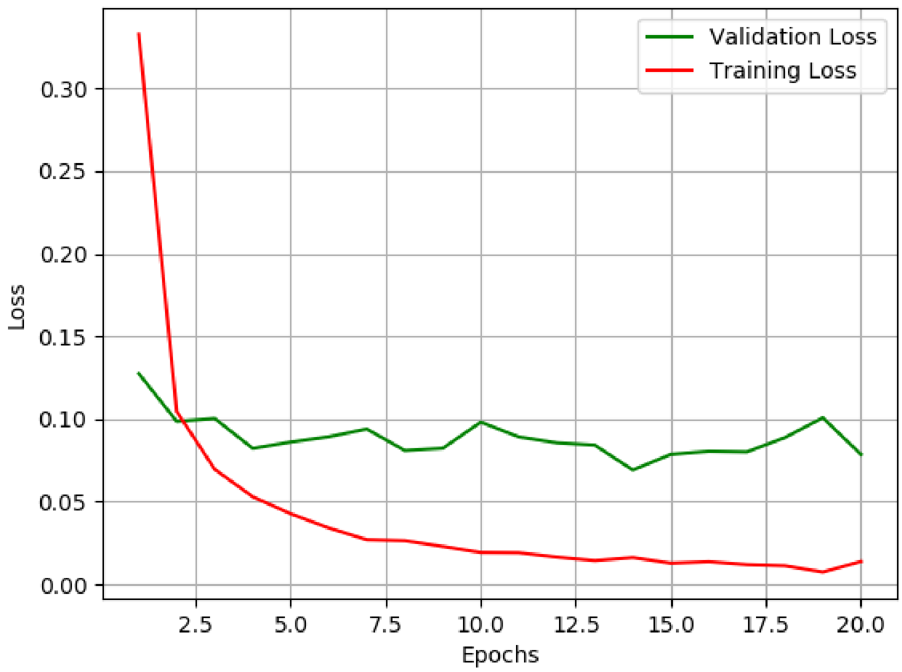
In [83]:

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

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

train_loss = results.history['loss']
val_loss = results.history['val_loss']

ax.plot(list_of_epoch, val_loss, 'g', label="Validation Loss")
ax.plot(list_of_epoch, train_loss, 'r', label="Training Loss")
plt.legend()
plt.grid()
plt.show();
```



In [84]:

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

```
Test accuracy: 0.9825999736785889
```

Conclusions

In [85]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Numer of Layers", "BN","Dropout", "Accuracy"]
x.add_row(["2", 'NO',"NO", 0.981])
```

```
x.add_row(["2", 'YES',0.5, 0.982])
x.add_row(["2", 'YES',0.8, 0.982])
x.add_row(["2", 'YES',0.2, 0.981])
x.add_row(["3", 'NO',"NO", 0.971])
x.add_row(["3", 'YES',0.5, 0.976])
x.add_row(["3", 'YES',0.8, 0.980])
x.add_row(["3", 'YES',0.2, 0.983])
x.add_row(["5", 'NO','NO', 0.976])
x.add_row(["5", 'YES',0.5, 0.976])
x.add_row(["5", 'YES',0.8, 0.981])
x.add_row(["5", 'YES',0.2, 0.982])
print(x)
```

```
+------------------+-----+---------+----------+
| Numer of Layers |  BN | Dropout | Accuracy |
+------------------+-----+---------+----------+
|        2         |  NO |    NO   |  0.981   |
|        2         | YES |   0.5   |  0.982   |
|        2         | YES |   0.8   |  0.982   |
|        2         | YES |   0.2   |  0.981   |
|        3         |  NO |    NO   |  0.971   |
|        3         | YES |   0.5   |  0.976   |
|        3         | YES |   0.8   |   0.98   |
|        3         | YES |   0.2   |  0.983   |
|        5         |  NO |    NO   |  0.976   |
|        5         | YES |   0.5   |  0.976   |
|        5         | YES |   0.8   |  0.981   |
|        5         | YES |   0.2   |  0.982   |
+------------------+-----+---------+----------+
```