Keras -- MLPs on MNIST

In [1]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensor
flow" use this command
from  keras.utils import np_utils
from keras.datasets import mnist
from keras.initializers import RandomNormal
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

In [0]:
```
import seaborn as sns
```

In [0]:
```
%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()
```

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

In [4]:
```
print("Number of training examples :", X_train.shape[0], X_train.shape[1],
X_train.shape[2])
```

```
Number of training examples : 60000 28 28
```

In [5]:
```
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 [0]:
```
# 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 [8]:
```
# 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 [9]:
```
# An example data point
print(X_train[0])
```

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

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

X_train = X_train/255
x_test = x_test/255
```

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

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

```
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.31764706 0.94117647 0.99215686
 0.99215686 0.46666667 0.09803922 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
 0.58823529 0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

In [11]:
```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

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

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

print("After converting the output into a vector : ", type(y_train))
```

```
Class label of first image : 5
After converting the output into a vector :  <class 'numpy.ndarray'>
```

In [12]:
```
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 10)
(10000, 10)
```

Softmax classifier

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

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

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

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

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

###

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

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initiali
zer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

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

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

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through th
e activation argument supported by all forward layers:

# from keras.layers import Activation, Dense

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

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

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

```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [0]:
```
# start building a model
model = Sequential()

# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape infe
rence)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

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

In [0]:
```
# Before training a model, you need to configure the learning process, whic
h is done via the compile method

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimize
r , https://keras.io/optimizers/
# A loss function. This is the objective that the model will try to minimiz
e., https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set th
is to metrics=['accuracy'].  https://keras.io/metrics/


# Note: when using the categorical_crossentropy loss, your targets should b
e in categorical format
# (e.g. if you have 10 classes, the target for each sample should be a 10-d
imensional vector that is all-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['a
ccuracy'])

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks
=None, validation_split=0.0,
# validation_data=None, shuffle=True, class_weight=None, sample_weight=Non
e, initial_epoch=0, steps_per_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations
 on a dataset).

# it returns A History object. Its History.history attribute is a record of
training loss values and
# metrics values at successive epochs, as well as validation loss values an
d validation metrics values (if applicable).

# https://github.com/openai/baselines/issues/20

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoc
h, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 49us/step - loss: 1.2935
- acc: 0.6829 - val_loss: 0.8171 - val_acc: 0.8312
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.7213
- acc: 0.8385 - val_loss: 0.6099 - val_acc: 0.8623
Epoch 3/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5904
- acc: 0.8584 - val_loss: 0.5275 - val_acc: 0.8741
Epoch 4/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5278
- acc: 0.8682 - val_loss: 0.4815 - val_acc: 0.8814
Epoch 5/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4897
- acc: 0.8747 - val_loss: 0.4515 - val_acc: 0.8870
Epoch 6/20
34432/60000 [================>.............] - ETA: 0s - loss: 0.4697 - ac
c: 0.877360000/60000 [==============================] - 2s 35us/step - los
s: 0.4635 - acc: 0.8799 - val_loss: 0.4303 - val_acc: 0.8898
Epoch 7/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4442
- acc: 0.8837 - val_loss: 0.4138 - val_acc: 0.8917
Epoch 8/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4290
- acc: 0.8866 - val_loss: 0.4010 - val_acc: 0.8952
Epoch 9/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4169
- acc: 0.8894 - val_loss: 0.3909 - val_acc: 0.8971
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4067
- acc: 0.8911 - val_loss: 0.3818 - val_acc: 0.8994
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3982
- acc: 0.8926 - val_loss: 0.3743 - val_acc: 0.9006
Epoch 12/20
 1792/60000 [..............................] - ETA: 1s - loss: 0.4077 - ac
c: 0.889560000/60000 [==============================] - 2s 35us/step - los
s: 0.3908 - acc: 0.8948 - val_loss: 0.3681 - val_acc: 0.9020
Epoch 13/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3844
- acc: 0.8960 - val_loss: 0.3624 - val_acc: 0.9039
Epoch 14/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3786
- acc: 0.8972 - val_loss: 0.3575 - val_acc: 0.9046
Epoch 15/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3735
- acc: 0.8981 - val_loss: 0.3528 - val_acc: 0.9058
Epoch 16/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3689
- acc: 0.8993 - val_loss: 0.3490 - val_acc: 0.9062
Epoch 17/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3648
- acc: 0.9004 - val_loss: 0.3455 - val_acc: 0.9066
Epoch 18/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3610
- acc: 0.9016 - val_loss: 0.3419 - val_acc: 0.9077
```

```
Epoch 19/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3575
- acc: 0.9024 - val_loss: 0.3389 - val_acc: 0.9088
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.3544
- acc: 0.9032 - val_loss: 0.3362 - val_acc: 0.9093
```

In [0]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.3362289469957352
Test accuracy: 0.9093
```

MLP + Sigmoid activation + SGDOptimizer

In [0]:
```
# Multilayer perceptron

model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim
,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', met
rics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs
=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 42us/step - loss: 2.2708
- acc: 0.2169 - val_loss: 2.2197 - val_acc: 0.2768
Epoch 2/20
60000/60000 [==============================] - 2s 42us/step - loss: 2.1739
- acc: 0.4237 - val_loss: 2.1143 - val_acc: 0.4877
Epoch 3/20
60000/60000 [==============================] - 2s 42us/step - loss: 2.0506
- acc: 0.5485 - val_loss: 1.9659 - val_acc: 0.5524
Epoch 4/20
60000/60000 [==============================] - 3s 42us/step - loss: 1.8796
- acc: 0.6135 - val_loss: 1.7673 - val_acc: 0.6481
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 1.6674
- acc: 0.6659 - val_loss: 1.5414 - val_acc: 0.7205
Epoch 6/20
 5376/60000 [=>............................] - ETA: 2s - loss: 1.5430 - ac
c: 0.698160000/60000 [==============================] - 2s 41us/step - los
s: 1.4449 - acc: 0.7125 - val_loss: 1.3233 - val_acc: 0.7513
Epoch 7/20
60000/60000 [==============================] - 2s 41us/step - loss: 1.2442
- acc: 0.7466 - val_loss: 1.1406 - val_acc: 0.7599
Epoch 8/20
60000/60000 [==============================] - 2s 41us/step - loss: 1.0815
- acc: 0.7722 - val_loss: 0.9974 - val_acc: 0.7968
Epoch 9/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.9544
- acc: 0.7940 - val_loss: 0.8867 - val_acc: 0.8060
Epoch 10/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.8556
- acc: 0.8082 - val_loss: 0.7984 - val_acc: 0.8227
Epoch 11/20
31232/60000 [==============>...............] - ETA: 1s - loss: 0.7920 - ac
c: 0.820760000/60000 [==============================] - 2s 42us/step - los
s: 0.7776 - acc: 0.8225 - val_loss: 0.7292 - val_acc: 0.8335
Epoch 12/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.7154
- acc: 0.8333 - val_loss: 0.6741 - val_acc: 0.8422
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.6650
- acc: 0.8412 - val_loss: 0.6282 - val_acc: 0.8500
Epoch 14/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.6237
- acc: 0.8485 - val_loss: 0.5906 - val_acc: 0.8585
Epoch 15/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.5893
- acc: 0.8546 - val_loss: 0.5591 - val_acc: 0.8616
Epoch 16/20
34432/60000 [================>.............] - ETA: 0s - loss: 0.5691 - ac
c: 0.857860000/60000 [==============================] - 2s 41us/step - los
s: 0.5606 - acc: 0.8599 - val_loss: 0.5329 - val_acc: 0.8672
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.5361
- acc: 0.8641 - val_loss: 0.5095 - val_acc: 0.8703
Epoch 18/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.5151
```

```
- acc: 0.8676 - val_loss: 0.4904 - val_acc: 0.8736
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.4969
- acc: 0.8710 - val_loss: 0.4732 - val_acc: 0.8782
Epoch 20/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.4810
- acc: 0.8739 - val_loss: 0.4583 - val_acc: 0.8816
```

In [0]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.4582893396139145
Test accuracy: 0.8816
```

In [0]:
```
w_after = model_sigmoid.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWa
rning: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWa
rning: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```

MLP + Sigmoid activation + ADAM

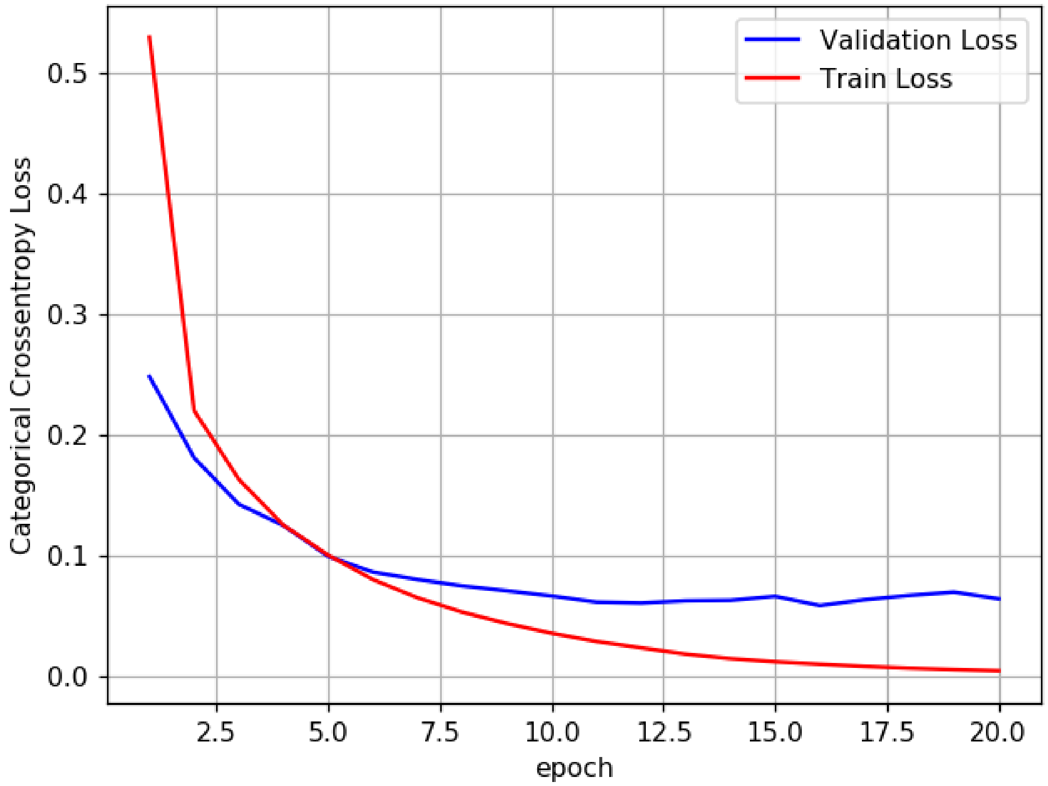
In [0]:
```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim
,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()

model_sigmoid.compile(optimizer='adam', loss='categorical_crossentropy', me
trics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs
=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_6 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.5308
- acc: 0.8636 - val_loss: 0.2550 - val_acc: 0.9259
Epoch 2/20
53120/60000 [=========================>....] - ETA: 0s - loss: 0.2244 - ac
c: 0.934060000/60000 [==============================] - 3s 51us/step - los
s: 0.2205 - acc: 0.9351 - val_loss: 0.1946 - val_acc: 0.9417
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1650
- acc: 0.9512 - val_loss: 0.1421 - val_acc: 0.9570
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1279
- acc: 0.9614 - val_loss: 0.1238 - val_acc: 0.9645
Epoch 5/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.1010
- acc: 0.9704 - val_loss: 0.1029 - val_acc: 0.9693
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0801
- acc: 0.9763 - val_loss: 0.0877 - val_acc: 0.9725
Epoch 7/20
 4480/60000 [=>............................] - ETA: 2s - loss: 0.0632 - ac
c: 0.979560000/60000 [==============================] - 3s 51us/step - los
s: 0.0645 - acc: 0.9809 - val_loss: 0.0831 - val_acc: 0.9751
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0519
- acc: 0.9842 - val_loss: 0.0724 - val_acc: 0.9780
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0433
- acc: 0.9872 - val_loss: 0.0714 - val_acc: 0.9786
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0347
- acc: 0.9898 - val_loss: 0.0695 - val_acc: 0.9776
Epoch 11/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0268
- acc: 0.9930 - val_loss: 0.0659 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0219
- acc: 0.9944 - val_loss: 0.0642 - val_acc: 0.9809
Epoch 13/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0180
- acc: 0.9953 - val_loss: 0.0677 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0133
```

```
- acc: 0.9970 - val_loss: 0.0647 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0114
- acc: 0.9975 - val_loss: 0.0628 - val_acc: 0.9812
Epoch 16/20
57344/60000 [===========================>..] - ETA: 0s - loss: 0.0085 - ac
c: 0.998260000/60000 [==============================] - 3s 50us/step - los
s: 0.0085 - acc: 0.9982 - val_loss: 0.0666 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0070
- acc: 0.9986 - val_loss: 0.0643 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0061
- acc: 0.9986 - val_loss: 0.0656 - val_acc: 0.9818
Epoch 19/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0055
- acc: 0.9988 - val_loss: 0.0811 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0038
- acc: 0.9992 - val_loss: 0.0723 - val_acc: 0.9818
```

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.06385514608082886
Test accuracy: 0.9824
```



In [0]:
```
w_after = model_sigmoid.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWa
rning: remove_na is deprecated and is a private function. Do not use.
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWa
rning: remove_na is deprecated and is a private function. Do not use.
  violin_data = remove_na(group_data)
```

MLP + ReLU +SGD

In [0]:
```
# Multilayer perceptron

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy this co
ndition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.062  => N(0,σ) = N(0,0.062)
# h2 =>  σ=√(2/(fan_in) = 0.125  => N(0,σ) = N(0,0.125)
# out =>  σ=√(2/(fan_in+1) = 0.120  => N(0,σ) = N(0,0.120)

model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kern
el_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNorma
l(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

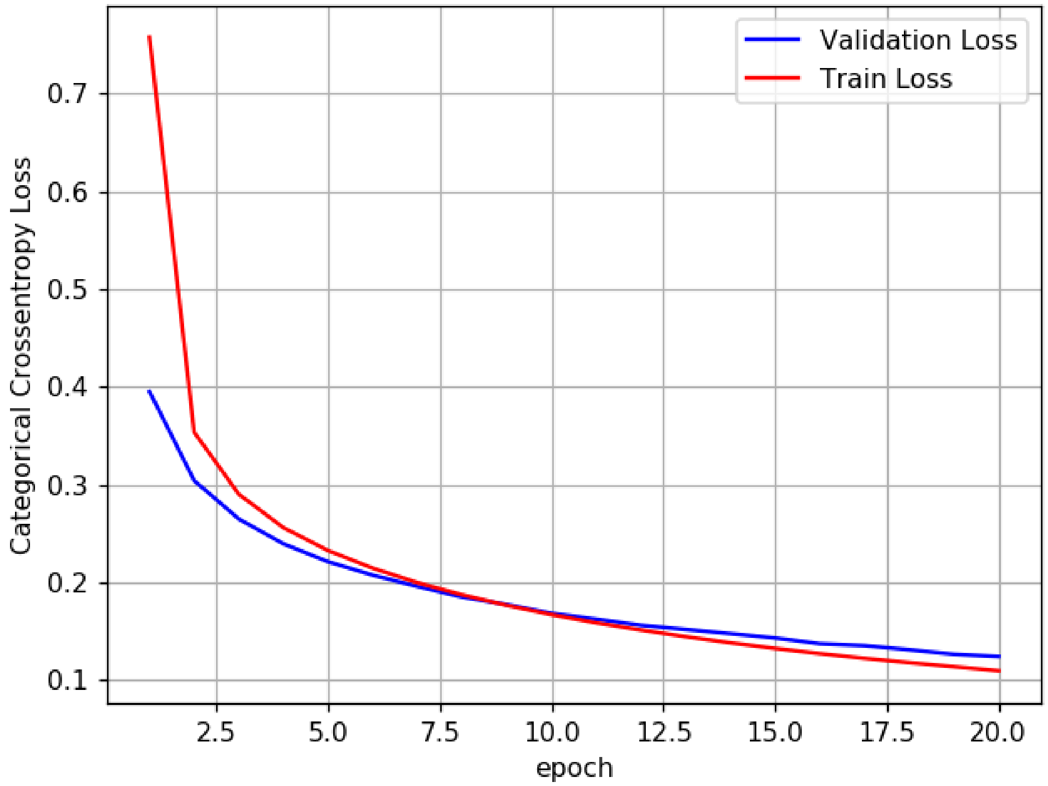
In [0]:
```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy', metric
s=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb
_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.7579
- acc: 0.7812 - val_loss: 0.3951 - val_acc: 0.8921
Epoch 2/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.3535
- acc: 0.8998 - val_loss: 0.3040 - val_acc: 0.9153
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2900
- acc: 0.9172 - val_loss: 0.2648 - val_acc: 0.9253
Epoch 4/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.2558
- acc: 0.9269 - val_loss: 0.2393 - val_acc: 0.9316
Epoch 5/20
60000/60000 [==============================] - 4s 58us/step - loss: 0.2324
- acc: 0.9340 - val_loss: 0.2210 - val_acc: 0.9371
Epoch 6/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.2144
- acc: 0.9391 - val_loss: 0.2072 - val_acc: 0.9400
Epoch 7/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1995
- acc: 0.9443 - val_loss: 0.1957 - val_acc: 0.9444
Epoch 8/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.1872
- acc: 0.9476 - val_loss: 0.1848 - val_acc: 0.9456
Epoch 9/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.1763
- acc: 0.9507 - val_loss: 0.1771 - val_acc: 0.9488
Epoch 10/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1668
- acc: 0.9539 - val_loss: 0.1682 - val_acc: 0.9506
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1585
- acc: 0.9560 - val_loss: 0.1623 - val_acc: 0.9518
Epoch 12/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.1511
- acc: 0.9577 - val_loss: 0.1560 - val_acc: 0.9543
Epoch 13/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.1443
- acc: 0.9596 - val_loss: 0.1517 - val_acc: 0.9557
Epoch 14/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.1379
- acc: 0.9615 - val_loss: 0.1474 - val_acc: 0.9572
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1323
- acc: 0.9628 - val_loss: 0.1429 - val_acc: 0.9580
Epoch 16/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.1270
- acc: 0.9645 - val_loss: 0.1371 - val_acc: 0.9598
Epoch 17/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.1221
- acc: 0.9661 - val_loss: 0.1351 - val_acc: 0.9602
Epoch 18/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.1177
- acc: 0.9671 - val_loss: 0.1309 - val_acc: 0.9618
Epoch 19/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.1136
```

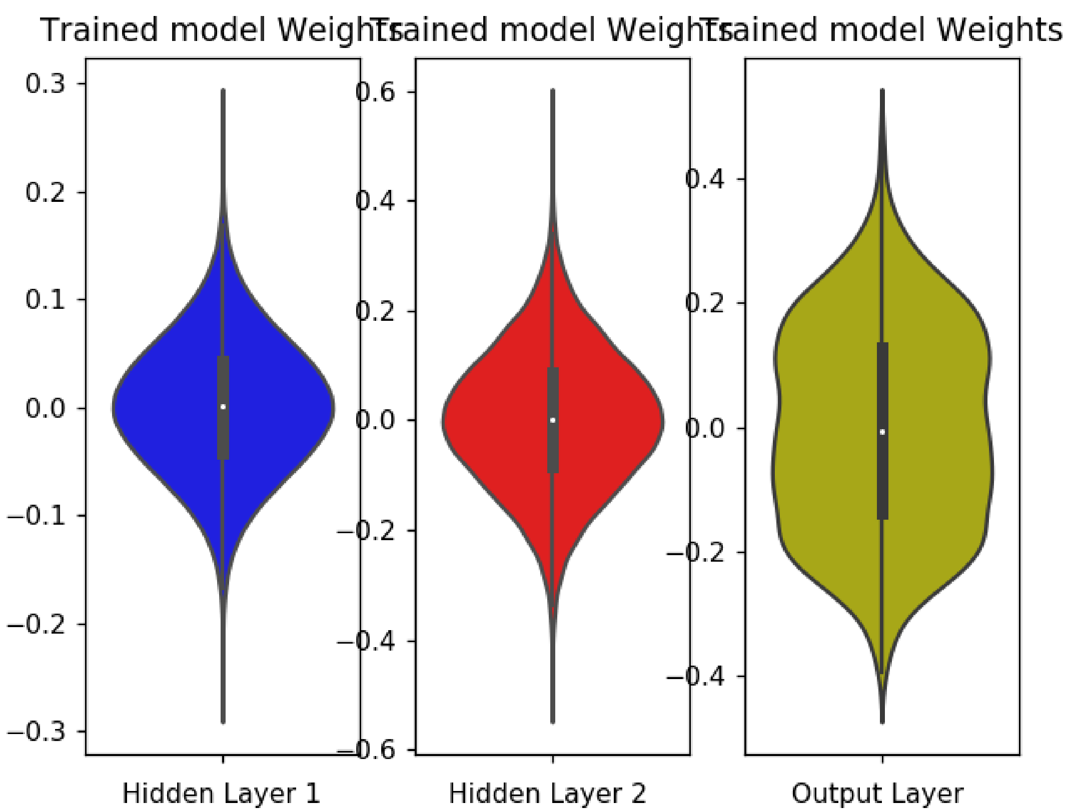
```
- acc: 0.9685 - val_loss: 0.1263 - val_acc: 0.9631
Epoch 20/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1094
- acc: 0.9694 - val_loss: 0.1241 - val_acc: 0.9631
```

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.12405014228336513
Test accuracy: 0.9631
```



In [0]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + ReLU + ADAM

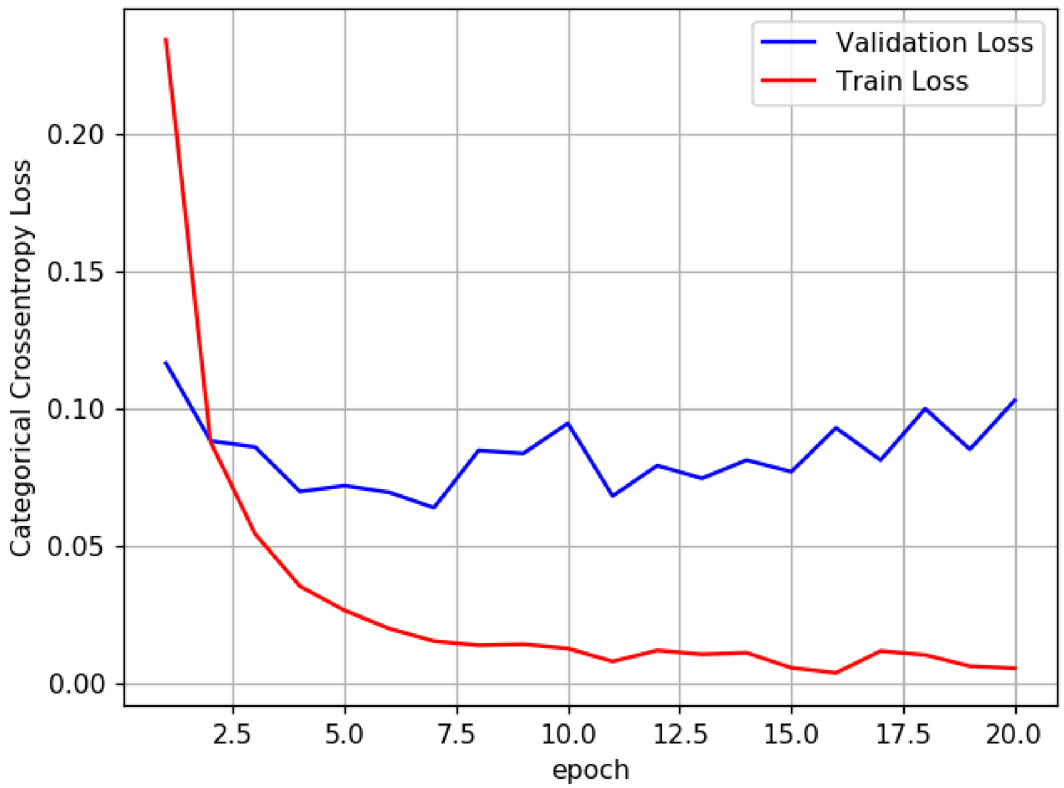
In [0]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kern
el_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNorma
l(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb
_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_12 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.2341
- acc: 0.9295 - val_loss: 0.1165 - val_acc: 0.9652
Epoch 2/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0878
- acc: 0.9729 - val_loss: 0.0883 - val_acc: 0.9720
Epoch 3/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0544
- acc: 0.9825 - val_loss: 0.0860 - val_acc: 0.9729
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0354
- acc: 0.9885 - val_loss: 0.0699 - val_acc: 0.9797
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0266
- acc: 0.9914 - val_loss: 0.0720 - val_acc: 0.9788
Epoch 6/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0200
- acc: 0.9941 - val_loss: 0.0696 - val_acc: 0.9803
Epoch 7/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0155
- acc: 0.9951 - val_loss: 0.0640 - val_acc: 0.9829
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0140
- acc: 0.9952 - val_loss: 0.0848 - val_acc: 0.9792
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0143
- acc: 0.9952 - val_loss: 0.0837 - val_acc: 0.9796
Epoch 10/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0128
- acc: 0.9958 - val_loss: 0.0946 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0081
- acc: 0.9974 - val_loss: 0.0682 - val_acc: 0.9826
Epoch 12/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0121
- acc: 0.9959 - val_loss: 0.0793 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0107
- acc: 0.9963 - val_loss: 0.0746 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0113
- acc: 0.9960 - val_loss: 0.0813 - val_acc: 0.9816
```

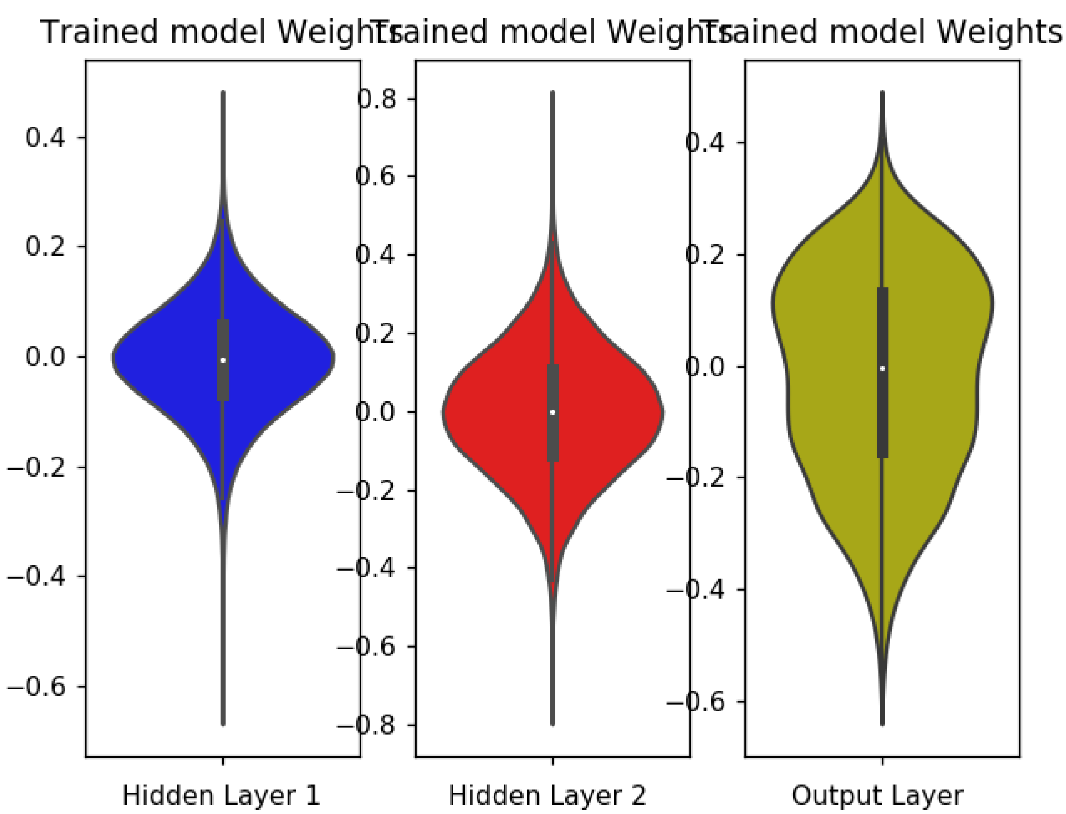
```
Epoch 15/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0058
- acc: 0.9982 - val_loss: 0.0770 - val_acc: 0.9842
Epoch 16/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0040
- acc: 0.9987 - val_loss: 0.0930 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0119
- acc: 0.9959 - val_loss: 0.0813 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0105
- acc: 0.9966 - val_loss: 0.1000 - val_acc: 0.9803
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0064
- acc: 0.9981 - val_loss: 0.0852 - val_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0056
- acc: 0.9982 - val_loss: 0.1029 - val_acc: 0.9805
```

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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


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

```
Test score: 0.10294274219236926
Test accuracy: 0.9805
```



In [0]:
```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



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

In [0]:
```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(128, activation='sigmoid', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

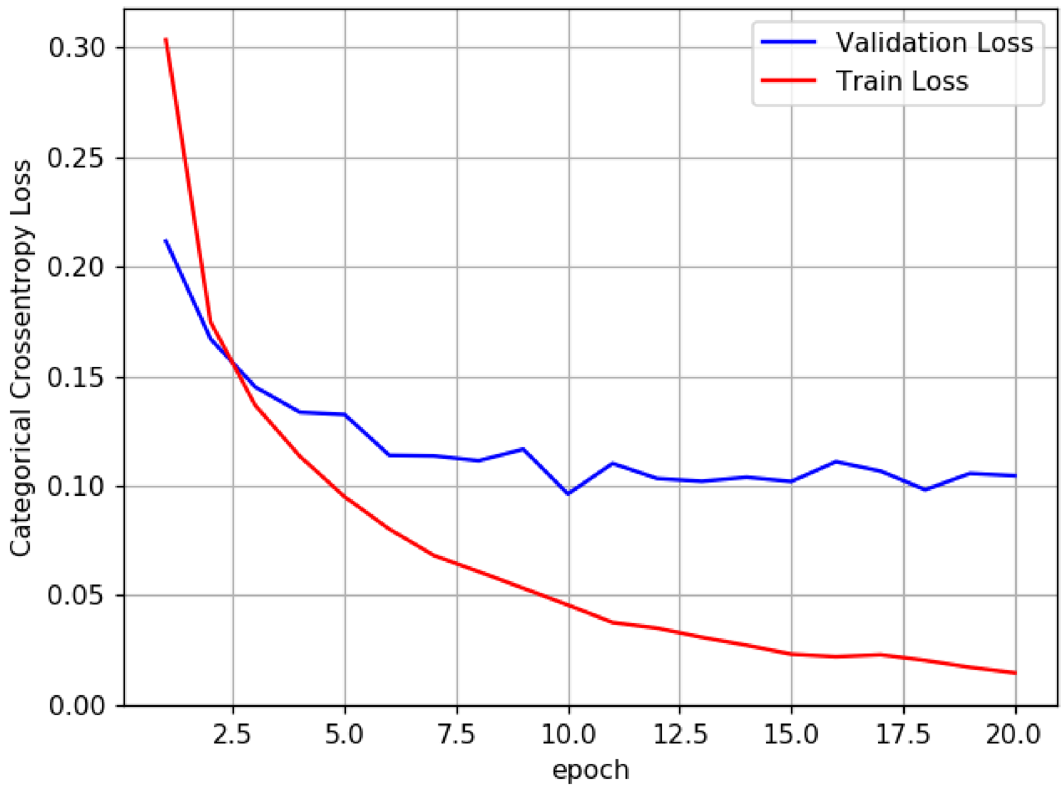
In [0]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.3036
- acc: 0.9104 - val_loss: 0.2116 - val_acc: 0.9376
Epoch 2/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.174
7 - acc: 0.9483 - val_loss: 0.1670 - val_acc: 0.9505
Epoch 3/20
60000/60000 [==============================] - 13s 220us/step - loss: 0.136
7 - acc: 0.9599 - val_loss: 0.1451 - val_acc: 0.9567
Epoch 4/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.1134
- acc: 0.9666 - val_loss: 0.1335 - val_acc: 0.9603
Epoch 5/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.094
9 - acc: 0.9703 - val_loss: 0.1325 - val_acc: 0.9589
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0802
- acc: 0.9758 - val_loss: 0.1139 - val_acc: 0.9652
Epoch 7/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0682
- acc: 0.9787 - val_loss: 0.1136 - val_acc: 0.9666
Epoch 8/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0608
- acc: 0.9815 - val_loss: 0.1114 - val_acc: 0.9666
Epoch 9/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0532
- acc: 0.9837 - val_loss: 0.1167 - val_acc: 0.9666
Epoch 10/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0455
- acc: 0.9856 - val_loss: 0.0962 - val_acc: 0.9718
Epoch 11/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0376
- acc: 0.9880 - val_loss: 0.1102 - val_acc: 0.9673
Epoch 12/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0350
- acc: 0.9889 - val_loss: 0.1033 - val_acc: 0.9710
Epoch 13/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0308
- acc: 0.9903 - val_loss: 0.1020 - val_acc: 0.9712
Epoch 14/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0271
- acc: 0.9913 - val_loss: 0.1038 - val_acc: 0.9727
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0231
- acc: 0.9926 - val_loss: 0.1019 - val_acc: 0.9717
Epoch 16/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0220
- acc: 0.9928 - val_loss: 0.1110 - val_acc: 0.9703
Epoch 17/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0229
- acc: 0.9928 - val_loss: 0.1067 - val_acc: 0.9739
Epoch 18/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0203
- acc: 0.9935 - val_loss: 0.0982 - val_acc: 0.9738
Epoch 19/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0171
```

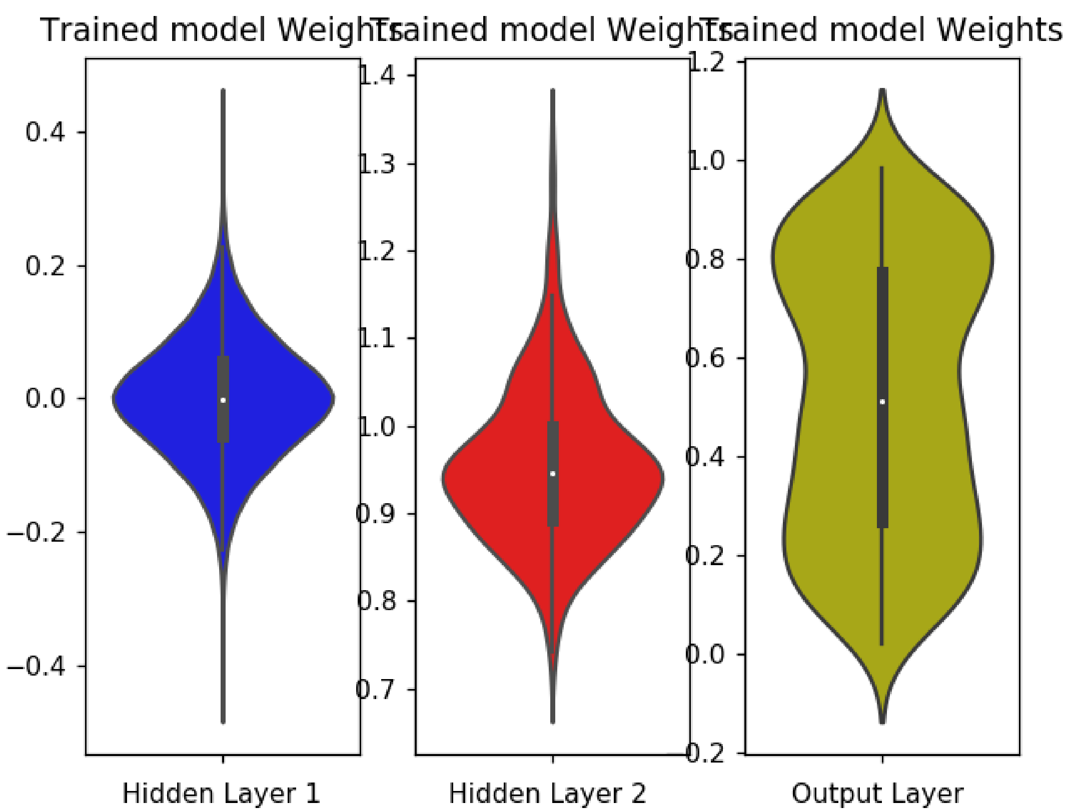
```
- acc: 0.9944 - val_loss: 0.1056 - val_acc: 0.9706
Epoch 20/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.014
6 - acc: 0.9952 - val_loss: 0.1046 - val_acc: 0.9732
```

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.10456635547156475
Test accuracy: 0.9732
```



In [0]:
```
w_after = model_batch.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



5. MLP + Dropout + AdamOptimizer

In [0]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnor
malization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), k
ernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

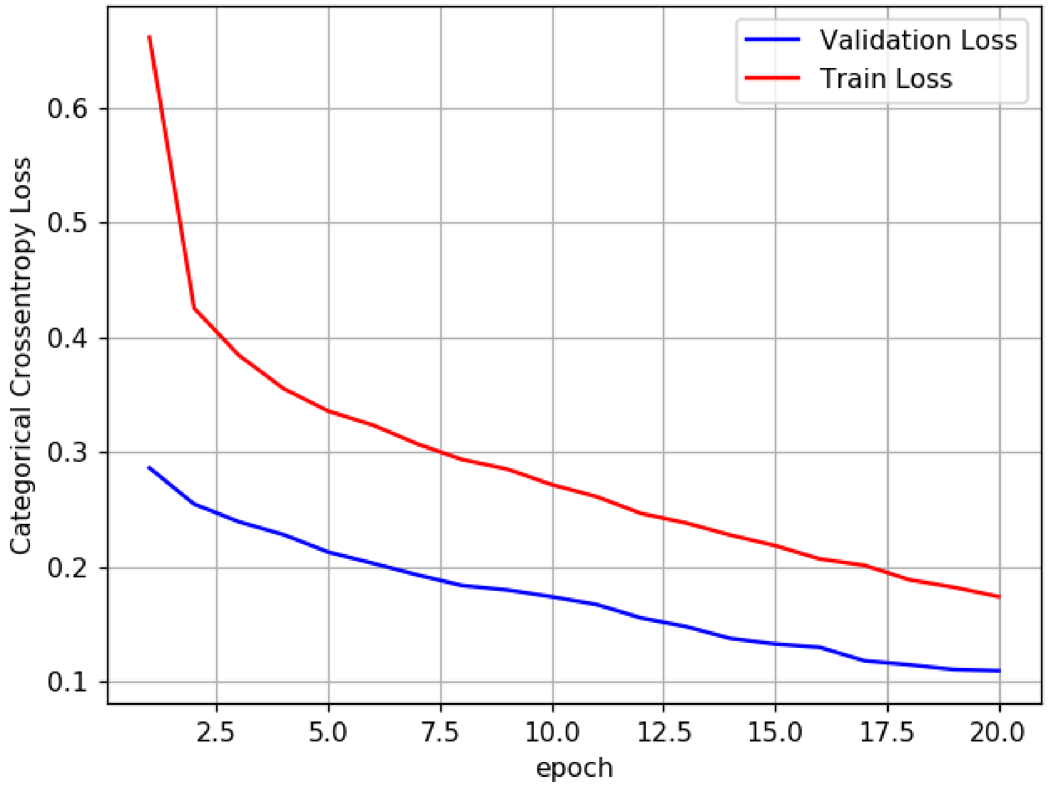
In [0]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metri
cs=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb
_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.661
2 - acc: 0.7951 - val_loss: 0.2860 - val_acc: 0.9166
Epoch 2/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.4250
- acc: 0.8710 - val_loss: 0.2545 - val_acc: 0.9252
Epoch 3/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.384
1 - acc: 0.8846 - val_loss: 0.2391 - val_acc: 0.9298
Epoch 4/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.3551
- acc: 0.8927 - val_loss: 0.2279 - val_acc: 0.9325
Epoch 5/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.3355
- acc: 0.8986 - val_loss: 0.2127 - val_acc: 0.9356
Epoch 6/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.3234
- acc: 0.9031 - val_loss: 0.2029 - val_acc: 0.9387: 1s - loss:
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.3068
- acc: 0.9077 - val_loss: 0.1927 - val_acc: 0.9421
Epoch 8/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.293
3 - acc: 0.9113 - val_loss: 0.1836 - val_acc: 0.9453
Epoch 9/20
60000/60000 [==============================] - 13s 222us/step - loss: 0.285
0 - acc: 0.9131 - val_loss: 0.1797 - val_acc: 0.9451
Epoch 10/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.271
5 - acc: 0.9187 - val_loss: 0.1738 - val_acc: 0.9465
Epoch 11/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.2611
- acc: 0.9214 - val_loss: 0.1671 - val_acc: 0.9506
Epoch 12/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2464
- acc: 0.9252 - val_loss: 0.1554 - val_acc: 0.9525
Epoch 13/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2382
- acc: 0.9278 - val_loss: 0.1479 - val_acc: 0.9554
Epoch 14/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2275
- acc: 0.9313 - val_loss: 0.1375 - val_acc: 0.9580
Epoch 15/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2183
- acc: 0.9337 - val_loss: 0.1326 - val_acc: 0.9599
Epoch 16/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2068
- acc: 0.9384 - val_loss: 0.1297 - val_acc: 0.9613 loss: 0.2066 - ac
Epoch 17/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.2011
- acc: 0.9395 - val_loss: 0.1181 - val_acc: 0.9646
Epoch 18/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1886
- acc: 0.9435 - val_loss: 0.1145 - val_acc: 0.9658
Epoch 19/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.1821
```

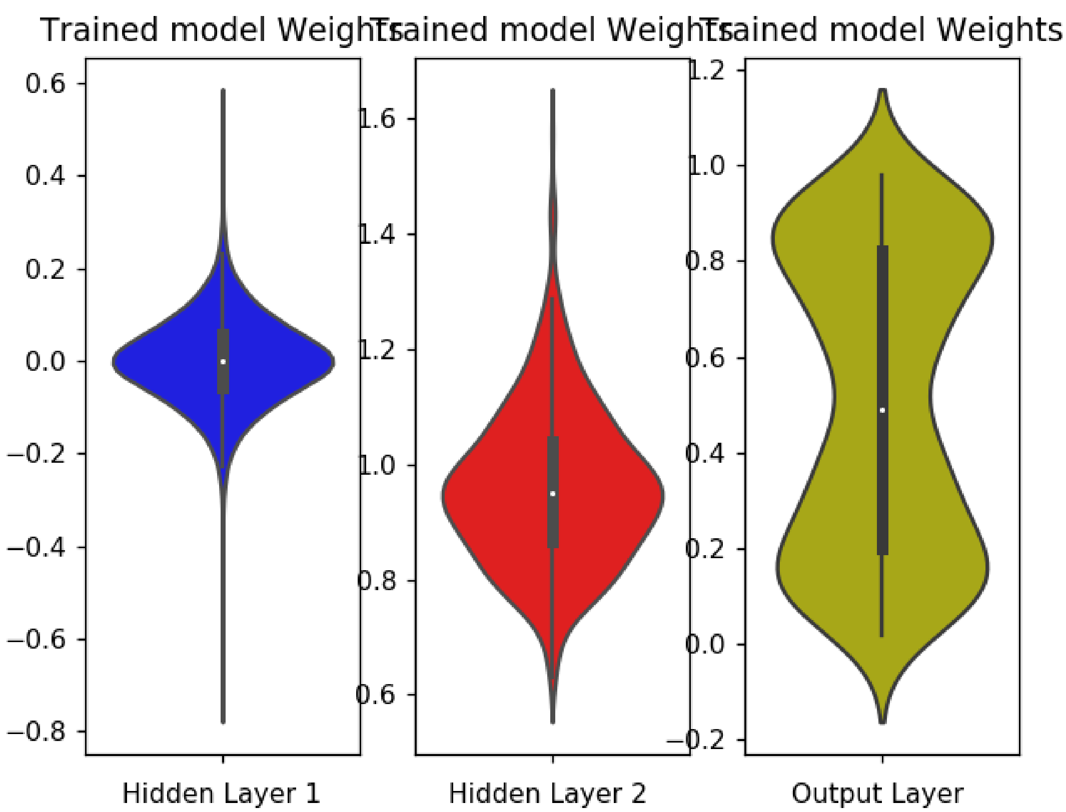
```
- acc: 0.9451 - val_loss: 0.1104 - val_acc: 0.9662
Epoch 20/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1739
- acc: 0.9473 - val_loss: 0.1093 - val_acc: 0.9679
```

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=
nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.1093290721397847
Test accuracy: 0.9679
```



In [0]:
```
w_after = model_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



Hyper-parameter tuning of Keras models using Sklearn

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

    model = Sequential()
    model.add(Dense(512, activation=activ, input_shape=(input_dim,), kernel
_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
    model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(
mean=0.0, stddev=0.125, seed=None)) )
    model.add(Dense(output_dim, activation='softmax'))


    model.compile(loss='categorical_crossentropy', metrics=['accuracy'], op
timizer='adam')

    return model
```

In [0]:
```
# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learn
ing-models-python-keras/

activ = ['sigmoid','relu']

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

model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch, bat
ch_size=batch_size, verbose=0)
param_grid = dict(activ=activ)

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

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

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

```
Best: 0.975633 using {'activ': 'relu'}
0.974650 (0.001138) with: {'activ': 'sigmoid'}
0.975633 (0.002812) with: {'activ': 'relu'}
```

2-layer MLP + Relu + adam

No Droput and Batch normalization

In [0]:
```
%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_mdl_res(x_val, trn_los, tst_los, tst_scr, tst_acc):
  # Visualize loss history
  plt.figure(figsize=(16,16))
  plt.plot(x_val, trn_los, 'r--')
  plt.plot(x_val, tst_los, 'b-')
  plt.legend(['Training Loss', 'Test Loss'])
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.grid()
  plt.show();

  print('Test score:', tst_scr)
  print('Test accuracy:', tst_acc)
```

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20
```

In [0]:
```
#MLP
mdl_relu = Sequential()
mdl_relu.add(Dense(512, activation='relu',input_shape=(input_dim,),kernel_i
nitializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu.add(Dense(128, activation='relu',kernel_initializer=RandomNormal(m
ean=0.0,stddev=0.150, seed=42)))
mdl_relu.add(Dense(output_dim,activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Pl
ease use tf.compat.v1.get_default_graph instead.
```

In [0]:
```
mdl_relu.summary()
print(y_train.shape, y_test.shape)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
(60000, 10) (10000, 10)
```

In [0]:
```
#set optimizer and loss
mdl_relu.compile(optimizer='adam',loss='categorical_crossentropy', metrics=
['accuracy'])
```

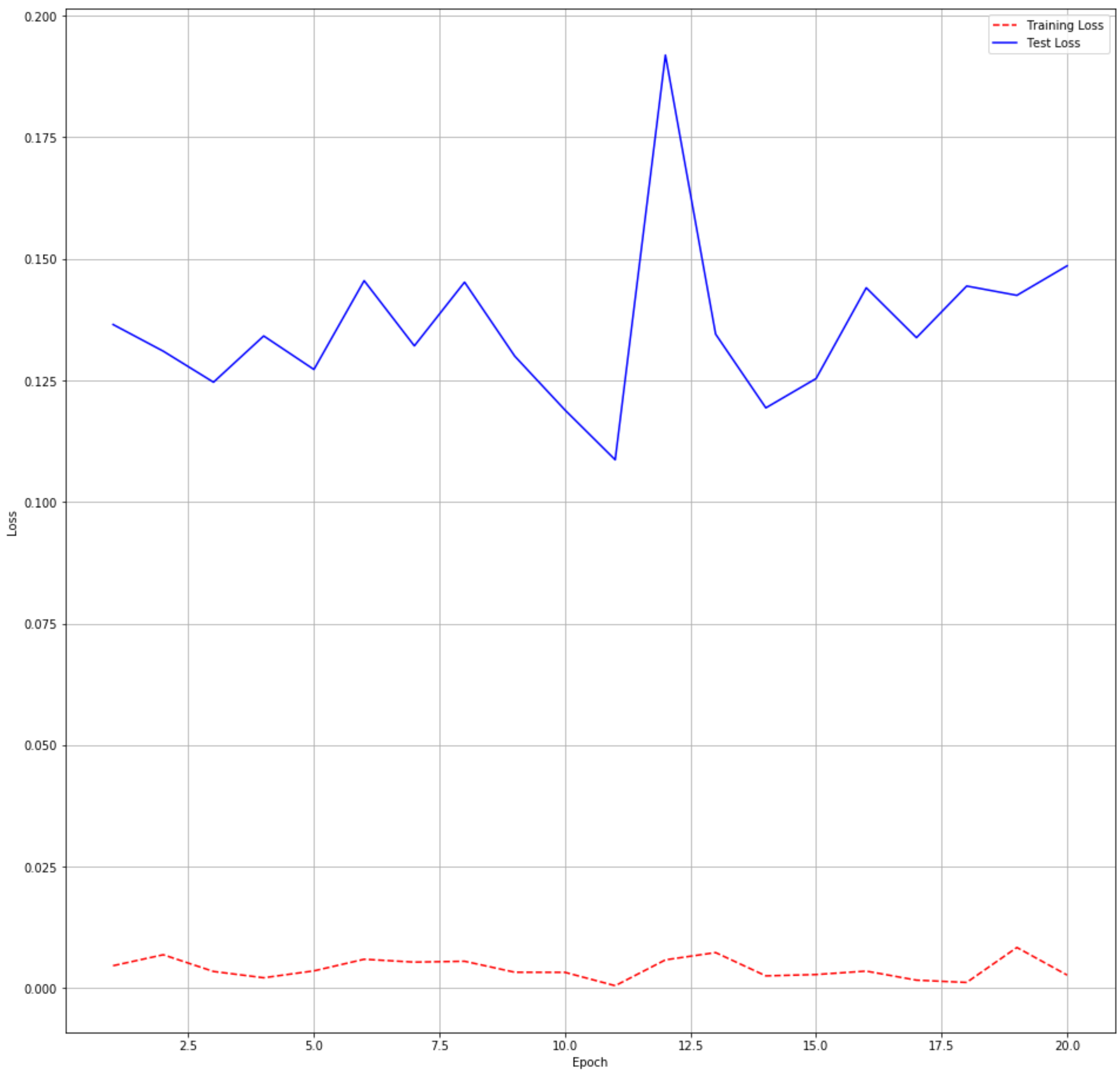
In [0]:
```
history = mdl_relu.fit(X_train,y_train,batch_size=batch_size,epochs=no_epoc
h,verbose=1,validation_data=(x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0046
- acc: 0.9987 - val_loss: 0.1365 - val_acc: 0.9821
Epoch 2/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0069
- acc: 0.9981 - val_loss: 0.1310 - val_acc: 0.9809
Epoch 3/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0034
- acc: 0.9989 - val_loss: 0.1246 - val_acc: 0.9818
Epoch 4/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0021
- acc: 0.9993 - val_loss: 0.1341 - val_acc: 0.9814
Epoch 5/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0036
- acc: 0.9990 - val_loss: 0.1272 - val_acc: 0.9814
Epoch 6/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0059
- acc: 0.9985 - val_loss: 0.1455 - val_acc: 0.9788
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0054
- acc: 0.9986 - val_loss: 0.1321 - val_acc: 0.9821
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0055
- acc: 0.9985 - val_loss: 0.1452 - val_acc: 0.9799
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0033
- acc: 0.9990 - val_loss: 0.1300 - val_acc: 0.9821
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0033
- acc: 0.9992 - val_loss: 0.1189 - val_acc: 0.9833
Epoch 11/20
60000/60000 [==============================] - 3s 52us/step - loss: 5.0557e
-04 - acc: 0.9999 - val_loss: 0.1087 - val_acc: 0.9840
Epoch 12/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0058
- acc: 0.9985 - val_loss: 0.1919 - val_acc: 0.9756
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0073
- acc: 0.9983 - val_loss: 0.1345 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0025
- acc: 0.9993 - val_loss: 0.1193 - val_acc: 0.9834
Epoch 15/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0028
- acc: 0.9992 - val_loss: 0.1254 - val_acc: 0.9825
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0035
- acc: 0.9991 - val_loss: 0.1440 - val_acc: 0.9816
Epoch 17/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0016
- acc: 0.9995 - val_loss: 0.1338 - val_acc: 0.9820
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0012
- acc: 0.9996 - val_loss: 0.1444 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0084
```

```
- acc: 0.9981 - val_loss: 0.1425 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.0027
- acc: 0.9993 - val_loss: 0.1486 - val_acc: 0.9796
```

In [0]:

```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Create count of the number of epochs
score = mdl_relu.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.14856573140081578
Test accuracy: 0.9796
```

2-Layer MLP + ReLu + Adam + Dropout

No Batch Normalization

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20
```

In [0]:
```
#MLP
mdl_relu_2 = Sequential()
mdl_relu_2.add(Dense(512, activation='relu',input_shape=(input_dim,),kernel
_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_2.add(Dropout(rate=0.5))
mdl_relu_2.add(Dense(128, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_2.add(Dropout(rate=0.2))
mdl_relu_2.add(Dense(output_dim,activation='softmax'))
```

In [0]:
```
mdl_relu_2.summary()
print(y_train.shape, y_test.shape)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
(60000, 10) (10000, 10)
```

In [0]:
```
#set optimizer and loss
mdl_relu_2.compile(optimizer='adam',loss='categorical_crossentropy', metric
s=['accuracy'])
```

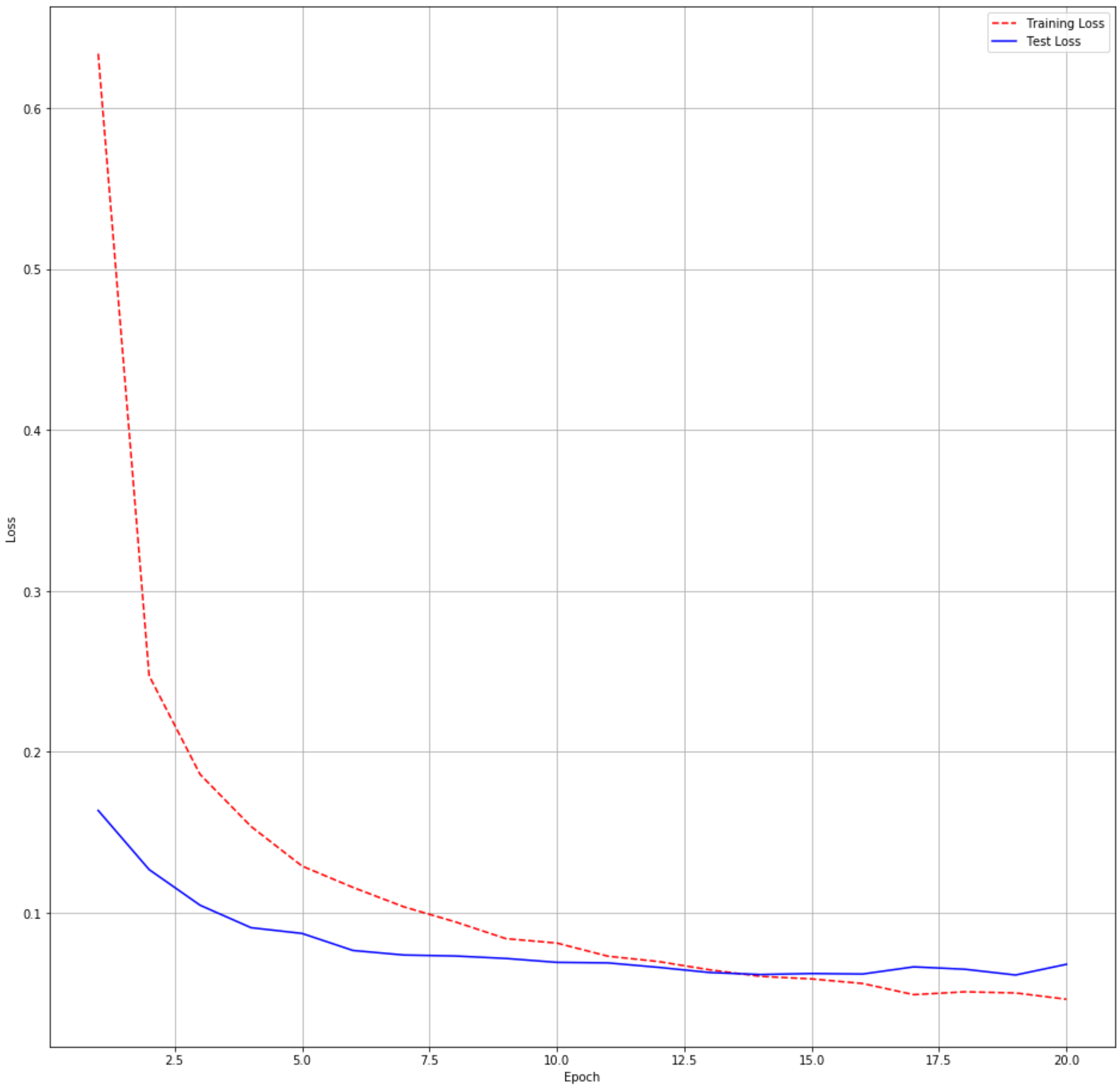
In [0]:
```
history_2 = mdl_relu_2.fit(X_train,y_train,batch_size=batch_size,epochs=no_
epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.6340
- acc: 0.8135 - val_loss: 0.1636 - val_acc: 0.9517
Epoch 2/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.2473
- acc: 0.9236 - val_loss: 0.1268 - val_acc: 0.9604
Epoch 3/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1860
- acc: 0.9437 - val_loss: 0.1047 - val_acc: 0.9671
Epoch 4/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1536
- acc: 0.9528 - val_loss: 0.0907 - val_acc: 0.9712
Epoch 5/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.1290
- acc: 0.9602 - val_loss: 0.0871 - val_acc: 0.9737
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1157
- acc: 0.9640 - val_loss: 0.0765 - val_acc: 0.9759
Epoch 7/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.1036
- acc: 0.9676 - val_loss: 0.0737 - val_acc: 0.9774
Epoch 8/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0944
- acc: 0.9699 - val_loss: 0.0731 - val_acc: 0.9789
Epoch 9/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0839
- acc: 0.9732 - val_loss: 0.0716 - val_acc: 0.9800
Epoch 10/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0811
- acc: 0.9746 - val_loss: 0.0692 - val_acc: 0.9788
Epoch 11/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0730
- acc: 0.9764 - val_loss: 0.0688 - val_acc: 0.9798
Epoch 12/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0696
- acc: 0.9781 - val_loss: 0.0660 - val_acc: 0.9807
Epoch 13/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0645
- acc: 0.9790 - val_loss: 0.0628 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0604
- acc: 0.9810 - val_loss: 0.0616 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0589
- acc: 0.9807 - val_loss: 0.0622 - val_acc: 0.9826
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.0560
- acc: 0.9819 - val_loss: 0.0619 - val_acc: 0.9820
Epoch 17/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0491
- acc: 0.9843 - val_loss: 0.0663 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0508
- acc: 0.9836 - val_loss: 0.0649 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.0502
```

```
- acc: 0.9839 - val_loss: 0.0613 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0462
- acc: 0.9852 - val_loss: 0.0680 - val_acc: 0.9816
```

In [0]:

```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_2.history['loss']
test_loss = history_2.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_2.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.06795536802747083
Test accuracy: 0.9816
```

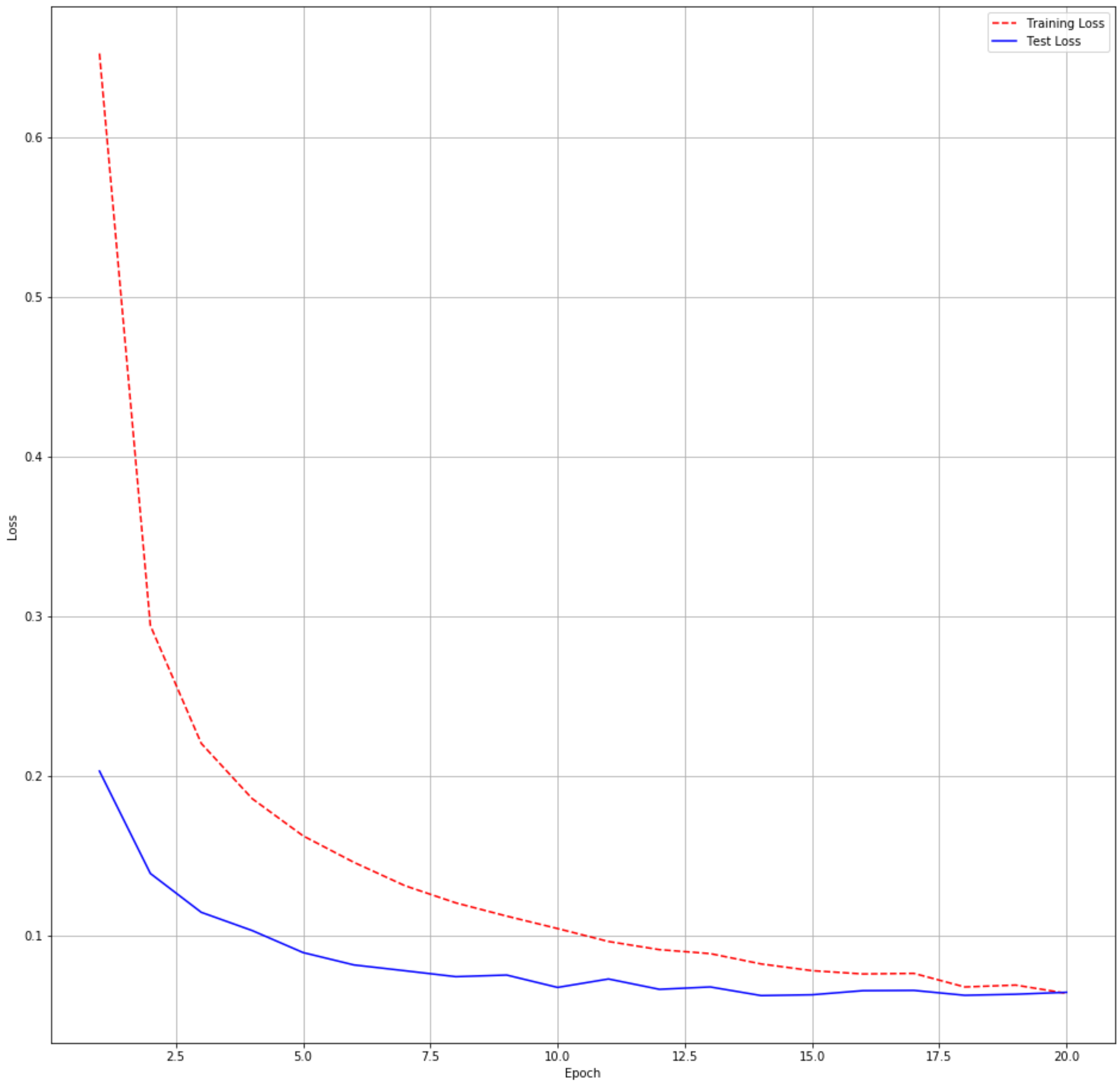
2-Layer MLP + ReLu + Adam + Dropout + Batch Normalization

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20

#MLP
mdl_relu_3 = Sequential()
mdl_relu_3.add(Dense(512, activation='relu',input_shape=(input_dim,),kernel
_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_3.add(Dropout(rate=0.5))
mdl_relu_3.add(BatchNormalization())
mdl_relu_3.add(Dense(128, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_3.add(Dropout(rate=0.2))
mdl_relu_3.add(BatchNormalization())
mdl_relu_3.add(Dense(output_dim,activation='softmax'))

mdl_relu_3.summary()
mdl_relu_3.compile(optimizer='adam',loss='categorical_crossentropy', metric
s=['accuracy'])
history_3 = mdl_relu_3.fit(X_train,y_train,batch_size=batch_size,epochs=no_
epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_15 (Dropout)         (None, 512)               0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048      
_________________________________________________________________
dense_26 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0         
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512       
_________________________________________________________________
dense_27 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.5043
- acc: 0.8422 - val_loss: 0.1645 - val_acc: 0.9514
Epoch 2/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2239
- acc: 0.9333 - val_loss: 0.1166 - val_acc: 0.9645
Epoch 3/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1709
- acc: 0.9476 - val_loss: 0.0951 - val_acc: 0.9696
Epoch 4/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.1390
- acc: 0.9577 - val_loss: 0.0830 - val_acc: 0.9743
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1246
- acc: 0.9616 - val_loss: 0.0769 - val_acc: 0.9757
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1081
- acc: 0.9663 - val_loss: 0.0716 - val_acc: 0.9772
Epoch 7/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0996
- acc: 0.9687 - val_loss: 0.0690 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0889
- acc: 0.9719 - val_loss: 0.0658 - val_acc: 0.9798
Epoch 9/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0841
- acc: 0.9733 - val_loss: 0.0595 - val_acc: 0.9821
Epoch 10/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0760
- acc: 0.9752 - val_loss: 0.0597 - val_acc: 0.9812
Epoch 11/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0788
- acc: 0.9747 - val_loss: 0.0597 - val_acc: 0.9818
Epoch 12/20
```

```
60000/60000 [==============================] - 5s 87us/step - loss: 0.0722
- acc: 0.9765 - val_loss: 0.0592 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0667
- acc: 0.9789 - val_loss: 0.0584 - val_acc: 0.9824
Epoch 14/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0623
- acc: 0.9795 - val_loss: 0.0585 - val_acc: 0.9827
Epoch 15/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0608
- acc: 0.9805 - val_loss: 0.0529 - val_acc: 0.9847
Epoch 16/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0555
- acc: 0.9815 - val_loss: 0.0559 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0569
- acc: 0.9815 - val_loss: 0.0545 - val_acc: 0.9825
Epoch 18/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0510
- acc: 0.9831 - val_loss: 0.0544 - val_acc: 0.9842
Epoch 19/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0503
- acc: 0.9836 - val_loss: 0.0559 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0504
- acc: 0.9838 - val_loss: 0.0561 - val_acc: 0.9831
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_3.history['loss']
test_loss = history_3.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_3.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.06436893670770805
Test accuracy: 0.9809
```

2-Layer MLP + Adam + Relu + Batch Normalization

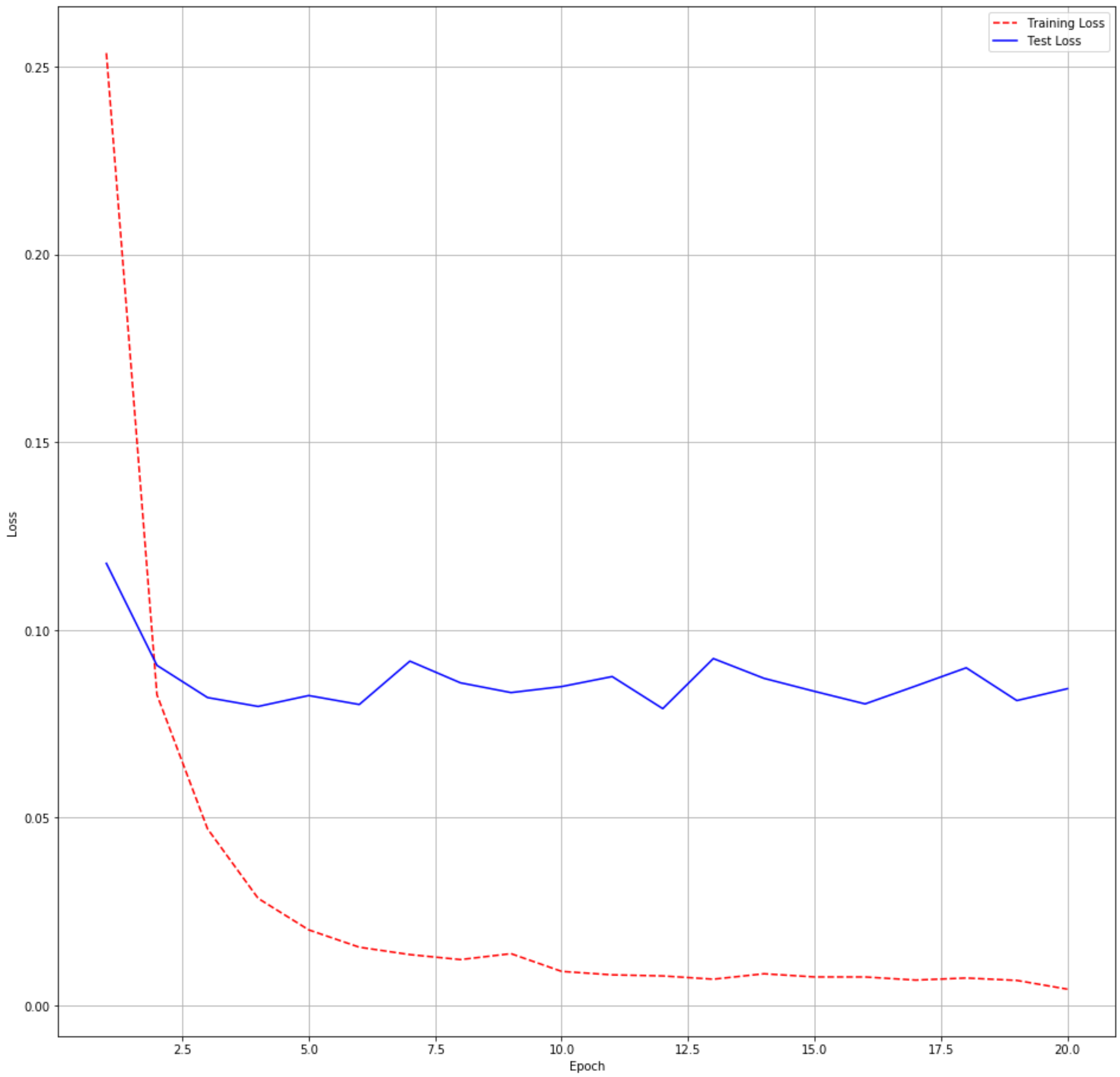
No Dropout

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20

#MLP
mdl_relu_4 = Sequential()
mdl_relu_4.add(Dense(512, activation='relu',input_shape=(input_dim,),kernel
_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_4.add(BatchNormalization())
mdl_relu_4.add(Dense(128, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_4.add(BatchNormalization())
mdl_relu_4.add(Dense(output_dim,activation='softmax'))

mdl_relu_4.summary()
mdl_relu_4.compile(optimizer='adam',loss='categorical_crossentropy', metric
s=['accuracy'])
history_4 = mdl_relu_4.fit(X_train,y_train,batch_size=batch_size,epochs=no_
epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_34 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_13 (Batc (None, 512)               2048      
_________________________________________________________________
dense_35 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_14 (Batc (None, 128)               512       
_________________________________________________________________
dense_36 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.2537
- acc: 0.9250 - val_loss: 0.1178 - val_acc: 0.9638
Epoch 2/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0828
- acc: 0.9762 - val_loss: 0.0906 - val_acc: 0.9737
Epoch 3/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0470
- acc: 0.9861 - val_loss: 0.0820 - val_acc: 0.9743
Epoch 4/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0285
- acc: 0.9922 - val_loss: 0.0796 - val_acc: 0.9750
Epoch 5/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0201
- acc: 0.9940 - val_loss: 0.0826 - val_acc: 0.9767
Epoch 6/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0155
- acc: 0.9957 - val_loss: 0.0802 - val_acc: 0.9752
Epoch 7/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0136
- acc: 0.9960 - val_loss: 0.0917 - val_acc: 0.9744
Epoch 8/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0122
- acc: 0.9964 - val_loss: 0.0859 - val_acc: 0.9770
Epoch 9/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0138
- acc: 0.9957 - val_loss: 0.0833 - val_acc: 0.9769
Epoch 10/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0091
- acc: 0.9971 - val_loss: 0.0849 - val_acc: 0.9779
Epoch 11/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.0082
- acc: 0.9975 - val_loss: 0.0876 - val_acc: 0.9792
Epoch 12/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0079
- acc: 0.9975 - val_loss: 0.0791 - val_acc: 0.9800
Epoch 13/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0070
```

```
- acc: 0.9980 - val_loss: 0.0924 - val_acc: 0.9773
Epoch 14/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0084
- acc: 0.9974 - val_loss: 0.0871 - val_acc: 0.9790
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0076
- acc: 0.9975 - val_loss: 0.0837 - val_acc: 0.9801
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0076
- acc: 0.9976 - val_loss: 0.0803 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0068
- acc: 0.9980 - val_loss: 0.0851 - val_acc: 0.9798
Epoch 18/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0073
- acc: 0.9975 - val_loss: 0.0899 - val_acc: 0.9799
Epoch 19/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0067
- acc: 0.9979 - val_loss: 0.0812 - val_acc: 0.9790
Epoch 20/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0043
- acc: 0.9986 - val_loss: 0.0844 - val_acc: 0.9810
```

In [0]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_4.history['loss']
test_loss = history_4.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_4.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.08437914842601021
Test accuracy: 0.981
```

[3.1] 3-Layer MLP + ReLu + Adam

No Dropout and Batch Normalization

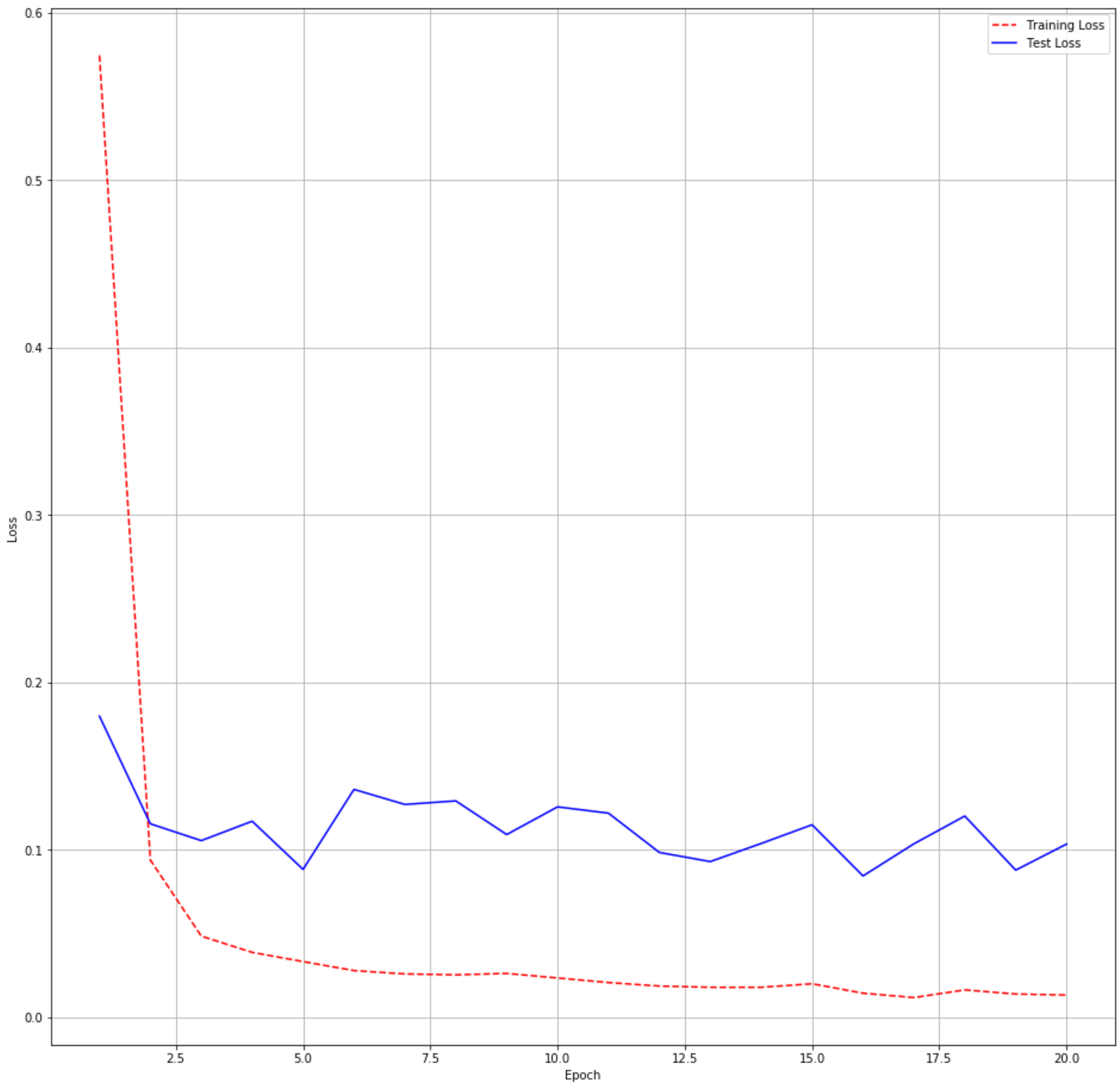
In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
no_epoch = 20
```

In [15]:
```
#MLP
mdl_relu_31 = Sequential()
mdl_relu_31.add(Dense(1024, activation='relu',input_shape=(input_dim,),kern
el_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_31.add(Dense(512, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_31.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_31.add(Dense(output_dim,activation='softmax'))
mdl_relu_31.summary()
mdl_relu_31.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_31 = mdl_relu_31.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 1024)              803840    
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800    
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,395,594
Trainable params: 1,395,594
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimi
zers.py:793: The name tf.train.Optimizer is deprecated. Please use tf.compa
t.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_c
ore/python/ops/math_grad.py:1424: where (from tensorflow.python.ops.array_o
ps) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.5745
- acc: 0.9065 - val_loss: 0.1798 - val_acc: 0.9523
Epoch 2/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0939
- acc: 0.9722 - val_loss: 0.1155 - val_acc: 0.9648
Epoch 3/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0485
- acc: 0.9845 - val_loss: 0.1055 - val_acc: 0.9713
Epoch 4/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0387
- acc: 0.9876 - val_loss: 0.1171 - val_acc: 0.9689
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0333
- acc: 0.9893 - val_loss: 0.0883 - val_acc: 0.9761
Epoch 6/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0279
- acc: 0.9912 - val_loss: 0.1361 - val_acc: 0.9656
Epoch 7/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0259
- acc: 0.9912 - val_loss: 0.1271 - val_acc: 0.9665
Epoch 8/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0253
- acc: 0.9918 - val_loss: 0.1292 - val_acc: 0.9685
Epoch 9/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0262
- acc: 0.9918 - val_loss: 0.1091 - val_acc: 0.9736
Epoch 10/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0235
- acc: 0.9929 - val_loss: 0.1257 - val_acc: 0.9724
Epoch 11/20
```

```
60000/60000 [==============================] - 4s 68us/step - loss: 0.0207
- acc: 0.9934 - val_loss: 0.1219 - val_acc: 0.9748
Epoch 12/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0186
- acc: 0.9943 - val_loss: 0.0984 - val_acc: 0.9765
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0179
- acc: 0.9940 - val_loss: 0.0930 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0179
- acc: 0.9946 - val_loss: 0.1038 - val_acc: 0.9792
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0200
- acc: 0.9937 - val_loss: 0.1149 - val_acc: 0.9780
Epoch 16/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0144
- acc: 0.9956 - val_loss: 0.0844 - val_acc: 0.9815
Epoch 17/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0118
- acc: 0.9965 - val_loss: 0.1036 - val_acc: 0.9791
Epoch 18/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0163
- acc: 0.9952 - val_loss: 0.1202 - val_acc: 0.9774
Epoch 19/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0139
- acc: 0.9960 - val_loss: 0.0879 - val_acc: 0.9808
Epoch 20/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0133
- acc: 0.9961 - val_loss: 0.1034 - val_acc: 0.9798
```

In [19]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_31.history['loss']
test_loss = history_31.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_31.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.10343968682153404
Test accuracy: 0.9798
```

[3.2] 3-Layer MLP + ReLu + Adam + Dropout

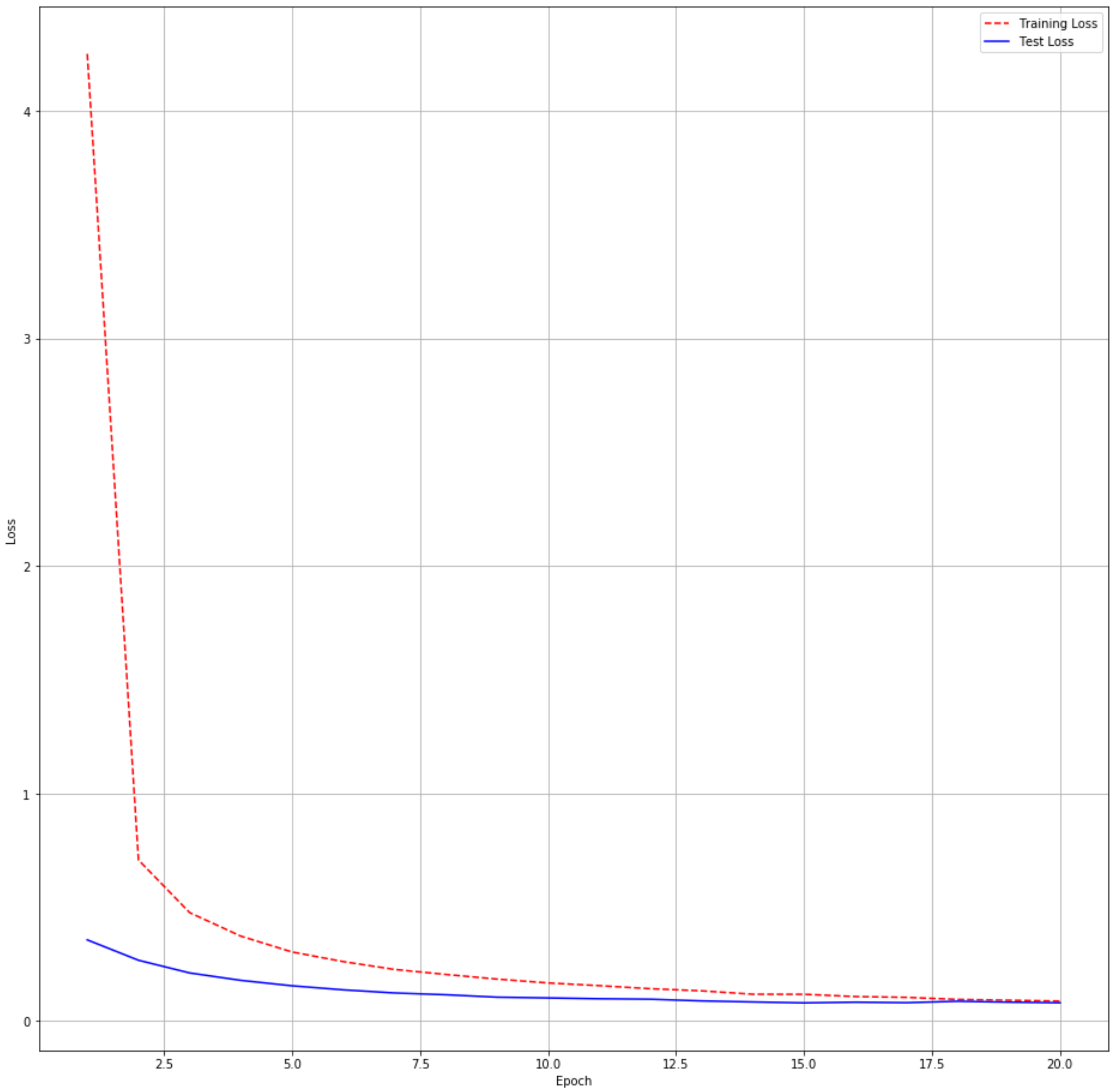
No Batch Normalization

In [20]:
```
mdl_relu_32 = Sequential()
mdl_relu_32.add(Dense(1024, activation='relu',input_shape=(input_dim,),kern
el_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_32.add(Dropout(rate=0.5))
mdl_relu_32.add(Dense(512, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_32.add(Dropout(rate=0.5))
mdl_relu_32.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_32.add(Dropout(rate=0.5))
mdl_relu_32.add(Dense(output_dim,activation='softmax'))
mdl_relu_32.summary()
mdl_relu_32.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_32 = mdl_relu_32.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/tensorflow_backend.py:3733: calling dropout (from tensorflow.python.ops.n
n_ops) with keep_prob is deprecated and will be removed in a future versio
n.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 1024)              803840    
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 512)               524800    
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,395,594
Trainable params: 1,395,594
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 80us/step - loss: 4.2516
- acc: 0.5804 - val_loss: 0.3571 - val_acc: 0.9075
Epoch 2/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.7084
- acc: 0.7998 - val_loss: 0.2675 - val_acc: 0.9302
Epoch 3/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.4768
- acc: 0.8635 - val_loss: 0.2117 - val_acc: 0.9404
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.3735
- acc: 0.8947 - val_loss: 0.1791 - val_acc: 0.9517
Epoch 5/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.3039
- acc: 0.9147 - val_loss: 0.1552 - val_acc: 0.9584
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2615
- acc: 0.9261 - val_loss: 0.1374 - val_acc: 0.9628
Epoch 7/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2271
- acc: 0.9359 - val_loss: 0.1238 - val_acc: 0.9650
Epoch 8/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.2051
- acc: 0.9426 - val_loss: 0.1159 - val_acc: 0.9684
Epoch 9/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.1847
- acc: 0.9478 - val_loss: 0.1052 - val_acc: 0.9696
```

```
Epoch 10/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1678
- acc: 0.9532 - val_loss: 0.1019 - val_acc: 0.9721
Epoch 11/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1558
- acc: 0.9569 - val_loss: 0.0982 - val_acc: 0.9722
Epoch 12/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1424
- acc: 0.9610 - val_loss: 0.0963 - val_acc: 0.9716
Epoch 13/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1330
- acc: 0.9624 - val_loss: 0.0888 - val_acc: 0.9763
Epoch 14/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1182
- acc: 0.9667 - val_loss: 0.0842 - val_acc: 0.9784
Epoch 15/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.1180
- acc: 0.9667 - val_loss: 0.0800 - val_acc: 0.9784
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1079
- acc: 0.9696 - val_loss: 0.0832 - val_acc: 0.9774
Epoch 17/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1044
- acc: 0.9701 - val_loss: 0.0809 - val_acc: 0.9782
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0953
- acc: 0.9734 - val_loss: 0.0872 - val_acc: 0.9789
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0919
- acc: 0.9732 - val_loss: 0.0833 - val_acc: 0.9796
Epoch 20/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0878
- acc: 0.9755 - val_loss: 0.0805 - val_acc: 0.9797
```

In [21]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_32.history['loss']
test_loss = history_32.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_32.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.08047824457662764
Test accuracy: 0.9797
```

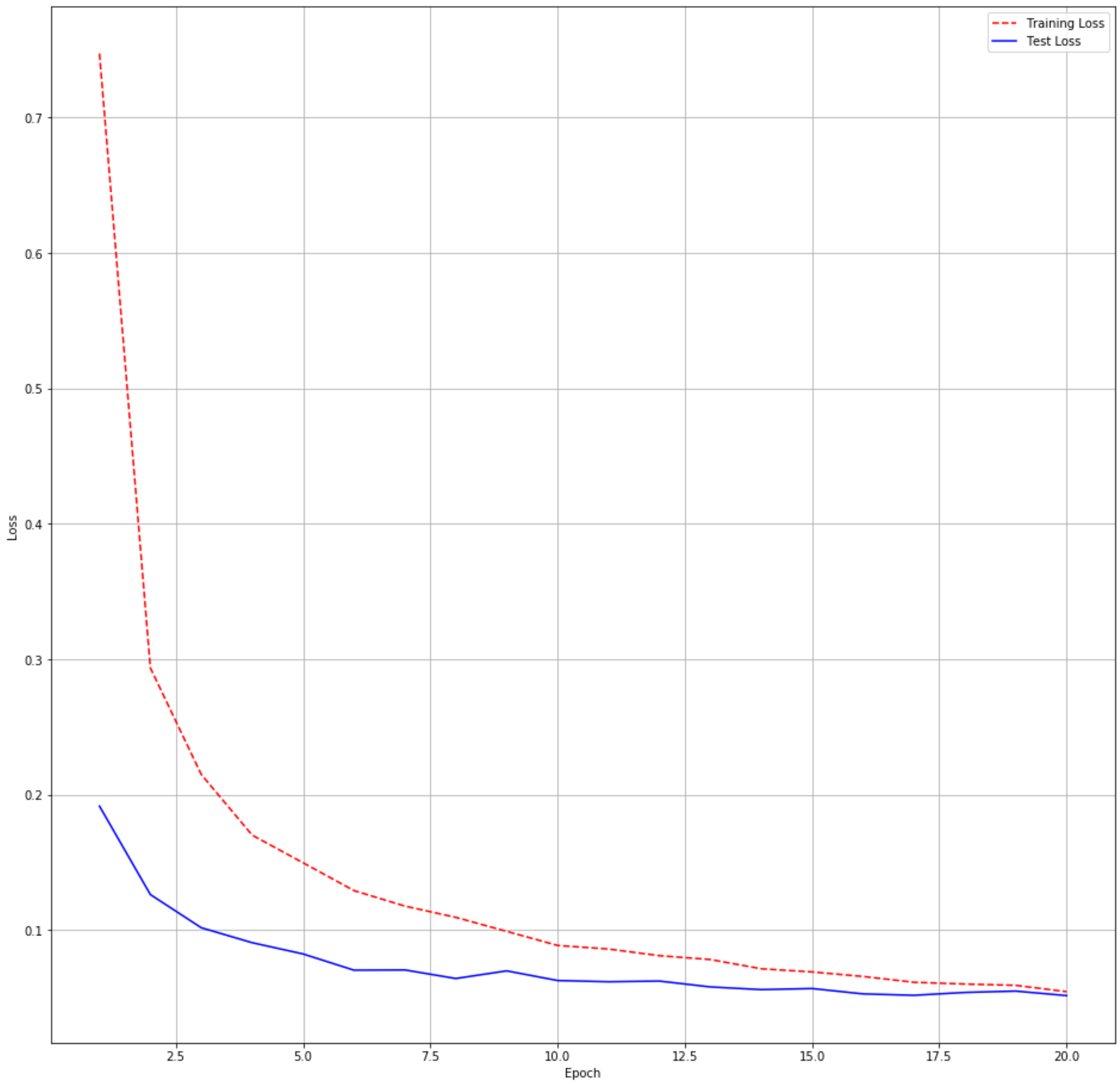
[3.3] 3-Layer MLP + ReLu + Adam + Dropout + Batch Normalization

In [22]:
```
mdl_relu_33 = Sequential()
mdl_relu_33.add(Dense(1024, activation='relu',input_shape=(input_dim,),kern
el_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_33.add(Dropout(rate=0.5))
mdl_relu_33.add(BatchNormalization())
mdl_relu_33.add(Dense(512, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_33.add(Dropout(rate=0.5))
mdl_relu_33.add(BatchNormalization())
mdl_relu_33.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_33.add(Dropout(rate=0.5))
mdl_relu_33.add(BatchNormalization())
mdl_relu_33.add(Dense(output_dim,activation='softmax'))
mdl_relu_33.summary()
mdl_relu_33.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_33 = mdl_relu_33.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_9 (Dense)              (None, 1024)              803840    
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 1024)              4096      
_________________________________________________________________
dense_10 (Dense)             (None, 512)               524800    
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048      
_________________________________________________________________
dense_11 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512       
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,402,250
Trainable params: 1,398,922
Non-trainable params: 3,328
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.7473
- acc: 0.7661 - val_loss: 0.1916 - val_acc: 0.9441
Epoch 2/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.2936
- acc: 0.9141 - val_loss: 0.1262 - val_acc: 0.9609
Epoch 3/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.2149
- acc: 0.9385 - val_loss: 0.1017 - val_acc: 0.9691
Epoch 4/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.1701
- acc: 0.9505 - val_loss: 0.0907 - val_acc: 0.9719
Epoch 5/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.1497
- acc: 0.9563 - val_loss: 0.0824 - val_acc: 0.9747
Epoch 6/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1292
- acc: 0.9614 - val_loss: 0.0704 - val_acc: 0.9771
Epoch 7/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.1177
- acc: 0.9653 - val_loss: 0.0706 - val_acc: 0.9780
Epoch 8/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.1094
- acc: 0.9674 - val_loss: 0.0642 - val_acc: 0.9795
Epoch 9/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0991
- acc: 0.9714 - val_loss: 0.0699 - val_acc: 0.9789
Epoch 10/20
```

```
60000/60000 [==============================] - 8s 127us/step - loss: 0.0886
- acc: 0.9724 - val_loss: 0.0628 - val_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0860
- acc: 0.9740 - val_loss: 0.0619 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0811
- acc: 0.9749 - val_loss: 0.0624 - val_acc: 0.9795
Epoch 13/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0783
- acc: 0.9767 - val_loss: 0.0580 - val_acc: 0.9820
Epoch 14/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0714
- acc: 0.9783 - val_loss: 0.0561 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0691
- acc: 0.9791 - val_loss: 0.0568 - val_acc: 0.9827
Epoch 16/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0658
- acc: 0.9807 - val_loss: 0.0529 - val_acc: 0.9837
Epoch 17/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0615
- acc: 0.9813 - val_loss: 0.0519 - val_acc: 0.9836
Epoch 18/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0602
- acc: 0.9823 - val_loss: 0.0540 - val_acc: 0.9829
Epoch 19/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0592
- acc: 0.9817 - val_loss: 0.0549 - val_acc: 0.9827
Epoch 20/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0545
- acc: 0.9838 - val_loss: 0.0516 - val_acc: 0.9846
```

In [23]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_33.history['loss']
test_loss = history_33.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_33.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.05163941752879764
Test accuracy: 0.9846
```

[3.4] 3-Layer MLP + ReLu + Adam + Batch Normalization

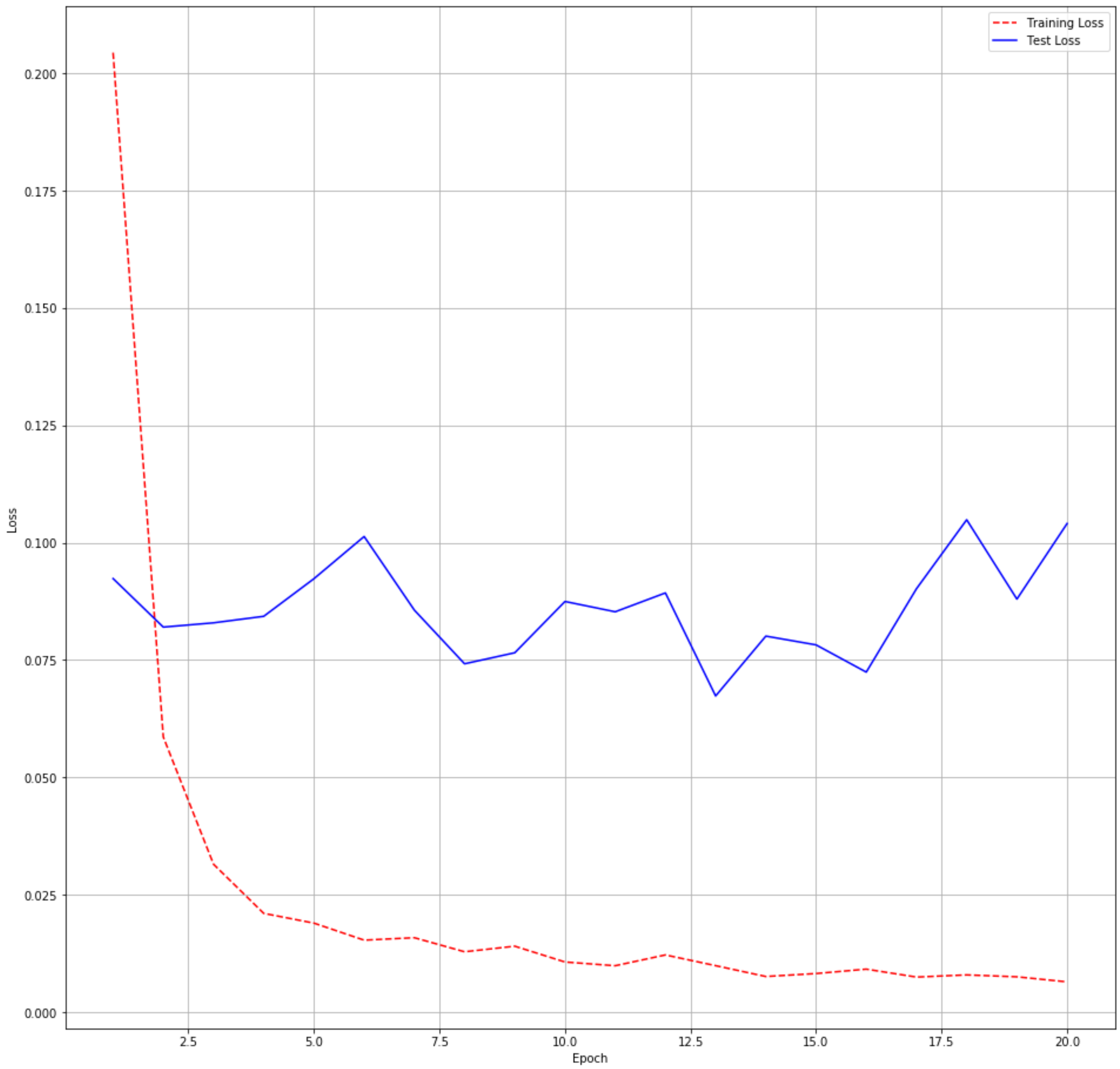
No Dropout

In [24]:
```
mdl_relu_34 = Sequential()
mdl_relu_34.add(Dense(1024, activation='relu',input_shape=(input_dim,),kern
el_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_34.add(BatchNormalization())
mdl_relu_34.add(Dense(512, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_34.add(BatchNormalization())
mdl_relu_34.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_34.add(BatchNormalization())
mdl_relu_34.add(Dense(output_dim,activation='softmax'))
mdl_relu_34.summary()
mdl_relu_34.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_34 = mdl_relu_34.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_4 (Batch (None, 1024)              4096      
_________________________________________________________________
dense_14 (Dense)             (None, 512)               524800    
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048      
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512       
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,402,250
Trainable params: 1,398,922
Non-trainable params: 3,328
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.2044
- acc: 0.9391 - val_loss: 0.0923 - val_acc: 0.9718
Epoch 2/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0586
- acc: 0.9827 - val_loss: 0.0820 - val_acc: 0.9730
Epoch 3/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0315
- acc: 0.9906 - val_loss: 0.0829 - val_acc: 0.9768
Epoch 4/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0210
- acc: 0.9935 - val_loss: 0.0843 - val_acc: 0.9752
Epoch 5/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0190
- acc: 0.9939 - val_loss: 0.0923 - val_acc: 0.9737
Epoch 6/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0153
- acc: 0.9952 - val_loss: 0.1013 - val_acc: 0.9718
Epoch 7/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0158
- acc: 0.9951 - val_loss: 0.0856 - val_acc: 0.9753
Epoch 8/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0128
- acc: 0.9959 - val_loss: 0.0742 - val_acc: 0.9809
Epoch 9/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0140
- acc: 0.9953 - val_loss: 0.0765 - val_acc: 0.9793
Epoch 10/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0107
- acc: 0.9966 - val_loss: 0.0875 - val_acc: 0.9779
Epoch 11/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0099
- acc: 0.9968 - val_loss: 0.0853 - val_acc: 0.9790
Epoch 12/20
```

```
60000/60000 [==============================] - 7s 124us/step - loss: 0.0122
- acc: 0.9958 - val_loss: 0.0893 - val_acc: 0.9767
Epoch 13/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0099
- acc: 0.9966 - val_loss: 0.0673 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0076
- acc: 0.9974 - val_loss: 0.0801 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0082
- acc: 0.9973 - val_loss: 0.0782 - val_acc: 0.9809
Epoch 16/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0091
- acc: 0.9969 - val_loss: 0.0724 - val_acc: 0.9817
Epoch 17/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0075
- acc: 0.9976 - val_loss: 0.0902 - val_acc: 0.9772
Epoch 18/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0079
- acc: 0.9973 - val_loss: 0.1049 - val_acc: 0.9773
Epoch 19/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0075
- acc: 0.9976 - val_loss: 0.0880 - val_acc: 0.9788
Epoch 20/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0064
- acc: 0.9977 - val_loss: 0.1041 - val_acc: 0.9771
```

In [26]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_34.history['loss']
test_loss = history_34.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_34.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.10407820313923867
Test accuracy: 0.9771
```

[4.1] 3-Layer MLP + ReLu + Adam

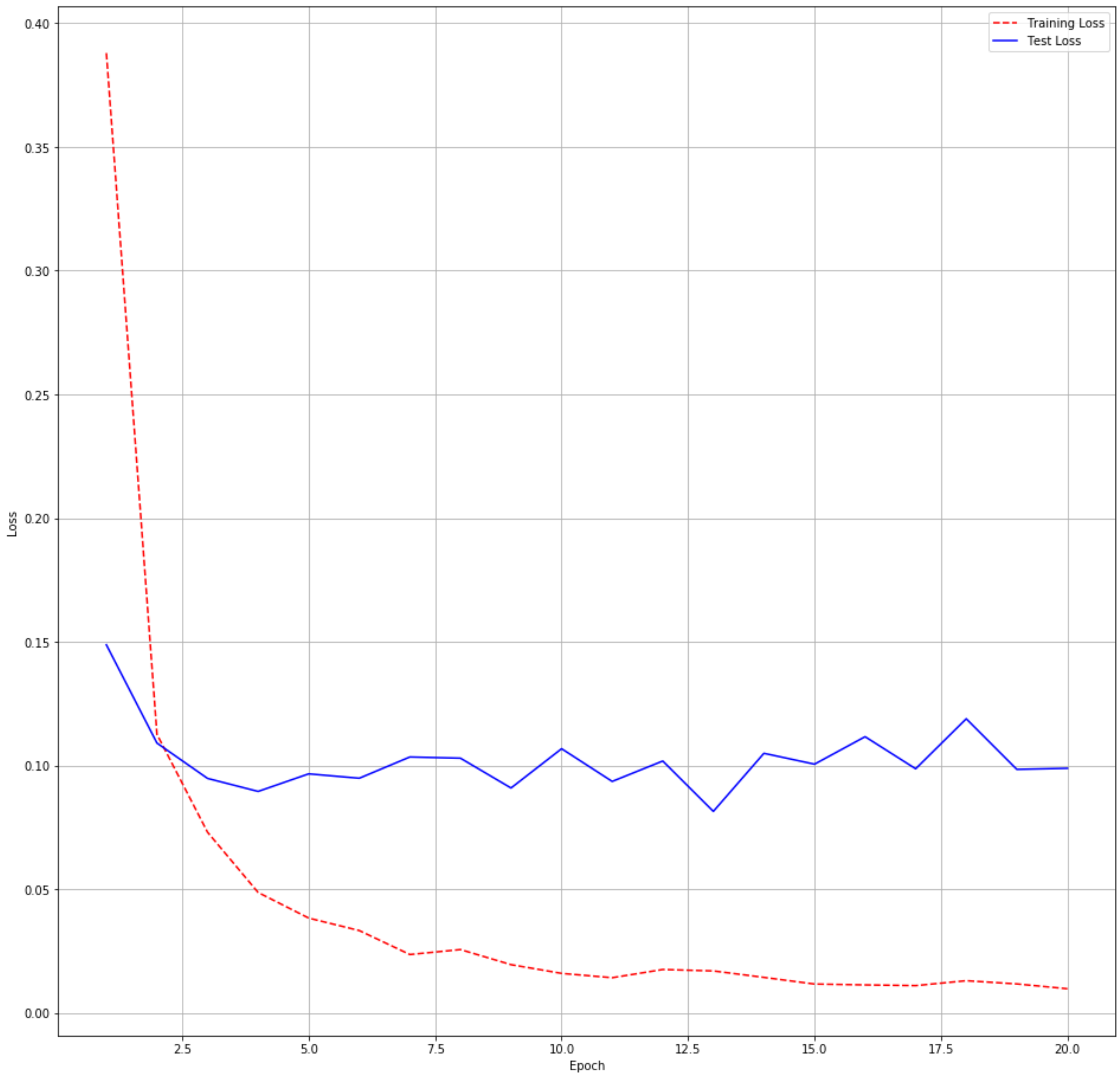
No Dropout and Batch Normalization

In [27]:
```
mdl_relu_41 = Sequential()
mdl_relu_41.add(Dense(512, activation='relu',input_shape=(input_dim,),kerne
l_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_41.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_41.add(Dense(64, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_41.add(Dense(32, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.180, seed=42)))
mdl_relu_41.add(Dense(16, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.190, seed=42)))
mdl_relu_41.add(Dense(output_dim,activation='softmax'))
mdl_relu_41.summary()
mdl_relu_41.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_41 = mdl_relu_41.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_17 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_20 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_21 (Dense)             (None, 16)                528       
_________________________________________________________________
dense_22 (Dense)             (None, 10)                170       
=================================================================
Total params: 478,618
Trainable params: 478,618
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.3879
- acc: 0.8846 - val_loss: 0.1488 - val_acc: 0.9567
Epoch 2/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1126
- acc: 0.9656 - val_loss: 0.1091 - val_acc: 0.9668
Epoch 3/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0731
- acc: 0.9779 - val_loss: 0.0948 - val_acc: 0.9720
Epoch 4/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0487
- acc: 0.9850 - val_loss: 0.0895 - val_acc: 0.9736
Epoch 5/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0384
- acc: 0.9874 - val_loss: 0.0966 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0333
- acc: 0.9890 - val_loss: 0.0949 - val_acc: 0.9732
Epoch 7/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0236
- acc: 0.9921 - val_loss: 0.1035 - val_acc: 0.9741
Epoch 8/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0256
- acc: 0.9914 - val_loss: 0.1030 - val_acc: 0.9724
Epoch 9/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0195
- acc: 0.9935 - val_loss: 0.0909 - val_acc: 0.9781
Epoch 10/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0160
- acc: 0.9948 - val_loss: 0.1068 - val_acc: 0.9747
Epoch 11/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0143
- acc: 0.9957 - val_loss: 0.0936 - val_acc: 0.9798
Epoch 12/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0176
- acc: 0.9942 - val_loss: 0.1018 - val_acc: 0.9758
```

```
Epoch 13/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0170
- acc: 0.9947 - val_loss: 0.0815 - val_acc: 0.9807
Epoch 14/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0144
- acc: 0.9955 - val_loss: 0.1050 - val_acc: 0.9758
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0117
- acc: 0.9959 - val_loss: 0.1006 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0114
- acc: 0.9965 - val_loss: 0.1117 - val_acc: 0.9774
Epoch 17/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0111
- acc: 0.9962 - val_loss: 0.0987 - val_acc: 0.9788
Epoch 18/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0130
- acc: 0.9959 - val_loss: 0.1189 - val_acc: 0.9745
Epoch 19/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0117
- acc: 0.9962 - val_loss: 0.0984 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0098
- acc: 0.9970 - val_loss: 0.0989 - val_acc: 0.9782
```

In [28]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_41.history['loss']
test_loss = history_41.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_41.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.09889793922199387
Test accuracy: 0.9782
```

[4.2] 3-Layer MLP + ReLu + Adam + Dropout

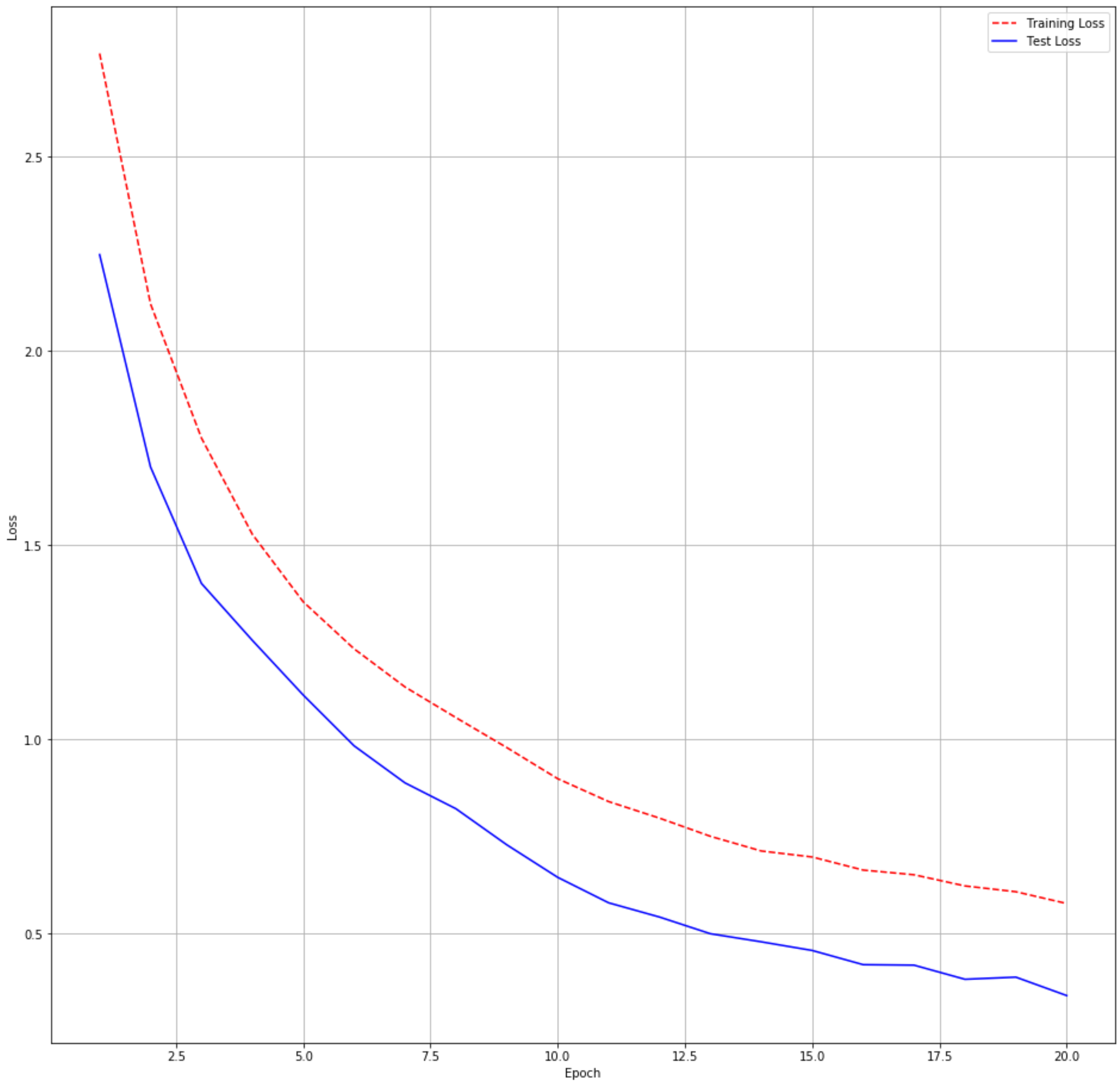
No Batch Normalization

In [29]:
```
mdl_relu_42 = Sequential()
mdl_relu_42.add(Dense(512, activation='relu',input_shape=(input_dim,),kerne
l_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_42.add(Dropout(rate=0.5))
mdl_relu_42.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_42.add(Dropout(rate=0.5))
mdl_relu_42.add(Dense(64, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_42.add(Dropout(rate=0.5))
mdl_relu_42.add(Dense(32, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.180, seed=42)))
mdl_relu_42.add(Dropout(rate=0.5))
mdl_relu_42.add(Dense(16, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.190, seed=42)))
mdl_relu_42.add(Dropout(rate=0.5))
mdl_relu_42.add(Dense(output_dim,activation='softmax'))
mdl_relu_42.summary()
mdl_relu_42.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_42 = mdl_relu_42.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_23 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_25 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_26 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_27 (Dense)             (None, 16)                528       
_________________________________________________________________
dropout_11 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_28 (Dense)             (None, 10)                170       
=================================================================
Total params: 478,618
Trainable params: 478,618
Non-trainable params: 0

_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 90us/step - loss: 2.7650
- acc: 0.1245 - val_loss: 2.2475 - val_acc: 0.1779
Epoch 2/20
60000/60000 [==============================] - 4s 71us/step - loss: 2.1208
- acc: 0.2122 - val_loss: 1.7012 - val_acc: 0.3865
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step - loss: 1.7758
- acc: 0.3201 - val_loss: 1.4016 - val_acc: 0.4645
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 1.5281
- acc: 0.4101 - val_loss: 1.2552 - val_acc: 0.5403
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 1.3543
- acc: 0.4754 - val_loss: 1.1145 - val_acc: 0.5795
Epoch 6/20
60000/60000 [==============================] - 4s 73us/step - loss: 1.2332
- acc: 0.5396 - val_loss: 0.9837 - val_acc: 0.6578
Epoch 7/20
60000/60000 [==============================] - 4s 71us/step - loss: 1.1352
- acc: 0.5759 - val_loss: 0.8884 - val_acc: 0.6652
Epoch 8/20
60000/60000 [==============================] - 4s 72us/step - loss: 1.0563
- acc: 0.6120 - val_loss: 0.8219 - val_acc: 0.7128
Epoch 9/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.9792
```

```
- acc: 0.6512 - val_loss: 0.7290 - val_acc: 0.7849
Epoch 10/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.8991
- acc: 0.6950 - val_loss: 0.6454 - val_acc: 0.8035
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.8403
- acc: 0.7196 - val_loss: 0.5799 - val_acc: 0.8399
Epoch 12/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.7977
- acc: 0.7374 - val_loss: 0.5432 - val_acc: 0.8696
Epoch 13/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.7509
- acc: 0.7566 - val_loss: 0.5003 - val_acc: 0.8430
Epoch 14/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.7132
- acc: 0.7731 - val_loss: 0.4795 - val_acc: 0.8792
Epoch 15/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.6977
- acc: 0.7852 - val_loss: 0.4573 - val_acc: 0.8894
Epoch 16/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.6641
- acc: 0.8003 - val_loss: 0.4208 - val_acc: 0.9031
Epoch 17/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.6520
- acc: 0.8060 - val_loss: 0.4195 - val_acc: 0.8997
Epoch 18/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.6233
- acc: 0.8197 - val_loss: 0.3831 - val_acc: 0.9206
Epoch 19/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.6084
- acc: 0.8303 - val_loss: 0.3886 - val_acc: 0.9292
Epoch 20/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.5780
- acc: 0.8404 - val_loss: 0.3412 - val_acc: 0.9306
```

In [30]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_42.history['loss']
test_loss = history_42.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_42.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.3411893347978592
Test accuracy: 0.9306
```

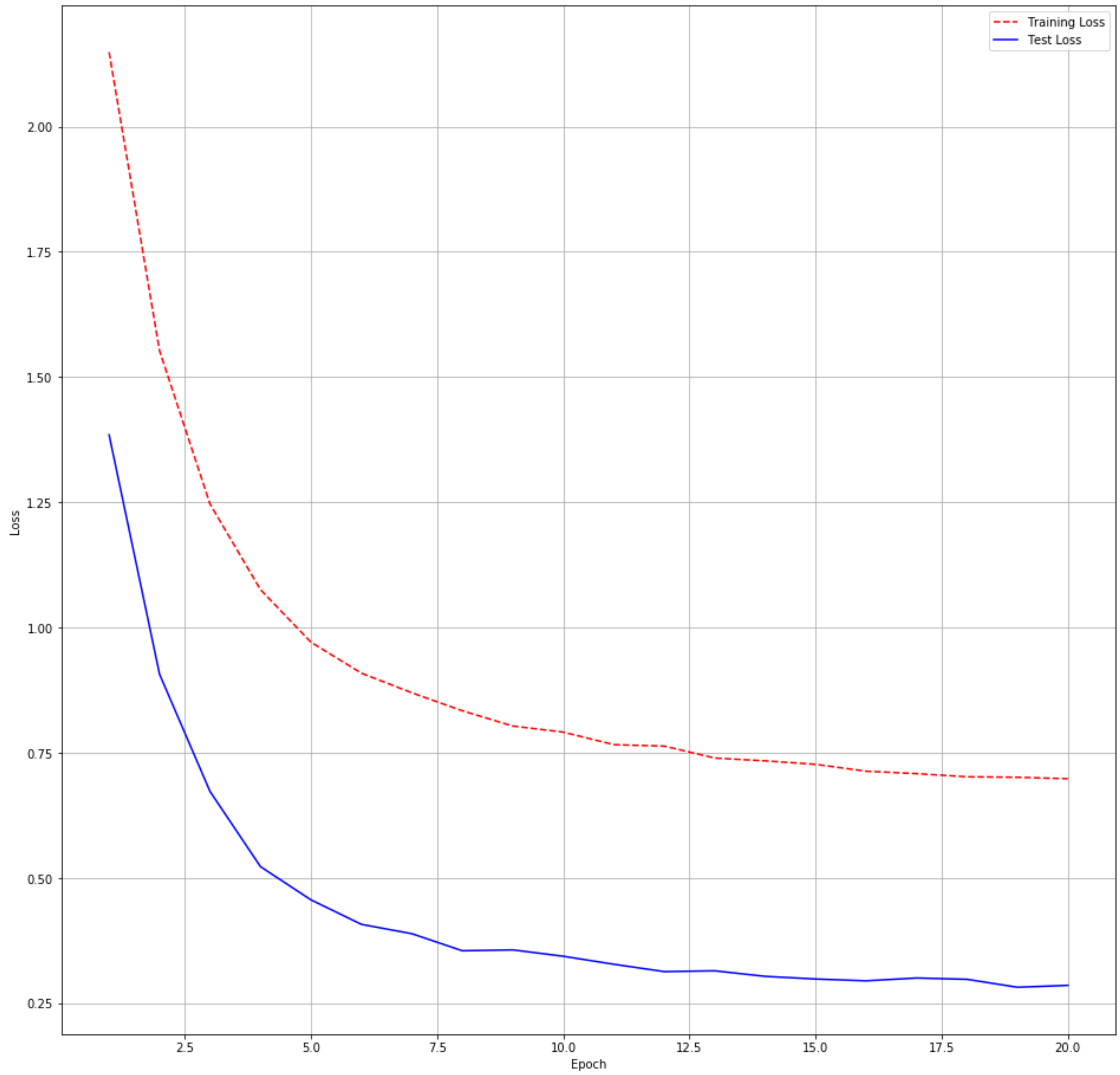
[4.3] 3-Layer MLP + ReLu + Adam + Dropout + Batch Normalization

In [31]:
```
mdl_relu_43 = Sequential()
mdl_relu_43.add(Dense(512, activation='relu',input_shape=(input_dim,),kerne
l_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_43.add(Dropout(rate=0.5))
mdl_relu_43.add(BatchNormalization())
mdl_relu_43.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_43.add(Dropout(rate=0.5))
mdl_relu_43.add(BatchNormalization())
mdl_relu_43.add(Dense(64, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_43.add(Dropout(rate=0.5))
mdl_relu_43.add(BatchNormalization())
mdl_relu_43.add(Dense(32, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.180, seed=42)))
mdl_relu_43.add(Dropout(rate=0.5))
mdl_relu_43.add(BatchNormalization())
mdl_relu_43.add(Dense(16, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.190, seed=42)))
mdl_relu_43.add(Dropout(rate=0.5))
mdl_relu_43.add(BatchNormalization())
mdl_relu_43.add(Dense(output_dim,activation='softmax'))
mdl_relu_43.summary()
mdl_relu_43.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_43 = mdl_relu_43.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048      
_________________________________________________________________
dense_30 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0         
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512       
_________________________________________________________________
dense_31 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256       
_________________________________________________________________
dense_32 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_15 (Dropout)         (None, 32)                0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128       
_________________________________________________________________
dense_33 (Dense)             (None, 16)                528       
_________________________________________________________________
dropout_16 (Dropout)         (None, 16)                0         
_________________________________________________________________
batch_normalization_11 (Batc (None, 16)                64        
_________________________________________________________________
dense_34 (Dense)             (None, 10)                170       
=================================================================
Total params: 481,626
Trainable params: 480,122
Non-trainable params: 1,504
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 193us/step - loss: 2.148
7 - acc: 0.2260 - val_loss: 1.3847 - val_acc: 0.6565
Epoch 2/20
60000/60000 [==============================] - 9s 156us/step - loss: 1.5526
- acc: 0.4431 - val_loss: 0.9069 - val_acc: 0.7928
Epoch 3/20
60000/60000 [==============================] - 9s 154us/step - loss: 1.2466
- acc: 0.5564 - val_loss: 0.6729 - val_acc: 0.8246
Epoch 4/20
60000/60000 [==============================] - 9s 157us/step - loss: 1.0757
- acc: 0.6197 - val_loss: 0.5228 - val_acc: 0.8483
Epoch 5/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.970
8 - acc: 0.6539 - val_loss: 0.4565 - val_acc: 0.8560
Epoch 6/20
```

```
60000/60000 [==============================] - 9s 156us/step - loss: 0.9089
- acc: 0.6775 - val_loss: 0.4076 - val_acc: 0.8588
Epoch 7/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.869
8 - acc: 0.6878 - val_loss: 0.3889 - val_acc: 0.8634
Epoch 8/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.8338
- acc: 0.6996 - val_loss: 0.3548 - val_acc: 0.8719
Epoch 9/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.803
4 - acc: 0.7104 - val_loss: 0.3566 - val_acc: 0.8721
Epoch 10/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.7913
- acc: 0.7120 - val_loss: 0.3439 - val_acc: 0.8742
Epoch 11/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.7662
- acc: 0.7214 - val_loss: 0.3279 - val_acc: 0.8767
Epoch 12/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.7634
- acc: 0.7200 - val_loss: 0.3133 - val_acc: 0.8794
Epoch 13/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.7395
- acc: 0.7262 - val_loss: 0.3148 - val_acc: 0.8792
Epoch 14/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.733
9 - acc: 0.7276 - val_loss: 0.3037 - val_acc: 0.8816
Epoch 15/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.727
0 - acc: 0.7300 - val_loss: 0.2985 - val_acc: 0.8827
Epoch 16/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.7133
- acc: 0.7360 - val_loss: 0.2949 - val_acc: 0.8851
Epoch 17/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.7084
- acc: 0.7366 - val_loss: 0.3005 - val_acc: 0.8811
Epoch 18/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.7022
- acc: 0.7363 - val_loss: 0.2979 - val_acc: 0.8818
Epoch 19/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.7012
- acc: 0.7376 - val_loss: 0.2820 - val_acc: 0.8871
Epoch 20/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.698
2 - acc: 0.7382 - val_loss: 0.2858 - val_acc: 0.8852
```

In [32]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_43.history['loss']
test_loss = history_43.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_43.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.28580667269229887
Test accuracy: 0.8852
```

[4.4] 3-Layer MLP + ReLu + Adam + Batch Normalization

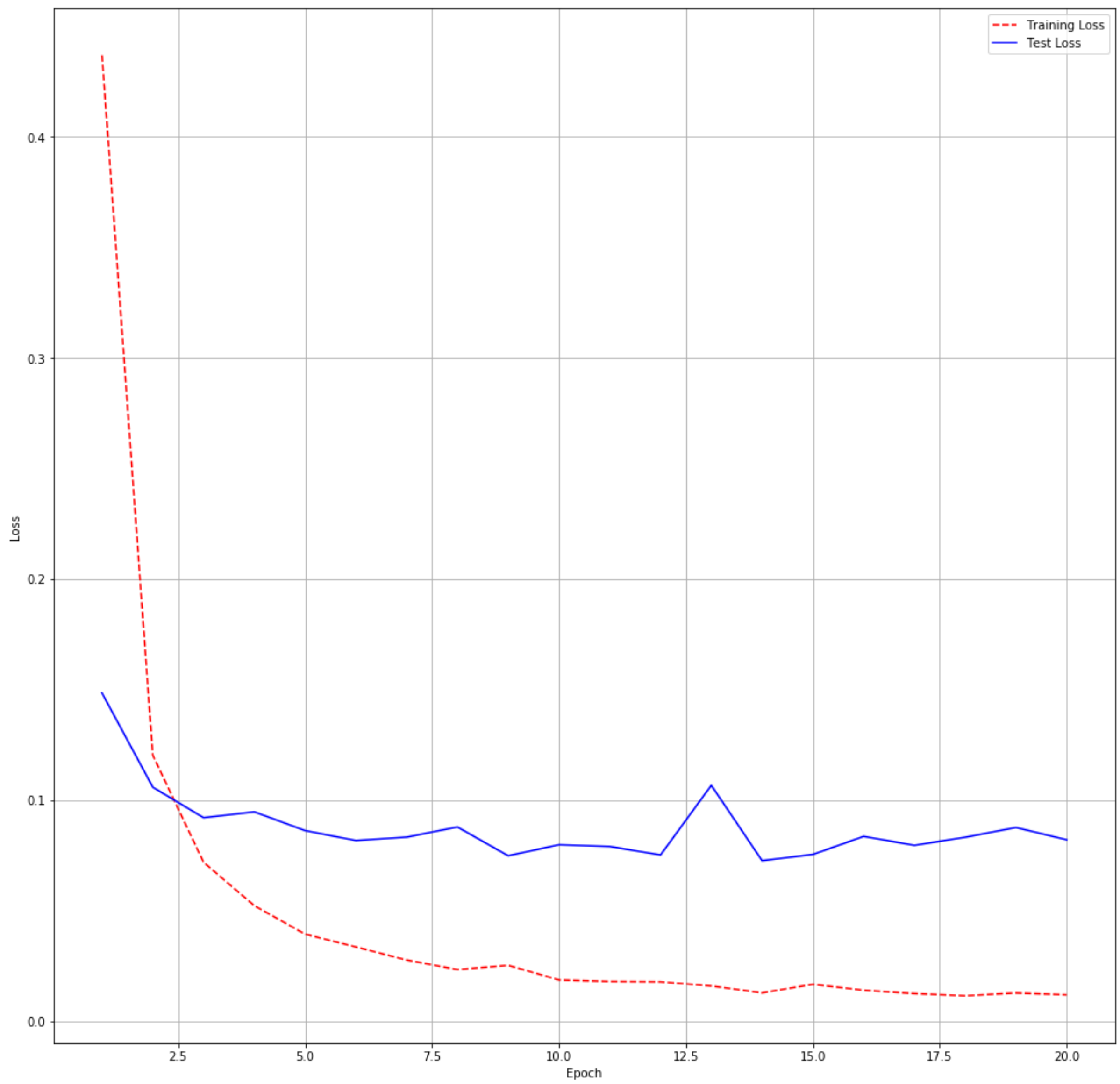
No Dropout

In [33]:
```
mdl_relu_44 = Sequential()
mdl_relu_44.add(Dense(512, activation='relu',input_shape=(input_dim,),kerne
l_initializer=RandomNormal(mean=0.0,stddev=0.125, seed=42)))
mdl_relu_44.add(BatchNormalization())
mdl_relu_44.add(Dense(128, activation='relu',kernel_initializer=RandomNorma
l(mean=0.0,stddev=0.150, seed=42)))
mdl_relu_44.add(BatchNormalization())
mdl_relu_44.add(Dense(64, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.175, seed=42)))
mdl_relu_44.add(BatchNormalization())
mdl_relu_44.add(Dense(32, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.180, seed=42)))
mdl_relu_44.add(BatchNormalization())
mdl_relu_44.add(Dense(16, activation='relu',kernel_initializer=RandomNormal
(mean=0.0,stddev=0.190, seed=42)))
mdl_relu_44.add(BatchNormalization())
mdl_relu_44.add(Dense(output_dim,activation='softmax'))
mdl_relu_44.summary()
mdl_relu_44.compile(optimizer='adam',loss='categorical_crossentropy', metri
cs=['accuracy'])
history_44 = mdl_relu_44.fit(X_train,y_train,batch_size=batch_size,epochs=n
o_epoch,verbose=1,validation_data=(x_test,y_test))
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048      
_________________________________________________________________
dense_36 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512       
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_14 (Batc (None, 64)                256       
_________________________________________________________________
dense_38 (Dense)             (None, 32)                2080      
_________________________________________________________________
batch_normalization_15 (Batc (None, 32)                128       
_________________________________________________________________
dense_39 (Dense)             (None, 16)                528       
_________________________________________________________________
batch_normalization_16 (Batc (None, 16)                64        
_________________________________________________________________
dense_40 (Dense)             (None, 10)                170       
=================================================================
Total params: 481,626
Trainable params: 480,122
Non-trainable params: 1,504
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.436
9 - acc: 0.8869 - val_loss: 0.1484 - val_acc: 0.9578
Epoch 2/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.1202
- acc: 0.9670 - val_loss: 0.1058 - val_acc: 0.9687
Epoch 3/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0718
- acc: 0.9797 - val_loss: 0.0920 - val_acc: 0.9715
Epoch 4/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0522
- acc: 0.9845 - val_loss: 0.0946 - val_acc: 0.9715
Epoch 5/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0393
- acc: 0.9884 - val_loss: 0.0861 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0335
- acc: 0.9897 - val_loss: 0.0817 - val_acc: 0.9768
Epoch 7/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0276
- acc: 0.9914 - val_loss: 0.0832 - val_acc: 0.9762
Epoch 8/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0233
- acc: 0.9924 - val_loss: 0.0878 - val_acc: 0.9743
Epoch 9/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0252
```

```
- acc: 0.9920 - val_loss: 0.0747 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0186
- acc: 0.9938 - val_loss: 0.0797 - val_acc: 0.9785
Epoch 11/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0179
- acc: 0.9940 - val_loss: 0.0790 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0177
- acc: 0.9939 - val_loss: 0.0751 - val_acc: 0.9825
Epoch 13/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0159
- acc: 0.9948 - val_loss: 0.1066 - val_acc: 0.9744
Epoch 14/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0128
- acc: 0.9960 - val_loss: 0.0725 - val_acc: 0.9818
Epoch 15/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0166
- acc: 0.9945 - val_loss: 0.0753 - val_acc: 0.9797
Epoch 16/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0139
- acc: 0.9955 - val_loss: 0.0835 - val_acc: 0.9809
Epoch 17/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0124
- acc: 0.9960 - val_loss: 0.0795 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.011
4 - acc: 0.9962 - val_loss: 0.0831 - val_acc: 0.9806
Epoch 19/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0127
- acc: 0.9959 - val_loss: 0.0875 - val_acc: 0.9792
Epoch 20/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0119
- acc: 0.9961 - val_loss: 0.0820 - val_acc: 0.9800
```

In [34]:
```
%matplotlib inline
import keras
from matplotlib import pyplot as plt

epoch_count = list(range(1,no_epoch+1))

# Get training and test loss histories
training_loss = history_44.history['loss']
test_loss = history_44.history['val_loss']

# Create count of the number of epochs
score = mdl_relu_44.evaluate(x_test, y_test, verbose=0)
plt_mdl_res(epoch_count, training_loss, test_loss, score[0], score[1])
```




```
Test score: 0.08198474683784589
Test accuracy: 0.98
```

Conclusion

In [1]:
```
import tabulate
```

2- Hidden Layer results

In [9]:
```
res_no_2nn =[['Hidden \nLayer', 'Number of \nNeurons'],
             [1,512],
             [2,128]
            ]
print(tabulate.tabulate(res_no_2nn, tablefmt='fancy_grid'))
```

Hidden Layer	Number of Neurons
1	512
2	128

In [10]:
```
res_tab_2 =[['Dropout','Batch \nNormalization', 'Test \nAccuracy'],
           ['No','No',0.9796],
           ['Yes','No',0.9816],
           ['No','Yes',0.981],
           ['Yes','Yes',0.9809]
          ]
print(tabulate.tabulate(res_tab_2,tablefmt='fancy_grid'))
```

Dropout	Batch Normalization	Test Accuracy
No	No	0.9796
Yes	No	0.9816
No	Yes	0.981
Yes	Yes	0.9809

3- Hidden Layer results

In [7]:
```
res_no_3nn =[['Hidden \nLayer', 'Number of \nNeurons'],
            [1,1024],
            [2,512],
            [3,128]
           ]
print(tabulate.tabulate(res_no_3nn, tablefmt='fancy_grid'))
```

Hidden Layer	Number of Neurons
1	1024
2	512
3	128

In [8]:
```
res_tab_3 =[['Dropout','Batch \nNormalization', 'Test \nAccuracy'],
          ['No','No',0.9798],
          ['Yes','No',0.9797],
          ['No','Yes',0.9771],
          ['Yes','Yes',0.9846]
         ]
print(tabulate.tabulate(res_tab_3,tablefmt='fancy_grid'))
```

Dropout	Batch Normalization	Test Accuracy
No	No	0.9798
Yes	No	0.9797
No	Yes	0.9771
Yes	Yes	0.9846

5- Hidden Layer results

In [5]:
```
res_no_5nn =[['Hidden \nLayer', 'Number of \nNeurons'],
             [1,512],
             [2,128],
             [3,64],
             [4,32],
             [5,16]
            ]
print(tabulate.tabulate(res_no_5nn, tablefmt='fancy_grid'))
```

Hidden Layer	Number of Neurons
1	512
2	128
3	64
4	32
5	16

In [6]:
```
res_tab_5 =[['Dropout','Batch \nNormalization', 'Test \nAccuracy'],
           ['No','No',0.9872],
           ['Yes','No',0.9306],
           ['No','Yes',0.98],
           ['Yes','Yes',0.8852]
          ]
print(tabulate.tabulate(res_tab_5,tablefmt='fancy_grid'))
```

Dropout	Batch Normalization	Test Accuracy
No	No	0.9872
Yes	No	0.9306
No	Yes	0.98
Yes	Yes	0.8852

The best and worst performance comes from the model that has 5-hidden layers.

The best performance model is the one that has no Dropout or Batch Normalization with a Test-accuracy of 0.9872

The worst performance model has both Dropout and Batch Normalization enabled with a Test-accuracy of 0.8852