Neural Networks (MLP) in Keras - MNIST Digit Classification

In this notebook, we will learn to:

- **import MNIST dataset and visualize some example images**
- **define deep neural network model with single as well as multiple hidden layers**
- **train the model and plot the accuracy or loss at each epoch**
- **study the effect of varying the learning rate, batch size and number of epochs**
- **use SGD and Adam optimizers**
- **save model weights every 10 epochs**
- **resume training by loading a saved model**
- **earlystop the training if there is negligiable improvement in the performance**

Import modules

In [18]:

```
# Use GPU for Theano, comment to use CPU instead of GPU
# Tensorflow uses GPU by default
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN, device=gpu, floatX=float32"
```

In [19]:

```
# If using tensorflow, set image dimensions order
from keras import backend as K
#if K.backend()=='tensorflow':
#    K.set_image_dim_ordering("th")
```

In [20]:

```
import time
import matplotlib.pyplot as plt
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
% matplotlib inline
np.random.seed(2019)
```

```
UsageError: Line magic function `%` not found.
```

Load MNIST dataset

In [21]:

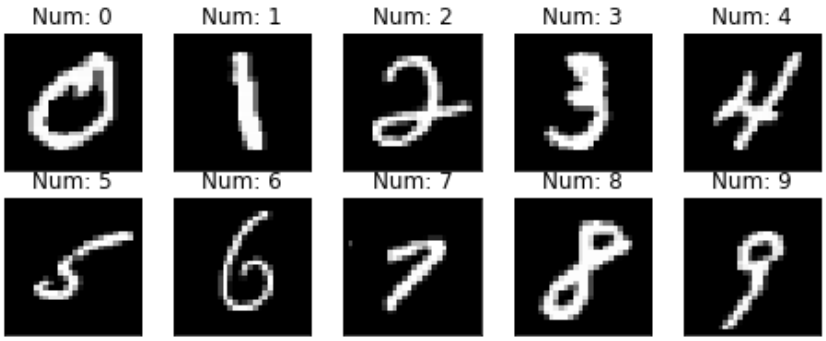
```
from keras.datasets import mnist
(train_features, train_labels), (test_features, test_labels) = mnist.load_data()
_, img_rows, img_cols =  train_features.shape
num_classes = len(np.unique(train_labels))
num_input_nodes = img_rows*img_cols
print("Number of training samples: %d"%train_features.shape[0])
print("Number of test samples: %d"%test_features.shape[0])
print("Image rows: %d"%train_features.shape[1])
print("Image columns: %d"%train_features.shape[2])
print("Number of classes: %d"%num_classes)
```

```
Number of training samples: 60000
Number of test samples: 10000
Image rows: 28
Image columns: 28
Number of classes: 10
```

Show Examples from Each Class

In [22]:

```
fig = plt.figure(figsize=(8,3))
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    features_idx = train_features[train_labels[:]==i,:]
    ax.set_title("Num: " + str(i))
    plt.imshow(features_idx[1], cmap="gray")
plt.show()
```



Pre-processing

In [23]:

```
# reshape images to column vectors
train_features = train_features.reshape(train_features.shape[0], img_rows*img_cols)
test_features = test_features.reshape(test_features.shape[0], img_rows*img_cols)

# convert class labels to binary class labels
train_labels = np_utils.to_categorical(train_labels, num_classes)
test_labels = np_utils.to_categorical(test_labels, num_classes)
```

Define a Neural Network Model with a Single Hidden Layer

In [24]:

```
def simple_nn():
    # initialize model
    model = Sequential()
    # add an input layer and a hidden layer
    model.add(Dense(100, input_dim = num_input_nodes))
    # add activation layer to add non-linearity
    model.add(Activation('sigmoid'))
    # to add ReLu instead of sigmoid: model.add(Activation('relu'))
    # combine above 2 layers: model.add(Dense(100, input_dim=784),Activation('sigmoid'))
    # add output layer
    model.add(Dense(num_classes))
    # add softmax layer
    model.add(Activation('softmax'))
    return model
```

In [25]:

```
# define model
model = simple_nn()
# define optimizer
sgd = SGD(lr=0.01)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# print model information
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 100)               78500
_________________________________________________________________
activation_3 (Activation)    (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1010
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

Train the model

In [26]:

```
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=64, \
                       nb_epoch=10, verbose=2, validation_split=0.2)
end = time.time()
print("Model took %0.2f seconds to train"%(end - start))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:3: UserWarning: The `nb_epo
ch` argument in `fit` has been renamed `epochs`.
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
 - 1s - loss: 0.0823 - accuracy: 0.3158 - val_loss: 0.0736 - val_accuracy: 0.4423
Epoch 2/10
 - 1s - loss: 0.0687 - accuracy: 0.4994 - val_loss: 0.0636 - val_accuracy: 0.5633
Epoch 3/10
 - 1s - loss: 0.0606 - accuracy: 0.5908 - val_loss: 0.0565 - val_accuracy: 0.6307
Epoch 4/10
 - 1s - loss: 0.0544 - accuracy: 0.6442 - val_loss: 0.0509 - val_accuracy: 0.6771
Epoch 5/10
 - 1s - loss: 0.0494 - accuracy: 0.6864 - val_loss: 0.0461 - val_accuracy: 0.7221
Epoch 6/10
 - 1s - loss: 0.0452 - accuracy: 0.7259 - val_loss: 0.0422 - val_accuracy: 0.7571
Epoch 7/10
 - 1s - loss: 0.0416 - accuracy: 0.7571 - val_loss: 0.0388 - val_accuracy: 0.7837
Epoch 8/10
 - 1s - loss: 0.0386 - accuracy: 0.7805 - val_loss: 0.0359 - val_accuracy: 0.8025
Epoch 9/10
 - 1s - loss: 0.0360 - accuracy: 0.7990 - val_loss: 0.0336 - val_accuracy: 0.8200
Epoch 10/10
 - 1s - loss: 0.0338 - accuracy: 0.8143 - val_loss: 0.0316 - val_accuracy: 0.8311
Model took 10.75 seconds to train
```

Plot Accuracy or Loss as a Function of Number of Epoch

In [27]:

```
model_info.history
```

Out[27]:

```
{'val_loss': [0.07362993979454041,
  0.06356644713878631,
  0.05649255094925563,
  0.05085254844029744,
  0.04611803019046783,
  0.042151380737622576,
  0.03878947735826174,
  0.03590123076240222,
  0.03357117299735546,
  0.031601133411129315],
 'val_accuracy': [0.4423333406448364,
  0.5633333325386047,
  0.6306666731834412,
  0.6770833134651184,
  0.722083330154419,
  0.7570833563804626,
  0.7837499976158142,
  0.8025000095367432,
  0.8199999928474426,
  0.831083357334137],
 'loss': [0.08226729214191436,
  0.06872004761298497,
  0.06057885051270326,
  0.05442646891375383,
  0.04942252914607525,
  0.04520062144597371,
  0.04162858049819867,
  0.03855494243651628,
  0.03599927596996228,
  0.033773494134346646],
 'accuracy': [0.3158125,
  0.49939585,
  0.59079164,
  0.6442292,
  0.686375,
  0.7258958,
  0.7571458,
  0.78045833,
  0.79897916,
  0.8143333]}
```

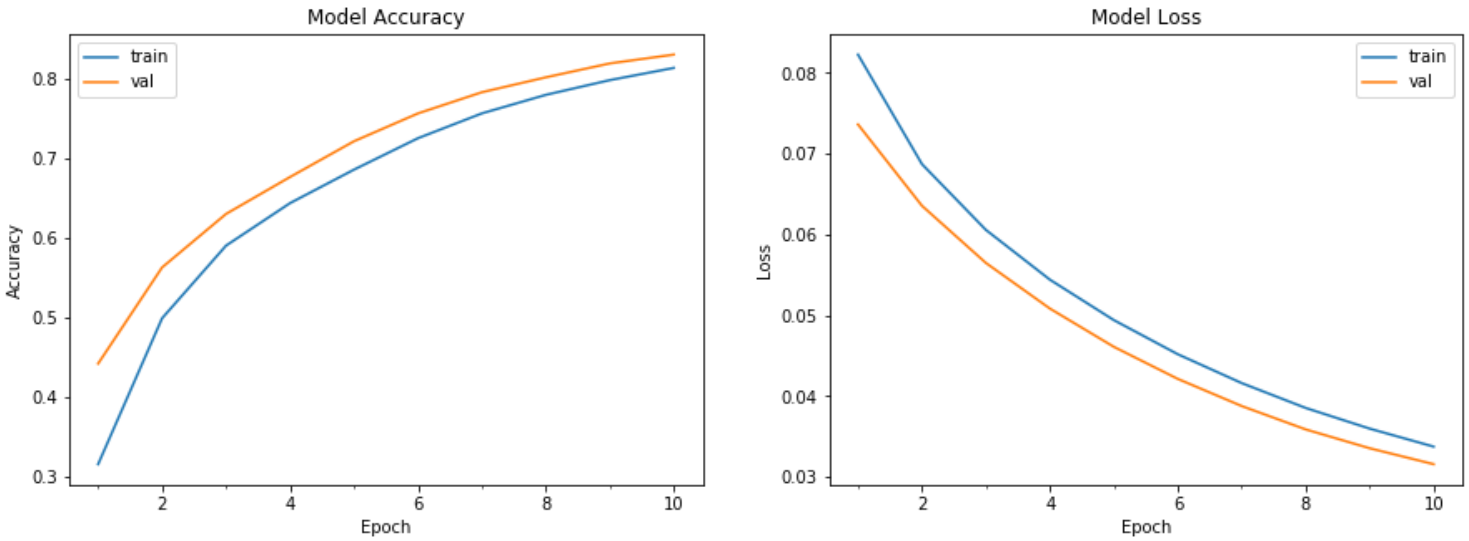
In [30]:

```
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['ac
curacy'])
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.history
['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_histo
ry.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss']
)
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['va
l_loss'])
```

```
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.h
istory['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()
```

In [31]:

```
plot_model_history(model_info)
```



Test the Model

In [32]:

```
def accuracy(test_x, test_y, model):
    result = model.predict(test_x)
    predicted_class = np.argmax(result, axis=1)
    true_class = np.argmax(test_y, axis=1)
    num_correct = np.sum(predicted_class == true_class)
    accuracy = float(num_correct)/result.shape[0]
    return (accuracy * 100)
```

In [33]:

```
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model)
     )
```

```
Accuracy on test data is: 82.59
```

Observation:

Both training and validation accuracy increase as the number of epochs increase. More information is learned in each epoch.

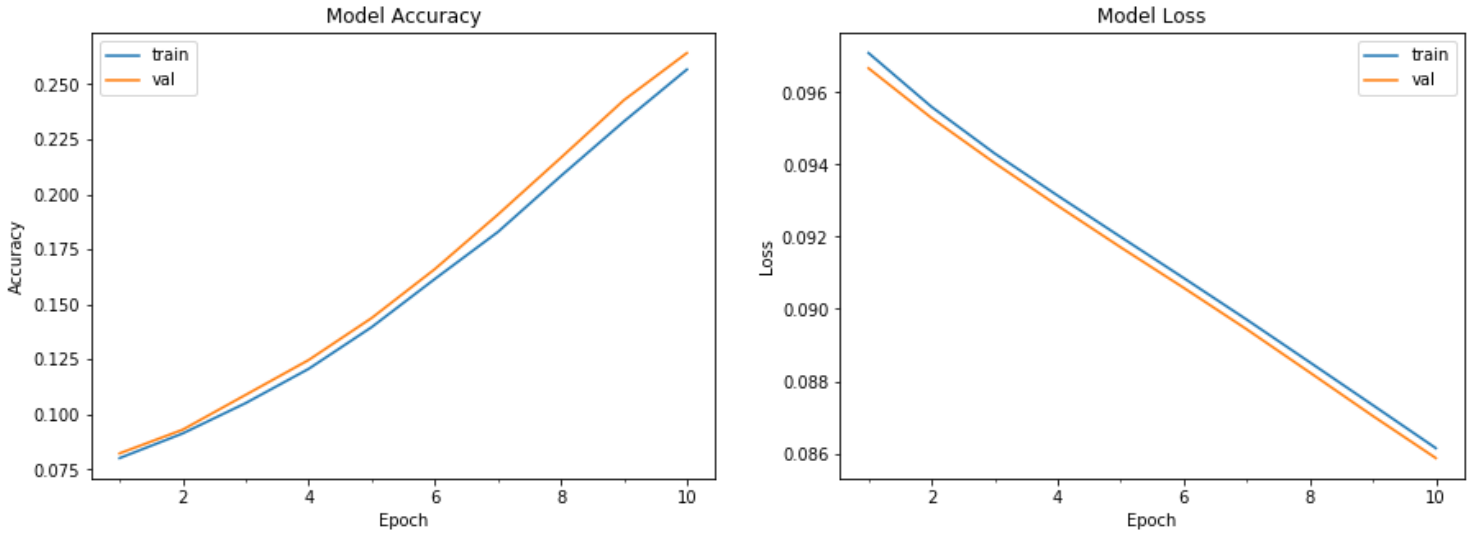
Vary the Learning Rate

In [34]:

```
# decrease the learning rate
# define model
model = simple_nn()
```

```
# define optimizer
sgd = SGD(lr=0.001)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])

# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=64, \
                       nb_epoch=10, verbose=0, validation_split=0.2)
end = time.time()

# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))

# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:12: UserWarning: The `nb_ep
och` argument in `fit` has been renamed `epochs`.
  if sys.path[0] == '':
```

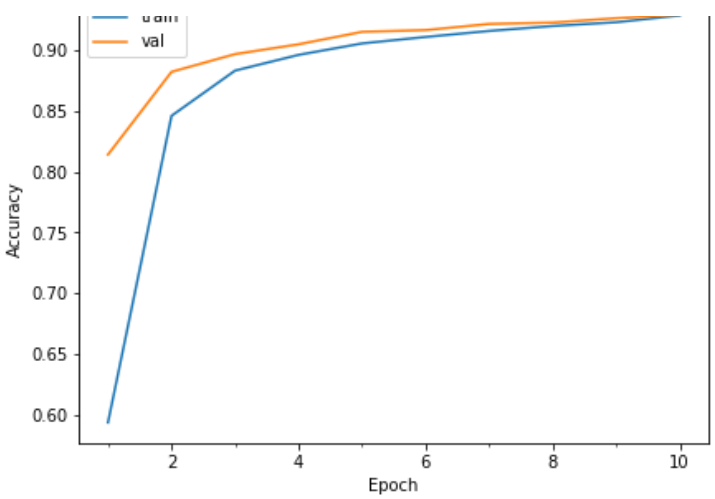


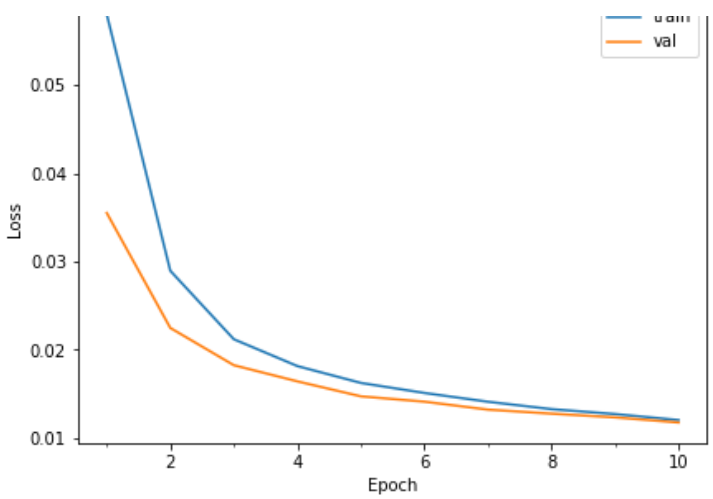

```
Model took 10.03 seconds to train
Accuracy on test data is: 27.30
```

In [35]:

```
# increase the learning rate
# define model
model = simple_nn()
# define optimizer
sgd = SGD(lr=0.1)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=64, \
                       nb_epoch=10, verbose=0, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:10: UserWarning: The `nb_ep
och` argument in `fit` has been renamed `epochs`.
  # Remove the CWD from sys.path while we load stuff.
```






```
Model took 9.36 seconds to train
Accuracy on test data is: 92.84
```

Observation:

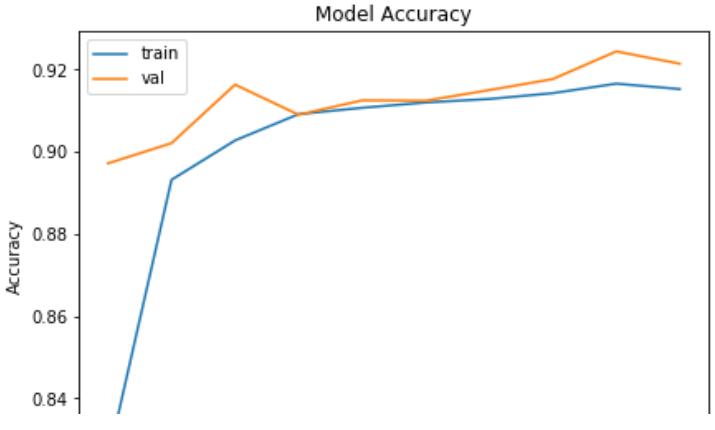
If the learning rate is decreased, less information is learned in each epoch and more epochs are required to learn a good model. If the learning rate is increased, more information is learned in each epoch and less epochs are required to learn a good model. When using SGD, learning rate needs to be decided emperically for a given dataset.

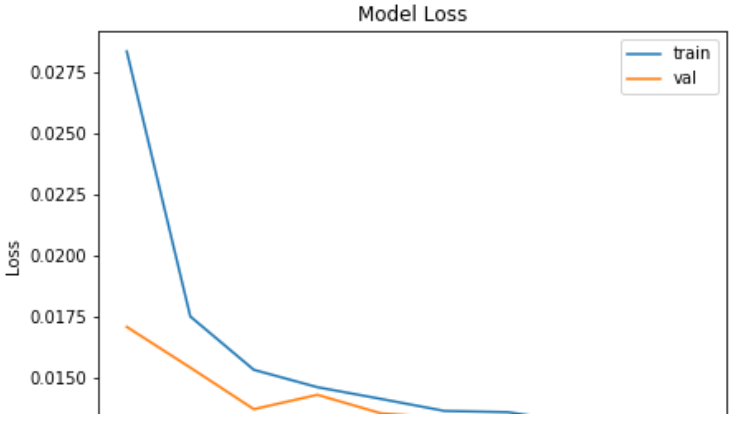
Use Adam optimizer instead of SGD

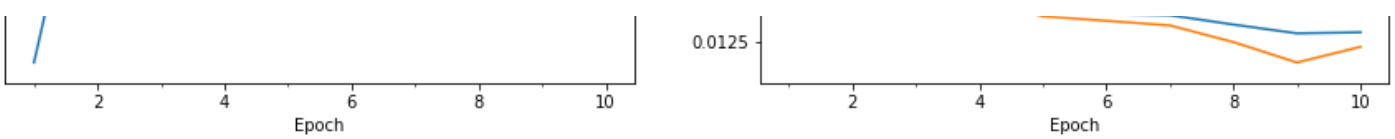
In [36]:

```
# Define model
model = simple_nn()
# define optimizer, loss function
model.compile(optimizer='adam',
              loss='mse',
              metrics=['accuracy'])
# Train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=64, \
                       nb_epoch=10, verbose=0, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```








```
Model took 12.87 seconds to train
Accuracy on test data is: 91.83
```

Observation:

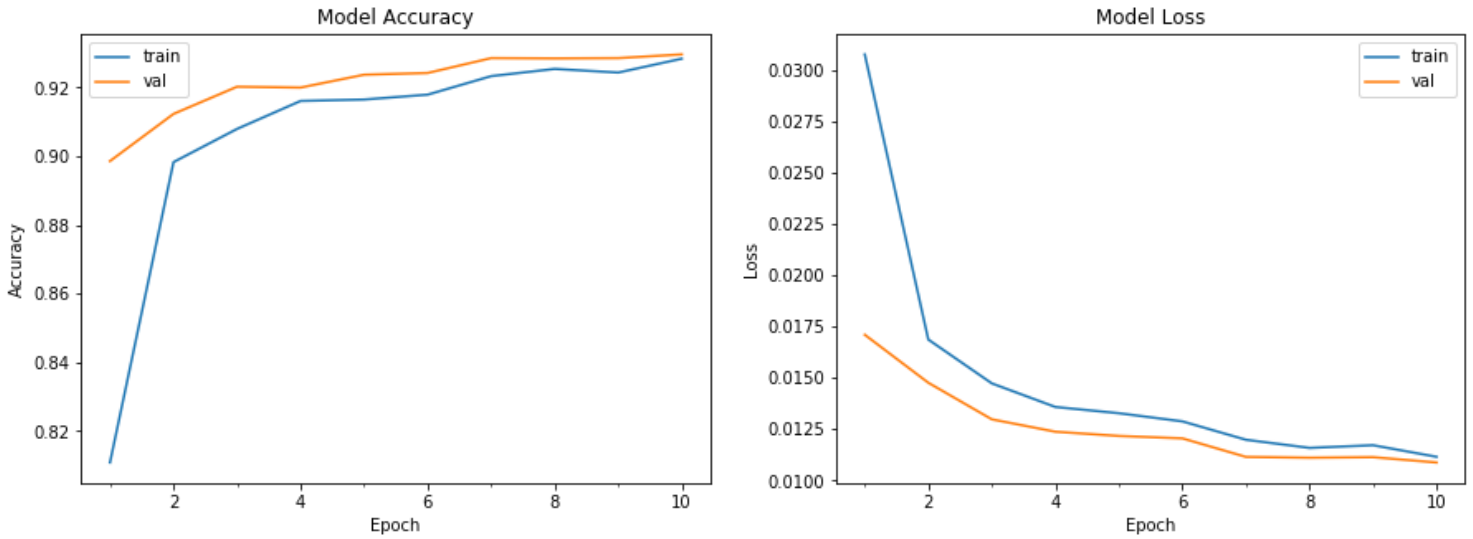
Using Adam optimizer, we don't need to specify a learning rate. However, the training time increases. Refer this tutorial for an interesting comparison of optimizers.

Vary the Batch Size

In [37]:

```
# increase the batch size
# define model
model = simple_nn()
# define optimizer
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# train the model
start = time.time()
model_info = model.fit(train_features,
                       train_labels,
                       batch_size=128,
                       nb_epoch=10,
                       verbose=0,
                       validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:13: UserWarning: The `nb_ep
och` argument in `fit` has been renamed `epochs`.
  del sys.path[0]
```

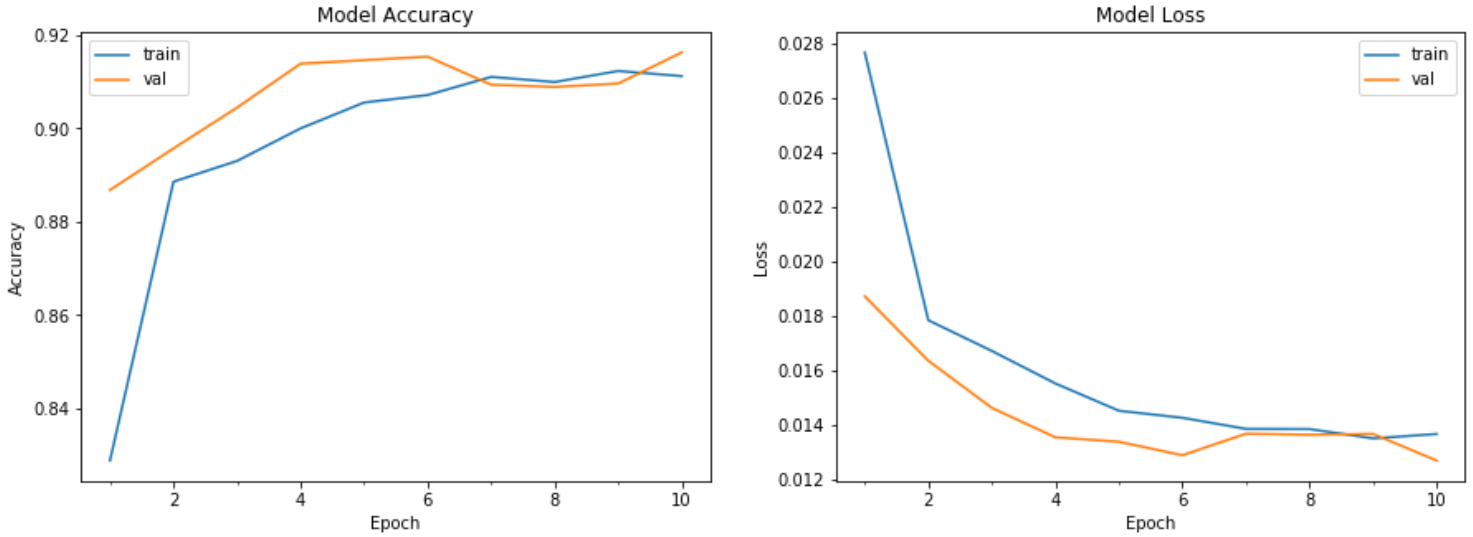



```
Model took 9.52 seconds to train
Accuracy on test data is: 92.63
```

In [38]:

```
# decrease the batch size
# define model
model = simple_nn()
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=32, \
                       nb_epoch=10, verbose=0, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: The `nb_epo
ch` argument in `fit` has been renamed `epochs`.
```




```
Model took 22.81 seconds to train
Accuracy on test data is: 91.64
```

Observation:

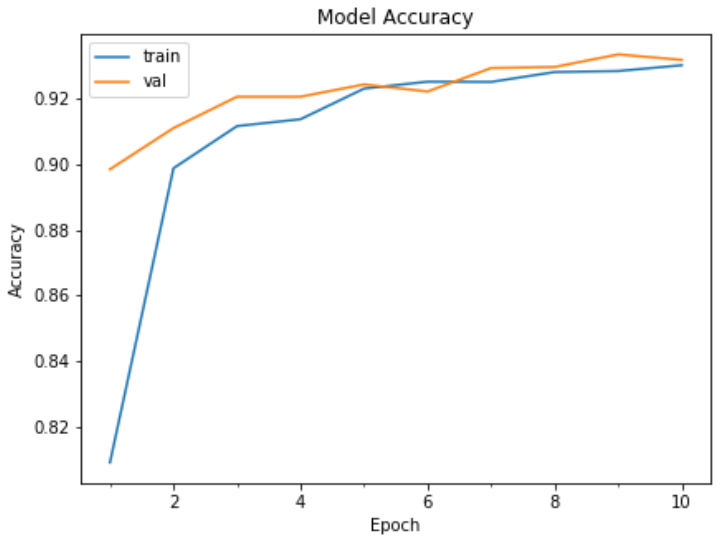
Increasing the batch size decreases the training time but reduces the rate of learning.

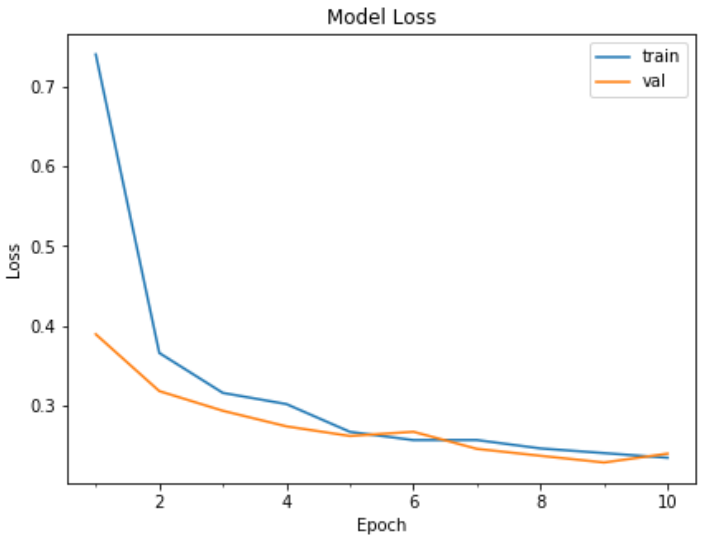
Change the Cost Function to Categorical Crossentropy

In [39]:

```
# define model
model = simple_nn()
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=10, verbose=0, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:9: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  if __name__ == '__main__':
```






```
Model took 9.49 seconds to train
Accuracy on test data is: 93.00
```

Observation:

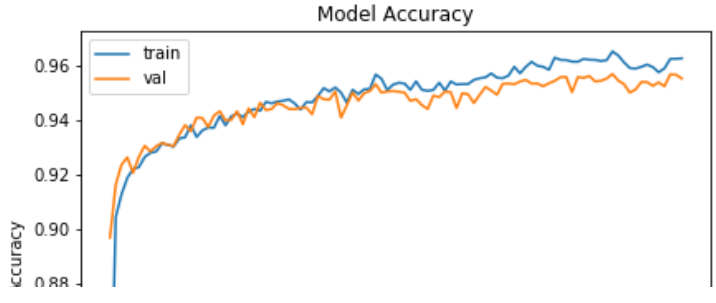
Changing the cost function to categorical crossentropy reduced the training time. The decrease in training time is significant when using SGD for this experiment.

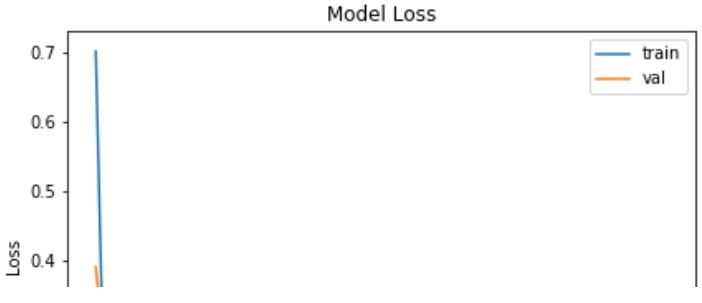
Increase the Number of Epochs

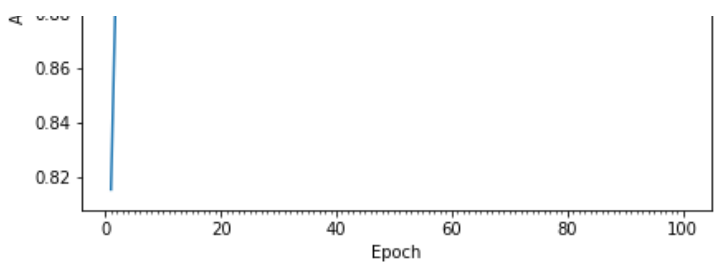
In [40]:

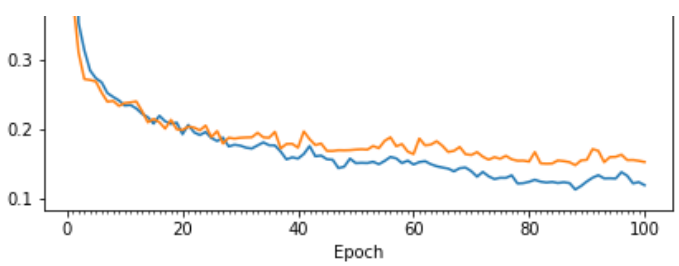
```
# define model
model = simple_nn()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=100, verbose=0, validation_split=0.2)
end = time.time()

# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))

# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```










```
Model took 97.96 seconds to train
Accuracy on test data is: 95.43
```

Observation:

As the number of epochs increase, more information is learned. The training as well as validation accuracy increases and then stabilizes.

Deep Neural Network

In [42]:

```
def deep_nn():
    # Define a deep neural network
    model = Sequential()
    model.add(Dense(512, input_dim=num_input_nodes))
    model.add(Activation('relu'))
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))
    return model
```

In [ ]:

```
model = deep_nn()
# Define optimizer
sgd = SGD(lr=0.1)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# Train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=100, verbose=0, validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
print("Model took %0.2f seconds to train"%(end - start))
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

```
C:\Users\Vihan\anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

Observation:

By adding more hidden layers, training time as well as information learned in each epoch increases. It helps to improve the performance for complex tasks but may not help significantly for relatively simple datasets such as MNIST.

Save model every 10th epoch

In [ ]:

```
from keras.callbacks import ModelCheckpoint
import os

# define a deep neural network
model = deep_nn()
# define optimizer
sgd = SGD(lr=0.1)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# checkpoint
outputFolder = './output-mnist'
if not os.path.exists(outputFolder):
    os.makedirs(outputFolder)
filepath=outputFolder+"/weights-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, \
                             save_best_only=False, save_weights_only=True, \
                             mode='auto', period=10)
callbacks_list = [checkpoint]
# train the model
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=80, callbacks=callbacks_list, verbose=0, \
                       validation_split=0.2)
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

Resume training by loading a saved model

In [ ]:

```
# define model
model = deep_nn()
# load weights
import os, glob
epoch_num = 79
outputFolder = './output-mnist'
file_ini = outputFolder+'/weights-'+ str(epoch_num)+'*'
filename =  glob.glob(file_ini)
if os.path.isfile(filename[0]):
    model.load_weights(filename[0])
else:
    print "%s does not exist"%filename[0]
# define optimizer
sgd = SGD(lr=0.1)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# checkpoint
outputFolder = './output-mnist'
if not os.path.exists(outputFolder):
    os.makedirs(outputFolder)
filepath=outputFolder+"/weights-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, \
                             save_best_only=False, save_weights_only=True,\
                             mode='auto', period=10)
callbacks_list = [checkpoint]
# train the model
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=100, callbacks=callbacks_list, verbose=0, \
                       validation_split=0.2, initial_epoch = epoch_num+1)
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```

Early Stopping

In [ ]:

```
from keras.callbacks import EarlyStopping
#define model
model = deep_nn()
# define optimizer
sgd = SGD(lr=0.1)
model.compile(optimizer=sgd, loss='mse', metrics=['accuracy'])
# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience=5, \
                          verbose=1, mode='auto')
callbacks_list = [earlystop]
# train the model
start = time.time()
model_info = model.fit(train_features, train_labels, batch_size=128, \
                       nb_epoch=100, callbacks=callbacks_list, verbose=0, \
                       validation_split=0.2)
end = time.time()
# plot model history
plot_model_history(model_info)
# compute test accuracy
print("Accuracy on test data is: %0.2f"%accuracy(test_features, test_labels, model))
```