MNIST MLP

In [9]:

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

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

In [14]:

```
keras.utils.to_categorical(y_train, 10)[:3]
```

Out[14]:

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

MODEL 1 : 2 Layers

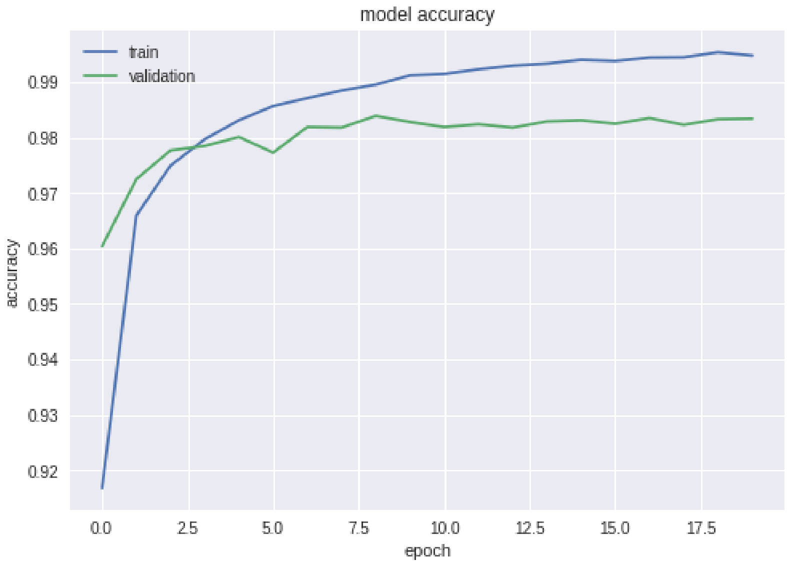
- layer1 -- 512
- layer2 -- 128
- DropOut -- 0.2

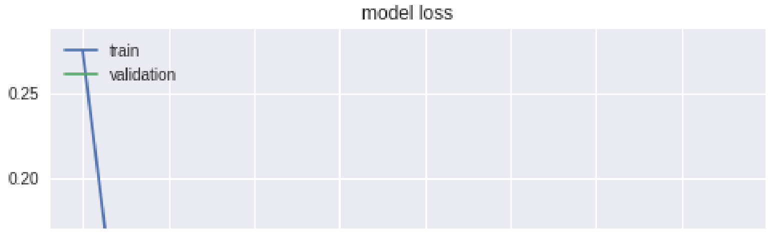
In [15]:

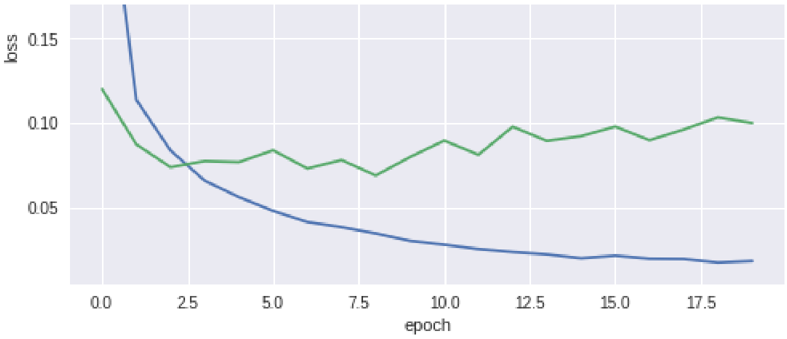
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])


print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_22 (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 53us/step - loss: 0.2751 - acc: 0.9168 -
val_loss: 0.1199 - val_acc: 0.9604
Epoch 2/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1135 - acc: 0.9659 -
val_loss: 0.0871 - val_acc: 0.9725
Epoch 3/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0835 - acc: 0.9750 -
val_loss: 0.0736 - val_acc: 0.9777
Epoch 4/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0656 - acc: 0.9797 -
val_loss: 0.0772 - val_acc: 0.9785
Epoch 5/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0558 - acc: 0.9831 -
val_loss: 0.0766 - val_acc: 0.9801
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0477 - acc: 0.9857 -
val_loss: 0.0836 - val_acc: 0.9773
```

```
Epoch 7/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0411 - acc: 0.9871 -
val_loss: 0.0728 - val_acc: 0.9819
Epoch 8/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0381 - acc: 0.9885 -
val_loss: 0.0778 - val_acc: 0.9818
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0343 - acc: 0.9896 -
val_loss: 0.0687 - val_acc: 0.9839
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0300 - acc: 0.9912 -
val_loss: 0.0796 - val_acc: 0.9828
Epoch 11/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0277 - acc: 0.9915 -
val_loss: 0.0894 - val_acc: 0.9819
Epoch 12/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0251 - acc: 0.9923 -
val_loss: 0.0810 - val_acc: 0.9824
Epoch 13/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0234 - acc: 0.9930 -
val_loss: 0.0976 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0219 - acc: 0.9933 -
val_loss: 0.0892 - val_acc: 0.9829
Epoch 15/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0196 - acc: 0.9940 -
val_loss: 0.0920 - val_acc: 0.9831
Epoch 16/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0211 - acc: 0.9938 -
val_loss: 0.0977 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0193 - acc: 0.9944 -
val_loss: 0.0896 - val_acc: 0.9835
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0192 - acc: 0.9945 -
val_loss: 0.0959 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0171 - acc: 0.9954 -
val_loss: 0.1031 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0180 - acc: 0.9948 -
val_loss: 0.0998 - val_acc: 0.9834
Test loss: 0.09979067730780562
Test accuracy: 0.9834
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```







In [0]:

```
class Dense(Layer):
...
    def __init__(self, units,
                 activation=None,
                 use_bias=True,
                 kernel_initializer='glorot_uniform',
                 bias_initializer='zeros',
                 kernel_regularizer=None,
                 bias_regularizer=None,
                 activity_regularizer=None,
                 kernel_constraint=None,
                 bias_constraint=None,
                 **kwargs):
```

MODEL 1 : 2 Layers

- layer1 -- 512 ,he-normal initalization
- layer2 -- 128 ,he-normal initalization
- DropOut -- 0.2

In [18]:

```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,),kernel_initializer='he_normal'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu',kernel_initializer='he_normal'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_28 (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 56us/step - loss: 0.2791 - acc: 0.9153 -
val_loss: 0.1181 - val_acc: 0.9622
Epoch 2/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1151 - acc: 0.9653 -
val_loss: 0.0844 - val_acc: 0.9751
Epoch 3/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0840 - acc: 0.9748 -
val_loss: 0.0814 - val_acc: 0.9767
Epoch 4/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0679 - acc: 0.9793 -
val_loss: 0.0778 - val_acc: 0.9772
Epoch 5/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0559 - acc: 0.9832 -
val_loss: 0.0717 - val_acc: 0.9816
Epoch 6/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0484 - acc: 0.9848 -
val_loss: 0.0758 - val_acc: 0.9793
Epoch 7/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0414 - acc: 0.9874 -
val_loss: 0.0742 - val_acc: 0.9818
Epoch 8/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0387 - acc: 0.9886 -
val_loss: 0.0899 - val_acc: 0.9807
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0357 - acc: 0.9897 -
val_loss: 0.0724 - val_acc: 0.9833
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0304 - acc: 0.9907 -
val_loss: 0.0788 - val_acc: 0.9827
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0285 - acc: 0.9916 -
val_loss: 0.0754 - val_acc: 0.9840
Epoch 12/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0265 - acc: 0.9925 -
val_loss: 0.0779 - val_acc: 0.9843
```

```
Epoch 13/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0262 - acc: 0.9924 -
val_loss: 0.0858 - val_acc: 0.9805
Epoch 14/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0243 - acc: 0.9927 -
val_loss: 0.0834 - val_acc: 0.9841
Epoch 15/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0235 - acc: 0.9934 -
val_loss: 0.0906 - val_acc: 0.9830
Epoch 16/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0213 - acc: 0.9942 -
val_loss: 0.0929 - val_acc: 0.9844
Epoch 17/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0190 - acc: 0.9940 -
val_loss: 0.0919 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0211 - acc: 0.9942 -
val_loss: 0.0907 - val_acc: 0.9838
Epoch 19/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0190 - acc: 0.9941 -
val_loss: 0.0955 - val_acc: 0.9841
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0186 - acc: 0.9945 -
val_loss: 0.0967 - val_acc: 0.9842
Test loss: 0.09672942254301424
Test accuracy: 0.9842
```

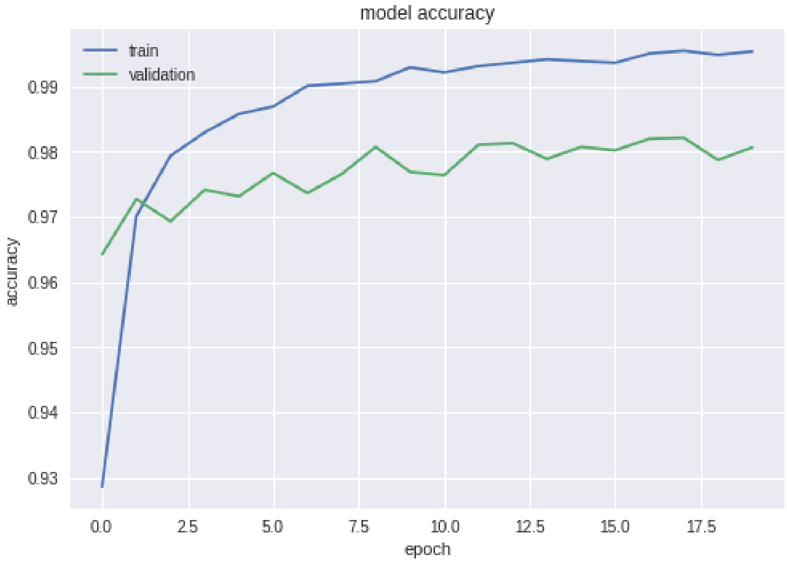
MODEL 1 : 2 Layers

- layer1 -- 1024
- layer2 -- 1024
- DropOut -- 0.2

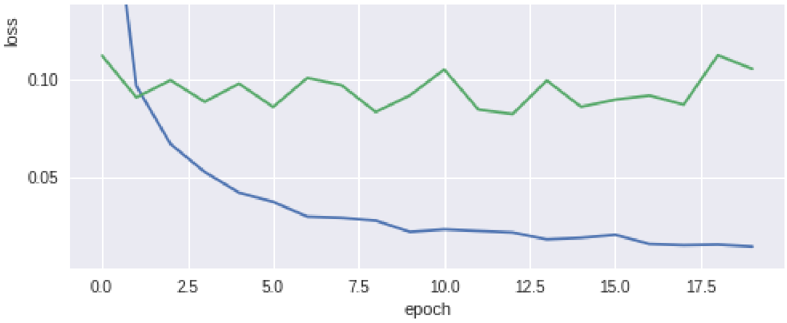
In [19]:

```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 1024)              803840    
_________________________________________________________________
dropout_19 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_30 (Dense)             (None, 1024)              1049600   
_________________________________________________________________
dropout_20 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_31 (Dense)             (None, 10)                10250     
=================================================================
Total params: 1,863,690
Trainable params: 1,863,690
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 4s 90us/step - loss: 0.2377 - acc: 0.9287 -
val_loss: 0.1122 - val_acc: 0.9642
Epoch 2/20
48000/48000 [==============================] - 3s 68us/step - loss: 0.0967 - acc: 0.9701 -
val_loss: 0.0908 - val_acc: 0.9728
Epoch 3/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0669 - acc: 0.9794 -
val_loss: 0.0996 - val_acc: 0.9693
Epoch 4/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0525 - acc: 0.9830 -
val_loss: 0.0886 - val_acc: 0.9742
Epoch 5/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0419 - acc: 0.9858 -
val_loss: 0.0978 - val_acc: 0.9732
Epoch 6/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0372 - acc: 0.9869 -
val_loss: 0.0859 - val_acc: 0.9768
Epoch 7/20
Epoch 7/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0296 - acc: 0.9901 -
val_loss: 0.1008 - val_acc: 0.9737
Epoch 8/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0290 - acc: 0.9905 -
val_loss: 0.0970 - val_acc: 0.9766
Epoch 9/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0275 - acc: 0.9908 -
val_loss: 0.0833 - val_acc: 0.9807
Epoch 10/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0217 - acc: 0.9930 -
val_loss: 0.0919 - val_acc: 0.9769
Epoch 11/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0231 - acc: 0.9922 -
val_loss: 0.1051 - val_acc: 0.9764
Epoch 12/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0222 - acc: 0.9931 -
val_loss: 0.0846 - val_acc: 0.9811
Epoch 13/20
48000/48000 [==============================] - 3s 67us/step - loss: 0.0214 - acc: 0.9937 -
val_loss: 0.0823 - val_acc: 0.9813
Epoch 14/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0179 - acc: 0.9942 -
val_loss: 0.0994 - val_acc: 0.9789
Epoch 15/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0187 - acc: 0.9939 -
val_loss: 0.0860 - val_acc: 0.9807
Epoch 16/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0203 - acc: 0.9937 -
val_loss: 0.0897 - val_acc: 0.9802
Epoch 17/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0155 - acc: 0.9951 -
val_loss: 0.0918 - val_acc: 0.9820
Epoch 18/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0149 - acc: 0.9955 -
val_loss: 0.0871 - val_acc: 0.9822
Epoch 19/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0152 - acc: 0.9949 -
val_loss: 0.1124 - val_acc: 0.9787
Epoch 20/20
48000/48000 [==============================] - 3s 66us/step - loss: 0.0142 - acc: 0.9954 -
val_loss: 0.1055 - val_acc: 0.9807
Test loss: 0.09314459968612027
Test accuracy: 0.9833
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



model loss
train
validation
0.20
0.15



In [0]:

MODEL 1 : 2 Layers

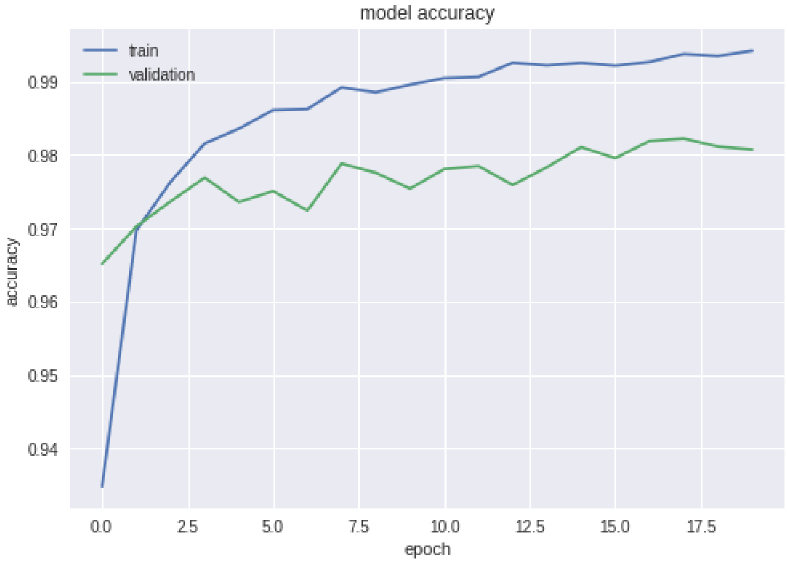
- layer1 -- 1024
  - batch Normalazation
  - Drop Out
- layer2 -- 1024
  - batch Normalazation
  - Drop Out

In [21]:

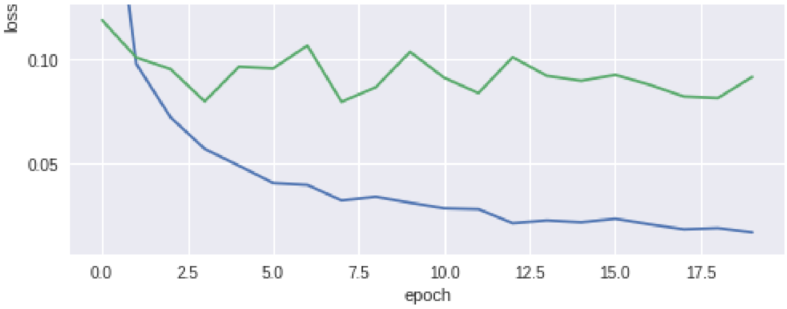
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(784,)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_7 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_22 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_36 (Dense)             (None, 1024)              1049600   
_________________________________________________________________
batch_normalization_8 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_23 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_37 (Dense)             (None, 10)                10250     
=================================================================
Total params: 1,871,882
Trainable params: 1,867,786
Non-trainable params: 4,096
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 6s 134us/step - loss: 0.2259 - acc: 0.9348 -
val_loss: 0.1189 - val_acc: 0.9652
Epoch 2/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0979 - acc: 0.9697 -
val_loss: 0.1010 - val_acc: 0.9702
Epoch 3/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0723 - acc: 0.9763 -
val_loss: 0.0954 - val_acc: 0.9737
Epoch 4/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0572 - acc: 0.9816 -
val_loss: 0.0800 - val_acc: 0.9769
Epoch 5/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0491 - acc: 0.9836 -
val_loss: 0.0966 - val_acc: 0.9736
Epoch 6/20
48000/48000 [==============================] - 5s 101us/step - loss: 0.0409 - acc: 0.9861 -
val_loss: 0.0958 - val_acc: 0.9751
Epoch 7/20
```

```
48000/48000 [==============================] - 5s 102us/step - loss: 0.0400 - acc: 0.9863 -
val_loss: 0.1067 - val_acc: 0.9724
Epoch 8/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0325 - acc: 0.9892 -
val_loss: 0.0798 - val_acc: 0.9788
Epoch 9/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0343 - acc: 0.9886 -
val_loss: 0.0867 - val_acc: 0.9776
Epoch 10/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0314 - acc: 0.9896 -
val_loss: 0.1037 - val_acc: 0.9754
Epoch 11/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0287 - acc: 0.9905 -
val_loss: 0.0913 - val_acc: 0.9781
Epoch 12/20
48000/48000 [==============================] - 5s 101us/step - loss: 0.0283 - acc: 0.9907 -
val_loss: 0.0840 - val_acc: 0.9785
Epoch 13/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0216 - acc: 0.9926 -
val_loss: 0.1011 - val_acc: 0.9759
Epoch 14/20
48000/48000 [==============================] - 5s 101us/step - loss: 0.0229 - acc: 0.9923 -
val_loss: 0.0922 - val_acc: 0.9783
Epoch 15/20
48000/48000 [==============================] - 5s 101us/step - loss: 0.0220 - acc: 0.9926 -
val_loss: 0.0898 - val_acc: 0.9811
Epoch 16/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0237 - acc: 0.9922 -
val_loss: 0.0927 - val_acc: 0.9796
Epoch 17/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0211 - acc: 0.9927 -
val_loss: 0.0880 - val_acc: 0.9819
Epoch 18/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0186 - acc: 0.9938 -
val_loss: 0.0823 - val_acc: 0.9822
Epoch 19/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0192 - acc: 0.9935 -
val_loss: 0.0816 - val_acc: 0.9812
Epoch 20/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0172 - acc: 0.9942 -
val_loss: 0.0917 - val_acc: 0.9807
Test loss: 0.078490325065972
Test accuracy: 0.9815
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



model loss
train
validation
0.20
0.15



MODEL 1 : 2 Layers

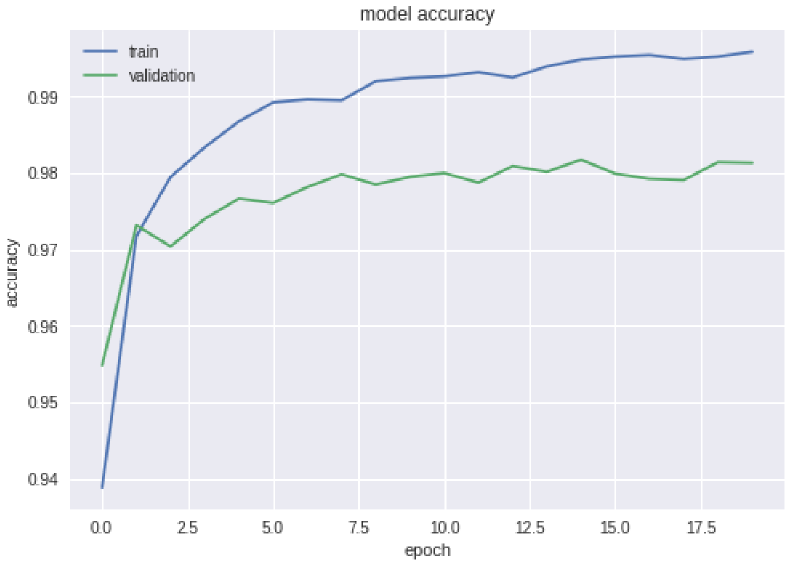
- layer1 -- 1024
    - batch Normalazation
    - Activation
    - Drop Out
- layer2 -- 1024
    - batch Normalazation
    - Activation
    - Drop Out

In [22]:

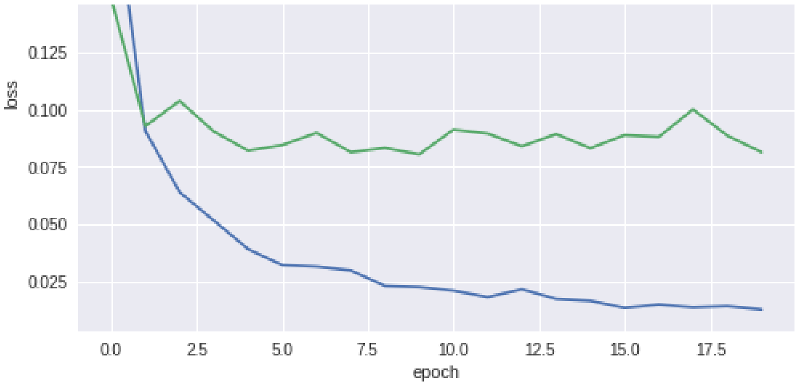
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, input_shape=(784,)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_38 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_9 (Batch (None, 1024)              4096      
_________________________________________________________________
activation_4 (Activation)    (None, 1024)              0         
_________________________________________________________________
dropout_24 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_39 (Dense)             (None, 1024)              1049600   
_________________________________________________________________
batch_normalization_10 (Batc (None, 1024)              4096      
_________________________________________________________________
activation_5 (Activation)    (None, 1024)              0         
_________________________________________________________________
dropout_25 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_40 (Dense)             (None, 10)                10250     
=================================================================
Total params: 1,871,882
Trainable params: 1,867,786
Non-trainable params: 4,096
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 7s 140us/step - loss: 0.2029 - acc: 0.9389 -
val_loss: 0.1485 - val_acc: 0.9548
Epoch 2/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0906 - acc: 0.9717 -
val_loss: 0.0928 - val_acc: 0.9732
Epoch 3/20
48000/48000 [==============================] - 5s 102us/step - loss: 0.0640 - acc: 0.9795 -
val_loss: 0.1038 - val_acc: 0.9704
Epoch 4/20
48000/48000 [==============================] - 5s 107us/step - loss: 0.0516 - acc: 0.9834 -
val_loss: 0.0904 - val_acc: 0.9740
Epoch 5/20
48000/48000 [==============================] - 7s 138us/step - loss: 0.0391 - acc: 0.9868 -
val_loss: 0.0821 - val_acc: 0.9767
Epoch 6/20
```

```
48000/48000 [==============================] - 7s 138us/step - loss: 0.0322 - acc: 0.9893 -
val_loss: 0.0845 - val_acc: 0.9761
Epoch 7/20
48000/48000 [==============================] - 7s 138us/step - loss: 0.0316 - acc: 0.9896 -
val_loss: 0.0898 - val_acc: 0.9782
Epoch 8/20
48000/48000 [==============================] - 7s 137us/step - loss: 0.0299 - acc: 0.9895 -
val_loss: 0.0814 - val_acc: 0.9798
Epoch 9/20
48000/48000 [==============================] - 7s 136us/step - loss: 0.0231 - acc: 0.9920 -
val_loss: 0.0833 - val_acc: 0.9785
Epoch 10/20
48000/48000 [==============================] - 6s 133us/step - loss: 0.0226 - acc: 0.9925 -
val_loss: 0.0805 - val_acc: 0.9795
Epoch 11/20
48000/48000 [==============================] - 7s 140us/step - loss: 0.0211 - acc: 0.9927 -
val_loss: 0.0912 - val_acc: 0.9800
Epoch 12/20
48000/48000 [==============================] - 6s 134us/step - loss: 0.0182 - acc: 0.9932 -
val_loss: 0.0895 - val_acc: 0.9787
Epoch 13/20
48000/48000 [==============================] - 6s 135us/step - loss: 0.0216 - acc: 0.9925 -
val_loss: 0.0840 - val_acc: 0.9809
Epoch 14/20
48000/48000 [==============================] - 7s 139us/step - loss: 0.0174 - acc: 0.9940 -
val_loss: 0.0893 - val_acc: 0.9802
Epoch 15/20
48000/48000 [==============================] - 6s 135us/step - loss: 0.0166 - acc: 0.9949 -
val_loss: 0.0832 - val_acc: 0.9818
Epoch 16/20
48000/48000 [==============================] - 7s 136us/step - loss: 0.0136 - acc: 0.9952 -
val_loss: 0.0888 - val_acc: 0.9799
Epoch 17/20
48000/48000 [==============================] - 7s 137us/step - loss: 0.0150 - acc: 0.9955 -
val_loss: 0.0881 - val_acc: 0.9792
Epoch 18/20
48000/48000 [==============================] - 7s 138us/step - loss: 0.0138 - acc: 0.9949 -
val_loss: 0.1001 - val_acc: 0.9791
Epoch 19/20
48000/48000 [==============================] - 7s 139us/step - loss: 0.0143 - acc: 0.9952 -
val_loss: 0.0886 - val_acc: 0.9814
Epoch 20/20
48000/48000 [==============================] - 7s 136us/step - loss: 0.0128 - acc: 0.9959 -
val_loss: 0.0814 - val_acc: 0.9813
Test loss: 0.06953526535720353
Test accuracy: 0.9824
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



model loss
train
validation
0.200
0.175
0.150



MODEL 1 : 3 Layers

- layer1 -- 1024
    - batch Normalazation
    - Activation
    - Drop Out (0.2)
- layer2 -- 512
    - batch Normalazation
    - Activation
    - Drop Out (0.2)
    - layer3 -- 128
        - batch Normalazation
        - Activation
        - Drop Out(0.4)

In [0]:

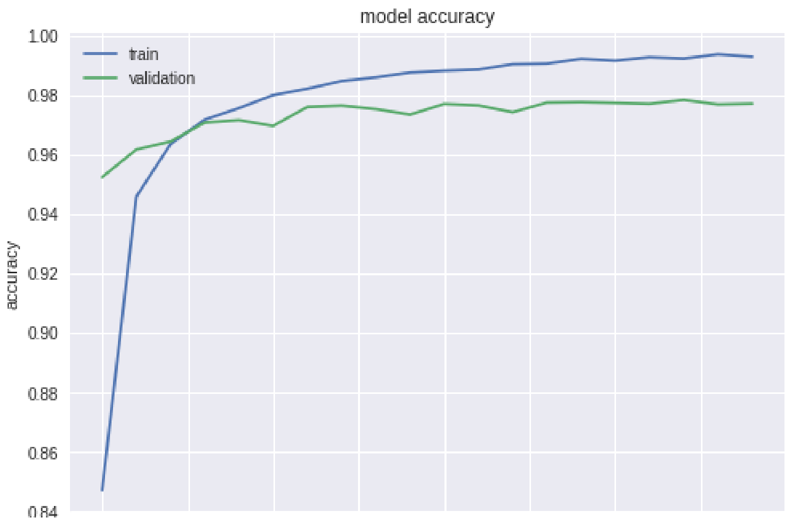
```

```

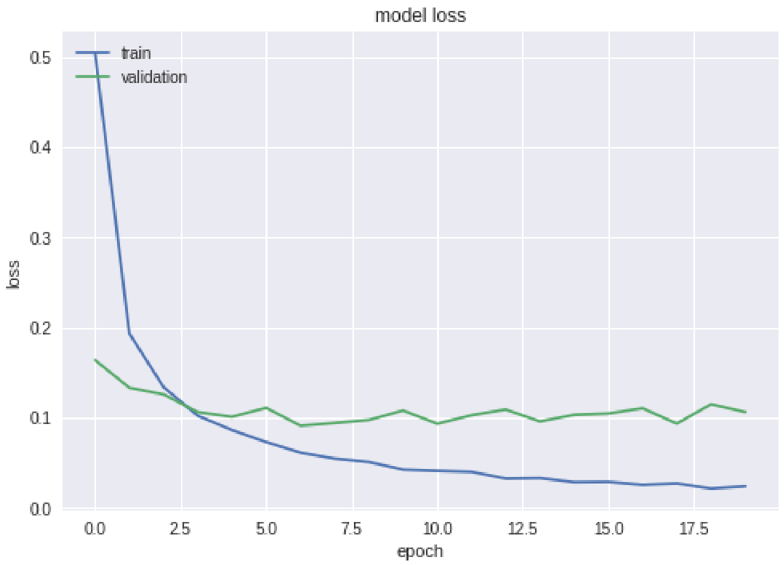
In [5]:

```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model = Sequential()
model.add(Dense(1024, activation='relu', input_shape=(784,)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.33)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 64)                8256      
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                650       
=================================================================
Total params: 476,490
Trainable params: 476,490
Non-trainable params: 0
_________________________________________________________________
Train on 40199 samples, validate on 19801 samples
Epoch 1/20
40199/40199 [==============================] - 3s 69us/step - loss: 0.5041 - acc: 0.8471 - 
val_loss: 0.1638 - val_acc: 0.9523
Epoch 2/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.1933 - acc: 0.9458 - 
val_loss: 0.1330 - val_acc: 0.9617
Epoch 3/20
```

```
40199/40199 [==============================] - 2s 55us/step - loss: 0.1338 - acc: 0.9635 -
val_loss: 0.1257 - val_acc: 0.9643
Epoch 4/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.1020 - acc: 0.9717 -
val_loss: 0.1060 - val_acc: 0.9707
Epoch 5/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0862 - acc: 0.9756 -
val_loss: 0.1008 - val_acc: 0.9715
Epoch 6/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0729 - acc: 0.9799 -
val_loss: 0.1107 - val_acc: 0.9696
Epoch 7/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0612 - acc: 0.9821 -
val_loss: 0.0910 - val_acc: 0.9760
Epoch 8/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0545 - acc: 0.9847 -
val_loss: 0.0940 - val_acc: 0.9764
Epoch 9/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0508 - acc: 0.9859 -
val_loss: 0.0973 - val_acc: 0.9753
Epoch 10/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0423 - acc: 0.9875 -
val_loss: 0.1078 - val_acc: 0.9734
Epoch 11/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0409 - acc: 0.9881 -
val_loss: 0.0932 - val_acc: 0.9770
Epoch 12/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0398 - acc: 0.9886 -
val_loss: 0.1025 - val_acc: 0.9765
Epoch 13/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0324 - acc: 0.9903 -
val_loss: 0.1089 - val_acc: 0.9742
Epoch 14/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0329 - acc: 0.9905 -
val_loss: 0.0957 - val_acc: 0.9774
Epoch 15/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0284 - acc: 0.9921 -
val_loss: 0.1030 - val_acc: 0.9776
Epoch 16/20
40199/40199 [==============================] - 2s 55us/step - loss: 0.0287 - acc: 0.9915 -
val_loss: 0.1044 - val_acc: 0.9773
Epoch 17/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0254 - acc: 0.9927 -
val_loss: 0.1103 - val_acc: 0.9770
Epoch 18/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0270 - acc: 0.9922 -
val_loss: 0.0933 - val_acc: 0.9783
Epoch 19/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0213 - acc: 0.9937 -
val_loss: 0.1147 - val_acc: 0.9768
Epoch 20/20
40199/40199 [==============================] - 2s 56us/step - loss: 0.0238 - acc: 0.9928 -
val_loss: 0.1061 - val_acc: 0.9771
Test loss: 0.0902806506058223
Test accuracy: 0.9803
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





In [0]:

```

```

MODEL 1 : 3 Layers

- layer1 -- 512
  - batch Normalazation
  - Activation
  - Drop Out (0.2)
- layer2 -- 128
  - batch Normalazation
  - Activation
  - Drop Out (0.2)
- layer3 -- 64
  - batch Normalazation
  - Activation
  - Drop Out(0.4)

In [23]:

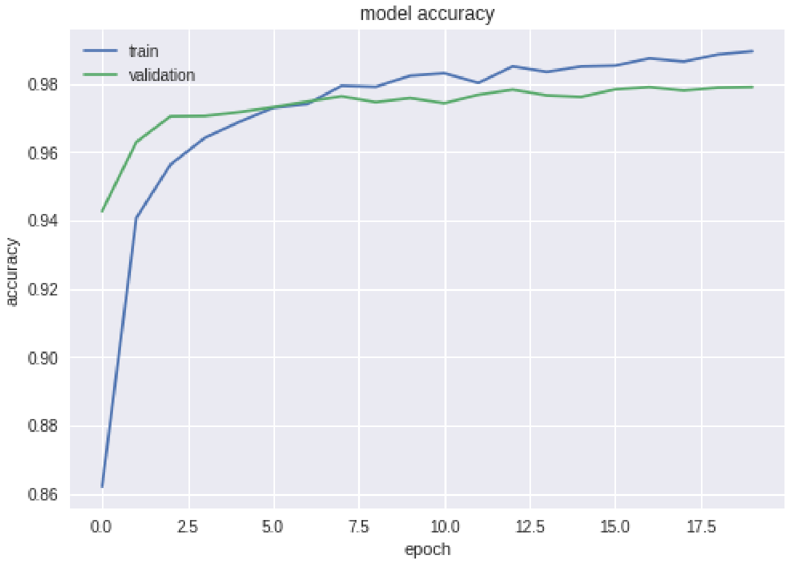
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,BatchNormalization ,Activation
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.33)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

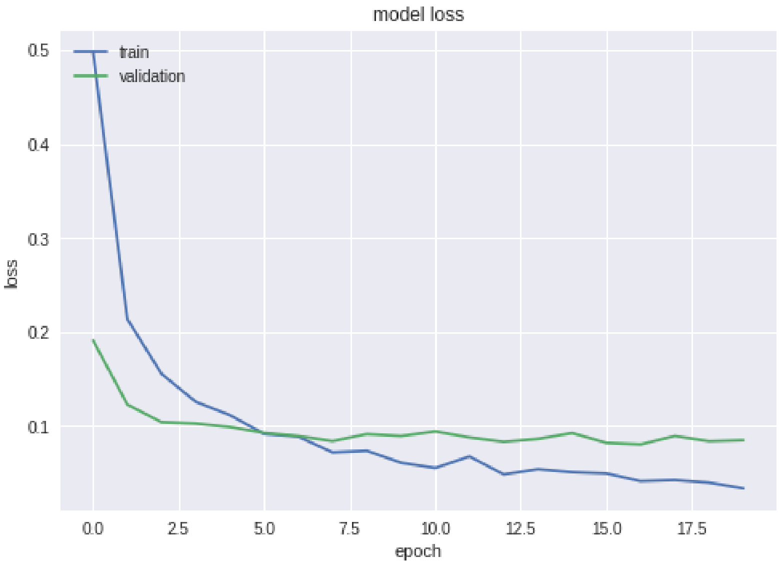
```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
activation_6 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_26 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_42 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_12 (Batc (None, 128)               512
_________________________________________________________________
activation_7 (Activation)    (None, 128)               0
```

```
_________________________________________________________________
dropout_27 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_43 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_13 (Batc (None, 64)                256
_________________________________________________________________
activation_8 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_28 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_44 (Dense)             (None, 10)                650
=================================================================
Total params: 479,306
Trainable params: 477,898
Non-trainable params: 1,408
_________________________________________________________________
Train on 40199 samples, validate on 19801 samples
Epoch 1/20
40199/40199 [==============================] - 9s 234us/step - loss: 0.4978 - acc: 0.8620 -
val_loss: 0.1910 - val_acc: 0.9427
Epoch 2/20
40199/40199 [==============================] - 6s 143us/step - loss: 0.2136 - acc: 0.9407 -
val_loss: 0.1226 - val_acc: 0.9629
Epoch 3/20
40199/40199 [==============================] - 6s 148us/step - loss: 0.1553 - acc: 0.9563 -
val_loss: 0.1038 - val_acc: 0.9705
Epoch 4/20
40199/40199 [==============================] - 6s 148us/step - loss: 0.1255 - acc: 0.9642 -
val_loss: 0.1024 - val_acc: 0.9706
Epoch 5/20
40199/40199 [==============================] - 6s 144us/step - loss: 0.1113 - acc: 0.9688 -
val_loss: 0.0986 - val_acc: 0.9717
Epoch 6/20
40199/40199 [==============================] - 6s 144us/step - loss: 0.0912 - acc: 0.9729 -
val_loss: 0.0924 - val_acc: 0.9731
Epoch 7/20
40199/40199 [==============================] - 6s 146us/step - loss: 0.0883 - acc: 0.9741 -
val_loss: 0.0891 - val_acc: 0.9748
Epoch 8/20
40199/40199 [==============================] - 6s 148us/step - loss: 0.0716 - acc: 0.9794 -
val_loss: 0.0837 - val_acc: 0.9763
Epoch 9/20
40199/40199 [==============================] - 6s 148us/step - loss: 0.0733 - acc: 0.9791 -
val_loss: 0.0911 - val_acc: 0.9746
Epoch 10/20
40199/40199 [==============================] - 6s 146us/step - loss: 0.0606 - acc: 0.9823 -
val_loss: 0.0890 - val_acc: 0.9758
Epoch 11/20
40199/40199 [==============================] - 6s 150us/step - loss: 0.0551 - acc: 0.9831 -
val_loss: 0.0940 - val_acc: 0.9743
Epoch 12/20
40199/40199 [==============================] - 6s 148us/step - loss: 0.0673 - acc: 0.9802 -
val_loss: 0.0875 - val_acc: 0.9768
Epoch 13/20
40199/40199 [==============================] - 6s 149us/step - loss: 0.0482 - acc: 0.9851 -
val_loss: 0.0829 - val_acc: 0.9783
Epoch 14/20
40199/40199 [==============================] - 6s 149us/step - loss: 0.0537 - acc: 0.9835 -
val_loss: 0.0860 - val_acc: 0.9766
Epoch 15/20
40199/40199 [==============================] - 6s 146us/step - loss: 0.0506 - acc: 0.9851 -
val_loss: 0.0922 - val_acc: 0.9761
Epoch 16/20
40199/40199 [==============================] - 6s 147us/step - loss: 0.0492 - acc: 0.9853 -
val_loss: 0.0818 - val_acc: 0.9784
Epoch 17/20
40199/40199 [==============================] - 6s 147us/step - loss: 0.0411 - acc: 0.9874 -
val_loss: 0.0802 - val_acc: 0.9790
Epoch 18/20
40199/40199 [==============================] - 6s 150us/step - loss: 0.0422 - acc: 0.9865 -
val_loss: 0.0891 - val_acc: 0.9780
Epoch 19/20
40199/40199 [==============================] - 6s 150us/step - loss: 0.0394 - acc: 0.9886 -
val_loss: 0.0836 - val_acc: 0.9789
Epoch 20/20
40199/40199 [==============================] - 6s 147us/step - loss: 0.0333 - acc: 0.9895 -
```

```
val_loss: 0.0846 - val_acc: 0.9790
Test loss: 0.06931114477205483
Test accuracy: 0.9827
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



model accuracy



model loss

MODEL 1 : 5 Layers

- layer1 -- 1024
  - batch Normalazation
  - Activation
  - Drop Out (0.2)
- layer2 -- 512
  - batch Normalazation
  - Activation
  - Drop Out (0.2)
- layer3 -- 128
  - batch Normalazation
  - Activation
  - Drop Out(0.4)
- layer4 -- 64
  - batch Normalazation
  - Activation
  - Drop Out (0.5)
- layer5 -- 64
  - batch Normalazation
  - Activation
  - Drop Out(0.5)

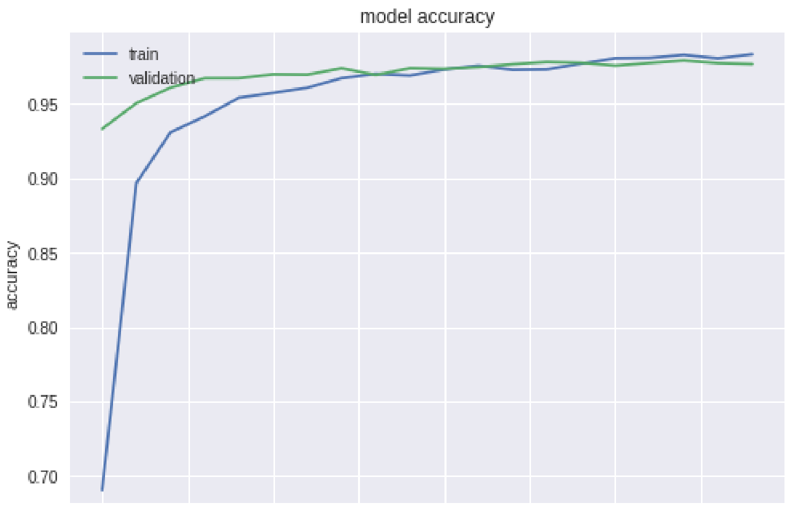
In [24]:

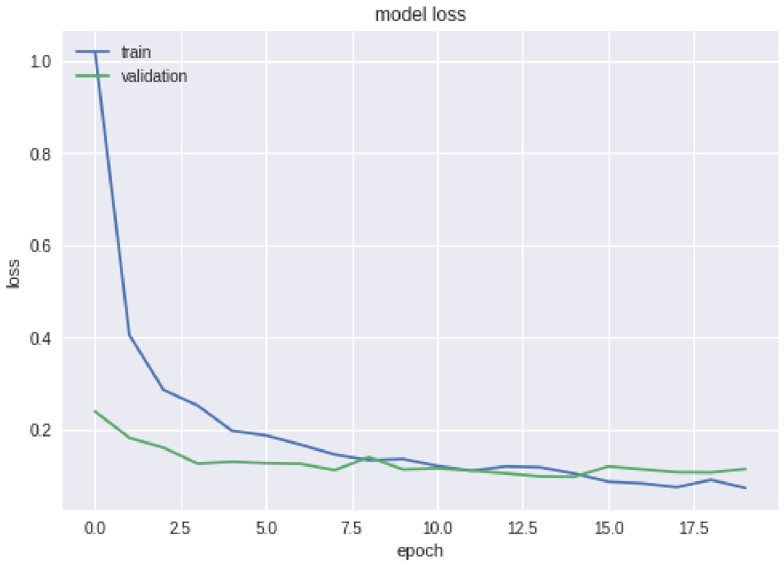
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,BatchNormalization ,Activation
from keras.optimizers import Adam

batch_size = 128
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, input_shape=(784,)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.33)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())
```

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_45 (Dense)             (None, 1024)              803840
_________________________________________________________________
batch_normalization_14 (Batc (None, 1024)              4096
_________________________________________________________________
activation_9 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_29 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_46 (Dense)             (None, 512)               524800
_________________________________________________________________
batch_normalization_15 (Batc (None, 512)               2048
_________________________________________________________________
activation_10 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_30 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_47 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_16 (Batc (None, 128)               512
_________________________________________________________________
activation_11 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_31 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_48 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_17 (Batc (None, 64)                256
_________________________________________________________________
activation_12 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_32 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_49 (Dense)             (None, 64)                4160
_________________________________________________________________
batch_normalization_18 (Batc (None, 64)                256
_________________________________________________________________
activation_13 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_33 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_50 (Dense)             (None, 10)                650
=================================================================
Total params: 1,414,538
Trainable params: 1,410,954
Non-trainable params: 3,584
_________________________________________________________________
Train on 40199 samples, validate on 19801 samples
Epoch 1/20
40199/40199 [==============================] - 14s 348us/step - loss: 1.0198 - acc: 0.6905 - val_l
oss: 0.2388 - val_acc: 0.9332
Epoch 2/20
40199/40199 [==============================] - 9s 224us/step - loss: 0.4052 - acc: 0.8968 -
val_loss: 0.1815 - val_acc: 0.9505
```

```
Epoch 3/20
40199/40199 [==============================] - 9s 223us/step - loss: 0.2859 - acc: 0.9309 -
val_loss: 0.1603 - val_acc: 0.9610
Epoch 4/20
40199/40199 [==============================] - 9s 218us/step - loss: 0.2516 - acc: 0.9416 -
val_loss: 0.1252 - val_acc: 0.9673
Epoch 5/20
40199/40199 [==============================] - 8s 208us/step - loss: 0.1967 - acc: 0.9542 -
val_loss: 0.1296 - val_acc: 0.9674
Epoch 6/20
40199/40199 [==============================] - 9s 215us/step - loss: 0.1866 - acc: 0.9575 -
val_loss: 0.1263 - val_acc: 0.9699
Epoch 7/20
40199/40199 [==============================] - 9s 219us/step - loss: 0.1669 - acc: 0.9609 -
val_loss: 0.1249 - val_acc: 0.9697
Epoch 8/20
40199/40199 [==============================] - 8s 210us/step - loss: 0.1453 - acc: 0.9674 -
val_loss: 0.1112 - val_acc: 0.9740
Epoch 9/20
40199/40199 [==============================] - 9s 222us/step - loss: 0.1326 - acc: 0.9702 -
val_loss: 0.1394 - val_acc: 0.9695
Epoch 10/20
40199/40199 [==============================] - 9s 220us/step - loss: 0.1354 - acc: 0.9691 -
val_loss: 0.1128 - val_acc: 0.9740
Epoch 11/20
40199/40199 [==============================] - 9s 218us/step - loss: 0.1204 - acc: 0.9732 -
val_loss: 0.1152 - val_acc: 0.9736
Epoch 12/20
40199/40199 [==============================] - 9s 218us/step - loss: 0.1098 - acc: 0.9757 -
val_loss: 0.1101 - val_acc: 0.9747
Epoch 13/20
40199/40199 [==============================] - 9s 216us/step - loss: 0.1193 - acc: 0.9730 -
val_loss: 0.1042 - val_acc: 0.9767
Epoch 14/20
40199/40199 [==============================] - 9s 218us/step - loss: 0.1174 - acc: 0.9732 -
val_loss: 0.0976 - val_acc: 0.9782
Epoch 15/20
40199/40199 [==============================] - 9s 218us/step - loss: 0.1039 - acc: 0.9771 -
val_loss: 0.0966 - val_acc: 0.9777
Epoch 16/20
40199/40199 [==============================] - 9s 220us/step - loss: 0.0861 - acc: 0.9807 -
val_loss: 0.1193 - val_acc: 0.9758
Epoch 17/20
40199/40199 [==============================] - 9s 215us/step - loss: 0.0821 - acc: 0.9809 -
val_loss: 0.1128 - val_acc: 0.9775
Epoch 18/20
40199/40199 [==============================] - 9s 216us/step - loss: 0.0739 - acc: 0.9830 -
val_loss: 0.1068 - val_acc: 0.9792
Epoch 19/20
40199/40199 [==============================] - 9s 216us/step - loss: 0.0900 - acc: 0.9806 -
val_loss: 0.1062 - val_acc: 0.9774
Epoch 20/20
40199/40199 [==============================] - 9s 214us/step - loss: 0.0724 - acc: 0.9834 -
val_loss: 0.1137 - val_acc: 0.9768
Test loss: 0.10103141384326846
Test accuracy: 0.9795
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





In [0]:

```

```

MODEL 1 : 5 Layers

- layer1 -- 1024
  - batch Normalazation
  - Activation
  - Drop Out (0.4)
- layer2 -- 512
  - batch Normalazation
  - Activation
  - Drop Out (0.4)
- layer3 -- 128
  - batch Normalazation
  - Activation
  - Drop Out(0.4)
- layer4 -- 64
  - batch Normalazation
  - Activation
  - Drop Out (0.5)
- layer5 -- 64
  - batch Normalazation
  - Activation
  - Drop Out(0.5)

In [38]:

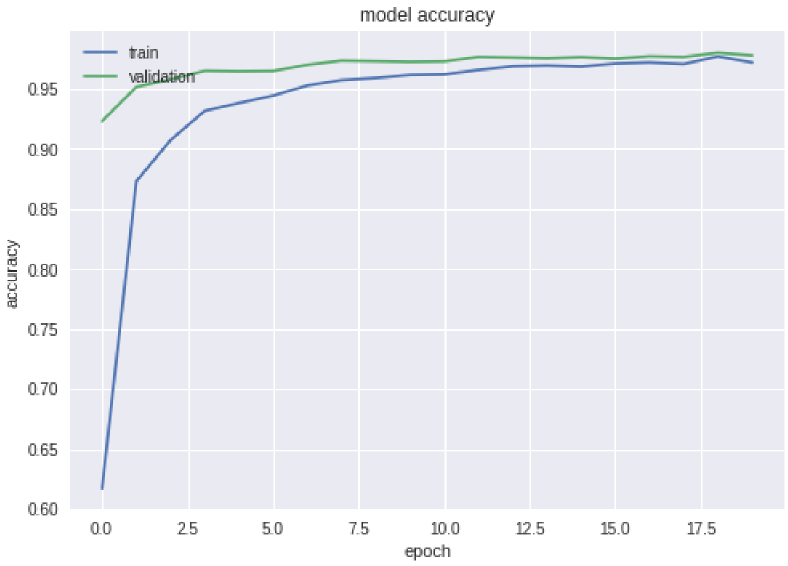
```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''
import matplotlib.pyplot as plt

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,BatchNormalization ,Activation
from keras.optimizers import Adam

batch_size = 128
```

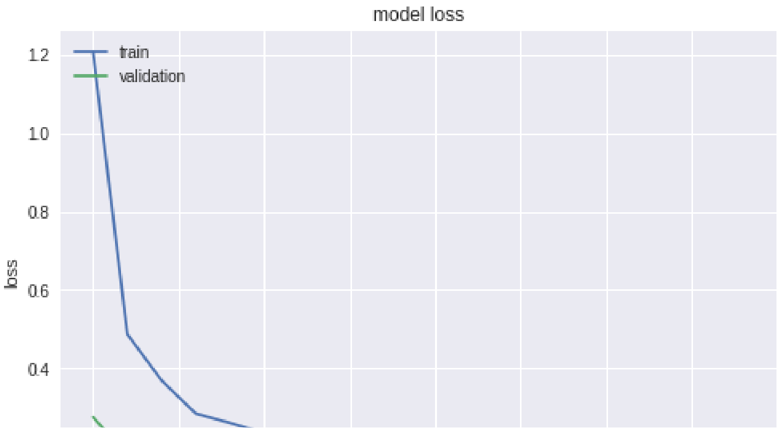
```python
num_classes = 10
epochs = 20

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Dense(1024, input_shape=(784,)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.33)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print(history.history.keys())

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

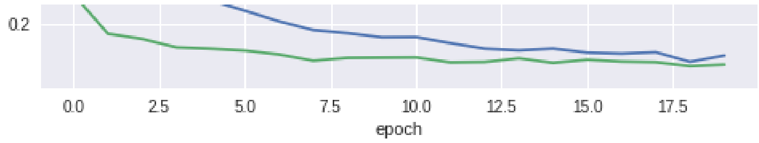
```
60000 train samples
```

```
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_63 (Dense)             (None, 1024)              803840    
_________________________________________________________________
batch_normalization_29 (Batc (None, 1024)              4096      
_________________________________________________________________
activation_24 (Activation)   (None, 1024)              0         
_________________________________________________________________
dropout_44 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_64 (Dense)             (None, 512)               524800    
_________________________________________________________________
batch_normalization_30 (Batc (None, 512)               2048      
_________________________________________________________________
activation_25 (Activation)   (None, 512)               0         
_________________________________________________________________
dropout_45 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_65 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_31 (Batc (None, 128)               512       
_________________________________________________________________
activation_26 (Activation)   (None, 128)               0         
_________________________________________________________________
dropout_46 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_66 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_32 (Batc (None, 64)                256       
_________________________________________________________________
activation_27 (Activation)   (None, 64)                0         
_________________________________________________________________
dropout_47 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_67 (Dense)             (None, 64)                4160      
_________________________________________________________________
batch_normalization_33 (Batc (None, 64)                256       
_________________________________________________________________
activation_28 (Activation)   (None, 64)                0         
_________________________________________________________________
dropout_48 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_68 (Dense)             (None, 10)                650       
=================================================================
Total params: 1,414,538
Trainable params: 1,410,954
Non-trainable params: 3,584
_________________________________________________________________
Train on 40199 samples, validate on 19801 samples
Epoch 1/20
40199/40199 [==============================] - 11s 261us/step - loss: 1.2068 - acc: 0.6170 - val_l
oss: 0.2751 - val_acc: 0.9230
Epoch 2/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.4867 - acc: 0.8729 -
val_loss: 0.1754 - val_acc: 0.9515
Epoch 3/20
40199/40199 [==============================] - 6s 162us/step - loss: 0.3689 - acc: 0.9072 -
val_loss: 0.1621 - val_acc: 0.9578
Epoch 4/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.2840 - acc: 0.9316 -
val_loss: 0.1405 - val_acc: 0.9650
Epoch 5/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.2601 - acc: 0.9381 -
val_loss: 0.1373 - val_acc: 0.9645
Epoch 6/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.2339 - acc: 0.9443 -
val_loss: 0.1323 - val_acc: 0.9648
Epoch 7/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.2068 - acc: 0.9527 -
val_loss: 0.1223 - val_acc: 0.9700
Epoch 8/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1848 - acc: 0.9572 -
val_loss: 0.1065 - val_acc: 0.9734
Epoch 9/20
```

```
40199/40199 [==============================] - 6s 161us/step - loss: 0.1769 - acc: 0.9590 -
val_loss: 0.1138 - val_acc: 0.9729
Epoch 10/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1663 - acc: 0.9615 -
val_loss: 0.1145 - val_acc: 0.9724
Epoch 11/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1667 - acc: 0.9620 -
val_loss: 0.1150 - val_acc: 0.9728
Epoch 12/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1514 - acc: 0.9656 -
val_loss: 0.1021 - val_acc: 0.9765
Epoch 13/20
40199/40199 [==============================] - 6s 160us/step - loss: 0.1373 - acc: 0.9688 -
val_loss: 0.1031 - val_acc: 0.9760
Epoch 14/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.1329 - acc: 0.9694 -
val_loss: 0.1125 - val_acc: 0.9754
Epoch 15/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1375 - acc: 0.9685 -
val_loss: 0.1010 - val_acc: 0.9764
Epoch 16/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.1271 - acc: 0.9710 -
val_loss: 0.1089 - val_acc: 0.9751
Epoch 17/20
40199/40199 [==============================] - 6s 161us/step - loss: 0.1246 - acc: 0.9719 -
val_loss: 0.1042 - val_acc: 0.9771
Epoch 18/20
40199/40199 [==============================] - 7s 162us/step - loss: 0.1281 - acc: 0.9706 -
val_loss: 0.1026 - val_acc: 0.9765
Epoch 19/20
40199/40199 [==============================] - 6s 160us/step - loss: 0.1038 - acc: 0.9769 -
val_loss: 0.0925 - val_acc: 0.9800
Epoch 20/20
40199/40199 [==============================] - 6s 160us/step - loss: 0.1196 - acc: 0.9719 -
val_loss: 0.0965 - val_acc: 0.9777
Test loss: 0.09310851992171229
Test accuracy: 0.9785
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```







In [27]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["No_Of_Layers", 'Dense_Layer_units' , 'Dropout_values', "BatchNormalazation" , "Te
st_loss" , "Test_acc"]

x.add_row(["2", [512,128],[0.2,0.2], ['NO','NO'], 0.0997 , 0.9834 ])
x.add_row(["2", [512,128],[0.2,0.2], ["HE-NOrmalizatio",'NO','NO'], 0.0967 , 0.9842 ])
x.add_row(["2", [1024,1024],[0.2,0.2], ['NO','NO'], 0.0931 , 0.9833 ])
x.add_row(["2", [1024,1024],[0.2,0.2], ['YES','AfterActivationFunction'], 0.0784 , 0.9815 ])
x.add_row(["2", [1024,1024],[0.2,0.2], ['YES','BeforeActivationFunction'], 0.0695 , 0.9824 ])
x.add_row(["3", [1024,512,128],[0.2,0.2,0.4], ['YES','BeforeActivationFunction'], 0.0902 , 0.9803 ]
)
x.add_row(["3", [512,128,64],[0.2,0.2,0.4], ['YES','BeforeActivationFunction'], 0.0693 , 0.9827 ])
x.add_row(["5", [1024,512,128,64,64],[0.2,0.2,0.2,0.4,0.5], ['YES','BeforeActivationFunction'], 0.1
010 , 0.9795 ])
x.add_row(["5", [1024,512,128,64,64],[0.4,0.4,0.4,0.5,0.5], ['YES','BeforeActivationFunction'], 0.0
868 , 0.9796 ])


print(x)
```

```
+--------------+---------------------------+---------------------------+---------------------------
-----+-----------+----------+
| No_Of_Layers |     Dense_Layer_units     |       Dropout_values      |          BatchNormalazatior
| Test_loss | Test_acc |
+--------------+---------------------------+---------------------------+---------------------------
-----+-----------+----------+
|      2       |         [512, 128]        |         [0.2, 0.2]        |              ['NO', 'NO']
|   0.0997  |  0.9834  |
|      2       |         [512, 128]        |         [0.2, 0.2]        |   ['HE-NOrmalizatio', 'NO',
'NO']    |   0.0967  |  0.9842  |
|      2       |        [1024, 1024]       |         [0.2, 0.2]        |              ['NO', 'NO']
|   0.0931  |  0.9833  |
|      2       |        [1024, 1024]       |         [0.2, 0.2]        |  ['YES',
'AfterActivationFunction'] |   0.0784  |  0.9815  |
|      2       |        [1024, 1024]       |         [0.2, 0.2]        | ['YES',
'BeforeActivationFunction'] |   0.0695  |  0.9824  |
|      3       |      [1024, 512, 128]     |      [0.2, 0.2, 0.4]      | ['YES',
'BeforeActivationFunction'] |   0.0902  |  0.9803  |
|      3       |       [512, 128, 64]      |      [0.2, 0.2, 0.4]      | ['YES',
'BeforeActivationFunction'] |   0.0693  |  0.9827  |
|      5       | [1024, 512, 128, 64, 64]  | [0.2, 0.2, 0.2, 0.4, 0.5] | ['YES',
'BeforeActivationFunction'] |   0.101   |  0.9795  |
|      5       | [1024, 512, 128, 64, 64]  | [0.4, 0.4, 0.4, 0.5, 0.5] | ['YES',
'BeforeActivationFunction'] |   0.0868  |  0.9796  |
+--------------+---------------------------+---------------------------+---------------------------
-----+-----------+----------+
```