In [11]:

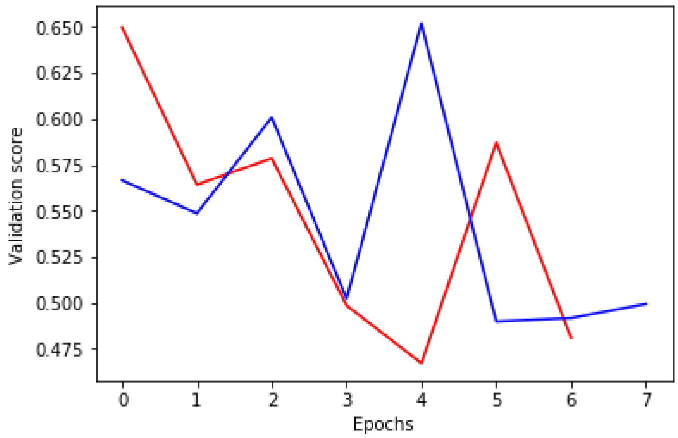
```
# Import necessary modules
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
import pandas as pd
import numpy as np

#pred_data = pd.read_csv('pred_data.csv').as_matrix()
pred_data = pd.read_csv('pred_data.csv').values

df = pd.read_csv('titanic_all_numeric.csv')
#predictors = df.drop(['survived'], axis=1).as_matrix()
predictors = df.drop(['survived'], axis=1).values

n_cols = predictors.shape[1]

# Convert the target to categorical: target
target = to_categorical(df.survived)
# Import the SGD optimizer
from keras.optimizers import SGD
input_shape = (n_cols,)

def get_new_model(input_shape = input_shape):
  model = Sequential()
  model.add(Dense(100, activation='relu', input_shape = input_shape))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(2, activation='softmax'))
  return(model)


# Create list of learning rates: lr_to_test
lr_to_test = [0.000001, 0.01, 1]

# Loop over learning rates
for lr in lr_to_test:
  print('\n\nTesting model with learning rate: %f\n'%lr )

  # Build new model to test, unaffected by previous models
  model = get_new_model()

  # Create SGD optimizer with specified learning rate: my_optimizer
  my_optimizer = SGD(lr=lr)

  # Compile the model
  model.compile(optimizer=my_optimizer, loss='categorical_crossentropy', metrics=['accuracy'
])

  # Fit the model
  model.fit(predictors, target, validation_split=0.3, epochs=10)
```

```
Testing model with learning rate: 0.000001

Train on 623 samples, validate on 268 samples
Epoch 1/10
623/623 [==============================] - 1s 1ms/step - loss: 1.4010 - acc: 0.6051 - val_los
s: 1.5255 - val_acc: 0.6418
Epoch 2/10
623/623 [==============================] - 0s 59us/step - loss: 1.3914 - acc: 0.6051 - val_lo
ss: 1.5177 - val_acc: 0.6418
Epoch 3/10
623/623 [==============================] - 0s 61us/step - loss: 1.3819 - acc: 0.6051 - val_lo
ss: 1.5102 - val_acc: 0.6418
Epoch 4/10
623/623 [==============================] - 0s 59us/step - loss: 1.3727 - acc: 0.6051 - val_lo
ss: 1.5026 - val_acc: 0.6418
Epoch 5/10
623/623 [==============================] - 0s 66us/step - loss: 1.3634 - acc: 0.6051 - val_lo
ss: 1.4951 - val_acc: 0.6418
Epoch 6/10
623/623 [==============================] - 0s 62us/step - loss: 1.3541 - acc: 0.6051 - val_lo
ss: 1.4878 - val_acc: 0.6418
Epoch 7/10
623/623 [==============================] - 0s 62us/step - loss: 1.3452 - acc: 0.6051 - val_lo
ss: 1.4803 - val_acc: 0.6418
Epoch 8/10
623/623 [==============================] - 0s 61us/step - loss: 1.3363 - acc: 0.6051 - val_lo
ss: 1.4724 - val_acc: 0.6418
Epoch 9/10
623/623 [==============================] - 0s 67us/step - loss: 1.3272 - acc: 0.6051 - val_lo
ss: 1.4640 - val_acc: 0.6418
Epoch 10/10
623/623 [==============================] - 0s 64us/step - loss: 1.3181 - acc: 0.6051 - val_lo
ss: 1.4557 - val_acc: 0.6418


Testing model with learning rate: 0.010000

Train on 623 samples, validate on 268 samples
Epoch 1/10
623/623 [==============================] - 1s 1ms/step - loss: 2.5427 - acc: 0.5907 - val_los
s: 2.1156 - val_acc: 0.6418
Epoch 2/10
623/623 [==============================] - 0s 61us/step - loss: 1.2885 - acc: 0.6035 - val_lo
ss: 0.6121 - val_acc: 0.6642
Epoch 3/10
623/623 [==============================] - 0s 67us/step - loss: 0.8634 - acc: 0.6083 - val_lo
ss: 0.5659 - val_acc: 0.7090
Epoch 4/10
623/623 [==============================] - 0s 59us/step - loss: 0.7486 - acc: 0.6469 - val_lo
ss: 0.5464 - val_acc: 0.7351
Epoch 5/10
623/623 [==============================] - 0s 64us/step - loss: 0.6992 - acc: 0.6581 - val_lo
ss: 0.6390 - val_acc: 0.6604
Epoch 6/10
623/623 [==============================] - 0s 62us/step - loss: 0.6656 - acc: 0.6581 - val_lo
ss: 0.9213 - val_acc: 0.6455
Epoch 7/10
623/623 [==============================] - 0s 61us/step - loss: 0.6419 - acc: 0.6806 - val_lo
ss: 0.5680 - val_acc: 0.7015
Epoch 8/10
623/623 [==============================] - 0s 56us/step - loss: 0.6434 - acc: 0.6758 - val_lo
ss: 0.5387 - val_acc: 0.7239
Epoch 9/10
623/623 [==============================] - 0s 61us/step - loss: 0.6342 - acc: 0.6709 - val_lo
ss: 0.5869 - val_acc: 0.6978
```

```
Epoch 10/10
623/623 [==============================] - 0s 64us/step - loss: 0.6169 - acc: 0.6870 - val_lo
ss: 0.5297 - val_acc: 0.7351


Testing model with learning rate: 1.000000

Train on 623 samples, validate on 268 samples
Epoch 1/10
623/623 [==============================] - 1s 2ms/step - loss: 9.3286 - acc: 0.4029 - val_los
s: 10.3444 - val_acc: 0.3582
Epoch 2/10
623/623 [==============================] - 0s 58us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 3/10
623/623 [==============================] - 0s 61us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 4/10
623/623 [==============================] - 0s 64us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 5/10
623/623 [==============================] - 0s 59us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 6/10
623/623 [==============================] - 0s 58us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 7/10
623/623 [==============================] - 0s 70us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 8/10
623/623 [==============================] - 0s 77us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 9/10
623/623 [==============================] - 0s 61us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
Epoch 10/10
623/623 [==============================] - 0s 61us/step - loss: 9.7536 - acc: 0.3949 - val_lo
ss: 10.3444 - val_acc: 0.3582
```

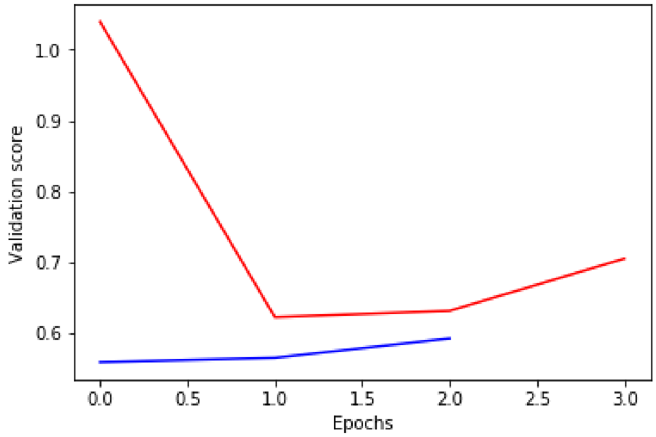
In [9]:
```python
# Import necessary modules
#import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
import pandas as pd
#import numpy as np
#from keras.optimizers import SGD
import matplotlib.pyplot as plt

#pred_data = pd.read_csv('pred_data.csv').as_matrix()
pred_data = pd.read_csv('pred_data.csv').values

df = pd.read_csv('titanic_all_numeric.csv')
#predictors = df.drop(['survived'], axis=1).as_matrix()
predictors = df.drop(['survived'], axis=1).values
n_cols = predictors.shape[1]

# Convert the target to categorical: target
target = to_categorical(df.survived)
input_shape = (n_cols,)

# Import EarlyStopping
from keras.callbacks import EarlyStopping

# Save the number of columns in predictors: n_cols
#n_cols = predictors.shape[1]
#input_shape = (n_cols,)

# Specify the model
model_1 = Sequential()
model_1.add(Dense(100, activation='relu', input_shape = input_shape))
model_1.add(Dense(100, activation='relu'))
model_1.add(Dense(2, activation='softmax'))

# Compile the model
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])



# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Create the new model: model_2
model_2 = Sequential()

# Add the first and second layers
model_2.add(Dense(100, activation='relu', input_shape=input_shape))
model_2.add(Dense(100, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit model_1
model_1_training = model_1.fit(predictors, target, epochs=15, validation_split=0.2, callbacks
=[early_stopping_monitor], verbose=False)

# Fit model_2
model_2_training = model_2.fit(predictors, target, epochs=15, validation_split=0.2, callbacks
=[early_stopping_monitor], verbose=False)

# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b'
```

```
)
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```



In [8]:
```python
# Import necessary modules
#import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
import pandas as pd
#import numpy as np
#from keras.optimizers import SGD
import matplotlib.pyplot as plt

#pred_data = pd.read_csv('pred_data.csv').as_matrix()
pred_data = pd.read_csv('pred_data.csv').values

df = pd.read_csv('titanic_all_numeric.csv')
#predictors = df.drop(['survived'], axis=1).as_matrix()
predictors = df.drop(['survived'], axis=1).values

n_cols = predictors.shape[1]

# Convert the target to categorical: target
target = to_categorical(df.survived)
input_shape = (n_cols,)

# Import EarlyStopping
from keras.callbacks import EarlyStopping
# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Save the number of columns in predictors: n_cols
#n_cols = predictors.shape[1]
#input_shape = (n_cols,)

# Specify the model
model_1 = Sequential()
model_1.add(Dense(100, activation='relu', input_shape = input_shape))
model_1.add(Dense(100, activation='relu'))
model_1.add(Dense(2, activation='softmax'))

# Compile the model
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# The input shape to use in the first hidden layer
input_shape = (n_cols,)

# Create the new model: model_2
model_2 = Sequential()

# Add the first, second, and third hidden layers
model_2.add(Dense(50, activation='relu', input_shape=input_shape))
model_2.add(Dense(50, activation='relu'))
model_2.add(Dense(50, activation='relu'))

# Add the output layer
model_2.add(Dense(2, activation='softmax'))

# Compile model_2
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit model 1
model_1_training = model_1.fit(predictors, target, epochs=20, validation_split=0.4, callbacks
=[early_stopping_monitor], verbose=False)

# Fit model 2
model_2_training = model_2.fit(predictors, target, epochs=20, validation_split=0.4, callbacks
=[early_stopping_monitor], verbose=False)
```

```
# Create the plot
plt.plot(model_1_training.history['val_loss'], 'r', model_2_training.history['val_loss'], 'b'
)
plt.xlabel('Epochs')
plt.ylabel('Validation score')
plt.show()
```



In [5]:

```
# Import necessary modules
#import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
import pandas as pd
#import numpy as np
#from keras.optimizers import SGD
#import matplotlib.pyplot as plt

#pred_data = pd.read_csv('pred_data.csv').as_matrix()
#predictors = df.drop(df.iloc[:,0], axis=1).as_matrix()

df = pd.read_csv('mnist.csv')
print(df.shape)

X = df.iloc[:,1:785]
n_cols = X.shape[1]

# Convert the target to categorical: target
y = to_categorical(df.iloc[:,0])
input_shape = (n_cols,)

# Import EarlyStopping
from keras.callbacks import EarlyStopping
# Define early_stopping_monitor
early_stopping_monitor = EarlyStopping(patience=2)

# Save the number of columns in predictors: n_cols
#n_cols = predictors.shape[1]
#input_shape = (n_cols,)

# Create the model: model
model = Sequential()

# Add the first hidden layer
model.add(Dense(50, activation='relu', input_shape=(784,)))

# Add the second hidden layer
model.add(Dense(50, activation='relu'))

# Add the output layer
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit the model
model.fit(X, y, validation_split=0.3, epochs=60, callbacks=[early_stopping_monitor])
```

```
(2000, 785)
Train on 1400 samples, validate on 600 samples
Epoch 1/60
1400/1400 [==============================] - 1s 484us/step - loss: 14.0097 - acc: 0.1286 - va
l_loss: 14.0524 - val_acc: 0.1250
Epoch 2/60
1400/1400 [==============================] - 0s 82us/step - loss: 13.4422 - acc: 0.1650 - val
_loss: 13.8749 - val_acc: 0.1383
Epoch 3/60
1400/1400 [==============================] - 0s 83us/step - loss: 13.4031 - acc: 0.1671 - val
_loss: 13.6745 - val_acc: 0.1517
Epoch 4/60
1400/1400 [==============================] - 0s 83us/step - loss: 13.2744 - acc: 0.1764 - val
_loss: 13.5604 - val_acc: 0.1567
Epoch 5/60
1400/1400 [==============================] - 0s 82us/step - loss: 13.2304 - acc: 0.1786 - val
_loss: 13.5650 - val_acc: 0.1583
Epoch 6/60
1400/1400 [==============================] - 0s 83us/step - loss: 13.1594 - acc: 0.1814 - val
_loss: 13.4537 - val_acc: 0.1650
Epoch 7/60
1400/1400 [==============================] - 0s 89us/step - loss: 13.3231 - acc: 0.1729 - val
_loss: 13.5095 - val_acc: 0.1617
Epoch 8/60
1400/1400 [==============================] - 0s 86us/step - loss: 13.1708 - acc: 0.1829 - val
_loss: 13.3583 - val_acc: 0.1700
Epoch 9/60
1400/1400 [==============================] - 0s 82us/step - loss: 13.1566 - acc: 0.1829 - val
_loss: 13.4227 - val_acc: 0.1667
Epoch 10/60
1400/1400 [==============================] - 0s 85us/step - loss: 13.1587 - acc: 0.1829 - val
_loss: 14.0634 - val_acc: 0.1250
```

Out[5]:
```
<keras.callbacks.History at 0x1da5132c1d0>
```

In [ ]:
```
```