In [1]:
```
import random
from keras.models import Sequential
from keras.layers import Dense, LSTM, TimeDistributed, RepeatVector
from keras.callbacks import EarlyStopping
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
import matplotlib.pyplot as plt
epoches = 90

def generate_pairs():
    pairs = []
    for i in range(100):
        for j in range(100):
            if random.random() < 0.5:
                operation = '+'
                if(i+j>=0):
                    answer = "+"+str(i+j)
                else:
                    answer = str(i+j)
            else:
                operation = '-'
                if(i-j>=0):
                    answer = "+"+str(i-j)
                else:
                    answer = str(i-j)
            query = str(i) + operation + str(j)
            query = query.ljust(5)
            answer = answer.ljust(4, ' ')
            answer = answer.rjust(3, '+')
            pairs.append((query, answer))
    return pairs


def generate_reverse_pairs():
    pairs = []
    for i in range(100):
        for j in range(100):
            if random.random() < 0.5:
                operation = '+'
                if(i+j>=0):
                    answer = "+"+str(i+j)
                else:
                    answer = str(i+j)
            else:
                operation = '-'
                if(i-j>=0):
                    answer = "+"+str(i-j)
                else:
                    answer = str(i-j)
            query = str(i) + operation + str(j)
            query = query.ljust(5)
            answer = answer.ljust(4, ' ')
            answer = answer.rjust(3, '+')
            query = query[::-1]
            answer = answer[::-1]
            pairs.append((query, answer))
```

```
    return pairs

def encode_string(string):
    alphabet = ['0','1','2','3','4','5','6','7','8','9','+','-',' ']
    encoding = []
    for char in string:
        vector = [0]*len(alphabet)
        vector[alphabet.index(char)] = 1
        encoding.append(vector)
    return encoding
```

In [2]:
```
query = '5-20 '
encoded_query = encode_string(query)
print(encoded_query)

answer = '-15 '
encoded_answer = encode_string(answer)
print(encoded_answer)
```

```
[[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0], [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1]]
```

In [3]:
```
pairs = generate_pairs()
```

In [4]:
```
pairs
```

```
('0-21 ', '-21 '),
('0-22 ', '-22 '),
('0+23 ', '+23 '),
('0+24 ', '+24 '),
('0+25 ', '+25 '),
('0-26 ', '-26 '),
('0+27 ', '+27 '),
('0+28 ', '+28 '),
('0-29 ', '-29 '),
('0+30 ', '+30 '),
('0+31 ', '+31 '),
('0+32 ', '+32 '),
('0-33 ', '-33 '),
('0-34 ', '-34 '),
('0-35 ', '-35 '),
('0-36 ', '-36 '),
('0-37 ', '-37 '),
('0+38 ', '+38 '),
('0-39 ', '-39 '),
('0+40 ', '+40 '),
```

In [5]:
```
encoded_pairs = [(encode_string(pair[0]), encode_string(pair[1])) for pair in
```

In [6]:
```
encoded_pairs[0]
```

Out[6]:
```
([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]],
 [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

In [7]:
```
X = np.array([q[0] for q in encoded_pairs])
y = np.array([q[1] for q in encoded_pairs])
```

In [8]:
```
X[0]
```

Out[8]:
```
array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

In [9]:
```
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.1
```

In [10]:
```
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, tes
```

In [11]:
```
model = Sequential()
model.add(LSTM(128, input_shape=(5, 13), return_sequences=False))
model.add(RepeatVector(4))
model.add(LSTM(128, return_sequences=True))
model.add(Dense(13, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
```

Above network implements an Encoder-Decoder LSTM network by stacking two LSTM layers. The first LSTM layer acts as the encoder, which takes the input sequence of length 5 and converts it into a fixed-length vector representation. The second LSTM layer acts as the decoder, which takes the fixed-length vector representation generated by the encoder and produces the output sequence of length 4. The RepeatVector layer repeats the fixed-length vector representation 4 times so that it can be used as input to the second LSTM layer. Finally, a Dense layer with softmax activation is used to produce the output sequence.

In [12]:
```
history = model.fit(X_train, y_train, batch_size=64, epochs=epoches, validati
```

```
accuracy: 0.6248 - val_loss: 1.0155 - val_accuracy: 0.6365
Epoch 9/90
113/113 [==============================] - 3s 27ms/step - loss: 1.0157 -
accuracy: 0.6340 - val_loss: 1.0262 - val_accuracy: 0.6137
Epoch 10/90
113/113 [==============================] - 3s 28ms/step - loss: 0.9809 -
accuracy: 0.6489 - val_loss: 0.9754 - val_accuracy: 0.6443
Epoch 11/90
113/113 [==============================] - 3s 28ms/step - loss: 0.9787 -
accuracy: 0.6433 - val_loss: 0.9630 - val_accuracy: 0.6545
Epoch 12/90
113/113 [==============================] - 3s 27ms/step - loss: 0.9405 -
accuracy: 0.6649 - val_loss: 0.9392 - val_accuracy: 0.6665
Epoch 13/90
113/113 [==============================] - 3s 28ms/step - loss: 0.9172 -
accuracy: 0.6744 - val_loss: 0.9111 - val_accuracy: 0.6759
Epoch 14/90
113/113 [==============================] - 3s 27ms/step - loss: 0.9083 -
accuracy: 0.6736 - val_loss: 0.9022 - val_accuracy: 0.6769
Epoch 15/90
```

In [13]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)
print('Test accuracy:', test_acc)
```

```
47/47 [==============================] - 1s 10ms/step - loss: 0.0984 - accur
acy: 0.9692
Test accuracy: 0.9691666960716248
```

In [14]:
```
# # plt.plot(history.history['accuracy'])
# plt.plot(history.history['val_accuracy'])
# plt.title('Model accuracy')
# plt.ylabel('Accuracy')
# plt.xlabel('Epoch')
# plt.legend(['Test'], loc='upper left')
# plt.show()
```

In [15]:
```
reverse_pairs = generate_reverse_pairs()
```

In [16]:
```
reverse_pairs
```

Out[16]:
```
[('  0-0', '  0+'),
 ('  1-0', '  1-'),
 ('  2+0', '  2+'),
 ('  3+0', '  3+'),
 ('  4+0', '  4+'),
 ('  5+0', '  5+'),
 ('  6-0', '  6-'),
 ('  7-0', '  7-'),
 ('  8+0', '  8+'),
 ('  9+0', '  9+'),
 (' 01-0', ' 01-'),
 (' 11+0', ' 11+'),
 (' 21+0', ' 21+'),
 (' 31+0', ' 31+'),
 (' 41-0', ' 41-'),
 (' 51+0', ' 51+'),
 (' 61+0', ' 61+'),
 (' 71-0', ' 71-'),
 (' 81-0', ' 81-'),
```

In [17]:
```
encoded_reverse_pairs = [(encode_string(pair[0]), encode_string(pair[1])) for
```

In [18]:
```
X_rev = np.array([q[0] for q in encoded_reverse_pairs])
y_rev = np.array([q[1] for q in encoded_reverse_pairs])
```

In [19]:
```
X_trainval, X_test, y_trainval, y_test = train_test_split(X_rev, y_rev, test_
```

In [20]:
```
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, tes
```

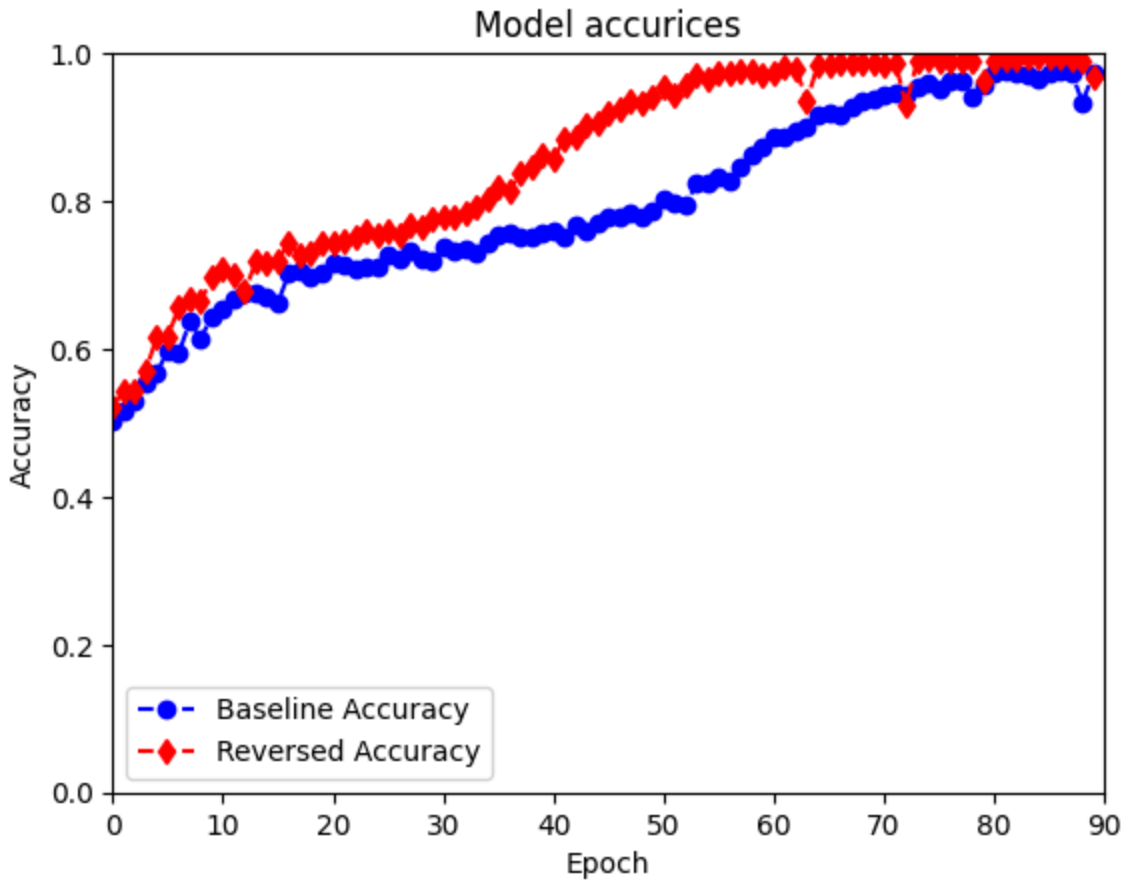
In [21]:
```
model = Sequential()
model.add(LSTM(128, input_shape=(5, 13), return_sequences=False))
model.add(RepeatVector(4))
model.add(LSTM(128, return_sequences=True))
model.add(Dense(13, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

history_rev = model.fit(X_train, y_train, batch_size=64, epochs=epoches, vali
```

```
Epoch 1/90
113/113 [==============================] - 13s 46ms/step - loss: 1.8452 -
accuracy: 0.4017 - val_loss: 1.3534 - val_accuracy: 0.5214
Epoch 2/90
113/113 [==============================] - 3s 29ms/step - loss: 1.3068 -
accuracy: 0.5319 - val_loss: 1.2682 - val_accuracy: 0.5441
Epoch 3/90
113/113 [==============================] - 3s 29ms/step - loss: 1.2573 -
accuracy: 0.5447 - val_loss: 1.2440 - val_accuracy: 0.5429
Epoch 4/90
113/113 [==============================] - 3s 29ms/step - loss: 1.2053 -
accuracy: 0.5590 - val_loss: 1.1667 - val_accuracy: 0.5694
Epoch 5/90
113/113 [==============================] - 3s 28ms/step - loss: 1.1252 -
accuracy: 0.5909 - val_loss: 1.0791 - val_accuracy: 0.6159
Epoch 6/90
113/113 [==============================] - 3s 29ms/step - loss: 1.0605 -
accuracy: 0.6243 - val_loss: 1.0416 - val_accuracy: 0.6163
Epoch 7/90
```

In [22]:
```
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=1)
print('Test accuracy:', test_acc)
```

```
47/47 [==============================] - 1s 10ms/step - loss: 0.1064 - accur
acy: 0.9637
Test accuracy: 0.9636666774749756
```

In [23]:
```
plt.plot(history.history['val_accuracy'],'--bo')
plt.plot(history_rev.history['val_accuracy'],'--rd')
plt.xlim([0, epoches])
plt.ylim([0, 1])
plt.title('Model accurices')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Baseline Accuracy', 'Reversed Accuracy'], loc='lower left')
plt.show()
```



In [ ]:
```
```