【要点まとめ】
RNNとは、時系列データに対応可能なニューラルネットワークである。
例えば以下のようなものがある。

- 音声データ
- テキストデータ
- 株価のデータ

これまでのニューラルネットワークの考え方に、時系列の重みのパラメータが追加される。
時系列モデルを扱うには、初期の状態と過去の状態(t-1)を保持し、そこから次の状態(t)を求める再帰的な構造が必要となる。

BPTTとは、RNNにおけるパラメータ調整方法の一つ。 RNNは時系列の重みが追加されたNN、つまり逆伝播も適用可能という考えのもと考案された。

【実装演習】
※ 参考資料：3_1_simple_RNN_after.ipynb
BPTTの時刻tを使用した逆伝播の処理を写経

```
# 時系列ループ
for t in range(binary_dim):
    # 入力値
    X = np.array([a_bin[ - t - 1], b_bin[ - t - 1]]).reshape(1, -1)
    # 時刻tにおける正解データ
    dd = np.array([d_bin[binary_dim - t - 1]])

    u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t].reshape(1, -1), W)
    z[:,t+1] = functions.sigmoid(u[:,t+1])
    y[:,t] = functions.sigmoid(np.dot(z[:,t+1].reshape(1, -1), W_out))

    #誤差
    loss = functions.mean_squared_error(dd, y[:,t])
    delta_out[:,t] = functions.d_mean_squared_error(dd, y[:,t]) *
functions.d_sigmoid(y[:,t])
    all_loss += loss
    out_bin[binary_dim - t - 1] = np.round(y[:,t])

for t in range(binary_dim)[::-1]:
    X = np.array([a_bin[-t-1],b_bin[-t-1]]).reshape(1, -1)

    delta[:,t] = (np.dot(delta[:,t+1].T, W.T) + np.dot(delta_out[:,t].T,
W_out.T)) * functions.d_sigmoid(u[:,t+1])

    # 勾配を更新
    W_out_grad += np.dot(z[:,t+1].reshape(-1,1),
delta_out[:,t].reshape(-1,1))
    W_grad += np.dot(z[:,t].reshape(-1,1), delta[:,t].reshape(1,-1))
    W_in_grad += np.dot(X.T, delta[:,t].reshape(1,-1))
```

【実装演習考察】
BPTTの概念をふんわり理解することができました。
説明では前の時間は(t-1)でしたが、プログラムは変数の初期値が0ということもあり、前の時間を(t)、今の時間を(t+1)と置き換えて考えるなど 実際に手で実装しなければ気づきづらいこともありました。

【自己学習】
※参考資料：ゼロから作るDeepLearning②
BPTTによって、RNNの学習を容易に行えそうに見えるが、解決しなければならない問題もある。
長い時系列データを学習する場合、それらは全てメモリに保持しておく必要があるため、時系列データが長くなるに従って膨大なメモリが必要になってくる。