①再帰型ニューラルネットワークの概念

=====

目次:

1.simple RNN(例:バイナリ加算)

【要約】

・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。

・時間的なつながりをどのように組み込むか？　→前回の中間層の出力をループさせる(次の中間層の入力とする)ことを実施。

・RNNでは重みが3か所に存在。:1.入力層から中間層の重み、2.中間層から出力層の重み、3.前の中間層からの重み(中間層から中間層の重み)

・BPTT(Backpropagation Through Time)とは、RNNにおける誤差逆伝播の一種。

　→更新すべきパラメータ　重み(W_in、W_out、W):3つ、バイアス(b、c):2つ

・バイナリ加算を例にとり、RNNを実装。

In [1]:

```
import sys
sys.path.append('C:/Users/NIF/Desktop/4.深層学習_後編/DNN_code_colab_lesson_3_4/DNN_code_colab_lesson_3_4')
```

=====

1.simple RNN(例:バイナリ加算)

In [2]:

```
import numpy as np
from common import functions
import matplotlib.pyplot as plt


def d_tanh(x):
    return 1/(np.cosh(x) ** 2)

# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)
# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)],dtype=np.uint8).T,axis=1)

input_layer_size = 2
hidden_layer_size = 16
output_layer_size = 1

weight_init_std = 1
learning_rate = 0.1

iters_num = 10000
plot_interval = 100

# ウェイト初期化 (バイアスは簡単のため省略)
W_in = weight_init_std * np.random.randn(input_layer_size, hidden_layer_size)
W_out = weight_init_std * np.random.randn(hidden_layer_size, output_layer_size)
W = weight_init_std * np.random.randn(hidden_layer_size, hidden_layer_size)
# Xavier
# W_in = np.random.randn(input_layer_size, hidden_layer_size) / (np.sqrt(input_layer_size))
# W_out = np.random.randn(hidden_layer_size, output_layer_size) / (np.sqrt(hidden_layer_size))
# W = np.random.randn(hidden_layer_size, hidden_layer_size) / (np.sqrt(hidden_layer_size))
# He
# W_in = np.random.randn(input_layer_size, hidden_layer_size) / (np.sqrt(input_layer_size)) * n
p.sqrt(2)
# W_out = np.random.randn(hidden_layer_size, output_layer_size) / (np.sqrt(hidden_layer_size)) *
np.sqrt(2)
# W = np.random.randn(hidden_layer_size, hidden_layer_size) / (np.sqrt(hidden_layer_size)) * np.
sqrt(2)


# 勾配
W_in_grad = np.zeros_like(W_in)
W_out_grad = np.zeros_like(W_out)
W_grad = np.zeros_like(W)

u = np.zeros((hidden_layer_size, binary_dim + 1))
z = np.zeros((hidden_layer_size, binary_dim + 1))
y = np.zeros((output_layer_size, binary_dim))

delta_out = np.zeros((output_layer_size, binary_dim))
delta = np.zeros((hidden_layer_size, binary_dim + 1))

all_losses = []

for i in range(iters_num):
```

```
    # A, B初期化 (a + b = d)
    a_int = np.random.randint(largest_number/2)
    a_bin = binary[a_int] # binary encoding
    b_int = np.random.randint(largest_number/2)
    b_bin = binary[b_int] # binary encoding

    # 正解データ
    d_int = a_int + b_int
    d_bin = binary[d_int]

    # 出力バイナリ
    out_bin = np.zeros_like(d_bin)

    # 時系列全体の誤差
    all_loss = 0

    # 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])
#         z[:,t+1] = functions.relu(u[:,t+1])
#         z[:,t+1] = np.tanh(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)) * functio
ns.d_sigmoid(u[:,t+1])
#         delta[:,t] = (np.dot(delta[:,t+1].T, W.T) + np.dot(delta_out[:,t].T, W_out.T)) * funct
ions.d_relu(u[:,t+1])
#         delta[:,t] = (np.dot(delta[:,t+1].T, W.T) + np.dot(delta_out[:,t].T, W_out.T)) * d_tan
h(u[:,t+1])

        # 勾配更新
        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))

    # 勾配適用
    W_in -= learning_rate * W_in_grad
    W_out -= learning_rate * W_out_grad
    W -= learning_rate * W_grad
```

```
    W_in_grad *= 0
    W_out_grad *= 0
    W_grad *= 0


    if(i % plot_interval == 0):
        all_losses.append(all_loss)
        print("iters:" + str(i))
        print("Loss:" + str(all_loss))
        print("Pred:" + str(out_bin))
        print("True:" + str(d_bin))
        out_int = 0
        for index,x in enumerate(reversed(out_bin)):
            out_int += x * pow(2, index)
        print(str(a_int) + " + " + str(b_int) + " = " + str(out_int))
        print("------------")

lists = range(0, iters_num, plot_interval)
plt.plot(lists, all_losses, label="loss")
plt.show()
```

```
iters:0
Loss:1.036773441502257
Pred:[0 0 0 0 0 0 0 1]
True:[1 0 1 1 0 1 0 0]
93 + 87 = 1
------------
iters:100
Loss:1.2340115851746196
Pred:[0 0 0 0 0 0 0 0]
True:[0 1 1 1 1 1 0 0]
67 + 57 = 0
------------
iters:200
Loss:1.0028469227758388
Pred:[1 1 1 1 1 1 1 1]
True:[0 1 1 1 1 1 1 0]
92 + 34 = 255
------------
iters:300
Loss:1.0029024432390345
Pred:[0 0 0 0 0 0 1 1]
True:[1 0 1 0 1 1 1 1]
118 + 57 = 3
------------
iters:400
Loss:1.0820306421812869
Pred:[0 0 0 0 0 0 0 1]
True:[1 0 1 0 1 0 1 0]
98 + 72 = 1
------------
iters:500
Loss:1.0129344346593747
Pred:[0 1 1 1 1 1 0 0]
True:[1 0 0 0 1 0 1 0]
11 + 127 = 124
------------
iters:600
Loss:0.9310054398009453
Pred:[1 1 1 1 1 1 0 0]
True:[1 0 1 1 0 1 0 0]
92 + 88 = 252
------------
iters:700
Loss:0.9315681075593076
Pred:[1 1 1 1 1 1 0 0]
True:[1 0 1 1 1 0 1 0]
115 + 71 = 252
------------
iters:800
Loss:0.9878116447416826
Pred:[0 1 0 0 0 0 1 0]
True:[0 1 1 1 1 1 1 0]
74 + 52 = 66
------------
iters:900
Loss:1.088785906782315
Pred:[0 0 0 1 1 1 1 0]
True:[0 0 1 0 0 0 0 0]
31 + 1 = 30
------------
iters:1000
```

```
Loss:0.6812616917496612
Pred:[1 1 1 1 1 1 1 1]
True:[0 1 1 1 1 1 1 1]
23 + 104 = 255
------------
iters:1100
Loss:1.0650310500479856
Pred:[1 1 1 1 0 1 0 0]
True:[0 1 1 1 1 0 1 0]
32 + 90 = 244
------------
iters:1200
Loss:0.8427192215496554
Pred:[1 1 0 1 0 1 0 1]
True:[1 1 0 1 0 1 0 1]
107 + 106 = 213
------------
iters:1300
Loss:0.8671772492785703
Pred:[0 1 1 1 1 0 1 1]
True:[0 1 1 1 1 1 0 1]
97 + 28 = 123
------------
iters:1400
Loss:0.911356445578591
Pred:[0 1 0 0 0 1 1 0]
True:[0 1 1 0 1 0 0 0]
53 + 51 = 70
------------
iters:1500
Loss:0.4668269924282596
Pred:[0 0 0 0 1 0 0 0]
True:[0 0 0 0 1 1 0 0]
8 + 4 = 8
------------
iters:1600
Loss:1.1645183122170928
Pred:[0 1 1 1 1 1 1 1]
True:[1 0 0 0 0 1 0 1]
40 + 93 = 127
------------
iters:1700
Loss:0.9755167117884598
Pred:[1 1 1 0 0 0 1 0]
True:[1 0 0 1 1 0 1 0]
62 + 92 = 226
------------
iters:1800
Loss:1.1137840433353208
Pred:[1 1 0 1 1 1 1 0]
True:[1 0 1 0 0 0 0 0]
125 + 35 = 222
------------
iters:1900
Loss:1.124817804113314
Pred:[1 1 1 1 1 0 0 0]
True:[1 0 0 0 0 0 1 0]
47 + 83 = 248
------------
iters:2000
Loss:1.1265556576895686
```

```
Pred:[1 0 0 0 0 0 1 1]
True:[1 1 1 1 1 0 0 1]
124 + 125 = 131
------------
iters:2100
Loss:0.9998947260711665
Pred:[0 1 0 0 0 0 0 0]
True:[0 0 1 1 1 1 1 1]
28 + 35 = 64
------------
iters:2200
Loss:0.7694857251910335
Pred:[0 1 1 0 0 0 0 1]
True:[0 1 1 0 0 1 1 1]
89 + 14 = 97
------------
iters:2300
Loss:1.1424901717023468
Pred:[1 1 0 1 1 1 0 1]
True:[1 0 1 0 0 0 0 1]
107 + 54 = 221
------------
iters:2400
Loss:0.7276539927117841
Pred:[0 0 1 1 0 0 0 1]
True:[0 0 1 1 1 0 1 1]
12 + 47 = 49
------------
iters:2500
Loss:0.5833455603438876
Pred:[0 1 0 0 1 0 0 1]
True:[0 1 1 0 1 0 0 1]
71 + 34 = 73
------------
iters:2600
Loss:1.0349268851070446
Pred:[1 0 1 1 0 1 0 1]
True:[1 1 0 0 1 0 0 1]
74 + 127 = 181
------------
iters:2700
Loss:0.7172679052139137
Pred:[0 1 0 1 0 0 0 1]
True:[0 1 0 1 0 1 1 1]
40 + 47 = 81
------------
iters:2800
Loss:0.8793220141600379
Pred:[0 1 0 1 0 1 0 0]
True:[0 1 1 0 1 0 0 0]
46 + 58 = 84
------------
iters:2900
Loss:0.18231879326759218
Pred:[0 0 1 0 1 0 0 0]
True:[0 0 1 0 1 0 0 0]
32 + 8 = 40
------------
iters:3000
Loss:0.7024702443719486
Pred:[0 0 1 0 1 0 0 0]
```

```
True:[0 0 1 1 0 1 0 0]
21 + 31 = 40
------------
iters:3100
Loss:0.9725699622961711
Pred:[1 0 0 0 0 1 0 0]
True:[1 0 1 0 0 0 1 1]
58 + 105 = 132
------------
iters:3200
Loss:0.5943259467865718
Pred:[1 1 0 1 1 1 1 1]
True:[1 0 0 1 1 1 0 1]
37 + 120 = 223
------------
iters:3300
Loss:0.6089669407707451
Pred:[0 1 1 0 0 1 0 0]
True:[0 1 1 0 1 0 0 0]
91 + 13 = 100
------------
iters:3400
Loss:0.7269199387425666
Pred:[1 0 0 0 0 0 0 0]
True:[1 1 0 1 0 0 0 0]
125 + 83 = 128
------------
iters:3500
Loss:0.5651311115938213
Pred:[1 0 0 0 1 1 0 0]
True:[1 0 0 0 1 0 0 0]
69 + 67 = 140
------------
iters:3600
Loss:0.5175327076423313
Pred:[0 1 0 0 0 0 0 1]
True:[0 1 1 0 0 0 0 1]
37 + 60 = 65
------------
iters:3700
Loss:0.5313022018196188
Pred:[1 0 0 0 0 0 1 0]
True:[1 1 0 1 0 0 1 0]
126 + 84 = 130
------------
iters:3800
Loss:0.11560151738612934
Pred:[0 0 1 0 1 1 1 0]
True:[0 0 1 0 1 1 1 0]
24 + 22 = 46
------------
iters:3900
Loss:0.11405801465520668
Pred:[0 1 0 1 0 1 0 0]
True:[0 1 0 1 0 1 0 0]
66 + 18 = 84
------------
iters:4000
Loss:0.13284744620334285
Pred:[0 1 1 0 1 0 1 0]
True:[0 1 1 0 1 0 1 0]
```

```
6 + 100 = 106
------------
iters:4100
Loss:0.32283091218957755
Pred:[1 0 0 0 1 1 0 1]
True:[1 0 1 0 1 1 0 1]
61 + 112 = 141
------------
iters:4200
Loss:0.5251724567380124
Pred:[1 1 0 1 0 0 0 1]
True:[1 1 0 1 0 0 0 1]
86 + 123 = 209
------------
iters:4300
Loss:0.339694078546566
Pred:[1 1 0 0 1 0 1 1]
True:[1 1 1 0 1 0 1 1]
122 + 113 = 203
------------
iters:4400
Loss:0.22588731766821274
Pred:[0 1 0 0 0 0 0 1]
True:[0 1 0 0 0 0 0 1]
20 + 45 = 65
------------
iters:4500
Loss:0.2860324634591891
Pred:[1 1 0 0 0 1 1 0]
True:[1 1 0 0 0 1 1 0]
104 + 94 = 198
------------
iters:4600
Loss:0.06506367054708921
Pred:[0 1 0 1 1 1 0 1]
True:[0 1 0 1 1 1 0 1]
57 + 36 = 93
------------
iters:4700
Loss:0.0892992937878568
Pred:[1 0 0 0 1 0 1 0]
True:[1 0 0 0 1 0 1 0]
49 + 89 = 138
------------
iters:4800
Loss:0.26119893901376073
Pred:[1 0 1 1 1 0 0 0]
True:[1 0 1 1 1 0 0 0]
93 + 91 = 184
------------
iters:4900
Loss:0.23766221380249086
Pred:[0 1 1 0 1 0 1 0]
True:[0 1 1 0 1 0 1 0]
28 + 78 = 106
------------
iters:5000
Loss:0.21040014696194553
Pred:[1 1 0 1 0 0 1 0]
True:[1 1 0 1 0 0 1 0]
107 + 103 = 210
```

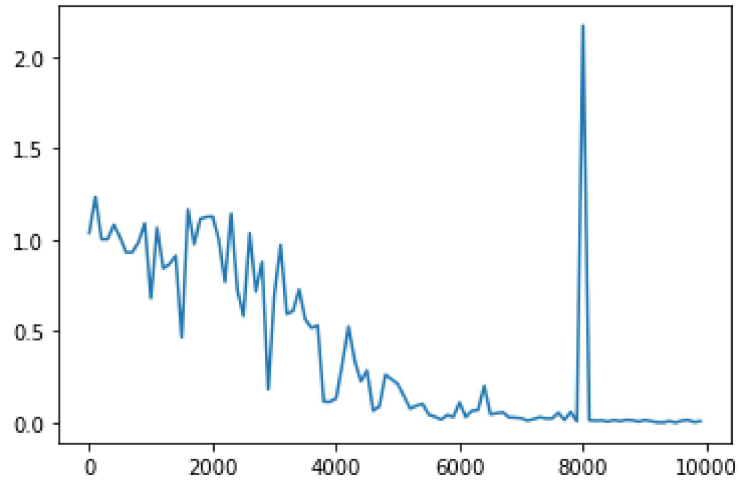
```
------------
iters:5100
Loss:0.144853939671018
Pred:[1 0 0 1 1 0 0 0]
True:[1 0 0 1 1 0 0 0]
46 + 106 = 152
------------
iters:5200
Loss:0.07709923843122105
Pred:[1 0 1 0 1 0 1 0]
True:[1 0 1 0 1 0 1 0]
68 + 102 = 170
------------
iters:5300
Loss:0.0923702861167269
Pred:[0 0 0 0 0 1 1 1]
True:[0 0 0 0 0 1 1 1]
0 + 7 = 7
------------
iters:5400
Loss:0.10268934111103979
Pred:[0 0 1 0 1 0 1 1]
True:[0 0 1 0 1 0 1 1]
3 + 40 = 43
------------
iters:5500
Loss:0.04348559961826527
Pred:[0 1 1 0 0 1 0 0]
True:[0 1 1 0 0 1 0 0]
82 + 18 = 100
------------
iters:5600
Loss:0.03199808536428666
Pred:[1 0 1 1 1 0 1 0]
True:[1 0 1 1 1 0 1 0]
104 + 82 = 186
------------
iters:5700
Loss:0.01658341093132214
Pred:[0 1 1 0 0 1 1 0]
True:[0 1 1 0 0 1 1 0]
17 + 85 = 102
------------
iters:5800
Loss:0.0411595291086146
Pred:[0 0 1 1 1 0 1 0]
True:[0 0 1 1 1 0 1 0]
52 + 6 = 58
------------
iters:5900
Loss:0.03080185307096661
Pred:[0 1 1 0 1 0 0 1]
True:[0 1 1 0 1 0 0 1]
21 + 84 = 105
------------
iters:6000
Loss:0.11052307805177695
Pred:[0 1 1 0 0 1 1 1]
True:[0 1 1 0 0 1 1 1]
39 + 64 = 103
------------
```

```
iters:6100
Loss:0.030689277094428292
Pred:[0 1 0 1 0 1 0 1]
True:[0 1 0 1 0 1 0 1]
32 + 53 = 85
------------
iters:6200
Loss:0.06473256910376082
Pred:[1 1 0 1 1 0 0 0]
True:[1 1 0 1 1 0 0 0]
122 + 94 = 216
------------
iters:6300
Loss:0.06930324400986768
Pred:[1 0 1 0 1 0 1 1]
True:[1 0 1 0 1 0 1 1]
77 + 94 = 171
------------
iters:6400
Loss:0.20151614740505502
Pred:[1 0 1 0 0 1 1 1]
True:[1 0 1 0 0 1 1 1]
55 + 112 = 167
------------
iters:6500
Loss:0.045598094332549924
Pred:[1 0 1 1 0 0 0 0]
True:[1 0 1 1 0 0 0 0]
114 + 62 = 176
------------
iters:6600
Loss:0.051496643454152634
Pred:[1 0 1 0 0 1 0 1]
True:[1 0 1 0 0 1 0 1]
38 + 127 = 165
------------
iters:6700
Loss:0.05616409946870304
Pred:[1 1 1 0 0 1 1 0]
True:[1 1 1 0 0 1 1 0]
106 + 124 = 230
------------
iters:6800
Loss:0.02909848654477517
Pred:[0 1 1 0 0 0 0 0]
True:[0 1 1 0 0 0 0 0]
86 + 10 = 96
------------
iters:6900
Loss:0.027233353604539136
Pred:[0 1 0 1 0 1 0 0]
True:[0 1 0 1 0 1 0 0]
31 + 53 = 84
------------
iters:7000
Loss:0.023727575083071585
Pred:[1 0 0 0 1 0 0 0]
True:[1 0 0 0 1 0 0 0]
32 + 104 = 136
------------
iters:7100
```

```
Loss:0.010941998981559556
Pred:[0 1 1 0 1 0 0 1]
True:[0 1 1 0 1 0 0 1]
17 + 88 = 105
------------
iters:7200
Loss:0.01916616980399557
Pred:[0 1 1 1 1 1 1 1]
True:[0 1 1 1 1 1 1 1]
114 + 13 = 127
------------
iters:7300
Loss:0.03002237943467423
Pred:[0 0 1 1 1 1 1 1]
True:[0 0 1 1 1 1 1 1]
0 + 63 = 63
------------
iters:7400
Loss:0.022424258995499918
Pred:[0 1 0 0 1 1 1 0]
True:[0 1 0 0 1 1 1 0]
54 + 24 = 78
------------
iters:7500
Loss:0.023136656182012945
Pred:[0 1 0 1 0 0 0 0]
True:[0 1 0 1 0 0 0 0]
56 + 24 = 80
------------
iters:7600
Loss:0.053714623026718256
Pred:[1 1 0 0 1 0 0 1]
True:[1 1 0 0 1 0 0 1]
77 + 124 = 201
------------
iters:7700
Loss:0.013969735399104056
Pred:[1 0 0 0 0 0 0 1]
True:[1 0 0 0 0 0 0 1]
92 + 37 = 129
------------
iters:7800
Loss:0.059026194054079134
Pred:[1 0 0 1 1 0 0 0]
True:[1 0 0 1 1 0 0 0]
78 + 74 = 152
------------
iters:7900
Loss:0.007315571549155983
Pred:[0 1 1 1 1 1 1 1]
True:[0 1 1 1 1 1 1 1]
86 + 41 = 127
------------
iters:8000
Loss:2.170564401718215
Pred:[1 0 1 0 0 0 1 1]
True:[0 1 0 1 1 1 1 1]
22 + 73 = 163
------------
iters:8100
Loss:0.012899375439051872
```

```
Pred:[0 1 1 0 0 1 0 1]
True:[0 1 1 0 0 1 0 1]
60 + 41 = 101
------------
iters:8200
Loss:0.010379703161255683
Pred:[0 1 0 0 0 1 0 0]
True:[0 1 0 0 0 1 0 0]
56 + 12 = 68
------------
iters:8300
Loss:0.011792328474052437
Pred:[1 0 1 0 0 1 1 0]
True:[1 0 1 0 0 1 1 0]
68 + 98 = 166
------------
iters:8400
Loss:0.005538598478914673
Pred:[0 1 1 1 1 0 1 0]
True:[0 1 1 1 1 0 1 0]
87 + 35 = 122
------------
iters:8500
Loss:0.013203922420390637
Pred:[0 1 1 0 0 1 1 0]
True:[0 1 1 0 0 1 1 0]
10 + 92 = 102
------------
iters:8600
Loss:0.008015702967344251
Pred:[0 0 1 0 0 1 0 1]
True:[0 0 1 0 0 1 0 1]
10 + 27 = 37
------------
iters:8700
Loss:0.014803527960042452
Pred:[1 1 1 1 0 0 1 1]
True:[1 1 1 1 0 0 1 1]
116 + 127 = 243
------------
iters:8800
Loss:0.012601878604686898
Pred:[0 1 1 0 1 0 1 1]
True:[0 1 1 0 1 0 1 1]
49 + 58 = 107
------------
iters:8900
Loss:0.005508672889729072
Pred:[0 1 1 0 1 1 1 1]
True:[0 1 1 0 1 1 1 1]
14 + 97 = 111
------------
iters:9000
Loss:0.013642328837614386
Pred:[1 0 1 0 1 1 0 0]
True:[1 0 1 0 1 1 0 0]
86 + 86 = 172
------------
iters:9100
Loss:0.008566502982367746
Pred:[1 0 0 1 1 0 0 0]
```

```
True:[1 0 0 1 1 0 0 0]
27 + 125 = 152
------------
iters:9200
Loss:0.0014295119589494139
Pred:[1 0 0 0 0 1 1 0]
True:[1 0 0 0 0 1 1 0]
25 + 109 = 134
------------
iters:9300
Loss:0.0006312526580920925
Pred:[1 0 0 1 1 0 1 0]
True:[1 0 0 1 1 0 1 0]
121 + 33 = 154
------------
iters:9400
Loss:0.008265501756635173
Pred:[1 0 0 1 0 1 1 1]
True:[1 0 0 1 0 1 1 1]
77 + 74 = 151
------------
iters:9500
Loss:0.0002864053627482471
Pred:[0 1 0 1 1 1 1 0]
True:[0 1 0 1 1 1 1 0]
85 + 9 = 94
------------
iters:9600
Loss:0.01037018484221477
Pred:[0 1 0 1 0 0 1 1]
True:[0 1 0 1 0 0 1 1]
37 + 46 = 83
------------
iters:9700
Loss:0.014095154278984683
Pred:[1 0 1 1 0 1 1 1]
True:[1 0 1 1 0 1 1 1]
124 + 59 = 183
------------
iters:9800
Loss:0.002445716022717299
Pred:[1 0 1 1 1 0 0 0]
True:[1 0 1 1 1 0 0 0]
65 + 119 = 184
------------
iters:9900
Loss:0.007546172432820459
Pred:[1 1 0 0 1 0 1 1]
True:[1 1 0 0 1 0 1 1]
116 + 87 = 203
------------
```



→バイナリ加算:2進数で足し算を行うこと。桁の繰り上がり処理は、過去から未来へ情報をつなげていくことに類似。

→3つの重み:W_in(入力層から中間層の重み)、W_out(中間層から出力層の重み)、W(中間層から中間層の重み)

→BPTT(RNNの誤差逆伝播)により更新。

→学習により、誤差0%に漸近。

In [ ]: