実装演習 4-1. RNN

入力 [1]:
```
import numpy as np
from common import functions
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

データ用意

2進数の予測値を実データに近づけていく

入力 [2]:
```
# 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  # 入力層の数
hidden_layer_size = 16  # 隠れ層の数
output_layer_size = 1  # 出力層の数

weight_init_std = 1  # 初期ウェイト
learning_rate = 0.1  # 学習率

iters_num = 10000  # 繰り返し回数
plot_interval = 100  # グラフへのプロット間隔
```

入力 [3]:
```
# 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)
```

入力 [4]:
```
# 勾配
W_in_grad = np.zeros_like(W_in)
W_out_grad = np.zeros_like(W_out)
W_grad = np.zeros_like(W)
```

入力 [5]:
```
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 = []
```

学習

入力 [5]:

```
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])

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

        # 誤差関数はMSEを使用する
        loss = functions.mean_squared_error(dd, y[:,t])

        delta_out[:, t] = functions.d_mean_squared_error(dd, y[:, t]) * functions.d_sigmoid

        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)) * fur

        # 勾配の更新
        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))

    # 更新された勾配を適用する
    W_in -= learning_rate * W_in_grad
    W_out -= learning_rate * W_out_grad
    W -= learning_rate * W_grad

    # 更新用の勾配を初期化
    W_in_grad *= 0
    W_out_grad *= 0
    W_grad *= 0

    # 途中経過の出力
    if i % plot_interval == 0:
        all_losses.append(all_loss)

        print(f"iters: {i}")
        print(f"Loss: {all_loss}")
        print(f"Pred: {out_bin}")
        print(f"True: {d_bin}")

        out_int = 0
```

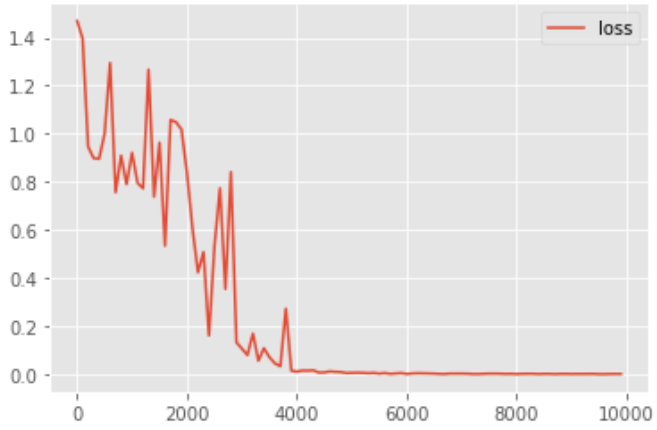
```
69
70          for index, x in enumerate(reversed(out_bin)):
71              out_int += x * pow(2, index)
72
73          print(f"{a_int} + {b_int} = {out_int}")
74          print("-"*20)
```

```
iters:0
Loss:1.469399340903874
Pred:[0 0 0 0 0 0 0 0]
True:[1 0 1 0 0 1 0 0]
125 + 39 = 0
------------
iters:100
Loss:1.3968157974462927
Pred:[0 0 0 0 0 0 0 0]
True:[0 1 0 1 1 1 0 1]
73 + 20 = 0
------------
iters:200
Loss:0.9470550270404142
Pred:[0 0 1 0 1 1 1 0]
True:[1 0 0 0 0 0 1 0]
20 + 110 = 46
------------
iters:300
```

グラフの描画

入力 [9]:

```
1  lists = range(0, iters_num, plot_interval)
2  plt.plot(lists, all_losses, label="loss")
3  plt.legend()
4  plt.show()
```



考察

iters:1600あたりで予測値が正解と一致するようになり、iters:2100あたりから正解値とよく一致するようになった。
iters:3000あたりからは、かなりの精度で正解と一致するようになった。
lossのグラフを見ても、iters:3000あたりからは0.1未満になるようになった。

入力 [ ]:

```
1
```