③GRU

=====

目次:

1.GRUの実装

【要約】

・従来のLSTMでは、パラメータが多数存在していたため、計算負荷が大きかった。

　→しかし、GRUでは、そのパラメータを大幅に削減し、精度は同等またはそれ以上が望める様になった構造。

・リセットゲート、更新ゲートを持つ。

　→リセットゲート:過去の情報をどれだけ忘れるかを決定する。

　→更新ゲート:過去の情報のどれだけを将来に渡す必要があるかを判断する。

=====

1.GRUの実装

→実装時のコードを確認

①順伝播

In [1]:

```
from common.np import *  # import numpy as np (or import cupy as np)
from common.layers import *
from common.functions import softmax, sigmoid

class GRU:
    def __init__(self, Wx, Wh, b):

        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]  ###
        self.cache = None

    def forward(self, x, h_prev):
        Wx, Wh, b = self.params
        H = Wh.shape[0]
        Wxz, Wxr, Wxh = Wx[:, :H], Wx[:, H:2 * H], Wx[:, 2 * H:]
        Whz, Whr, Whh = Wh[:, :H], Wh[:, H:2 * H], Wh[:, 2 * H:]
        bz, br, bh = b[:H], b[H:2 * H], b[2 * H:]

        z = sigmoid(np.dot(x, Wxz) + np.dot(h_prev, Whz) + bz)
        r = sigmoid(np.dot(x, Wxr) + np.dot(h_prev, Whr) + br)
        h_hat = np.tanh(np.dot(x, Wxh) + np.dot(r*h_prev, Whh) + bh)
        h_next = (1-z) * h_prev + z * h_hat

        self.cache = (x, h_prev, z, r, h_hat)

        return h_next
```

②逆伝播

In [2]:

```
def backward(self, dh_next):
        Wx, Wh, b = self.params
        H = Wh.shape[0]
        Wxz, Wxr, Wxh = Wx[:, :H], Wx[:, H:2 * H], Wx[:, 2 * H:]
        Whz, Whr, Whh = Wh[:, :H], Wh[:, H:2 * H], Wh[:, 2 * H:]
        x, h_prev, z, r, h_hat = self.cache

        dh_hat =dh_next * z
        dh_prev = dh_next * (1-z)

        # tanh
        dt = dh_hat * (1 - h_hat ** 2)
        dbh = np.sum(dt, axis=0)
        dWhh = np.dot((r * h_prev).T, dt)
        dhr = np.dot(dt, Whh.T)
        dWxh = np.dot(x.T, dt)
        dx = np.dot(dt, Wxh.T)
        dh_prev += r * dhr

        # update gate(z)
        dz = dh_next * h_hat - dh_next * h_prev
        dt = dz * z * (1-z)
        dbz = np.sum(dt, axis=0)
        dWhz = np.dot(h_prev.T, dt)
        dh_prev += np.dot(dt, Whz.T)
        dWxz = np.dot(x.T, dt)
        dx += np.dot(dt, Wxz.T)

        # rest gate(r)
        dr = dhr * h_prev
        dt = dr * r * (1-r)
        dbr = np.sum(dt, axis=0)
        dWhr = np.dot(h_prev.T, dt)
        dh_prev += np.dot(dt, Whr.T)
        dWxr = np.dot(x.T, dt)
        dx += np.dot(dt, Wxr.T)

        self.dWx = np.hstack((dWxz, dWxr, dWxh))
        self.dWh = np.hstack((dWhz, dWhr, dWhh))
        self.db = np.hstack((dbz, dbr, dbh))

        self.grads[0][...] = self.dWx
        self.grads[1][...] = self.dWh
        self.grads[2][...] = self.db

        return dx, dh_prev
```

In [ ]:

```
```