```
import torch
from torch import nn

def sigma(x):
    return torch.sigmoid(x)
def sigma_prime(x):
    return sigma(x)*(1-sigma(x))


torch.manual_seed(0)
L = 6
X_data = torch.rand(4, 1)
Y_data = torch.rand(1, 1)

A_list,b_list = [],[]
for _ in range(L-1):
    A_list.append(torch.rand(4, 4))
    b_list.append(torch.rand(4, 1))
A_list.append(torch.rand(1, 4))
b_list.append(torch.rand(1, 1))




# # Option 1: directly use PyTorch's autograd feature
# for A in A_list:
#     A.requires_grad = True
# for b in b_list:
#     b.requires_grad = True

# y = X_data
# for ell in range(L):
#     S = sigma if ell<L-1 else lambda x: x
#     y = S(A_list[ell]@y+b_list[ell])

# # backward pass in pytorch
# loss=torch.square(y-Y_data)/2
# loss.backward()

# print(A_list[0].grad)




# # Option 2: construct a NN model and use backprop
# class MLP(nn.Module) :
#     def __init__(self) :
#         super().__init__()
#         self.linear = nn.ModuleList([nn.Linear(4,4) for _ in range(L-1)])
#         self.linear.append(nn.Linear(4,1))
#         for ell in range(L):
#             self.linear[ell].weight.data = A_list[ell]
#             self.linear[ell].bias.data = b_list[ell].squeeze()
#     def forward(self, x) :
#         x = x.squeeze()
#         for ell in range(L-1):
#             x = sigma(self.linear[ell](x))
#         x = self.linear[-1](x)
#         return x

# model = MLP()

# loss = torch.square(model(X_data)-Y_data)/2
# loss.backward()

# print(model.linear[0].weight.grad)



# Option 3: implement backprop yourself
y_list = [X_data]
y = X_data
for ell in range(L):
    S = sigma if ell<L-1 else lambda x: x
    y = S(A_list[ell]@y+b_list[ell])
    y_list.append(y)


dA_list = []
db_list = []
dy = y-Y_data  # dloss/dy_L
for ell in reversed(range(L)):
    S = sigma_prime if ell<L-1 else lambda x: torch.ones(x.shape)
    A, b, y = A_list[ell], b_list[ell], y_list[ell]

    S_prime = torch.diag(S(A@y+b).reshape(-1))
    db = dy@S_prime if ell<L-1 else dy    # dloss/db_ell
    dA = (S_prime@dy.reshape(-1,1)) @ y.reshape(1,-1) if ell<L-1 else dy*y   # dloss/dA_ell
    dy = dy@S_prime@A if ell<L-1 else dy*A    # dloss/dy_{ell-1}

    dA_list.insert(0, dA)
    db_list.insert(0, db)

print(dA_list[0])
```

Result

```
tensor([[2.3943e-05, 3.7064e-05, 4.2687e-06, 6.3700e-06],
        [3.4104e-05, 5.2794e-05, 6.0804e-06, 9.0735e-06],
        [2.4438e-05, 3.7831e-05, 4.3571e-06, 6.5019e-06],
        [2.0187e-05, 3.1250e-05, 3.5991e-06, 5.3707e-06]])
```

```
# 주석을 바꿔서 돌린 값과 일치한다.
tensor([[2.3943e-05, 3.7064e-05, 4.2687e-06, 6.3700e-06],
        [3.4104e-05, 5.2794e-05, 6.0804e-06, 9.0735e-06],
        [2.4438e-05, 3.7831e-05, 4.3571e-06, 6.5019e-06],
        [2.0187e-05, 3.1250e-05, 3.5991e-06, 5.3707e-06]])
```