通信理論に特化した深層学習
第9回ゼミ資料
AMPネットの学習1

豊橋技術科学大学
電気・電子情報工学系
准教授　竹内啓悟

近似的メッセージ伝播法(AMP)

モジュールA

$$\boldsymbol{x}_{\mathrm{A}}^{t} = \boldsymbol{x}_{\mathrm{B}}^{t} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{z}_{t},$$

$$\boldsymbol{z}_{t} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}_{\mathrm{B}}^{t} + \alpha\xi_{t-1}\boldsymbol{z}_{t-1}.$$

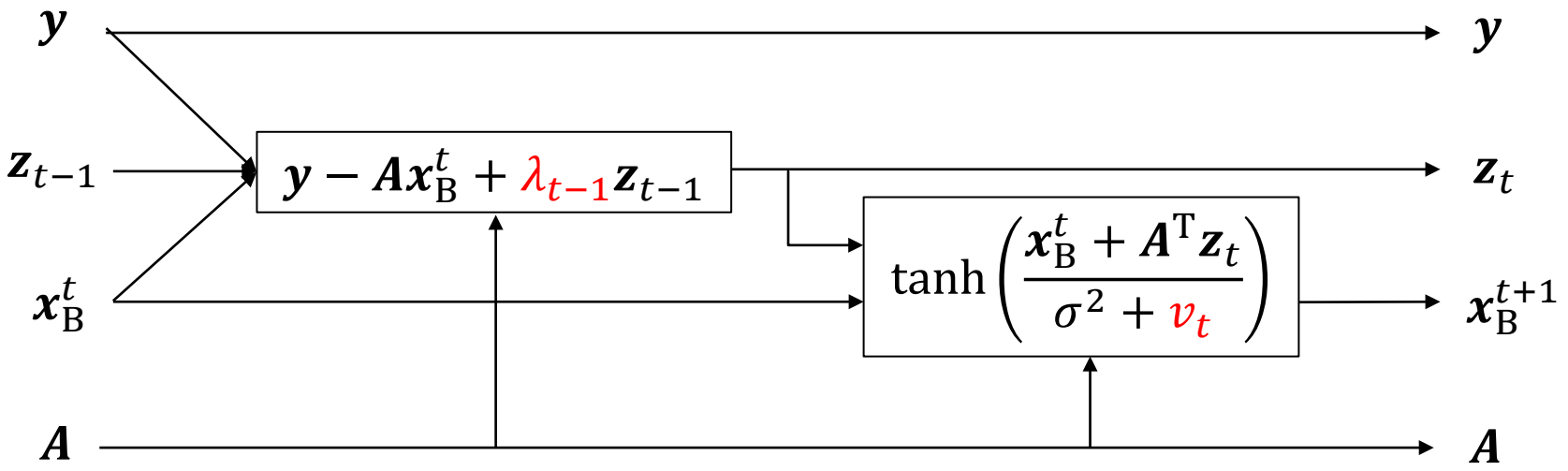
モジュールB

$$\boldsymbol{x}_{\mathrm{B}}^{t+1} = \tanh\left(\frac{\boldsymbol{x}_{\mathrm{A}}^{t}}{\sigma^{2} + \alpha v_{\mathrm{B}}^{t}}\right).$$

関数を要素ごとに適用

反復回数TのAMPをT層順伝播型ネットワークとみなして、
パラメータ$\alpha\xi_{t-1}$と$\alpha v_{\mathrm{B}}^{t}$をデータから学習したい。

第$t+1$層目のレイヤー



注意

- 通信路行列$\boldsymbol{A}$は真の値を使い、学習対象としない。
- 入力層では$\boldsymbol{z}_{-1} = \boldsymbol{0}$と$\boldsymbol{x}_{\mathrm{B}}^{0} = \boldsymbol{0}$を使用する。
- パラメータ$\{\lambda_{t-1}, v_{t}\}$をデータから学習する。
- 初期値は$\lambda_{t-1} = 0$と$v_{t} = 0$とする。

残留干渉雑音を無視した場合の並列干渉除去から学習開始

行ベクトル表現

入力を列ベクトルのシステムから行ベクトルのシステムに変換する。

通信路モデル

$$\boldsymbol{y} = \boldsymbol{xA} + \boldsymbol{w} \in \mathbb{R}^{1\times M}, \qquad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_M).$$

$\boldsymbol{A} \in \mathbb{R}^{N\times M}$: 通信路行列

$\boldsymbol{z}_t \in \mathbb{R}^{1\times M}$ の再定義

$$\boldsymbol{z}_t = \boldsymbol{y} - \boldsymbol{x}_{\mathrm{B}}^{t}\boldsymbol{A} + \lambda_{t-1}\boldsymbol{z}_{t-1}$$

$\boldsymbol{x}_{\mathrm{B}}^{t+1} \in \mathbb{R}^{1\times N}$ の再定義

$$\boldsymbol{x}_{\mathrm{B}}^{t+1} = \tanh\left(\frac{\boldsymbol{x}_{\mathrm{B}}^{t} + \boldsymbol{z}_t\boldsymbol{A}^{\mathrm{T}}}{\sigma^2 + v_t}\right)$$

訓練データの生成関数

```
import numpy as np
import tensorflow as tf
import math

def MIMO_channel(M, N, L, L_INNER, SNR_dB):
    inputs = 1. – 2.*np.random.randint(2, size = (L, L_INNER, N))
    #一様乱数行列X[l] ∈ {1, −1}^(L_inner×N)をL個用意
    #L_innerは伝送に同じ通信路行列を使いまわす回数
    inputs = inputs.astype(np.float32)
    A = np.random.normal(size = (L, N, M), scale = 1.0/math.sqrt(M))
    #平均0、標準偏差scaleのガウス乱数行列A[l] ∈ ℝ^(N×M)をL個用意
    sigma = math.pow(10., -SNR_dB/20.)
    outputs = np.matmul(inputs, A) + np.random.normal(size = (L,
L_INNER, M), scale = sigma)
    #第1項は行列・行列積X[l]A[l] ∈ ℝ^(L_inner×M)をL回計算する
    sigma2 = tf.constant(sigma*sigma, dtype = 'float32')
    #固定テンソル（パラメータ）の定義
    return inputs, outputs, A, sigma2
```

AMPに対応するレイヤー

```
class AMPLayer(tf.keras.layers.Layer):
    def __init__(self, sigma2, init_v, **kwargs):
        super(AMPLayer, self).__init__(**kwargs)
        self.sigma2 = sigma2
        self.init_v = init_v

    def build(self, input_shape):
        self.Lambda = self.add_weight("Lambda", initializer =
'zeros', trainable = True, constraint = 'non_neg')
        self.v = self.add_weight("v", initializer =
tf.keras.initializers.Constant(self.init_v), trainable = True,
constraint = 'non_neg')

    def call(self, inputs):
        ....
```

名前の定義に後で使用

#非負制約

AMPに対応するレイヤーのcallメソッド

```
def call(self, inputs):
    x = inputs[0]
    y = inputs[1]
    z = inputs[2]
    A = inputs[3]

    output_z = y - tf.linalg.matmul(x, A) + self.Lambda*z
    output_x = x + tf.linalg.matmul(output_z, A, transpose_b = True)
    output_x = tf.math.tanh(output_x/(self.sigma2 + self.v))

    return [output_x, y, output_z, A]
```

$(\boldsymbol{x}_{\mathrm{B}}^{t}, \boldsymbol{y}, \boldsymbol{z}_{t-1}, \boldsymbol{A})$を$(\boldsymbol{x}_{\mathrm{B}}^{t+1}, \boldsymbol{y}, \boldsymbol{z}_{t}, \boldsymbol{A})$に写像する。