通信理論に特化した深層学習
第10回ゼミ資料
AMPネットの学習２

豊橋技術科学大学
電気・電子情報工学系
准教授　竹内啓悟

TOYOHASHI UNIVERSITY OF TECHNOLOGY

損失関数の定義

Multi-loss型の平均二乗誤差

$$\mathrm{Loss}(\boldsymbol{x}, \{\boldsymbol{x}_{\mathrm{B}}^{t}\}_{t=1}^{T}) = \sum_{t=0}^{T-1} \eta^{T-t-1} \left\| \boldsymbol{x} - \boldsymbol{x}_{\mathrm{B}}^{t+1} \right\|^{2}, \qquad \eta \in [0, 1]$$

最終結果だけでなく、反復途中の誤差も小さくなるようにする。

$\eta = 0$の場合

$$\mathrm{Loss}(\boldsymbol{x}, \{\boldsymbol{x}_{\mathrm{B}}^{t}\}_{t=1}^{T}) = \left\| \boldsymbol{x} - \boldsymbol{x}_{\mathrm{B}}^{T} \right\|^{2}$$

最終結果のみを考慮する。

TensorFlowは複数個の損失関数の重み付き和を最適化できる。

クラスOutputの準備

```
import tensorflow as tf

class Output():
    def BER(self, y_true, y_pred):
        a = tf.math.not_equal(y_true, tf.math.sign(y_pred))
        return tf.keras.backend.mean(a)
        #y_trueとy_predの要素の符号が異なる割合を計算

    def plot_weights(self, weights):
        T = int(len(weights)/2)
        with open( 'weight.txt' , mode = 'w') as f:
            for t in range(T):
                print(weights[2*t], weights[2*t + 1], file = f)
```

評価関数の追加

plot_weights関数の修正

出力データの形式　(オンサーガ項の係数λ_{t-1}, モジュールBの分散v_t)

⋮

TOYOHASHI UNIVERSITY OF TECHNOLOGY

必要な関数等のインポート

以降のコードを順に記述せよ。

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
import numpy as np

from tools.AMP import MIMO_channel, AMPLayer, Output
out = Output()
```

訓練データの生成関数MIMO_channelおよびAMPレイヤーを表すクラスAMPLayerを./tools/AMP.pyに記述した。

第9回ゼミ資料を参照

パラメータの設定とデータの生成

```
M = 64 #受信次元
N = 64 #送信次元
T = 3 #レイヤー数(AMPの反復回数に相当)
L_INNER = 64
L_TRAIN = 1024
L_VAL = 512
L_TEST = 1024
#訓練用、検証用、評価用のデータサイズ
L = L_TRAIN + L_VAL + L_TEST
SNR_dB = 10 #SNRを10 dBに設定
BATCH_SIZE = 64
EPOCS = 20 #エポック数
EPSILON = 0.005 #学習率の設定
ETA = 0.9 #損失関数の重み

inputs, outputs, A, sigma2 = MIMO_channel(M, N, L, L_INNER, SNR_dB)
```

訓練用データ形式の調整

$\boldsymbol{x}_{\mathrm{B}}^{0} = \boldsymbol{0}$、$\boldsymbol{z}_{-1} = \boldsymbol{0}$となるように、データに0を詰める。

```
x_train = [
    np.zeros((L_TRAIN, L_INNER, N)),
    outputs[:L_ TRAIN],
    np.zeros((L_TRAIN, L_INNER, M)),
    A[:L_TRAIN]]

y_train = []        空のリスト
for t in range(T):
    y_train += [inputs[:L_TRAIN]]        リストの結合
```

検証用データ形式の調整

$\boldsymbol{x}_{\mathrm{B}}^{0}=\mathbf{0}$、$\boldsymbol{z}_{-1}=\mathbf{0}$となるように、データに0を詰める。

```
x_val = [
    np.zeros((L_VAL, L_INNER, N)),
    outputs[L_TRAIN:L_TRAIN + L_VAL],
    np.zeros((L_VAL, L_INNER, M)),
    A[L_TRAIN:L_TRAIN + L_VAL]]

y_val = []
for t in range(T):
    y_val += [inputs[L_TRAIN:L_TRAIN + L_VAL]]
```

評価用データ形式の調整

$\boldsymbol{x}_{\mathrm{B}}^{0}=\mathbf{0}$、$\boldsymbol{z}_{-1}=\mathbf{0}$となるように、データに0を詰める。

```
x_test = [
    np.zeros((L_TEST, L_INNER, N)),
    outputs[L_TRAIN + L_VAL:],
    np.zeros((L_TEST, L_INNER, M)),
    A[L_TRAIN + L_VAL:]]

y_test = []
for t in range(T):
    y_test += [inputs[L_TRAIN + L_VAL:]]
```

AMPネットの生成

```
init_v = []
for t in range(T):
   init_v += [[0.]]

inputs = [
  tf.keras.Input(shape = (L_INNER, N,)),
  tf.keras.Input(shape = (L_INNER, M,)),
  tf.keras.Input(shape = (L_INNER, M,)),
  tf.keras.Input(shape = (N, M,))]
v = inputs
u = []
for t in range(T):
    v = AMPLayer(sigma2, init_v[t], name = 'AMPLayer' + str(t))(v)
    u += v
```

v_tの初期値をすべて0に設定

$(\boldsymbol{x}_{\mathrm{B}}^0,\ \boldsymbol{y},\ \boldsymbol{z}_{-1}, \boldsymbol{A})$を入力とする。

レイヤー名をAMPLayer2等に設定

AMPネットの生成

```
outputs = []
loss_weights = {}
eta_t = 1.0
for t in range(T - 1, -1, -1):
    outputs += [u[4*t]]
    loss_weights['AMPLayer' + str(t)] = eta_t
    eta_t *= ETA

model = tf.keras.Model(inputs = inputs, outputs = outputs)

model.summary()
```

空の辞書

重みを降順で $\{1, \eta, \ \eta^2, \dots\}$に設定

入力をinputs、出力を$\{\boldsymbol{x}_{\mathrm{B}}^{T}, \boldsymbol{x}_{\mathrm{B}}^{T-1}, \dots\}$に設定

TOYOHASHI UNIVERSITY OF TECHNOLOGY

model.summary()の実行結果

```
________________________________________________________________________________
Layer (type)                Output Shape         Param #     Connected to
==================================================================================
input_1 (InputLayer)        [(None, 64, 64)]     0
________________________________________________________________________________
input_2 (InputLayer)        [(None, 64, 64)]     0
________________________________________________________________________________
input_3 (InputLayer)        [(None, 64, 64)]     0
________________________________________________________________________________
input_4 (InputLayer)        [(None, 64, 64)]     0
________________________________________________________________________________
AMPLayer0 (AMPLayer)        [(None, 64, 64), (Non 2     input_1[0][0]
                                  input_2[0][0]     input_3[0][0]     input_4[0][0]
________________________________________________________________________________
AMPLayer1 (AMPLayer)        [(None, 64, 64), (Non 2     AMPLayer0[0][0]
                                  AMPLayer0[0][1]   AMPLayer0[0][2]   AMPLayer0[0][3]
________________________________________________________________________________
AMPLayer2 (AMPLayer)        [(None, 64, 64), (Non 2     AMPLayer1[0][0]
                                  AMPLayer1[0][1]   AMPLayer1[0][2]   AMPLayer1[0][3]
==================================================================================
Total params: 6
Trainable params: 6
Non-trainable params: 0
________________________________________________________________________________
```

学習の実行

```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate =
EPSILON), loss = 'mean_squared_error', loss_weights =
loss_weights, metrics = [out.BER])
#個別の損失関数を平均二乗誤差、損失関数の重みを
辞書loss_weightsで設定、評価関数をBERに設定
#SGDの場合はoptimizer = tf.keras.optimizers.SGD(learning_rate =
EPSILON)

training_history = model.fit(x_train, y_train, epochs = EPOCS,
batch_size = BATCH_SIZE, validation_data = (x_val, y_val))

weights = model.get_weights()
out.plot_weights(weights)

BATCH_SIZE = 64 #計算が高速化されるように適切に設定
print(model.evaluate(x_test, y_test, batch_size = BATCH_SIZE))
```

学習時の出力

2/20 [..............................] - loss: 2.2093 - AMPLayer2_loss: 0.8715 - AMPLayer1_loss: 0.9185
- AMPLayer0_loss: 0.6310 - AMPLayer2_BER: 0.2251 - AMPLayer1_BER: 0.2471 - AMPLayer0_BER: 0.
3/20 [..............................] - loss: 2.0870 - AMPLayer2_loss: 0.7969 - AMPLayer1_loss: 0.8742
- AMPLayer0_loss: 0.6214 - AMPLayer2_BER: 0.2069 - AMPLayer1_BER: 0.2372 - AMPLayer0_BER:0.

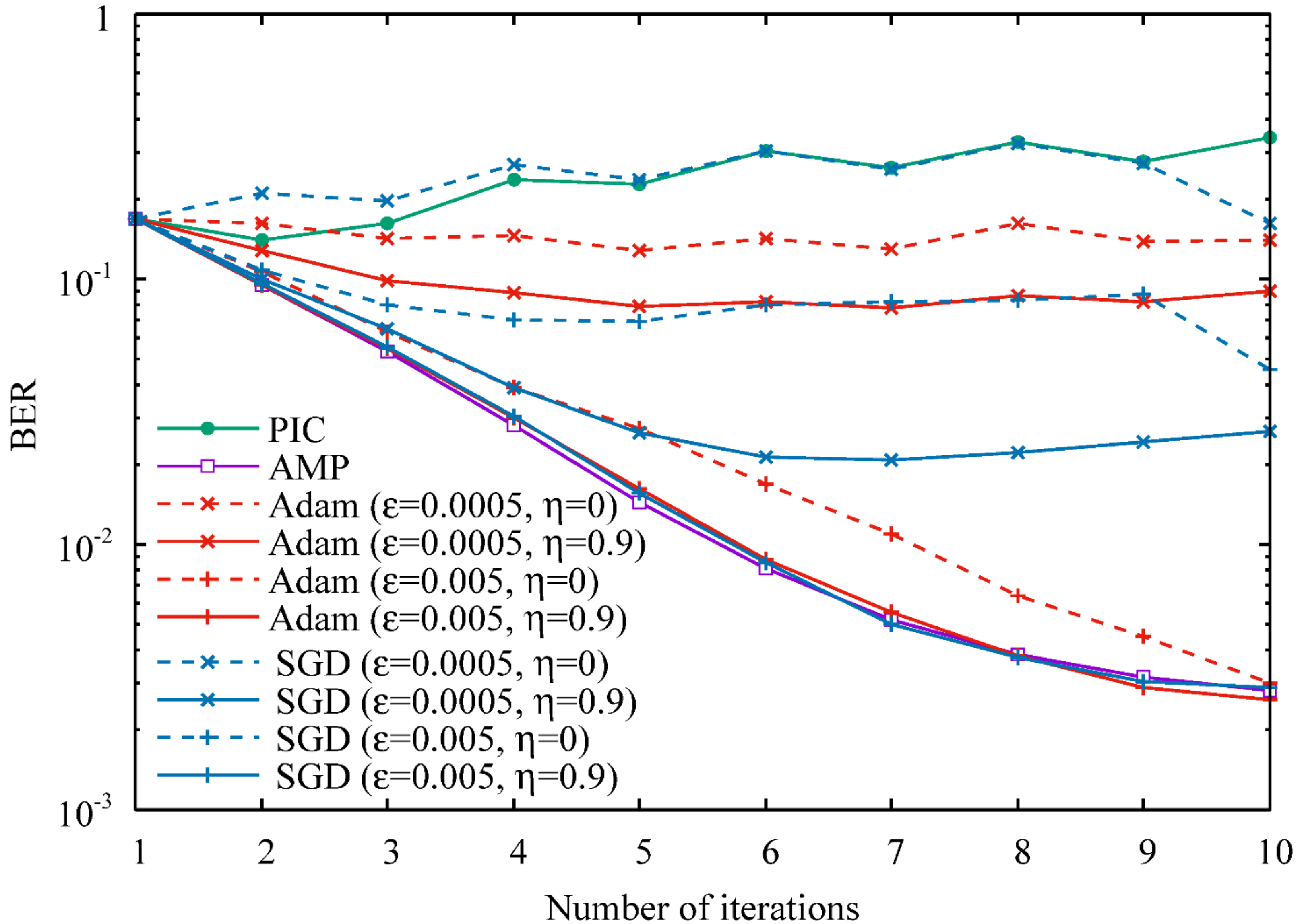
左から順に、損失関数の重み付き和、個別の損失関数値(重みを含まず)、個別の評価関数値

評価結果も同順で出力される。

[0.9177173115313053, 0.21792291, 0.32305843, 0.5049899, 0.071640015, 0.10925293, 0.16622925]

一部の表示を削除済み

数値シミュレーション



$M = N = 64$、$1/\sigma^2 = 10$ dB