作业一：MLQP

超并行机器学习与海量数据挖掘　EI328
2022 年春季工程实践与科技创新课程 IV-J

姓名：李子龙　学号：518070910095　日期：2022 年 3 月 8 日

目录

1 推导

问题 1. Suppose the output of each neuron in a multi- layer perceptron is:

$$x_{kj} = f\left(\sum_{i=1}^{N_{k-1}} (u_{kji} x_{k-1,i}^2 + v_{kji} x_{k-1,i}) + b_{kj}\right) \tag{1}$$

where both u_{kji} and v_{kji} are the weights connecting the i^{th} unit in the layer $k-1$ to the j^{th} unit in the layer $k \in [2, M]$, b_{kj} is the bias of the j^{th} unit in the layer $k \in [2, M]$, N_k is the number of units if $1 \leq k \leq M$ and $f(\cdot)$ is the sigmoidal activation function.

Please derive a back-propagation algorithm for multilayer quadratic perceptron (MLQP) in on-line or sequential mode.

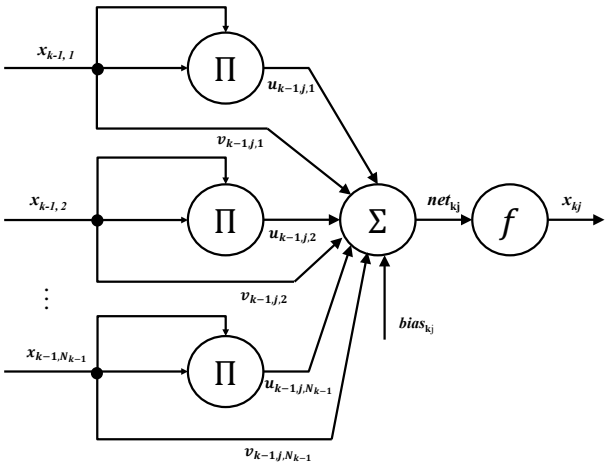


图 1: MLQP

1.1 输出神经元

对于输出神经元信号 x_{Mj} 来说，令 d_{Mj} 为目标值，则对应的误差为

$$e_{Mj} = d_{Mj} - x_{Mj}$$

使用均方误差计算损失函数

$$\mathcal{E} = \sum_{j=1}^{N_M} \frac{1}{2} e_{Mj}^2 \tag{2}$$

则对于 net_{Mj} 的梯度为

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial net_{Mj}} &= \frac{\partial \mathcal{E}}{\partial e_{Mj}} \frac{\partial e_{Mj}}{\partial x_{Mj}} \frac{\partial x_{Mj}}{\partial net_{Mj}} \\ &= -e_{Mj} f'(net_{Mj}) \end{aligned}$$

定义对应的局部梯度（local gradient）为

$$\delta_{Mj} = -\frac{\partial \mathcal{E}}{\partial net_{Mj}} = e_{Mj} f'(net_{Mj}) \tag{3}$$

1.2 隐藏层

对于隐藏神经元上的 $net_{k-1,j}$ 而言，假设后一层反向传播来的局部梯度已知：

$$\delta_{kj} = -\frac{\partial \mathcal{E}}{\partial net_{kj}}$$

则该层的局部梯度为

$$\begin{aligned}\delta_{k-1,i} = -\frac{\partial \mathcal{E}}{\partial net_{k-1,i}} &= -\frac{\partial \mathcal{E}}{\partial x_{k-1,i}}\frac{\partial x_{k-1,i}}{\partial net_{k-1,i}} \\ &= -f'(net_{k-1,i})\sum_{j=1}^{N_j}\frac{\partial \mathcal{E}}{\partial net_{kj}}\frac{\partial net_{kj}}{\partial x_{k-1,i}} \\ &= f'(net_{k-1,i})\sum_{j=1}^{N_j}\delta_{kj}(2u_{kji}x_{k-1,i} + v_{kji}) \end{aligned} \tag{4}$$

1.3 更新权值

而根据公式 (1)，

$$\frac{\partial net_{kj}}{\partial u_{kji}} = x_{k-1,i}^2$$
$$\frac{\partial net_{kj}}{\partial v_{kji}} = x_{k-1,i}$$

设定学习率分别为 η_1, η_2，则对应权重的修正值

$$\Delta u_{kji} = -\eta_1\frac{\partial \mathcal{E}}{\partial u_{kji}} = -\eta_1\frac{\partial \mathcal{E}}{\partial net_{kj}}\frac{\partial net_{kj}}{\partial u_{kji}} = \eta_1\delta_{kj}x_{k-1,i}^2 \tag{5}$$

$$\Delta v_{kji} = -\eta_2\frac{\partial \mathcal{E}}{\partial v_{kji}} = -\eta_2\frac{\partial \mathcal{E}}{\partial net_{kj}}\frac{\partial net_{kj}}{\partial v_{kji}} = \eta_2\delta_{kj}x_{k-1,i} \tag{6}$$

这个结果与 [1] 基本一致。

1.4 更新偏移

类似地由于

$$\frac{\partial net_{kj}}{\partial b_{kj}} = 1$$

则设定学习率为 η_3，则对应的偏移修正值

$$\Delta b_{kj} = -\eta_3\frac{\partial \mathcal{E}}{\partial b_{kj}} = -\eta_3\frac{\partial \mathcal{E}}{\partial net_{kj}}\frac{\partial net_{kj}}{\partial b_{kj}} = \eta_3\delta_{kj} \tag{7}$$

2 实现

问题 2. Please implement an on-line BP algorithm for MLQP (you can use any programming language), train an MLQP with one hidden layer to classify two spirals problem, and compare the training time and decision boundaries at three different learning rates.

2.1 结构设计

如图 2，输入为两个维度上的坐标 x, y。采用单输出与硬阈值分类，阈值设定为 0.5，按照 [2] 的说法，识别时将大于等于 0.5 的设为正类，小于 0.5 的定义为负类，但这并不影响训练过程，因为采用的是原始输出作为误差的计算。

由于只需要一个隐藏层，隐藏层神经元个数为 N_2，那么参数个数为

$$2(2N_2 + N_2) + N_2 + 1 = 7N_2 + 1$$

由于训练集样本数为 300，假设我们想要保持 10% 的分类错误误差，那么为了好的泛化能力，参数个数应当满足

$$300 = O\left(\frac{7N_2 + 1}{10\%}\right)$$

N_2 至少应当是 5 的倍数级别，仿照 [1]，这里取 10。

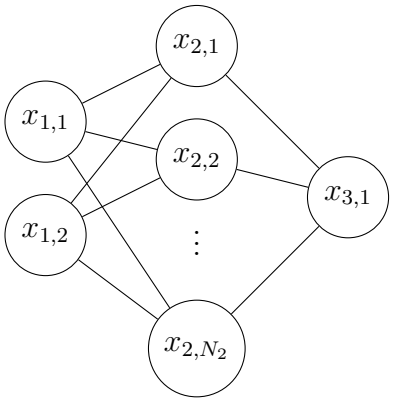


图 2: 网络结构

2.2 算法小结

下面将采用矩阵记号简化运算，以下 $\times$ 均表示矩阵的逐点相乘。设第 $k-1$ 层的数值矩阵为 $\mathbf{X}_{k-1} = (x_{k-1,i})_{N_{k-1}\times 1}$，其逐项平方矩阵为 $\mathbf{Y}_{k-1} = (x^2_{k-1,i})_{N_{k-1}\times 1} = \mathbf{X}_{k-1} \times \mathbf{X}_{k-1}$，向后的权重矩阵为 $\mathbf{U}_k = (u_{kji})_{N_k \times N_{k-1}}$, $\mathbf{V}_k = (v_{kji})_{N_k \times N_{k-1}}$，偏移量矩阵 $\mathbf{B}_k = (b_{kj})_{N_k \times 1}$。

2.2.1 前向运算

公式 (1) 将被改写为

$$\mathbf{X}_k = f\left(\mathbf{U}_k\mathbf{Y}_{k-1} + \mathbf{V}_k\mathbf{X}_{k-1} + \mathbf{B}_k\right)$$

为了简单起见，保留中间结果

$$\mathbf{N}_k = \mathbf{U}_k\mathbf{Y}_{k-1} + \mathbf{V}_k\mathbf{X}_{k-1} + \mathbf{B}_k \tag{8}$$

$$\mathbf{X}_k = f(\mathbf{N}_k) \tag{9}$$

2.2.2 后向运算

令 $\mathbf{D}_M$ 为目标输出矩阵，将公式 (3) 改写，输出层的梯度计算为

$$\boldsymbol{\delta}_M = (\delta_{M,i})_{N_M \times 1} = (\mathbf{D}_M - \mathbf{X}_M) \times f'(\mathbf{N}_M) \tag{10}$$

改写公式 (4)，隐含层的梯度计算为

$$\boldsymbol{\delta}_{k-1} = \left(\mathbf{U}_k^T \boldsymbol{\delta}_k \times 2\mathbf{X}_{k-1} + \mathbf{V}_k^T \boldsymbol{\delta}_k\right) \times f'(\mathbf{N}_{k-1}) \tag{11}$$

改写公式 (5) 和公式 (6)，权重修正矩阵

$$\begin{aligned}\boldsymbol{\Delta}\mathbf{U}_k &= (\Delta u_{kji})_{N_k \times N_{k-1}} = \eta_1 \boldsymbol{\delta}_k \mathbf{Y}_{k-1}^T \\ \boldsymbol{\Delta}\mathbf{V}_k &= (\Delta v_{kji})_{N_k \times N_{k-1}} = \eta_2 \boldsymbol{\delta}_k \mathbf{X}_{k-1}^T\end{aligned}$$

改写公式 (7)，偏移修正矩阵

$$\boldsymbol{\Delta}\mathbf{B}_k = (\Delta b_{kj})_{N_k \times 1} = \eta_3 \boldsymbol{\delta}_k$$

2.2.3 动量项

为了加速收敛过程，对修正项添加动量项

$$\boldsymbol{\Delta}\mathbf{U}_k(t+1) = \alpha_1 \boldsymbol{\Delta}\mathbf{U}_k(t) + \eta_1 \boldsymbol{\delta}_k \mathbf{Y}_{k-1}^T \tag{12}$$

$$\boldsymbol{\Delta}\mathbf{V}_k(t+1) = \alpha_2 \boldsymbol{\Delta}\mathbf{V}_k(t) + \eta_2 \boldsymbol{\delta}_k \mathbf{X}_{k-1}^T \tag{13}$$

$$\boldsymbol{\Delta}\mathbf{B}_k(t+1) = \alpha_3 \boldsymbol{\Delta}\mathbf{B}_k(t) + \eta_3 \boldsymbol{\delta}_k \tag{14}$$

其中 $\alpha_1, \alpha_2, \alpha_3$ 为动量常数。

2.3 训练过程

参数初始化 简便起见，将三个参数的学习率设置为一个相等的值 $\eta_1 = \eta_2 = \eta_3 = \eta$，将动量常数也设为相同的值 $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$。

类似于 [3]，由于参数数量相对于线性层加倍，将参数初始化为 $\mathcal{U}\left(-\sqrt{\frac{1}{2\times 2}}, \sqrt{\frac{1}{2\times 2}}\right) = \mathcal{U}(-0.5, 0.5)$ 上的随机值，其中 $\mathcal{U}$ 为一致分布。

训练 数据范围是 $x \in [-6, 6], y \in [-6, 6]$。训练与验证将采用 3 折交叉验证，测试集只用于测试最终的训练效果。

早停 类似 [4] 中早停机制的实现，将会在最佳验证误差后没有进展的 `MAX_EPOCHS`= 200 后停止训练，并恢复最佳模型。

可以设定一个训练轮次阈值来防止无限训练。

2.4 实验结果

为公平起见，防止偶然性，对 $\eta = 0.01, 0.05, 0.1$ 三个学习率，将动量常数固定为 $\alpha = 0.8$，都用同样的一组随机种子取平均得到其对应的平均训练时间以及平均测试误差，结果如表 1 所示。由表可见，$\eta = 0.05$ 综合表现最好。

表 1: 不同学习率训练时间

η	0.01	0.05	0.1
平均训练时间(s)	108	71	32
平均测试误差(MSE)	0.1227	0.1188	0.1303

都采用 42 随机种子，可视化边界结果如图 3 所示，图的下方为训练集误差、验证集误差、监测测试误差对于不同轮的趋势图。

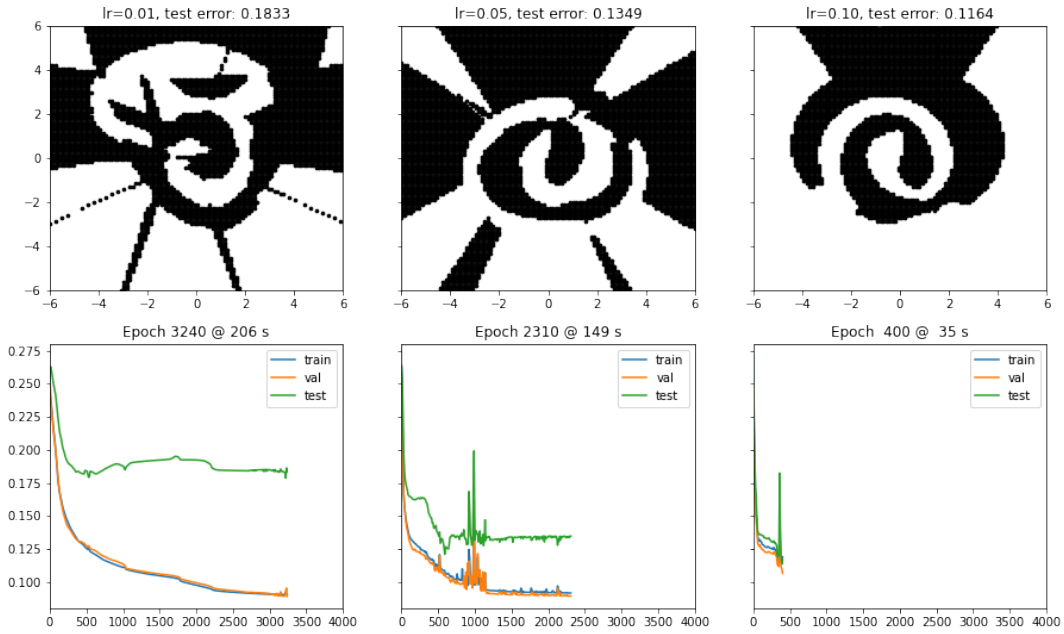


图 3: 不同学习率测试结果（随机种子 42）

3 最小最大模块网络

问题 3. This section will construct a min-max modular network.

(1) Divide the two spirals problem into four or nine sub-problems randomly and with prior knowledge, respectively.

(2) Train MLQP on these sub-problems and construct two min-max modular networks.

(3) Compare training time and decision boundaries of the above two min-max modular networks.

3.1 分解子问题

根据 [5]，对于子问题有两个原则。

引理 1 (最小化原则). 训练集具有相等正类集合的模块应当进行最小化合并。

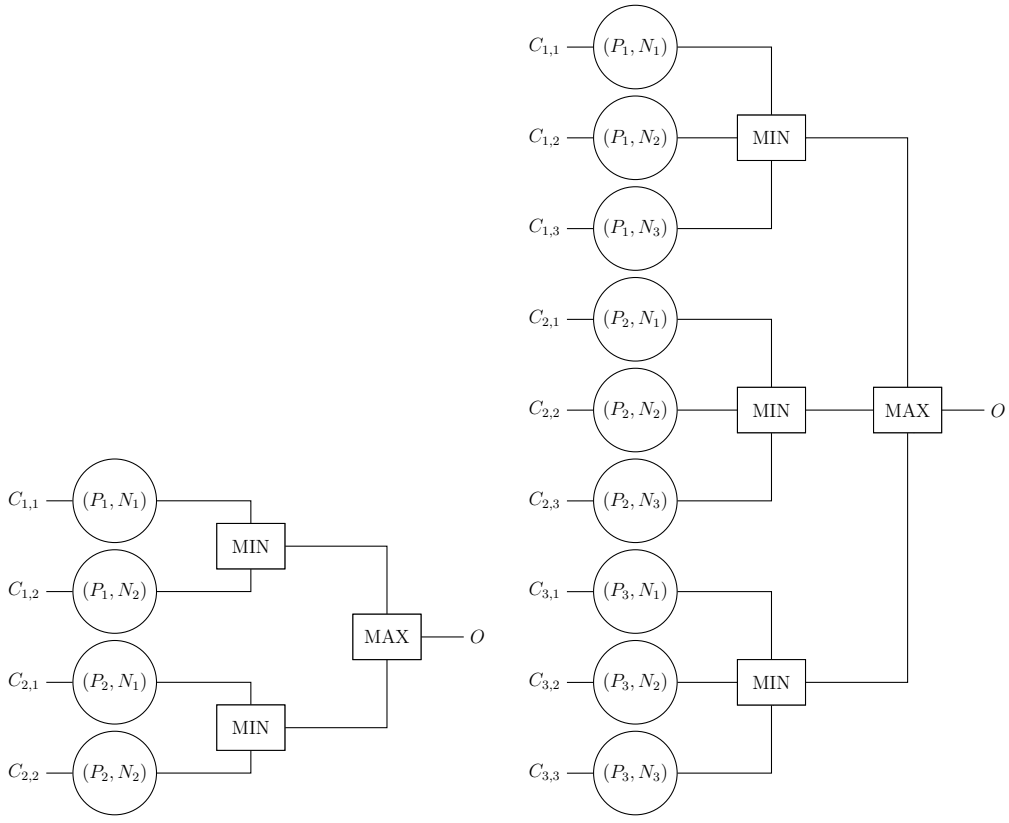
引理 2 (最大化原则). 训练集具有相等负类集合的模块应当进行最大化合并。

分解方法是将训练集按照正类集合 P 和负类集合 N 分割，然后将正类集合随机平分为 k 份$P_1, P_2, \cdots, P_k$，负类集合也同样分为 $N_1, N_2, \cdots, N_k$，之后从正类小集和负类小集里分别取一个组合成新的训练集

$$C_{ij} = (P_i, N_j) \quad \forall i, j \in \mathbb{N} \cap [1, k]$$

3.2 构造最小最大模块网络

如图 4 所示，MIN 节点负责将同正类集的结果合并，MAX 节点负责将同负类集的节点合并。需要合并 k^2 个训练单元的结果。



(a) 4 个子问题 (b) 9 个子问题

图 4: 网络结构

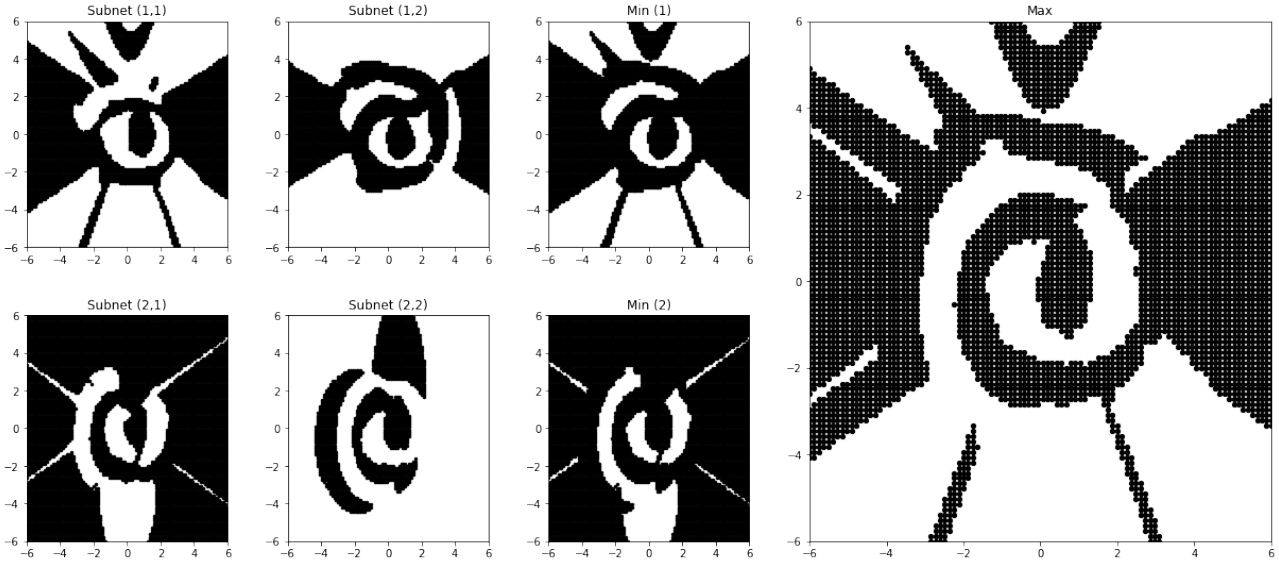
3.3 测试结果

对于两种情况，采用同一组随机种子进行测试，固定 $\eta = 0.05, \alpha = 0.8$，测试结果取平均如表 2 所示。结果显示，随着 k 的增大，每个训练集上的训练时间会减少（由于训练集的

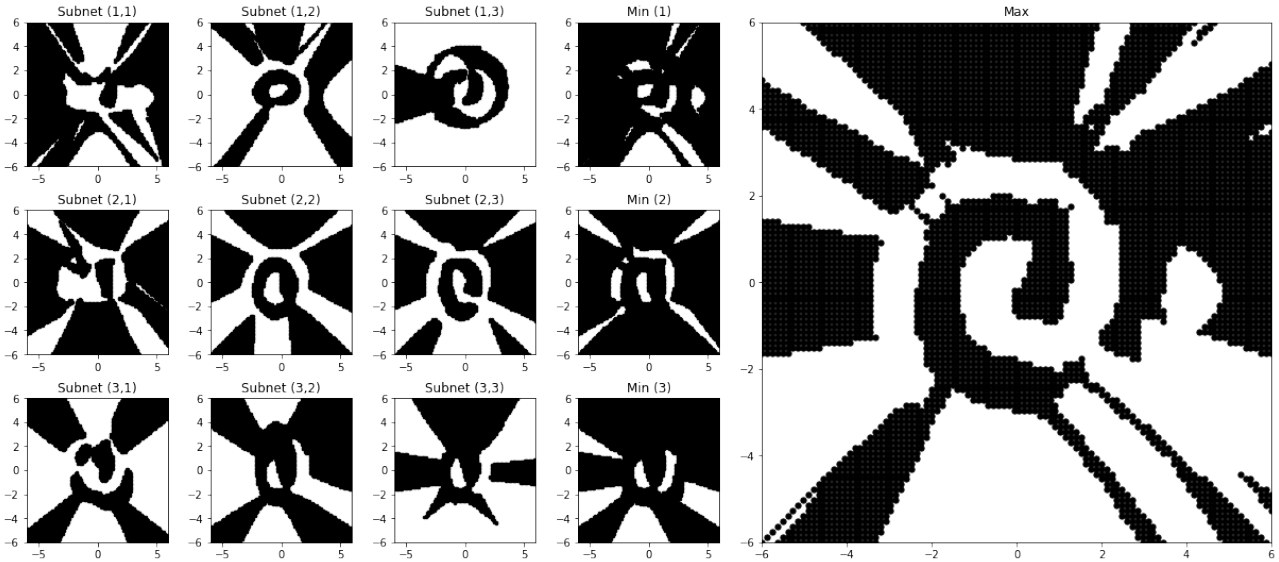
体量小，早停比较快），但是总体测试误差会增大。对于随机种子 42 的网络结果可视化结果如图 5 所示。

表 2: 最大最小模块网络测试结果

k	2	3
平均最大时间(s)	118	66
平均测试误差(MSE)	0.1339	0.1459



(a) 4 个子问题



(b) 9 个子问题

图 5: 最小最大模块网络测试结果（随机种子 42）

参考文献

[1] LU B L, BAI Y, KITA H, et al. An efficient multilayer quadratic perceptron for pattern classification and function approximation[C/OL]//Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan): volume 2. IEEE, 1993: 1385-1388. DOI: 10.1109/ijcnn.1993.716802.

[2] 胡晓武, 秦婷婷, 李超, 等. 智能之门[M]. 高等教育出版社, 2020.

[3] Linear — pytorch master documentation[M/OL]. PyTorch, 2022. https://pytorch.org/docs/master/generated/torch.nn.Linear.html#torch.nn.Linear.

[4] TensorFlow early stopping[EB/OL]. 2022. https://github.com/tensorflow/estimator/blob/f87c8d324de9a895dbf4eefb7a862518e65e7fdd/tensorflow_estimator/python/estimator/early_stopping.py#L399-L420.

[5] LU B L, ITO M. Task decomposition and module combination based on class relations: a modular neural network for pattern classification[J/OL]. IEEE Transactions on Neural Networks, 1999, 10(5): 1244-1256. DOI: 10.1109/72.788664.

A 运行方法

所有的代码文件放在 src 文件夹中。运行程序前需要进入 `src` 目录。

图 6: 源文件目录结构

A.1 MLQP 模型

运行 `src/model.py` 。

```
python model.py [-h] [--epochs N] [--train_file TR] [--test_file TE] [--lr
    LR] [--alpha A]

optional arguments:
  -h, --help        show this help message and exit
  --epochs N        number of epochs to train (default: 100)
  --train_file TR   training data file
```

```
  --test_file TE   test data file
  --lr LR          learning rate (default: 0.1)
  --alpha A        Momentum constant (default: 0.8)
```

可以设定训练轮次阈值、训练文件、测试文件、学习率以及动量常数。均具有默认参数。

A.2 MIN-MAX 模块网络

运行 src/minmax.py 。

```
python minmax.py [-h] [--k K] [--epochs N] [--lr LR] [--train_file TR] [--
   test_file TE]

optional arguments:
  -h, --help       show this help message and exit
  --k K            number of split to train (default: 3)
  --epochs N       number of epochs to train (default: 100)
  --lr LR          learning rate (default: 0.1)
  --train_file TR  training data file
  --test_file TE   test data file
```

可以设定分割份数、训练轮次阈值、学习率、训练文件、测试文件。均具有默认参数。

A.3 结果可视化

打开 src/visual.ipynb 可以看到普通 MLQP 和 MIN-MAX 网络测试可视结果。调用更加底层的函数时可以设定随机种子。

B API 手册

API 手册详见 mlqp.pdf 。

mlqp

Release 1.0

LogCreative

Mar 08, 2022

CONTENTS

CHAPTER

ONE

MINMAX MODULE

class minmax.**Max**(*operands*)
Bases: minmax.Operator

Max module

forward(*src*)
Foward prediction for src input

Parameters **src** (*list*) – the input list [x,y]

class minmax.**Min**(*operands*)
Bases: minmax.Operator

Min module

forward(*src*)
Foward prediction for src input

Parameters **src** (*list*) – the input list [x,y]

class minmax.**Operator**(*operands*)
Bases: object

Base operator class for Min and Max module.

forward(*src*)
Foward prediction for src input

Parameters **src** (*list*) – the input list [x,y]

minmax.**divide**(*train_data*, *k=2*)
Divide the data into positive and negative merged 2D data array.

Parameters

- **train_data** (*array*) – the data to be divided
- **k** (*int*) – the number of split on positive/negative set

Returns the 2D divided data array for computation.

Return type array

minmax.**minmax**(*train_data*, *k*, *epochs*, *lr=0.05*, *random_seed=None*)
Train minmax network.

Parameters

- **train_data** (*array*) – the training data.
- **k** (*int*) – the number of split

- **epochs** (*int*) – the threshold of training epochs.
- **lr** (*float, optional*) – Learning rate. Defaults to 0.05.
- **random_seed** (*int, optional*) – Random Seed. Defaults to None.

Returns Target Network, subnets, min nets, maximum training time among units

Return type Max, array[Net], array[Min], float

minmax.**trainer**(*train_sub_data*, *epochs*, *lr=0.05*, *random_seed=None*)

Trainer worker

Parameters

- **train_sub_data** (*array*) – the input array for training.
- **epochs** (*int*) – the number threshold of epochs.
- **lr** (*float, optional*) – Learning rate. Defaults to 0.05.
- **random_seed** (*int, optional*) – Random Seed. Defaults to None.

Returns the trained network.

Return type Net

CHAPTER

TWO

MODEL MODULE

class model.**Net**(*lr=0.05*, *alpha=0.8*, *random_seed=None*, *hidden_num=10*)

Bases: object

MLQP Network

backward(*pred*, *target*)

Backward pass, update the parameters. NOTE: should run forward pass first before calling this function.

Parameters

- **pred** (*float*) – prediction based on foward pass
- **target** (*float*) – the target label

forward(*src*)

Foward pass, return the prediction based on the given data.

Parameters **src** (*list*) – the input list of data *[x,y]*

param_init()

Init parameters.

model.**cross_validation**(*model*, *split_data*)

Cross validation over split data.

Parameters

- **model** (*Net*) – the instance of Net
- **split_data** (*array*) – the splitted data generated from folds()

Returns the mean of training error and validation error among experiments.

Return type float, float

model.**folds**(*data*, *k*)

divide data sequencially into k portions

Parameters

- **data** (*array*) – the data to be divided
- **k** (*int*) – the number of portions

Returns the divided data

Return type array

model.**split**(*train_data*, *k*)

split train_data into k folds.

Parameters

- **train_data** (*array*) – the training data
- **k** (*int*) – fold number

Returns the splited data formatted [train_data, val_data] array.

Return type array

model.**step**(*model*, *data*, *with_grad=True*)
Common step for data on training or testing.

Parameters

- **model** (*Net*) – the instance of Net
- **data** (*array*) – data for training or testing
- **with_grad** (*bool, optional*) – If it needs backward process. Defaults to True.

Returns the mse loss of this batch of data

Return type loss

model.**test**(*model*, *test_data*)
Test the model

Parameters

- **model** (*Net*) – the instance of Net
- **test_data** (*arrat*) – the testing set

Returns the mse error over test set

Return type float

model.**test_step**(*model*, *test_data*)
Test the model for test_data

Parameters

- **model** (*Net*) – the instance of Net
- **test_data** (*array*) – the testing data

Returns the mse error over test_data

Return type array

model.**train**(*model*, *train_data*, *epochs*, *test_data=None*)
Train the model by epochs.

Parameters

- **model** (*Net*) – the instance of Net
- **train_data** (*array*) – the training set
- **epochs** (*int*) – the number of epochs
- **test_data** (*array, optional*) – if assigned, the test error will be tracked but will not go into the training process.

Returns trained model

Return type Net

model.**train_step**(*model*, *train_data*)
Train the model for one step.

Parameters

- **model** (*Net*) – the instance of Net
- **train_data** (*array*) – the training data

CHAPTER

THREE

UTIL MODULE

util.**mse**(*pred*, *target*)

util.**read_data**(*filename*)

Reads data from the file and return an array of data formatting: [x y label]

Parameters **filename** (*str*) – the path of file

Returns the array of the data read from file

Return type data

util.**sigmoid**(*x*)

util.**sigmoid_prime**(*x*)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX