本科生毕业论文

题目： ADMM-Net: An Algorithm Unrolling Approach For Network Resource Allocation

姓　　名：	谢中林
学　　号：	1700016908
院　　系：	数学科学学院
专　　业：	信息与计算科学
指导教师：	文再文

二〇二一年六月

摘要

网络资源分配是一个极为重要的问题. 一种常用的建模方法是将其抽象为满足网络带宽限制下的效用函数最大化 (NUM) 问题, 这一方法简单有效, 且其原始-对偶算法容易分布式实现. 但在广域网中, 各数据流的可行路径往往不止一条, 各自的需求也不一致, 此时需要发展新的框架来重新分配网络的资源. 在保证大带宽与低延时的需求被满足的情况下, 网络中各线路的负载应尽可能均衡. 为了满足及时性的要求, 应当发展快速求解新模型的方法. 本文的主要工作可以归纳为:

研究了 NUM 问题的背景, 在此基础上提出了一个同时提高分配的公平性并降低延时的多目标优化模型 MOBAC. 这是一个非凸且非光滑的问题, 利用线性化的 ADMM 给出了求解算法. 调研了基于学习的优化范式, 它提供了加速传统算法的途径. 利用算法展开这一技术, 构造了两种基于线性化 ADMM 的 ADMM-Net, 并极大地加速了收敛过程.

关键词: 网络效用最大化, 交替方向乘子法, 基于学习的优化范式, 算法展开

ADMM-Net: An Algorithm Unrolling Approach For Network Resource Allocation

Zhonglin Xie (Computational Mathematics)
Directed by Zaiwen Wen

ABSTRACT

Network resource allocation is an extremely important problem. A commonly used modeling method is to abstract it as a utility function maximization (NUM) problem under the network bandwidth limit. This method is simple and effective, and its primitive-dual algorithm is easy Distributed implementation. However, in a wide area network, there is often more than one feasible path for each data stream, and their respective needs are not consistent. At this time, a new framework needs to be developed to redistribute network resources. The need to ensure large bandwidth and low latency are If it is satisfied, a load of each line in the network should be as balanced as possible. To meet the requirements of timeliness, a method of quickly solving the new model should be developed. The main work of this paper can be summarized as:

The background of the NUM problem is studied, and on this basis, a multi-objective optimization model MOBAC that improves the fairness of distribution and reduces the delay at the same time is proposed. This is a non-convex and non-smooth problem, and the linearized ADMM is used to give Solving algorithm. Learn to optimize is investigated, which provides a way to speed up traditional algorithms. Using the algorithm rolling, two types of ADMM-Net based on linearized ADMM are constructed, and the convergence process is greatly accelerated.

KEY WORDS: Network Utility Maximization, ADMM, Learn to Optimize, Algorithm Unrolling

目录

第一章　Introduction

1.1　Background and Motivation

In wide area networks (WANs), the resource is shared among different various applications. Such as high-denition video applications[1], online gaming[2], and video conferencing[3]. Some of them require low latency, and some of them require high throughput. It is challenging to satisfy their demands at the same time. There are several reasons. First, the network resource is limited but the demands are extreme. Second, the number of available paths is quite different from the demands. A desired solution should cover all of them, such as single-path routing or sparse routing.

Network utility maximization (NUM) is a popular framework for network resource allocation. But the objective function in NUM usually concerns only one measurement performance, we want to build a new framework to meet the various requirements of the applications. This framework leads to a non-smooth non-convex optimization problem, which is quite hard to solve. Fortunately, by introducing an auxiliary variable, the linearized ADMM (LADMM), an algorithm of growing attention, is well suited to solve it. However, LADMM-based algorithm requires many iterations to derive the solution. We need to accelerate the it.

Learn to optimize (L2O) gives a promising increase of the numerical performance. We use one of the L2O paradigm named algorithm unrolling to accelerate the convergence speed of LADMM-based algorithm. The numerical experiments indicates the success of the rolled algorithm.

1.2　Organization

In chapter 二, we give a detailed description of the system model. Then, we propose two frameworks to model the network resource problem. After that, we give a brief introduction of ADMM. The linearized ADMM for two frameworks are derived based on a specific utility function, which meets the demand of various applications.

In chapter 三, we introduce the learn to optimize (L2O). It includes three paradigms: model-free, Plug-and-Play (PnP), and algorithm unrolling, while the first one is more like a neural network and the others maintain the architecture of classical iterations. Based on these works, we propose the ADMM-Net for network resource allocation. Besides, the backward of

ADMM-Net is not trivial even with autograd. We introduce the Moreau envelope to overcome it. This trick is also valid for other ADMM-based networks.

In chapter 四, we compare the different algorithms in various performance measurements. ADMM-Net shows its advantage by reducing the inference cost and give a high-quality approximation of the ground-truth solution.

We conclude the paper in 五. Some reflections on the algorithm unrolling are given. Besides, we discuss the future directions of our work.

第二章　Linearized ADMM for Network Resource Allocation

In this section, we describe the general network resource allocation problem, then give a popular framework named network utility maximization (NUM). Based on NUM, we propose a framework for multi-objective (e.g., utility, load balancing) allocation and path selection problem. Finally, we utilize the separable structure of these frameworks and derive the iterative algorithm based on linearized ADMM.

2.1 Model Settings

We consider K flows indexed by k $(k = 1, 2, \cdots, K)$ which are needed to be delivered from source nodes to destination nodes. The size of the k-th flow is s_k $(s_k > 0, \forall k)$. We denote the overall source sizes vector as $\boldsymbol{s} = (s_1, s_2, \cdots, s_K)^\top$. There are L uni-directional links indexed by l $(l = 1, 2, \cdots, L)$ in the network. The capacity of the l-th link is c_l $(c_l > 0, \forall l)$. We also denote the overall network capacity as a vector $\boldsymbol{c} = (c_1, c_2, \cdots, c_L)^\top$.

For the k-th flow, there is an affiliated source-destination pair and P_k available paths indexed by j $(j = 1, 2, \cdots, P_k, \forall k)$. $\boldsymbol{x}_k = (x_{k,1}, x_{k,2}, \cdots, x_{k,P_k})^\top$ $(x_{k,j} \geq 0, \forall k, j)$ is the resource allocation vector of the k-th flow, where $x_{k,j}$ is the resource allocated at the j-th path for the k-th flow. It is worth mentioning that the path with 0 resource allocated is not selected. As we mentioned in 一, the number of available paths is constrained by $\|\boldsymbol{x}_k\|_0 \leq w_k$. We denote the overall network resource allocation vector as $\boldsymbol{x} = (\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \cdots, \boldsymbol{x}_K^\top)^\top$.

The overall available paths information is represented by a routing matrix. For the k-th flow, the routing matrix

$$\boldsymbol{R}_k = \begin{pmatrix} R^k_{1,1} & R^k_{1,2} & \cdots & R^k_{1,P_k} \\ R^k_{2,1} & R^k_{2,2} & \cdots & R^k_{2,P_k} \\ \vdots & \vdots & \ddots & \vdots \\ R^k_{L,1} & R^k_{L,2} & \cdots & R^k_{L,P_k} \end{pmatrix},$$

where $R^k_{i,j} \in \{0, 1\}$. $R^k_{i,j} = 1$ means the path j of the k-th flow passes the l-th link and vice versa. We define the overall network routing matrix as

$$\boldsymbol{R} = (\boldsymbol{R}_1, \boldsymbol{R}_2, \cdots, \boldsymbol{R}_K), \text{ where } \boldsymbol{R} \in \{0, 1\}^{L \times P}, \ P = \sum_{k=1}^{K} P_k.$$

Fig. 2.1 gives an example of a small network with five links and two flows (black and red).

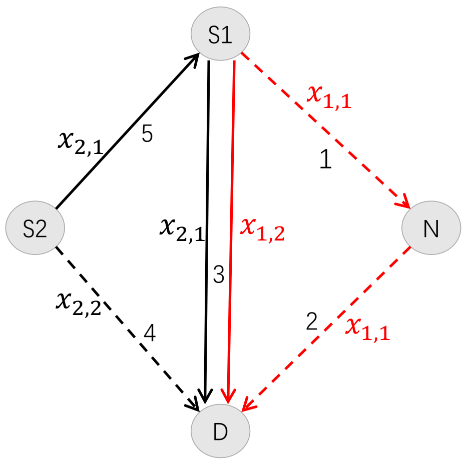

图 2.1 An illustrative example network with ve links and two users. The rst user (red line), whose source node is $S1$ and destination node is D, has rate $x_{1,1}$ and $x_{1,2}$ on its two paths. The second user (black line), whose source node is $S2$ and destination node is D, has rate $x_{2,1}$ and $x_{2,2}$ on its two paths. The numbers next to the lines represent the link indices.

Each flow has two available paths (solid line and dashed line) and the corresponding routing matrices for the two flows are given by

$$\boldsymbol{R}_1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \boldsymbol{R}_2 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

In general, since each path only passes a small portion of all links over the entire network, $\boldsymbol{R}$ is actually very sparse.

2.2 Network Utility Maximization (NUM)

In the 1997 and 1998, the seminal papers[4, 5] proposed an innovative framework for resource allocation. They reduce the resource allocation problem to an optimization problem:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) \\ \text{s.t.} \quad & \boldsymbol{Rx} \le \boldsymbol{c} \\ & \boldsymbol{x} \ge 0 \\ & \|\boldsymbol{x}_k\|_0 \le w_k, \forall k, \end{aligned} \tag{2.1}$$

where $U_k(\cdot)$ is the utility function of the k-th flow. Since $x_{k,j} \geq 0, \forall k, j$, we have $\|\boldsymbol{x}_k\|_1 = \sum_{j=1}^{P_k} x_{k,j}$. Utility functions may be non-decreasing and may depend on delay ($\frac{s_k}{\|\boldsymbol{x}_k\|_1}$), fairness ($\log(\|\boldsymbol{x}_k\|_1)$), etc.

2.3 Multi-objective Bandwidth Allocation with Path Cardinality Constraints (MOBAC)

To avoid the congestion, it is important to balance the load of different links in the network. Thus, we add the link utilization rate term $\frac{\boldsymbol{R}[l]\boldsymbol{x}}{c_l}$ to measure the network load, where $\boldsymbol{R}[l]$ is the l-th row of the routing matrix $\boldsymbol{R}$. We want to minimize the link load in the worst case. Mathematically speaking, we consider the following problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{k=1}^{K} -U_k(\|\boldsymbol{x}_k\|_1) + \alpha \max_{l} \frac{\boldsymbol{R}[l]\boldsymbol{x}}{c_l} \\ \text{s.t.} \quad & \boldsymbol{R}\boldsymbol{x} \leq \boldsymbol{c} \\ & \|\boldsymbol{x}_k\|_0 \leq w_k, \quad \forall k \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{2.2}$$

where α, β is the predefined parameters to balance the different term.

2.4 Linearized ADMM for Network Resource Allocation

In this section, we first give a brief introduction to the alternating direction method of multipliers (ADMM). Then, we derive the linearized ADMM for NUM and MOBAC given an utility function of the following form:

$$U_k(\boldsymbol{x}_k) = \beta \log(\|\boldsymbol{x}_k\|_1) - \frac{s_k}{K\|\boldsymbol{x}_k\|_1}. \tag{2.3}$$

2.4.1 Linearized ADMM

The alternating direction method of multipliers[6] (ADMM) is a variable splitting algorithm of growing popularity due to both simplicity and efficiency. ADMM solves the problem in the following form:

$$\begin{aligned} \min \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c, \end{aligned} \tag{2.4}$$

where $x \in \mathbb{R}^n$, $z \in \mathbb{R}^m$, A is a $p \times n$ matrix, B is a $p \times m$ matrix. Given the penalty parameter ρ, the augmented Lagrangian function of the problem (2.4) is

$$L_\rho(x, z, y) = f(x) + g(z) - y^\top(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2,$$

where y is the Lagrangian multiplier vector of the constrain $Ax + Bz = c$. ADMM recursively performs the iterations:

$$\begin{aligned} x^{k+1} &:= \underset{x}{\operatorname{argmin}} L_\rho\left(x, z^k, y^k\right) \\ z^{k+1} &:= \underset{z}{\operatorname{argmin}} L_\rho\left(x^{k+1}, z, y^k\right) \\ y^{k+1} &:= y^k + \rho\left(Ax^{k+1} + Bz^{k+1} - c\right). \end{aligned}$$

However, the subproblems in above iteration may have no explicit solutions. Many variants of the ADMM have been developed to address this issue. One of them, named linearized ADMM, transforms the x subproblem to

$$x^{k+1} := \underset{x}{\operatorname{argmin}}(\nabla_x L_\rho\left(x^k, z^k, y^k\right)^\top (x - x^k) + \frac{1}{\eta_k}\|x - x^k\|_2^2). \tag{2.5}$$

Actually, the solution of (2.5) equals to performing one step gradient descent to the original subproblem with stepsize η_k. Linearized ADMM fails when $f(x)$ is non-differentiable. We can utilize the linearization process only in the quadratic term $\|Ax + Bz - c\|$ and optimize following problem:

$$x^{k+1} := \underset{x}{\operatorname{argmin}}\big((\rho A^\top(Ax^k + Bz^k - c - \frac{y^k}{\rho}))^\top(x - x^k) + f(x) + \frac{1}{\eta_k}\|x - x^k\|_2^2\big).$$

It is worth mentioning that above problem is separable when $f(x)$ owns the separable structure, which means we can solve the problem by component. This property may simplify the problem significantly.

2.4.2 Linearized ADMM for NUM

Introducing the auxiliary variable $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$, problem (2.1) is converted to an equivalent form:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) \\ \text{s.t.} \quad & \boldsymbol{y} = \boldsymbol{R}\boldsymbol{x} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & \boldsymbol{y} \leq \boldsymbol{c} \\ & \|\boldsymbol{x}_k\|_0 \leq w_k, \forall k. \end{aligned} \tag{2.6}$$

The augmented Lagrangian function of above problem is

$$L_\rho(\boldsymbol{x},\boldsymbol{y};\boldsymbol{z}) = -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) + \boldsymbol{z}^\top(\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}\|_2^2, \tag{2.7}$$

where $\boldsymbol{z}$ is a Lagrangian multiplier vector associated with the constraint $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ in (2.6), and $\rho > 0$ is a penalty parameter. The ADMM for problem in (2.6) is derived by alternatively minimizing L_ρ in (2.7) with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$ with the other variables fixed. Specifically, the iterative steps are given by

$$\begin{aligned} \boldsymbol{x}^j &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} L_\rho(\boldsymbol{x},\boldsymbol{y}^{j-1};\boldsymbol{z}^{j-1}) \\ &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) + \frac{\rho}{2}\|\boldsymbol{y}^{j-1} - \boldsymbol{R}\boldsymbol{x} + \frac{\boldsymbol{z}^{j-1}}{\rho}\|^2, \end{aligned} \tag{2.8}$$

$$\begin{aligned} \boldsymbol{y}^j &= \arg\min_{\boldsymbol{y}\leq\boldsymbol{c}} L_\rho(\boldsymbol{x}^j,\boldsymbol{y};\boldsymbol{z}^{j-1}) \\ &= \arg\min_{\boldsymbol{y}\leq\boldsymbol{c}} \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}^j + \frac{\boldsymbol{z}^{j-1}}{\rho}\|^2, \end{aligned} \tag{2.9}$$

where $\mathcal{X} = \{\boldsymbol{x} \mid \boldsymbol{x} \geq 0, \|\boldsymbol{x}_k\|_0 \leq w_k, \forall k\}$, j is the step index, $\boldsymbol{x}^j = (\boldsymbol{x}_1^j;\boldsymbol{x}_2^j;\cdots;\boldsymbol{x}_K^j)$, and $\boldsymbol{x}_k^j = (\boldsymbol{x}_{k,1}^j,\boldsymbol{x}_{k,2}^j,\cdots,\boldsymbol{x}_{k,P_k}^j)^\top$, $k = 1,2,\cdots,K$. After the update steps of $\boldsymbol{x}$ and $\boldsymbol{y}$ as above, the update of the multiplier $\boldsymbol{z}$ is given by

$$\boldsymbol{z}^j = \boldsymbol{z}^{j-1} + \rho(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j). \tag{2.10}$$

The $\boldsymbol{x}$-update subproblem is hard to solve because different blocks of $\boldsymbol{x}$ are coupled together. To circumvent such an issue, we linearize the quadratic term in (2.7) at $\boldsymbol{x}^{j-1}$ and add a proximal term:

$$\begin{aligned}\boldsymbol{x}^j &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) - \rho\langle \boldsymbol{R}^\top(\boldsymbol{y}^{j-1} - \boldsymbol{R}\boldsymbol{x}^{j-1} - \frac{\boldsymbol{z}^{j-1}}{\rho}), \boldsymbol{x} - \boldsymbol{x}^{j-1}\rangle + \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{x}^{j-1}\|_2^2 \\ &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) + \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{x}^{j-1} - \frac{\rho}{\mu}\boldsymbol{R}^\top(\boldsymbol{y}^{j-1} - \boldsymbol{R}\boldsymbol{x}^{j-1} - \frac{\boldsymbol{z}^{j-1}}{\rho})\|_2^2. \end{aligned} \tag{2.11}$$

Now, problem (2.11) can be separated for different sources. Specifically, for the k-th source, the update of $\boldsymbol{x}_k$ is given by

$$\boldsymbol{x}_k^j = \arg\min_{\boldsymbol{x}_k\in\mathcal{X}_k} -U_k(\|\boldsymbol{x}_k\|_1) + \frac{\mu}{2}\|\boldsymbol{x}_k - \boldsymbol{v}_k^{j-1}\|_2^2, \tag{2.12}$$

where $\mathcal{X}_k = \{\boldsymbol{x}_k \mid \boldsymbol{x}_k \geq 0, \|\boldsymbol{x}_k\|_0 \leq w_k\}$, $\boldsymbol{v}^{j-1} = \boldsymbol{x}^{j-1} + \frac{\rho}{\mu}\boldsymbol{R}^\top(\boldsymbol{y}^{j-1} - \boldsymbol{R}\boldsymbol{x}^{j-1} - \frac{\boldsymbol{z}^{j-1}}{\rho})$. Without loss of generality, suppose the elements of $\boldsymbol{v}_k^{j-1} = (v_{k,1}^{j-1}, v_{k,2}^{j-1}, \cdots, v_{k,P_k}^{j-1})^\top$ are in descending order. The solution of problem in (2.12) is given by

$$x_{k,i}^j = \max(0, v_{k,i}^{j-1} + \zeta_k), \text{ where } \mu i'\zeta_k = U_k'(\sum_{i=1}^{i'} \max(0, v_{k,i}^{j-1} + \zeta_k)),$$

i' is the maximal index such that $U_k'(\sum_{i=1}^{i'} \max(0, v_{k,i}^{j-1} - v_{k,i'}^{j-1})) \geq -\mu v_{k,i'}^{j-1}$ and $i' \leq w_k$. For details, one may refer to[7]. Invoking (2.3), ζ_k can be found by solving

$$r_k - \frac{\beta}{\mu}\frac{1}{r_k} - \frac{s_k}{\mu K}\frac{1}{r_k^2} = \frac{\sum_{i=1}^{i'} v_{k,i}^{j-1}}{\mu} \tag{2.13}$$

where $r_k = \sum_{i=1}^{i'}(v_{k,i}^{j-1} + \zeta_k)$. Since $\frac{\beta}{\mu} > 0$, $\frac{s_k}{\mu K} > 0$, equation (2.13) has exactly one positive root that can be found by Cardano's formula[8], see Fig. 2.2. We denote the mapping between

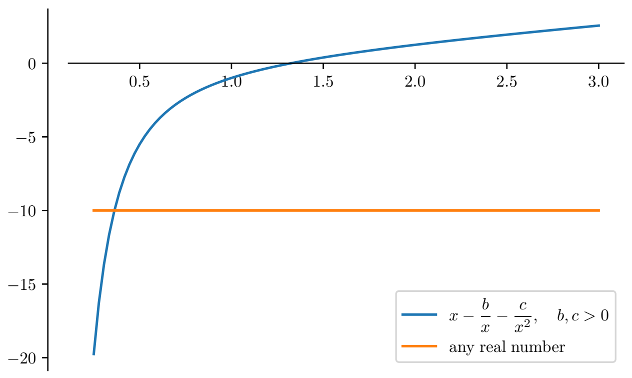

图 2.2　An illustration of finding ζ_k.

v_k^{j-1} and x_k^j as

$$x_k^j = C_k(v_k^{j-1}). \tag{2.14}$$

The update of $\boldsymbol{y}$ writes:

$$\boldsymbol{y}^j = \arg\min_{\boldsymbol{y}\le\boldsymbol{c}} \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}^j\|_2^2 - (z^{j-1})^\top(\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}^j). \tag{2.15}$$

The solution of (2.15) is

$$\boldsymbol{y}^j = -\mathcal{P}_{\mathbb{R}_+^L}(\boldsymbol{c} - \boldsymbol{R}\boldsymbol{x}^j - \frac{z^{j-1}}{\rho}) + \boldsymbol{c},$$

where $\mathcal{P}_{\mathbb{R}_+^L}$ is the Euclidean projection on $\mathbb{R}_+^L = \{(x_1, x_2, \cdots, x_L)^\top \mid x_i \geqslant 0,\ i = 1, 2, \cdots, L\}$. Using the above results, we get the Linearized ADMM for NUM:

$$\begin{cases} \boldsymbol{x}_k^j \leftarrow C_k(\boldsymbol{v}_k^{j-1}), k = 1, \ldots, K, & \text{(2.16a)} \\ \boldsymbol{y}^j \leftarrow -\mathcal{P}_{\mathbb{R}_+^L}(\boldsymbol{c} - \boldsymbol{R}\boldsymbol{x}^j - \frac{z^{j-1}}{\rho}) + \boldsymbol{c}, & \text{(2.16b)} \\ z^j \leftarrow z^{j-1} - \gamma\rho(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j), & \text{(2.16c)} \\ \boldsymbol{v}^j \leftarrow \boldsymbol{x}^j + \frac{\rho}{\mu}\boldsymbol{R}^\top(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j - \frac{z^j}{\rho}), & \text{(2.16d)} \end{cases}$$

where γ is the update coefficient of Lagrangian multiplier.

2.5 Linearized ADMM for MOBAC

We derive the linearized ADMM for MOBAC like the NUM. Introducing the auxiliary variable $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$, problem (2.2) equals to

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{k=1}^{K} -U_k(\|\boldsymbol{x}_k\|_1) + \alpha \max_l \frac{\boldsymbol{y}_l}{c_l} \\ \text{s.t.} \quad & \boldsymbol{y} \le \boldsymbol{R}\boldsymbol{x} \\ & \boldsymbol{y} \le \boldsymbol{c} \\ & \|\boldsymbol{x}_k\|_0 \le w_k, \quad \forall\, k \\ & \boldsymbol{x} \ge \boldsymbol{0}, \end{aligned} \tag{2.17}$$

The augmented Lagrangian function of above problem is

$$L_\rho(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{z}) = -\sum_{k=1}^{K} U_k(\|\boldsymbol{x}_k\|_1) + \alpha \max_l \frac{\boldsymbol{y}_l}{c_l} + \boldsymbol{z}^\top(\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{y} - \boldsymbol{R}\boldsymbol{x}\|_2^2, \tag{2.18}$$

where z is a Lagrangian multiplier vector associated with the constraint $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ in (2.6), and $\rho > 0$ is a penalty parameter.

The $\boldsymbol{x}$-update of linearized ADMM for MOBAC is the same as NUM. We only consider the $\boldsymbol{y}$-update of linearized ADMM for MOBAC. It writes:

$$\begin{aligned}\boldsymbol{y}^{j+1} &= \arg\min_{\boldsymbol{y}\in\mathcal{Y}} L_\rho(\boldsymbol{x}^{j+1},\boldsymbol{y};\boldsymbol{z}^j)\\ &= \arg\min_{\boldsymbol{y}\in\mathcal{Y}} \alpha \max_l \frac{y_l}{c_l} + \frac{\rho}{2}\|\boldsymbol{y}-\boldsymbol{R}\boldsymbol{x}^{j+1}+\frac{\boldsymbol{z}^j}{\rho}\|^2,\end{aligned} \tag{2.19}$$

where $\mathcal{Y} = \{\boldsymbol{y} \mid 0 \le \boldsymbol{y} \le \boldsymbol{c}\}$. It is non-smooth and hard to obtain an explicit solution of problem (2.19). Thus, we introduce a new variable $\boldsymbol{y} = \boldsymbol{R}\boldsymbol{x}$ and consider the following quadratic program:

$$\begin{aligned}\min_{t,\boldsymbol{y}} \quad & \phi(t,\boldsymbol{y}) = \alpha t + \frac{\rho}{2}\|\boldsymbol{y}-\boldsymbol{R}\boldsymbol{x}^{j+1}+\boldsymbol{z}^j/\rho\|^2\\ \text{s.t.} \quad & \boldsymbol{y} \le t\boldsymbol{c}\\ & \boldsymbol{y} \ge \boldsymbol{0}\\ & \boldsymbol{y} \le \boldsymbol{c},\end{aligned} \tag{2.20}$$

which is equivalent to problem (2.19). Define an auxiliary function

$$y_l(t) = \begin{cases} \theta_l^j, & if \quad 0 \le \theta_l^j \le \min(c_l, tc_l),\\ \min(c_l, tc_l), & if \quad \theta_l^j > \min(c_l, tc_l),\\ 0, & otherwise,\end{cases} \tag{2.21}$$

where $\boldsymbol{\theta}^j = \boldsymbol{R}\boldsymbol{x}^{j+1} - \boldsymbol{z}^j/\rho$. Problem (2.20) is equivalent to

$$\min_t \Phi(t) = \phi(t,\boldsymbol{y}(t)) \quad \text{s.t.} \quad 0 \le t \le 1, \text{ where } \boldsymbol{y}(t) = (y_1(t),\cdots,y_L(t))^\top. \tag{2.22}$$

Suppose t^* is the optimal solution of problem (2.22). The solution of problem (2.20) is given by $\boldsymbol{y}(t^*)$. One may refer to[7] for detailed proof. Problem (2.21) can be tackled by optimization methods based on function value (e.g. `fminbnd` in MATLAB).

第三章　ADMM-Net for Network Resource Allocation

Even though the linearized ADMM can solve problem (2.1) with simple iterations and theoretical guarantee. In practice, it usually requires hundreds of iterations to converge, which is not entirely satisfactory for *real-time* decision making even with certain acceleration techniques[9]. In this section, we introduce a novel method named *Algorithm unrolling* to overcome this difficulty.

3.1　Learn to Optimize (L2O)

In recent days, a research field named *learn to optimize* attracts much attention[10]. This approach aims to automate the designation of optimization algorithms. We call an optimization problem as optimizee and an optimization algorithm as optimizer. As illustrated in Fig. 3.1, classical optimizers are designed for a wide range of problems with particular properties. Learn to optimize leverages machine learning to improve the performance of optimizers for similar optimizees. The paradigm of learn to optimize is shown in Fig. 3.2. We train a L2O optimizer offline and test it online. There are three approaches to construct a L2O optimizer: model-free, plug-and-play (PnP), and algorithm unrolling. Model-free methods construct a L2O optimizer from a machine learning model directly. Plug-and-play methods utilize the modular architecture of optimizers (e.g. ADMM) and substitute the module with a learnable model. Algorithm unrolling methods only make some predefined parameters in classical optimizer become learnable, which preserve the original architecture.

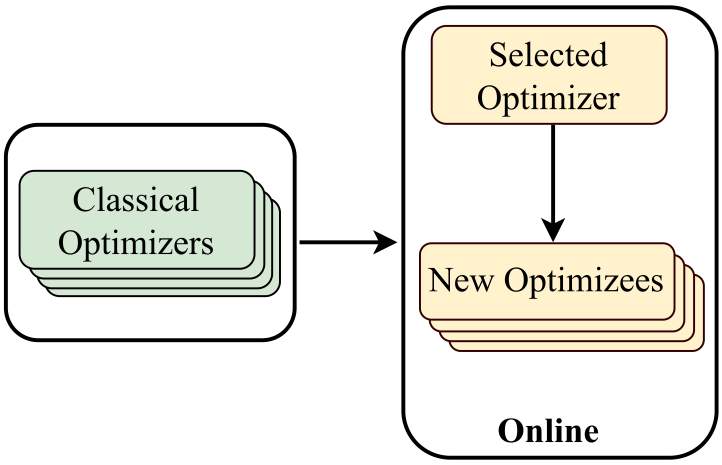

图 3.1　The framework of classical algorithm

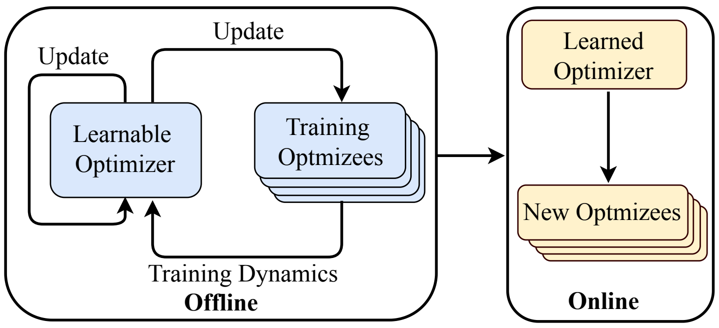

图 3.2 The framework of learn to optimize

3.1.1 Model-free

Gradient-type algorithms share a common formulation[11]:

$$x^{(i)} \leftarrow x^{(i-1)} + \pi(f, \{x^{(0)}, \dots, x^{(i-1)}\}), \tag{3.1}$$

where π is a functional of the objective function and past locations. In vanilla gradient descent method, we choose:

$$\pi(f, \{x^{(0)}, \dots, x^{(i-1)}\}) = -\gamma \Delta f(x^{(i-1)}),$$

where γ is the stepsize. In gradient descent method with momentum, we have:

$$\pi(f, \{x^{(0)}, \dots, x^{(i-1)}\}) = -\gamma(\sum_{j=0}^{i-1} \alpha^{i-1-j} \nabla f(x^{(j)})),$$

where γ is the stepsize and α is the decay coefficient. π is actually a policy based on trajectory and objective function value. Thus, we can model π using reinforcement learning (RL). The simulation results in simple examples show that this idea is quite powerful.

In[12], the authors demonstrate a method that converts the gradient descent algorithm into long short term memory (LSTM) network, also gains much improvement. There are many other examples of the model-free method[13][14][15]. They have a common framework and show advantages in numerical performance. However, on the one hand, them are hard to get theoretical results. On the other hand, most of them are based on gradient-type algorithm. Many problems with objective functions that difficult to get gradient cannot be solved by these methods, several iteration methods have been developed to solve such problems e.g. ADMM[6], iterative shrinkage thresholding algorithm (ISTA) & fast iterative shrinkage thresholding algorithm (FISTA)[16]. These methods find the minima of subproblems directly, do not need to compute gradient, which makes the implementation of learning to optimize that described above become hard.

3.1.2 Plug-and-play (PnP)

ADMM is popular for solving maximum a posteriori (MAP) inverse problems. Given the measurement y of the ground truth x, the MAP estimation is

$$l(x, y) + \beta s(x),$$

where l is the 'distance' between x and y, s is the regularization term, β is the regular coefficient. MAP problem equals to

$$\min_{x,v} \; l(x, y) + \beta s(v), \text{ s.t. } x = v.$$

ADMM for above problem consists of the iterations:

$$\begin{aligned}
x^{j+1} &\leftarrow \arg\min_{x} l(x, y) + \tfrac{\lambda}{2}\|x - x^j\|_2^2 \\
v^{j+1} &\leftarrow \arg\min_{v} \beta s(v) + \tfrac{\lambda}{2}\|v - v^j\|_2^2 \\
u^{j+1} &\leftarrow u^j + (x^j - v^j).
\end{aligned}$$

Treating the v-update as a functional F from β, s, v^j, λ to v^{j+1}, we get a high-level viewpoint of ADMM. In ADMM for image processing problems, the v-update may be viewed as the denoising process. However, there are many existing ad-hoc denoisers, e.g. K-SVD, BM3D, Total Variation (TV). We can substitute the v-update with aforementioned denoisers[17]. The numerical result shows the success of this idea. The modified ADMM outperforms the original ADMM.

The PnP method is not connected to the L2O until the recent work[18-20]. Instead of the manually designed denoisers, they substitute the updates with trainable models. Since PnP owns modular structure, convergence can be guaranteed under some conditions like bounded denoisers[21], monotone operator and constrained Lipschitz constant[22], by using the proof technique in ADMM. Since the learnable parts in PnP is small, the training cost of PnP is usually lower than model-free methods.

3.1.3 Algorithm Unrolling

Iterative algorithms can be viewed as a recurrent neural networks (RNN) with no trainable parameters. A natural idea is making the predefined parameters in iterative algorithms trainable.

In the last decade, a novel technique called *algorithm unrolling* has been developed. In the seminal paper[23], the authors, for the first time, propose the approach that unrolls an iterative algorithm into a deep neural network. It achieved great success by fixing the computational complexity to a quite small size with an acceptable approximation solution. This technique starts a different line of work in *learn to optimize*.

3.1.3.1 LISTA

Consider a sparse coding problem that is classical in source coding, signal reconstruction, pattern recognition and feature selection. There is an unknown sparse vector $x^* = [x_1^*, \cdots, x_M^*]^\top \in \mathbb{R}^M$. We have its noisy linear measurements:

$$b = \sum_{m=1}^{M} d_m x_m^* + \varepsilon = Dx^* + \varepsilon,$$

where $b \in \mathbb{R}^N, D = [d_1, \cdots, d_M] \in \mathbb{R}^{N\times M}$ is the dictionary, and $\varepsilon \in \mathbb{R}^N$ is additive Gaussian white noise. This is an undetermined system with $N \leqq M$. The expensive inference algorithms prohibits it real-time applications. A popular approach is least absolute shrinkage and selection operator (LASSO):

$$\min_x \frac{1}{2}\|b - Dx\|_2^2 + \lambda\|x\|_1, \text{ where } b = Dx^* + \varepsilon.$$

Iterative shrinkage thresholding algorithm (ISTA) is a general solution for LASSO, it performs:

$$x^{k+1} = \eta_{\lambda/L}(x^k + \frac{1}{L}D^\top(b - Dx^k)), \quad k = 0, 1, 2, \ldots,$$

where $\eta_\theta(x) = \text{sign}(x)\max(0, |x| - \theta)$ and L is usually taken as the largest eigenvalue of $D^\top D$, λ is a hyper parameter. Let $W_1 = \frac{1}{L}D^\top$, $W_2 = I - \frac{1}{L}D^\top D, \theta = \frac{1}{L}\lambda$. ISTA can be written as

$$x^{k+1} = \eta_\theta(W_1 b + W_2 x^k).$$

Note the ISTA can be recognized as a RNN with no learnable weights, see Fig. 3.3. We can untie the weights in different steps and make them learnable:

$$x^{k+1} = \eta_{\theta^k}(W_1^k b + W_2^k x^k), \quad k = 1, 2, \cdots, K - 1.$$

We call this model as learned ISTA (LISTA)[23], see Fig. 3.4. LISTA has the trainable weights $\Theta = \{W_1^k, W_2^k, \theta^k\}_{k=1}^K$. Invoking the origin of LISTA, we can make only $W_1 = W = \frac{1}{L}D^\top$ learnable and let $W_2 = I - WD$. Since $N \leqq M$, this approach will reduce the memory significantly:

$$x^{k+1} = \eta_{\theta^k}(x^k + W^k(b - Dx^k)), \quad k = 1, 2, \cdots, K - 1,$$

where the learnable parameters are $\Theta = \{\theta^k, W^k\}_{k=1}^K$. This variant of LISTA is named LISTA with coupling weights (LISTA-CP). The necessary condition for the convergence of LISTA

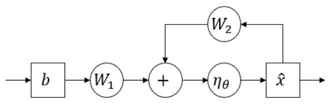

图 3.3　RNN Structure of ISTA

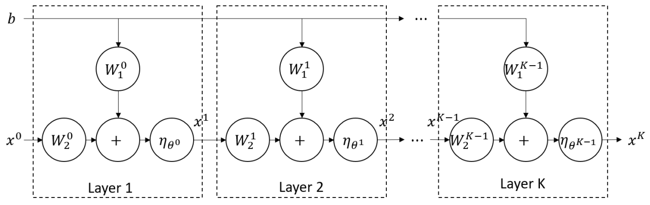

图 3.4　Unrolled Learned ISTA Network

表 3.1　Summary: variants of LISTA and the number of parameters to learn.

LISTA	LISTA-CP	TiLISTA	ALISTA
$O\left(KM^2+K+KMN\right)$	$O(KNM+K)$	$O(NM+K)$	$O(K)$

has been proved[24]:

$$\theta^k \to 0,\ W_2^k - (I - W_1^k D) \to 0,\ \text{as } k \to \infty,$$

which provides the theoretical equivalence of LISTA and LISTA-CP. The further work[25] proposes an analysis that reduces the size of learnable parameters. Based on the LISTA-CP, we can tie the weights in different steps and result in tied LISTA (TiLISTA):

$$x^{k+1} = \eta_{\theta^k}(x^k + \gamma^k W(b - Dx^k)), \quad k = 1, 2, \cdots, K-1$$

with learnable parameters $\Theta = \{\theta^k, \gamma^k\}_{k=1}^K \cup W$. The W actually aims to solve the following problem[25]:

$$\tilde{\mu}(D) = \inf_{\substack{W \in \mathbb{R}^{N\times M} \\ W_{:,i}^\top D_{:,i}=1}} \{ \max_{\substack{i\neq j \\ 1\leq i,j\leq M}} W_{:,i}^\top D_{:,j} \}. \tag{3.2}$$

This is a linear program with a piece-wise linear objective function and linear constraints. Since it is feasible, and

$$0 \leq \tilde{\mu}(D) \leq \max_{\substack{i\neq j \\ 1\leq i,j\leq M}} D_i^\top D_j.$$

$\tilde{\mu}$ is bounded, there exists optimal solution. We denote the solution set of the problem (3.2) as $\mathcal{W}(D) = \left\{W \in \mathbb{R}^{N\times M} : W \text{ attains the infimum}\right\}$. Substituting the W in TiLISTA with a solution $\tilde{W} \in \mathcal{W}(D)$, we get the analytic LISTA (ALISTA):

$$x^{k+1} = \eta_{\theta^k}(x^k + \gamma^k \tilde{W}(b - Dx^k)), \quad k = 1, 2, \cdots, K-1$$

with the only learnable parameters $\Theta = \{\gamma^k, \theta^k\}_{k=1}^K$. For each model, x^K depends on Θ, b, x^0. Denote x^K as $x^K(\Theta, b, x^0)$. Given the distribution of b, x^*, we train the model by solving the optimization problem:

$$\min_{\Theta} \mathbb{E}_{(b,x^*)} \|x^K(\Theta, b, x^0) - x^*\|_2^2.$$

Stochastic gradient descent (SGD) can be applied to solve this minimization problem. The gradient w.r.t. x^K on Θ are obtained with the chain rule. The difficulties of training RNNs make us adopt layer-wise training in practical[26]. Theoretical linear convergence has been proved both for the models. Numerical experiments reveal the fact that the reduction of learnable parameters (see Tab. 3.1) would not influent the convergence rate of the models, see Fig. 3.5.

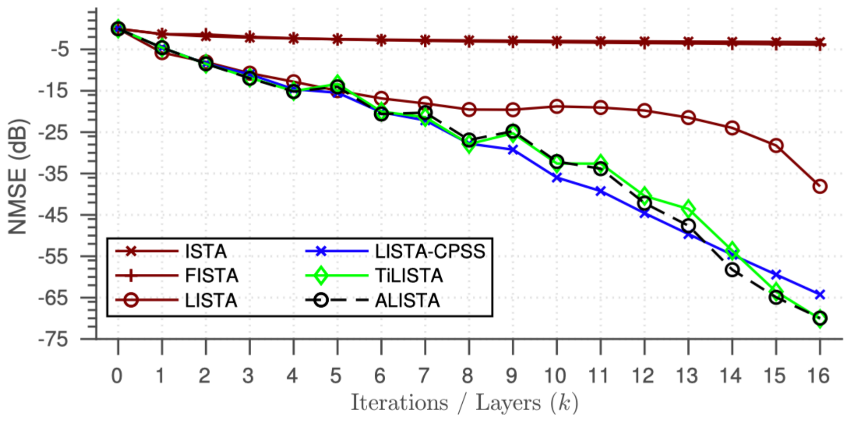

图 3.5 NMSE = $10\log_{10}\left(\frac{\mathbb{E}\|x^K(\Theta)-x^*\|_2}{\mathbb{E}\|x^*\|_2}\right)$

3.1.3.2 ADMM-Net

Inspired by the success of LISTA,[27] first applies the unrolling technique in ADMM in a mixed sparse model. The result indicates that unrolling technique is compatible. Since ADMM is widely adopted in image processing. The researchers have proposed many versions of ADMM-based deep unrolling algorithm.

Consider a compressive sensing magnetic resonance imaging (CS-MRI) problem. Assume $x \in \mathbb{C}^N$ is an MRI image to be reconstructed, $y \in \mathbb{C}^{N'}(N' < N)$ is the under-sampled data. The reconstructed image can be estimated by solving:

$$\hat{x} = \arg\min_{x}\{\frac{1}{2}\|Ax - y\|_2^2 + \sum_{l=1}^{L} \lambda_l g(D_l x)\},$$

where $A = PF \in \mathbb{R}^{N' \times N}$ is a measurement matrix, $P \in \mathbb{R}^{N' \times N}$ is a under-sampling matrix, and F is a Fourier transform. D_l denotes a transform matrix for a filtering operation. $g(\cdot)$ is a regularization function. λ_l is a regularization parameter. Introduce auxiliary variables

$z = \{z_1, z_2, \cdots, z_L\}$, above problem equals to:

$$\min_{x,z} \frac{1}{2}\|Ax - y\|_2^2 + \sum_{l=1}^{L} \lambda_l g(z_l) \quad \text{s.t. } z_l = D_l x, \quad l = 1, 2, \cdots, L.$$

The augmented Lagrangian function is:

$$L_\rho(x, z, \alpha) = \frac{1}{2}\|Ax - y\|_2^2 + \sum_{l=1}^{L} \lambda_l g(z_l) - \sum_{l=1}^{L} \langle \alpha_l, z_l - D_l x \rangle + \sum_{l=1}^{L} \frac{\rho_l}{2}\|z_l - D_l x\|_2^2,$$

where $\alpha = \{\alpha_l\}$ are Lagrangian multipliers and $\rho = \{\rho_l\}$ are penalty parameters.

We alternatively optimizes $\{x, z, \alpha\}$ and substitute $A = PF, \beta_l = \frac{\alpha_l}{\rho_l}$:

$$\begin{cases} x^n = F^\top G^{-1}[P^\top y + \sum_{l=1}^{L} \rho_l F D_l^\top (z_l^{n-1} - \beta_l^{n-1})] \\ z_l^n = S(D_l x^n + \beta_l^{n-1}; \lambda_l / \rho_l) \\ \beta_l^n = \beta_l^{n-1} + \eta_l (D_l x^n - z_l^n) \end{cases}$$

where $G = P^\top P + \sum_{l=1}^{L} \rho_l F D_l^\top D_l F^\top$, $S(\cdot)$ is a nonlinear shrinkage function, η_l is an update rate. But ADMM needs to run dozens of iterations to get a satisfactory result. It is also challenging to choose the transform D_l and shrinkage function $S(\cdot)$ for general regularization function $g(\cdot)$. For different data, tuning the parameters ρ_l and η_l is not trivial. The data flow of ADMM (Fig. 3.6) encourages us to unroll the ADMM and use machine learning to address these issues. Due to the similarity of LISTA and ADMM-Net, we omit the unrolling details of

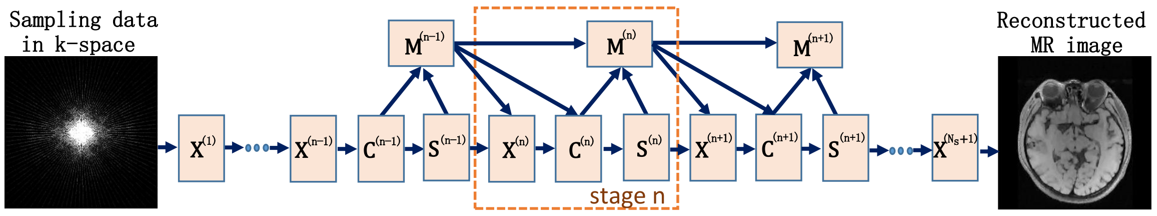

图 3.6　The data flow graph for the ADMM. This graph consists of four types of nodes: reconstruction (**X**), convolution (**C**), non-linear transform (**Z**), and multiplier update (**M**).

ADMM-Net here. The deep ADMM-Net for compressive sensing[28] outperforms the existing methods, this work shows the potential of deep unrolling ADMM, the further work[29] exploits the architecture used to approximate additive layer and gains better performance. The work in[30] present an enlightenment idea that comes from statistical perspective, using a function to composite the terms in proximal term of ISTA. Based on this idea, the authors successfully establish a more general network compare to[29]. It is worth mentioning that the proposed network in[30] outperforms[29] in compressive sensing tasks with a much simpler architecture, which shows the power of the idea. Since linearization tricks often be adopted in ADMM, the deep unrolling version of linearized ADMM (D-LADMM)[31] extends the area of unrolling technique and provides the convergence proof using variational inequality[32].

There are numerous papers aim to exploit the potential of deep unrolling, for more results in image and signal processing, one can refer to[33], this paper reviews the previous notable work and explains the three advantages of deep unrolling—*fast*, *interpretability* and *generalizability*. For more results in communication systems, one may refer to[34] and[35],[35] represents a type of work that apply deep learning (DL) in Weighted Minimize Mean Square Error (WMMSE), however, this work has no guarantee for convergence.

3.2 ADMM-Net for Network Resource Allocation

In this section, we describe the details of unrolling the ADMM for NUM, say equations (2.16), into a deep neural network (DNN). Then, we use the Moreau envelope[36] to derive the backward of $\boldsymbol{y}$-update in MOBAC and give the ADMM-Net for MOBAC. We first realize the Recurrent Neural Network (RNN) structure of ADMM, then unroll the RNN to get ADMM-Net.

RNN structure of ADMM ADMM is an iteration algorithm. As shown in the Fig. 3.7, its data flow graph has a recurrent structure. From the perspective of Deep Learning (DL),

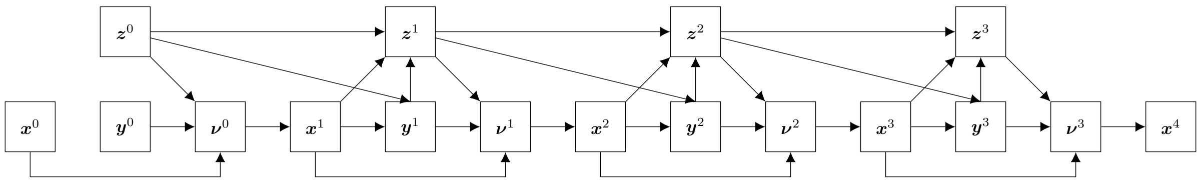

图 3.7 Data flow graph of ADMM, where $\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{z}^0$ are fixed initial values, $\boldsymbol{s}$ is the input.

ADMM is a RNN with non-trainable weights. A natural question that arises here is can we parameterize some constants in (2.16) to get trainable weights?

ADMM-Net1 To get a DNN with trainable weights, we parameterize several matrices in (2.16) to make them learnable, then change the corresponding operations to make them compatible with the weights. Finally, we terminate the process after T-th update.

We use $\odot$, $\oslash$ to represent element wise multiplication and division, respectively. Prewisely, at j-th update, we substitute the first $\boldsymbol{R}$ in (2.16d) with weight $\boldsymbol{W}^j$. To gain more freedom, we substitute the scalar $\frac{1}{\rho}$ with σ^j , where σ is a n-dimension vector. Naturally, we apply element wise operations to replace their corresponding scalar version. We introduce the n-dimension threshold vector $\boldsymbol{t}^j$ and add it to the variable inside proximal mapping of (2.16b) to adjust the threshold of proximal mapping. Adopting the Cardano's formula in (2.16), we get the first version of deep unrolling ADMM:

$$\begin{cases} \boldsymbol{x}_k^j \leftarrow C_k(\boldsymbol{\nu}_k^{j-1}), k = 1, \ldots, K, & (3.3a) \\ \boldsymbol{y}^j \leftarrow -\mathcal{P}_{\mathbb{R}_+^L}(\boldsymbol{R}\boldsymbol{x}^j - \sigma \odot \boldsymbol{z}^{j-1} + \boldsymbol{t}^j) + \boldsymbol{c}, & (3.3b) \\ \boldsymbol{z}^j \leftarrow \boldsymbol{z}^{j-1} - \gamma(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j) \oslash \sigma^j, & (3.3c) \\ \boldsymbol{\nu}^j \leftarrow \boldsymbol{x}^j + \frac{\rho}{\mu}(\boldsymbol{W}^j)^\top(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j - \sigma \odot \boldsymbol{z}^j), & (3.3d) \end{cases}$$

where $\Theta = \{\boldsymbol{W}^j, \sigma^j, \boldsymbol{t}^j\}_{j=1}^T$ are the trainable weights. We use $x^j(\boldsymbol{s}; \{\boldsymbol{W}^\tau, \sigma^\tau, \boldsymbol{t}^\tau\}_{\tau=1}^{j-1})$ to denote the output of ADMM-Net1 at j-th layer. A typical layer of ADMM-Net1 is shown in Fig. 3.8.

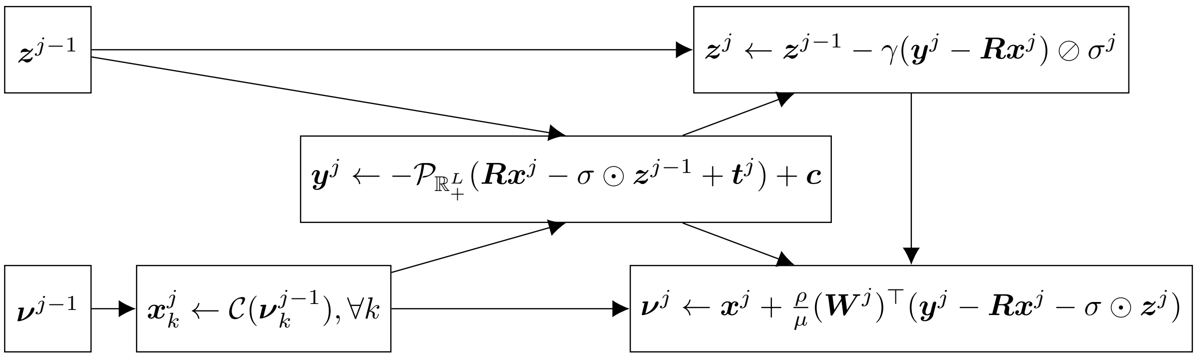
$\boldsymbol{z}^{j-1}$
$\boldsymbol{z}^j \leftarrow \boldsymbol{z}^{j-1} - \gamma(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j) \oslash \sigma^j$
$\boldsymbol{y}^j \leftarrow -\mathcal{P}_{\mathbb{R}_+^L}(\boldsymbol{R}\boldsymbol{x}^j - \sigma \odot \boldsymbol{z}^{j-1} + \boldsymbol{t}^j) + \boldsymbol{c}$
$\boldsymbol{\nu}^{j-1}$
$\boldsymbol{x}_k^j \leftarrow \mathcal{C}(\boldsymbol{\nu}_k^{j-1}), \forall k$
$\boldsymbol{\nu}^j \leftarrow \boldsymbol{x}^j + \frac{\rho}{\mu}(\boldsymbol{W}^j)^\top(\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j - \sigma \odot \boldsymbol{z}^j)$

图 3.8 A typical layer of ADMM-Net.

ADMM-Net2 The Cardano's formula in (2.14) is actually a special case of proximal mapping. Consider the general cubic equation

$$x^3 + ax^2 - bx - c = 0 \Leftrightarrow x - \frac{b}{x} - \frac{c}{x^2} = -a, \tag{3.4}$$

where $b, c > 0$. Fixing b, c, denote the only positive root of (3.4) as $r(a)$. As illustrated in Fig. 2.2, we have

$$r(a) \to 0, \quad \text{as } a \to +\infty,$$
$$r(a) \to -a, \quad \text{as } a \to -\infty.$$

This fact inspires us to use a single branch of the rotated hyperbola to approximate it. Since the asymptotes are fixed, we have the fixed eccentricity. Notice that our aim is approximating C_k in *finite* interval, we use three parameters to control this process, the first one is the shape parameter λ_k, others are translation parameters m_k, n_k corresponding to x axis and y axis, respectively.

$$[(a+m_k)+(y+n_k)](y+n_k)=\lambda_k \Rightarrow y=\sqrt{\frac{(a+m_k)^2}{4}+\lambda_k}-\frac{a+m_k}{2}-n_k. \tag{3.5}$$

The numerical result in Fig. 3.9 indicates that our method is reasonable.

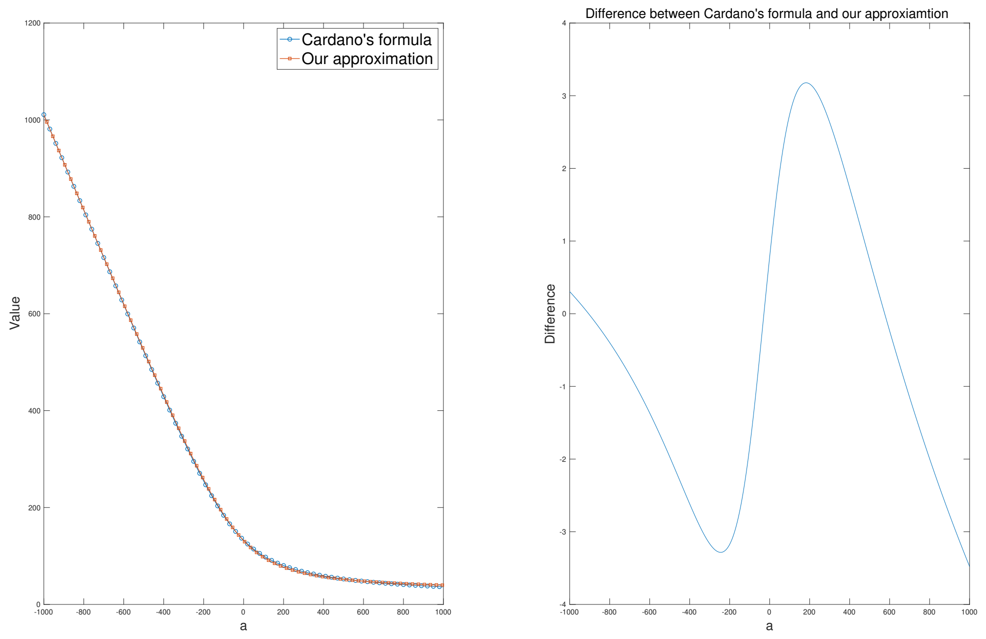

图 3.9　Set $b=-10^4, c=-10^6$, vary the parameter a from -10^3 to 10^3, the only positive root calculated by Cardano's formula and the value of our approximation.

Denote the approximation y as $\mathcal{A}(\cdot;\lambda_k,m_k,n_k)$, where λ_k,m_k,n_k are learnable parameters. Substituting $C_k(\cdot)$ with $\mathcal{A}(\cdot;\lambda_k,m_k,n_k)$ in (3.3a), we get the second version of deep unrolling ADMM (dubbed as ADMM-Net2). A typical layer of ADMM-Net2 is

$$\begin{cases} \boldsymbol{x}_k^j \leftarrow \mathcal{A}(\boldsymbol{v}_k^{j-1};\lambda_k,m_k,n_k), k=1,\dots,K, & (3.6a) \\ \boldsymbol{y}^j \leftarrow -\mathcal{P}_{\mathbb{R}_+^L}(\boldsymbol{R}\boldsymbol{x}^j-\sigma\odot\boldsymbol{z}^{j-1}+\boldsymbol{t}^j)+\boldsymbol{c}, & (3.6b) \\ \boldsymbol{z}^j \leftarrow \boldsymbol{z}^{j-1}-\gamma(\boldsymbol{y}^j-\boldsymbol{R}\boldsymbol{x}^j)\oslash\sigma^j, & (3.6c) \\ \boldsymbol{v}^j \leftarrow \boldsymbol{x}^j+\dfrac{\rho}{\mu}(\boldsymbol{W}^j)^\top(\boldsymbol{y}^j-\boldsymbol{R}\boldsymbol{x}^j-\sigma\odot\boldsymbol{z}^j), & (3.6d) \end{cases}$$

where $\Theta = \{\boldsymbol{W}^j, \sigma^j, \boldsymbol{t}^j\}_{j=1}^T \cup \{\lambda_k, m_k, n_k\}_{k=1}^K$ are the trainable weights. $x^j(\boldsymbol{s}; \{\boldsymbol{W}^\tau, \sigma^\tau, \boldsymbol{t}^\tau\}_{\tau=1}^{j-1} \cup \{\lambda_k, m_k, n_k\}_{k=1}^K)$ is the output of ADMM-Net2 at j-th layer.

ADMM-Net for MOBAC Recall the ADMM-Net for NUM, the output is differentiable with respect to the input in each layer. However, the autograd meets its limit when comes to the y-update of the MOBAC. In the y-update, we get the output by solving a quadratic program, which has no closed form solution. Differentiable optimization[37, 38] provides a method to tackle this problem. But it needs to involve a package in the code. Utilizing the property of the Moreau envelope may give a simpler and lightweight solution.

Given a non-smooth function f, the Moreau envelope is given by

$$f_\mu(x) = \inf_y \{f(y) + \frac{1}{2\mu}\|x - y\|_2^2\}.$$

We note that dom $f_\mu(x) = \mathbb{R}^n$, and that $f_\mu(x)$ is convex. When $f(x) = |x|$, the Moreau envelope is the Huber function:

$$f_\mu(x) = \inf_y \left\{|y| + \frac{1}{2\mu}(x-y)^2\right\} = \begin{cases} \frac{1}{2\mu}x^2, & |x| \le \mu, \\ |x| - \frac{\mu}{2}, & |x| > \mu. \end{cases}$$

$f_\mu(x)$ can be written as

$$\begin{aligned} f_\mu(x) &= \frac{1}{2\mu}\|x\|^2 - \frac{1}{\mu}\sup_y \left\{x^T y - \mu f(y) - \frac{1}{2}\|y\|^2\right\} \\ &= \frac{1}{2\mu}\|x\|^2 - \frac{1}{\mu}\left(\mu f + \frac{1}{2}\|\cdot\|^2\right)^*(x). \end{aligned}$$

Therefore, the derivation of $f_\mu(x)$ is given by

$$\begin{aligned} \nabla f_\mu(x) &= \frac{x}{\mu} - \frac{1}{\mu}\operatorname{argmax}_y \left\{x^T y - \mu f(y) - \frac{1}{2}\|y\|^2\right\} \\ &= \frac{1}{\mu}\left(x - \operatorname{prox}_{\mu f}(x)\right) \end{aligned}$$

In the final step, we use the important point that the gradient of the conjugate function $f^*(x)$ equals to the optimal y^* at which $f^*(x) = \sup_{y\in\operatorname{dom}(f)} x^T y - f(y)$ is achieved.

Substituting $f(y) = \max_l y_l/c_l, \mu = 1/\rho$, the y-update can be written as $f_\mu(\boldsymbol{\theta}^j)$ which is a mapping from $\boldsymbol{\theta}^j = \boldsymbol{R}\boldsymbol{x}^{j+1} - \boldsymbol{z}^j/\rho$ into $\boldsymbol{y}^{j+1}$. We have:

$$\frac{\partial \boldsymbol{y}^{j+1}}{\partial \boldsymbol{\theta}^j} = \nabla f_\mu(\boldsymbol{\theta}^j) = \rho(\boldsymbol{\theta}^j - \boldsymbol{y}^{j+1}).$$

Thus, we can calculate the derivative using the customized backward. The ADMM-Net for MOBAC is the same as the ADMM-Net for NUM except the y-update is replaced by a layer using quadratic program to forward and Moreau envelope to backward.

3.2.1 Training Strategy

Given a flow distribution $\boldsymbol{s}$, we denote the solution by linearized ADMM for NUM or MOBAC as $\boldsymbol{x}^*(s)$. Define the normalized square error between ADMM-Net and $\boldsymbol{x}^*(s)$ as

$$\ell(\boldsymbol{x}^T(\boldsymbol{s};\Theta),\boldsymbol{x}^*(\boldsymbol{s})) = \frac{\|\boldsymbol{x}^T(\boldsymbol{s};\Theta) - \boldsymbol{x}^*(\boldsymbol{s})\|_2}{\|\boldsymbol{x}^*(\boldsymbol{s})\|_2},$$

where the version of the ADMM-Net is determined by Θ, T is the number of layers. The loss function is defined as:

$$\mathcal{L}(\Theta) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{x}^*)\sim\Gamma}\ell(\boldsymbol{x}^T(\boldsymbol{s};\Theta),\boldsymbol{x}^*(\boldsymbol{s})) + \lambda r(\Theta), \tag{3.7}$$

where Γ is the training set, r is the regular term for parameters, λ is the regular coefficient. In our implementation, we choose r as ℓ_1 norm to guarantee the sparsity of the weights.

第四章 Numerical Experiment

In this section, we perform the numerical simulation to illustrate the performance gain of our two versions of deep unfolding ADMM. We first consider a baseline given by classical linearized ADMM, then compare **ADMM-Net1** and **ADMM-Net2** in various standards.

4.1 Data set Descriptions

Small Example In order to get ground-truth solution, we set

- number of links: $L = 5$,
- number of flows: $K = 6$,
- capacity of links: $\boldsymbol{c} = (1024, 10240, 10240, 40960, 102400)^\top$,
- the number of available paths: $P = [1, 1, 1, 1, 1, 1]^\top$,
- the constraint of available paths: $W = [1, 1, 1, 1, 1, 1]^\top$,
- routing matrix:

$$\boldsymbol{R} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Large Example In this scenario, the classical ADMM only give suboptimal solution, thus the training strategy has systematic error. We set

- number of links: $L = 460$,
- number of flows: $K = 561$,
- capacity of links: the distribution of $\boldsymbol{c}$ is given in Fig. 4.1
- P and W are pre defined in the code according to the real world records.
- routing matrix: we generate $\boldsymbol{R}$ using `sprand(L, K)` in MATLAB, then set the nonzero elements to 1.

The flow size is generated from gaussian distribution in two examples, which means $s \sim \mathcal{N}(\delta, \Sigma)$, where $\delta \in \mathbb{R}^K$ is the mean value vector and $\Sigma \in \mathbb{R}^{K\times K}$ is the correlation matrix. In our setting, $\delta = \mathbf{0}_K, \Sigma = I_{K\times K}$, where I is identical matrix.

We first sample a flow size s from $\mathcal{N}(\mathbf{0}_K, I_{K\times K})$, then run classic ADMM to get ground-

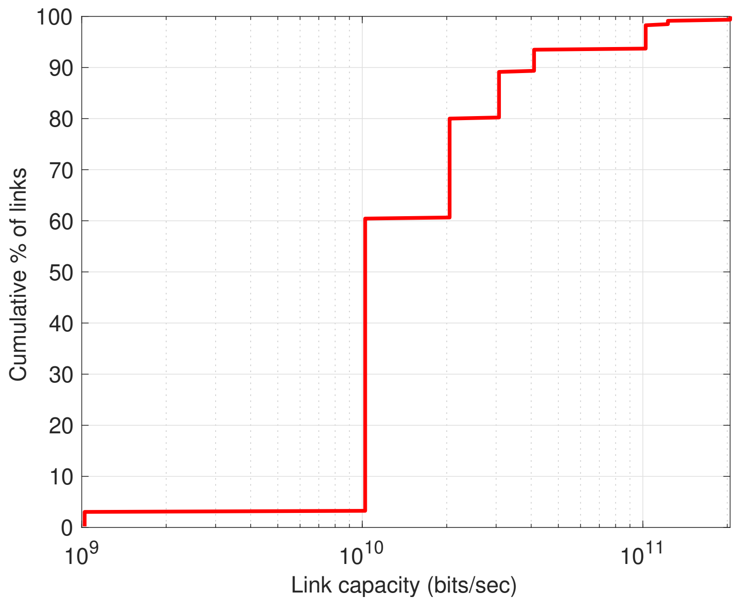

图 4.1 The cumulative distribution function of link capacity.

truth solution $\boldsymbol{x}^{gt}$, repeat this process $n_{\text{train}} + n_{\text{test}}$ times, where n_{train} is the number of pairs contained in training set and n_{test} is the number of pairs contained in testing set.

4.1.1 Parameters Setting

In this section, we compare **ADMM-Net1** and **ADMM-Net2** in several performance measures. Our simulation is performed in MATLAB on a PC with an Intel Core i7 at 2.3GHz and 16GB of memory. For the parameters in our objective function, β is set to be 0.05. In the classical ADMM, we adjust the parameter ρ according to the Section 3.4 .1 in[6], and set the parameter $\mu = 1.1 \times \rho \|\boldsymbol{R}\|_2^2$. In the z -update, we take an additional step length with $\gamma = 1.618$, which demonstrates better convergence performance. We set the maximal iteration number to 20000, and adopt following stopping criteria:

- primal residual:

$$\|\boldsymbol{y}^j - \boldsymbol{R}\boldsymbol{x}^j\|_2 \leq \sqrt{L}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|\boldsymbol{y}^j\|_2, \|\boldsymbol{R}\boldsymbol{x}^j\|_2\},$$

- dual residual:

$$\|\rho \boldsymbol{R}^\top (\boldsymbol{y}^j - \boldsymbol{y}^{j-1})\|_2 \leq \sqrt{P}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \|\boldsymbol{R}^\top \boldsymbol{z}^j\|_2,$$

- constraint violation:

$$\|\max(\boldsymbol{R}\boldsymbol{x} - \boldsymbol{c}, \boldsymbol{0})\|_2 / \max(\sqrt{L}, \|\boldsymbol{c}\|_2) \leq \epsilon^{\text{tol}},$$

where we set $\epsilon^{\text{abs}} = 10^{-6}, \epsilon^{\text{rel}} = 10^{-6}$, and $\epsilon^{\text{tol}} = 10^{-10}$. We utilize

- the averaged normalized root mean square error (NRMSE), which indicates the loss between output and ground-truth solution,

$$E(\Theta) = \frac{1}{|\Gamma|} \sum_{(s, \boldsymbol{x}^{gt}) \in \Gamma} \frac{\|\hat{\boldsymbol{x}}(s, \Theta) - \boldsymbol{x}^{gt}\|_2}{\|\boldsymbol{x}^{gt}\|_2},$$

  where Γ denotes testing set.
- the total completion time, namely delay

$$\text{delay} = \sum_{k=1}^{K} \frac{s_k}{\|\boldsymbol{x}_k\|_1},$$

- the proportional fairness,

$$\text{fairness} = \beta \sum_{k=1}^{K} \log(\|\boldsymbol{x}_k\|_1),$$

- the maximal (say the worst-case) link utilization ratio,

$$\text{load} = \max_{l} \frac{\boldsymbol{R}[l]\boldsymbol{x}}{\boldsymbol{c}_l},$$

- the objective function value in NUM problem,

$$\text{obj} = \text{total} - \text{fairness},$$

as performance measures, all of them are averaged over testing set in practice.

4.1.2 Performance Comparison

表 4.1 CLASSICAL ADMM VS DEEP UNROLLING ADMM IN SMALL EXAMPLE.

method	loss	obj	delay	fairness	load	iteration/layers
ADMM	0	-0.619	1.944	2.563	1.00	3207
ADMM-Net1	0.026	-2.389	0.298	2.687	31.45	1
ADMM-Net2	0.072	0.645	3.230	2.585	1.00	1
ADMM-Net1	0.022	-2.358	0.328	2.686	35.73	3
ADMM-Net2	0.074	0.589	3.179	2.590	1.05	3

表 4.2 CLASSICAL ADMM VS DEEP UNROLLING ADMM IN LARGE EXAMPLE.

method	loss	obj	delay	fairness	load	iteration/layers
ADMM	0	-183.355	4.377	187.732	1.00	20000
ADMM-Net1	0.152	-185.756	5.218	190.802	3.344	2
ADMM-Net2	0.249	-164.463	24.378	188.840	1.01	2
ADMM-Net1	0.128	-185.920	4.898	190.818	3.573	3
ADMM-Net2	0.248	-164.400	24.286	188.891	1.01	3

Our nets reduce the computational complexity significantly. In both examples, we attain the approximate solution in with only 3 layers, which is equivalent to 3 iterations in classical ADMM. However, the main drawback is that we can not beats the classical ADMM in all performance measures at one model. ADMM-Net1 gives a load larger than 1. ADMM-Net2 is too conservative and give a higher delay than the classical ADMM. Our future work is constructing a new model to address this issue.

Another interesting observation is that ADMM-Net could give a fast approximate solution, but if we want to get a precise result, which means converge to the solution derived by traditional methods, the architecture of network become untrainable. However, when we pay attention to the convergence process of ADMM, see Fig 4.2. We find the distance between iterative solution and true solution decrease sharply. Besides, the approximate solution produced by ADMM-Net acctually attain the relative error of 10^{-2}, which is the point that the curve in Fig 4.2 begin to decrease sharply.

Above observations implies an idea: Can we treat the ADMM-Net as a good warm-start solver and concatenate it with ADMM? The answer is yes, we generate a new network by mixing these methods, the convergence process is in Fig 4.2. It is quite stimulating, we successfully combine the convergence feature of ADMM and fast approximation of ADMM-Net.

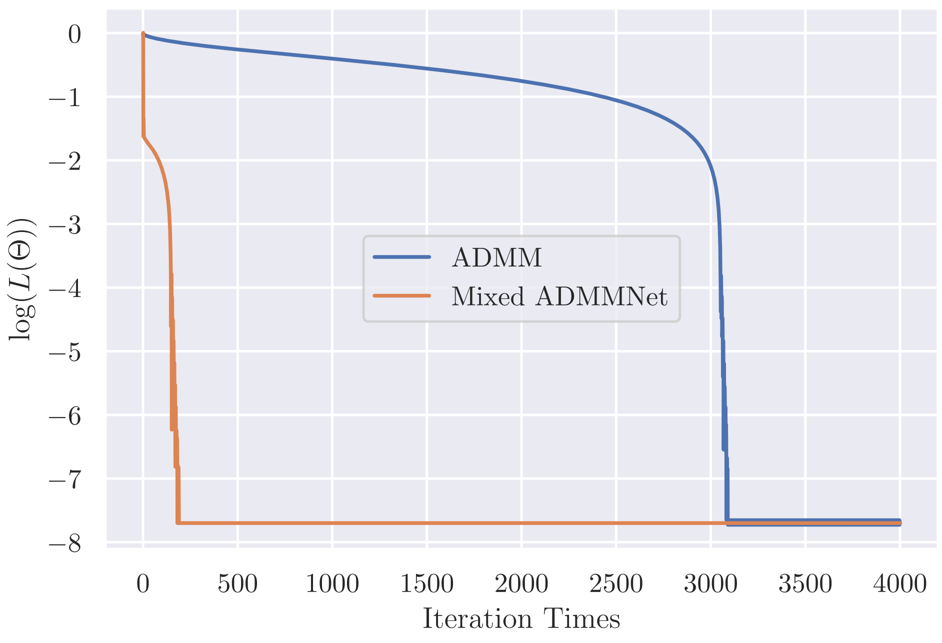

图 4.2 The convergence process of mixed ADMM-Net, where $L(\Theta) = \frac{1}{|\Gamma|}\sum \ell(f(s;\Theta), \boldsymbol{x}^{\text{gt}}(s))$.

第五章　Conclusions

In this paper, we give several methods for network resource allocation. The linearized ADMM has the simplest form. But it requires dozens of iterations. This drawback prohibits the application of linearized ADMM. We want to accelerate the linearized ADMM to meets real-time needs. An acceleration method is utilizing learn to optimize. We introduce the three paradigms in L2O. The most suitable method for linearized ADMM is algorithm unrolling. Thus, we construct two ADMM-Nets based on the linearized ADMM. The numerical performance indicates the success of the unrolling. However, the theoretical perspective of ADMM-Nets is still not clear. Unlike the LISTA, we do not understand the learned weights in our nets and how to reduce the memory further. Besides theory, the numerical performance is also can be improved. For instance, can we combine the two ADMM-Nets to reduce both delay and load in one model? Can we utilize the Plug-and-Play to construct an implicit utility function and train it in different datasets?

参考文献

[1] KIMBALL J, WYPYCH T, KUESTER F. Low Bandwidth Desktop and Video Streaming for Collaborative Tiled Display Environments[J/OL]. Future Generation Computer Systems, 2016, 54: 336-343 [2021-05-28]. https://linkinghub.elsevier.com/retrieve/pii/S0167739X15002320. DOI: 10/f77zx7.

[2] LEE K, CHU D, CUERVO E, et al. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming[C]//Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services. [S.l. : s.n.], 2015: 151-165. DOI: 10/gg3hbn.

[3] ELBAMBY M S, PERFECTO C, BENNIS M, et al. Toward Low-Latency and Ultra-Reliable Virtual Reality[J]. IEEE Network, 2018, 32(2): 78-84. DOI: 10/ggr96d.

[4] KELLY F. Charging and Rate Control for Elastic Traffic[J/OL]. European Transactions on Telecommunications, 1997, 8(1): 33-37 [2021-04-14]. https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4460080106. DOI: 10.1002/ett.4460080106.

[5] KELLY F P, MAULLOO A K, TAN D K H. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability[J/OL]. Journal of the Operational Research Society, 1998, 49(3): 237-252 [2021-05-28]. https://www.tandfonline.com/doi/full/10.1057/palgrave.jors.2600523. DOI: 10/c8kvzc.

[6] BOYD S. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers[J/OL]. Foundations and Trendső in Machine Learning, 2010, 3(1): 1-122 [2020-08-07]. http://www.nowpublishers.com/article/Details/MAL-016. DOI: 10.1561/2200000016.

[7] WANG J, ZHANG F, XIE Z, et al. Joint Bandwidth Allocation and Path Selection in WANs with Path Cardinality Constraints[EB/OL]. (2020-08-10) [2020-10-31]. http://arxiv.org/abs/2008.03942. arXiv: 2008.03942 [eess].

[8] Cubic Equation. //Wikipedia. [S.l. : s.n.], 2021 [2021-04-18]. https://en.wikipedia.org/w/index.php?title=Cubic_equation&oldid=1018308997.

[9] ZHANG J, PENG Y, OUYANG W, et al. Accelerating ADMM for Efficient Simulation and Optimization[EB/OL]. (2019-09-01) [2020-08-07]. http://arxiv.org/abs/1909.00470. arXiv: 1909.00470 [cs, math].

[10] CHEN T, CHEN X, CHEN W, et al. Learning to Optimize: A Primer and A Benchmark[EB/OL]. (2021-03-23) [2021-03-26]. http://arxiv.org/abs/2103.12828. arXiv: 2103.12828 [cs, math, stat].

[11] LI K, MALIK J. Learning to Optimize[EB/OL]. (2016-06-06) [2020-08-07]. http://arxiv.org/abs/1606.01885. arXiv: 1606.01885 [cs, math, stat].

[12] ANDRYCHOWICZ M, DENIL M, GOMEZ S, et al. Learning to Learn by Gradient Descent by Gradient Descent[EB/OL]. (2016-11-30) [2020-08-07]. http://arxiv.org/abs/1606.04474. arXiv: 1606.04474 [cs].

[13] CHEN Y, HOFFMAN M W, COLMENAREJO S G, et al. Learning to Learn without Gradient Descent by Gradient Descent[C/OL]//International Conference on Machine Learning. [S.l.]: PMLR, 2017: 748-756 [2021-03-27]. http://proceedings.mlr.press/v70/chen17e.html.

[14] CAO Y, CHEN T, WANG Z, et al. Learning to Optimize in Swarms[EB/OL]. (2019-11-16) [2021-05-14]. http://arxiv.org/abs/1911.03787. arXiv: 1911.03787 [cs, q-bio, stat].

[15] LI K, MALIK J. Learning to Optimize Neural Nets[EB/OL]. (2017-11-30) [2020-08-07]. http://arxiv.org/abs/1703.00441. arXiv: 1703.00441 [cs, math, stat].

[16] BECK A, TEBOULLE M. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems[J/OL]. SIAM Journal on Imaging Sciences, 2009, 2(1): 183-202 [2020-08-09]. http://epubs.siam.org/doi/10.1137/080716542. DOI: 10.1137/080716542.

[17] VENKATAKRISHNAN S V, BOUMAN C A, WOHLBERG B. Plug-and-Play Priors for Model Based Reconstruction[C]//2013 IEEE Global Conference on Signal and Information Processing. [S.l. : s.n.], 2013: 945-948. DOI: 10.1109/GlobalSIP.2013.6737048.

[18] MEINHARDT T, MOLLER M, HAZIRBAS C, et al. Learning Proximal Operators: Using Denoising Networks for Regularizing Inverse Imaging Problems[C/OL]//. [S.l. : s.n.], 2017: 1781-1790 [2021-04-01]. https://openaccess.thecvf.com/content_iccv_2017/html/Meinhardt_Learning_Proximal_Operators_ICCV_2017_paper.html.

[19] RICK CHANG J H, LI C L, POCZOS B, et al. One Network to Solve Them All – Solving Linear Inverse Problems Using Deep Projection Models[C/OL]//. [S.l. : s.n.], 2017: 5888-5897 [2021-04-01]. https://openaccess.thecvf.com/content_iccv_2017/html/Chang_One_Network_to_ICCV_2017_paper.html.

[20] ZHANG K, ZUO W, GU S, et al. Learning Deep CNN Denoiser Prior for Image Restoration[C/OL]//. [S.l. : s.n.], 2017: 3929-3938 [2021-04-01]. https://openaccess.thecvf.com/content_cvpr_2017/html/Zhang_Learning_Deep_CNN_CVPR_2017_paper.html.

[21] CHAN S H, WANG X, ELGENDY O A. Plug-and-Play ADMM for Image Restoration: Fixed-Point Convergence and Applications[J]. IEEE Transactions on Computational Imaging, 2017, 3(1): 84-98. DOI: 10.1109/TCI.2016.2629286.

[22] TERRIS M, REPETTI A, PESQUET J C, et al. ENHANCED CONVERGENT PNP ALGORITHMS FOR IMAGE RESTORATION[J]., 6.

[23] GREGOR K, LECUN Y. Learning fast approximations of sparse coding[C/OL]//ICML 2010 - Proceedings, 27th International Conference on Machine Learning. [S.l. : s.n.], 2010: 399-406 [2020-08-07]. https://nyuscholars.nyu.edu/en/publications/learning-fast-approximations-of-sparse-coding.

[24] CHEN X, LIU J, WANG Z, et al. Theoretical Linear Convergence of Unfolded ISTA and Its Practical Weights and Thresholds[EB/OL]. (2018-11-03) [2021-04-11]. http://arxiv.org/abs/1808.10038. arXiv: 1808.10038 [cs, stat].

[25] LIU J, CHEN X, WANG Z, et al. ALISTA: ANALYTIC WEIGHTS ARE AS GOOD AS LEARNED WEIGHTS IN LISTA[J]., 2019: 33.

[26] BENGIO Y, LAMBLIN P, POPOVICI D, et al. Greedy Layer-Wise Training of Deep Networks[M]. [S.l. : s.n.], 2007.

[27] SPRECHMANN P, LITMAN R, YAKAR T B, et al. Supervised Sparse Analysis and Synthesis Operators[J]., 9.

[28] YANG Y, SUN J, LI H, et al. Deep ADMM-Net for Compressive Sensing MRI[C]//NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc., 2016: 10-18.

[29] YANG Y, SUN J, LI H, et al. ADMM-CSNet: A Deep Learning Approach for Image Compressive Sensing[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 42(3): 521-538(2020-03-01) [2020-08-07]. https://ieeexplore.ieee.org/document/8550778/. DOI: 10/ghbx9q.

[30] ZHANG J, GHANEM B. ISTA-Net: Interpretable Optimization-Inspired Deep Network for Image Compressive Sensing[C/OL]//2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT: IEEE, 2018: 1828-1837 [2020-08-07]. https://ieeexplore.ieee.org/document/8578294/. DOI: 10.1109/CVPR.2018.00196.

[31] XIE X, WU J, ZHONG Z, et al. Differentiable Linearized ADMM[EB/OL]. (2019-05-15) [2020-08-07]. http://arxiv.org/abs/1905.06179. arXiv: 1905.06179 [cs, stat].

[32] HE B, YUAN X. On the $O(1/n)$ Convergence Rate of the Douglas–Rachford Alternating Direction Method[J/OL]. SIAM Journal on Numerical Analysis, 2012, 50(2): 700-709 [2021-05-28]. http://epubs.siam.org/doi/10.1137/110836936. DOI: 10/ggmmsw.

[33] MONGA V, LI Y, ELDAR Y C. Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing[EB/OL]. (2020-07-09) [2020-08-06]. http://arxiv.org/abs/1912.10557. arXiv: 1912.10557 [cs, eess].

[34] BALATSOUKAS-STIMMING A, STUDER C. Deep Unfolding for Communications Systems: A Survey and Some New Directions[EB/OL]. (2019-10-08) [2020-08-07]. http://arxiv.org/abs/1906.05774. arXiv: 1906.05774 [cs, eess, math].

[35] SUN H, CHEN X, SHI Q, et al. Learning to Optimize: Training Deep Neural Networks for Interference Management[J]. IEEE TRANSACTIONS ON SIGNAL PROCESSING, 2018, 66(20): 16.

[36] JOURANI A, THIBAULT L, ZAGRODNY D. Differential Properties of the Moreau Envelope[J/OL]. Journal of Functional Analysis, 2014, 266(3): 1185-1237 [2021-05-28]. https://linkinghub.elsevier.com/retrieve/pii/S0022123613004424. DOI: 10/f5p7cm.

[37] AGRAWAL A, AMOS B, BARRATT S, et al. Differentiable Convex Optimization Layers[G/OL]// WALLACH H, LAROCHELLE H, BEYGELZIMER A, et al. Advances in Neural Information Processing Systems 32. [S.l.]: Curran Associates, Inc., 2019: 9562-9574 [2020-08-07]. http://papers.nips.cc/paper/9152-differentiable-convex-optimization-layers.pdf.

[38] AMOS B, KOLTER J Z. OPTNET: Differentiable Optimization as a Layer in Neural Networks[EB/OL]. 4th version. (2019-10-14) [2020-08-07]. http://arxiv.org/abs/1703.00443. arXiv: 1703.00443 [cs, math, stat].

致谢

本科四年倏忽而过, 转眼便到了给毕业论文写致谢的时候. 离别的气息尚未弥漫燕园, 夏日的蝉鸣却已经催人起程.

首先感谢我的导师文再文副教授, 是他带领我领略了优化研究的美妙. 在他悉心的指导下, 我才得以建立起对研究工作的理性认知. 也是在他的言传身教下, 我意识到不管是论文的写作, 轻重缓急各不相同的任务如何安排, 还是 slides 的制作, 都有可以琢磨的地方. 他的高标准也让我窥见了一名卓越的学者对待自己的研究该有的态度, 让我时刻提醒自己, 不能轻易放松要求.

其次感谢我的室友们, 他们是马佳磊, 付瑞昊, 熊景洋. 很幸运能在 45 乙 211 这个" 二十一世纪一流寝室" 相遇, 他们的乐观开朗与善解人意让寝室成为了温馨的港湾, 不管是每晚的卧谈会, 还是周末的火锅局, 我都能感受到敞开心扉交谈的美好.

感谢在树洞遇见的群友们, 他们分别是马佳磊, 王宇萱, 曾楚原, 李润珩, 朱哲毅, 谈忆萱, 康佳楠, 王彤, 游方宜. 我们相识于大一的失眠夜晚, 在四年的时间里一起成长, 分享了青春岁月里的种种喜悦与忧愁, 是他们的陪伴给了我完整的大学生活.

感谢李欣怡, 她陪伴我走过了低谷, 教会了我该怎样对待生活, 怎样爱自己和爱身边的人, 这或许是大学最重要的一课.

感谢我的父亲, 他锤炼了我的品格. 感谢我的母亲, 是她无私的爱和无条件的信任支撑我走到今天, 让我敢于做出一个又一个不寻常的选择.

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　　　　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因须要延迟发布学位论文电子版，授权学校在 □ 一年 / □ 两年 / □ 三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　　　　　导师签名：文再文　日期：　　年　　月　　日