Shiraz University

M.Sc. Thesis

Distributed Multi-Task Learning

Author:
Mahsa Asadi

Supervisor:
Dr. Sattar Hashemi
Dr. Haitham Bou Ammar

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Computer Science, Engineering and IT department
School of Electrical and Computer Engineering

March 2016

Declaration of Authorship

I, Mahsa Asadi, declare that this thesis titled, "Distributed Multi-Task Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Mahsa Asadi

Date: March 2016

"The jungle does not know it's countless dimensions yet!"

Sohrab Sepehri

SHIRAZ UNIVERSITY

Abstract

Artificial Intelligence group
School of Electrical and Computer Engineering

Master of Science

Distributed Multi-Task Learning

by Mahsa Asadi

There exists lots of methods solving Multi-Task Learning(MTL) problems all of which have been centralized so far. On the other hand, taking advantage from the knowledge of many tasks to overcome the difficulty of solving MTL problems seems to be a reasonable approach. In this thesis we have introduced a distributed framework so as to reduce the computational time and amount of memory required for the system to solve the MTL problem for the first time. Using this framework, we would be able to obtain a better performance by employing more tasks without loss of knowledge. The method we are going to introduce here has a convergence rate of $O(\frac{1}{K})$, where K is the number of iterations to reach the optimal solution, which is the best considering current distributed optimization algorithms.

Acknowledgements

I have to thank my supervisor, Dr. Sattar Hashemi for his support throughout these years at Shiraz University and for helping me overcome problems that encountered during my thesis. I certainly could not do this research without his guidance and supervision.
I would like to thank Dr. Haitham Bou Ammar from Princeton University for helping me and guiding me throughout this research and to make this collaboration work. He though me not only science, but also was a guide to be a better human. His role in this research is truly inevitable.
And special thanks to Dr. Ali Hamzeh as my advisor for listening to my questions thoroughly and answering them so as to help me continue my research more confidently.
I would like to thank my friends who have been [1]"better than the passing river" and I would never forget how they have been there for me in every situation.
Lastly, many thanks to my dear mother and father for supporting me all these years without whom I could not be the person I am and be where I am standing now.

[1] A Poem by Sohrab Sepehri

Contents

List of Figures

List of Tables

List of Abbreviations

ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
MTL	**M**ulti-**T**ask **L**earning
DMTL	**D**istributed framework for **M**ulti-**T**ask **L**earning
ADMM	**A**lternating **D**irectional **M**ethod of **M**ultipliers
DADMM	**D**istributed **A**lternating **D**irectional **M**ethod of **M**ultipliers
GOMTL	**G**rouping and **O**verlap in **M**ulti-**T**ask **L**earning
ELLA	**E**fficient **L**ifelong **L**earning **A**lgorithm
MDP	**M**arkov **D**ecision **P**rocess

To my great mother, kind father and wonderful brother...

Chapter 1

Introduction

Multi-Task learning(MTL) is one of Machine Learning(ML) approaches that solves a number of learning tasks simultaneously by using task commonalities and task differences so as to obtain a more efficient and accurate model. This approach is appropriate especially when we have few samples for each of the problems 1.2. Thus, taking advantage from sharing of knowledge by learning the models jointly can help us obtain more robust models.
In this chapter, we are going to:

- introduce the "MTL problem",
- discuss the drawback of MTL,
- talk about the motivation of our proposed approach,
- introduce the prerequisites needed for the rest of the chapters and
- define the structure of this document.

1.1 Multi-Task Learning

MTL is a form of knowledge reuse which learns a number of problems jointly with the goal of improving the performance of learning algorithms. For instance, you may want to learn models for a number of related tasks such as the ones shown in figure 1.1. You should be learning models to hold your phone, hold a clock, hold your notebook, hold a laptop, etc. together as one person.
Obviously, the tasks should be related in order to share knowledge. As you have seen, the concept behind all the tasks in figure 1.1 is holding and all the tasks share the information required for holding an object.
One of the approaches in MTL is to use a ***shared repository*** in order to model the relationship between tasks and one of the existing methods to model a shared repository is by using a shared space of d latent task parameters. We are going to explain more about this approach in the next chapter.
The convections we are using throughout this document is such that we would represents matrices with bold uppercase letters, vectors with bold lowercase letters and scalars with normal letters. For instance, $\mathbf{X}, \mathbf{x}, and\alpha$ represent a matrix, vector and scalar respectively.
Parenthetical subscripts represents task related quantities(e.g., matrix $\mathbf{A}^{(t)}$ and vector $\mathbf{a}^{(t)}$ are related to task t).

FIGURE 1.1: Learning a number of tasks simultaneously in everyday life

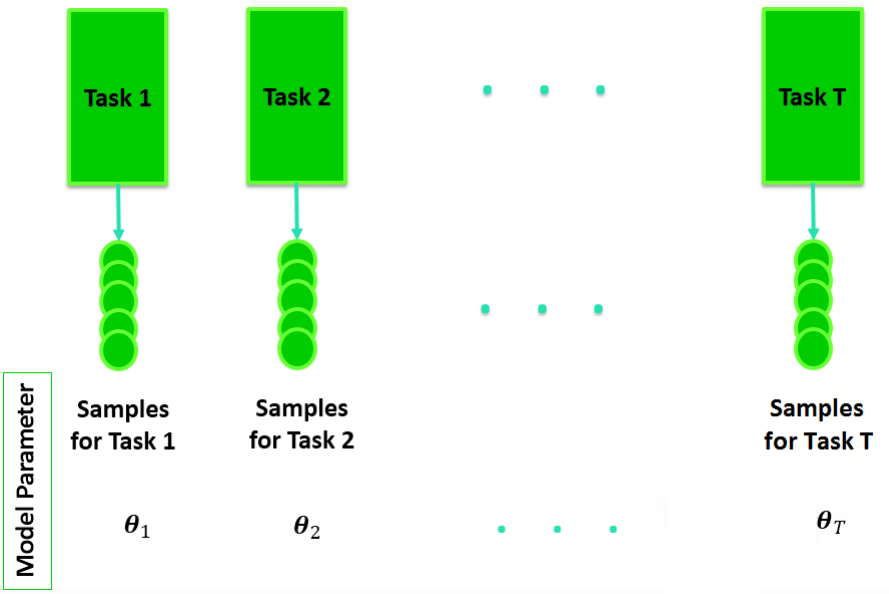

FIGURE 1.2: Multiple tasks with their parameters

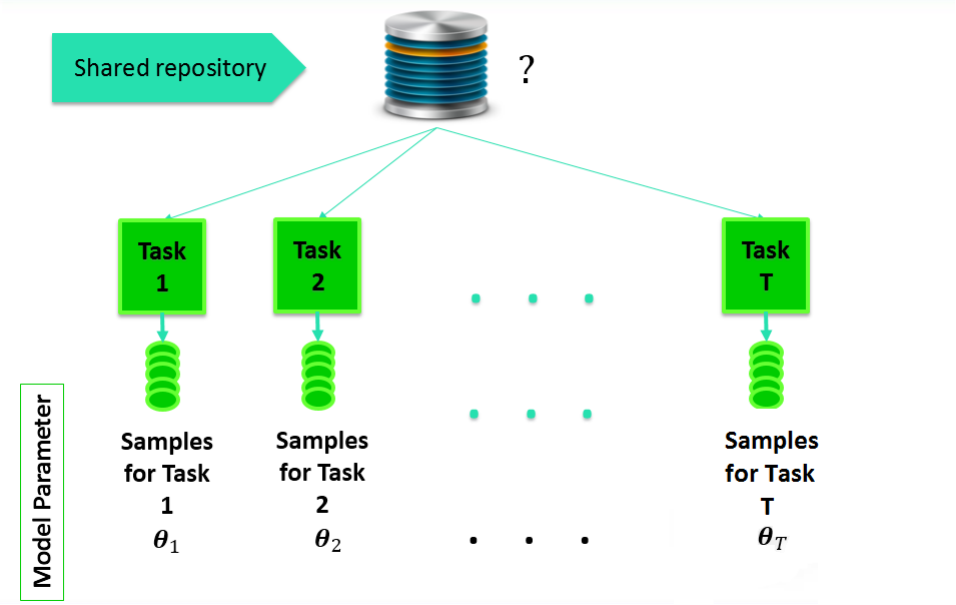

FIGURE 1.3: Learning a general model using a shared repository

1.2 Formal Definition

Consider T tasks numbered 1 through T, with parameter vectors:

$$\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T)}\}$$

By considering n to be the maximum among all task feature sizes, we can assume that all tasks have n features while concatenating zeros to the end of the ones with less than n features.
Therefore, we can consider n to be the number of each task's features without loss of generality; thus $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^{n \times 1}$.
In our MTL formulation, each of $\boldsymbol{\theta}^{(i)}$s are calculated using a linear combination of d latent components of a shared repository, $\mathbf{L} \in \mathbb{R}^{n \times d}$, as shown in figure 1.3.
Thus, $\boldsymbol{\theta}^{(t)}$ can be decomposed into a matrix $\mathbf{L} \in \mathbb{R}^{n \times d}$ and a weight vector $\mathbf{s}^{(t)} \in \mathbb{R}^{d \times 1}$ and be written it as $\mathbf{L}\mathbf{s}^{(t)}$.
We may say that each column of matrix $\mathbf{L}$, represents an abstract task with dimension n and each of the T task parameters can be represented by a weighting of these d abstract tasks.
Each of the T columns of Matrix $\mathbf{S}$ represents the weight, $\mathbf{s}^{(t)}$, for each of the tasks. Hence, $\mathbf{s}^{(t)}$ is task t's weight for each of the d components. So, in this formulation each of the task models use the shared repository's knowledge.
The main objective is to minimize the predictive loss over all tasks using the models with shared repository and It is defined as:

$$\min_{\mathbf{L},\mathbf{S}} e_T(\mathbf{L}, \mathbf{S}) \tag{1.1}$$

As we have divided each of the model parameters into a shared repository and a task specific weighting part, $\mathbf{s}^{(t)}$s are independent. Gaining advantage from the

shared repository model of $\mathbf{L}\mathbf{s}^{(t)}$ and the fact that $\mathbf{s}^{(t)}$s are independent, the main objective would be written as:

$$= \min_{\mathbf{L},\mathbf{S}} \frac{1}{T} \sum_{t=1}^{T} \Big\{ \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}\big(f\big(\mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)}\big), y_i^{(t)}\big) + \mu_1 ||\mathbf{s}^{(t)}||_1 \Big\} + \mu_2 ||\mathbf{L}||_F^2 \tag{1.2}$$

In the coming chapters we will explain more about the existing methods and our proposed idea.

1.3 Motivation

As mentioned before, we have few samples for each of the tasks. Therefore, we can not find a robust parameter using each task's few data. One of the approaches to enhance the prediction performance of each task would be to gain advantage from the shared knowledge and one idea is to use a shared repository. There exists lots of methods which use shared repository and we are going to explain them in more detail in chapter 2. However, one possible idea to learn a better shared repository is by increasing the number of tasks we are going to use. To learn a complex model with many features, we would need a large number of tasks which would increase the time and space required to obtain the result. Therefore, we are going to propose a distributed framework to increase the learning speed and reduce space requirements.

1.4 Machine Learning Problems

Machine learning(ML) problems are divided into three dominant categories: Supervised, Unsupervised and Reinforcement Learning approaches.
In this section, we are going to explain two instances from the first and last group of Machine Learning problems. The first one is the "Regression problem" and the second one is the "Policy Gradient Reinforcement Learning problem". In this research we are going to use these two problems to demonstrate the effectiveness of our proposed approach in MTL. Thus, we are going to solve these problems while taking advantage from MTL.

1.4.1 Regression Problem

Regression problem is one of supervised learning type problems **bishop2006pattern** Given n_t training samples, $\mathbf{X}^{(t)} \in \mathbb{R}^{n_t \times n}$, and their corresponding labels, $\mathbf{y}^{(t)} \in \mathbb{R}^{n_t \times 1}$, for task t, the goal is to predict the correct label $y \in \mathbb{R}$ for a new instance $\mathbf{x} \in \mathbb{R}^{n \times 1}$. Assuming a function, $f(\mathbf{x}) : \mathbb{R}^{n \times 1} \to \mathbb{R}$, to be the model of the problem, we have to estimate model parameters by introducing a loss function, $\mathcal{L}(f(\mathbf{x}), y)$.
A common choice of loss function for regression is squared loss given by:

$$\mathcal{L}(f(\mathbf{x}; \boldsymbol{\theta}), y) = \{y - f(\mathbf{x}; \boldsymbol{\theta})\}^2$$

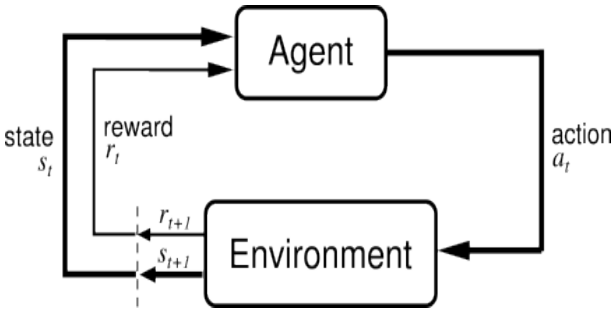

FIGURE 1.4: Sequential Decision Making Problem

Therefore, the objective function would be to minimize the loss function over all instances:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n_t} \mathcal{L}(f(\mathbf{x}_i^{(t)}; \boldsymbol{\theta}), y_i^{(t)}) = \min_{\boldsymbol{\theta}} \sum_{i=1}^{n_t} \{y_i^{(t)} - f(\mathbf{x}_i^{(t)}; \boldsymbol{\theta})\}^2$$

An example of regression problem is predicting the house prices. The problem is to predict the value of a house price given features like it's area, number of rooms, the house's location, etc. .

1.4.2 Reinforcement Learning

People always make decisions that depend on each other. In other words, their choices are based on the sequence of some states and actions that are taken place in past. The mathematical model for sequential decision making is a Markov Decision Process(MDP).

As described in **puterman2014markov** and **szepesvari2010algorithms** Consider an agent observing a state at a specified point of time. It makes a decision based on it's current state and receives an instant reward while the environment changes to a new state. This process can be demonstrated in figure 1.4.

A MDP is a triplet $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P})$ where it's elements are defined as:

- $\mathcal{X}$: countable nonempty set of environment **states**,
- $\mathcal{A}$: countable nonempty set of environment **actions**,
- $\mathcal{P}(x, a, y) : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \mapsto \mathbb{R}$: The **state transition probability function** which for every triplet (x, a, y) gives the probability of moving from state x to y provided that action a has been chosen in state x,
- $r(x, a) : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$: **immediate reward function** which gives the reward of choosing action a in state x for every state-action pair (x, a).

Each reinforcement learning(RL) task can be modeled by a Markov Decision Process. RL is a type of ML problems that uses some sampling approach in which an agent tries to find MDP's solution by employing simulation. In other words, RL is learning of what to do - how to map situations to actions - so as to maximize a numerical reward signal by learning sequential actions. To be more precise:

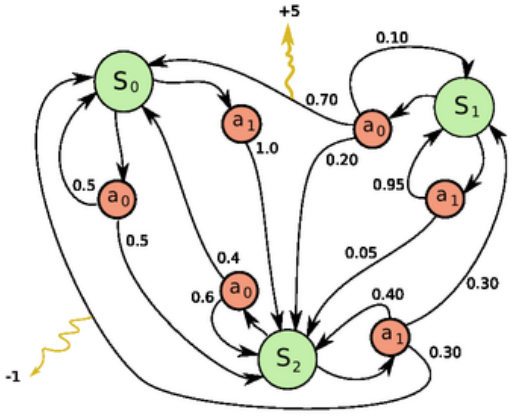

FIGURE 1.5: Example of a simple MDP with three states and two actions - Wikipedia

- τ : A **trajectory** is a sequence of state,action, rewards, $s_1, a_1, r_1, s_2, a_2, r_2, \ldots s_n, a_n, r_n, s_n+1$, till the agent reaches the terminal state which also can be referred to as behaviour.
- $R(\tau) \mapsto \Re$: **Return function** of a trajectory which is the discounted sum of rewards for a given trajectory:

$$R(\tau) = \sum_{i=1}^{n} \gamma^i * r(s_i, s_{i+1}, a_i) \tag{1.3}$$

- γ: discount factor

Figure 1.5 is an example of a RL task represented by a three-state and two-action MDP. This method has become more and more popular due to it's ability to perform well even on complex real world problems.
A RL problem can be solved using either "Direct" or "Indirect" learning approaches. "Value-based Learning" approaches are among Indirect approaches while we are going to investigate Policy Gradient RL(PGRL) methods that are among the Direct approaches.

1.4.3 Policy Gradient Reinforcement Learning

To build up an intuition, consider a continuous state-action environment. If we choose a parameter as policy of the agent and define the RL problem with an optimization problem in terms of the policy parameter, we could solve the problem by searching the policy space.
The idea to use policy search methods for episodic reinforcement learning tasks and it's relation to policy gradient methods is introduced in **kober2009policy** and we are going to investigate this approach here in brief.
What we are searching for is the policy parameter to maximize the cumulative reward of the agent in long term. So, let's introduce a policy $\pi(a_i|x_i)$ which is a

probability distribution with parameter $\theta \in \mathbb{R}^n$ given state at time t. The agent does action a_i and reaches the state x_{i+1} while receiving reward r_i. As we are focusing on episodic RL tasks with episodic restarts, we use n to represent the length of each trajectory.
Since the general goal in reinforcement learning is maximizing the expected return of the agent, we can write objective function for our optimization problem as below:

$$\max_{\theta} J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \int_{\tau} p_{\boldsymbol{\theta}}(\tau) R(\tau) d\tau \tag{1.4}$$

where

$$p_{\boldsymbol{\theta}}(\tau) = p_0(x_1) \prod_{h=1}^{H} p(\mathbf{x}_{h+1}|\mathbf{x}_h, a_h) \pi_{\boldsymbol{\theta}}(a_h|\mathbf{x}_h) \tag{1.5}$$

and $p_0(x_1)$ denotes the initial state distribution, $p(x_{i+1}|x_i, a_i)$ the probability of reaching the next state x_{i+1}, given the current state x_i, and action a_i while R_τ represents the expected return of the given trajectory. For now, let's assume that $\boldsymbol{\theta}$ is our current parameter and $\boldsymbol{\theta}'$ to be the parameter we are searching for. By taking the logarithm of $J(\boldsymbol{\theta}')$, we can reach the blow equations as:

$$\log J(\boldsymbol{\theta}) = \log \int_{\tau} p_{\boldsymbol{\theta}}(\tau) R(\tau) d\tau$$

$$= \log \int_{\tau} p_{\boldsymbol{\theta}}(\tau) R(\tau) \frac{p_{\boldsymbol{\theta}'}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} d\tau \tag{1.6}$$

Since the logarithm function is concave, equation 1.6 leads to the below equation using Jensen's inequality:

$$\log \int_{\tau} p_{\boldsymbol{\theta}}(\tau) R(\tau) \frac{p_{\boldsymbol{\theta}'}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} d\tau \geq \int_{\tau} p_{\boldsymbol{\theta}}(\tau) R(\tau) \log \frac{p_{\boldsymbol{\theta}'}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} d\tau + \text{const.} \tag{1.7}$$

which is proportional to:

$$-D(p_{\boldsymbol{\theta}}(\tau) R(\tau) || p_{\boldsymbol{\theta}'}(\tau)) = Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}')$$

where,

$$D(p(\tau)||q(\tau)) = \int p(\tau) \log \frac{p(\tau)}{q(\tau)} \tag{1.8}$$

defines the Kullback-Leibler divergence which is introduced by **kullback1951information** Taking the logarithm of $p(\tau)$ which is defined in 1.5 leads to:

$$\log p_{\boldsymbol{\theta}}(\tau) = \log p_0(x_1) + \sum_{h=1}^{H} \log p(\mathbf{x}_{h+1}|\mathbf{x}_h, a_h) + \sum_{h=1}^{H} \log \pi_{\boldsymbol{\theta}}(a_h|\mathbf{x}_h) \tag{1.9}$$

Next, we have to compute $\log \pi_{\boldsymbol{\theta}}(a_h|\mathbf{x}_h)$ and by assuming that our policy is a Gaussian function, we will reach:

$$\log \pi_{\boldsymbol{\theta}}(a_h|\mathbf{x}_h) = \frac{-1}{2}\Big[\big(\Sigma^{\frac{-1}{2}}[\mathbf{a}_h - \mathbf{X}_h\boldsymbol{\theta}]\big)^T\big(\Sigma^{\frac{-1}{2}}[\mathbf{a}_h - \mathbf{X}_h\boldsymbol{\theta}]\big)\Big] \tag{1.10}$$

Using 1.7, 1.9 and 1.10, we want to write the optimization equation. Considering that we have n trajectories for the current task,

$$\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^{n}\big\{R(\tau_i) * (\mathbf{a}^{(i)} - \mathbf{X}^{(i)}\boldsymbol{\theta})^T\Sigma^{-1}(\mathbf{a}^{(i)} - \mathbf{X}^{(i)}\boldsymbol{\theta})\big\} \tag{1.11}$$

Where $\mathbf{a}^{(i)}$ represents a vector consisting of i^{th} trajectory's corresponding actions and $\mathbf{X}^{(i)}$ represents their corresponding states.
Obviously, the objective function for RL looks like the one for weighted regression problem. These two problems would be used in 4 as the problems we are going to use to solve using MTL.

1.5 Structure of this thesis

This thesis includes six chapters. After the "Introduction" chapter, the previous approaches of MTL are investigated in the second chapter.
In the third chapter, we would get familiar with one of the famous distributed and parallel computing approaches to solve an optimization problem.
The proposed method and all the details revolving our work will be presented in chapter four and we would report the results and analyze our proposed approach using the experimental results in the fifth chapter.
Finally, in the sixth chapter we would conclude and suggest some of the possible future extensions for the presented research.
Moreover, some of the mathematical proofs are included in appendices.

Chapter 2

Brief History of MTL Approaches

As explained by Salvucci in the psychological publication of **MultiTaskingMind** the story of Multi-task learning problems goes back to 1890 when James discussed human attention as "several simultaneously possible train of thoughts" - figure 2.1. Not only in psychology, MTL have also been introduced in computer science from long time ago. We are going to introduce some of the previous approaches in this chapter and use some of them to compare our model with.

2.1 Related Work

2.1.1 First steps through Multitask Learning

One of the first attempts on MTL goes back to 1993. In the article of **caruana1997multitask** it is described as an inductive transfer that increases the performance of a number of related tasks by taking advantage from generalization and use of general knowledge.
One of the approaches used in this work is to measure task relatedness using back propagation networks as can be seen in figure 2.2. This publication demonstrates that using extra tasks can improve the performance and also clarifies how MTL can enhance generalization.

2.1.2 Regularized MTL

In 2004 when regularized methods were becoming popular, regularized MTL has been introduced in **RegMTL** article. It is an extension of previously existed regularization based learning methods for single task to a MTL problems.

FIGURE 2.1: Train of thought

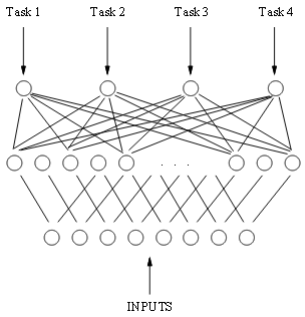

FIGURE 2.2: Multitask backpropagation of four tasks with the same inputs(from **caruana1997multitask**)

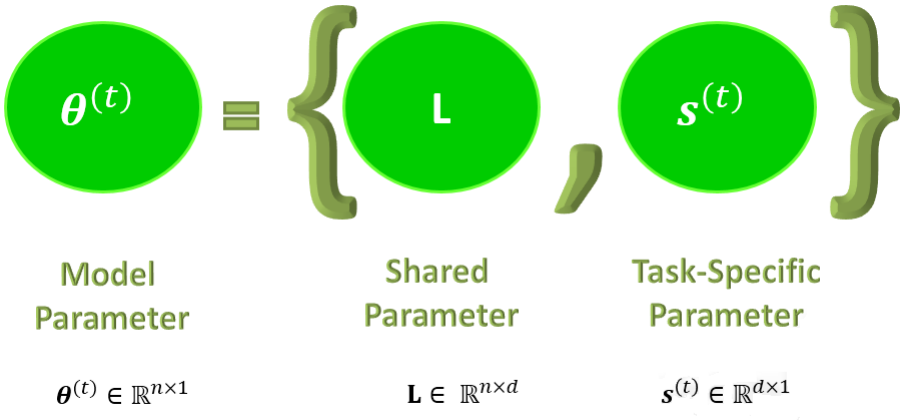

FIGURE 2.3: Task specific and general part of the model parameter

Evgeniou and Pontil representing each task's parameter by $\mathbf{w}^{(t)}$ and assume that:

$$\mathbf{w}^{(t)} = \mathbf{w}_0 + \mathbf{v}^{(t)}$$

where $\mathbf{v}^{(t)}$ is small when tasks are related and true models are close to $\mathbf{w}_0$ and try to estimate $\mathbf{w}_0$ and $\mathbf{v}^{(t)}$ simultaneously.
By dividing the problem's parameter into two general and task-specific parts and taking advantage from regularization of parameters, this article has analyzed the influence of each part.
In otherwords, by considering the two extremes of having a fully-general or fully-task-specified models. Other models can be found by adjusting the weight of each of these two parts.

2.1.3 Learning Task Grouping and Overlap in Multi-Task Learning(GOMTL)

In 2012, Kumar et al. used an alternating approach to solve the introduced problem of 1.2 equation.
In their article **kumar2012learning** the discriminating part suggested is to use a

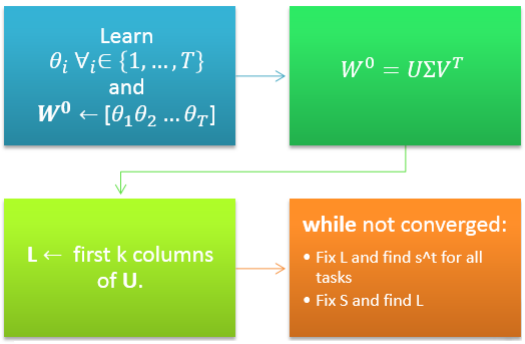

FIGURE 2.4: GOMTL approach's flowchart

sparsity penalty that enforces each task to be sparse enough to be able to be reconstructed by few number of the abstract tasks. Thus, this approach results in producing almost same task specific parameters for related tasks.
Moreover, this approach uses Singular Value Decomposition(SVD) for initialization which would reduce the convergence time.
To be more concrete, this method which is called GOMTL does the below steps to reach the solution:

1. It first solves each task independently and it's corresponding learned parameter is found using only it's own samples.

2. Matrix $\mathbf{W}$ is introduced while each of it's columns should be replaced by one of the task parameters.

3. The initial value of $\mathbf{L}$, which is the shared repository in this publication, would be set to the d most discriminating basis vectors found by SVD of matrix $\mathbf{W}$.

4. Having initialized $\mathbf{L}$, an iterative alternating approach would be used to find the values of $\mathbf{s}^t \forall t \in 1, \ldots, T$ and $\mathbf{L}$.

You can find the whole procedure in the flowchart of picture 2.4.
Although this approach finds the exact solution of the problem, it's time complexity is high. Therefore, the next approaches have been proposed to decrease the time requirement of solving regularized MTL problems.

2.1.4 An Efficient Lifelong Learning Algorithm(ELLA)

The work of **ELLA** proposed an online solution for mutli-task learning as in figure 2.5. This method is called ELLA and it approximates the value of 1.2 using Taylor expansion around each task's parameter.
ELLA approach also uses an alternating method to solve the problem. The advantages of this method compared to other previous approaches is the online formulation of the problem which decreases the computational complexity.
In addition, this approach enforces two approximation steps which increases the convergence speed. While the found solution of this approach is close to the exact answer, one drawback of the algorithm is that it does not reach the optimum value

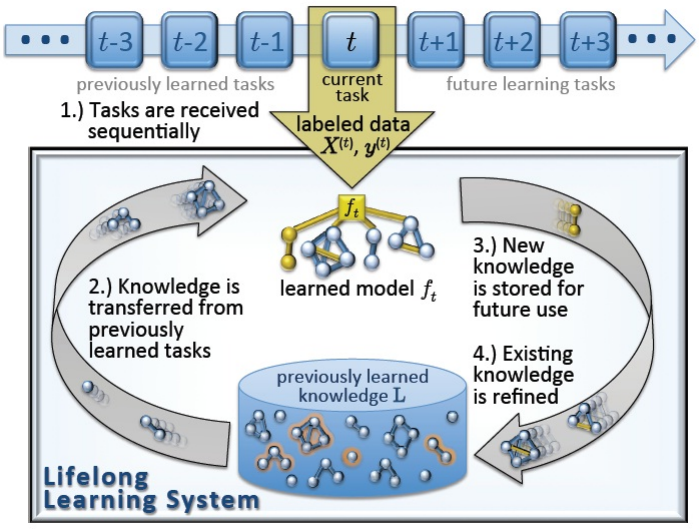

FIGURE 2.5: An online method for MTL

for all problems.
The first approximation assumption takes place when using taylor expansion and only using the first and second derivatives. Thus, this method would not work for problems that are not polynomial or with power of higher than two.
The second approximation is encountered when the online procedure is introduced. To be concrete, we have to update $\mathbf{s}^t \forall t \in 1, \ldots, T$ when $\mathbf{L}$ is updated; however, this article only updates the value of $\mathbf{s}^t$ only for the recent task.
There exists lots of other methods presented in the literature of MTL such as: **micchelli2004kernels pentina2016active zhong2016flexible gupta2016new jacob2009clustered pontilmulti jalali2010dirty** which demonstrates the importance of MTL throughout the years.

2.2 Conclusion of previous approaches

We have presented some of the previous methods in the literature of MTL. It can be seen that all the methods presented so far are centralized. They are trying to use approximation and learn the task models and not any of them has taken advantage from a distributed approach. Considering the increasing amount of data and information, the need for processing the data with higher speed and lower complexity has arisen and the need for distributed computing is obvious. Therefore, we are going to propose a distributed framework to solve MTL problems.

Chapter 3

Parallel Computing and Distributed Framework

Parallel Computing is a category of methods for which lots of calculations is done by splitting the task into a number of subtasks and doing the computation for each part separately. The results should be combined and the final outcome is computed in the last resort.
In this chapter, we are going to focus on a distributed approach to solve an optimization problem which would be our main tool in the proceeding chapters.

3.1 Method of Multipliers

Methods of multipliers started by Hestenes and Powell in 1969 **hestenes1969multiplier** . Afterwards, in 1976, Alternating Directional Method of Multipliers(ADMM) was introduced by Gabay et al in **gabay1976dual**
This approach tries to solve an optimization problem consisting of two local cost functions. It tries to find a global solution using an equality constraint on local solutions which guides the algorithm to reach a global optimum in a robust way.
There also exists other approaches in this context such as the survey of **boyd2011distributed** on the ADMM approach and analysis of the work.
Afterward, an extension of this approach was introduced by Wei and Ozdaglar **DADMM** which is called Distributed Alternating Directional Method of Multipliers(DADMM). DADMM tries to solve the optimization problem in a distributed fashion using N local cost functions while each agent's parameter is a to-be-learned scalar. We are going to explore the extended form of this approach in more detail next.

3.2 Distributed Alternating Directional Method of Multipliers

In the DADMM approach a network of agents try to solve a global unconstrained optimization problem cooperatively where the objective function is the sum of the local privately known objective function of each agent. While **DADMM** has considered each agent to have a scalar variable as the local parameter, we have extended the approach to vector form local parameters.

3.2.1 Formulation

DADMM has concentrated on solving an optimization problem consisting of N different local cost functions with N processors. Each of these N processors tries to find it's local solution while satisfying the equality constraint on each model's

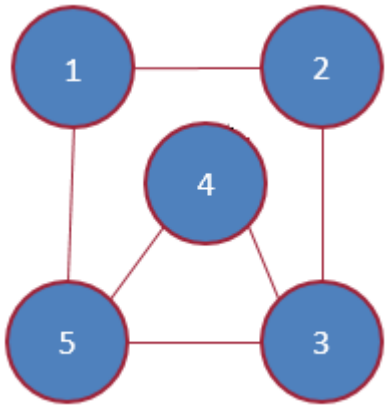

FIGURE 3.1: Five processors' communication links

obtained response so as to reach a global solution.
You may see an example of how five different processors interact in figure 3.1. So, we consider having a network such as 3.1, represented by a graph G = V, E with N nodes and M edges where V symbolizes the set of nodes and E refers to the set of undirected edges.
Each node has its own cost function $f_i : \mathcal{R}^d \rightarrow \mathcal{R}$ which does not have to be differentiable. Therefore, as explained before, our main objective is the sum of the local objectives defined as:

$$\min_{\boldsymbol{\theta} \in \mathcal{R}^d} \sum_{i=1}^{N} f_i(\boldsymbol{\theta})$$

Now, consider that each agent has its own privately known local variable. To find the global solution, we have to assure that the local variables are equal. Therefore, the below objective would equal the previous one:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}_i) \tag{3.1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_1 = \cdots = \boldsymbol{\theta}_N$$

Where $\boldsymbol{\theta} \in \mathcal{R}^{N \times d}$ and $\boldsymbol{\theta}_i \in \mathcal{R}^d \times 1$ for all i. Therefore, $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]^T$.
On the other hand, consider a matrix $\mathbf{A}$, having M rows, each row representing an equality constraint and N columns corresponding to the $\boldsymbol{\theta_i}$s.
All the elements of each row are zero except for two equality constraint variables. For instance, if we want to show $\boldsymbol{\theta}_i - \boldsymbol{\theta}_j = \mathbf{0}$, the i^{th} element should be 1 and the j^{th} element should be -1.
Employing this rule, we can represent processor connections with a directed graph. For each edge, we consider the node with positive multiplier as output and the node with negative weight as input. Consequently, each node would have a set of successors and predecessors.
It is worthed mentioning that if we want all parameters to reach a global solution,

the graph we are using should be a tree.
Using the introduced definitions above, the matrix of figure 3.1 would be:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

In addition, we define a function F as:

$$F(\boldsymbol{\theta}) = \sum_{i=1}^{N} f_i(\boldsymbol{\theta}_i) \tag{3.2}$$

Using matrix $\mathbf{A}$ and relation 3.2, equation 3.1 could be rewritten as:

$$\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}_i) \tag{3.3}$$

$$\text{s.t.} \quad \mathbf{A}\boldsymbol{\theta} = 0$$

Equation 3.3 shows the mathematical formulation of the introduced problem which is well-known to be General Consensus Problem(GCP). In other words, GCP is the problem where we want to reach a global solution using a set of local solutions while these local ones should agree with each other in the last resort.
This approach has two main assumptions:

- **Convexity** of all local cost functions.
- Existence of **Saddle Point** for the Lagrangian function of the global problem 3.1.

The Lagrangian function for the vector form DADMM approach would be:

$$L(\boldsymbol{\theta}, \boldsymbol{\lambda}) = F(\boldsymbol{\theta}) - \boldsymbol{\lambda}^T \text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big) \tag{3.4}$$

where $L : \mathcal{R}^{N\times d} \times \mathcal{R}^{Md\times 1} \to \mathcal{R}$, $\boldsymbol{\lambda} \in \mathcal{R}^{Md\times 1}$ is the vector of Lagrange multipliers and $\boldsymbol{\lambda}_{ij} \in \mathcal{R}^{d\times 1}$ refers to the Lagrange multiplier vector corresponding to edge (i, j).
This method uses the Augmented Lagrangian approach by augmenting a penalty them to the the Lagrangian function as defined below:

$$L(\boldsymbol{\theta}, \boldsymbol{\lambda})_\beta = F(\boldsymbol{\theta}) - \boldsymbol{\lambda}^T \text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big) + \frac{\beta}{2}\Big(\text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big)\Big)^T \text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big)$$

$$= F(\boldsymbol{\theta}) + \frac{\beta}{2}\Big(\text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big) - \frac{\boldsymbol{\lambda}}{\beta}\Big)^T \Big(\text{vec}\big([\mathbf{A}\boldsymbol{\theta}]^T\big) - \frac{\boldsymbol{\lambda}}{\beta}\Big) - \frac{\boldsymbol{\lambda}^T\boldsymbol{\lambda}}{2\beta} \tag{3.5}$$

Algorithm 1 Distributed Alternating Directions Method of Multipliers

Require: the Adjacency matrix **A**

1: Initialization: Choose some arbitrary $\boldsymbol{\theta}_i^0$ in $\mathcal{R}$ for i = 1 ... N which are not necessarily all equal;

2: **while** k ≥ 0 **do**

3: 1. each agent i updates it's estimate L_i^k in a sequential order with:

4:

$$\boldsymbol{\theta}_i^{k+1} = \arg\min_{\boldsymbol{\theta}_i} \Big\{ f_i(\boldsymbol{\theta}_i)$$

$$+\frac{\beta}{2} \sum_{\boldsymbol{\theta}_j \in \text{Pred}(\boldsymbol{\theta}_i)} ||\boldsymbol{\theta}_j^{k+1} - \boldsymbol{\theta}_i - \frac{1}{\beta}\boldsymbol{\lambda}_{ji}^k||^2$$

$$+\frac{\beta}{2} \sum_{\boldsymbol{\theta}_j \in \text{Succ}(\boldsymbol{\theta}_i)} ||\boldsymbol{\theta}_i - \boldsymbol{\theta}_j^k - \frac{1}{\beta}\boldsymbol{\lambda}_{ij}^k||^2 \Big\} \tag{3.6}$$

5: 2. Each agent updates $\boldsymbol{\lambda}_{ji}$ that he owns, for all j in P(i),

6:

$$\boldsymbol{\lambda}_{ji}^{k+1} = \boldsymbol{\lambda}_{ji}^k - \beta(\boldsymbol{\theta}_j^{k+1} - \boldsymbol{\theta}_i^{k+1}) \tag{3.7}$$

7: **end while**

Where β is the penalty parameter.
The approach used here to solve this distributed problem is an alternating approach in the primal and dual spaces which is presented in algorithm 1 with more details. This algorithm does a two step update for a number of iterations. In the first step, the parameter values in the primal space are computed and updated while in the second step, the same procedure happens for the dual parameters. The primal space parameters are $\boldsymbol{\theta_i}$ which are updated using the predecessor and successor nodes.
By assuming that the updates in 3.6 and 3.7 are done in $O(1)$, it can be proved that this method would converge to the optimal solution with the convergence rate of $O(\frac{1}{K})$.

3.2.2 Dominant properties of DADMM

As you could see in previous parts, DADMM approach has some important characteristics.
First, it provides us with a distributed approach to solve an optimization problem which can decrease the computational complexity and memory consumption for

each processor.
Second, this method guarantees converging to the optimal solution in case the assumptions are satisfied.
And finally, the rate of convergence for DADMM is faster than existing distributed approaches as this approach has a convergence rate of O($\frac{1}{k}$) while best existing approaches converge with the rate of O($\frac{1}{\sqrt{k}}$).

3.3 Summary

To sum up, we have introduced the vector form of DADMM approach in this chapter and have proved that It has $O(\frac{1}{k})$ rate of convergence in appendix A.
In the next chapter, we are going to extend this model to Multi-task learning problems with matrix form parameters.

Chapter 4

Proposing a Distributed framework for Multi-Task learning (DMTL)

Investigating the literature of MTL, it is apparent that all previous methods have focused on centralized approaches to find the proper model for the tasks. As number of samples for each task is not enough, should we increase the number of tasks, we can have a more accurate estimate for the shared repository model and therefore more precise model for each of the tasks.
Obviously, as the number of tasks increases, not only the time and space complexity for finding a solution would increase, but also it would not be possible to solve the problems using a centralized approach in some cases. Therefore, taking advantage from a distributed approach could result in better performance. So, the idea is to distribute the tasks among N different processors and solve the problem using DADMM which would obviously decrease the time and space complexity while guaranteeing to reach the optimum solution(provided that the assumptions are satisfied).

4.1 Multi-Task Learning using a Shared Repository

As stated in Chapter 1, a MTL problem could be formulated as an optimization problem to minimize the error over all tasks using their corresponding models. To make it more concrete, we can describe it using the below formulation:

$$\min_{\mathbf{L},\mathbf{S}} e_T(\mathbf{L},\mathbf{S})$$

$$= \min_{\mathbf{L},\mathbf{S}} \frac{1}{T}\sum_{t=1}^{T}\left\{\frac{1}{n_t}\sum_{l=1}^{n_t}\mathcal{L}\big(f\big(\mathbf{x}_l^{(t)};\mathbf{L}\mathbf{s}^{(t)}\big),y_l^{(t)}\big)+\mu_1||\mathbf{s}^{(t)}||_1\right\}+\mu_2||\mathbf{L}||_F^2$$

In which, $\mathbf{L} \in \mathbb{R}^{n\times d}$ is considered to be the shared repository. And each task's parameter is obtained using a weighting of the shared repository dimensions. Since each column of $\mathbf{S} \in \mathbb{R}^{T\times d}$ is a weighting multiplier for one of the tasks, each of the columns are independent from each other. Moreover, we have considered all abstract tasks to have d dimensions.
As discussed before, we have assumed that all tasks have dimension n without loss of generality and for notation simplicity. By assuming that the task parameters are

independent, we can rewrite the above equation as:

$$= \min_{\mathbf{L}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{s}^{(t)}} \Big\{ \sum_{l=1}^{n_t} \frac{1}{n_t} \mathcal{L}\big(f\big(\mathbf{x}_l^{(t)}; \mathbf{L}\mathbf{s}^{(t)}\big), y_l^{(t)}\big) + \mu_1 ||\mathbf{s}^{(t)}||_1 \Big\} + \mu_2 ||\mathbf{L}||_F^2 \tag{4.1}$$

where $\big(x_l^{(t)}, y_l^{(t)}\big)$ is the i^{th} training instance for task t, $\mathcal{L}$ is a known loss function, and the L_1 norm of s_t, $||\mathbf{s}^{(t)}||_1 = \sum_{i=1}^{n} |\mathbf{s}_i^{(t)}|$, is used as a convex approximation to the true vector sparsity.
Although equation 4.1 is not jointly convex considering $\mathbf{L}$ and $\mathbf{S}$, it would be convex by fixing one of the parameters and solving for the other. Therefore, it can be solved by using an alternating approach of solving optimization problems. For simplicity, we define function $M_t(\mathbf{L}, \mathbf{s}^{(t)})$ as below:

$$M_t(\mathbf{L}, \mathbf{s}^{(t)}) = \sum_{l=1}^{n_t} \frac{1}{T * n_t} \mathcal{L}\big(f\big(\mathbf{x}_l^{(t)}; \mathbf{L}\mathbf{s}^{(t)}\big), \mathbf{y}_l^{(t)}\big) + \frac{\mu_1}{T} ||\mathbf{s}^{(t)}||_1 \tag{4.2}$$

Therefore, the main equation can be rewritten as:

$$= \min_{\mathbf{L}} \sum_{t=1}^{T} \min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}, \mathbf{s}^{(t)}) + \mu_2 ||\mathbf{L}||_F^2 \tag{4.3}$$

We are going to discuss how to reformulate the above equation so as to be able to solve the MTL problem by employing a distributed framework next.

4.2 Splitting the tasks

In this section, we are going to split the tasks among N processors and solve the problem using a distributed framework.
By introducing N disjoint sets and splitting the tasks among them, we can divide 4.3 into N parts and rewrite it as below:

$$= \min_{\mathbf{L}} \sum_{t \in T_1} \min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}, \mathbf{s}^{(t)}) + \frac{\mu_2}{N} ||\mathbf{L}||_F^2 + \cdots + \sum_{t \in T_N} \min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}, \mathbf{s}^{(t)}) + \frac{\mu_2}{N} ||\mathbf{L}||_F^2 \tag{4.4}$$

N different agents solve each of these N parts in parallel while trying to reach a global solution in the end. So, the above equation can be written as:

$$= \min_{\mathbf{L}} \sum_{i=1}^{N} \Big\{ \sum_{t \in T_i} \min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}, \mathbf{s}^{(t)}) + \frac{\mu_2}{N} ||\mathbf{L}||_F^2 \Big\} \tag{4.5}$$

In the above formulation, T_i is a set containing i^{th} processor's tasks. If we let each of these N parts work independently and enforce them to have equal solution by putting an equality constraint on them, the solution would still be the same as equation 4.5 which turns into:

$$= \min_{\mathbf{L}} \sum_{i=1}^{N} \Big\{ \sum_{t \in T_i} \min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}_i, \mathbf{s}^{(t)}) + \frac{\mu_2}{N} ||\mathbf{L}_i||_F^2 \Big\} \tag{4.6}$$

$$\text{s.t.} \quad \mathbf{L}_1 = \cdots = \mathbf{L}_N$$

where, we have redefined $\mathbf{L}$ to be:

$$\mathbf{L} = \begin{bmatrix} \text{vec}(\mathbf{L}_1)^T \\ . \\ . \\ . \\ \text{vec}(\mathbf{L}_N)^T \end{bmatrix}$$

from now on and $\mathbf{L} \in \mathcal{R}^{N \times nd}$.

Similar to the DADMM approach, we require a two-step procedure to solve this problem.
First, we can to find the minimum of 4.2 by solving for $\mathbf{s}^{(t)}$. In this step, the value of $\mathbf{L}$ is considered fixed and we can simply find $\mathbf{s}^{\star(t)}$ for all tasks using one of the available dictionary learning approaches.
After updating $\{\mathbf{s}^{\star(t)} | \forall t \in \{1, \ldots, T\}\}$, these parameters are considered fixed in the second step and we have to solve the equation for $\mathbf{L}_i$. It is obvious that by fixing $\mathbf{S}$ and assuming $\mathcal{L}$ to be a convex cost function, each local cost function would be convex considering the parameter $\mathbf{L}_i$. By defining:

$$f_i(\mathbf{L}_i) = \sum_{t \in T_i} M_t(\mathbf{L_i}, \mathbf{s}^{\star(t)}) + \frac{1}{N} \mu_2 ||\mathbf{L}_i||_F^2 \tag{4.7}$$

and finding the solution for below equation:

$$F(\mathbf{L}) = \sum_{i=1}^{N} f_i(\mathbf{L}_i) \tag{4.8}$$

$$\text{s.t.} \quad \mathbf{L}_1 = \cdots = \mathbf{L}_N \tag{4.9}$$

we have reach the vector form DADMM problem and therefore, we can solve it using a distributed framework.
There is merely one unsolved problem in our distributed formulation of MTL. As we know, the DADMM framework works for the cost functions with scalar or vector input parameters while we have a matrix parameter in our formulation. We are going to extend the DADMM approach to matrix form parameters using vectorization and complete the DMTL framework.

4.3 DADMM for Matrix form Parameters

In this part, we are going to rewrite the main equation so as to reform the input matrix to be a vector. If we can write each agent's parameter in the vector form and rewrite the objective function using the vectorized element, we can straightforwardly solve MTL problem using DADMM. Thus, we have used the vectorization technique.

First, we can simply rewrite the constraint in 4.9 as:

$$\text{vec}(\mathbf{L}_1) = \cdots = \text{vec}(\mathbf{L}_N)$$

And we can rewrite the equation in a closed-form as:

$$\text{vec}([\mathbf{AL}]^T) = \mathbf{0} \tag{4.10}$$

which is equal to the constraint in 4.9. Second, we have to reformulate the main problem. As the value of the main problem is a scalar, and vec of scalar has the same value, we can pass 4.8 through a vec operator. So it would be:

$$\text{vec}(F(\mathbf{L}))$$

$$= \text{vec}(\sum_{i=1}^{N} f_i(\mathbf{L}_i))$$

$$= \sum_{i=1}^{N} \text{vec}(f_i(\mathbf{L}_i))$$

Since $f_i(\mathbf{L}_i) : \mathbb{R}^{n\times d} \to \mathbb{R}$, the vec operator passes through the function. Therefore:

$$\text{vec}(f_i(\mathbf{L}_i)) = \text{vec}\Big(\sum_{t\in T_i} M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)}) + \frac{\mu_2}{N}||\mathbf{L}_i||_F^2\Big)$$

$$= \sum_{t\in T_i} \text{vec}\Big(M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)})\Big) + \frac{\mu_2}{N}||\mathbf{L}_i||_F^2$$

$$= \sum_{t\in T_i} \text{vec}\Big(M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)})\Big) + \frac{\mu_2}{N}\text{vec}(\mathbf{L}_i)^T\text{vec}(\mathbf{L}_i)$$

and again since $M_t(\mathbf{L}_i, \mathbf{s}^{(t)}) : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times 1} \to \mathbb{R}$:

$$\text{vec}\Big(M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)})\Big) = \sum_{l=1}^{n_t} \frac{1}{T * n_t}\text{vec}\Big(\mathcal{L}\big(f\big(\mathbf{x}_l^{(t)}; \mathbf{L}_i\mathbf{s}^{(t)}\big), \mathbf{y}_l^{(t)}\big)\Big) + \frac{\mu_1}{T}||\mathbf{s}^t||_1 \tag{4.11}$$

So, if we can rewrite the loss function in terms of $\text{vec}(\mathbf{L}_i)$, the problem could be solved.
As you can see, this is a general framework and can help for solving many MTL problems. We call this framework Distributed Multi-task Learning framework or **DMTL**.
In the next sections, we are going to solve two examples of MTL problems using our distributed approach by introducing their loss functions. One is the **Regression** problem and the other is the **Reinforcement Learning** problem.

4.4 Regression Problem

The first problem to trace our proposed framework with is the well known regression problem. Assume that the agent is trying to solve a number of related regression problems. Therefore, we can solve the problem using MTL and a shared repository. We merely have to choose a loss function representing our current problem:

$$\mathcal{L}\big(f\big(\mathbf{x}_l^{(t)};\mathbf{L}\mathbf{s}^{(t)}\big),y_l^{(t)}\big)=(y_l^{(t)}-\mathbf{x}_l^{(t)}\mathbf{L}\mathbf{s}^{(t)})^T(y_l^{(t)}-\mathbf{x}_l^{(t)}\mathbf{L}\mathbf{s}^{(t)}) \tag{4.12}$$

where

$$f\big(\mathbf{x};\boldsymbol{\theta}\big)=\mathbf{x}\boldsymbol{\theta}$$

while $(\mathbf{x}_l,y_l)$ is a paired instance. Consequently:

$$\min_{\mathbf{s}^{(t)}} M_t(\mathbf{L}_i,\mathbf{s}^{(t)})=\min_{\mathbf{s}^{(t)}}\Big\{\frac{1}{n_t*T}\sum_{l=1}^{n_t}\big\{(y_l-\mathbf{x}_l\mathbf{L}_i\mathbf{s}^{(t)})^T(y_l-\mathbf{x}_l\mathbf{L}_i\mathbf{s}^{(t)})\big\}+\frac{\mu_1}{T}||\mathbf{s}^t||_1\Big\}$$

$$=\min_{s^{(t)}}\frac{1}{n_t*T}\Big(\mathbf{y}-\mathbf{X}\mathbf{L}_i\mathbf{s}^{(t)}\Big)^T\Big(\mathbf{y}-\mathbf{X}\mathbf{L}_i\mathbf{s}^{(t)}\Big)+\frac{\mu_1}{T}||\mathbf{s}^t||_1 \tag{4.13}$$

which can be found using one of the dictionary learning approaches such as lasso. Now, we have to use vectorization to reshape $\mathbf{L}_i$s to be $vec(\mathbf{L}_i)$:

$$=\min_{s^{(t)}}\frac{1}{n_t*T}\Big(\mathbf{y}-\big({\mathbf{s}^{(t)}}^T\otimes\mathbf{X}\big)vec(\mathbf{L}_i)\Big)^T\Big(\mathbf{y}-\big({\mathbf{s}^{(t)}}^T\otimes\mathbf{X}\big)vec(\mathbf{L}_i)\Big)+\frac{\mu_1}{T}||\mathbf{s}^t||_1 \tag{4.14}$$

Therefore, we have provided the equal vectorized formulation of $M_t(\mathbf{L}_i,\mathbf{s}^{(t)})$. Having found $\mathbf{s}^{\star(t)}$ and fixing it's value, we can find $\mathbf{L}_i\forall i\in\{1,\dots,N\}$ by solving 4.8 for $vec(\mathbf{L}_i)$. By nulling it's gradient, we can find the solution. The resulting equation for the regression problem would be in the form of $\mathbf{A}vec(\mathbf{L}_i)=\mathbf{b}$ where $\mathbf{A}$ and $\mathbf{b}$ can be found by:

$$\mathbf{A}=\sum_{t=1}^{t_i}\Big\{\frac{2}{n_t*T}(\mathbf{s}^{(t)}{\mathbf{s}^{(t)}}^T\otimes\mathbf{X}^T\mathbf{X})+\frac{2\mu_2}{T*N}\mathbf{I}_{dn\times dn}\Big\}+\beta\sum_{j\in Pred(i)\cup Succ(i)}1 \tag{4.15}$$

$$\mathbf{b}=\sum_{t=1}^{t_i}\frac{2}{n_t*T}({\mathbf{s}^{(t)}}^T\otimes\mathbf{X})^T\mathbf{y} \tag{4.16}$$

$$+\beta\Big\{\sum_{j\in Pred(i)}-vec(\mathbf{L}_i)+\frac{\boldsymbol{\lambda}_{ji}}{\beta}+\sum_{j\in Succ(i)}-vec(\mathbf{L}_j)-\frac{\boldsymbol{\lambda}_{ij}}{\beta}\Big\}$$

4.5 Reinforcement Learning using policy gradients

In this section, we are going to find the optimal policy for a number of Reinforcement Learning(RL) tasks in a multi-task learning setting.
Having introduced the policy gradient approaches in chapter 1, the objective for RL problem would be the same as the general multi-task learning and the loss function should be introduced. So,

$$\min_{\mathbf{L},\mathbf{S}} e_T(\mathbf{L},\mathbf{S}) = \min_{\mathbf{L}} \frac{1}{T}\sum_{t=1}^{T}\Big\{\min_{\mathbf{s}^{(t)}} -J(\mathbf{L}\mathbf{s}^{(t)})) + \mu_1||\mathbf{s}^{(t)}||_1\Big\} + \mu_2||\mathbf{L}||_F^2 \tag{4.17}$$

The policy we are using here is a Gaussian policy with parameter θ and the samples are trajectories. Therefore, the loss function would be:

$$\mathcal{L}\big(f\big(\mathbf{X}_l;\mathbf{L}\mathbf{s}^{(t)}\big),\mathbf{a}_l\big) = R(\tau_l) * (\mathbf{a}_l - \mathbf{X}_l\mathbf{L}\mathbf{s}^{(t)})^T(\mathbf{a}_l - \mathbf{X}_l\mathbf{L}\mathbf{s}^{(t)}) \tag{4.18}$$

where each column of $\mathbf{X}$ represents a training sample and $\mathbf{a}_l$ represents the l^{th} trajectory's corresponding actions.
Employing this loss function in the general distributed multi-task learning setting of equation 4.2 would lead to:

$$M_t(\mathbf{L}_i,\mathbf{s}^{(t)}) = \frac{1}{n_t * T}\sum_{l=1}^{n_t}\big\{R(\tau_l) * (\mathbf{a}_l - \mathbf{X}_l\mathbf{L}\mathbf{s}^{(t)})^T(\mathbf{a}_l - \mathbf{X}_l\mathbf{L}\mathbf{s}^{(t)})\big\} + \frac{\mu_1}{T}||\mathbf{s}^t||_1 \tag{4.19}$$

which is equal to the below vectorized format:

$$= \frac{1}{n_t * T}\sum_{l=1}^{n_t}\big\{R(\tau_l) * (\mathbf{a}_l - \big(\mathbf{s}^{(t)^T}\otimes\mathbf{X}_l\big)vec(\mathbf{L}_i))^T(\mathbf{a}_l - \big(\mathbf{s}^{(t)^T}\otimes\mathbf{X}_l\big)vec(\mathbf{L}_i))\big\}$$

$$+\frac{\mu_1}{T}||\mathbf{s}^t||_1 \tag{4.20}$$

This can simply be solved by using simple lasso algorithm. Having found $\mathbf{s}^{\star(t)}$s, the equation in the distributed form to be solved for $\mathbf{L}$ would be:

$$\min_{\mathbf{L}}\sum_{i=1}^{N}\sum_{t=1}^{t_i}\Big\{M_t(\mathbf{L}_i,\mathbf{s}^{\star(\mathbf{t})}) + \frac{\mu_2}{N}||\mathbf{L}_i||_F^2\Big\} \tag{4.21}$$

$$f_i(\mathbf{L}_i) = \sum_{t=1}^{t_i} M_t(\mathbf{L}_i,\mathbf{s}^{\star(t)}) + \frac{\mu_2}{N}||\mathbf{L}_i||_F^2 \tag{4.22}$$

For simplicity, we have to rewrite the equation in the vectorized format and solve equation 4.23 for $vec(\mathbf{L})$.

$$L(\mathbf{L},\boldsymbol{\lambda}) = \sum_{i=1}^{N} f_i(\mathbf{L}_i) - \boldsymbol{\lambda}^T\mathrm{vec}\big([\mathbf{A}\mathbf{L}]^T\big)$$

$$+\frac{\beta}{2}\text{vec}\big([\mathbf{AL}]^T\big)^T\text{vec}\big([\mathbf{AL}]^T\big) \tag{4.23}$$

And the next step is to re-write $f_k(\mathbf{L}_k)$ in the vectorized form so as to use the DADMM approach:

$$f_i(\mathbf{L}_i) = \sum_{t=1}^{t_i} M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)}) + \frac{\mu_2}{N}\text{vec}\big(\mathbf{L}_i\big)^T\text{vec}(\mathbf{L}_i) \tag{4.24}$$

where:

$$M_t(\mathbf{L}_i, \mathbf{s}^{\star(t)}) = \tag{4.25}$$

$$\frac{1}{n_t * T}\sum_{l=1}^{n_t}\big\{R(\tau_l) * (\mathbf{a}_l - (\mathbf{s}^{(t)^T} \otimes \mathbf{X}_l) * \text{vec}(\mathbf{L}_i))^T(\mathbf{a}_l - (\mathbf{s}^{(t)^T} \otimes \mathbf{X}_l) * \text{vec}(\mathbf{L}_i))$$

$$+\frac{\mu_1}{T}||\mathbf{s}^t||_1\big\}$$

In order to use DADMM, we have to find the minimum of $f_i(\mathbf{L}_i)$ by solving $\bigtriangledown_{\mathbf{L}_i} f_i(\mathbf{L}_i) = 0$. Therefore the equation can be written as $\mathbf{A}\text{vec}(\mathbf{L}_i) = \mathbf{b}$, where:

$$\mathbf{A} = \sum_{t=1}^{t_i}\Big\{\sum_{l=1}^{n_t}\frac{2}{n_t * T}(\mathbf{s}^{(t)}\mathbf{s}^{(t)^T} \otimes \mathbf{X}_l^T\mathbf{X}_l) + \frac{2\mu_2}{T * N}\mathbf{I}_{dn\times dn}\Big\} + \beta\sum_{j\in Pred(i)\cup Succ(i)} 1 \tag{4.26}$$

$$\mathbf{b} = \sum_{t=1}^{t_i}\sum_{l=1}^{n_t}\frac{2}{n_t * T}(\mathbf{s}^{(t)^T} \otimes \mathbf{X}_l)^T\mathbf{a}_l \tag{4.27}$$

$$+\beta\Big\{\sum_{j\in Pred(i)} -vec(\mathbf{L}_i) + \frac{\boldsymbol{\lambda}_{ji}}{\beta} + \sum_{j\in Succ(i)} -vec(\mathbf{L}_j) - \frac{\boldsymbol{\lambda}_{ij}}{\beta}\Big\}$$

4.6 Summary

So far, we have proposed the **DMTL** framework by extending the DADMM approach to the matrix form and distributing the MTL optimization equation. In other words, we have proposed a way to make the matrix form MTL suitable for vector form DADMM and provided it's rate of convergence proof. So, we have guaranteed that our approach would reach the optimal solution provided that the assumptions are satisfied. In the next chapter, we are going to investigate this framework using experimental analysis.

Chapter 5

Experimental Results and analysis

5.1 Introduction

In this chapter we are going to prove our claims using empirical results. Moreover, we are going to compare DMTL approach for the regression problem with previous MTL approaches.
Since there has been no distributed approach introduced for MTL, we are going to compare our model with two non-distributed models:

1. a simple **centralized** solver or Batch MTL: Which simply uses the whole data and solves for **L** and then **S** iteratively.
2. **GOMTL**: a centralized batch solver as defined in **GOMTL**
3. **ELLA**: a centralized online solver as defined in **ELLA**

We will use the first model to demonstrate that our distributed approach reaches the same solution as the centralized method and the other two to compare DMTL method's convergence value and time with.
As reported in **ELLA** GOMTL approach converges to a solution with lower MSE compared to current approaches while ELLA has the characteristic of being faster among all approaches.
Using the provided theoretical gauratees of appendix A, we know that DMTL's rate of convergence is $O(\frac{1}{k})$ while ELLA has $O(\frac{1}{\sqrt{k}})$ where k is the number of iterations. It is obvious that our method is faster using large enough datasets. Knowing that GOMTL is more accurate that ELLA, we are going to demonstrate that our approach is more exact while converging faster comparing other two methods.
As we have seen in the previous chapter, the main equation in the centralized approach exactly equals the equation obtained in the distributed MTL framework. Therefore, the obtained solution of the distributed framework would be the same as the centralized approach while the former has a faster rate of convergence.
In this chapter, we are going to introduce some datasets first and analyze our approach.

5.2 Datasets

There are four datasets used to analyze our work. They can be grouped into two categories of "Artificial Multi-task Learning Problem" and "Multi-task Learning Problem" datasets.

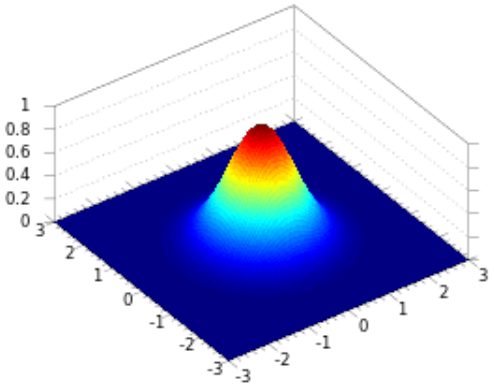

FIGURE 5.1: Synthetic Samples are generated from a Gaussian distribution

5.2.1 Artificial Multi-task Learning Datasets

These tasks are not MTL problems by themselves but they are simple regression tasks that have enough instances so as to be divided into groups. We have used them for two reasons:
First, the groups are related because they actually represent the same regression problem.
Second, they can be called big data and would show the advantages of our approach over the centralized methods.

Real Regression Dataset

The real dataset we have used for this experiment is one of the UCI repository's (**Lichman:2013**) regression problem datasets named "Physicochemical Properties of Protein Tertiary Structure(PPPTS)". This dataset is taken from CASP datasets. CASP or Critical Assessment of protein Structure Prediction is a worldwide experiment for protein structure prediction. PPPTS dataset includes:

- 45730 Instances
- 9 Features.

Artificial Regression Dataset

To be able to analyze the work from different perspectives, we also have artificially created a number of datasets by varying the number of samples m and features n. We will refer to this data by "synthetic" dataset.
Having set the value for m and n, the procedure would be done in two steps:

1. Generate samples $\mathbf{X} \in \mathcal{R}^{m \times n}$, randomly from a Gaussian distribution such as figure 5.1
2. Construct the labels using $\mathbf{y} = 3\mathbf{X} + \boldsymbol{\epsilon}$

Construction of an Artificial MTL dataset

To analyze the proposed framework, we have constructed a Multi-Task dataset using "Real Regression Dataset" and "Artificial Regression Dataset" by three steps described below:

1. Set T as the number of tasks your problem encompasses.
2. Shuffle the samples.
3. Split the data to T different groups.

Each of these T groups represent a task. Therefore, we would have T tasks while each one includes a number of instances.

5.2.2 Multi-task Learning Datasets

- **Ella's Synthetic Regression Tasks**: A set of 100 random tasks with 13 features and $n_t = 100$ instances per task. The task parameter vectors were generated as a linear combination of 6 randomly generated latent components in $\mathcal{R}^{12}$. The vectors s(t) had a sparsity level of 0.5. The training data $\mathbf{X}(t)$ was generated from a standard normal distribution. The training labels were given as $y(t) = \mathbf{X}(t)^T \boldsymbol{\theta}(t) + \epsilon$, where each element of ϵ is independent univariate Gaussian noise. A bias term was added as the 13th feature prior to learning.
- **London School dataset:** It consists of 15362 students from 139 schools' examination scores. We have considered the grades for each school as a separate task and we ought to predict each student's examination score by the end. The features are the same as the ones used in **ELLA** and **GOMTL** and we have also added a bias term. It is clear that each instance would have 27 features.

We are going to present the results for above datasets so as to empirically demonstrate the advantages of our approach.

5.3 Evaluation

There exists two aspects that we are going to discuss in order to evaluate our approach.
On the one hand, we are going to show that our approach's solution converge to the same value as the simple batch centralized approach.
On the other hand, we are going to compare our model's convergence time with the centralized approaches and demonstrate that DMTL would converge faster in most cases.

5.3.1 Evaluation metrics

To evaluate the models we have used "Mean Square Error" and "General Consensus Problem Error" standard metrics which have been defined below:

- **Mean Square Error(MSE)**: It is defined as the average of squares of errors where the error is the difference between the estimator and the estimated value. More formally if we show the estimator by y and the estimated by $\hat{y}$:

$$\text{MSE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

  Obviously, we have used this metric in our regression objective function to estimate the model's error in section 4.4.

- **General Consensus Problem(GCP) Error**: When all local variables should agree, i.e., be equal for finding the solution, we have a GCP. Here, we have defined GCP error as the standard deviation of all local solutions:

$$\text{GCP}(\boldsymbol{\theta}_1, ...\boldsymbol{\theta}_N) = \text{std}(\boldsymbol{\theta}_1, ...\boldsymbol{\theta}_N) \tag{5.2}$$

5.3.2 Evaluation Setting

To evaluate the approaches, we have randomly partitioned the tasks among agents for 5 times and found the average and standard deviation of the error values of each approach.

Optimal Value Convergence setting: For the first part, the parameters μ_1 and μ_2 are the same for both methods and β and k,which is the number of iterations each agent iterates to find **L**, is chosen using cross validation. We are going to show that both approaches reach almost the same solution.
The horizontal line of the figure shows the iteration count while the vertical line represents the main objective value of 4.8 which is MSE for our implementation. Note that the algorithms would be represented by blue color as soon as they converge and stop learning.
In addition, we will plot the GCP Error to show that the agents almost agree and have almost found the same solution.

Computation time setting: For the second part, we have varied the number of processors(agents) and shown that increasing the number of agents can help reduce the computation time as the number of tasks increase and they become more complex. Moreover, the value for β and k is found using cross validation.
We have measure the DMTL method's time by adding all iteration's times till the algorithm converges. Each iteration's time is computed by adding the maximum time among all processor's for updating **S**, which is referred by 3.6, or **L** and the time required to update the $\boldsymbol{\lambda}$ parameter when the agent's are updating **L**, which is referred by 3.7.

5.3.3 Optimal Value Convergence Analysis

In order to demonstrate that our DMTL approach can reach the same solution as the simple centralized approach, we have run the algorithm on the "London Schools" and "Synthetic" Datasets.

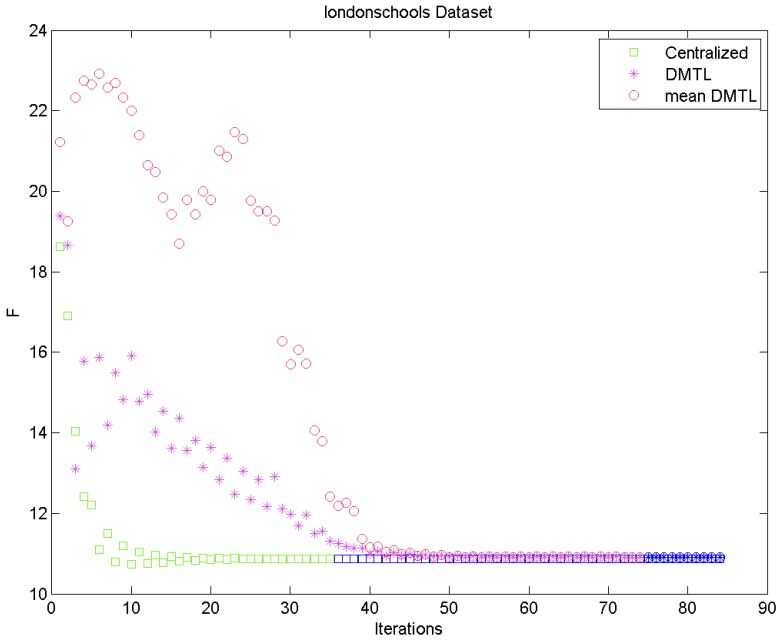

FIGURE 5.2: $\beta = 0.1$ and $k = 1$ - London Schools Dataset

In addition, other than the centralized and DMTL approach, we have plotted another model and named it "Mean MTL". This model is the same as DMTL while the error is computed by replacing all $\mathbf{L}_i$s with the average of all agent's $\mathbf{L}_i$ parameters in the evaluation part. It will help us to see how the model reduces GCP error and the agent's share their knowledge.

Result

First, we are going to show the effect of increasing the value of k in the convergence value result.
As you can see in table 5.1 and figures 5.2, 5.3, 5.4, 5.5 and 5.6 of London Schools dataset, although the algorithm converges to the solution in the last resort, the behavior of DMTL becomes more close to the behavior of the GOMTL approach as the k value increases.

We can see the same outcome for the Synthetic dataset in 5.7, 5.8, 5.9, 5.10 and 5.11.

In 5.14 and 5.12 figures, we see that both of the Centralized and DMTL approaches have converged almost to the same objective value for Synthetic and London Schools datasets.
The result of running the experiment 10 times and averaging the results would lead to the results in 5.2.

Moreover, by taking the difference between the value of the "mean DMTL" and simple "DMTL" approach into account, we can see that in the initial steps of the algorithm values are far from each other. This is a sign to show that the agent's have not shared any knowledge in the first steps. However, as the algorithm continues and more iterations pass, the values become closer and both models reach the same

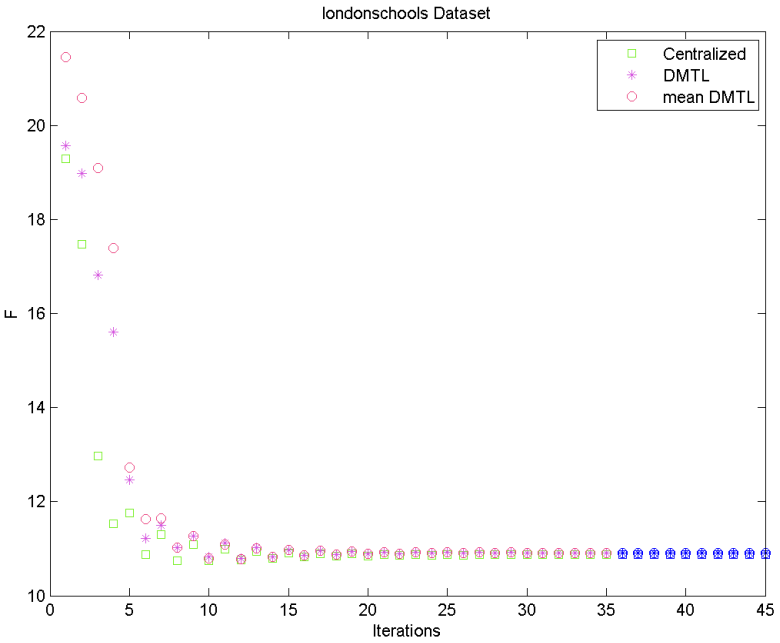

FIGURE 5.3: $\beta = 0.1$ and $k = 5$ - London Schools Dataset

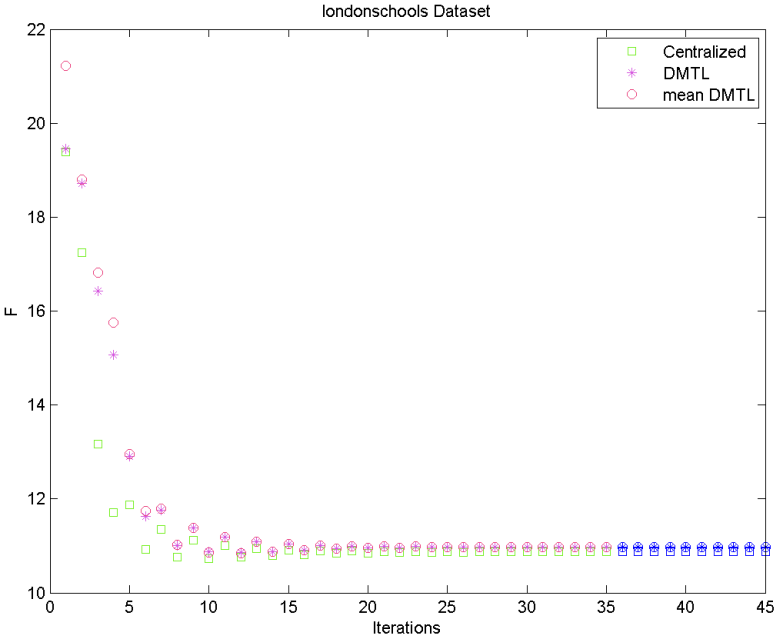

FIGURE 5.4: $\beta = 0.1$ and $k = 10$ - London Schools Dataset

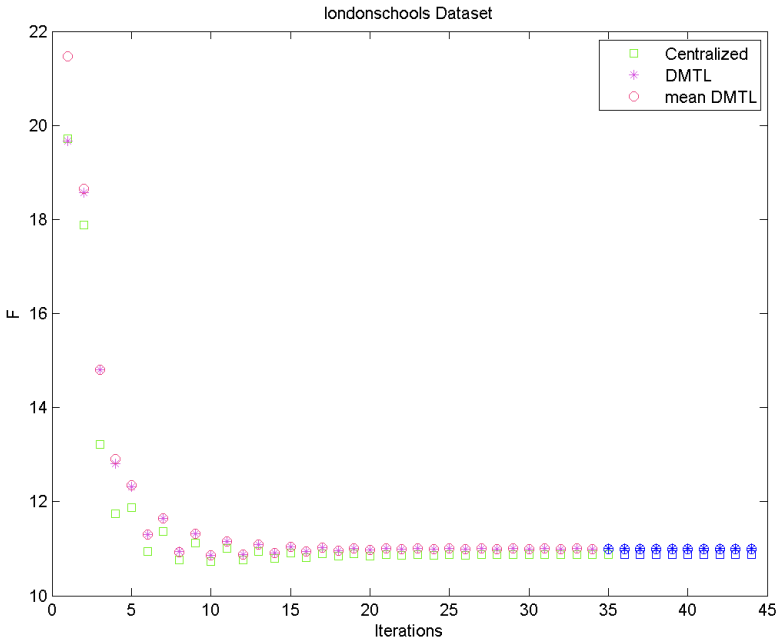

FIGURE 5.5: $\beta = 0.1$ and $k = 15$ - London Schools Dataset

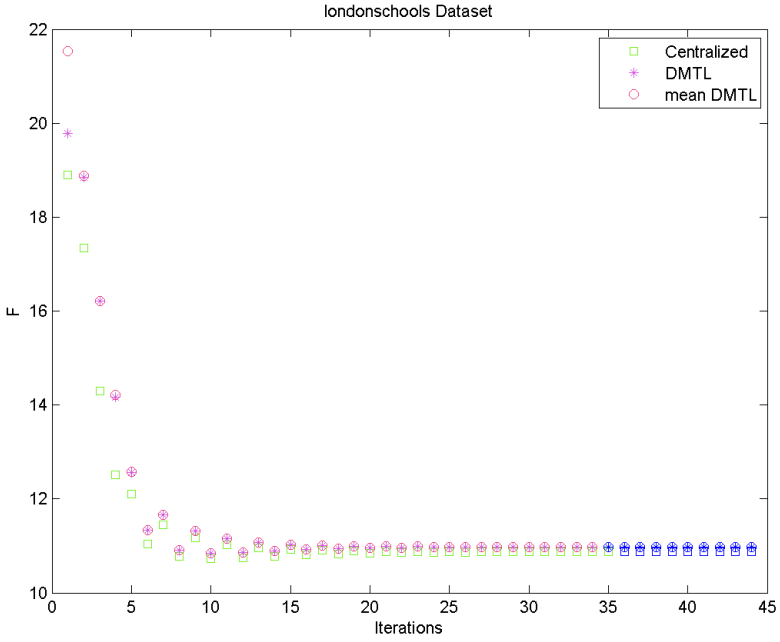

FIGURE 5.6: $\beta = 0.1$ and $k = 20$ - London Schools Dataset

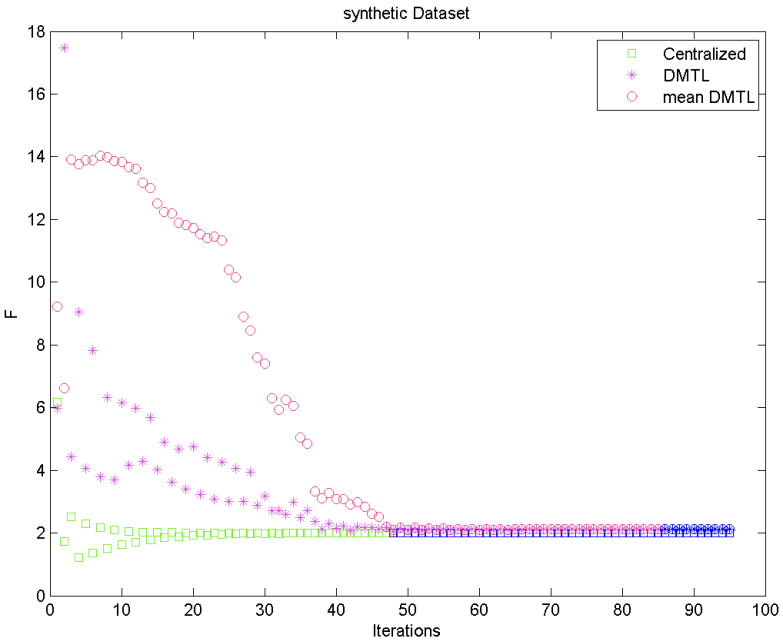

FIGURE 5.7: $\beta = 0.1$ and $k = 1$ - Synthetic Dataset

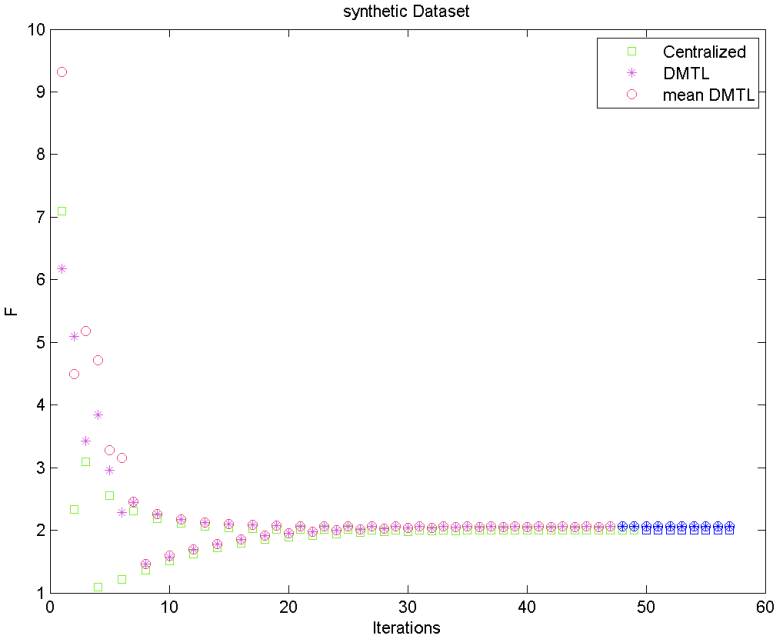

FIGURE 5.8: $\beta = 0.1$ and $k = 5$ - Synthetic Dataset

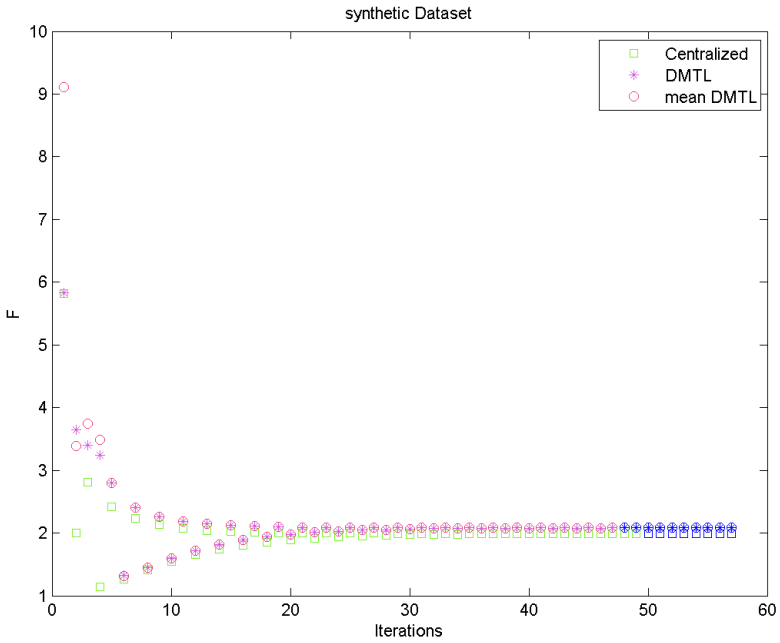

FIGURE 5.9: $\beta = 0.1$ and $k = 10$ - Synthetic Dataset

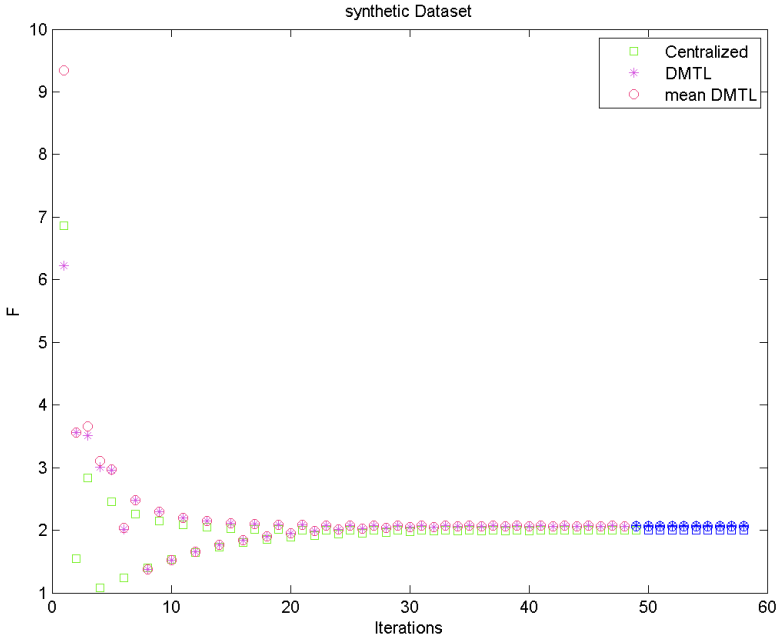

FIGURE 5.10: $\beta = 0.1$ and $k = 15$ - Synthetic Dataset

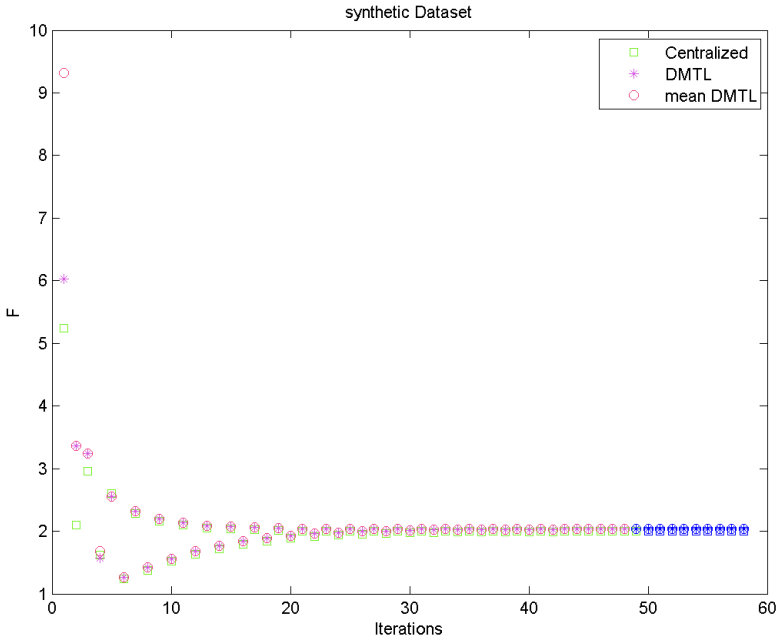

FIGURE 5.11: $\beta = 0.1$ and $k = 20$ - Synthetic Dataset

TABLE 5.1: Relative Accuracy and Convergence Solution of DMTL by varying k and $\beta = 0.1$ - London Schools Dataset

k	DMTL value	Relative Accuracy
1	10.9821 ± 0.052582	0.98989
5	10.9693 ± 0.034803	0.99106
10	10.99 ± 0.030575	0.98918
15	10.9981 ± 0.038333	0.98843
20	10.9319 ± 0.034606	0.99452

TABLE 5.2: Evaluation result of Centralized and DMTL

Dataset\Model	Centralized	DMTL	Relative Accuracy
Synthetic	2.5534 ± 0.020506	2.5745 ± 0.028577	0.9917
London Schools	11.2309 ± 0.060715	11.3377 ± 0.25453	0.9904

value in the end.
From another perspective, by looking at the GCP error of the DMTL approach in figures of 5.15 and 5.13, the model error has decreased and the agents have reached an agreement. So given a desired convergence error limit, our model has found a general solution for all the agents.

5.3.4 Computational Complexity Analysis

Since we have completed analyzing the solution value of our approach, we are going to investigate out method to watch the running time of DMTL in this part. Although many of MTL approaches can be extended to this DMTL framework, the base method we have used to solve MTL problem is Batch MTL for two reasons:

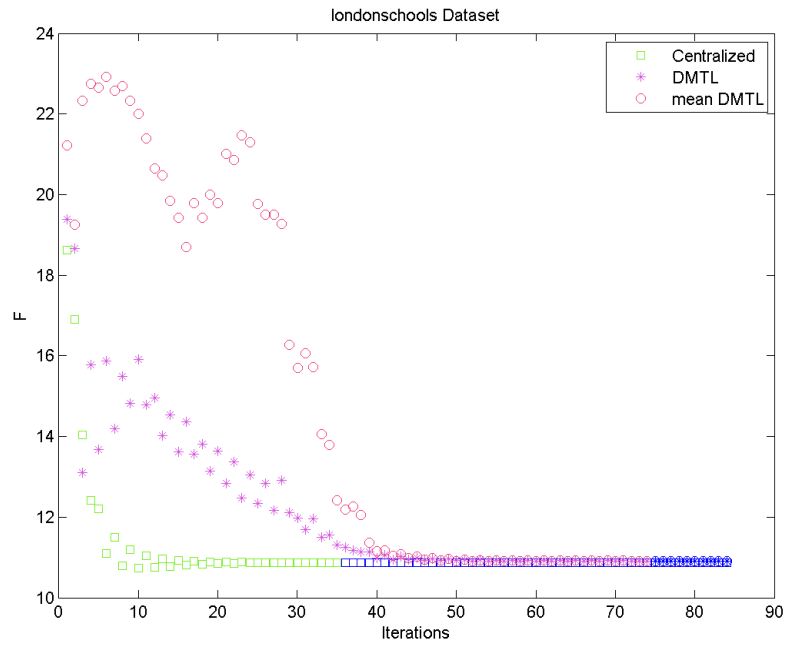

FIGURE 5.12: The centralized and DMTL approaches have converged to the same solution

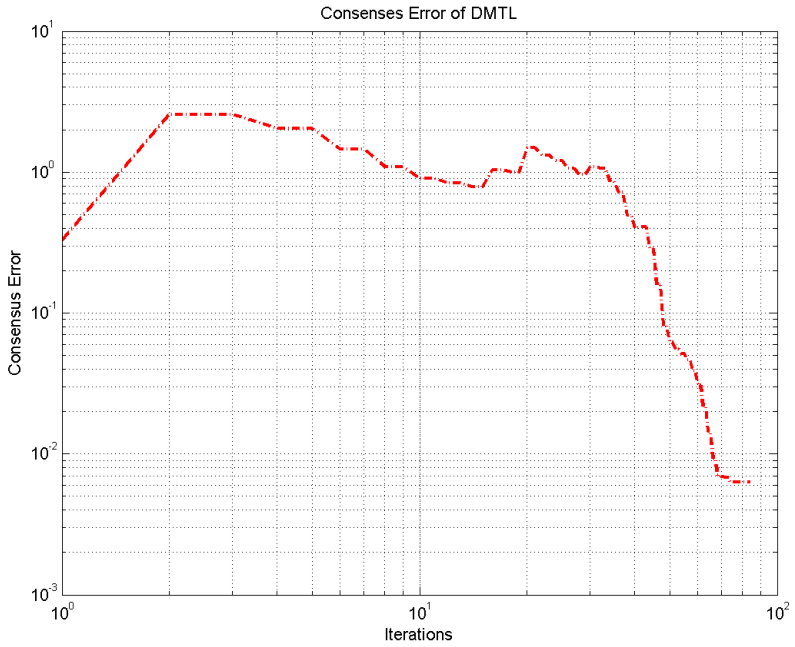

FIGURE 5.13: Consensus error of London Schools dataset has decreased as the algorithm converges

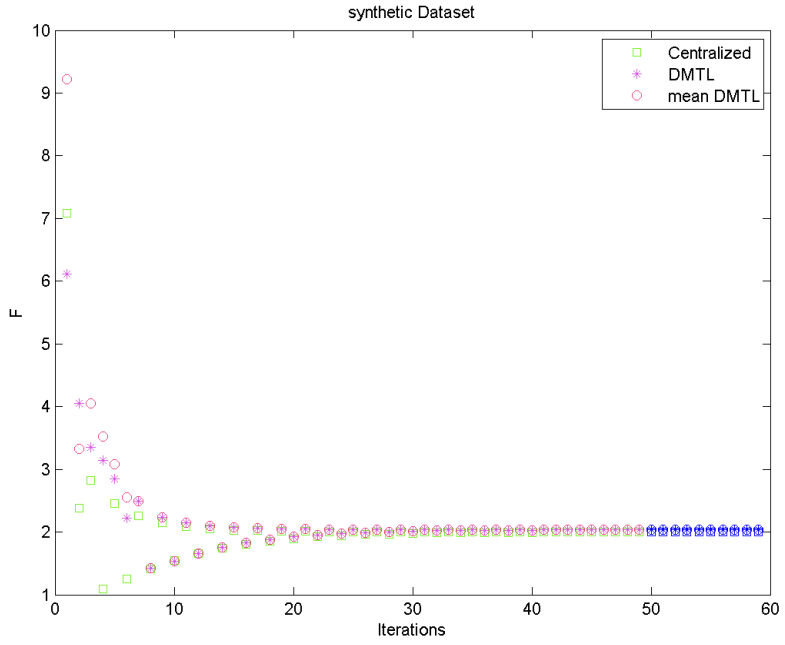

FIGURE 5.14: The centralized and DMTL approaches have converged to the same solution

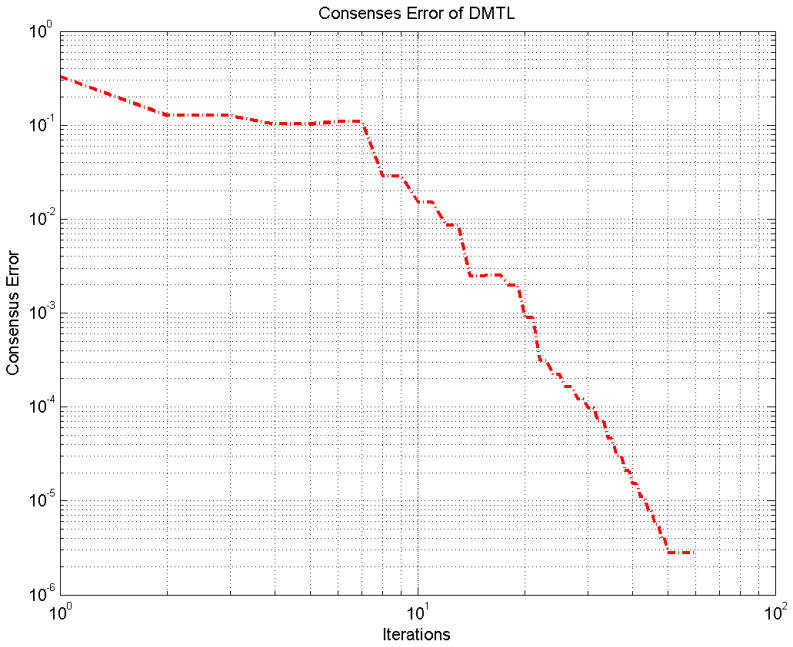

FIGURE 5.15: Consensus error of Synthetic dataset has decreased as the algorithm converges

TABLE 5.3: Comparing

Dataset\Model	Centralized	DMTL	Relative Accuracy
Synthetic	2.5534±0.020506	2.5745 ± 0.028577	0.9917
London Schools	11.2309 ± 0.060715	11.3377 ± 0.25453	0.9904

First, it has the best performance compared to all previous approaches and would lead to a better solution.
And second, it does require more time to reach the solution and our distributed approach can make it faster.

5.4 Summary

In this section, we presented our analysis of the convergence properties of our DMTL model for multi-task regression problems and demonstrated that our method is robust, fast and requires less memory requirement for each processor compared to the centralized approaches.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have successfully proposed a framework for MTL problems to be solved in a distributed manner for the first time.
Moreover, we have proposed the solution of increasing the number of tasks so as to be able to find a more robust model to solve the MTL problem and the idea to solve the problem using our proposed distributed framework has resulted in reducing the time and space complexity which makes large data processing more efficient.
In addition, our approach converges to the same value to which the closed form solution would converge. In conclusion, we have contributed:

1. A Distributed framework for Multi-task learning,
2. A fast algorithm with low convergence rate by distributing the tasks among K processors,
3. Decreased the memory consumption for each agent as the number of tasks assigned to each agent is less,
4. Extended the DADMM approach to optimization problems with vector and matrix form.
5. Proved that the Matrix form DADMM has $O(\frac{1}{k})$ rate of convergence where k is the number of iterations.

6.2 Future Work

Looking through the process of this research, there exists lots of ideas to be worked on in future.

1. The experimental analysis of reinforcement learning problems can also be investigated.
2. One of the possible ideas is to extend this method for lifelong learning problems by introducing an online setting.
3. We can also extend this method to Reinforcement Learning problems in an online setting.
4. This framework can also be extended to other applications such as face recognition or inferring social network structure.

Appendix A

Vector form DADMM Convergence Analysis

In this part, we are going to present our analysis for the DADMM approach with the input element to be in vector form. This analysis is mostly based on **boyd2011distributed** and **DADMM**
We demonstrate that the rate of convergence for this case is also $O(\frac{1}{k})$ where k is the number of iterations. Moreover, we also count all node updates in each iteration as one.

A.1 Notations, definitions and assumptions

As introduced in the main chapters, $\mathbf{A} \in \mathcal{R}^{M \times N}$, is an adjacency matrix while each row represents a constraint(or an edge in the graph) and each column is related to one of the agents.
We define matrix $\mathbf{B} \in \mathcal{R}^{M \times N}$ whose value is assigned by $\mathbf{B} = \min(0, \mathbf{A})$. Each row of matrix **B** correspond to one edge (i, j) with $i < j$. All the elements of this row would be zero except the value of -1 in position j.
Note that in this chapter $\mathbf{X}$ do not represent the samples. The parameter we are searching for is $\mathbf{X} \in \mathcal{R}^{N \times d}$, each row of which represents the learning element of each agent. Therefore it can be shown as:

$$\mathbf{X} = \begin{bmatrix} \dots & \mathbf{X}_1^T & \dots \\ \dots & \mathbf{X}_2^T & \dots \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ \dots & \mathbf{X}_N^T & \dots \end{bmatrix}$$

While we represent the i^{th} row and j^{th} column of matrix $\mathbf{X}$ by $[\mathbf{X}]^i$ and $[\mathbf{X}]_j$ respectively, for simplicity, we have defined $X_i = \left([X]^i\right)^T$ here which represents the i^{th} agent's parameter vector.
$\boldsymbol{\lambda} \in \mathcal{R}^{Md \times 1}$ is the Lagrange multiplier and by $\boldsymbol{\lambda}_{ij} \in \mathcal{R}^{d \times 1}$, we are referring to the Lagrange multiplier vector corresponding to edge (i, j).
This proof is also based on the two assumptions of **Convexity** and **Existence of Saddle point**.

A.2 Convergence Analysis

Since we have introduced the required notation, we are going to introduce three lemmas and provide main theorem's proof in this section.

A.2.1 Lemma 1

Let $\{\mathbf{X}^k, \boldsymbol{\lambda}^k\}$ be the iterates generated by our distributed algorithm for 3.3 while $\mathbf{X}^k = (\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ \mathbf{X}_1 & \mathbf{X}_2 & \dots & \mathbf{X}_N \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix})^T$ and vector $\boldsymbol{\lambda}^k = [\boldsymbol{\lambda}_{ij}^k]_{ij,e_{ij}\in E}$. Then the below relation holds for all k:

$$F(\mathbf{X}) - F(\mathbf{X}^{k+1}) - \text{vec}\Big(\big[\mathbf{A}(\mathbf{X} - \mathbf{X}^{k+1})\big]^T\Big)^T \boldsymbol{\lambda}^{k+1}$$

$$+\beta \text{vec}\Big((\mathbf{X} - \mathbf{X}^{k+1})^T\Big)^T \text{vec}\Big(\big[(-\mathbf{A} + \mathbf{B})^T \mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)\big]^T\Big) \geq 0 \tag{A.1}$$

for any $\mathbf{X} \in \mathcal{R}^{N\times d}$, where $\mathbf{A}$ is the edge-node incidence matrix of the graph and $\mathbf{B} = \min(0, A)$.

Proof: We denote $g_i : \mathcal{R}^d \to \mathcal{R}$ to be the function

$$g_i^k(\mathbf{X}_i) = \frac{\beta}{2} \sum_{j\in P(i)} ||\mathbf{X}_j^{k+1} - \mathbf{X}_i - \frac{1}{\beta}\boldsymbol{\lambda}_{ji}^k||_F^2 + \frac{\beta}{2} \sum_{j\in S(i)} ||\mathbf{X}_i - \mathbf{X}_j^k - \frac{1}{\beta}\boldsymbol{\lambda}_{ij}^k||_F^2 \tag{A.2}$$

and due to the update in 3.6, X_i^{k+1} is the optimizer of $g_i^k + f_i$. The optimality implies that:

$$\exists h(\mathbf{X}_i^{k+1}) \in \partial f_i(\mathbf{X}_i^{k+1}) : \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \big[h(\mathbf{X}_i^{k+1}) + \nabla g_i^k(\mathbf{X}_i^{k+1})\big] = 0 \tag{A.3}$$

for all $\mathbf{X}_i \in \mathcal{R}^d$.
In addition, considering the definition of sub-gradient, we have:

$$f_i(\mathbf{X}_i) \geq f_i(\mathbf{X}_i^{k+1}) + \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T h(\mathbf{X}_i^{k+1}) \tag{A.4}$$

Therefore, by using A.3 and A.4 will conclude that:

$$f_i(\mathbf{X}_i) - f_i(\mathbb{X}_i^{k+1}) + \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \nabla g_i^k(\mathbf{X}_i^{k+1}) \geq 0 \tag{A.5}$$

By taking the derivative of g_i^k, we obtain:

$$\nabla g_i^k(\mathbf{X}_i) = -\beta \sum_{j \in P(i)} (\mathbf{X}_j^{k+1} - \mathbf{X}_i - \frac{1}{\beta}\boldsymbol{\lambda}_{ji}^k) + \beta \sum_{j \in S(i)} (\mathbf{X}_i - \mathbb{X}_j^k - \frac{1}{\beta}\boldsymbol{\lambda}_{ij}^k) \tag{A.6}$$

and by substituting $\nabla g_i^k(\mathbf{X}_i^{k+1})$ in A.5, we obtain:

$$f_i(\mathbf{X}_i) - f_i(\mathbb{X}_i^{k+1}) + \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[-\beta \sum_{j \in P(i)} (\mathbf{X}_j^{k+1} - \mathbf{X}_i^{k+1} - \frac{1}{\beta}\boldsymbol{\lambda}_{ji}^k)$$

$$+\beta \sum_{j \in S(i)} (\mathbf{X}_i^{k+1} - \mathbb{X}_j^k - \frac{1}{\beta}\boldsymbol{\lambda}_{ij}^k) \Big] \geq 0 \tag{A.7}$$

Using 3.7 we get:

$$f_i(\mathbf{X}_i) - f_i(\mathbb{X}_i^{k+1}) + \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[\sum_{j \in P(i)} (\boldsymbol{\lambda}_{ji}^{k+1}) + \sum_{j \in S(i)} (-\boldsymbol{\lambda}_{ij}^{k+1})$$

$$+ \sum_{j \in S(i)} \beta(\mathbf{X}_j^{k+1} - \mathbb{X}_j^k) \Big] \geq 0 \tag{A.8}$$

If we sum the above equation over all agents, it turns into:

$$\sum_{i=1}^{n} f_i(\mathbf{X}_i) - \sum_{i=1}^{n} f_i(\mathbb{X}_i^{k+1}) + \sum_{i=1}^{n} \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[\sum_{j \in P(i)} (\boldsymbol{\lambda}_{ji}^{k+1}) + \sum_{j \in S(i)} (-\boldsymbol{\lambda}_{ij}^{k+1})$$

$$+ \sum_{j \in S(i)} \beta(\mathbf{X}_j^{k+1} - \mathbb{X}_j^k) \Big] \geq 0 \tag{A.9}$$

On the one hand, we know that:

$$\sum_{i=1}^{n} \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[\sum_{j \in P(i)} \boldsymbol{\lambda}_{ji}^{k+1} + \sum_{j \in S(i)} -\boldsymbol{\lambda}_{ij}^{k+1} \Big]$$

$$= -\Big[([A] \otimes \mathbf{I}_{d \times d}) \text{vec}(\mathbf{X}^T - (\mathbf{X}^{k+1})^T) \Big]^T \boldsymbol{\lambda}^{k+1}$$

$$= -\text{vec}\big([A(\mathbf{X} - \mathbf{X}^{k+1})]^T\big)^T \boldsymbol{\lambda}^{k+1} \tag{A.10}$$

And on the other hand, we have:

$$\sum_{i=1}^{n} \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[\sum_{j \in S(i)} \beta(\mathbf{X}_j^{k+1} - \mathbb{X}_j^k) \Big]$$

$$= \sum_{i=1}^{n} \big(\mathbf{X}_i - \mathbf{X}_i^{k+1}\big)^T \Big[\sum_{j \in S(i)} \beta(-[\mathbf{B}]^{e_{ij}} \otimes \mathbf{1}_{1\times d}) \text{vec}\big([\mathbf{X}^{k+1} - \mathbf{X}^k]^T\big) \Big]$$

$$= \beta \text{vec}\big([\mathbf{X} - \mathbf{X}^{k+1}]^T\big)^T \Big(\big[(-\mathbf{A} + \mathbf{B})^T \mathbf{B}\big] \otimes \mathbf{I}_{d\times d} \Big) \text{vec}\big([\mathbf{X}^{k+1} - \mathbf{X}^k]^T\big)$$

$$= \beta \text{vec}\big([\mathbf{X} - \mathbf{X}^{k+1}]^T\big)^T \text{vec}\Big(\big[(-\mathbf{A} + \mathbf{B})^T \mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)\big]^T \Big) \tag{A.11}$$

Therefore, using A.9, A.10 and A.11 the proof of A.2.1 would be complete:

$$\sum_{i=1}^{n} f_i(\mathbf{X}_i) - \sum_{i=1}^{n} f_i(\mathbf{X}_i^{k+1}) - \text{vec}\Big(\big[\mathbf{A}(\mathbf{X} - \mathbf{X}^{k+1})\big]^T \Big)^T \boldsymbol{\lambda}^{k+1}$$

$$+\beta \text{vec}\Big((\mathbf{X} - \mathbf{X}^{k+1})^T \Big)^T \text{vec}\Big(\big[(-\mathbf{A} + \mathbf{B})^T \mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)\big]^T \Big) \geq 0$$

A.2.2 Lemma 2

Let $\{\mathbf{X}^k, \boldsymbol{\lambda}^k\}$ be the iterates generated by the DADMM approach for problem 3.3. The matrix $\mathbf{X}^k = \big(\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ \mathbf{X}_1 & \mathbf{X}_2 & \dots & \mathbf{X}_N \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \big)^T$ and vector $\boldsymbol{\lambda}^k = [\boldsymbol{\lambda}_{ij}^k]_{ij, e_{ij}\in E}$. Then the below relation holds for all k:

$$2\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T (\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star) + 2\beta \text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big)$$

$$+2\text{vec}\big([\mathbf{B}(\mathbf{X}^\star - \mathbf{X}^{k+1})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big) \tag{A.12}$$

$$= \frac{1}{\beta}\big(||\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^\star||^2 - ||\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star||^2\big) + \beta\Big(||\text{vec}\big([\mathbf{B}(\mathbf{X}^k - \mathbf{X}^\star)]^T\big)||^2 - ||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)||^2 \Big)$$

$$-\beta ||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big) - \text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)||^2 \tag{A.13}$$

Proof. We know that $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k - \beta \text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)$. Therefore, $\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big) = \frac{1}{\beta}(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)$. In addition, we know that $||\mathbf{a} + \mathbf{b}||^2 = ||\mathbf{a}||^2 + ||\mathbf{b}||^2 + 2\mathbf{a}^T\mathbf{b}$ and $||\mathbf{a}||^2 - ||\mathbf{b}||^2 = (\mathbf{a} - \mathbf{b})^T(\mathbf{a} + \mathbf{b})$. By rewriting A.12 and A.13 using the latter two mentioned equations, we can prove A.2.2.
By rewriting A.12 using above formulas we obtain:

$$\frac{2}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T (\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star) + 2\beta \text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big)$$

$$+2\text{vec}\big([\mathbf{B}(\mathbf{X}^\star - \mathbf{X}^{k+1})]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star) - \mathbf{B}(\mathbf{X}^k - \mathbf{X}^\star)]^T\big)$$

Which is equal to:

$$= \frac{2}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star) \tag{A.14}$$

$$+2\beta\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big) \tag{A.15}$$

$$-2\beta\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big) \tag{A.16}$$

$$+2\beta\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^k - \mathbf{X}^\star)]^T\big) \tag{A.17}$$

By rewriting different parts of the above equation we can reach the right-hand side of the A.2.2 which is A.13.
Equation A.16 equals:

$$-\beta||\text{vec}\big(\mathbf{B}[\mathbf{X}^{k+1} - \mathbf{X}^\star]^T\big)||^2 - \beta\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big) \tag{A.18}$$

And equation A.14 equals:

$$= \frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star) + \frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^\star)$$

$$-\frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^\star) + \frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star)$$

$$= \frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^{k+1} + \boldsymbol{\lambda}^k - 2\boldsymbol{\lambda}^\star) - \frac{1}{\beta}(\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^{k+1})^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)$$

$$= \frac{1}{\beta}\big(||\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^\star||^2 - ||\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star||^2\big) - \beta\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big) \tag{A.19}$$

Using A.18 and A.19, A.12 changes into :

$$\frac{1}{\beta}\big(||\boldsymbol{\lambda}^k - \boldsymbol{\lambda}^\star||^2 - ||\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^\star||^2\big) - \beta||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)||^2$$

$$+2\beta\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^k)]^T\big) - \beta\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)^T\text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big) \tag{A.20}$$

$$-\beta\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big) + 2\beta\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1} - \mathbf{X}^\star)]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^k - \mathbf{X}^\star)]^T\big) \tag{A.21}$$

The third line of the above equation, known as A.21 equals:

$$-\beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big) + \beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)$$

$$= -\beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{K})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big)$$

$$+\beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{k+1})]^T\big) + \beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)$$

$$= -\beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{K})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big) + \beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)$$

$$= -\beta \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big) + \beta ||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)||^2$$

Replacing A.21 with the above equation would lead to:

$$= \frac{1}{\beta}\big(||\boldsymbol{\lambda}^{k}-\boldsymbol{\lambda}^{\star}||^2 - ||\boldsymbol{\lambda}^{k+1}-\boldsymbol{\lambda}^{\star}||^2\big) + \beta\Big(||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)||^2 - ||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)||^2\Big)$$

$$-\beta\Bigg(\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big) - 2\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{\star})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k}-\mathbf{X}^{\star})]^T\big)$$

$$+\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big)^T \text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big)\Bigg) \tag{A.22}$$

where the last part which is in parentheses equals:

$$-\beta||\text{vec}\big([\mathbf{B}(\mathbf{X}^{k+1}-\mathbf{X}^{k})]^T\big) - \text{vec}\big([\mathbf{A}\mathbf{X}^{k+1}]^T\big)||^2$$

and the proof is complete.

A.2.3 Lemma 3

Let $\{\mathbf{X}^{\star}, \boldsymbol{\lambda}^{\star}\}$ be a saddle point of the Lagrangian function defined as in 3.4. Then

$$\text{vec}\big([\mathbf{A}\mathbf{X}^{\star}]^T\big) = \mathbf{0} \tag{A.23}$$

Proof. Considering the definition of saddle point, for any multiplier $(\mathbf{X}, \boldsymbol{\lambda})$ pair where $\mathbf{X} \in \mathcal{R}^{N\times d}$ and $\boldsymbol{\lambda} \in \mathcal{R}^{Md}$, we have:

$$F(\mathbf{X}^{\star}) - \boldsymbol{\lambda}^T \text{vec}\big([\mathbf{A}\mathbf{X}^{\star}]^T\big) \leq F(\mathbf{X}^{\star}) - (\boldsymbol{\lambda}^{\star})^T \text{vec}\big([\mathbf{A}\mathbf{X}^{\star}]^T\big)$$

As the above relation holds for all $\boldsymbol{\lambda}$, $\mathbf{A}\mathbf{X}^{\star} = \mathbf{0}$.

Using the three introduced lemmas, we are going to prove the main theorem and demonstrate that the vector form DADMM approach also has the $O(\frac{1}{k})$ rate of convergence.

A.2.4 Theorem

Let $\{\mathbf{X}, \boldsymbol{\lambda}\}$ be the iterates generated by our distributed algorithm for 3.3 while $\mathbf{X}^k = (\begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ \mathbf{X}_1 & \mathbf{X}_2 & \dots & \mathbf{X}_N \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix})^T$ and vector $\boldsymbol{\lambda}^k = [\boldsymbol{\lambda}_{ij}^k]_{ij,e_{ij}\in E}$.
Let matrix $\mathbf{A}$ be the edge-node incidence matrix of the network and matrix $\mathbf{B} = \min 0, \mathbf{A}$. Let $\mathbf{Y}^k = \frac{1}{k}\sum_{s=0}^{k-1}\mathbf{X}^s$ be the ergodic average of $\mathbf{X}^k$ up to time t. Then the following relation holds for all t:

$$\mathbf{0} \leq L(\mathbf{Y}^k, \boldsymbol{\lambda}^\star) - L(\mathbf{X}^\star, \boldsymbol{\lambda}^\star) \leq \frac{1}{k}\Big(\frac{1}{2\beta}||\boldsymbol{\lambda}^0 - \boldsymbol{\lambda}^\star||^2 + \frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^0 - \mathbf{X}^\star)]^T\big)||^2\Big) \tag{A.24}$$

Proof. The first inequality is true using the definition of saddle point of the Lagrangian function.
So we are going to prove the second inequality. We have to start with A.1 from Lemma 1 and set $\mathbf{X}^\star$ to variable $\mathbf{X}$. For all the iterations we have:

$$F(\mathbf{X}^\star) - F(\mathbf{X}^{s+1}) - \text{vec}\big([\mathbf{A}(\mathbf{X}^\star - \mathbf{X}^{s+1})]^T\big)^T\boldsymbol{\lambda}^{s+1}$$

$$+\beta\text{vec}\big([\mathbf{X}^\star - \mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([(-\mathbf{A}+\mathbf{B})^T\mathbf{B}(\mathbf{X}^{s+1} - \mathbf{X}^s)]^T\big) \geq \mathbf{0} \tag{A.25}$$

We have shown the s^{th}iteration above. Knowing that $\text{vec}\big([\mathbf{A}\mathbf{X}^\star]^T\big) = \mathbf{0}$ due to feasibility of the optimal solution $\mathbf{X}^\star$, the above relation would be:

$$F(\mathbf{X}^\star) - F(\mathbf{X}^{s+1}) - \text{vec}\big([-\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^{s+1} + \beta\text{vec}\big([\mathbf{X}^\star - \mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([-\mathbf{A}^T\mathbf{B}(\mathbf{X}^{s+1} - \mathbf{X}^s)]^T\big)$$

$$+\beta\text{vec}\big([\mathbf{X}^\star - \mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}^T\mathbf{B}(\mathbf{X}^{s+1} - \mathbf{X}^s)]^T\big)$$

$$= F(\mathbf{X}^\star) - F(\mathbf{X}^{s+1}) + \text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^{s+1}$$

$$-\beta\text{vec}\big([\mathbf{X}^\star - \mathbf{X}^{s+1}]^T\big)^T(\mathbf{A}\otimes\mathbf{I})^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1} - \mathbf{X}^s)]^T\big)$$

$$+\beta\text{vec}\big([\mathbf{X}^\star - \mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}^T\mathbf{B}(\mathbf{X}^{s+1} - \mathbf{X}^s)]^T\big)$$

$$= F(\mathbf{X}^\star) - F(\mathbf{X}^{s+1}) + \text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^{s+1}$$

$$-\beta\text{vec}\big([\mathbf{A}(\mathbf{X}^\star-\mathbf{X}^{s+1})]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)+\beta\text{vec}\big([\mathbf{X}^\star-\mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}^T\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)$$

$$=F(\mathbf{X}^\star)-F(\mathbf{X}^{s+1})+\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^{s+1}$$

$$+\beta\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)+\beta\text{vec}\big([\mathbf{X}^\star-\mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}^T\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)$$

Using A.25, the above equation and adding $\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^\star$ to both sides of the inequality we obtain:

$$F(\mathbf{X}^\star)-F(\mathbf{X}^{s+1})+\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\boldsymbol{\lambda}^\star+\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T(\boldsymbol{\lambda}^{s+1}-\boldsymbol{\lambda}^\star)$$

$$+\beta\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)+\beta\text{vec}\big([\mathbf{X}^\star-\mathbf{X}^{s+1}]^T\big)^T\text{vec}\big([\mathbf{B}^T\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)\geq 0$$

Now we can use Lemma A.2.2 and reform the equation to be:

$$F(\mathbf{X}^\star)-F(\mathbf{X}^{s+1})+(\boldsymbol{\lambda}^\star)^T\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)+\frac{1}{2\beta}||\boldsymbol{\lambda}^s-\boldsymbol{\lambda}^\star||^2$$

$$+\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^s-\mathbf{X}^\star)]^T\big)||^2\geq\frac{1}{2\beta}||\boldsymbol{\lambda}^{s+1}-\boldsymbol{\lambda}^\star||^2+\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^\star)]^T\big)||^2$$

$$+\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)-\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)||^2 \tag{A.26}$$

For all s. Therefore, by summing over $s=0,\dots,k-1$ we reach:

$$kF(\mathbf{X}^\star)-\sum_{s=0}^{k-1}F(\mathbf{X}^{s+1})+(\boldsymbol{\lambda}^\star)^T\sum_{s=0}^{k-1}\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)+\frac{1}{2\beta}||\boldsymbol{\lambda}^0-\boldsymbol{\lambda}^\star||^2$$

$$+\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^0-\mathbf{X}^\star)]^T\big)||^2\geq\frac{1}{2\beta}||\boldsymbol{\lambda}^k-\boldsymbol{\lambda}^\star||^2+\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^k-\mathbf{X}^\star)]^T\big)||^2$$

$$+\sum_{s=0}^{k-1}\frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^{s+1}-\mathbf{X}^s)]^T\big)-\text{vec}\big([\mathbf{A}\mathbf{X}^{s+1}]^T\big)||^2 \tag{A.27}$$

As we know, the function F is convex. Considering the definition of $\mathbf{Y}^k$, $\sum_{s=0}^{k-1}F(\mathbf{X}^s)\geq kF(\mathbf{Y}^k)$. Therefore,

$$\sum_{s=0}^{k-1}F(\mathbf{X}^s)\geq kF(\mathbf{Y}^k)$$

So,

$$kF(\mathbf{X}^\star) - kF(\mathbf{Y}^k) + k(\boldsymbol{\lambda}^\star)^T \text{vec}\big([\mathbf{A}\mathbf{Y}^k]^T\big)$$

$$+\frac{1}{2\beta}||\boldsymbol{\lambda}^0 - \boldsymbol{\lambda}^\star||^2 + \frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^0 - \mathbf{X}^\star)]^T\big)||^2 \geq 0$$

By multiplying both sides of the equation by $\frac{-1}{k}$ we reach:

$$F(\mathbf{Y}^k) - F(\mathbf{X}^\star) - (\boldsymbol{\lambda}^\star)^T \text{vec}\big([\mathbf{A}\mathbf{Y}^k]^T\big)$$

$$\leq \frac{1}{k}\Big(\frac{1}{2\beta}||\boldsymbol{\lambda}^0 - \boldsymbol{\lambda}^\star||^2 + \frac{\beta}{2}||\text{vec}\big([\mathbf{B}(\mathbf{X}^0 - \mathbf{X}^\star)]^T\big)||^2\Big)$$

Using $(\boldsymbol{\lambda}^\star)^T \text{vec}\big(\mathbf{A}\mathbf{X}^\star\big) = 0$ and the definition of Lagrangian function results in the anticipated relation of Theorem A.2.4. ■

In summary, since function F is strictly convex and $(\boldsymbol{\lambda}^\star)^T \text{vec}\big(\mathbf{A}\mathbf{X}\big)$ is a linear term; therefore, the Lagrangian is also strictly convex. Thus, $L(\mathbf{X}, \boldsymbol{\lambda}^\star)$ has a unique minimizer which is $\mathbf{X}^\star$ regarding the saddle point assumption.
In the last resort, having proven theorem A.2.4, the value of the Lagrangian function converges to $L(\mathbf{X}^\star, \boldsymbol{\lambda}^\star)$ at the rate of $O(\frac{1}{k})$ using the ergodic average sequence $\mathbf{Y}^k$.