Knowledge sharing: From atomic to parametrised context and shallow to deep models

Yongxin Yang

August 2017

Submitted in partial fulfilment of the requirements of the Degree of Doctor of Philosophy

I, Yongxin Yang, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature: YONGXIN YANG

Date: 13/06/2017

The Texts of Chapters

Chapter 3, in part, is a reprint of the material as it appears in Proceedings of the 3rd International Conference on Learning Representations (ICLR) 2015. "A Unified Perspective on Multi-Domain and Multi-Task Learning". Yongxin Yang and Timothy Hospedales.

Chapter 4, in part, is a reprint of the material as it appears as a book chapter of Domain Adaptation in Computer Vision Applications (Springer Series: Advances in Computer Vision and Pattern Recognition, Edited by Gabriela Csurka). "Unifying Multi-Domain Multi-Task Learning: Tensor and Neural Network Perspectives". Yongxin Yang and Timothy Hospedales.

Chapter 5, in part, is a reprint of the material as it appears in Proceedings of the 5th International Conference on Learning Representations (ICLR) 2017. "Deep Multi-task Representation Learning: A Tensor Factorisation Approach". Yongxin Yang and Timothy Hospedales.

Chapter 6, in part, is a reprint of the material as it appears in the workshop track of ICLR 2017. "Trace Norm Regularised Deep Multi-Task Learning". Yongxin Yang and Timothy Hospedales.

Full Publication List

Conference

Yongxin Yang and Timothy Hospedales. Deep Multi-task Representation Learning: A Tensor Factorisation Approach. In *International Conference on Learning Representations (ICLR)*, 2017.

Yongxin Yang, Yu Zheng, and Timothy Hospedales. Gated Neural Networks for Option Pricing: Rationality by Design. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Yongxin Yang and Timothy Hospedales. Multivariate Regression on the Grassmannian for Predicting Novel Domains. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Qian Yu*, Yongxin Yang*, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-Net that Beats Humans. In *British Machine Vision Conference (BMVC)*, 2015.

Yongxin Yang and Timothy Hospedales. A Unified Perspective on Multi-Domain and Multi-Task Learning. In *International Conference on Learning Representations (ICLR)*, 2015.

Yanwei Fu, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Shaogang Gong. Transductive Multi-label Zero-shot Learning. In *British Machine Vision Conference (BMVC)*, 2014.

Zhiyuan Shi, Yongxin Yang, Timothy Hospedales, and Tao Xiang. Weakly Supervised Learning of Objects, Attributes and their Associations. In *European Conference on Computer Vision (ECCV)*, 2014.

Journal

Mingying Song, Ali Karatutlu, Isma Ali, Osman Ersoy, Yun Zhou, Yongxin Yang, Yuanpeng Zhang, William R. Little, Ann P. Wheeler, and Andrei V. Sapelkin. Spectroscopic super-resolution fluorescence cell imaging using ultra-small Ge quantum dots. In *Optics Express*, 2017.

Zhiyuan Shi, Yongxin Yang, Timothy Hospedales, and Tao Xiang. Weakly-Supervised Image Annotation and Segmentation with Objects and Attributes. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016.

Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy Hospedales. Sketch-a-Net: A Deep Neural Network that Beats Humans. In *International Journal of Computer Vision (IJCV)*, 2016.

Abstract

Key to achieving more effective machine intelligence is the capability to generalise knowledge across different contexts. In this thesis, we develop a new and very general perspective on knowledge sharing that unifies and generalises many existing methodologies, while being practically effective, simple to implement, and opening up new problem settings.

Knowledge sharing across tasks and domains has conventionally been studied disparately. We first introduce the concept of a semantic descriptor and a flexible neural network approach to knowledge sharing that together unify multi-task/multi-domain learning, and encompass various classic and recent multi-domain learning (MDL) and multi-task learning (MTL) algorithms as special cases.

We next generalise this framework from single-output to multi-output problems and from shallow to deep models. To achieve this, we establish the equivalence between classic tensor decomposition methods, and specific neural network architectures. This makes it possible to implement our framework within modern deep learning stacks. We present both explicit low-rank, and trace norm regularisation solutions.

From a practical perspective, we also explore a new problem setting of zero-shot domain adaptation (ZSDA) where a model can be calibrated solely based on some abstract information of a new domain, e.g., some metadata like the capture device of photos, without collecting or labelling the data.

Contents

1 Introduction **9**

1.1 Machine Learning . . . 9

1.2 Knowledge sharing in machine learning models . . . 10

1.2.1 Multi-Task Learning . . . 11

1.2.2 Multi-Domain Learning . . . 11

1.3 Contributions of Thesis . . . 13

1.4 Organisation of Thesis . . . 14

2 Related Work **15**

2.1 Linear Regression . . . 15

2.1.1 A numerical issue and an engineering trick . . . 16

2.1.2 A regularisation-based solution . . . 16

2.1.3 An equivalent Bayesian perspective . . . 17

2.2 An introduction Multi-Task Learning . . . 18

2.2.1 Single-Task Learning . . . 18

2.2.2 Additive MTL model . . . 19

2.2.3 Multiplicative MTL model . . . 21

2.3 Literature Review . . . 23

2.3.1 Multi-Task Learning . . . 24

2.3.2 Multi-Domain Learning . . . 26

2.3.3 Zero-Shot Learning and Zero-Shot Domain Adaptation . . . 27

2.3.4 Heterogeneous MTL and Deep MTL . . . 27

2.4 The Intra- and Inter- Connections . . . 28

2.4.1 Intra-Connections . . . 28

2.4.2 Inter-Connections . . . 29

3 Single Output **32**

3.1 Background . . . 32

3.2 Methodology . . . 33

3.2.1 General Framework . . . 33

3.2.2 Semantic Descriptor Design . . . 34

3.2.3 Unification of Existing Algorithms . . . 36

3.2.4 Learning Settings 37
3.2.5 Connection to Multilinear MTL 38
3.3 Experiments 39
3.3.1 School Dataset - MDL and ZSDA 40
3.3.2 Audio Recognition - MDL and ZSDA 40
3.3.3 Animal with Attributes - MTL and ZSL 41
3.3.4 Restaurant & Consumer Dataset - MDMT 42
3.4 Summary 43

4 Multi Output 44
4.1 Background 44
4.2 Methodology 45
4.2.1 Formulation 46
4.2.2 Tensor Decomposition 46
4.2.3 Gated Neural Network Architectures 49
4.2.4 Zero-Shot Domain Adaptation 50
4.3 Experiments 51
4.3.1 Surveillance Image Classification 51
4.3.2 Gait-based Soft-Biometrics and Recognition 53
4.3.3 Multi-domain Multi-task Object Recognition 55
4.4 Summary 56

5 Going Deep: Tensor Factorisation for Deep MTL 58
5.1 Background 58
5.2 Methodology 60
5.2.1 Tensor Factorisation for Knowledge Sharing 61
5.2.2 Deep Multi-Task Representation Learning 61
5.3 Experiments 62
5.3.1 Homogeneous MTL 64
5.3.2 Heterogeneous MTL: Face Analysis 66
5.3.3 Heterogeneous MTL: Multi-Alphabet Recognition 67
5.3.4 Multi-Domain Learning 69
5.4 Summary 71

6 Tensor Trace Norm Regularised Deep MTL 72
6.1 Background 72
6.2 Methodology 73
6.2.1 Tensor Tensor Norm 73
6.2.2 Optimisation 74
6.3 Experiment 75
6.3.1 Omniglot 75
6.4 Summary 77

7 Conclusion and outlook **78**
7.1 Parametrised DNNs 78
7.2 Unlabelled Tasks/Domains 79
7.3 Limitations and Future Works 79

Appendices **80**

A Notations **80**
A.1 Variables 80
A.2 Indexing 80
A.3 Slicing and Stacking 81
A.4 Operators 81
A.4.1 Product operators 81
A.4.2 Norm functions 81
A.4.3 Matrix operators 81
A.4.4 Tensor operators 82

Acknowledgement

First, I would like to express my sincere gratitude to my supervisor, Timothy Hospedales, for his support during my PhD. I am, semi-officially, Tim's first PhD student. I guess it is *particularly* worth celebrating that we make it because (i) Tim is a professional dancer who happens to do research as a formal job, and (ii) I am a professional accountant who accidentally slips into academia. I would also like to thank my colleagues, in particular, Yun Zhou, Mingying Song, Zhiyuan Shi, Yanwei Fu, Qian Yu, Guosheng Hu, and Yu Zheng.

Second, I want to thank my parents – I started feeling how hard it is to raise up a kid. It is very challenging to take care of a kid like me (or my daughter).

Finally, and most importantly, I would like to thank my wife, Qing Li, I can not survive without her – she keeps me fed, fat and "happy". I love her forever.

Last but not least, my ten-month-old daughter Nina helped me type the last symbol(s) of this sentence7 m7u w zfrm-e ;fs vszcx

Chapter 1

Introduction

1.1 Machine Learning

Machine learning [Hastie et al., 2001, Bishop, 2006, Barber, 2012, Murphy, 2012] is an interdisciplinary field of computer science, statistics, mathematics (esp. optimisation), and neuroscience. While a formal definition of machine learning leads to a philosophical problem: can a machine *learn*? we sidestep the philosophical question and introduce machine learning in a more straightforwardly quantifiable way.

Think about making a computer program that answers the following question: what is it in a picture, an apple or a banana? This program is expected to be fed an image and output a correct label (either apple or banana).

A naive implementation is simply detecting the colour, if (colour==red) {return apple} else {return banana}. We denote this approach as *explicit programming*, because we directly code the solution based on our own knowledge in a way that a computer can execute it. We can continue to improve this program by adding more rules (if-else statements), e.g., if (colour==red and shape==round) {return apple} else {return banana}.

We can tell that the above program does not learn anything, as it just follows our instructions. Instead, machine learning is looking for an alternative way: we present a number of apple images as well as banana images, and let the program itself figure out how to distinguish them by *training* it. We denote this approach as *data driven programming*. While it sounds a bit magic without sufficient details on how training is achieved, it turns out that machine learning is particularly good at this kind of tasks and eventually outperforms humans in many cases [He et al., 2015, Yu et al., 2015, Lu and Tang, 2015].

In summary, machine learning is the study of how to *train* a model (the program) so that it can solve the problem that we, humans, can do. Conventionally machine learning is categorised into three different settings, namely,

Supervised learning The objective is to find the mapping between the given pairs of (input, output). There is ground truth for the output (true labels). Most recognition problems and those involved with predictions fall into this category.

Unsupervised learning It finds the underlying structure of the given data by transforming it into another representation. There is no right answer in this case. Dimension reduction and clustering are typical studies in unsupervised learning.

Reinforcement learning An agent is trying to maximise its rewards in an environment by taking actions in different states. Reinforcement learning is about how to pick the best action for the agent.

We point out that this kind of taxonomy, though being well accepted, may be kind of misleading because overlap exists in those categories. For example, autoencoder [Bourlard and Kamp, 1988], a classic unsupervised learning method, can be seen supervised learning where the output is exactly the input: it "compresses" the data and "reconstructs" them. The objective of autoencoder is usually to minimise the reconstruction error, which is of the same form as regression. Another example is policy gradient [Sutton et al., 1999], a popular reinforcement learning algorithm, is essentially a supervised learning approach with dynamical labels and instance-wise re-weightings for each round of training. On the other hand, some machine learning problems are hard to fit into any of three categories, e.g., semi-supervised learning is a bridge between supervised and unsupervised learning.

1.2 Knowledge sharing in machine learning models

One typical practice of machine learning is (i) collect the data (ii) train the model, and (iii) deploy it. When a new problem comes, the practitioner usually starts again from scratch, and the existing model becomes useless. This process differs from humans significantly, as humans rarely starts from nothing, instead they can leverage the experience in past, and master a new skill quickly. For example, it is easy to teach a kid to recognise digit 8 given that (s)he knew how to recognise 0, as it appears that 8 is just two stacked 0s. Humans are extremely good at building the connections between a new task and the tasks that they have seen, thus they can reuse their knowledge. It however is very hard for machines to do so.

Back to the problem of recognising digits, assuming we want to train a model that recognise 0, we collect and annotate a number of images that are zeros (labelled as "is zero") and another set of images that are not zeros (labelled as "is not zero"), from which we can train a model. Though the trained model can do an excellent job on recognising 0, it is useless when we switch the problem into recognising 8: to deal with this new problem, we need to do it again: get data and train the model.

One can easily tell that the cost of dealing with the new problem is (almost) the same as the old one, but it should not be. The problem here is that we completely abandoned the existing model.

The idea of storing the knowledge learned from previous problems and applying it to a different but related problem is studied as *transfer learning* in machine learning literature. In this subsection, we briefly introduce two sub-fields of transfer learning, namely, multi-task learning (MTL) and multi-domain learning (MDL), in a non-technical fashion. A rigorous review can be found in Chapter 2.

1.2.1 Multi-Task Learning

Before introducing multi-task learning, it is useful to carefully define the term *task* in the context of this thesis. A task roughly corresponds to a problem, e.g., in a recognition system, it is a distinct class: recognising a cat is a task, and recognising a dog is another (different but related) task.

Multi-task Learning [Caruana, 1997], in general, suggests a strategy that can potentially benefit from jointly learning multiple tasks instead of treating them independently. Recent survey on this topic can be found in [Ruder, 2017, Zhang and Yang, 2017].

Besides of the classic settings of MTL, there are paradigms involving with multiple tasks, e.g., one-shot learning and zero-short learning. Here we briefly introduce these topics due to the connections with MTL.

One-Shot Learning

Theoretically, the minimal requirement for training a machine learning model is one positive example. This is intuitive, to recognise a cat, we should at least present *one* image of cat. The study for this extreme case is usually called *one-shot learning* [Fei-Fei et al., 2006], and it assumes the existence of a system that has already been able to recognise many classes, and the target is to extend the system such that it can recognise a new class with only one example. To achieve this, a mechanism that reuses the knowledge from previously learned classes is the key.

Zero-Shot Learning

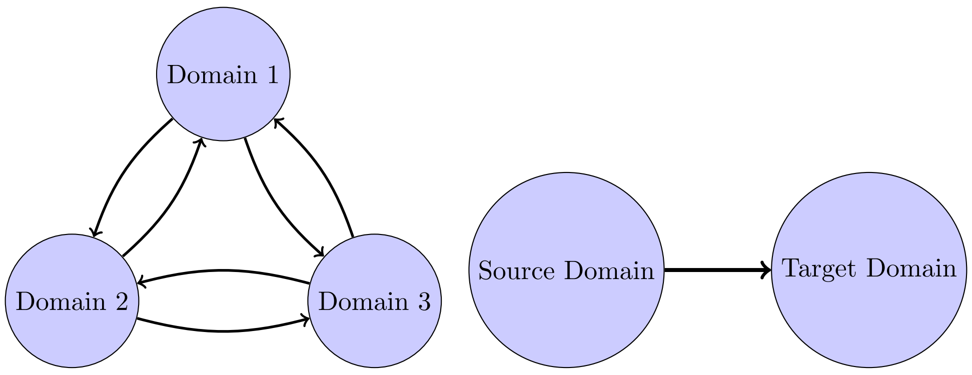
If we take one step further from one-shot learning, the only example is not in the form of image, but more abstract, e.g., a semantic description, we reach a research area called *zero-shot learning* [Lampert et al., 2009]. This is achievable for humans, for example, we can teach kids how to recognise a unicorn without presenting any unicorn images, instead we simply tell them unicorn is a horse with horn on its head (assuming that they knew horse and horn). Zero-shot learning leads to a possibility that constructs a model on-the-fly *without* training.

1.2.2 Multi-Domain Learning

The term *domain* is sometimes confused with *task* that we discussed in the previous section. In this thesis, a domain roughly refers to a dataset, or more specifically, the data distribution where a model is trained. A common assumption made in conventional supervised learning is that the training data (from which a model is obtained) and the testing data (to which a model is deployed) are drawn from the same distribution, when this assumption is not held, the model performance can be significantly degraded. For example, a face recognition system trained in the collection of front faces may not work well when it is used for face images in extreme poses.

The problem setting of multi-domain learning [Daumé III, 2007] is similar to multi-task learning, and the difference is that, the distinction of domains originates from different data distributions (sometimes it is referred to as *domain shift* or *domain bias*). In contrast, for multi-task learning, the distinction of tasks originates from different objectives of tasks. The distinction between multi-task learning and multi-domain learning applies to the context of this thesis only, and for most of MTL algorithms, they can solve MDL setting seamlessly.

Domain Adaptation



(a) Multi-Domain Learning (b) Domain Adaptation

Figure 1.1: Two different settings involving with more than one domains

Multi-domain learning is illustrated Fig. 1.1(a), where multiple domains are learned jointly during training and the knowledge sharing happens in every pair of domains. But a more common problem setting is that, we have a source domain where a model can be trained, and a target domain for which the model is applied. The data distribution of target domain is related but not identical to the source domain, so we want to *calibrate* the model so that the model can perform well on the target domain. An option is to re-train the model on the target domain regardless of the existing model, but it costs an effort to collecting sufficient data for retraining. Domain adaptation (Fig. 1.1(b)) studies adapting the existing model to the target domain with minimal effort. The knowledge sharing in domain adaptation happens in a single direction: source domain $\rightarrow$ target domain.

Two main settings of domain adaptation are,

Supervised Domain Adaptation Both source and target domain are labelled, but the data volume of target domain is much smaller than source domain thus it is impractical to train a model on it solely.

Unsupervised Domain Adaptation Source domain is labelled, but target domain is unlabelled. I.e., we can not obtain the conditional distribution $P(\text{Label}|\text{Data})$ for the target domain, thus the option left is to align the marginal distribution of target domain $P(\text{Target Data})$ with that of source data $Q(\text{Source Data})$.

Zero-Shot Domain Adaptation

To realise domain adaptation, it appears that we at least need some unlabelled data from the target domain.

Inspired by zero-shot learning, we propose a novel problem setting, instead of collecting some data instances of the target domain, we study a scenario that we have some high-level descriptions (e.g., meta-data like capture devices, lighting conditions.) of the target domain only, and we want to calibrate the existing model on-the-fly without training. We denote this new problem setting as zero-shot domain adaptation (ZSDA) [Yang and Hospedales, 2015, 2016].

An alternative to ZSDA is to train a domain-invariant model, such that it can apply for any unseen domains. This topic is usually called domain generalisation [Muandet et al., 2013, Ganin and Lempitsky, 2015]. Domain generalisation seems to be appealing as it does not even need the meta-data of target domain, but if such meta-data is available, ZSDA can potentially exploit it to outperform domain generalisation.

1.3 Contributions of Thesis

In the following parts of this thesis, we will present a line of work in multi-task/multi-domain learning, and the key contributions are,

A general framework encompassing classic methods It provides insights about many existing methods by draw connections between them (e.g., additive or multiplicative MTL) and our general framework.

Atomic to parametrised domain/task By introducing the semantic descriptor, we take the domain or task meta-data as an input directly to the model formulation, rather than just use it as a means to distinguishes different domains. This provides a new viewpoint on modelling this kind of problems.

Shallow to deep models Deep MTL was previously studied in an empirical way, and we propose the first systematic study in this area. The softly-sharing mechanism finds a mid-road between fully-sharing and non-sharing – these two designs have to be determined by the user in a try-and-error fashion for every single layer of neural networks, instead our approach can handle this automatically. This can be seen as a deep generalisation of existing soft sharing but shallow methods as well.

1.4 Organisation of Thesis

The following part of thesis consists of six chapters,

Chapter 2 We review the classic work in the area of multi-task learning and multi-domain learning. Based on the low-level mathematical tools, we classify them into two main categories: matrix-based and tensor based. More importantly, we draw the link between those classic work with the methodologies proposed in Chapter 3, Chapter 4, Chapter 5, and Chapter 6.

Chapter 3 We start from matrix-based approach MTL/MDL, and introduce the core concept: semantic descriptor, by which we can encompass several classic methods into a unified framework, and further it enables a new problem setting – zero-shot domain adaptation. Besides, we detail the connection of the proposed approach and the tensor-based one when it comes to the case of multi-indexed tasks/domains.

Chapter 4 We extend the work in Chapter 3 so that it can work for the case when each task/domain involves multiple outputs. We generalise the equivalence of matrix factorisation and gated neural network to the equivalence of tensor factorisation and another family of gated neural networks.

Chapter 5 We extend the work in Chapter 4 so that it applies to multi-layer neural network models (deep learning [Schmidhuber, 2015, Lecun et al., 2015]). We point out that it is an automatic method to design deep MTL architecture, as a competitive choice to manual design.

Chapter 6 We propose to use a regularisation based approach to deal with the same problem in Chapter 5. The methodology in Chapter 5 is essentially explicit tensor factorisation, and the work in Chapter 6 can be seen as a continuous relaxation of it.

Chapter 7 We conclude the thesis, briefly introduce some application directions, and discuss some further directions.

Chapter 2

Related Work

In this chapter, we start from a simple linear regression problem, and sequentially introduce many important concepts in machine learning, such as parametric model, regularisation, and Bayesian prior. We aim at demonstrating that the ideas that appear to be different at the first glance may actually be the same but derived from different perspectives.

Then we introduce multi-task learning. Instead of reviewing the existing work as originally presented, we reproduce them by ourselves, may or may not follow the path of the original authors. Further, we exploit the key idea behind those methods, so we can tell that, though being formulated differently, some methods are actually built upon the same motivation.

Finally, we draw the connection of the methodology proposed in this thesis and various classic methods, and pose our work as different directions of extension of existing methods.

2.1 Linear Regression

Let $\{x_1, x_2, \dots, x_N\}$ be a set of instances, each represented by a D-dimensional vector, i.e., $x \in \mathbb{R}^D$, and $\{y_1, y_2, \dots, y_N\}$ be a set of (continuous) labels and $y \in \mathbb{R}^1$. A (single-output) regression problem seeks for a mapping function f such that the difference between $f(x)$ (prediction/estimation) and y (ground-truth) is minimised, formally,

$$\operatorname*{argmin}_{f} \sum_{n=1}^{N} \ell(f(x_n), y_n) \tag{2.1}$$

Here $\ell(\hat{y}, y)$ is a loss function that measures the error (loss) produced by a pair of prediction $\hat{y} = f(x)$ and ground-truth y, and a popular choice is squared error, i.e., $\ell(\hat{y}, y) = (\hat{y} - y)^2$.

It is less practical to look for f from all possible functions, and we usually choose $f(\cdot)$ to be a certain kind of functions. In this case, we choose $f(\cdot)$ to be a linear function parametrised by w, i.e., $f_w(x) = w^T x$ where $w \in \mathbb{R}^D$.

With the choice on $\ell(\cdot, \cdot)$ and $f(\cdot)$, we can rewrite Eq. 2.1 as,

$$\underset{w}{\operatorname{argmin}} \sum_{n=1}^{N} (w^T x_n - y_n)^2 \tag{2.2}$$

Eq. 2.2 has a closed-form solution, to see this, we first rewrite it into its matrix form. Denote X to be a stack of instances in a row-wise manner, i.e., $X = [x_1; x_2; \ldots, x_N] \in \mathbb{R}^{N \times D}$, and y to be a stack of labels, i.e., $y = [y_1; y_2; \ldots, y_N] \in \mathbb{R}^N$, and then Eq. 2.2 can be rewritten as,

$$\underset{w}{\operatorname{argmin}} ||Xw - y||_2^2 \tag{2.3}$$

Here $|| \cdot ||_2$ is the ℓ_2 norm for vector (see Sec. A.4.2 for all norm functions used in this thesis). Now we take the derivative with respect to w,

$$\frac{\partial}{\partial w} ||Xw - y||_2^2 = 2X^T(Xw - y) \tag{2.4}$$

By setting $2X^T(Xw-y) = 0$, we get the closed-form solution of w as $w = (X^T X)^{-1} X^T y$.

2.1.1 A numerical issue and an engineering trick

One may find a mistake here immediately, we implicitly use an assumption that $(X^T X)$ is invertible without verifying it. When $N < D$, $(X^T X)$ is definitely non-invertible. In fact, even for the case when $N > D$, it is not numerically safe to invert $(X^T X)$ because X itself might be a low-rank matrix, a widely used engineering trick is to add a small value to the diagonal of $(X^T X)$, so the solution becomes $(X^T X + \lambda I_D)^{-1} X^T y$, where λ is a small number, e.g., $\lambda = 10^{-5}$, and I_D is a $D \times D$ identity matrix.

2.1.2 A regularisation-based solution

The aforementioned engineering trick is widely used, and for a numerical stability perspective, it works for an obvious reason – $(X^T X + \lambda I_D)$ is definitely invertible. Now we derive the same trick from a different perspective – regularisation.

We modify the objective function in Eq. 2.2 by adding a new term $\lambda ||w||_2^2$.

$$\underset{w}{\operatorname{argmin}} ||Xw - y||_2^2 + \lambda ||w||_2^2 \tag{2.5}$$

In machine learning, this term is usually called "regularisation term" or "penalty term", and it is used to control model complexity. Now the objective function contains two parts (i) a loss term that measures how well the model fits the data and (ii) a regularisation term that measures how complex the model is. The ratio λ is a positive number that controls the balance between the loss term and the regularisation term: (i) if $\lambda = 0$, we care about the model performance measured by the fitting quality only

(ii) if $\lambda \to +\infty$, we can find that the optimal solution is that $w = 0_D$ (here 0_D is a D-dimensional all-zero vector) regardless of the training data.

The meaning of w being a all-zero vector can be interpreted two ways: (i) From modelling perspective, it implies that none of the feature is useful, since we ignore all training data, this may be the least worst thing. (ii) From predicting perspective, the model will always output zero for any instances. Note that we do not have a bias term in this model, which implies the output has been normalised with zero mean (or if we estimated the bias term when $w = 0$, we will get $b = \bar{y}$). a model that always produces zero (for normalised output) or produces the mean (for un-normalised output) is the simplest model without involving any input data.

It is easy to verity that the solution for Eq. 2.5 is the same as the engineering trick mentioned in Sec. 2.1.1.

2.1.3 An equivalent Bayesian perspective

Now we re-derive the ℓ_2 regularisation approach from a Bayesian perspective. We assume that the residual $(\hat{y} - y)$ is from a Gaussian distribution with mean 0 and standard deviation σ, the probabilistic form of Eq. 2.2 is,

$$\underset{w}{\operatorname{argmax}} \prod_{n=1}^{N} \mathcal{N}(w^T x_n - y_n | 0, \sigma) \tag{2.6}$$

To add the prior knowledge on each w_d, we impose a Gaussian prior with mean 0 and standard deviation ϕ, then Eq .2.6 becomes

$$\underset{w}{\operatorname{argmax}} \prod_{d=1}^{D} \mathcal{N}(w_d | 0, \phi) \prod_{n=1}^{N} \mathcal{N}(w^T x_n - y_n | 0, \sigma) \tag{2.7}$$

Take the logarithm of Eq. 2.7, we can get,

$$\underset{w}{\operatorname{argmax}} - \sum_{n=1}^{N} \frac{1}{\sigma^2}(w^T x_n - y_n)^2 - \frac{1}{\phi^2} \sum_{d=1}^{D} w_d^2 + \text{constant} \tag{2.8}$$

Eq. 2.8 can be rewritten as,

$$\underset{w}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{1}{\sigma^2}(w^T x_n - y_n)^2 + \frac{1}{\phi^2} \sum_{d=1}^{D} w_d^2 \to \underset{w}{\operatorname{argmin}} ||Xw - y||_2^2 + \frac{\sigma^2}{\phi^2} ||w||_2^2 \tag{2.9}$$

Eq. 2.9 is equivalent with Eq. 2.5 when $\lambda = \frac{\sigma^2}{\phi^2}$, therefore we verify that the ℓ_2 regularisation is the same as posing a zero-mean Gaussian prior on the model parameter w. This again verifies that the regularisation approach and Bayesian approach are both built upon the following core idea: we believe $w = 0$ initially unless the later coming evidences (training data) prove it wrong to a certain extent, and the hyper-parameter λ is a measure that how strongly we believe $w = 0$.

2.2 An introduction Multi-Task Learning

In this section, we give an introduction to the problem setting of multi-task learning, and re-derive two classes of classic MTL models (additive and multiplicative) from a slightly different perspective than their original presentations, in order to draw new connections between them and set the stage for our later generalisations. Note that, these MTL methods are applicable to multi-domain learning as well, but we call them MTL for simplicity.

2.2.1 Single-Task Learning

Assume we now have multiple regression problems instead of one, and these problems are different but related. To give a concrete example, we introduce *School Dataset* [Mortimore et al., 1988] that was brought to multi-task learning research by [Bakker and Heskes, 2003]. The problem setting is as follows, the objective is to predict students' exam scores based on their information, e.g., gender, ethnic group. There are 139 distinct schools, from which we build 139 regression problems (tasks).

Apart from multi-task learning, we have two approaches to modelling this problem,

Train 139 regression models independently We choose not to share any knowledge between models, and the obtained models should be school-specific. The disadvantage is clear: when some school have a small number of instances (students), it is hard to train a good model.

Train 1 regression model from aggregated data We ignore school difference, and concatenate all training instances from all schools to train a single model. The obtained model is school-agnostic. This may lead to a good model because the training data is sufficient, but it may not work well for all schools as the school bias is completely missing.

Note that, though school dataset was introduced for multi-task learning originally, it fits more with the definition of multi-domain learning in this thesis (Sec. 1.2.2), thus the second approach is valid because it actually looks for a universal model for all schools. For most multi-task learning problems, it is pointless to carry out the second approach because the tasks are essentially different, which means there will not be a universal model. Nevertheless, the task or domain distinction is subtle for school dataset, and to follow the trace of research history, we still call it a multi-task learning problem, and temporarily ignore the second approach.

Formally, we denote the first approach as single task learning (STL), also known as independent task learning (ITL). Adapting two key choices from the regression model in Sec. 2.1: linear model and squared error, the STL can be formulated as,

$$\underset{w^{(1)},w^{(2)},\dots,w^{(T)}}{\operatorname{argmin}} \sum_{i=1}^{T}\sum_{j=1}^{N^{(i)}}(w^{(i)}\cdot x_j^{(i)} - y_j^{(i)})^2 \tag{2.10}$$

Here i is the index of tasks, and $T = 139$ is the number of schools (tasks) in total. For the i-th school there are $N^{(i)}$ students (training instances). j is the index of students within the same school, and $y_j^{(i)}$ stands for the j-th student's exam score in the i-th school. Here we assume that all students are represented by D-dimensional feature vectors regardless which school they come from, thus we have $x_j^{(i)} \in \mathbb{R}^D, \forall i \in [1, 2, \dots, T]$. Consequentially we have $w^{(i)} \in \mathbb{R}^D, \forall i \in [1, 2, \dots, T]$ so that we can stack all model parameters and form a matrix $W = [w_1, w_2, \dots, w_T] \in \mathbb{R}^{D \times T}$. We slightly amend Eq. 2.10 to get its matrix form,

$$\operatorname*{argmin}_W \sum_{i=1}^{T} \sum_{j=1}^{N^{(i)}} (W_{\cdot,i} \cdot x_j^{(i)} - y_j^{(i)})^2 \to \operatorname*{argmin}_W \sum_{i=1}^{T} ||X^{(i)} W_{\cdot,i} - y^{(i)}||_2^2 \tag{2.11}$$

Here $X^{(i)}$ is an $N^{(i)} \times D$ matrix stacking all instances in the i-th task and $y^{(i)}$ is a $N^{(i)}$-length vector stacking all labels in the i-th task. It is obvious that the learning for $W_{\cdot,i}$ involves $(X^{(i)}, y^{(i)})$ only, thus each task is independent to others.

2.2.2 Additive MTL model

We illustrate the first MTL algorithm, denoted as *Additive MTL model*, and the key idea here is to add one shared term for each single model parameter such that a degree of knowledge sharing is introduced.

$$w^{(i)} \leftarrow \hat{w}^{(i)} + \hat{w}^{(0)} \tag{2.12}$$

Intuitively, the model parameter has two terms: one *task-specific* term or its own use and one *shared* term that is used by all tasks. The mechanism of MTL is as follows: the learning for $\hat{w}^{(0)}$ is affected by all $\hat{w}^{(i)}$ terms, and simultaneously $\hat{w}^{(0)}$ affects the learning for every $\hat{w}^{(i)}$, therefore $\hat{w}^{(i)}$ is no longer an isolated term, it is indirectly linked with all $\hat{w}^{(k)}, k \neq i$ terms bridged by $\hat{w}^{(0)}$. To see this in detail, we rewrite Eq. 2.11 as,

$$\operatorname*{argmin}_{\hat{W}, \hat{w}^{(0)}} \sum_{i=1}^{T} ||X^{(i)} \hat{W}_{\cdot,i} + X^{(i)} \hat{w}^{(0)} - y^{(i)}||_2^2 \tag{2.13}$$

Here $\hat{W} = [\hat{w}^{(1)}, \hat{w}^{(2)}, \dots, \hat{w}^{(T)}]$. From Eq. 2.13 we can see the gradient of $\hat{W}$ is related with $\hat{w}^{(0)}$ and vice versa.

Next, we present two ways to understand this model.

Regularisation perspective

It is easy to interpret $\hat{w}^{(0)}$ as a shared term intuitively, but one may wonder what is the physical meaning of $\hat{w}^{(0)}$ and how exactly it is involved in the learning process. [Evgeniou and Pontil, 2004] derives a perspective from regularisation, here we reproduce this work. Note that the original paper presented the methodology for a binary classification problem with hinge loss, but it is applicable to the regression here.

First, we add two ℓ_2 norms for $\hat{W}$ and $\hat{w}^{(0)}$ respectively for regularising the model in Eq. 2.13,

$$\underset{\hat{W},\hat{w}^{(0)}}{\operatorname{argmin}} \sum_{i=1}^{T} ||X^{(i)}\hat{W}_{\cdot,i} + X^{(i)}\hat{w}^{(0)} - y^{(i)}||_2^2 + \lambda_1 \sum_{i=1}^{T} ||\hat{W}_{\cdot,i}||_2^2 + \lambda_2 ||\hat{w}^{(0)}||_2^2 \tag{2.14}$$

We denote the objective function in Eq. 2.14 as $J(\hat{W}, \hat{w}^{(0)})$, when the optimal solution for Eq. 2.14 is $\hat{W}^*$ and $\hat{w}_*^{(0)}$, we have $\frac{\partial J}{\partial \hat{W}_{\cdot,i}}|_{\hat{W}_{\cdot,i}=\hat{W}^*_{\cdot,i}, \hat{w}^{(0)}=\hat{w}^{(0)}_*} = 0, \forall i \in [1, 2, \ldots, T]$ and $\frac{\partial J}{\partial \hat{w}^{(0)}}|_{\hat{W}=\hat{W}^*, \hat{w}^{(0)}=\hat{w}^{(0)}_*} = 0$. By extending these two equations, we have,

$$X^{(i)^T}(X^{(i)}\hat{W}^*_{\cdot,i} + X^{(i)}\hat{w}^{(0)}_* - y^{(i)}) + \lambda_1 \hat{W}^*_{\cdot,i} = 0 \tag{2.15}$$

$$\sum_{i=1}^{T} X^{(i)^T}(X^{(i)}\hat{W}^*_{\cdot,i} + X^{(i)}\hat{w}^{(0)}_* - y^{(i)}) + \lambda_2 \hat{w}^{(0)}_* = 0 \tag{2.16}$$

Aggregating all Eq. 2.15 for $i \in [1, 2, \ldots, T]$ gives the equation,

$$\sum_{i=1}^{T} X^{(i)^T}(X^{(i)}\hat{W}^*_{\cdot,i} + X^{(i)}\hat{w}^{(0)}_* - y^{(i)}) + \lambda_1 \sum_{i=1}^{T} \hat{W}^*_{\cdot,i} = 0 \tag{2.17}$$

By combining Eq. 2.16 and Eq. 2.17 we obtain that,

$$\hat{w}^{(0)}_* = \frac{\lambda_1}{\lambda_2} \sum_{i=1}^{T} \hat{W}^*_{\cdot,i} \tag{2.18}$$

By combining Eq. 2.12 and Eq. 2.18 we further get,

$$\hat{w}^{(0)}_* = \frac{\lambda_1}{\lambda_2 + \lambda_1 T} \sum_{i=1}^{T} w^{(i)}_* = \frac{1}{\frac{\lambda_2}{\lambda_1} + T} \sum_{i=1}^{T} w^{(i)}_* \tag{2.19}$$

Eq. 2.19 indicates that the shared term $\hat{w}^{(0)}$ is a (smoothed) average of final model parameters $w^{(i)}$. We can also find an equivalent formulation of Eq. 2.14,

$$\underset{W}{\operatorname{argmin}} \sum_{i=1}^{T} ||X^{(i)}W_{\cdot,i} - y^{(i)}||_2^2 + \frac{\lambda_1\lambda_2}{\lambda_2 + \lambda_1 T} \sum_{i=1}^{T} ||W_{\cdot,i}||_2^2 + \frac{\lambda_1^2 T}{\lambda_2 + \lambda_1 T} \sum_{i=1}^{T} \left|\left| W_{\cdot,i} - \frac{\sum_{j=1}^{T} W_{\cdot,j}}{T} \right|\right|_2^2 \tag{2.20}$$

Eq. 2.20 implies the core assumption behind this additive MTL model is that all model parameters are "close" to their empirical mean, or, from a Bayesian viewpoint, the model parameters are drawn from the Gaussian that has the mean of the empirical mean of these model parameters.

Feature augmentation perspective

Apart from above regularisation perspective, we present another view of the additive MTL model. The idea is very simple, just to *copy* the feature one more time and put them as well as zero-paddings into an augmented space [Evgeniou and Pontil, 2004, Evgeniou et al., 2005, Daumé III, 2007].

To illustrate this algorithm, we assume there are three tasks ($T = 3$): the training datasets are $\{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), (X^{(3)}, y^{(3)})\}$.

The original feature for the first task is $X^{(1)} \in \mathbb{R}^{N^{(1)} \times D}$, and we transform it into $[X^{(1)}, X^{(1)}, 0_{N^{(1)} \times D}, 0_{N^{(1)} \times D}] \in \mathbb{R}^{N^{(1)} \times 4D}$, and for the task, we have a transformed feature $[X^{(2)}, 0_{N^{(2)} \times D}, X^{(2)}, 0_{N^{(2)} \times D}] \in \mathbb{R}^{N^{(2)} \times 4D}$, and finally, for the third task, we have $[X^{(3)}, 0_{N^{(3)} \times D}, 0_{N^{(3)} \times D}, X^{(3)}] \in \mathbb{R}^{N^{(3)} \times 4D}$. By concatenating three augmented features, we have $(N^{(1)} + N^{(2)} + N^{(3)}) \times 4D$ sized training data for which we want to predict $(N^{(1)} + N^{(2)} + N^{(3)})$ labels (concatenated labels from all tasks). Again we build *one* linear model for it, and the model parameter is $w \in \mathbb{R}^{4D}$.

We can tell the model parameter w here corresponds to $[\hat{w}^{(0)}; \hat{w}^{(1)}; \hat{w}^{(2)}; \hat{w}^{(3)}]$ in the previous approach, thus these two formulations agree.

2.2.3 Multiplicative MTL model

Recall that the core of additive MTL model is $w^{(i)} \leftarrow \hat{w}^{(0)} + \hat{w}^{(0)}$, instead of *addition* operator, we present an alternative MTL approach based on *Multiplication* operator.

The core of the so-called multiplicative MTL model is,

$$w^{(i)} \leftarrow L s^{(i)} \tag{2.21}$$

Similar to additive MTL model, the model parameter has a task-specific term $s^{(i)}$ and a shared term L, but the final model parameter is produced by dot-product this time instead of sum. This approach introduces a new hyper-parameter, namely the length of $s^{(i)}$ (or the number of columns of L). One way to understand this hyper-parameter is that it is (assumed) rank of matrix $W = [w^{(1)}, w^{(2)}, \dots, w^{(T)}] \in \mathbb{R}^{D \times T}$. By stacking all $s^{(i)}$ to form a matrix $S = [s^{(1)}, s^{(2)}, \dots, s^{(T)}]$, we have $W = LS$. Replacing the model matrix W with two factor matrices L and S, we write the general form of the objective function for the multiplicative MTL model as,

$$\underset{L,S}{\text{argmin}} \sum_{i=1}^{T} ||X^{(i)} L S_{\cdot,i} - y^{(i)}||_2^2 \tag{2.22}$$

Here $L \in \mathbb{R}^{D \times K}$ and $S \in \mathbb{R}^{K \times T}$ where K is a hyper-parameter. The mechanism of MTL is clear: L will affect each individual $s^{(i)}$ and all $s^{(i)}$ terms jointly will affect L. Next we present two ways to understand this method.

Dictionary learning of W

We can see this approach as an explicit matrix factorisation of W, or more generally dictionary learning of W where L is the dictionary matrix and S is the coding matrix.

When $K \ll \min(D,T)$, the effectiveness of this approach can be explained as that it effectively reduces the number of parameters to learn from DT (single task learning mode) to $K(D+T)$ – the statement holds true when $K < \frac{DT}{D+T}$. By reducing the number of parameters to learn, it reduces the complexity of model, i.e., regularise the model.

However, the model may still work when $K \geq \frac{DT}{D+T}$ if we further regularise L and/or S. For example, we can set K sufficiently large (even larger than the maximum possible rank) while imposing the ℓ_1 norm on the columns of S such that the coding of each task will be sparse [Kumar and Daumé III, 2012], which corresponds to the over-complete case of sparse coding.

It is worth mentioning that, apart from explicitly factorising W, it is also possible to impose an appropriate matrix norm on W to achieve similar purposes, then the Eq. 2.22 becomes,

$$\underset{W}{\operatorname{argmin}} \sum_{i=1}^{T} ||X^{(i)}W_{\cdot,i} - y^{(i)}||_2^2 + \lambda\Omega(W) \tag{2.23}$$

where $\Omega(W)$ is a matrix norm function on W. Two typical papers in this line are,

To encourage sparsity along the rows of W [Argyriou et al., 2008] propose to use $\ell_{2,1}$ norm which is defined as $||W||_{2,1} = \sum_{d=1}^{D} |(\sum_{i=1}^{T} |W_{d,i}|^2)^{\frac{1}{2}}|$. We can see $\ell_{2,1}$ norm as an ℓ_1 norm on a D-dimensional vector $[||W_{1,\cdot}||_2, ||W_{2,\cdot}||_2, \cdots, ||W_{D,\cdot}||_2,$, of which each element is the ℓ_2 norm of a row in W. It is well known that ℓ_1 norm will encourage sparsity, thus the Euclidean norm of any row of W is encouraged to be zero – this means a row in W is all-zero, or more intuitively, this feature is not useful for any tasks. Therefore, this approach can be understood as a joint feature selection (lasso regression) across tasks, which selects the most useful features for all tasks. As $\ell_{2,1}$ norm favours the matrix with more all-zero rows, consequently W will be a low-rank matrix. A reduced version to [Argyriou et al., 2008], without the feature mapping, is developed in [Obozinski et al., 2010].

To encourage low-rank of W Instead of explicitly factorising W and choosing a low rank number as hyper-parameter, [Ji and Ye, 2009] propose to encourage low-rank property of W by imposing a trace norm (a.k.a., nuclear norm) on W, i.e., $\Omega(W) = ||W||_*$. Trace norm is the tightest convex relation of matrix rank [Recht et al., 2010], which makes it a good proxy when we do not want to work on rank directly as it is an NP-hard problem that we want to secure a certain (small) rank number. [Argyriou et al., 2008] propose the same formulation, but they have opted for the alternating minimisation strategy instead of optimising for trace norm directly due to the non-smoothness nature of trace norm.

L as a universally useful representation learning

Alternatively L can be seen as a linear transform that applies to all instances in all tasks universally. This viewpoint leads to a way of positioning this method within a neural network framework. First we see single task learning mode as training a number of neural network models, where each task is modelled by a two-layer neural network (one input layer, one output layer, and no hidden layer) $f_i(x) = \sigma(W_i x)$, where $\sigma(\cdot)$ is the activation function, and W_i is the weight matrix for input-to-output layer. Then we add a hidden layer with activation function $\psi(\cdot)$ so that the model for each task becomes $f_i(x) = \sigma(W_i^{(2)} \psi(W_i^{(1)} x))$, where $W_i^{(2)}$ is the weight matrix for hidden-to-output layer and $W_i^{(1)}$ is the weight matrix for input-to-hidden layer. Finally, we *tie* all input-to-hidden weight matrices, i.e, $W_1^{(1)} = W_2^{(1)} = \cdots = W_T^{(1)} = W^{(1)}$ and choose two activation functions $\sigma(\cdot)$ and $\psi(\cdot)$ to be linear activation. Now we re-discover Eq. 2.21 by observing that $W^{(1)} = L^T$ and $W_i^{(2)} = s_i^T$. Note that this kind of architecture design is sometimes referred to as "shared layer", "shared weight", or "tied weight" in some neural network literature, e.g., Siamese Network [Chopra et al., 2005].

Additive MTL model as a special case

One interesting connection is that we can see Additive MTL model in Section 2.2.2 as a special case of the multiplicative MTL model here.

By default, both L and S are the parameters to learn in Eq. 2.22, however, we can recover additive MTL model by setting S as a constant where $S = [I_T; 1_{T\times 1}] \in \mathbb{R}^{(T+1)\times T}$. We can see that, for the i-th task, its coding in S, i.e., $S_{\cdot,i}$ is a binary vector that has two units activated: the i-th one and the last one. Then we have $w^{(i)} = LS_{\cdot,i} = L_{\cdot,i} + L_{\cdot,T+1}$ where $L_{\cdot,i}$ corresponds to $\hat{w}^{(i)}$ and $L_{\cdot,T+1}$ corresponds to $\hat{w}^{(0)}$ in Eq. 2.12.

2.3 Literature Review

In this section, we review some related work in multi-task and multi-domain learning. As we briefly discussed in Section 1.2.1 and Section 1.2.2, the difference between domains and tasks could be subtle, and some multi-domain learning problems can be addressed by methods proposed for multi-task learning and vice-versa. However, to better understand the work in these two areas, it is useful to distinguish them clearly. Domains refer to multiple datasets addressing the same task, but with differing statistical bias. For example camera type for object recognition; time of day or year for surveillance video analysis; or more subtle human biases in data collection [Torralba and Efros, 2011]. Tasks, on the other hand would refer to different object categories to recognise. In other words, a task change affects the output label-space of a supervised learning problem, while a domain change does not.

A classic benchmark with multiple domains is the Office dataset [Saenko et al., 2010]. It contains images of the same set of categories (e.g., mug, laptop, keyboard) from three data sources (Amazon website, webcam, and DSLR). In this context, multi-task learning could improve performance by sharing information about how to recognise keyboard and laptop; while multi-domain learning could improve performance by sharing knowledge about how to recognise those classes in Amazon product versus webcam images. Some problems can be interpreted as either setting. E.g., in the School dataset the goal is to predict students' exam scores based on their characteristics. This dataset is widely used to evaluate MTL algorithms, where students from different schools are grouped into different tasks. However, one can argue that school groupings are better interpreted as domains than tasks.

As a rule of thumb, multi-domain learning problems occur when a model from domain A could be directly applied to domain B albeit with reduced performance; while multi-task learning problems occur where a model for task A can not meaningfully be applied to task B because their label-spaces are fundamentally different. In some problems, the multi-domain and multi-task settings occur simultaneously. E.g., in the Office dataset there are both multiple camera types, and multiple objects to recognise. Some existing methods, especially those based on tensor methods can potentially deal with this setting, but this setting is relatively less studied. Most classic MTL methods break a multi-class problem into multiple one-vs-all tasks and share information across tasks [Argyriou et al., 2008, Kumar and Daumé III, 2012], while MDL methods typically deal with a single-output problem in multiple domains [Daumé III, 2007].

2.3.1 Multi-Task Learning

Matrix-based MTL

Matrix-based MTL algorithms assume that the input and model are both D-dimensional vectors. The models of T tasks can then be stacked into a $D \times T$ sized matrix W. Despite different motivations and implementations, many matrix-based MTL methods work by placing constraints on W.

An early study [Evgeniou and Pontil, 2004] assumes a linear model for ith task can be written as $w_i \leftarrow w_0 + v_i$ where w_0 can be considered as the *shared knowledge* which benefits all tasks and v_i is the *task-specific knowledge*. A hierarchical model proposed by [Salakhutdinov et al., 2011] is similar to this motivation, where a tree-structured model for one object is generated by the sum of itself (*task-specific knowledge*) and its parents (*shared knowledge*), i.e., $w^{(\text{van})} \leftarrow w^{(\text{global})} + w^{(\text{vehicle})} + w^{(\text{van})}$.

Another common assumption of MTL is that the predictors lie in a low dimensional subspace. [Argyriou et al., 2008] imposes the $\ell_{2,1}$ norm on the predictor matrix W, where each column is a task, results in a low-rank W by forcing more rows of W to be all-zero. [Ji and Ye, 2009] places the trace norm on W that encourages the lower rank of W. However, this assumes that all tasks are related, which is likely violated in practice. Forcing predictors to be shared across unrelated tasks can significantly degrade the performance – a phenomenon called negative transfer [Rosenstein et al., 2005]. A task grouping framework is thus proposed by [Kang et al., 2011] that partitions all tasks into disjoint groups where each group shares a low dimensional structure. This partially alleviates the unrelated task problem, but misses any fundamental information shared by all tasks, as there is no overlap between the subspaces of each group.

As a middle ground, the GO-MTL algorithm [Kumar and Daumé III, 2012] allows information to be shared between different groups, by representing the model of each task as a linear combination of latent predictors. Thus the concept of grouping is no longer explicit, but determined by the coefficients of the linear combination. Intuitively, model construction can be thought of as: $W = LS$ where L is the matrix of which each column is a latent predictor (*shared knowledge*), and $S = [s_1, s_2, \dots, s_M]$ where s_i is a coefficient vector that cues how to construct the model for the ith task (*task-specific knowledge*). It is worth noting that this kind of predictor matrix factorisation approach – $W = LS$ – can explain several models: [Kumar and Daumé III, 2012] is ℓ_1/ℓ_2 regularised decomposition, [Passos et al., 2012] is linear Gaussian model with Indian Buffet Process [Griffiths and Ghahramani, 2011] prior and an earlier study [Xue et al., 2007] assumes s_i are unit vectors generated by a Dirichlet Process (DP).

Apart from learning W only, some studies suggest to model the task relations explicitly, e.g., [Lee et al., 2016, Zhang and Yang, 2017] introduce an extra parameter – a PSD matrix – to estimate the pair-wise task relatedness. This parameter is learned with the task predictors in an alternating fashion.

Classic MTL setting usually assumes that all tasks are labelled, in contrast, [Pentina and Lampert, 2017] study for the case either only some of the tasks are labelled (semi-supervised learning) or the learner has to actively select tasks for annotation (active learning).

Tensor-based MTL

Most MTL methods in literature assume that each task is an atomic entity indexed by a single categorical variable. Though it is a classic setting that each task is indexed by a single factor in MTL studies, in many real-world problems, tasks are indexed by multiple factors. For example, to build a restaurant recommendation system, we want a regression model that predicts the scores for different aspects (food quality, environment) by different customers. Then the task is indexed by aspects $\times$ customers. For this case, the collection of all linear models for all tasks is then a 3-way tensor $\mathcal{W}$ of size $D \times T_1 \times T_2$, where T_1 and T_2 are the number of aspects and the number of customers respectively. This 3-way tensor has to be flattened for matrix-based MTL to be applied, and some recent studies [Romera-Paredes et al., 2013, Wimalawarne et al., 2014] noticed the drawback of flattening is that the structural information will be lost. Thus they look for some techniques that natively apply to tensors, e.g., to impose a variety of regularisations by tensor norms, such as sum of the ranks of the matriciations[1] of the tensors [Romera-Paredes et al., 2013] and scaled latent trace norm [Wimalawarne et al., 2014].

[1]Matriciation is also known as tensor unfolding or flattening.

Besides regularisation, [Romera-Paredes et al., 2013] considers a solution based on Tucker decomposition for tensors [Tucker, 1966], where the model parameters are core tensor and factor matrices in the context of Tucker decomposition, and we generate the predictors by reconstructing the tensor from Tucker composition. In deep learning, such tensor factorisation techniques have been used to exploit factorised tensors' fewer parameters than the original (e.g., 4-way convolutional kernel) tensor, and thus compress and/or speed up the model, e.g., [Lebedev et al., 2015, Novikov et al., 2015].

An alternative solution is to concatenate the one-hot encodings of task factors and feed it as input into a two-branch neural network model [Yang and Hospedales, 2015], in which there are two input channels for feature vector and encoded task factor.

2.3.2 Multi-Domain Learning

Domain Adaptation

There has been extensive work on domain adaptation (DA) [Beijbom, 2012]. A variety of studies have proposed both supervised [Saenko et al., 2010, Duan et al., 2012] and unsupervised [Gong et al., 2012, Sun and Saenko, 2014] methods. As we have mentioned, the typical assumption is that domains are indexed by a single categorical variable: For example a data source such as Amazon/DSLR/Webcam [Saenko et al., 2010], a benchmark dataset such as PASCAL/ImageNet/Caltech [Gong et al., 2012], or a modality such as image/video [Duan et al., 2012].

Despite the majority of research with the categorical assumption on domains, it has recently been generalised by studies considering domains with a (single) continuous parameter such as time [Hoffman et al., 2014] or viewing angle [Qiu et al., 2012]. In this thesis, we take an alternative approach to generalising the conventional categorical formulation of domains, and instead investigate information sharing with domains described by a *vector* of discrete parameters.

Multi-Domain Learning

Multi-domain learning [Dredze et al., 2010, Joshi et al., 2012] is a relatively independent line of research to multi-task learning. There is close relation to domain adaptation (DA), especially supervised DA where all domains have some labelled data, e.g., [Saenko et al., 2010, Duan et al., 2012]. DA and MDL differ in their goals: with DA focusing on improving performance in a specific target domain given a model trained in a different source, and MDL focusing on simultaneously improving performance in all domains analogously to MTL. Though some MTL methods have been applied to MDL scenarios [Argyriou et al., 2008, Kumar and Daumé III, 2012], they are restricted to single-output problems.

Although some existing MTL algorithms reviewed in the previous section tackle MDL as well, we distinguish them by the key difference during testing time: MDL makes prediction for same problem (binary classification like "is laptop") across multiple domains (e.g., datasets or camera type), but MTL handles different problems (such as "is laptop" versus "is mouse").

2.3.3 Zero-Shot Learning and Zero-Shot Domain Adaptation

Zero-Shot Learning (ZSL) aims to eliminate the need for training data for a particular task. It has been widely studied in different areas, such as character [Larochelle et al., 2008] and object recognition [Lampert et al., 2009, Socher et al., 2013, Fu et al., 2014]. Typically for ZSL, the label space of training and test data are disjoint, so no data has been seen for test-time categories. Instead, test-time classifiers are constructed given some mid-level information. Although diverse in other ways, most existing ZSL methods follow the pipeline in [Palatucci et al., 2009]: $X \rightarrow Z \rightarrow Y$ where Z is some "semantic descriptor", which refers to attributes [Lampert et al., 2009] or semantic word vectors [Socher et al., 2013]. Our work can be considered as an alternative pipeline, which is more similar to [Larochelle et al., 2008] and [Frome et al., 2013] in the light of the following illustration: $\begin{tikzcd} Z \arrow[rr, bend left] & X \arrow[r] & Y \end{tikzcd}$.

Going beyond conventional ZSL, we generalise the notion of zero-shot learning of tasks to zero-shot learning of domains. In this context, zero-shot means no training data has been seen for the target domain prior to testing. The challenge is to construct a good model for a novel test domain based solely on its semantic descriptor. This is the analogous problem for domain-adaptation that zero-shot learning poses for recognition. The closest work to our zero-shot domain adaptation setting is [Ding et al., 2014], which addresses the issue of a missing modality with the help of the partially overlapped modalities that have been previously seen. However they use a single fixed modality pair, rather than synergistically exploiting an arbitrary number of auxiliary domains in a multi-domain way as in our framework. Note that despite the title, [Blitzer et al., 2009] actually considers unsupervised domain adaptation without target domain labels, but *with* target data.

2.3.4 Heterogeneous MTL and Deep MTL

Some studies consider heterogeneous MTL, where tasks may have different numbers of outputs [Caruana, 1997]. This differs from the previously discussed studies [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Bonilla et al., 2007, Jacob et al., 2009, Kumar and Daumé III, 2012, Romera-Paredes et al., 2013, Wimalawarne et al., 2014] which implicitly assume that each task has a single output.

Heterogeneous MTL typically uses neural networks with multiple sets of outputs and losses. E.g., [Huang et al., 2013] proposes a shared-hidden-layer DNN model for multilingual speech processing, where each task corresponds to an individual language. [Zhang et al., 2014] uses a DNN to find facial landmarks (regression) as well as recognise facial attributes (classification). [Liu et al., 2015] proposes a DNN for query classification and information retrieval (ranking for web search). [Arik et al., 2017] apply MTL to the text-to-speech problem by modelling both the phoneme duration and frequency profile jointly. A key commonality of these studies is that they all require a user-defined parameter sharing strategy. A typical design pattern is to use shared layers (same parameters) for lower layers of the DNN and then split (independent parameters) for the top layers. This kind of architecture design can be traced back to 2000s [Bakker and Heskes, 2003]. However, there is no systematic way to make such design choices, so researchers usually rely on trial-and-error, further complicating the already somewhat dark art of DNN design. In contrast, in Chapter 5 and Chapter 6, we propose methods that learn where and how much to share representation parameters across the tasks, hence significantly reducing the space of DNN design choices.

2.4 The Intra- and Inter- Connections

2.4.1 Intra-Connections

We draw connections for the methodologies in Chapter 3, Chapter 4, Chapter 5, and Chapter 6.

Chapter 3 focuses on the single-output case only, where each task or domain is essentially a binary classification problem or a single-output regression problem. It introduces a key concept called semantic descriptor, as a way to deal with multi-indexed task or domain, e.g., a domain is distinct from others by more than one factors. The introduced semantic descriptor is also the key to enable zero-shot learning and zero-shot domain adaptation.

Chapter 4 extends Chapter 3 to the multi-output case, where each task or domain involves multiple outputs, e.g., a domain corresponds a multi-class classification problem for a certain dataset. The core part is switch the model generating function from a setting of being parametrised by a matrix (Chapter 3) to a setting of being parametrised by a tensor. Naturally tensor factorisation becomes the key mathematical tool in Chapter 4 replacing matrix factorisation used in Chapter 3. Not surprisingly, the methodology developed in Chapter 3 is a special case of the methodology in Chapter 4 because tensor factorisation is a generalisation of matrix factorisation.

Chapter 5 considers a deep model instead of the shallow models in Chapter 3 and Chapter 4. The model block for Chapter 5 is directly borrowed from Chapter 4, but the motivation is different: the key motivation in Chapter 5 is that we want to find a data-driven way to design a deep neural network architecture for multi-task learning, which was previously carried out manually by the model designers.

Chapter 6 is a sister work of Chapter 5. Recall that the key technique in Chapter 4 and Chapter 5 is based on tensor factorisation, but in Chapter 6 we try to impose a family of tensor norms to encourage low-rank tensor rather than explicitly factorise the tensor with the chosen low-rank number(s).

To better illustrate these connections, see Fig. 2.1.

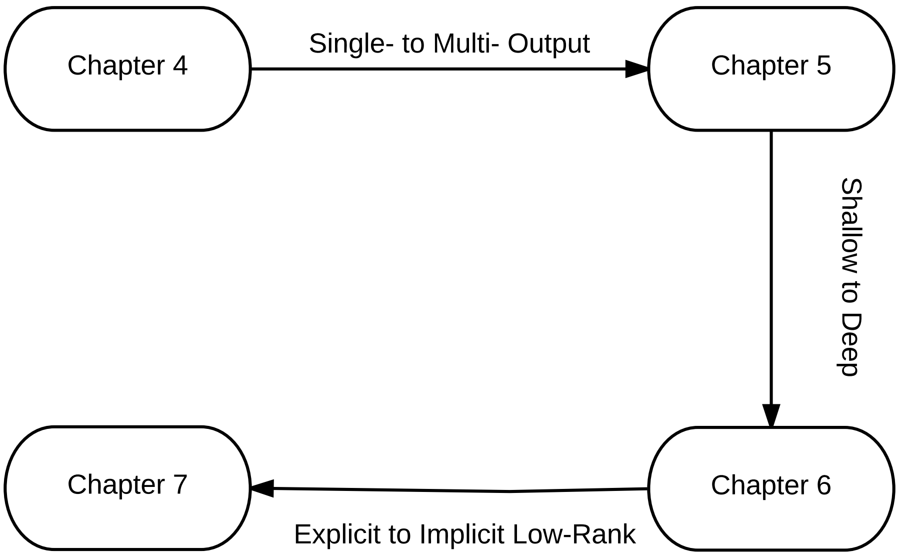


Figure 2.1: Intra-connections of methodologies in this thesis

2.4.2 Inter-Connections

We draw connections for our methodologies and existing studies. Table 2.1 shows one direction: from atomic task/domain to parametrised task/domain. Table 2.2 shows another direction: from shallow models to deep models.

Atomic-to-parametrised is illustrated in Table 2.1, we categorise the problem settings in multi-task learning by two aspects: (i) how the task is indexed: by one factor (single-index) or more than one factors (multi-index) (ii) how many outputs of an individual task: one (single-output) or many (multi-output). Therefore we have four distinct problem settings,

Single-Index, Single-Output This is the classic problem setting that is studied by the matrix-based multi-task learning. Our method in Chapter 3 encompasses many classic matrix-based MTL methods.

Multi-Index, Single-Output This is studied by tensor-based MTL. Our method in Chapter 3 can deal with case as well, but by a different strategy (see the first part of Section 3.2.5 for a detailed discussion).

Single-Index, Multi-Output This is a mirror to the setting of multi-index-single-output, as we can see the axis of outputs as yet another task axis. The view angle, i.e., it is an output unit or a task, is usually less important, and the tensor-based MTL methods naturally work for this setting. However, our method in Chapter 3 can not be used for this case directly under certain application scenarios, for example, we need to cast multi-class into multiple binary classifications (see the second part of Section 3.2.5 for a detailed discussion). To make it more flexible, we develop an extended approach in Chapter 4.

Multi-Index, Multi-Output Though not being empirically studied, tensor-based MTL methods apply to this case in principle. The method in Chapter 4 is also capable of handling this setting.

The benefits of introducing of semantic descriptor (parametrised tasks/domains), or feeding the task/domain metadata directly into the model rather than just using it on distinguishing tasks/domains are four-fold:

- It encompasses many existing methods in a unified framework.
- It can deal with multi-index case as an alternative to tensor based methods. In fact, it makes the difference on single- or -multi- index trivial.
- It enables ZSDA by a generating model parameter mechanism. This is in contrast to task imputation realised in [Romera-Paredes et al., 2013, Wimalawarne et al., 2014] as a side product of low-rank tensor assumption – tensor completion.
- It can be realised in the neural network framework which makes it simple to implement and end-to-end trainable.

Shallow-to-deep is illustrated in Table 2.2, where we classify the MTL methods by two aspects: (i) the core mathematical object is matrix or tensor, and (ii) the key technique is based on explicit factorisation or regularisation. We can see that our methods in Chapter 5 and Chapter 6 are natural extensions from shallow to deep models backed by either factorisation or regularisation.

	Atomic		Parametrised	
	Single-Index	Multi-Index	Single-Index	Multi-Index
Single-Output	[Evgeniou and Pontil, 2004], [Xue et al., 2007], [Argyriou et al., 2008], [Ji and Ye, 2009], [Kumar and Daumé III, 2012], [Passos et al., 2012]	[Romera-Paredes et al., 2013], [Wimalawarne et al., 2014]	Chapter 3	
Multi-Output	[Romera-Paredes et al., 2013], [Wimalawarne et al., 2014]	[Romera-Paredes et al., 2013], [Wimalawarne et al., 2014]	Chapter 4	

Table 2.1: From atomic to parametrised

		Explicit Factorisation	Regularisation
Shallow Model	Matrix-based	[Evgeniou and Pontil, 2004], [Xue et al., 2007], [Kumar and Daumé III, 2012], [Passos et al., 2012]	[Argyriou et al., 2008], [Ji and Ye, 2009]
	Tensor-based	MLMTL-NC of [Romera-Paredes et al., 2013]	MLMTL-C of [Romera-Paredes et al., 2013], [Wimalawarne et al., 2014]
Deep Model	Matrix-based	DMTRL-LAF in Chapter 5	TNRDMTL-LAF in Chapter 6
	Tensor-based	DMTRL-Tucker, TT in Chapter 5	TNRDMTL-Tucker, TT in Chapter 6

Table 2.2: From shallow to deep

Chapter 3

Single Output

In this chapter, we provide a new neural-network based perspective on multi-task learning (MTL) and multi-domain learning (MDL). By introducing the concept of a semantic descriptor, this framework unifies MDL and MTL as well as encompassing various classic and recent MTL/MDL algorithms by interpreting them as different ways of constructing semantic descriptors. Our interpretation provides an alternative pipeline for zero-shot learning (ZSL), where a model for a novel class can be constructed without training data. Moreover, it leads to a new and practically relevant problem setting of zero-shot domain adaptation (ZSDA), which is the analogous to ZSL but for novel domains: A model for an unseen domain can be generated by its semantic descriptor. Experiments across this range of problems demonstrate that our framework outperforms a variety of alternatives.

3.1 Background

Multi-task and multi-domain learning are established strategies to improve learning by sharing knowledge across different but related tasks or domains. Multi-domain learning refers to sharing information about the same problem across different contextual domains, while multi-task learning addresses sharing information about different problems in the same domain. Because the domain/task distinction is sometimes subtle, and some methods proposed for MTL can also address MDL and vice-versa, the two settings are sometimes loosely used interchangeably. A detailed way to distinguish MTL and MDL clearly can be found in Section 2.3, and we recap the key idea briefly here: Domain relates to some covariate, such as the bias implicitly captured in a particular dataset [Torralba and Efros, 2011], or the specific data capture device. For example the Office Dataset [Saenko et al., 2010] contains three domains related to image source: Amazon, webcam, and DSLR. A multi-domain learning problem can then be posed by training a particular object recogniser across these three domains (same task, different domains). In contrast, a multi-task problem would be to share information across the recognisers for individual object categories (same domain, different tasks).

In this chapter, we propose a neural network framework that addresses both multi-domain and multi-task learning, and can perform simultaneous multi-domain multi-task learning. A key concept in our framework is the idea of a multivariate "*semantic descriptor*" for tasks and domains. Such a descriptor is often available as metadata, and can be exploited to improve information sharing for MTL and MDL. We show that various classic and recent MTL/MDL methods are special cases of our framework that make particular assumptions about this descriptor: Existing algorithms typically implicitly assume categorical domains/tasks, which is less effective for information sharing when more detailed task/domain metadata is available. For example, the classic "school dataset" poses a task of predicting students' grades, and is typically interpreted as containing a domain corresponding to each school. However, since each school has three year groups, representing domains by a semantic descriptor tuple (school-id, year-group) is better for information sharing. Our framework exploits such multi-variate semantic descriptors effectively, while existing MTL/MDL algorithms would struggle to do so, as they implicitly consider tasks/domains to be atomic.

Going beyond information sharing for known tasks, an exciting related paradigm for task-transfer is "zero-shot" learning (ZSL) [Larochelle et al., 2008, Lampert et al., 2009, Fu et al., 2014]. This setting addresses automatically constructing a test-time classifier for categories which are unseen at training time. Our neural-network framework provides an alternative pipeline for ZSL. More interestingly, it leads to the novel problem setting of zero-shot domain adaptation (ZSDA): Synthesising a model appropriate for a new unseen domain given only its semantic descriptor. For example, suppose we have an audio recogniser trained for a variety of acoustic playback environments, and for a variety of microphone types: Can we synthesise a recogniser for an arbitrary environment-microphone combination? To answer this question, zero-shot domain adaptation is addressed here.

3.2 Methodology

3.2.1 General Framework

Assume that we have M domains (tasks), and the ith domain has N_i instances. We denote the feature vector of the jth instance in the ith domain (task) and its associated semantic descriptor by the pair $\{\{x_j^{(i)}, z^{(i)}\}_{j=1,2,\cdots,N_i}\}_{i=1,2,\cdots,M}$ and the corresponding label as $\{\{y_j^{(i)}\}_{j=1,2,\cdots,N_i}\}_{i=1,2,\cdots,M}$. Note that, in multi-domain or multi-task learning, all the instances are effectively associated with a semantic descriptor indicating their domain (task). Without loss of generality, we propose an objective function that minimises the empirical risk for all domains (tasks),

$$\underset{P,Q}{\arg\min} \frac{1}{M}\sum_{i=1}^{M}\left(\frac{1}{N_i}\sum_{j=1}^{N_i}\mathcal{L}\Big(\hat{y}_j^{(i)}, y_j^{(i)}\Big)\right), \quad \text{where} \quad \hat{y}_j^{(i)} = f_P(x_j^{(i)}) \cdot g_Q(z^{(i)}) \tag{3.1}$$

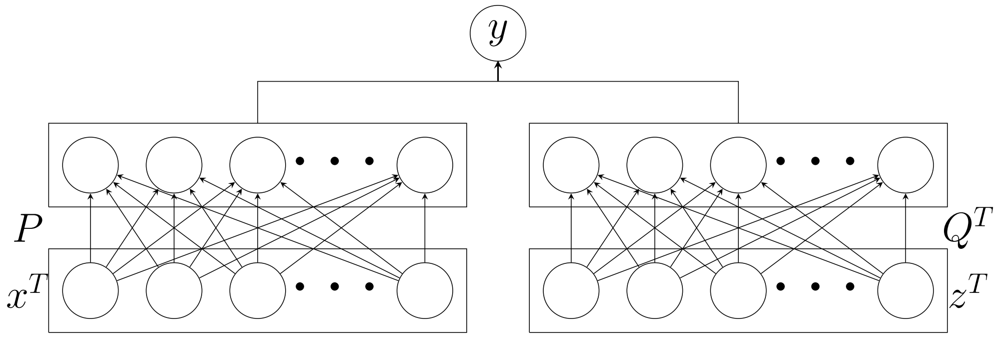


Figure 3.1: Two-sided Neural Network for Multi-Task/Multi-Domain Learning

This model can be understood as a two-sided neural network illustrated by Figure 3.1. One can see it contains two learning processes: the left-hand side is representation learning $f_P(\cdot)$, starting with the original feature vector x; and the right-hand side is model construction $g_Q(\cdot)$, starting with an associated semantic descriptor z. P and Q are the weights to train for each side. To train P and Q, standard back propagation can be performed by the loss $\mathcal{L}(\cdot)$ calculated between ground truth y and the prediction $\hat{y}$.

With this neural network interpretation, the two sides can be arbitrarily complex but we find that one inner product layer for each is sufficient to unify some existing MDL/MTL algorithms and demonstrate the efficacy of the approach. In this case, P is a D-by-K matrix and Q is a B-by-K matrix, where K is the number of units in the middle layer; D and B is the length of feature vector x and semantic descriptor z respectively. The prediction is then based on $(x_j^{(i)}P)(z^{(i)}Q)'$.

It is worth mentioning that such 'two-sided neural network' topology exists in many other works, e.g., siamese architecture [Bromley et al., 1994, Chopra et al., 2005] and conditional GANs [Mirza and Osindero, 2014], though they are not related to MTL/MDL directly.

3.2.2 Semantic Descriptor Design

One-hot encoding z

In the simplest scenario z is a one-hot encoding vector that *indexes* domains/tasks (Fig. 3.2). The model generation function $f(z^{(i)})$ then just *selects* one column from the matrix W. For example, $z^{(1)} = [1,0,0]^T$, $z^{(2)} = [0,1,0]^T$, $z^{(3)} = [0,0,1]^T$ if there are $M = 3$ domains/tasks. In this case, the length of the descriptor and the number of unique domains (tasks) are equal $B = M$, and the stack of all $z^{(i)}$ vectors (denoted $Z = [z^{(1)}, z^{(2)}, \dots])$ is an $M \times M$ identity matrix.

$$Z = \left[\begin{array}{cccc} & \text{Domain-1} & \text{Domain-2} & \text{Domain-3} \\ \text{Index-1} & 1 & 0 & 0 \\ \text{Index-2} & 0 & 1 & 0 \\ \text{Index-3} & 0 & 0 & 1 \end{array}\right] \qquad Z = \left[\begin{array}{cccc} & \text{Domain-1} & \text{Domain-2} & \text{Domain-3} \\ \text{Index-1} & 1 & 0 & 0 \\ \text{Index-2} & 0 & 1 & 0 \\ \text{Index-3} & 0 & 0 & 1 \\ \text{Shared} & 1 & 1 & 1 \end{array}\right]$$

Figure 3.2: Domain descriptor for categorical/atomic domains. One-hot encoding (left), and one-hot with constant encoding (right).

One-hot encoding z with a constant

A drawback of one-hot encoding is that the $z^{(i)}$'s are orthogonal to each other, which suggests that all domains/tasks are independent – there is no cross domain/task information sharing. To encode an expected sharing structure of an underlying commonality across all domains/tasks, an alternative approach to constructing z is to append a constant term after the one-hot encoding. For the case of $M = 3$, we might have $z^{(1)} = [1, 0, 0, 1]^T$, $z^{(2)} = [0, 1, 0, 1]^T$, $z^{(3)} = [0, 0, 1, 1]^T$. Fig. 3.2 shows the resulting $B \times M$ matrix Z (in this case, $B = M + 1$). The prediction of task (domain) i is given as $\hat{y}^{(i)} = {x^{(i)}}^T W z^{(i)} = {x^{(i)}}^T (w^{(i)} + w^{(4)})$, i.e., the sum of a task/domain specific and a globally shared prediction. Learning the weight generator corresponds to training both the local and shared predictors. This approach is implicitly used by some classic MDL/MTL algorithms [Evgeniou and Pontil, 2004, Daumé III, 2007].

Distributed encoding z

In most studies of MDL/MTL, domain or task is assumed to be an atomic category which can be effectively encoded by the above indexing approaches. However more structured domain/task-metadata is often available, such that a domain (task) is indexed by multiple factors (e.g., time: day/night, and date: weekday/weekend, for batches of video surveillance data). Suppose two categorical variables (A,B) describe a domain, and each of them has two states (1,2), then at most four distinct domains exist. Fig. 3.3(left) illustrates the semantic descriptors for one-hot encoding. However this encoding does not exploit the sharing cues encoded in the metadata (e.g., in the surveillance example that day-weekday should be more similar to day-weekend and night-weekday than to night-weekend). Thus we propose to use a distributed encoding of the task/domain descriptor (Fig. 3.3(right)). Now prediction weights are given by a linear combination of W's columns given by the descriptor, and learning the weight generating function means learning weights for each factor (e.g., day, night, weekday, weekend). We will demonstrate that the ability to exploit structured domain/task descriptors where available, improves information sharing compared to existing MTL/MDL methods in later experiments.

$$Z = \begin{bmatrix} & \text{Domain-1} & \text{Domain-2} & \text{Domain-3} & \text{Domain-4} \\ \text{A-1-B-1} & 1 & 0 & 0 & 0 \\ \text{A-1-B-2} & 0 & 1 & 0 & 0 \\ \text{A-2-B-1} & 0 & 0 & 1 & 0 \\ \text{A-2-B-2} & 0 & 0 & 0 & 1 \end{bmatrix} \qquad Z = \begin{bmatrix} & \text{Domain-1} & \text{Domain-2} & \text{Domain-3} & \text{Domain-4} \\ \text{A-1} & 1 & 1 & 0 & 0 \\ \text{A-2} & 0 & 0 & 1 & 1 \\ \text{B-1} & 1 & 0 & 1 & 0 \\ \text{B-2} & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 3.3: Example domain descriptor for domains with multiple factors. One-hot encoding (left). Distributed encoding (right).

3.2.3 Unification of Existing Algorithms

We next demonstrate how a variety of existing algorithms[1] are special cases of our general framework. For clarity we show this in an MDL/MTL setting with $M = 3$ domains/tasks. Observe that RMTL [Evgeniou and Pontil, 2004], FEDA [Daumé III, 2007], MTFL [Argyriou et al., 2008], TNMTL [Ji and Ye, 2009], and GO-MTL [Kumar and Daumé III, 2012] each assumes specific settings of Z, P and Q' as in Table 3.1.

Table 3.1: A Unifying Review of Some Existing MTL/MDL Algorithms

	Z	P	Norm on P	Q'	Norm on Q'
RMTL	$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	Identity	None	$\begin{bmatrix} \vert & \vert & \vert & \vert \\ v_1 & v_2 & v_3 & w_0 \\ \vert & \vert & \vert & \vert \end{bmatrix}$	Frobenius
FEDA	$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	Identity	None	$\begin{bmatrix} \vert & \vert & \vert & \vert \\ v_1 & v_2 & v_3 & w_0 \\ \vert & \vert & \vert & \vert \end{bmatrix}$	None
MTFL	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Identity	None	W	$\ell_{2,1}$-Norm
TNMTL	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Identity	None	W	Trace Norm
GO-MTL	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	L	Frobenius	S	Entry-wise ℓ_1

The notion used is kept same with the original paper, e.g., P here is analogous to L in Kumar and Daumé III [2012]. Each row of the matrices in the second (Z) column is the corresponding domain's semantic descriptor in different methods. These methods are implicitly assuming a single categorical domain/task index: with 1-of-N encoding as semantic descriptor (sometimes with a constant term).

We argue that more structured domain/task-metadata is often available, and with our framework it can be directly exploited to improve information sharing (with the distributed encoding in Section 3.2.2) compared to simple categorical indices (corresponding to the one-hot encoding in Section 3.2.2). The ability to exploit more structured domain/task descriptors Z where available, improves information sharing compared to

[1]RMTL: **R**egularized **M**ulti–**T**ask **L**earning, FEDA: **F**rustratingly **E**asy **D**omain **A**daptation, MTFL: **M**ulti–**T**ask **F**eature **L**earning, TNMTL: **T**race-**N**orm **M**ulti–**T**ask **L**earning, and GO-MTL: **G**rouping and **O**verlap for **M**ulti–**T**ask **L**earning

existing MTL/MDL methods, and more importantly, it enables zero-shot domain adaptation. In our experiments, we will demonstrate examples of problems with multivariate domain/task metadata, and its efficacy to improve learning.

3.2.4 Learning Settings

Multi-Domain Multi-Task (MDMT)

Existing frameworks have focused on either MDL or MTL settings but not considered both together. Our interpretation provides a simple means to exploit them both simultaneously for better information sharing when multiple tasks in multiple domains are available. If $z^{(d)}$ and $z^{(t)}$ are the domain and task descriptors respectively, then MDMT learning can be performed by simply concatenating the descriptors $[z^{(d)}, z^{(t)}]$ corresponding to the domain and task of each individual instance during learning.

Zero-Shot Learning (ZSL)

As mentioned, the dominant zero-shot (task) learning pipeline is $X \to Z \to Y$. At training time, the $X \to Z$ mapping is learned by classifier/regressor, where Z is a task descriptor, such as a binary attribute vector [Lampert et al., 2009, Fu et al., 2014], or a continuous word-vector describing the task name [Socher et al., 2013, Fu et al., 2014]. At testing time, the "prototype" semantic vector for a novel class z is presented, and zero-shot recognition is performed by matching the $X \to Z$ estimate and prototype z, e.g., by nearest neighbour [Fu et al., 2014].

In our framework, ZSL is achieved by presenting each novel semantic vector z_j^* (each testing category is indexed by j) in turn along with novel category instances x^*. Zero-shot recognition then is given by: $j^* = \underset{j}{\operatorname{argmax}} f_P(x^*) \cdot f_Q(z_j^*)$. Note that this pipeline is more similar to [Larochelle et al., 2008, Frome et al., 2013] in the light of the following illustration: $\begin{tikzcd} Z \arrow[rr, bend left] & & X \arrow[r] & Y \end{tikzcd}$.

If z is the binary vector that contains the one-hot encoding attributes, Q can be understood as the word embedding matrix, and it can be initialised by pre-trained word2vec models [Mikolov et al., 2013] optionally, where each row of Q is a dense word vector.

Zero-Shot Domain Adaptation (ZSDA)

ZSDA becomes possible with a distributed rather than one-hot encoded domain descriptor, as in practice only a subset of domains is necessary to effectively learn Q. Thus a model $w^{(*)}$ suitable for an *unseen* domain can be constructed without any training data – by applying its semantic descriptor $z^{(*)}$ to the model generation matrix Q: $w^{(*)} = Qz^{(*)}$. The generated domain-specific model – $w^{(*)}$ – is then used to make predictions for the re-represented input: $x_*^T P w^{(*)}$.

3.2.5 Connection to Multilinear MTL

In this section, we discuss the connection of the proposed method and a variant of these tensor-based MTL methods, e.g., [Wimalawarne et al., 2014, Romera-Paredes et al., 2013].

Our method and tensor-based MTL can deal with the case when a task is indexed by more than one factors. For example, in the School Dataset, we have $T_1 = 139$ schools and each school has $T_2 = 3$ grades, thus we can (at most) have $T_1 \times T_2 = 139 \times 3 = 417$ tasks. Assume we use two one-hot encoding vectors to represent the task identity: z_1 is for school factor and of length $T_1 = 139$ and z_2 is for grade factor and of length $T_2 = 3$.

The model generating function in our approach can be written as $w = W[z_1; z_2]$, where $W \in \mathbb{R}^{D\times(T_1+T_2)}$, in contrast, the model generating function in tensor-based MTL can be written as $w = \mathcal{W} \bullet z_2 \bullet z_1$. Here $\bullet$ operator indicates tensor dot product (see Appx. A.4.4 for the detailed definition).

Note that we can (optionally) factorise W in our approach, then the model is produced by $w = UV[z_1; z_2]$ where $U \in \mathbb{R}^{D\times K}$, $V \in \mathbb{R}^{K\times(T_1+T_2)}$, and K is a hyper-parameter (matrix rank). Further we expand V as $V = [V_1, V_2]$ where $V_1 \in \mathbb{R}^{K\times T_1}$ and $V_2 \in \mathbb{R}^{K\times T_2}$, then we have $w = UV[z_1; z_2] = U[V_1, V_2][z_1; z_2] = U((V_1 z_1) + (V_2 z_2))$.

MLMTL-NC [Romera-Paredes et al., 2013] factorises the tensor $\mathcal{W}$ according to the formulation of Tucker decomposition. Here we choose to use CP decomposition instead, i.e., $\mathcal{W} = \mathcal{I} \bullet_{(1,2)} U \bullet_{(1,1)} V_1 \bullet_{(1,1)} V_2$, where, $\mathcal{I}$ is a $K \times K \times K$ identity tensor (constant), $U \in \mathbb{R}^{D\times K}$, $V_1 \in \mathbb{R}^{K\times T_1}$, $V_2 \in \mathbb{R}^{K\times T_2}$, and K is a hyper-parameter (tensor CP-rank). Then we have $w = \mathcal{W} \bullet z_2 \bullet z_1 = U((V_1 z_1) \circ (V_2 z_2))$.

Now we have two observations: (i) For the number of model parameters, our approach and this variant of MLMTL-NC has the same: $K \times (D + T_1 + T_2)$. (ii) If we see U as the basis of latent tasks, then the key difference of these two approaches is about how to *aggregate* the task coding vectors produced by different task factors, i.e., $V_1 z_1$ and $V_2 z_2$. Our method chooses to sum together, i.e., $(V_1 z_1) + (V_2 z_2)$, but the variant of MLMTL-NC chooses to do element-wise product, i.e., $(V^{(1)} z_1) \circ (V^{(2)} z_2)$.

We can not make a comment about which one is better, as this kind of essential design choice is very likely to be problem dependent. One minor advantage of our approach over the original MLMTL-NC is that our method has fewer hyper-parameters.

Discussion on Multi-Class v.s. One-vs-All

It is common that in MTL studies, a multi-class classification problem with C unique classes is usually built up as C binary one-vs-all binary classification problems, and each class will be treated as a task. If we have more than one factors distinguishing tasks, e.g., apart from class, we have another factor indicating data source, our method can still be applied, as detailed in multi-domain-multi-task in Section 3.2.4.

However, we have to cast the multi-class problem into multiple binary one-vs-all problems for applying our method. This is unnecessary for tensor-based methods such as [Wimalawarne et al., 2014, Romera-Paredes et al., 2013] because they are built on modelling predictor tensors, so the multi-class problem can be set as-it-is, e.g., one can use cross-entropy loss for multi-class.

Theoretically our approach can also work in this fashion as it essentially models the predictors like tensor-based MTL methods, but the bottleneck happens in the architecture of neural network in Fig. 3.1: there is only one output unit. This this motivates us to generalise our method in Chapter 4.

3.3 Experiments

For the method in Chapter 3, named as GNNMTL (Gated Neural Network Multi-Task Learning), we demonstrate on five experimental settings: MDL, ZSDA, MTL, ZSL and MDMT. These experiments are for two main purposes: (i) to compare our general unified method to prior special cases (ii) to validate that semantic descriptors can be used to improve performance when available.

Implementation We implement the model with the help of Caffe framework [Jia et al., 2014]. Though we don't place regularisation terms on P or Q, a non-linear function $\sigma(x) = max(0, x)$ (i.e., ReLU activation function) is placed to encourage sparse models $g_Q(z^{(i)}) = \sigma(z^{(i)}Q)$. The choice of loss function for regression and classification is Euclidean loss and Hinge loss respectively. For the hyper-parameter K, i.e., the rank of W (or the number of hidden neurons), the preliminary experiments show $K = \frac{D}{\log(D)}$ leads to satisfactory solutions for all datasets in this section.

MTL/MDL Baselines: We compare the proposed method with a single task learning baseline – linear or logistic regression with ℓ_2 regularisation (LR), and four multi-task learning methods: (i) RMTL [Evgeniou and Pontil, 2004], (ii) FEDA [Daumé III, 2007], (iii) MTFL [Argyriou et al., 2008] and (iv) GO-MTL [Kumar and Daumé III, 2012]. Note that these methods are re-implemented within the proposed framework. We have verified our implementations with the original ones and found that the performance difference is not significant. Baseline methods use traditional 1-of-N encoding, while we use a distributed descriptor encoding based on metadata for each problem.

Zero-Shot Domain Adaptation: We follow the MDL setting to learn P and Q except that one domain is held out each time. We construct test-time models for held out domains using their semantic descriptor. We evaluate against two baselines: (i) Blind-transfer (LR): learning a single linear/logistic regression model on aggregated data from all seen domains. To ensure fair comparison, distributed semantic descriptors are concatenated with the feature vectors for baselines, i.e., they are included as a plain feature. (ii) Tensor-completion (TC): we use a tensor $W \in \mathcal{R}^{D,p_1,p_2,\cdots,p_N}$ to store all the linear models trained by SVM where N is the number of categorical variables and p_i is the number of states in the ith categorical variable ($p_1 + p_2 + \cdots + p_N = B$ in our context and $p_1 * p_2 * \cdots * p_N = M$ if there is always a domain for each of possible combinations). ZSDA can be formalised by setting the model parameters for the held-out domain to missing values, and recovering them by a low-rank tensor completion algorithm [Kressner et al., 2014]. This low-rank strategy corresponds to our implementation of [Romera-Paredes et al., 2013].

3.3.1 School Dataset - MDL and ZSDA

Data This classic dataset[2] collects exam grades of 15,362 students from 139 schools. Given the 23 features[3], a regression problem is to predict each student's exam grade. There are 139 schools and three year groups. School IDs and year groups naturally form multivariate domains. Note that 64 of 139 schools have the data of students for all three year groups, and we also choose the school of which each year group has more than 50 students so that each domain has sufficient training data. Finally there are $23 \times 3 = 69$ distinct domains given these two categorical variables.

Settings and Results For MDL we learn all domains together, and for ZSDA we use a leave-one-domain-out strategy, constructing the test-time model based on the held-out domain's descriptor with P and Q learned from the training domains. In each case the training/test split is 50%/50%. Note that the test sets for MDL and ZSDA are the same. The results in Table 3.2 are averages over the test set for all domains (MDL), and averages over the held-out domain performance when holding out each of the 69 domains in turn (ZSDA). Our method outperforms the alternatives in each case.

Table 3.2: School Dataset (RMSE)

	LR	RMTL	FEDA	MTFL	GO-MTL	TC	GNNMTL
MDL	9.51	9.46	10.75	10.22	10.00	-	**9.37**
ZSDA	10.35	-	-	-	-	12.41	**10.19**

3.3.2 Audio Recognition - MDL and ZSDA

Audio analysis tasks are affected by a variety of covariates, notably the playback device / environment (e.g., studio recording versus live concert hall), and the listening device (e.g., smartphone versus professional microphone). Directly applying a model trained in one condition/domain to another will result in poor performance. Moreover, as the covariates/domains are combinatorial: (i) models cannot be trained for all situations, and (ii) even applying conventional domain adaptation is not scalable. Zero-shot domain adaptation has potential to address this, because a model could be calibrated on the fly for a given environment.

Data We investigate recognition in a complex set of noise domains: covering both acoustic environment and microphone type. We consider a music-speech discrimination task introduced in [Tzanetakis and Cook, 2002], which includes 64 music and speech

[2]Available at http://multilevel.ioe.ac.uk/intro/datasets.html

[3]The original dataset has 26 features, but 3 that indicate student year group are used in semantic descriptors.

tracks. Two categorical variables are *smartphone microphone* and *live concert hall* environment, and each of them has two states: on or off. Then the four domains are generated as: (i) Original (ii) Live Recording (LRc) (iii) Smartphone Recording (SRc) and (iv) smartphone in a live hall (LRSR). The noises are synthesised by Audio Degradation Toolbox [Mauch and Ewert, 2013].

Settings and Results We use MFCC [Ellis, 2005] to extract audio features and K-means to build a K = 64 bag-of-words representation. We split the data 50%/50% for training and test and keep test sets same for MDL and ZSDA. The results in Table 3.3 break down the results by each domain and overall (MDL), and each domain when held-out (ZSDA). In each case our method is best or joint-best due to better exploiting the semantic descriptor (recall that it does not have any additional information; for fairness the descriptor is also given to the other methods as a regular feature). The only exception is the least practical case of ZSDA recognition in a noise free environment given prior training only in noisy environments. The ZSDA result here generally demonstrates that models can be synthesised to deal effectively with new multivariate domains / covariate combinations without needing to exhaustively see data and explicitly train models for all, as would be conventionally required.

Table 3.3: Audio Recognition: Music versus Speech (Error Rate)

		Origin	LRc	SRc	LRSR	Avg
MDL	LR	**3.13**	18.75	6.25	17.19	11.33
	RMTL	6.25	18.75	6.25	17.19	12.11
	FEDA	7.81	18.75	9.38	18.75	13.67
	MTFL	6.25	21.88	9.38	**14.06**	12.89
	GO-MTL	**3.13**	**17.19**	6.25	18.75	11.33
	GNNMTL	**3.13**	**17.19**	**4.69**	**14.06**	**9.77**
ZSDA	LR	**32.81**	28.13	14.06	23.44	24.61
	TC	46.88	21.88	26.56	59.38	38.67
	GNNMTL	35.94	**9.38**	**12.50**	**18.75**	**19.14**

3.3.3 Animal with Attributes - MTL and ZSL

Animal with Attributes [Lampert et al., 2009] includes images from 50 animal categories, each with an 85-dimensional binary attribute vector. The attributes, such as "black", "furry", "stripes", describe an animal semantically, and provide a unique mapping from a combination of attributes to an animal. The original setting of ZSL with AwA is to split the 50 animals into 40 for training and hold out 10 for testing.

We evaluate this condition to investigate: (i) if multi-task learning of attributes and classes improves over the STL approaches typically taken when analysing AwA, (ii) if it helps to use the attributes as an MTL semantic task descriptor against the traditional setting of MTL where semantic descriptor is a 1-of-N unit vector indexing tasks. For MTL training on AwA, we decompose the multi-class problem with C categories to

C one-vs-rest binary classification tasks. For testing time, we try each testing class' descriptor z_* in turn and pick the best one (with the highest $f_P(x) \cdot f_Q(z_*)$ value).

Multi-Task Learning We use the recently released DeCAF feature [Donahue et al., 2015] for AwA. For MTL, we pick five animals from the training set with moderately overlapped attributes, and use the first half of the images for training then test on the rest. The results in Table 3.4 show limited improvement by existing MTL approaches over the standard STL. However, our attribute-descriptor approach to encoding tasks for MTL improves the accuracy by about 2% over STL.

Table 3.4: AwA: MTL Multi-Class Accuracy

	antelope	killer whale	otter	walrus	blue whale	Avg
LR	92.31	87.08	89.26	75.60	82.44	85.34
RMTL	86.08	71.22	80.99	61.90	**96.18**	79.28
FEDA	92.31	83.39	88.15	79.17	89.31	86.47
MTFL	92.67	85.61	90.36	79.76	87.02	87.09
GO-MTL	91.21	84.87	89.81	**80.36**	84.73	86.20
GNNMTL	**93.41**	**91.51**	**94.21**	79.76	79.39	**87.66**

Zero-Shot Learning For ZSL, we adopt the training/testing split in [Lampert et al., 2009]. The blind-transfer baseline is not meaningful because there are different binary classification problems, and aggregating does not lead to anything. Also, tensor-completion is not practical because of its exponential space ($D * 2^{85}$) against $D * 40$ observations. Our method achieves 43.79% multi-class accuracy, compared to 41.03% from direct-attribute prediction (DAP) approach [Lampert et al., 2009] using DeCAF features. A recent result using DeCAF feature is 44.20% in [Deng et al., 2014], but this uses additional higher order attribute correlation information. Given that we did not design a solution for AwA specifically, or exploit this higher order correlation cue, the result is encouraging.

3.3.4 Restaurant & Consumer Dataset - MDMT

The restaurant & Consumer Dataset, introduced in [Vargas-Govea et al., 2011] contains 1161 customer-to-restaurant scoring records, where each record has 43 features and three scores: food, service and overall. We build a multi-domain multi-task problem as follows: (i) a domain refers to a restaurant, (ii) a task is a regression problem to predict one of the three scores given an instance and (iii) an instance is a 43-dimensional feature vector based on customer's and restaurant's profile. The 1161 records cover 130 restaurants but most of them just have few scores, so we just pick 8 most frequently scored ones, and we split training and test sets equally. The semantic descriptor is constructed by concatenating 8-bit domain and 3-bit task indicator. Conventional MTL interpretations of this dataset consider $8 \times 3 = 24$ atomic tasks. Thus the task overlap across domain or domain overlap across task is ignored. Results in Table 3.5 shows that our approach outperforms this traditional MTL setting by better representing it as a distributed MDMT problem.

Table 3.5: Restaurant & Consumer Dataset (RMSE)

LR	RMTL	FEDA	MTFL	GO-MTL	GNNMTL
2.32	1.23	1.17	1.13	1.06	**0.78**

3.4 Summary

In this chapter we proposed a unified framework for multi-domain and multi-task learning. The core concept is a semantic descriptor for tasks or domains. This can be used to unify and improve on a variety of existing multi-task learning algorithms. Moreover it naturally extends the use of a single categorical variable to index domains/tasks to the multivariate case, which enables better information sharing where additional metadata is available. Beyond multi task/domain learning, it enables the novel task of zero-shot domain adaptation and provides an alternative pipeline for zero-shot learning.

Neural networks have also been used to address MTL/MDL by learning shared invariant features [Donahue et al., 2015]. Our contribution is complementary to this (as demonstrated e.g., with AwA) and the approaches are straightforward to combine by placing more complex structure on left-hand side $f_P(\cdot)$. Our future directions are: (i) The current semantic descriptor is formed by discrete variables. We want to extend this to continuous and periodic variable like the pose, brightness and time. (ii) We assume the semantic descriptor (task/domain) is always observed, an improvement for dealing with a missing descriptor is also of interest.

Chapter 4

Multi Output

In Chapter 3, we proposed a general framework for multi-domain and multi-task learning, but one key assumption that we made is that the each domain (or task) is associated with a single-output prediction problem, e.g., binary classification. In this chapter, we present a higher order generalisation of the framework in Chapter 3, which makes it capable of dealing with the case that each domain is associated with a multiple-output problem (simultaneous multi-domain-multi-task setting). This generalisation has two mathematically equivalent views in multi-linear algebra and gated neural networks respectively. In practice, this framework provides a powerful yet easy to implement method that can be flexibly applied to multi-domain learning, multi-task learning, and a mixture of these two.

4.1 Background

The multi-domain setting arises when there is data about a task in several different but related domains. For example in visual recognition of an object when viewed with different camera types. Multi-domain learning (MDL) models [Dredze et al., 2010, Daumé III, 2007, Yang and Hospedales, 2015] aim to learn a cross-domain parameter sharing strategy that reflects the domains' similarities and differences. Such selective parameter sharing aims to be robust to the differences in statistics across domains, while exploiting data from multiple domains to improve performance compared to learning each domain separately.

In this chapter we derive a general framework that encompasses MDL and MTL from both neural network and tensor-factorisation perspectives. Many classic and recent MDL/MTL algorithms can be understood by the assumptions about the cross domain/task sharing structure encoded in their designs. E.g., the assumption that each task/domain's model is a linear combination of a global and a task-specific parameter vector [Evgeniou and Pontil, 2004, Daumé III, 2007]. Our framework includes these as special cases corresponding to specific settings of a *semantic descriptor vector* parametrising tasks/domains [Yang and Hospedales, 2015]. This vector can be used to recover existing models from our framework, but more generally it allows one to relax the often implicit assumption that domains are atomic/categorical entities, and exploit available *metadata* about tasks/domains to guide sharing structure for better MDL/MTL [Yang and Hospedales, 2015, 2016]. For example, in surveillance video analysis, exploiting the knowledge of the time of day and day of week corresponding to each domain for better MDL. Finally, the idea of a semantic task/domain descriptor, allows our framework to go beyond the conventional MDL/MTL setting, and address both zero-shot learning [Yang and Hospedales, 2015] and zero-shot domain adaptation [Yang and Hospedales, 2015, 2016] – where a model can be deployed for a new task/domain without any training data, solely by specifying the task/domain's semantic descriptor metadata.

Relation to Domain Adaptation and Domain Generalisation

Dataset bias/domain-shift means that models trained on one domain often have weak performance when deployed in another domain. The community has proposed two different approaches to alleviate this: (i) Domain adaptation (DA): calibrating a pre-trained model to a target domain using a limited amount of labelled data – supervised DA [Saenko et al., 2010], or unlabelled data only – unsupervised DA [Gong et al., 2012], or both – semi-supervised DA [Li et al., 2014]. (ii) Domain generalisation (DG): to train a model that is insensitive to domain bias, e.g., learning domain invariant features [Muandet et al., 2013].

The objective of multi-domain learning is different from the mentioned domain adaptation and domain generalisation. MDL can be seen as a bi-directional generalisation of DA with each domain benefiting the others, so that all domains have maximal performance; rather than solely transferring knowledge from source $\rightarrow$ target domain as in DA. In its conventional form MDL does not overlap with DG, because it aims to improve the performance on the set of given domains, rather than address a held out domain. However, our zero-shot domain adaptation extension of MDL, relates to DG insofar as aiming to address a held-out domain. The difference is that we require a semantic descriptor for the held out domain, while DG does not. However where such a descriptor exists, ZSDA is expected to outperform DG.

4.2 Methodology

For the methodology developed in Section 3.2, the final output of each model is a scalar (single output). However for some practical applications, multiple outputs are desirable or required. For example, assume that we have $M = 2$ handwriting digit datasets (domains): MNIST and USPS. For any MNIST or USPS image, a D-dimensional feature vector is extracted. The task is to classify the image from 0 to 9 and thus we have $D \times C$ ($C = 10$) model parameters for each dataset. Therefore, the full model for all digits and datasets should contain $D \times C \times M$ parameters. We denote this setting of multiple domains, each of which has multiple tasks, as multi-domain-multi-task learning. In some recent literature [Romera-Paredes et al., 2013] a similar setting is named multi-linear multi-task learning.

4.2.1 Formulation

The core formulation in Chapter 3 can be understood as follows: the predictor for the ith task $w^{(i)}$ is generated from a function parametrised by the model parameters W in the way that $w^{(i)} = f_W(z^{(i)}) = Wz^{(i)}$. The key observation is this formulation generates a certain task's model parameter via its associated descriptor vector ($z^{(i)}$) hitting a matrix (W).

Now we have the situation that the desired model parameter itself is in a form of matrix, e.g, a $D \times C$ matrix in the above example. To adapt this setting, we propose to use the following formulation,

$$W^{(i)} = f_{\mathcal{W}}(z^{(i)}) = \mathcal{W} \bullet_{(3,1)} z^{(i)} \tag{4.1}$$

Here $\bullet$ operator indicates tensor dot product (see Appx. A.4.4 for the detailed definition). The generated model is now a weight *matrix* $W^{(i)}$ rather than a *vector* $w^{(i)}$. The weight generating function $f(\cdot)$ is now parametrised by a third-order tensor $\mathcal{W}$ of size $D \times C \times B$, and it synthesises the model matrix for the ith domain by hitting the tensor with its B-dimensional semantic descriptor $z^{(i)}$. This is a natural extension: if the required model is a vector (single output), the weight generating function is a matrix (second-order tensor) hits the semantic descriptor z; when the required model is a matrix (multiple outputs), the weight generating function is then z hits a third-order tensor.

Given one-hot encoding descriptors $z^{(1)} = [1, 0]^T$ and $z^{(2)} = [0, 1]^T$ indicating MNIST and USPS respectively. Eq. 4.1 would just *slice* an appropriate matrix from the tensor $\mathcal{W}$. However alternative and more powerful distributed encodings of $z^{(i)}$ are also applicable. The model prediction can be written as,

$$\hat{y}_j^{(i)} = \mathcal{W} \bullet_{(3,1)} z_j^{(i)} \bullet_{(1,1)} x_j^{(i)} \tag{4.2}$$

where $\hat{y}_j^{(i)}$ is now a C-dimensional vector instead of a scalar. Nevertheless, this method does not provide information sharing in the case of conventional categorical (one-hot encoded) domains. For this we turn to tensor factorisation next.

4.2.2 Tensor Decomposition

Recall that the key technique that underlies many classic (matrix-based) multi-task learning methods is to exploit the information sharing induced by the row-rank factorisation, i.e., $W := PQ$ thus the model generating function becomes $f_{P,Q}(z) = PQz$. Instead of learning the predictor W directly, it learns two factor matrices P and Q. For MDL with multiple outputs, we aim to extend this idea to the factorisation of the weight tensor $\mathcal{W}$. In contrast to the case with matrices, there are multiple approaches to factorising tensors, including CP [Hitchcock, 1927], Tucker [Tucker, 1966], and Tensor-Train [Oseledets, 2011] Decompositions.

CP decomposition

For a third-order tensor $\mathcal{W}$ of size $D \times C \times B$, the rank-K CP decomposition is:

$$\mathcal{W}_{d,c,b} = \sum_{k=1}^{K} U^{(D)}_{k,d} U^{(C)}_{k,c} U^{(B)}_{k,b} \tag{4.3}$$

$$\mathcal{W} = \mathcal{I} \bullet_{(1,1)} U \bullet_{(1,1)} U^{(D)} \bullet_{(1,1)} U^{(C)} \bullet_{(1,1)} U^{(B)} \tag{4.4}$$

The factor matrices $U^{(D)}$, $U^{(C)}$, and $U^{(B)}$ are of respective size $K \times D$, $K \times C$, and $K \times B$, and they are the parameters to learn as a replacement of original $\mathcal{W}$. $\mathcal{I}$ is a $K \times K \times K$ identity tensor (it is a constant rather than a parameter to learn).

Given a data point x and its corresponding descriptor z (we omit the upper- and lower- scripts for clarity.), Eq. 4.2 will produce a C-dimensional vector y (e.g., $C = 10$ the scores/logits of 10 digits for the MNIST/USPS example). By substituting Eq. 4.4 into Eq. 4.2 and some reorganising, we obtain

$$y = {U^{(C)}}^T((U^{(D)}x) \circ (U^{(B)}z)) \tag{4.5}$$

where $\circ$ is the element-wise product. It also can be written as,

$$y = {U^{(C)}}^T \operatorname{diag}(U^{(B)}z)U^{(D)}x \tag{4.6}$$

from which we obtain a specific form of the weight generating function in Eq. 4.1, which is motived by CP decomposition:

$$W^{(i)} = f_{U^{(D)},U^{(B)},U^{(C)}}(z^{(i)}) = {U^{(D)}}^T \operatorname{diag}(U^{(B)}z)U^{(C)} \tag{4.7}$$

$\operatorname{diag}(\cdot)$ is a function that converts a vector into a matrix by placing its elements along the diagonal of matrix. It is worth mentioning that this formulation has been used in the context of gated neural networks [Sigaud et al., 2015], such as Gated Autoencoders [Droniou and Sigaud, 2013]. However, [Droniou and Sigaud, 2013] uses the technique to model the relationship between two inputs (images), while we exploit it for knowledge sharing in multi-task/multi-domain learning.

Tucker decomposition

Given the same sized tensor $\mathcal{W}$, Tucker decomposition outputs a core tensor $\mathcal{S}$ of size $K_D \times K_C \times K_B$, and 3 matrices $U^{(D)}$ of size $K_D \times D$, $U^{(C)}$ of size $K_C \times C$, and $U^{(B)}$

of size $K_B \times B$, such that,

$$\begin{aligned}
\mathcal{W}_{d,c,b} &= \sum_{k_D=1}^{K_D} \sum_{k_C=1}^{K_C} \sum_{k_B=1}^{K_B} \mathcal{S}_{k_D,k_C,k_B} U^{(D)}_{k_D,d} U^{(C)}_{k_C,c} U^{(B)}_{k_B,b} && (4.8) \\
\mathcal{W} &= \mathcal{S} \bullet_{(1,1)} U^{(D)} \bullet_{(1,1)} U^{(C)} \bullet_{(1,1)} U^{(B)} && (4.9)
\end{aligned}$$

Substituting Eq. 4.9 into Eq. 4.2, we get the prediction for instance x in domain/task z

$$y = ((U^{(D)}x) \otimes (U^{(B)}z))\mathcal{S}_{(2)}^T U^{(C)} \tag{4.10}$$

where $\otimes$ is Kronecker product. $\mathcal{S}_{(2)}$ is the mode-2 unfolding of $\mathcal{S}$ which is a $K_C \times K_D K_B$ matrix, and its transpose $\mathcal{S}_{(2)}^T$ is a matrix of size $K_D K_B \times K_C$.

This formulation was used by studies of Gated Restricted Boltzmann Machines (GRBM) [Memisevic and Hinton, 2007] for similar image-transformation purposes as [Droniou and Sigaud, 2013]. The weight generating function (Eq. 4.1) for Tucker decomposition is

$$W^{(i)} = f_{\mathcal{S},U^{(D)},U^{(B)},U^{(C)}}(z^{(i)}) = \mathcal{S} \bullet_{(1,1)} U^{(D)} \bullet_{(1,1)} U^{(C)} \bullet_{(1,1)} (U^{(B)}z^{(i)}). \tag{4.11}$$

TT decomposition

Given the same sized tensor $\mathcal{W}$, Tensor-Train (TT) decomposition produces two matrices $U^{(D)}$ of size $D \times K_D$ and $U^{(B)}$ of size $K_B \times B$ and a third-order tensor $\mathcal{S}$ of size $K_D \times C \times K_B$, so that

$$\begin{aligned}
\mathcal{W}_{d,c,b} &= \sum_{k_D=1}^{K_D} \sum_{k_B=1}^{K_B} U^{(D)}_{d,k_D} \mathcal{S}_{k_D,c,k_B} U^{(B)}_{k_B,b}, && (4.12) \\
\mathcal{W} &= U^{(D)} \bullet \mathcal{S} \bullet U^{(B)} && (4.13)
\end{aligned}$$

Substituting Eq. 4.13 into Eq.4.2, we obtain the MDL/MTL prediction

$$y = ({U^{(D)}}^T x) \otimes (U^{(B)}z))\mathcal{S}_{(2)}^T \tag{4.14}$$

where $\mathcal{S}_{(2)}$ is the mode-2 unfolding of $\mathcal{S}$ which is a $C \times K_D K_B$ matrix, and its transpose $\mathcal{S}_{(2)}^T$ is a matrix of size $K_D K_B \times C$. The weight generating function (Eq. 4.1) for Tensor Train decomposition is

$$W^{(i)} = f_{\mathcal{S},U^{(D)},U^{(B)}}(z^{(i)}) = U^{(D)} \bullet \mathcal{S} \bullet (U^{(B)}z^{(i)}). \tag{4.15}$$

Method	Factors (Shape)			
CP	$U^{(D)}$ $(K \times D)$	$U^{(C)}$ $(K \times C)$	$U^{(B)}$ $(K \times B)$	
Tucker	$U^{(D)}$ $(K_D \times D)$	$U^{(C)}$ $(K_C \times C)$	$U^{(B)}$ $(K_B \times B)$	$\mathcal{S}$ $(K_D \times K_C \times K_B)$
TT	$U^{(D)}$ $(D \times K_D)$		$U^{(B)}$ $(K_B \times B)$	$\mathcal{S}$ $(K_D \times C \times K_B)$

Table 4.1: Summary of factors used by different tensor (de)composition methods.

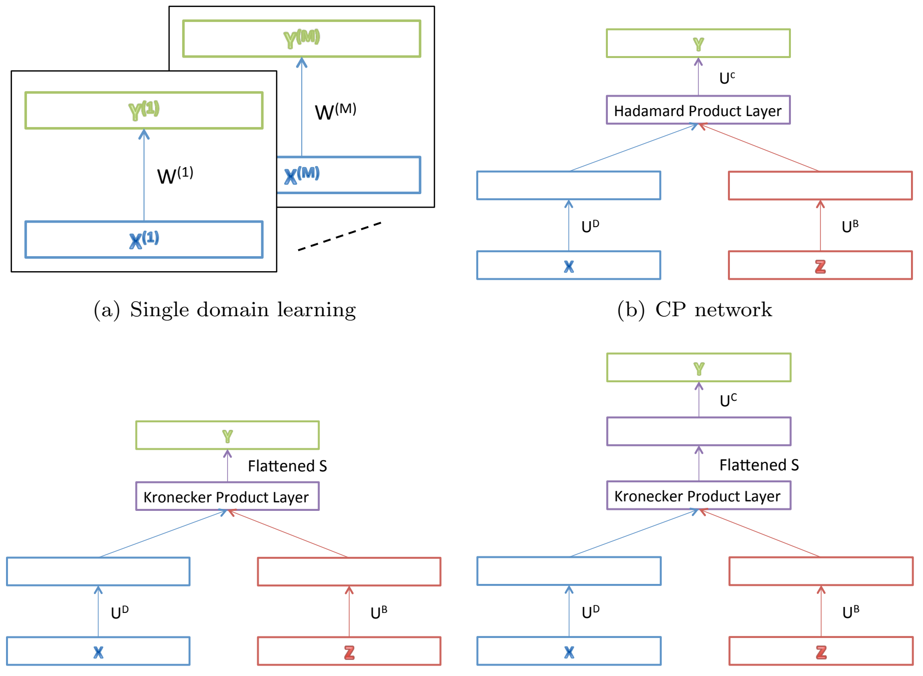
4.2.3 Gated Neural Network Architectures

We previously showed the connection between matrix factorisation for single-output models, and a two-sided neural network in Section 3.2.1. We will next draw the link between tensor factorisation and gated neural network [Sigaud et al., 2015] architectures. First we recap the factors used by different tensor (de)composition methods in Table 4.1.

To make the connection to neural networks, we need to introduce two new layers:

Hadamard Product Layer Takes as input two equal-length vectors u and v and outputs $[u_1v_1, u_2v_2, \cdots, u_Kv_K]$. It is a deterministic layer that does Hadamard (element-wise) product, where the input size is $K + K$ and output size is K.

Kronecker Product Layer Takes as input two arbitrary-length vectors u and v and outputs $[u_1v_1, u_1v_2, \cdots, u_1v_{K_u}, u_2v_1, \cdots, u_{K_u}v_{K_v}]$. It is a deterministic layer that takes input of size $K_u + K_v$ and returns the size K_uK_v Kronecker product.



(a) Single domain learning

(b) CP network

(c) TT network

(d) Tucker network

Figure 4.1: Learning multiple domains independently, versus learning with parametrised neural networks encoding the factorisation assumptions of various tensor decomposition methods.

Method/Factors	$U^{(D)}$	$U^{(C)}$	$U^{(B)}$	$\mathcal{S}$
Tucker	$U^{(D)}$	$U^{(C)}$	$U^{(B)}$	$\mathcal{S}$
CP	$U^{(D)}$	$U^{(C)}$	$U^{(B)}$	$\underline{K \times K \times K \text{ Identity Tensor}}$
TT	${U^{(D)}}^T$	$\underline{C \times C \text{ Identity Matrix}}$	$U^{(B)}$	$\mathcal{S}$
Single Output	P^T	$\underline{K \times 1 \text{ All-ones Vector}}$	Q	$\underline{K \times K \times K \text{ Identity Tensor}}$

Table 4.2: Tensor and the matrix-factorisation-based single output (Sec. 3.2.1, [Yang and Hospedales, 2015]) networks as special cases of the Tucker Network. Underlined variables are constant rather than parameters to learn.

Fig. 4.1 illustrates the approaches to multi-domain learning in terms of NNs. Single domain learning of M domains requires M single-layer NNs, each with a $D \times C$ weight matrix (Fig. 4.1(a)). Considering this set of weight matrices as the corresponding $D \times C \times M$ tensor, we can use the introduced layers to define gated networks (Figs. 4.1(b)-(d)) that model low-rank versions of this tensor with the corresponding tensor-factorisation assumptions in Eq. 4.5, 4.10, and 4.14 and summarised in Tab. 4.1. Rather than maintaining a separate NN for each domain as in Fig. 4.1(a), the networks in Fig. 4.1(b)-(d) maintain a single NN for all domains. The domain of each instance is signalled to the network via its corresponding descriptor, which the right hand side of the network uses to synthesise the recognition weights accordingly.

We note that we can further unify all three designs, as well as the single-output model proposed in Section 3.2.1, by casting them as special cases of the Tucker Network as shown in Table 4.2. Thus we can understand all these factorisation-based approaches by their connection to the idea of breaking down the stacked model parameters (matrix W or tensor $\mathcal{W}$) into a smaller number of parameters composing a domain-specific ($U^{(B)}$), task-specific ($U^{(C)}$) and shared components ($U^{(D)}$). It is important to note however that, despite our model's *factorised representation assumption* in common with tensor decomposition, the way to train our model is not by training a set of models and decomposing them – in fact matrix/tensor *decomposition* is not used at all. Rather a single Tucker network of Fig. 4.1(d) is trained end-to-end with backpropagation to minimise the multi-domain/task loss. The network architecture enforces that backpropagation trains the individual factors (Tab. 4.1) such that their corresponding tensor *composition* solves the multi-domain-multi-task problem. In summary, our framework can be seen as 'discriminatively trained' tensor factorisation, or as a gated neural network, where the NN's weights are dynamically *parametrised* by a second input, the descriptor z.

4.2.4 Zero-Shot Domain Adaptation

The mechanism of implementing ZSDA is similar to what appears in Sec. 3.2.4, i.e., we can feed an unseen domain descriptor $z^{(*)}$ into the weight generating function and get a model on-the-fly: $f_{\mathcal{W}}(z^{(*)})$.

4.3 Experiments

For the method[1] in Chapter 4, named as TuckerNN (Tucker decomposition Neural Network), we explore the application of TuckerNN for a variety of MDL and ZSDA problems including object recognition (Section 4.3.3), surveillance image analysis (Section 4.3.1) and person recognition/soft-biometrics (Section 4.3.2). The first recognition experiment follows the conventional setting of domains/tasks as atomic entities, and the latter experiments explore the potential benefits of informative domain descriptors, including zero-shot domain adaptation.

Implementation We implement our framework with TensorFlow [Abadi et al., 2015], taking the neural network interpretation of each method, thus allowing easy optimisation with SGD-based backpropagation. We use hinge loss for the binary classification problems and (categorical) cross-entropy loss for the multi-class classification problems.

4.3.1 Surveillance Image Classification

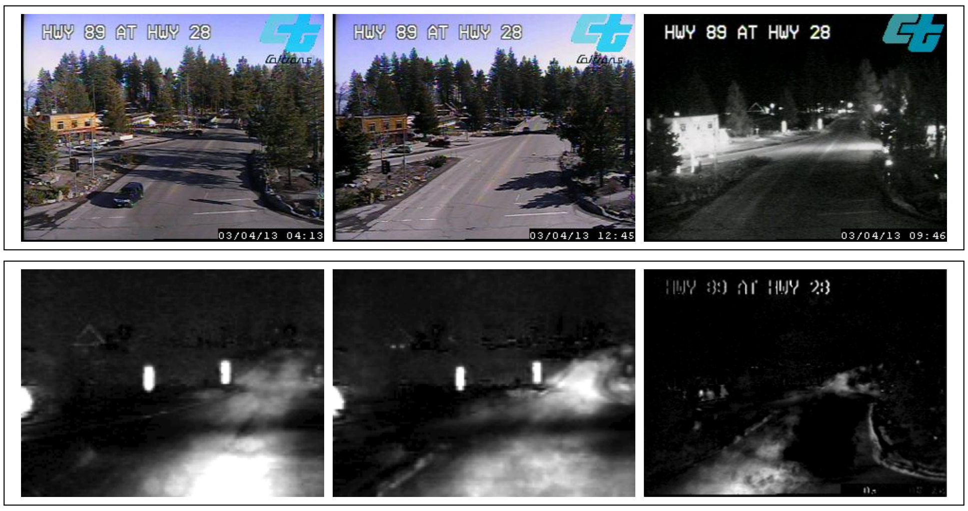


Figure 4.2: Illustration of domain factors in surveillance car recognition task. Above: Example frames illustrating day/night domain factor. (Left: With car. Middle and right: No cars.) Below: Activity map illustration of weekday/weekend domain factor. (Left: Weekday, Middle: Weekend: Right: Weekday-Weekend difference.)

In surveillance video analysis, there are many covariates such as season, workday/holiday and day/night. Each of these affects the distribution of image features, and thus introduces domain shift. Collecting the potentially years of training data required to train a single general model is both expensive and suboptimal (due to ignoring domain shift, and treating all data as a single domain). Thus in this section we explore the potential for multi-domain learning with distributed domain descriptors (Section 3.2.2) to improve performance by modelling the factorial structure in the do-

[1]We use the Tucker variant only because it generalises the other variants.

main shift. Furthermore, we demonstrate the potential of ZSDA to adapt a system to apply in a new set of conditions for which no training data exists.

Data We consider the surveillance image classification task proposed by [Hoffman et al., 2014]. This is a binary classification of each frame in a 12-day surveillance stream as being empty or containing cars. Originally, [Hoffman et al., 2014] investigated continuous domains (which can be seen as a 1-dimensional domain descriptor containing time-stamp). To explore a richer domain descriptor, we use a slightly different definition of domains, considering instead weekday/weekend and day/night as domain factors, generating $2 \times 2 = 4$ distinct domains, each encoded by a 2-of-4 binary domain descriptor. Figure 4.2(top) illustrates the more obvious domain factor: day/night. This domain-shift induces a larger image change than the task-relevant presence or absence of a car.

Settings We use the 512 dimensional GIST feature for each frame provided by [Hoffman et al., 2014]. We perform two experiments: *Multi-domain learning*, and *zero-shot domain adaptation*. For MDL, we split all domains' data into half training and half testing, and repeat for 10 random splits. We use our single-output network (Fig. 3.1, Tab. 4.2 bottom row) with a distributed domain descriptor for two categories with two states (i.e., same descriptor as Fig. 3.3, right). The baselines are: (i) **SDL**: train an independent model for each domain (ii) **Aggregation**: to train a single model covering all domains (ii) **Multi-Domain I**: a multi-domain model with low-rank factorisation of $\tilde{W}$ and one-hot encoding of domain descriptor (in this case, Z is an identity matrix thus $\tilde{W} = WZ = W$ – this roughly corresponds to our reimplementation of [Kumar and Daumé III, 2012]), and (iv) **Multi-Domain II**: a factorised multi-domain model with one-hot + constant term encoding (this is in fact the combination the sharing structure and factorisation proposed in [Evgeniou and Pontil, 2004] and [Kumar and Daumé III, 2012] respectively).

For ZSDA, we do leave-one-domain-out cross-validation: holding out one of the four domains for testing, and using the observed three domains' data for training. Although the train/test splits are not random, we still repeat the procedure 10 times to reduce randomness of the SGD optimisation. Our model is constructed on the fly for the held-out domain based on its semantic descriptor. As a baseline, we train an aggregated model from all observed domains' data, and apply it directly to the held-out domain (denoted as **Direct**). We set our rank hyper parameter via the heuristic $K = \frac{D}{\log(D)}$. We evaluate the the mean and standard deviation of error rate.

Results and Analysis The results shown in Table 4.3.1 demonstrate that our proposed method outperforms alternatives in both MDL and ZSDA settings. For MDL we see that training a per-domain model and ignoring domains altogether perform similarly (SDL vs Aggregation). By introducing more sharing structure, e.g., Multi-Domain I is built with low-rank assumption, and Multi-Domain II further assumes that there is a globally shared factor, the multi-domain models clearly improve performance. Finally our full method performs notably better than the others because it can benefit

Table 4.3: Surveillance Image Classification. Mean error rate (%) and standard deviation.

MDL Experiment	SDL	Aggregation	Multi-Domain I	Multi-Domain II	TuckerNN
Err. Rate	10.82 ± 3.90	11.05 ± 2.73	10.00 ± 0.90	8.86 ± 0.79	**8.61 ±0.51**

ZSDA Experiment	Direct	TuckerNN
Err. Rate	12.04 ±0.11	**9.72 ±0.08**

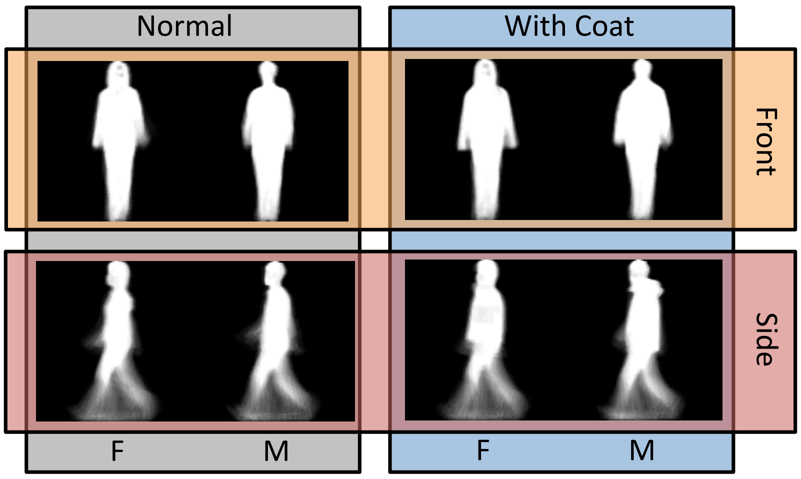


Figure 4.3: Example gait images illustrating independent domain factors.

from both low-rank modelling and also exploiting the structured information in the distributed encoding of domain semantic descriptor.

In ZSDA, our proposed method also clearly outperforms the baseline of directly training on all the source domains. What information is our model able to exploit to achieve this? One cue is that various directions including right turn are common on weekends and weekdays are primarily going straight (illustrated in Figure 4.2(below) by way of an activity map). This can, e.g., be learned from the weekend-day domain, and transferred to the held-out weekend-night domain because the domain factors inform us that those two domains have the weekend factor in common.

4.3.2 Gait-based Soft-Biometrics and Recognition

Gait-based person and soft biometric recognition are desirable capabilities due to not requiring subject cooperation [Zheng et al., 2011]. However they are challenging especially where there are multiple covariates such as viewing angle and accessory status (e.g., object carrying). Again training a model for every covariate combination is infeasible, and conventional domain adaptation is not scalable as the number of resulting domains grows exponentially with independent domain factors. In contrast, zero-shot domain adaptation could facilitate deploying a camera with a calibration step to specify covariates such as view-angle, but no data collection or re-training.

Data We investigate applying our framework to this setting using the CASIA gait analysis dataset [Zheng et al., 2011] with data from 124 individuals under 11 viewing angles. Each person has three situations: normal ('nm'), wearing overcoat ('cl') and carrying a bag ('bg'). This naturally forms $3 \times 11 = 33$ domains. We extract Gait Energy Image (GEI) features, followed by PCA reduction to 300 dimensions, retaining more than 97% of the variance.

Settings We consider two gait analysis problems: (i) Soft-biometrics: Female/Male classification and (ii) Person verification/matching. For matching each image pair x_i and x_j, generates a pairwise feature vector by $x_{ij} = |x_i - x_j|$. The objective is to learn a binary verification classifier on x_{ij} to predict if two images are the same person or not. All experiment settings (baseline methods, training/testing splits, experiments repeats, and the choice of hyper-parameter) are the same as in Section 4.3.1, except that for the verification problem we build a balanced (training and testing) set of positive/negative pairs by down-sampling negative pairs.

Results and Analysis Figure 4.3 illustrates the nature of the domain factors here, where the cross-domain variability is again large compared to the cross-class variability. Our framework uniquely models the fact that each domain factor (e.g., view angle and accessory status) can occur independently. The results shown in Tables 4.4 and 4.5 demonstrate the same conclusions – that explicitly modelling MDL structure improves performance (Multi-domain I and II improve on SDL and Aggregation), with our most general method performing best overall.

Table 4.4: Gait: Male/Female Biometrics. Error Rate (%) and Standard Deviation

MDL Experiment	SDL	Aggregation	Multi-Domain I	Multi-Domain II	TuckerNN
Err. Rate	2.35 (±0.20)	2.62 (±0.18)	2.25 (±0.20)	2.07 (± 0.15)	**1.64 (±0.14)**

ZSDA Experiment	Direct	TuckerNN
Err. Rate	3.01 (±0.08)	**2.19 (±0.05)**

Table 4.5: Gait: Person Verification. Error Rate(%) and Standard Deviation

MDL Experiment	SDL	Aggregation	Multi-Domain I	Multi-Domain II	TuckerNN
Err. Rate	23.30 (±0.28)	24.62 (±0.32)	22.18 (±0.25)	21.15 (± 0.13)	**19.36 (±0.09)**

ZSDA Experiment	Direct	TuckerNN
Err. Rate	26.93 (±0.10)	**23.67 (±0.11)**

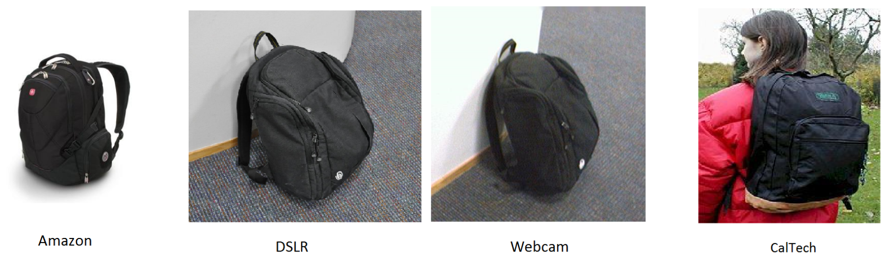


Figure 4.4: Illustration of the domains in the office dataset. An image of a backpack collected from four different sources.

4.3.3 Multi-domain Multi-task Object Recognition

In this section we assume conventional atomic domains (so domain descriptors are simply indicators rather than distributed codes), but explore a multi-domain multi-task (MDMTL) setting. Thus there is a multi-class problem within each domain, and our method (Section 4.2) exploits information sharing across both domains and tasks. To deal with multi-class recognition within each domain, it generalises existing vector-valued MTL/MDL methods, and implements a matrix-valued weight generating function parametrised by a low-rank tensor (Fig. 4.1).

Datasets We first evaluate the multi-domain multi-task setting using the well-known office dataset [Saenko et al., 2010]. Office includes three domains (data sources): *Amazon*: images downloaded from Amazon, *DSLR* high-quality images captured by digital camera, *webcam* low-quality images captured by webcam. For every domain, there are multiple classes of objects to recognise, e.g., keyboard, mug, headphones. In addition to the original Office dataset, add a 4th domain: Caltech-256 [Griffin et al., 2007], as suggested by [Gong et al., 2012]. Thus we evaluate recognising 10 classes in common the four domains. See Fig. 4.4 for an illustration. The feature is the 800-dimension SURF feature [Bay et al., 2006]. As suggested by [Gong et al., 2012], we pre-process the data by normalising the sum of each instance's feature vector to one then applying a z-score function.

Settings We compare the three proposed method variants: CP, Tucker, and TT-Networks with two baselines. **SDL:** training each domain independently and **Aggregation**: ignoring domains and training an aggregate model for all data. For these two baselines, we use a vanilla feed-forward neural network without hidden layers thus there are no hyper-parameters to tune. For our methods, the tensor rank(s), i.e., K for CP-Network, (K_D, K_C, K_B) for Tucker-Network, and (K_D, K_B) for TT-network are chosen by 10-fold cross validation. The grids of K_D, K_C, and K_B are respectively $[16, 64, 256]$, $[2, 4, 8]$, and $[2, 4]$. The multi-class recognition error rate at 9 increasing training-testing-ratios $(10\%, 20\% \ldots 90\%)$ is computed, and for each training-testing-ratio, we repeat the experiment 10 times with random splits.

Results and Analysis The result is shown in Fig. 4.5. We can see that the proposed

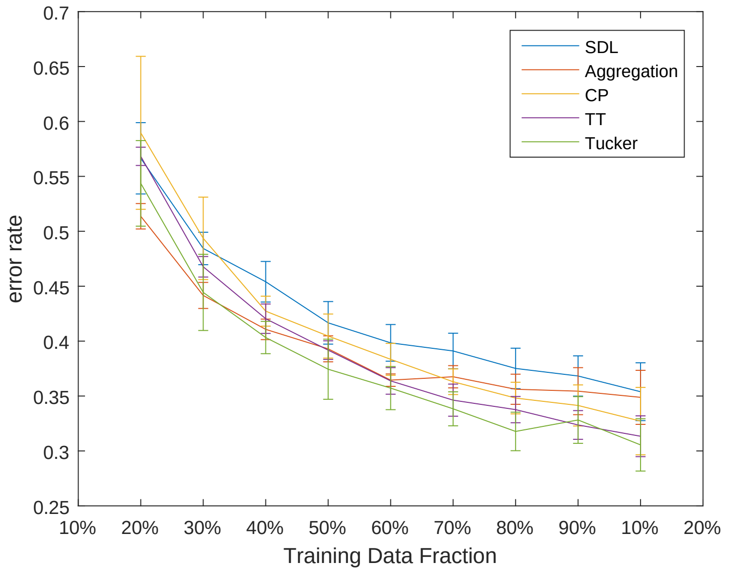


Figure 4.5: Office Dataset: Error mean and std. dev. recognising $C = 10$ classes across $M = 4$ domains. Comparison of three parametrised neural networks with SDL and Aggregation baselines.

methods perform well compared to *SDL*. When the training data is extremely small, *Aggregation* is a reasonably good choice as the benefit of more data outweighs the drawback of mixing domains. However, the existence of domain bias eventually prevents a single model from working well for all domains. Our proposed methods can produce different models for the various domains so they generally outperform the baselines. Tucker-and TT-Network are better than CP-Network because of their greater flexibility on choosing more than one tensor rank. However as a drawback, this also introduces more hyper-parameters to tune.

4.4 Summary

In this chapter, we discussed multi-domain learning – a bi-directional generalisation of domain adaptation, and zero-shot domain adaptation – an alternative to domain-generalisation approaches. We introduced a semantic domain/task descriptor to unify various existing multi-task/multi-domain algorithms within a single matrix factorisation framework. To go beyond the single output problems considered by prior methods, we generalised this framework to tensor factorisation, which allows knowledge sharing for methods parametrised by matrices rather than vectors. This allows multi-domain learning for multi-output problems or simultaneous multi-task-multi-domain learning. All these approaches turn out to have equivalent interpretations as neural networks, which allow easy implementation and optimisation with existing toolboxes. Promising lines of future enquiry include extending this framework for end-to-end learning in convolutional neural networks (Chapter 5), tensor rank-based regularisation (Chapter 6), and applying these ideas to solve practical computer vision problems.

Chapter 5

Going Deep: Tensor Factorisation for Deep MTL

Most contemporary multi-task learning methods assume linear models. This setting is considered *shallow* in the era of deep learning. In this chapter, we present a new deep multi-task representation learning framework that learns cross-task sharing structure *at every layer in a deep network*. In this chapter, we adapt the tensor factorisation techniques used in Chapter 4, and take a further step that applies it to deep neural networks in a layer-wise fashion, in order to realise automatic learning of end-to-end knowledge sharing in the neural networks. This is in contrast to existing deep learning approaches that need a user-defined multi-task sharing strategy – usually implemented by tying the parameters of different neural networks. Our approach applies to both homogeneous and heterogeneous MTL. Experiments demonstrate the efficacy of our deep multi-task representation learning in terms of both higher accuracy and fewer design choices.

5.1 Background

The paradigm of multi-task learning is to learn multiple related tasks simultaneously so that knowledge obtained from each task can be re-used by the others. Early work in this area focused on neural network models [Caruana, 1997], while more recent methods have shifted focus to kernel methods, sparsity and low-dimensional task representations of linear models [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012]. Nevertheless given the impressive practical efficacy of contemporary deep neural networks (DNN)s in many important applications, we are motivated to revisit MTL from a deep learning perspective.

While the machine learning community has focused on MTL for shallow linear models recently, applications have continued to exploit neural network MTL [Zhang et al., 2014, Liu et al., 2015]. The typical design pattern dates back at least 20 years [Caruana, 1997]: define a DNN with shared lower representation layers, which then forks into separate layers and losses for each task. The sharing structure is defined manually: full-sharing up to the fork, and full separation after the fork. However this complicates DNN architecture design because the user must specify the sharing structure: How many task specific layers? How many task independent layers? How to structure sharing if there are many tasks of varying relatedness?

In this chapter we present a method for end-to-end multi-task learning in DNNs. This contribution can be seen as generalising shallow MTL methods [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012] to learning how to share *at every layer* of a deep network; or as learning the sharing structure for deep MTL [Caruana, 1997, Zhang et al., 2014, Spieckermann et al., 2014, Liu et al., 2015] which currently must be defined manually on a problem-by-problem basis.

Before proceeding it is worth explicitly distinguishing some different problem settings, which have all been loosely referred to as MTL in the literature. **Homogeneous MTL:** Each task corresponds to a *single* output. For example, MNIST digit recognition is commonly used to evaluate MTL algorithms by casting it as 10 binary classification tasks [Kumar and Daumé III, 2012]. **Heterogeneous MTL:** Each task corresponds to a unique set of output(s) [Zhang et al., 2014]. For example, one may want simultaneously predict a person's age (task one: multi-class classification or regression) as well as identify their gender (task two: binary classification) from a face image. **Multi-Domain Learning:** Each "task" corresponds to a dataset [Yang and Hospedales, 2015]. For example, one jointly can train a multi-class object recognition model for images captured by an HD camera (task/domain one) and for those captured by a webcam (task/domain two). The key difference between MTL and MDL is that MTL deals with a change in the target label-space, while MDL deals with differences due input data statistics. Within the machine learning community, key MTL algorithms [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012] (linear or kernelised versions) have been designed for the problem with single-output (e.g., binary classification, or single-output regression), and thus have been applied to homogeneous MTL and a special case of MDL when each task just has a single output. Heterogeneous MTL has not been studied systemically, but is popular in applications, which typically use multi-objective (i.e., multiple loss function) neural networks [Zhang et al., 2014, Liu et al., 2015].

In this chapter, we propose a multi-task learning method that works on all these settings. The key idea is to use tensor factorisation to divide each set of model parameters (i.e., both FC weight matrices, and convolutional kernel tensors) into *shared* and *task-specific* parts. It is a natural generalisation of shallow MTL methods that explicitly or implicitly are based on matrix factorisation [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012, Daumé III, 2007]. As linear methods, these typically require pre-engineered features. In contrast, as a deep network, our generalisation can learn directly from raw image data, determining sharing structure in a layer-wise fashion. For the simplest NN architecture – no hidden layer, single output – our method reduces to matrix-based ones, therefore matrix-based methods including

[Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012, Daumé III, 2007] are special cases of ours.

5.2 Methodology

Preliminaries

The concept of tensor and its operations has been summarised in Appx. A.4.4

Matrix-based Knowledge Sharing

Assume we have T linear models (tasks) parametrised by D-dimensional weight vectors, so the collection of all models forms a size $D \times T$ matrix W. One commonly used MTL approach [Kumar and Daumé III, 2012] is to place a structure constraint on W, e.g., $W = LS$, where L is a $D \times K$ matrix and S is a $K \times T$ matrix. This factorisation recovers a *shared* factor L and a *task-specific* factor S. One can see the columns of L as latent basis tasks, and the model $w^{(i)}$ for the ith task is the linear combination of those latent basis tasks with task-specific information $S_{\cdot,i}$.

$$w^{(i)} := W_{\cdot,i} = LS_{\cdot,i} = \sum_{k=1}^{K} L_{\cdot,k} S_{k,i} \tag{5.1}$$

From Single to Multiple Outputs

Consider extending this matrix factorisation approach to the case of multiple outputs. The model for each task is then a $D_1 \times D_2$ matrix, for D_1 input and D_2 output dimensions. The collection of all those matrices constructs a $D_1 \times D_2 \times T$ tensor. A straightforward extension of Eq. 5.1 to this case is

$$W^{(i)} := \mathcal{W}_{\cdot,\cdot,i} = \sum_{k=1}^{K} \mathcal{L}_{\cdot,\cdot,k} S_{k,i} \tag{5.2}$$

This is equivalent to imposing the same structural constraint on $W_{(3)}^{T}$ (transposed mode-3 flattening of $\mathcal{W}$). It is important to note that this allows knowledge sharing across the tasks *only*. I.e., knowledge sharing is only across-tasks not across dimensions within a task. However it may be that the knowledge learned in the mapping to one output dimension may be useful to the others within one task. E.g., consider recognising photos of handwritten and print digits – it may be useful to share across handwritten-print; as well as across different digits within each. In order to support general knowledge sharing across both tasks and outputs within tasks, we propose to use more general tensor factorisation techniques. Unlike matrix factorisation, there are multiple definitions of tensor factorisation, and we use Tucker [Tucker, 1966] and Tensor Train (TT) [Oseledets, 2011] decompositions.

5.2.1 Tensor Factorisation for Knowledge Sharing

In Section 4.2.2 we reviewed three classic tensor decomposition methods for the case of 3-way tensors, in this section, we introduce the general form of Tucker and TT decomposition that applies to N-way tensors.

Tucker Decomposition

Given an N-way tensor of size $D_1 \times D_2 \cdots \times D_N$, Tucker decomposition outputs a core tensor $\mathcal{S}$ of size $K_1 \times K_2 \cdots \times K_N$, and N matrices $U^{(n)}$ of size $D_n \times K_n$, such that,

$$\mathcal{W} = \mathcal{S} \bullet_{(1,2)} U^{(1)} \bullet_{(1,2)} U^{(2)} \bullet \cdots \bullet_{(1,2)} U^{(N)} \tag{5.3}$$

Tensor Train Decomposition

Tensor Train (TT) Decomposition outputs 2 matrices $U^{(1)}$ and $U^{(N)}$ of size $D_1 \times K_1$ and $K_{N-1} \times D_N$ respectively, and $(N-2)$ 3-way tensors $\mathcal{U}^{(n)}$ of size $K_{n-1} \times D_n \times K_n$. The elements of $\mathcal{W}$ can be computed by,

$$\mathcal{W} = U^{(1)} \bullet \mathcal{U}^{(2)} \bullet \cdots \bullet U^{(N)} \tag{5.4}$$

Knowledge Sharing

If the final axis of the input tensor above indexes tasks, i.e. if $D_N = T$ then the last factor $U^{(N)}$ in both decompositions encodes a matrix of task specific knowledge, and the other factors encode shared knowledge.

5.2.2 Deep Multi-Task Representation Learning

To realise deep multi-task representation learning (DMTRL), we learn one DNN per-task each with the same architecture[1]. However each corresponding layer's weights are generated with one of the knowledge sharing structures in Eq. 5.2, Eq. 5.3 or Eq. 5.4. Note that we apply these 'right-to-left' in order to generate weight tensors with the specified sharing structure, rather than actually applying Tucker or TT to decompose an input tensor. Thus in the forward pass, we synthesise weight tensors $\mathcal{W}$ and perform inference as usual.

Our weight generation (construct tensors from smaller pieces) does not introduce non-differentiable terms, so our deep multi-task representation learner is trainable via standard backpropagation. Specifically, in the backward pass over FC layers, rather than directly learning the 3-way tensor $\mathcal{W}$, our methods learn either $\{\mathcal{S}, U_1, U_2, U_3\}$ (Tucker, Eq. 5.3), $\{U_1, \mathcal{U}_2, U_3\}$ (TT, Eq. 5.4), or in the simplest case $\{\mathcal{L}, S\}$ (SVD,

[1]Except heterogeneous MTL, where the output layer is necessarily unshared due to different dimensionality.

Eq. 5.2). Besides FC layers, contemporary DNN designs often exploit convolutional layers. Those layers usually contain kernel filter parameters that are 3-way tensors of size $H \times W \times C$, (where H is height, W is width, and C is the number of input channels) or 4-way tensors of size $H \times W \times C \times M$, where M is the number of filters in this layer (i.e., the number of output channels). The proposed methods naturally extend to convolution layers as convolution just adds more axes on the left-hand side. E.g., the collection of parameters from a given convolutional layer of T neural networks forms a tensor of shape $H \times W \times C \times M \times T$.

These knowledge sharing strategies provide a way to *softly* share parameters across the corresponding layers of each task's DNN: where, what, and how much to share are learned from data. This is in contrast to the conventional Deep-MTL approach of manually selecting a set of layers to undergo *hard* parameter sharing: by tying weights so each task uses exactly the same weight matrix/tensor for the corresponding layer [Zhang et al., 2014, Liu et al., 2015]; and a set of layers to be completely separate: by using independent weight matrices/tensors. In contrast our approach benefits from: (i) automatically learning this sharing structure from data rather than requiring user trial and error, and (ii) smoothly interpolating between fully shared and fully segregated layers, rather than a hard switching between these states. An illustration of the proposed framework for different problem settings can be found in Fig. 5.1.

5.3 Experiments

We next validate the Deep Multi-Task Representation Learning (DMTRL) method in Chapter 5 under three experimental settings: Homogeneous MTL, Heterogeneous MTL, and Multi-Domain Learning. We evaluate the low-rank tensor approach to knowledge sharing, and show that it provides an effective alternative to the traditional expert/exhaustive approach to sharing architecture design.

Implementation The method is implemented with TensorFlow [Abadi et al., 2015]. The code is released on GitHub[2]. For DMTRL-Tucker, DMTRL-TT, and DMTRL-LAF, we need to assign the rank of each weight tensor. The DNN architecture itself may be complicated and so can benefit from different ranks at different layers, but grid-search is impractical. However, since both Tucker and TT decomposition methods have SVD-based solutions ([Lathauwer et al., 2000, Oseledets, 2011]), and vanilla SVD is directly applicable to DMTRL-LAF, we can initialise the model and set the ranks as follows: First train the DNNs independently in single task learning mode. Then pack the layer-wise parameters as the input for tensor decomposition. When SVD is applied, set a threshold for relative error so SVD will pick the appropriate rank. Thus our method needs only a *single* hyper parameter of max reconstruction error (we set to $\epsilon = 10\%$ throughout) that indirectly specifies the ranks of every layer. Note that training from random initialisation also works, but the STL-based initialisation makes rank selection

[2]`https://github.com/wOOL/DMTRL`

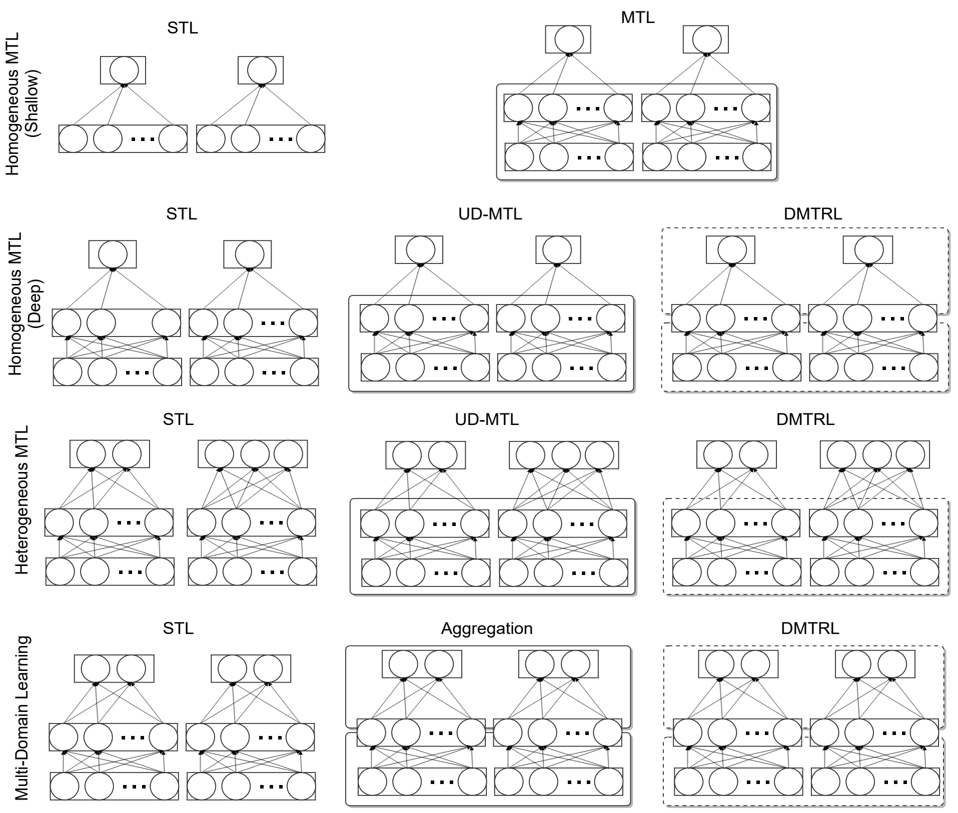


Figure 5.1: Illustrative example with two tasks corresponding to two neural networks in homogeneous (single output) and heterogeneous (different output dimension) cases. Weight layers grouped by solid rectangles are tied across networks. Weight layers grouped by dashed rectangles are softly shared across networks with our method. Ungrouped weights are independent.
Homogeneous MTL Shallow: Left is single task (two independent networks); right is MTL. In the case of vector input and no hidden layer, our method is equivalent to conventional matrix-based MTL methods. Homogeneous MTL Deep: Single task (Left) is independent networks. User-defined-MTL (UD-MTL) selects layers to share/separate. Our DMTRL learns sharing at every layer. Heterogeneous MTL: UD-MTL selects layers to share/separate. Our DMTRL learns sharing at every shareable layer. MDL: Left is single task (independent networks). Middle is to aggregate all domains. Right is our DMTRL where soft sharing structure for both layers is learned.

easy and transparent. Nevertheless, like [Kumar and Daumé III, 2012] the framework is not sensitive to rank choice so long as they are big enough. If random initialisation is desired to eliminate the pre-training requirement, good practice is to initialise parameter tensors by a suitable random weight distribution first, then do decomposition, and use the decomposed values for initialising the factors (the *real* learnable parameters in our framework). In this way, the resulting re-composed tensors will have approximately the intended distribution. Our sharing is applied to weight parameters only, bias terms are not shared. Apart from initialisation, decomposition is not used anywhere.

5.3.1 Homogeneous MTL

Dataset, Settings and Baselines We use MNIST handwritten digits. The task is to recognise digit images zero to nine. When this dataset is used for the evaluation of MTL methods, ten 1-vs-all binary classification problems usually define ten tasks [Kumar and Daumé III, 2012]. The dataset has a given train (60,000 images) and test (10,000 images) split. Each instance is a monochrome image of size $28 \times 28 \times 1$.

We use a modified LeNet [LeCun et al., 1998] as the CNN architecture. The first convolutional layer has 32 filters of size 5×5, followed by 2×2 max pooling. The second convolutional layer has 64 filters of size 4×4, and again a 2×2 max pooling. After these two convolutional layers, two fully connected layers with 512 and 1 output(s) are placed sequentially. The convolutional layers and first FC layer use RELU $f(x) = \max(x, 0)$ as the activation function. The loss is hinge loss, $\ell(y) = \max(0, 1 - \hat{y} \cdot y)$, where $y \in \pm 1$ is the true label and $\hat{y}$ is the output of each task's neural network.

Conventional matrix-based MTL methods [Evgeniou and Pontil, 2004, Argyriou et al., 2008, Kumar and Daumé III, 2012, Romera-Paredes et al., 2013, Wimalawarne et al., 2014] are linear models taking vector input only, so they need a preprocessing that flattens the image into a vector, and typically reduce dimension by PCA. As per our motivation for studying *Deep* MTL, our methods will decisively outperform such shallow linear baselines. Thus to find a stronger MTL competitor, we instead search user defined architectures for Deep-MTL parameter sharing (cf [Zhang et al., 2014, Liu et al., 2015, Caruana, 1997]). In all of the four parametrised layers (pooling has no parameters), we set the first N $(1 \leq N \leq 3)$ to be *hard* shared[3]. We then use cross-validation to select among the three user-defined MTL architectures and the best option is $N = 3$, i.e., the first three layers are fully shared (we denote this model UD-MTL). For our methods, all four parametrised layers are softly shared with the different factorisation approaches. To evaluate different MTL methods and a baseline of single task learning (STL), we take ten different fractions of the given 60K training split, train the model, and test on the 10K testing split. For each fraction, we repeat the experiment 5 times with randomly sampled training data. We report two performance metrics: (1) the mean error rate of the ten binary classification problems and (2) the error rate of recognising a digit by ranking each task's 1-vs-all output (multi-class classification error).

Results As we can see in Fig. 5.2, all MTL approaches outperform STL, and the advantage is more significant when the training data is small. The proposed methods, DMTRL-TT and DMTRL-Tucker outperform the best user-defined MTL when the training data is very small, and their performance is comparable when the training data is large.

[3]This is not strictly all possible user-defined sharing options. For example, another possibility is the first convolutional layer and the first FC layer could be fully shared, with the second convolutional layer being independent (task specific). However, this is against the intuition that lower/earlier layers are more task agnostic, and later layers more task specific. Note that sharing the last layer is technically possible but not intuitive, and in any case not meaningful unless at least one early layer is unshared, as the tasks are different.

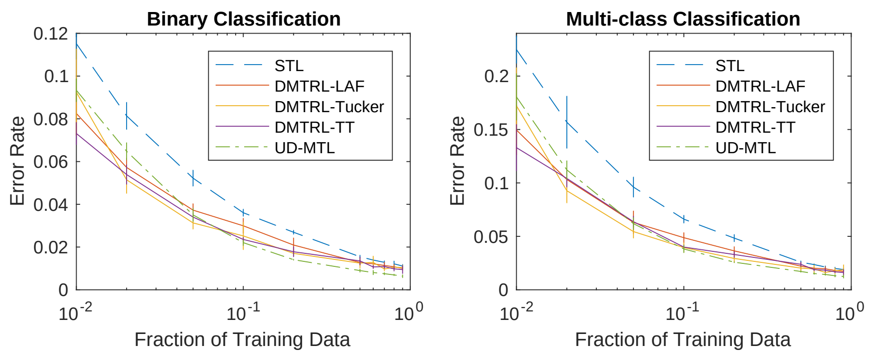


Figure 5.2: Homogeneous MTL: digit recognition on MNIST dataset. Each digit provides a task.

Further Discussion

For a slightly unfair comparison, in the case of binary classification with 1000 training data, shallow matrix-based MTL methods with PCA feature [Kang et al., 2011, Kumar and Daumé III, 2012] reported 14.0% / 13.4% error rate. With the same amount of data, our methods have error rate below 6%. This shows the importance of our deep end-to-end multi-task representation learning contribution versus conventional shallow MTL. Since the error rates in [Kang et al., 2011, Kumar and Daumé III, 2012] were produced on a private subset of MNIST dataset with PCA representations only, to ensure a direct comparison, we implement several classic MTL methods and compare them.

We provide a comparison with classic (shallow, matrix-based) MTL methods for the first experiment (MNIST, binary one-vs-rest classification, 1% training data, mean of error rates for 10-fold CV).

A subtlety in making this comparison is what feature should the classic methods use? Conventionally they use a PCA feature (obtained by flattening the image, then dimension reduction by PCA). However for visual recognition tasks, performance is better with deep features – a key motivation for our focus on deep approaches to MTL. We therefore also compare the classic methods when using a feature extracted from the penultimate layer of the CNN network used in our experiment.

Model	PCA Feature	CNN Feature
Single Task Learning	16.89	11.52
Evgeniou and Pontil [2004]	15.27	10.32
Argyriou et al. [2008]	15.64	9.56
Kumar and Daumé III [2012]	14.08	9.41
DMTRL-LAF	-	8.25
DMTRL-Tucker	-	9.24
DMTRL-TT	-	7.31
UD-MTL	-	9.34

Table 5.1: Comparison with classic MTL methods. MNIST binary classification error rate (%).

As expected, the classic methods improve on STL, and they perform significantly better with CNN than PCA features. However, our DMTRL methods still outperform the best classic methods, even when they are enhanced by CNN features. This is due to soft (cf hard) sharing of the feature extraction layers and the ability of end-to-end training of both the classifier and feature extractor. Finally, we note that more fundamentally, the classic methods are restricted to binary problems (due to their matrix-based nature) and so, unlike our tensor-based approach, they are unsuitable for multi-class problems like omniglot and age-group classification.

For readers interested in the connection to model capacity (number of parameters), we list the number of parameters for each model in the first experiment (MNIST, binary one-vs-rest classification) and the performance (1% training data, mean of error rate for 10-fold CV).

Model	Error Rate (%)	Number of parameters	Ratio
STL	11.52	4351K	1.00
DMTRL-LAF	8.25	1632K	0.38
DMTRL-Tucker	9.24	1740K	0.40
DMTRL-TT	7.31	2187K	0.50
UD-MTL	9.34	436K	0.10
UD-MTL-Large	9.39	1644K	0.38

Table 5.2: Comparison of deep models: Error rate and number of parameters.

The conventional hard-sharing method (UD-MTL) design is to share all layers except the top layer. Its number of parameter is roughly 10% of the single task learning method (STL), as most parameters are shared across the 10 tasks corresponding to 10 digits. Our soft-sharing methods also significantly reduce the number of parameters compared to STL, but are larger than UD-MTL's hard sharing.

To compare our method to UD-MTL, while controlling for network capacity, we expanded UD-MDL by adding more hidden neurons so its number of parameter is close to our methods (denoted UD-MTL-Large). However UD-MDL performance does not increase. This is evidence that our model's good performance is not simply due to greater capacity than UD-MTL.

5.3.2 Heterogeneous MTL: Face Analysis

Dataset, Settings and Baselines The AdienceFaces [Eidinger et al., 2014] is a large-scale face images dataset with the labels of each person's gender and age group. We use this dataset for the evaluation of heterogeneous MTL with two tasks: (i) gender classification (two classes) and (ii) age group classification (eight classes). Two independent CNN models for this benchmark are introduced in [Levi and Hassncer, 2015]. The two CNNs have the same architecture except for the last fully-connected layer, since the heterogeneous tasks have different number of outputs (two / eight). We take these CNNs from [Levi and Hassncer, 2015] as the STL baseline. We again search for the best possible user-defined MTL architecture as a strong competitor: the proposed

CNN has six layers – three convolutional and three fully-connected layers. The last fully-connected layer has non-shareable parameters because they are of different size. To search the MTL design-space, we try setting the first N ($1 \leq N \leq 5$) layers to be hard shared between the tasks. Running 5-fold cross-validation on the train set to evaluate the architectures, we find the best choice is $N = 5$ (i.e., all layers fully shared before the final heterogeneous outputs). For our proposed methods, all the layers before the last heterogeneous dimensionality FC layers are softly shared.

We select increasing fractions of the AdienceFaces train split randomly, train the model, and evaluate on the same test set. For reference, there are 12245 images with gender labelled for training, 4007 ones for testing, and 11823 images with age group labelled for training, and 4316 ones for testing.[4].

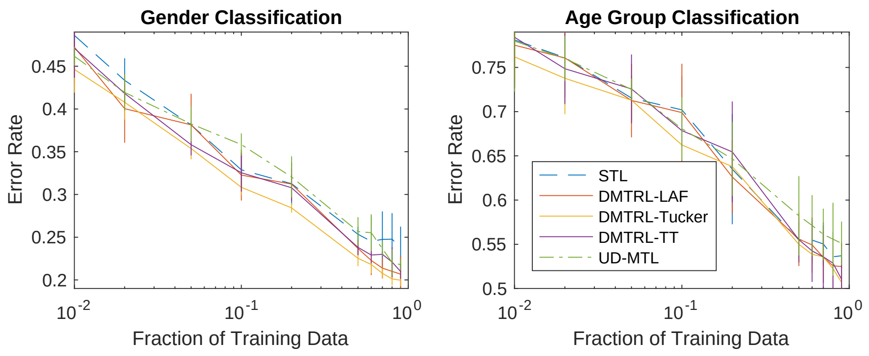


Figure 5.3: Heterogeneous MTL: Age and Gender recognition in AdienceFace dataset.

Results Fig. 5.3 shows the error rate for each task. For the gender recognition task, we find that: (i) User-defined MTL is not consistently better than STL, but (ii) our methods, esp., DMTRL-Tucker, consistently outperform both STL and the best user-defined MTL. For the harder age group classification task, our methods generally improve on STL. However UD-MTL does not consistently improve on STL, and even reduces performance when the training set is bigger. This is the negative transfer phenomenon [Rosenstein et al., 2005], where using a transfer learning algorithm is worse than not using it. This difference in outcomes is attributed to sufficient data eventually providing some effective task-specific representation. Our methods can discover and exploit this, but UD-MTL's hard switch between sharing and not sharing can not represent or exploit such increasing task-specificity of representation.

5.3.3 Heterogeneous MTL: Multi-Alphabet Recognition

Dataset, Settings and Baselines We next consider the task of learning to recognise handwritten letters *in multiple languages* using the Omniglot [Lake et al., 2015] dataset. Omniglot contains handwritten characters in 50 different alphabets (e.g., Cyrillic, Korean, Tengwar), each with its own number of unique characters ($14 \sim 55$). In total,

[4]`https://github.com/GilLevi/AgeGenderDeepLearning/tree/master/Folds/train_val_txt_files_per_fold/test_fold_is_0`

there are 1623 unique characters, and each has exactly 20 instances. Here each task corresponds to an alphabet, and the goal is to recognise its characters. MTL has a clear motivation here, as cross-alphabet knowledge sharing is likely to be useful as one is unlikely to have extensive training data for a wide variety of less common alphabets.

The images are monochrome of size 105×105. We design a CNN with 3 convolutional and 2 FC layers. The first conv layer has 8 filters of size 5×5; the second conv layer has 12 filters of size 3×3, and the third convolutional layer has 16 filters of size 3×3. Each convolutional layer is followed by a 2×2 max-pooling. The first FC layer has 64 neurons, and the second FC layer has size corresponding to the number of unique classes in the alphabet. The activation function is *tanh*.

We use a similar strategy to find the best user-defined MTL model: the CNN has 5 parametrised layers, of which 4 layers are potentially shareable. So we tried hard-sharing the first N $(1 \leq N \leq 4)$ layers. Evaluating these options by 5-fold cross-validation, the best option turned out to be $N = 3$, i.e., the first *three* layers are hard shared. For our methods, *all four* shareable layers are softly shared.

Since there is no standard train/test split for this dataset, we use the following setting: We repeatedly pick at random $5, \ldots 90\%$ of images per class for training. Note that 5% is the minimum, corresponding to one-shot learning. The remaining data are used for evaluation.

Results Fig. 5.4 reports the average error rate across all 50 tasks (alphabets). Our proposed MTL methods surpass the STL baseline in all cases. User-defined MTL does not work well when the training data is very small, but does help when training fraction is larger than 50%.

Measuring the Learned Sharing Compared to the conventional user-defined sharing architectures, our method learns how to share from data. We next try to quantify the amount of sharing estimated by our model on the Omniglot data. Returning to the key factorisation $\mathcal{W} = \mathcal{L}S$, we can find that S-like matrix appears in all variants of proposed method. It is S in DMTRL-SVD, the transposed $U^{(N)}$ in DMTRL-Tucker, and $U^{(N)}$ in DMTRL-TT (N is the last axis of $\mathcal{W}$). S is a $K \times T$ size matrix, where T is the number of tasks, and K is the number of latent tasks [Kumar and Daumé III, 2012] or the dimension of task coding [Yang and Hospedales, 2015]. Each column of S is a set of coefficients that produce the final weight matrix/tensor by linear combination. If we put STL and user-defined MTL (for a certain shared layer) in this framework, we see that STL is to *assign* (rather than *learn*) S to be an identity matrix I_T. Similarly, user-defined MTL (for a certain shared layer) is to assign S to be a matrix with all zeros but one particular row is all ones, e.g., $S = [\mathbf{1}_{1\times T}; \mathbf{0}]$. Between these two extremes, our method learns the sharing structure in S. We propose the following equation to measure the learned sharing strength:

$$\rho = \frac{1}{\binom{T}{2}} \sum_{i<j} \Omega(S_{\cdot,i}, S_{\cdot,j}) = \frac{2}{T(T-1)} \sum_{i<j} \Omega(S_{\cdot,i}, S_{\cdot,j}) \tag{5.5}$$

Here $\Omega(a, b)$ is a similarity measure for two vectors a and b and we use cosine

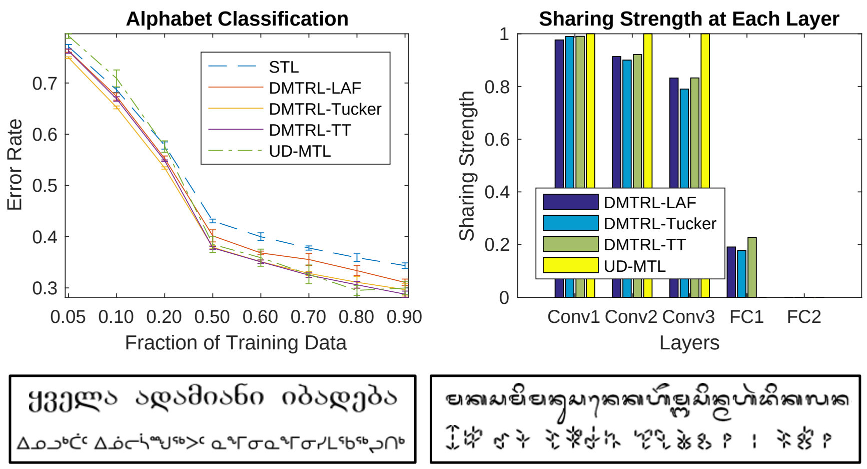


Figure 5.4: Results of multi-task learning of multilingual character recognition (Omniglot dataset). Below: Illustration of the language pairs estimated to be the most related (left - Georgian Mkhedruli and Inuktitut) and most unrelated (right - Balinese and ULOG) character recognition tasks.

similarity. ρ is the average on all combinations of column-wise similarity. So ρ measures how much sharing is encoded by S between $\rho = 0$ for STL (nothing to share) and $\rho = 1$ for user-defined MTL (completely shared).

Since S is a real-valued matrix in our scenario, we normalise it before applying Eq. 5.5: First we take absolute values, because large either positive or negative value suggests a significant coefficient. Second we normalise each column of S by applying a softmax function, so the sum of every column is 1. The motivation behind the second step is to make a matched range of our S with $S = I_T$ or $S = [\mathbf{1}_{1\times T}; \mathbf{0}]$, as for those two cases, the sum of each column is 1 and the range is $[0, 1]$.

For the Omniglot experiment, we plot the measured sharing amount for training fraction 10%. Fig. 5.4 reveals that three proposed methods tend to share more for bottom layers ('Conv1', 'Conv2', and 'Conv3') and share less for top layer ('FC1'). This is qualitatively similar to the best user-defined MTL, where the first three layers are fully shared ($\rho = 1$) and the 4th layer is completely not shared ($\rho = 0$). However, our methods: (i) learn this structure in a purely data-driven way and (ii) benefits from the ability to smoothly interpolate between high and low degrees of sharing as depth increases. As an illustration, Fig. 5.4 also shows example text from the most and least similar language pairs as estimated at our multilingual character recogniser's FC1 layer (the result can vary across layers).

5.3.4 Multi-Domain Learning

Dataset, Settings and Baselines For MDL, we consider a digit recognition problem with three different datasets – MNIST, USPS, and SVHN. Here a domain now corresponds to dataset, and the tasks are multi-class (digits 0-9) recognition. USPS is a smaller handwritten digit dataset with 7291 training and 2007 testing images. The Street View House Numbers (SVHN) [Netzer et al., 2011] is an image dataset that contains photos of door numbers. SVHN 'Format 2' has cropped digits, which makes it a MNIST-like problem. However it is harder than MNIST as more than one digit may appear in an image, and the model should correctly label the one in the centre. Directly applying a model trained on one of these tasks to another would not yield good performance due to domain shift [Saenko et al., 2010].

The multi-domain learning problem setting considered in this section is related to supervised domain adaptation (where all domains have labels) [Saenko et al., 2010], however the objective is to perform well in all domains rather than only on one special target domain. Few existing MTL methods can deal with this multi-class recognition problem, as they usually assume each task has a single output. Applying the strategy in the last section (heterogeneous MTL) is technically possible, but not intuitive because in this case each task is the same – classifying ten digits, and task difference is essentially from dataset bias. Therefore we compare our methods with two single task learning modes. STL is to train three independent models for MNIST, USPS, and SVHN respectively. ALL is to train a single model that ignores domain information, aggregating all data sources. We use the pre-given train/test splits for all three datasets in this experiment. We increase the size of MNIST/USPS images from $28 \times 28/16 \times 16$ to 32×32, and copy the data to each of three colour channels, so that all data have the same $32 \times 32 \times 3$ dimensions. For the network architecture, we use the same design as [Goodfellow et al., 2013]. **Results** Fig. 5.5 shows that USPS gets the most benefit

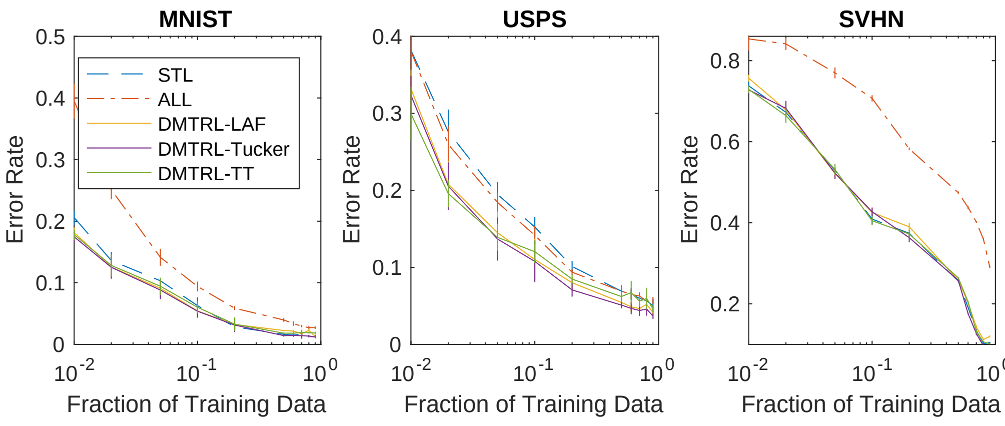


Figure 5.5: Multi-domain learning on MNIST, USPS and SVHN datasets. Each provides a 10-way multi-class recognition task. (Fraction of training data is domain-wise).

from multi-task learning with the other datasets. USPS dataset has a train-test bias, with the images in testing set being harder than the training set. MNIST images serve as a good auxiliary data for transfer in this case. MNIST benefits slightly from MTL in the small data condition, and SVHN not at all. Conceptually, the presence of USPS could be beneficial for MNIST but, as USPS is a much smaller dataset compared to MNIST, its contribution is not significant. It is perhaps unsurprising that MNIST/USPS does not help SVHN due to the extreme difference between written/photo digit appearances. Their significant difference can be seen in that a well trained MNIST model performs just slightly better than random on SVHN. On the other hand, it is noteworthy that the ALL baseline experiences significant negative transfer on both SVHN and MNIST in particular when the amount of data is small. In contrast, despite the huge dataset shift, by learning how much to share, our methods do not suffer from negative transfer.

5.4 Summary

In this chapter, we propose a novel framework for end-to-end multi-task representation learning in contemporary deep neural networks. The key idea is to generalise matrix factorisation-based multi-task ideas to tensor factorisation, in order to flexibly share knowledge in fully connected and convolutional DNN layers. Our method provides consistently better performance than single task learning and comparable or better performance than the best results from exhaustive search of user-defined MTL architectures. It reduces the design choices and architectural search space that must be explored in the workflow of Deep MTL architecture design [Caruana, 1997, Zhang et al., 2014, Liu et al., 2015], relieving researchers of the need to decide how to structure layer sharing/segregation. Instead sharing structure is determined in a data-driven way on a layer-by-layer basis that moreover allows a smooth interpolation between sharing and not sharing in progressively deeper layers. In the next chapter, we will present a regularisation-based solution for the same problem setting.

Chapter 6

Tensor Trace Norm Regularised Deep MTL

In this chapter, we propose an alternative approach for training multiple neural networks simultaneously, in contrast to the tensor factorisation based approach in Chapter 5. The core idea in this chapter is that the parameters from all models at the same layer are stacked and regularised by the tensor trace norm, so that each neural network is encouraged to reuse others' parameters at every layer. We can tell that this idea is related to the approach in Chapter 5 that can be understood as the parameters from all models at the same layer are stacked and factorised.

6.1 Background

This is the sister work of Chapter 5, and we take a regularisation approach instead of explicitly factorising the tensor. The motivation is the same: we aim at a data-driven sharing strategy instead of a manually-designed one. The reason why regularisation based solution is potentially better is that the trace norm is a continuous relaxation of discrete rank, which indicates that it is a more fine-grained control of model complexity than choosing a fixed low-rank factorisation. It is more obvious when the number of tasks is small, for example, when we have two tasks, we have the dilemma on choosing task-axis rank: if it is one, then each model is just a rescaled version of a single latent model, which is very unlikely unless these two tasks are very similar; if it is two, then it increases, rather than reduce the number of parameters, which is not very common in MTL methodology, and makes it necessary to regularise the latent tasks parameters. The drawback of this approach is that it is more computational expensive due to the non-differentiable nature of trace norm.

6.2 Methodology

Instead of predefining a parameter sharing strategy, we propose the following framework:

For T tasks, each is modelled by a neural network of the same architecture[1]. We collect the parameters into a single tensor of one higher order than the individual layer parameter tensors, and put a tensor norm on every collection.

We illustrate our framework by a simple example: assuming that we have $T = 2$ tasks, and each is modelled by a 4-layer convolution neural network (CNN). The CNN architecture is: (1) convolutional layer ('conv1') of size $5 \times 5 \times 3 \times 32$, (2) convolutional layer ('conv2') of size $3 \times 3 \times 32 \times 64$, (3) fully-connected layer ('fc1') of size 256×256, (4) fully-connected layer ('fc2'$^{(1)}$) of size 256×10 for the first task and fully-connected layer ('fc2'$^{(2)}$) of size 256×20 for the second task.

Note that we assume these two tasks have different number of outputs. In our framework, the shareable layers are 'conv1', 'conv2', and 'fc1'.

For single task learning mode, the parameters are 'conv1'$^{(1)}$, 'conv2'$^{(1)}$, 'fc1'$^{(1)}$, and 'fc2'$^{(1)}$ for the first task; 'conv1'$^{(2)}$, 'conv2'$^{(2)}$, 'fc1'$^{(2)}$, and 'fc2'$^{(2)}$ for the second task. We can see that there are not any parameter sharing between these two tasks.

For one of the possible predefined deep MTL, the parameters could be 'conv1', 'conv2', 'fc1'$^{(1)}$, and 'fc2'$^{(1)}$ for the first task; 'conv1', 'conv2', 'fc1'$^{(2)}$, and 'fc2'$^{(2)}$ for the second task, i.e., the first and second layer are fully shared in this case.

For our proposed method, the parameter setting is the same as single task learning mode, but we put three tensor norms on the stacked {'conv1'$^{(1)}$, 'conv1'$^{(2)}$} (a tensor of size $5 \times 5 \times 3 \times 32 \times 2$), the stacked {'conv2'$^{(1)}$, 'conv2'$^{(2)}$} (a tensor of size $3 \times 3 \times 32 \times 64 \times 2$), and the stacked {'fc1'$^{(1)}$, 'fc1'$^{(2)}$} (a tensor of size $256 \times 256 \times 2$) respectively.

6.2.1 Tensor Tensor Norm

We choose to use the trace norm, which is defined as the sum of matrix's singular values.

$$||W||_* = \sum_{i=1} \sigma_i(W) \tag{6.1}$$

The trace norm is named by the fact that when W is a positive semidefinite matrix, it is the trace of W. It is sometimes referred to as nuclear norm. Trace norm is the tightest convex relation of matrix rank [Recht et al., 2010].

The extension of trace norm from matrix to tensor is not unique, just like the rank of tensor has different definitions. How to define the rank of tensor largely depends on how the tensor is factorised, e.g., Tucker decomposition [Tucker, 1966] and Tensor-Train decomposition Oseledets [2011].

We propose three tensor trace norm designs here, which are corresponding to three variants of the proposed method.

For an N-way tensor $\mathcal{W}$ of size $D_1 \times D_2 \times \cdots \times D_N$. We define

[1]For the case that each task has a different number of outputs, the parameters of topmost layers from neural networks should be of different size, thus they are opted out for sharing.

Tensor Trace Norm Tucker

$$||\mathcal{W}||_* = \sum_{i=1}^{N} \gamma_i ||\mathcal{W}_{(i)}||_* \tag{6.2}$$

Here $\mathcal{W}_{(i)}$ is the mode-i tensor flattening/unfolding, which is obtained by,

$$\mathcal{W}_{(i)} := \text{reshape}(\text{permute}(\mathcal{W}, [i, 1, \ldots, i-1, i+1 \ldots, N]), [D_i, \prod_{j \neg i} D_j]) \tag{6.3}$$

Tensor Trace Norm TT

$$||\mathcal{W}||_* = \sum_{i=1}^{N-1} \gamma_i ||\mathcal{W}_{[i]}||_* \tag{6.4}$$

Here $\mathcal{W}_{[i]}$ is yet another way to unfold the tensor, which is obtained by,

$$\mathcal{W}_{[i]} = \text{reshape}(\mathcal{W}, [D_1 \times D_2 \ldots D_i, D_{i+1} \times D_{i+2} \ldots D_N]) \tag{6.5}$$

Tensor Trace Norm Last Axis Flattening

$$||\mathcal{W}||_* = \gamma ||\mathcal{W}_{(N)}||_* \tag{6.6}$$

This is the simplest definition. Given that in our framework, the last axis of tensor indexes the tasks, i.e., $D_N = T$, it is the most straightforward way to adapt the technique in matrix-based MTL – reshape the $D_1 \times D_2 \times \cdots \times T$ tensor to $D_1 D_2 \cdots \times T$ matrix.

6.2.2 Optimisation

For the regularisation terms defined in Eq. 6.2 and Eq. 6.4, we see that tensor trace norm is formulated as the sum of matrix trace norms, and Eq. 6.6 is also about matrix trace norm. Thus the optimisation is ultimately targeting on matrix trace norm. Using gradient-based method for optimisation involved with matrix trace norm is *not* a common choice, as there are better solutions based on semi-definite programming or proximal gradients since the trace norm is essentially non-differentiable. However, deep neural network is usually trained by gradient descent, and it is relatively harder to modify the training process of neural network. Therefore we derive a (sub-)gradient descent method for trace norm minimisation.

We start from an equivalent definition of trace norm instead of the sum of singular values,

$$||W||_* = \text{Trace}((W^T W)^{\frac{1}{2}}) = \text{Trace}((W W^T)^{\frac{1}{2}}) \tag{6.7}$$

$(\cdot)^{\frac{1}{2}}$ is the matrix square root. Given the property of the differential of the trace function,

$$\partial \operatorname{Trace}(f(A)) = f'(A^T) : \partial A \tag{6.8}$$

The colon : denotes the double-dot (a.k.a. Frobenius) product, i.e., $A : B = \operatorname{Trace}(AB^T)$. In this case, $A = W^T W$, $f(\cdot) = (\cdot)^{\frac{1}{2}}$ thus $f'(\cdot) = \frac{1}{2}(\cdot)^{-\frac{1}{2}}$, so we have,

$$\begin{aligned}
\partial \operatorname{Trace}((W^T W)^{\frac{1}{2}}) &= \frac{1}{2}(W^T W)^{-\frac{1}{2}} : \partial(W^T W) && (6.9)\\
&= \frac{1}{2}(W^T W)^{-\frac{1}{2}} : ((\partial W^T)W + W^T(\partial W)) && (6.10)\\
&= W(W^T W)^{-\frac{1}{2}} : \partial W && (6.11)
\end{aligned}$$

Therefore we have $\frac{\partial ||W||_*}{\partial W} = \frac{\partial \operatorname{Trace}((W^T W)^{\frac{1}{2}})}{\partial W} = W(W^T W)^{-\frac{1}{2}}$. In the case that $W^T W$ is not invertible, we can derive that $\frac{\partial ||W||_*}{\partial W} = \frac{\partial \operatorname{Trace}((W W^T)^{\frac{1}{2}})}{\partial W} = (WW^T)^{-\frac{1}{2}} W$ similarly. To avoid the check on whether $W^T W$ is invertible, and more importantly, to avoid the explicit computation of the matrix square root, which is usually not numerically safe, we use the following procedure.

First, we assume W is an $N \times P$ matrix ($N > P$) and let the (full) SVD of W be $W = U\Sigma V^T$. Σ is an $N \times P$ matrix in the form of $\Sigma = [\Sigma_*; \mathbf{0}_{(N-P)\times P}]$. Then we have

$$W(W^T W)^{-\frac{1}{2}} = U\Sigma V^T (V\Sigma_*^2 V^T)^{-\frac{1}{2}} = U\Sigma V^T V \Sigma_*^{-1} V^T = U\Sigma\Sigma_*^{-1} V^T = U[I_P; \mathbf{0}_{(N-P)\times P}]V^T \tag{6.12}$$

This indicates that we only need to compute the truncated SVD, i.e., $W = U_* \Sigma_* V_*^T$, and $W(W^T W)^{-\frac{1}{2}} = U_* V_*^T$. For the case when $N < P$, we have the same result as,

$$(WW^T)^{-\frac{1}{2}} W = (U\Sigma_*^2 U^T)^{-\frac{1}{2}} U\Sigma V^T = U\Sigma_*^{-1} U^T U \Sigma V^T = U\Sigma_*^{-1}\Sigma V^T = U[I_N, \mathbf{0}_{(P-N)\times N}]V^T \tag{6.13}$$

Now we have an agreed formulation[2]: $\frac{\partial ||W||_*}{\partial W} = U_* V_*^T$ that we can use for gradient descent. To compute SVD exactly is expensive, a possible solution is to use a fast randomized SVD [Halko et al., 2011].

6.3 Experiment

We evaluate our alternative tensor-trace norm regularisation approach to deep multi-task learning (TNRDMTL , Chapter 6) by experimenting on the Omniglot dataset.

Implementation Our method is implemented in TensorFlow [Abadi et al., 2015], and released on Github[3].

6.3.1 Omniglot

For a quick recap, Omniglot contains handwritten letters in 50 different alphabets (e.g., Cyrillic, Korean, Tengwar), each with its own number of unique characters ($14 \sim 55$).

[2]An alternative way to derive the same equation can be found in [Watson, 1992].

[3]`https://github.com/wOOL/TNRDMTL`

In total, there are 1623 unique characters, each with 20 instances. Each task is a multi-class character recognition problem for the corresponding alphabet. The images are monochrome of size 105×105. We design a CNN with 3 convolutional and 2 FC layers. The first conv layer has 8 filters of size 5×5; the second conv layer has 12 filters of size 3×3, and the third convolutional layer has 16 filters of size 3×3. Each convolutional layer is followed by a 2×2 max-pooling. The first FC layer has 64 neurons, and the second FC layer has size corresponding to the number of unique classes in the alphabet. The activation function is $tanh$. We compare the three variants of the proposed framework – TNRDMTL-LAF (Eq. 6.6), TNRDMTL-Tucker (Eq. 6.2), and TNRDMTL-TT (Eq. 6.4) with single task learning (STL). For every layer, there are one (LAF) or more (Tucker and TT) γ that control the trade-off between the classification loss (cross-entropy) and the trace norm terms, for which we set all $\gamma = 0.01$. The experiments are repeated 10 times, and every time 10% training data and 90% testing data are randomly selected.

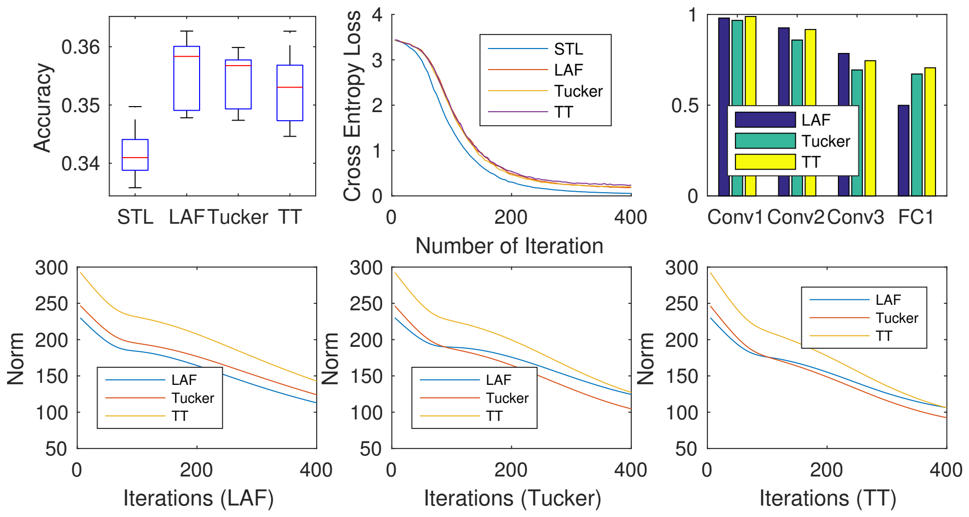


Figure 6.1: Top-left: Testing accuracy. Top-mid: Training loss. Top-right: sharing strength by layer. Bottom: Norms when optimising TNRDMTL-LAF (left), TNRDMTL-Tucker (middle), TNRDMTL-TT (right).

We plot the performance measured by accuracy, change of cross-entropy loss during training, strength of sharing across layers, and the values of norm terms with the neural networks' parameters updating. As we can see in Fig. 6.1, STL has the lowest training loss, but worst testing performance, suggesting over-fitting. Our methods alleviate the problem with multi-task regularisation. We roughly estimate the strength of parameter sharing by calculating $1 - \frac{\text{Norm of Optimised Param}}{\text{Norm of Initialised Param}}$, we can see the pattern that with bottom layers share more compared to the top ones. This reflects the common design intuition that the bottom layers are more data/task independent. Finally, it appears that the choice on LAF, Tucker, or TT may not be very sensitive as we observe that when optimising one, the loss of the other norms still reduces.

6.4 Summary

In this chapter, we propose a data-driven solution to the branching architecture design problem in deep multi-task learning. It is a flexible norm regularisation based alternative to explicit factorisation-based approaches to the same problem in Chapter 5.

Chapter 7

Conclusion and outlook

We have presented a line of work in multi-task/multi-domain learning. The key contributions of this thesis are (i) a general framework encompassing classic methods, (ii) extending MDL/MTL problem setting from atomic to parametrised domain/task, and (iii) extending MTL from shallow to deep models.

Finally, we outline some directions for future work, some of those were preliminarily studied in some of our work that is not covered in this thesis.

7.1 Parametrised DNNs

Our method in Chapter 5 is a *parametrised DNN* [Sigaud et al., 2015], in that DNN weights are dynamically generated given some side information – in the case of MTL, given the task identity. In a related example of speaker-adaptive speech recognition [Tan et al., 2016] there may be several clusters in the data (e.g., gender, acoustic conditions), and each speaker's model could be a linear combination of these latent task/clusters' models. They model each speaker i's weight matrix $W^{(i)}$ as a sum of K base models $\tilde{W}$, i.e., $W^{(i)} = \sum_{k=1}^{K} \lambda_p^{(i)} \tilde{W}^{(p)}$. The difference between speakers/tasks comes from λ and the base models are shared. An advantage of this is that, when new data come, one can choose to re-train λ parameters only, and keep $\tilde{W}$ fixed. This will significantly reduce the number of parameters to learn, and consequently the required training data. Beyond this, in Chapter 3 we show that it is possible to train another neural network to *predict* those λ values from some abstract metadata. Thus a model for an *unseen* task can be generated on-the-fly with *no* training instances given an abstract description of the task. The technique developed in Chapter 5 is, in theory, compatible with both these ideas of generating models with minimal or no effort.

7.2 Unlabelled Tasks/Domains

In the whole thesis, we assume that task/domain information, either in a form of task identity index or a semantic descriptor, is *always* available. However, in some practical problems, we do not have the task labels, but we believe that the problem potentially involves multiple partitions (tasks or domains).

A natural question is that if we can find the hidden task identities? We can think of it as constructing a multi-task learning problem meanwhile solving it. While it sounds a bit hard, we briefly propose a possible solution here.

Recall the basic formulation of Eq. 3.1 in Chapter 3, $\hat{y} = f_P(x) \cdot g_Q(z)$. Now we do not have the true z for x, so we have to infer it. Assume that we estimate $z^{(i)}$ using another neural network, i.e., $\hat{z} = h_\Theta(x)$, the equation becomes,

$$\hat{y} = f_P(x) \cdot g_Q(h_\Theta(x)) \tag{7.1}$$

Eq. 7.1 has some connections with several classic models: e.g., its simplest form is $\hat{y} = x^T W x$ (i.e., bilinear model), or we can think of it as a mixture of expert models [Jacobs et al., 1991], $f_P(x)$ produces some estimations, and $g_Q(h_\Theta(x))$ provides the weights for these estimations.

In [Yang et al., 2017], we apply this approach to a specific problem, and encouragingly, it shows some abilities to discover hidden tasks in a meaningful way, i.e., it divided data in several different groups (corresponding to tasks) that coincide with domain expert's choice.

7.3 Limitations and Future Works

This thesis focused on 'parallel' MTL where all models and data are ready before training, but how to do it in an incremental way remains an open question.

The methodologies in Chapter 5 and Chapter 6 heavily reply on the assumption that (most of) the architectures of each task's/domain's neural network is the same, however, it may be of interest to know what to do when this assumption is violated: can we share the parameters between a small neural network for a very specific problem (e.g., fine-grained classification – which species of flower is) and a large neural network for a more general purpose problem (classifying 1000 common objects). Furthermore, we discuss a lot on sharing across the same level of layers (horizontally), is it possible to share across different levels of layers (vertically) within a single neural network (or even cross different NNs) is an interesting direction.

Appendix A

Notations

We summarise the notations used throughout the thesis.

A.1 Variables

Variable type	Notation	Example
Scalar	Greek letter	α
Vector	Lower-case letter	a
Matrix	Upper-case letter	A
Tensor (3-way or higher)	Calligraphic upper-case letter	$\mathcal{A}$
Real/Complex Space	Blackboard bold letter	$\mathbb{R}^n$, $\mathbb{C}^n$

Table A.1: Variable Notations

All vectors are column vectors, unless otherwise specified, e.g., $a = [1, 2, 3]^T = [1; 2; 3]$.

A.2 Indexing

A lower-case letter n is usually used for indexing an integer sequence starting with 1 (inclusive) and ending with its upper-case letter N (inclusive), e.g.,

$$\sum_{n=1}^{N} x_n = x_1 + x_2 + \cdots + x_N$$

We use $\neg i$ indicating a certain integer i is excluded from the sequence, e.g.,

$$\sum_{n \neg i} x_n = \sum_{n=1:N, n \neq i} x_n = x_1 + x_2 + \cdots + x_{i-1} + x_{i+1} + \cdots + x_N$$

By default, when we use $n = 1 : N$, two bounds – 1 and N – are included.

A.3 Slicing and Stacking

For a matrix A, $A_{i,\cdot}$ is a vector containing the elements from the i-th row of A, similarly $A_{\cdot,i}$ is the i-th column of A. By default, we assume all vectors are column vectors.

When it comes to tensors, e.g., a 3-way tensor $\mathcal{A}$, $\mathcal{A}_{i,\cdot,\cdot}$ is a matrix sliced from the tensor along the first axis at the i-th index.

We use $[A; B]$ for stacking two variables A and B vertically (row-wise), and $[A, B]$ for stacking two variables A and B horizontally (column-wise).

A.4 Operators

A.4.1 Product operators

$\cdot$ stands for dot product, but it is usually omitted, e.g., $u \cdot v = u^T v$

$\circ$ is element-wise product (entry-wise product, or Hadamard product), e.g., $u \circ v = [u_1 v_1, u_2 v_2, \cdots]^T$.

Kronecker product (and its special case outer product) is denoted by $\otimes$, e.g., for an N-dimensional vector u and an M-dimensional vector v, we have

$$u \otimes v = [u_1 v_1, u_1 v_2, \cdots, u_1 v_M, u_2 v_1, \cdots, u_N v_M]^T$$

Tensor dot product is denoted by $\bullet_{(i,j)}$, e.g., $\mathcal{U} \bullet_{(i,j)} \mathcal{V}$ computes the tensor dot product of two tensors $\mathcal{U}$ and $\mathcal{V}$ along specified axes: the i-th axis of $\mathcal{U}$ and the j-th axis of $\mathcal{V}$. We detail how tensor dot product is implemented in Appx. A.4.4.

A.4.2 Norm functions

The norm function is denoted by $||\cdot||$, some frequently used norms in this thesis are as follows,

- vector ℓ_2 norm, $||a||_2 = \sqrt{\sum_{n=1}^{N} |a_n|^2}$
- vector ℓ_1 norm, $||a||_1 = \sum_{n=1}^{N} |a_n|$
- matrix Frobenius norm, $||A||_F = \sqrt{\sum_{m=1}^{M} \sum_{n=1}^{N} |A_{m,n}|^2}$
- matrix trace norm, $||A||_* = \sigma_1 + \sigma_2 + \cdots + \sigma_{\min(M,N)}$, i.e., the sum of singular values of A.

A.4.3 Matrix operators

The inverse of a square matrix A (assuming it is invertible) is denoted A^{-1}.

The determinant of a square matrix A is denoted by $\det(A)$.

The rank of a matrix A is denoted by $\operatorname{rank}(A)$, an $M \times N$ sized matrix A is called full rank matrix if its rank equals the largest possible rank, e.g., $\operatorname{rank}(A) = \min(M, N)$.

The following three statements are equivalent:

- A is an invertible matrix.
- The determinant of A is not zero, i.e., $\det(A) \neq 0$.
- A is a full rank matrix.

The trace of a square matrix A is the sum of A's diagonal elements, and it is denoted as $\mathrm{Tr}(A) = A_{11} + A_{22} + \cdots$.

A.4.4 Tensor operators

An N-way tensor $\mathcal{A}$ with shape $D_1 \times D_2 \times \cdots D_N$ is an N-dimensional array containing $\prod_{n=1}^{N} D_n$ elements. Scalars, vectors, and matrices can be seen as 0-, 1-, and 2-way tensors respectively, although the term tensor is usually used for 3-way or higher. A mode-i fibre of $\mathcal{A}$ is a D_i-dimensional vector obtained by fixing all but the i-th axis. The mode-i flattening (unfolding) $A_{(i)}$ of $\mathcal{A}$ is the matrix of size $D_i \times \prod_{n \neg i} D_n$ constructed by concatenating all of the $\prod_{n \neg i} D_n$ mode-i fibres along columns, i.e.,

$$A_{(i)} := \mathrm{reshape}(\mathrm{permute}(\mathcal{A}, [i, 1, 2, \ldots, i-1, i+1, \ldots, N]), [D_i, \prod_{n \neg i} D_n]) \tag{A.1}$$

reshape function gives a new shape to an array without changing its data and *permute* function permutes the dimensions of a multi-dimensional array. The dot product of two tensors is a natural extension of matrix dot product, e.g., if we have a tensor $\mathcal{A}$ of size $M_1 \times M_2 \times \cdots \times P$ and a tensor $\mathcal{B}$ of size $P \times N_1 \times N_2 \times \cdots$, the tensor dot product $\mathcal{A} \bullet \mathcal{B}$ will be a tensor of size $M_1 \times M_2 \times \cdots \times N_1 \times N_2 \times \cdots$ by matrix dot product $A_{(-1)}^T B_{(1)}$ and reshaping[1], i.e.,

$$\mathcal{A} \bullet \mathcal{B} := \mathrm{reshape}(A_{(-1)}^T B_{(1)}, [M_1, M_2, \ldots, N_1, N_2, \ldots]) \tag{A.2}$$

More generally, tensor dot product can be performed along specified axes, $\mathcal{A} \bullet_{(i,j)} \mathcal{B} = A_{(i)}^T B_{(j)}$ and reshaping. Here the subscripts indicate the axes of $\mathcal{A}$ and $\mathcal{B}$ at which dot product is performed. E.g., when $\mathcal{A}$ is of size $M_1 \times P \times M_3 \times \cdots \times M_I$ and $\mathcal{B}$ is of size $N_1 \times N_2 \times P \times N_4 \times \cdots \times N_J$, then we have

$$\mathcal{A} \bullet_{(2,3)} \mathcal{B} := \mathrm{reshape}(A_{(2)}^T B_{(3)}, [M_1, M_3, \ldots, M_I, N_1, N_2, N_4, \ldots, N_J]) \tag{A.3}$$

resulting a tensor of size $M_1 \times M_3 \times \cdots \times M_I \times N_1 \times N_2 \times N_4 \times \cdots \times N_J$.

Note that in Eq. A.1, the specification of permutation operator is usually less important. Two popular choices are $\mathrm{permute}(\mathcal{A}, [i, i+1, i+2, \ldots, N, 1, 2, \ldots, i-1])$ and $\mathrm{permute}(\mathcal{A}, [i, 1, 2, \ldots, i-1, i+1, \ldots, N])$, and we use the latter one in this thesis. It is very important to keep the specification consistent across related calculations, because it will, consequently, affect the calculation of tensor dot product.

[1]We slightly abuse '-1' referring to the last axis of the tensor.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 2008.

S. Ö. Arik, M. Chrzanowski, A. Coates, G. F. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Y. Ng, J. Raiman, S. Sengupta, and M. Shoeybi. Deep voice: Real-time neural text-to-speech. In *International Conference on Machine Learning (ICML)*, 2017.

B. Bakker and T. Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research (JMLR)*, 2003.

D. Barber. *Bayesian Reasoning and Machine Learning.* Cambridge University Press, 2012.

H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

O. Beijbom. Domain adaptations for computer vision applications. Technical report, UCSD, 2012.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

J. Blitzer, D. P. Foster, and S. M. Kakade. Zero-shot domain adaptation: A multi-view approach. Technical report, University of California, Berkeley, 2009.

E. V. Bonilla, K. M. Chai, and C. Williams. Multi-task gaussian process prediction. In *Neural Information Processing Systems (NIPS)*, 2007.

H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 1988.

J. Bromley, I. Guyon, Y. Lecun, E. Sckinger, and R. Shah. Signature verification using a “siamese” time delay neural network. In *Neural Information Processing Systems (NIPS)*, 1994.

R. Caruana. Multitask learning. *Machine Learning*, 1997.

S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition (CVPR)*, 2005.

H. Daumé III. Frustratingly easy domain adaptation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2007.

J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *ECCV*, pages 48–64, 2014.

Z. Ding, S. Ming, and Y. Fu. Latent low-rank transfer subspace learning for missing modality recognition. In *AAAI*, pages 1192–1198, 2014.

J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, 2015.

M. Dredze, A. Kulesza, and K. Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 2010.

A. Droniou and O. Sigaud. Gated autoencoders with tied input weights. In *International Conference on Machine Learning (ICML)*, 2013.

L. Duan, D. Xu, and S.-F. Chang. Exploiting web images for event recognition in consumer videos: A multiple source domain adaptation approach. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

E. Eidinger, R. Enbar, and T. Hassner. Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, 2014.

D. P. W. Ellis. PLP and RASTA (and MFCC, and inversion) in Matlab, 2005. URL `http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/`. online web resource.

T. Evgeniou and M. Pontil. Regularized multi–task learning. In *Knowledge Discovery and Data Mining (KDD)*, 2004.

T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research (JMLR)*, 6:615–637, Dec. 2005.

L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2006.

A. Frome, G. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov. Devise: A deep visual-semantic embedding model. In *NIPS*, 2013.

Y. Fu, T. Hospedales, T. Xiang, Z. Fu, and S. Gong. Transductive multi-view embedding for zero-shot recognition and annotation. In *European Conference on Computer Vision (ECCV)*, 2014.

Y. Ganin and V. S. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, 2015.

B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *International Conference on Machine Learning (ICML)*, 2013.

G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. URL `http://authors.library.caltech.edu/7694`.

T. L. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research (JMLR)*, 2011.

N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York Inc., 2001.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, 2015.

F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 1927.

J. Hoffman, T. Darrell, and K. Saenko. Continuous manifold based adaptation for evolving visual domains. In *Computer Vision and Pattern Recognition (CVPR)*, 2014.

J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013.

L. Jacob, J.-p. Vert, and F. R. Bach. Clustered multi-task learning: A convex formulation. In *Neural Information Processing Systems (NIPS)*, 2009.

R. Jacobs, M. I. Jordan, N. S. J., and G. E. Hinton. Adaptive mixtures of local experts. In *Neural Computation*, volume 3, pages 79–87, 1991.

S. Ji and J. Ye. An accelerated gradient method for trace norm minimization. In *International Conference on Machine Learning (ICML)*, 2009.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

M. Joshi, M. Dredze, W. W. Cohen, and C. P. Rosé. Multi-domain learning: When do domains matter? In *Empirical Methods on Natural Language Processing (EMNLP)*, 2012.

Z. Kang, K. Grauman, and F. Sha. Learning with whom to share in multi-task feature learning. In *International Conference on Machine Learning (ICML)*, 2011.

D. Kressner, M. Steinlechner, and B. Vandereycken. Low-rank tensor completion by riemannian optimization. *BIT Numerical Mathematics*, 54(2):447–468, 2014. ISSN 1572-9125. doi: 10.1007/s10543-013-0455-z. URL `http://dx.doi.org/10.1007/s10543-013-0455-z`.

A. Kumar and H. Daumé III. Learning task grouping and overlap in multi-task learning. In *International Conference on Machine Learning (ICML)*, 2012.

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

H. Larochelle, D. Erhan, and Y. Bengio. Zero-data learning of new tasks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 2000.

V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

G. Lee, E. Yang, and S. Hwang. Asymmetric multi-task learning based on task relatedness and loss. In *International Conference on Machine Learning (ICML)*, 2016.

G. Levi and T. Hassncer. Age and gender classification using convolutional neural networks. In *Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015.

W. Li, L. Duan, D. Xu, and I. W. Tsang. Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(6):1134–1148, June 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.167. URL `http://dx.doi.org/10.1109/TPAMI.2013.167`.

X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y.-Y. Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. *NAACL*, 2015.

C. Lu and X. Tang. Surpassing human-level face verification performance on LFW with gaussianface. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

M. Mauch and S. Ewert. The audio degradation toolbox and its application to robustness evaluation. In *ISMIR*, 2013.

R. Memisevic and G. E. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition (CVPR)*, 2007.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL `http://arxiv.org/abs/1411.1784`.

P. Mortimore, P. Sammons, L. Stoll, D. Lewis, and R. Ecob. *School Matters: the Junior Years.* Wells: Open Books, 1988.

K. Muandet, D. Balduzzi, and B. Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning (ICML)*, 2013.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. In *Neural Information Processing Systems (NIPS)*, 2015.

G. Obozinski, B. Taskar, and M. I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, Apr 2010.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 2011.

M. Palatucci, D. Pomerleau, G. Hinton, and T. Mitchell. Zero-shot learning with semantic output codes. In *Neural Information Processing Systems (NIPS)*, 2009.

A. Passos, P. Rai, J. Wainer, and H. Daumé III. Flexible modeling of latent task structures in multitask learning. In *International Conference on Machine Learning (ICML)*, 2012.

A. Pentina and C. H. Lampert. Multi-task learning with labeled and unlabeled tasks. In *International Conference on Machine Learning (ICML)*, 2017.

Q. Qiu, V. M. Patel, P. Turaga, and R. Chellappa. Domain adaptive dictionary learning. In *European Conference on Computer Vision (ECCV)*, 2012.

B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Rev.*, 2010.

B. Romera-Paredes, H. Aung, N. Bianchi-berthouze, and M. Pontil. Multilinear multitask learning. In *International Conference on Machine Learning (ICML)*, 2013.

M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *In NIPS Workshop, Inductive Transfer: 10 Years Later*, 2005.

S. Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *ArXiv e-prints*, June 2017.

K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, 2010.

R. Salakhutdinov, A. Torralba, and J. B. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, pages 1481–1488. IEEE, 2011.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

O. Sigaud, C. Masson, D. Filliat, and F. Stulp. Gated networks: an inventory. *CoRR*, abs/1512.03201, 2015.

R. Socher, M. Ganjoo, C. D. Manning, and A. Y. Ng. Zero-shot learning through cross-modal transfer. In *Neural Information Processing Systems (NIPS)*, 2013.

S. Spieckermann, S. Udluft, and T. Runkler. Data-effiicient temporal regression with multitask recurrent neural networks. In *NIPS Workshop on Transfer and Multi-Task Learning*, 2014.

B. Sun and K. Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *BMVC*, 2014.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems (NIPS)*, 1999.

T. Tan, Y. Qian, and K. Yu. Cluster adaptive training for deep neural network based acoustic model. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 24(3):459–468, 2016.

A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR)*, 2011.

L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.

G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002.

B. Vargas-Govea, G. González-Serna, and R. Ponce-Medellın. Effects of relevant contextual features in the performance of a restaurant recommender system. *ACM RecSys*, 11, 2011.

G. Watson. Characterization of the subdifferential of some matrix norms. *Linear Algebra and its Applications*, 170:33 – 45, 1992. ISSN 0024-3795.

K. Wimalawarne, M. Sugiyama, and R. Tomioka. Multitask learning meets tensor factorization: task imputation via convex optimization. In *Neural Information Processing Systems (NIPS)*, 2014.

Y. Xue, X. Liao, L. Carin, and B. Krishnapuram. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research (JMLR)*, 2007.

Y. Yang and T. M. Hospedales. A unified perspective on multi-domain and multi-task learning. In *International Conference on Learning Representations (ICLR)*, 2015.

Y. Yang and T. M. Hospedales. Multivariate regression on the grassmannian for predicting novel domains. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Y. Yang, Y. Zheng, and T. M. Hospedales. Gated neural networks for option pricing: Rationality by design. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Q. Yu, Y. Yang, Y. Song, T. Xiang, and T. M. Hospedales. Sketch-a-net that beats humans. In *British Machine Vision Conference 2015 (BMVC)*, 2015.

Y. Zhang and Q. Yang. A Survey on Multi-Task Learning. *ArXiv e-prints*, July 2017.

Y. Zhang and Q. Yang. Learning sparse task relations in multi-task learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision (ECCV)*, 2014.

S. Zheng, J. Zhang, K. Huang, R. He, and T. Tan. Robust view transformation model for gait recognition. In *International Conference on Image Processing (ICIP)*, 2011.