Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is well known that single the price movement of an individual stock not only depends on historical records but also highly correlated to other stocks [Lo and MacKinlay, 1990, Mech, 1993] and may change in a non-synchronous manner [Lo and MacKinlay, 1990, Brennan et al., 1993]. This correlated yet asynchronous price movement is sometimes referred to as the lead-lag relationship [Hou, 2007] between a group of stocks and is thought to arise from the different speed of information diffusion[Lo and MacKinlay, 1990, Badrinath et al., 1995, McQueen et al., 1996]. When new information hits the market, some stocks react faster than others and identification of these leading stocks and their lead-lag relationships to other lagging stocks provides strong predictive evidence to the latter's price movement.

Extracting informational price changes from market price data has been a long existing challenge in stock trading industry. Researchers have developed hundreds of technical indicators [Kirkpatrick II and Dahlquist, 2010] to recognize trend or predict volatility in future stock price movement. We are inspired by the significant progress in computer vision area where researchers developed neural networks which outperform hand-crafted features such as SIFT, HOG, and SURF. In this paper we try to investigate a multi-task hierarchical RNN neural networks in order to replace those hand-crafted technical indicators.

Part I

Directed Probabilistic Graphical Model

Chapter 2

The Skill-Action (SA) Architecture: Multi-timescale Sequence Learning under Hierarchical Reinforcement Learning Framework

Reinforcement Learning (RL) is a paradigm for imitating humans trial-and-error learning process: RL trains an agent to maximise rewards by taking actions in and receiving feedback from an environment. RL has achieved human-level performance in playing video and board games [Mnih et al., 2015, Silver et al., 2016]. However, while humans can abstract the hierarchical complexity of actions through interactions with the environment and make decisions at both macro and micro time scales, conventional RL agents have limited abilities to solve complex tasks [Daniel et al., 2016]: they only learn the most primitive actions and make decisions at the smallest time scale. Hierarchical Reinforcement Learning (HRL) attempts to resolve this gap between humans and RL by decomposing complex tasks into a hierarchy of abstracted actions at multiple time scales.

An HRL agent typically learns abstractions of actions on two levels: skills and primary actions. Skills are higher-level abstracted actions. Their executions are temporally extended to a variable amount of time. Primary actions are lower-level actions defined by the environment. They are executed at every time step. For example, for a humanoid robot, walking and jumping are two abstract skills, while movements of each joint are primary actions. One of the most promising HRL frameworks is the option framework [Sutton et al., 1999].

The option framework has achieved great success in representing actions at different time scales [Bacon et al., 2017], speeding and scaling up learning [Bacon, 2018], improving exploration [Harb et al., 2018], and facilitating transfer learning [Zhang and Whiteson, 2019].

In the option framework, an option is a primary action level sub-policy consisting of an action policy, a termination probability, and an initiation set. A master policy [Zhang and Whiteson, 2019] (aka policy-over-options [Sutton et al., 1999]) is used to compose those options and thus is a skill-level policy. The option framework is formulated as a Semi-Markov Decision Problem (SMDP) [Puterman, 1994]: an option sampled from a master policy is executed through a variable amount of time (until its termination function determines to stop). As highlighted in the literature, the SMDP formulation has the following limitations:

1. Sample inefficiency [Zhang and Whiteson, 2019]: a) For policy gradient based algorithms, the master policy cannot be updated until stop. As a result, one update consumes various time steps of samples. b) At each time step, only one (the executed) option's policies can be updated.

2. Large variance [Zhang and Whiteson, 2019, Haarnoja et al., 2018]: SMDP algorithms are notoriously sensitive to hyperparameters . Due to the SMDP formulation, more stable Markov Decision Process (MDP) policy gradient algorithms cannot be used.

3. Expensive to scale up [Riemer et al., 2018]: for M options, there are $2M$ action and termination policies. Each policy is a neural network that could have millions of parameters to train.

To address these problems, we propose a simple yet effective MDP implementation of the option framework, the Skill-Action (SA) architecture. The idea behind SA originates from our new discovery that the SMDP option framework has an MDP equivalence, which is achieved by adding extra dependencies into the master policy. However, those extra dependencies still prevent the master policy from being updated at every time step. Based on this equivalence, a "skill policy" which marginalizes those dependencies away is derived and hence can be updated at each time step.

In SA, knowledge of a skill is explicitly represented as a skill context vector (similar to an embedding vector [Vaswani et al., 2017] in Natural Language Processing (NLP) or capsule [Sabour et al., 2017] in Computer Vision (CV)): each dimension encodes a particular property of the skill[1]. The skill policy is similar to a compatibility function: it is used to replace the master policy and termination function while improving their functionalities by employing the attention mechanism [Vaswani et al., 2017]. At each time step, the skill policy measures the compatibility (suitability) of all skills with the current state and the executed skill from the last step. If the previous skill still fits the current situation, then the skill policy tends to continue with it; otherwise, a new skill with better compatibility will be sampled. Unlike the option framework, which requires M action policies for M skills, SA's action policy only needs one decoder to decode any skill context vector into primary actions. With this formulation, the entire framework is MDP-based while the skill can still be temporally extended, and its scalability, as well as stability, are significantly improved. All of these design choices have precursors in the existing literature (HRL [Sutton et al., 1999, Bacon, 2018, Zhang and Whiteson, 2019]; CV [Kosiorek et al., 2019]; NLP [Vaswani et al., 2017]). Our contribution is establishing them in reinforcement learning settings.

Compared to the SMDP option framework, SA has following advantages:

1. SA is more sample efficient, because a) SA is MDP formulated, thus sample at each time step can be used to update the skill policy; and b) Only one action policy decoder is needed. It learns to decode each dimension of the skill context vector at each time step whichever skill is activated;

2. SA has smaller variance, because a) The skill value upon arrival function (Eq. (2.28)) is theoretically and empirically proven to have smaller variance than the conventional value function; and b) SA only needs to train two (skill and action) policy networks; and c) SA can employ more stable MDP based policy gradient algorithms (e.g. PPO [Schulman et al., 2017]);

[1]For example, the first dimension may encode the orientation of a primary action. A jump skill context vector may have a large first dimension value which instructs the robot to emit primary actions vertically. A walk skill may have a small value and emit actions horizontally.

3. SA has better scalability, because a) Regardless of the number of skills, only two policies need to be trained; and b) Adding one more skill is as cheap as adding a context vector;

4. SA is more effective, because a) On infinite-horizon environments, SA significantly outperforms the other models; and b) On transfer learning environments, SA ranks the first in 5 out of 6 environments and shows its advantages in knowledge reuse tasks;

5. SA has better interpretability. Unlike the option framework encodes abstract knowledge implicitly in action policies, knowledge of a skill is explicitly encoded in each dimension of the skill context vector.

2.1 Related Works

To discover options automatically, Sutton et al. [1999] proposed Intra-option Q-learning to update the master Q value function at every time step. However, all policies under this formulation are approximated implicitly using the Q-learning method. Levy and Shimkin [2011] proposed to unify the Semi-Markov process into an augmented Markov process and explicitly learn an "overall policy" by applying MDP-based policy gradient algorithms. However, their method for updating the master policy is still SMDP-style thus sample inefficient. Bacon et al. [2017] proposed a policy gradient based Option Critic (OC) framework for explicitly learning intra-option policies and termination functions in an intra-option manner. However, for the master policy's policy gradients learning, OC still remains SMDP-style. Klissarov et al. [2017] attempted to combine OC with PPO in an intra-option learning manner (PPOC). However, as we show in Section 2.2.2, due to the SMDP formulation, gradients they use for updating master policy are inconsistent. Zhang and Whiteson [2019] reformulated the option framework into two augmented MDPs. Under this formulation all policies can be modeled explicitly and learned in MDP-style. However, their model is still expensive to scale up. On single task environments, DAC has no significant advantages over other baselines.

We must appreciate that Bacon ([Bacon, 2018]; Chapter 3.6) conceptually

discussed a vectorized option representation and directly approximated the marginalized master policy. However, no concrete formulations and policy gradients theorems were developed in their work. Daniel et al. [2016] proposed an MDP-formulated PGM similar to ours in Section 2.2. However, they did not prove the equivalence between the MDP-formulation and SMDP by employing conditional independencies. Furthermore, their learning algorithm is EM based while ours is policy gradient based. Our work is motivated by capsule networks Kosiorek et al. [2019] (more details in Section 3.4) and is developed independently from above literature.

With respect to optimization, Zhang and Whiteson [2019] pointed out that a large margin of performance boost of DAC comes from Proximal Policy Optimization [Schulman et al., 2017] (PPO). Since SA is MDP-based, it can be optimized directly with the PPO. Recent works show that the option framework trained under off-policy [Haarnoja et al., 2018] algorithms outperforms on-policy methods. For instance, HO2 [Wulfmeier et al., 2020] employs a trust-region constrained off-policy algorithm and shows that it exhibits significant advantages over on-policy methods on both sample efficiency and performance. In this paper, we propose SA as a general HRL framework which can be trained by both on-policy and off-policy algorithms. Our main contribution focuses on deriving MDPs of SA and its policy gradient theorems. Designing off-policy algorithms for SA remains open for future work.

Existing RL literature [Hausman et al., 2018, Li et al., 2017, Tirumala et al., 2019] also uses latent variables to learn skill embeddings. Typically, PEARL [Rakelly et al., 2019] learns a latent context vector for each task under the meta-reinforcement learning framework to improve the agent's sample and transfer learning efficiency. However, embeddings learned by RL frameworks only encode action level abstraction while SA learns abstractions at both action and temporal levels. It is also worth to mention that, the novel formulation of SA establishes a strong connection to causal reinforcement learning. When the number of skills equals one ($M = 1$), SA falls back to Generalized Hidden Parameter MDPs (GHP-MDPs) [Kolobov et al., 2012, Perez et al., 2020]. Causality properties of SA is a direction worth to be explored in the future work.

2.2 MDP Equivalence to the SMDP Option Framework

In this section, we prove that the the conventional Semi-Markov Decision Problem (SMDP) option framework which employs Markovian options actually has an MDP equivalence. We first follow Bishop [2006a]'s method and formulate the dynamics of the option framework as an Hidden Markov Model (HMM) [Bishop, 2006a] in section 2.2.2. With Probability Graphical Model (PGM) [Bishop, 2006a] and its conditional independence relationships (Chapter 8.2.1 [Bishop, 2006a]) in hand, we then move on to prove that MDP formulation has identical value functions (section 2.2.3), bellman equations as well as intra-option policy and termination policy gradients to SMDP formulation (section 2.2.4). To the best of our knowledge, this is the first work discovering the option framework's MDP equivalence and deriving the option framework from a PGM view.

2.2.1 Background: The Option Framework

Sutton et al. [1999] proposed the option framework to demonstrate the temporal abstraction problem. A scalar $\mathrm{o} \in \mathbb{Z}$ denotes the index of an option where $\mathbb{O} \subseteq \{1, 2, \ldots, M\}$ and M is the number of options. An Markovian option is a triple $(\mathbb{I}_o, P_o(\mathbf{a}|\mathbf{s}), P_o(\mathrm{b}|\mathbf{s}))$ in which $\mathbb{I}_o \subseteq \mathbb{S}$ is an initiation set where the option o can be initiated. $P_o(\mathbf{a}|\mathbf{s}) : \mathbb{S} \to \mathbb{A}$ is the intra-option policy which maps environment states $\mathbf{s} \in \mathbb{S}$ to an action vector $\mathbf{a} \in \mathbb{A}$. $P_o(\mathrm{b}|\mathbf{s}) : \mathbb{S} \to \mathbb{Z}_2$ is a termination function where b is a binary random variable. It is used to determine whether to terminate ($\mathrm{b} = 1$) the policy $P_o(\mathbf{a}|\mathbf{s})$ or not ($\mathrm{b} = 0$). Conventionally, $\beta_o = P_o(\mathrm{b} = 1|\mathbf{s})$. Since an option's execution may persist over a variable period of time, a set of options' execution together with its value functions constitutes a Semi-Markov Decision Problem (SMDP) [Puterman, 1994]. When an old option is terminated, a new option will be sampled from the master policy (policy-over-options) $o \sim P(o_{t+1}|\mathbf{s}_{t+1}) : \mathbb{S} \to \mathbb{O}$. Due to the SMDP formulation, an option can only be improved when the option terminates. We refer this as the SMDP-style learning which is sample inefficient and prevents applying SOTA MDP based algorithms such as the Proximal Policy Optimization (PPO) algorithm [Schulman et al., 2017].

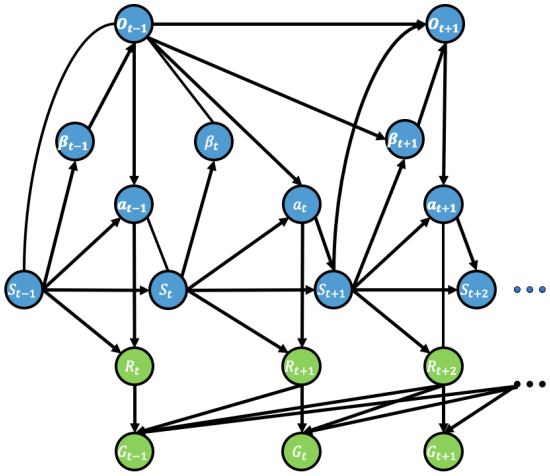


Figure 2.1: An Illustration of the SMDP Option Framework. An option o_{t-1} is selected by master policy $P(o_{t-1}|\mathbf{s}_{t-1})$ at time step $t-1$. At time step t, termination function $\beta_{o_{t-1}}(\mathbf{s}_t)$ determines to continue option o_{t-1}. So that there is no random variable o_t at time step t compared to there are random variables $\mathbf{o}$ at every time step in MDP formulation (figure 2.2).

2.2.2 HMM dynamics for the Option Framework

We follow Bishop [2006a]'s formulation of mixture distribution and Probabilistic Graphical Models (PGMs). By introducing option variables as latent variables and adding extra dependencies between them, we show that the conventional SMDP version of the option framework [Bacon et al., 2017, Sutton and Barto, 2018, Sutton et al., 1999, Harb et al., 2018, Zhang and Whiteson, 2019] has an MDP equivalence. Following Bishop [2006a]'s notation, we use

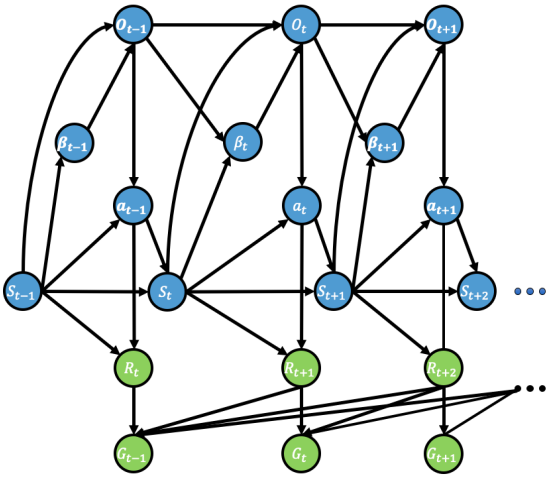


Figure 2.2: PGM of the MDP Option Framework

bolded letter $\mathbf{s} \in \mathbb{S}$ to denote a random variable and normal letter s to denote its realization. Without special clarification, a random vector can have either a vector of continuous or discrete entries. Vector $\mathbf{o} \in \mathbb{O}$ is an M-dimensional one-hot vector and each entry $\mathrm{o} \in \{0, 1\}$ is a binary random variable. $P(\mathbf{o}_t|\mathbf{s}_t)$ denotes the probability distribution over one-hot vector $\mathbf{o}$ at time step t conditioned on state $\mathbf{s}_t$. $P(\mathbf{o}_t = \mathrm{o}_t|\mathbf{s}_t)$ denotes a probability entry (a scalar value) of the random variable $\mathbf{o}_t$ with a realization at time step t where $\mathrm{o}_t = 1$ and $\mathrm{o} \in \mathbf{o}_t/\mathrm{o}_t = 0$.

In figure 2.2, $\mathbf{s} \in \mathbb{S}$, $\mathbf{o} \in \mathbb{O}^M$, $\mathbf{b} \in \mathbb{B}^M$ and $\mathbf{a} \in \mathbb{A}$, denote the state, option, termination and action random variable respectively. $\mathbf{o}$ is an M-dimensional one-hot vector and $\mathbf{b}$ is an M-dimensional binary vector where each entry $\mathrm{b} \in \{0, 1\}$. M is the number of options. R_{t+1} is the actual reward received from the environment after executing action $\mathbf{a}_t$ in state $\mathbf{s}_t$. $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots$ is the discounted expected return where $\gamma \in \mathbb{R}$ is a discount factor.

The termination policy distribution $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) : \mathbb{S} \times \mathbb{O} \to \mathbb{B}$ can be formulated as a mixture distribution[2] conditioned on option vector (the one-hot vector) $\mathbf{o}_{t-1}$ and state $\mathbf{s}_t$.

$$P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) = \prod_{\mathrm{i} \in \mathbf{o}_{t-1}} P_{\mathrm{i}}(\mathrm{b}_t|\mathbf{s}_t)^{\mathrm{i}}. \tag{2.1}$$

Because each option has its own **termination policy** $P_o(\mathbf{b}|\mathbf{s})$, with a slightly abuse of notation, in equation (2.1) we use $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the termination policy activated at time step t by previous chosen option $\mathbf{o}_{t-1}$. To keep notation uncluttered, we use $\beta_t = P(\mathbf{b}_t = 1|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the probability of option $\mathbf{o}_{t-1}$ terminates at time step t and $(1 - \beta_t) = P(\mathbf{b}_t = 0|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the probability of continuation.

Conventionally, master policy [Zhang and Whiteson, 2019] (also called "policy-over-options" [Sutton et al., 1999, Bacon et al., 2017])) is defined as:

$$P(\mathbf{o}_t|\mathbf{s}_t). \tag{2.2}$$

Similarly, we propose a novel **mixture master policy** as a mixture distribu-

[2]Different from conventional formulation which only depends on state $\mathbf{s}_t$, our termination function has an extra dependence on $\mathbf{o}_{t-1}$

tion[3]:

$$P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}) = P(\mathbf{o}_t|\mathbf{s}_t)^{\mathrm{b}_t} P(\mathbf{o}_t|\mathbf{o}_{t-1})^{1-\mathrm{b}_t}, \tag{2.3}$$

where $P(\mathbf{o}_t|\mathbf{o}_{t-1})$ is a degenerated probability distribution [Puterman, 1994]

$$P(\mathbf{o}_t|\mathbf{o}_{t-1}) = \begin{cases} 1 & \text{if } \mathbf{o}_t = \mathbf{o}_{t-1}, \\ 0 & \text{if } \mathbf{o}_t \neq \mathbf{o}_{t-1}. \end{cases} \tag{2.4}$$

As shown in equation (2.3), the master policy only exists when $\mathrm{b}_t = 1$ the option terminates. Therefore, PPOC [Klissarov et al., 2017] uses inaccurate gradients for updating the master policy during an option's execution.

According to the conditional dependency relationships in PGM (figure 2.2), the joint probability distribution of $\mathbf{o}_t$ and $\mathbf{b}_t$ can be written as:

$$P(\mathbf{o}_t, \mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) = P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}), \tag{2.5}$$

and the marginal probability distribution can be written as:

$$\begin{aligned} P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) &= \sum_{\mathbf{b}_t} P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}) \\ &= P(\mathbf{b}_t = 0|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{o}_{t-1}) + P(\mathbf{b}_t = 1|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t) \\ &= (1 - \beta_t) P(\mathbf{o}_t|\mathbf{o}_{t-1}) + \beta_t P(\mathbf{o}_t|\mathbf{s}_t) \\ &= (1 - \beta_t) \mathbf{1}_{\mathbf{o}_t = \mathbf{o}_{t-1}} + \beta_t P(\mathbf{o}_t|\mathbf{s}_t). \end{aligned} \tag{2.6}$$

The **intra-option (action) policy** distribution can also be formulated as a mixture distribution

$$P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) = \prod_{\mathrm{i} \in \mathbf{o}_t} P_{\mathrm{i}}(\mathbf{a}_t|\mathbf{s}_t)^{\mathrm{i}}. \tag{2.7}$$

[3]Different from conventional formulation which only depends on state $\mathbf{s}_t$, our mixture master policy has extra dependencies on $\mathbf{o}_{t-1}$ and $\mathbf{b}_t$

Therefore, the dynamics of the PGM in figure 2.2 can be written as:

$$\begin{aligned} P(\tau) =& P(\mathbf{s}_0)P(\mathbf{o}_0)P(\mathbf{a}_0|\mathbf{s}_0,\mathbf{o}_0) \\ & \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_{t-1})P(\mathbf{b}_t|\mathbf{s}_t,\mathbf{o}_{t-1})P(\mathbf{o}_t|\mathbf{b}_t,\mathbf{s}_t,\mathbf{o}_{t-1})P(\mathbf{a}_t|\mathbf{s}_t,\mathbf{o}_t), \end{aligned} \tag{2.8}$$

where $P(\tau) = P(\mathbf{s}_0, \mathbf{o}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{b}_1, \mathbf{o}_1, \mathbf{a}_1, \ldots)$ denotes the joint distribution of the PGM. Notice that under this formulation, $P(\tau)$ is actually an HMM with $\mathbf{s}_t$, $\mathbf{a}_t$ as observable random variables and $\mathbf{b}_t$, $\mathbf{o}_t$ as latent variables.

It is worth to mention that equation (2.4) is essentially the indicator function $\mathbf{1}_{\mathbf{o}_t=\mathbf{o}_{t-1}}$ used in conventional SMDP option framework papers and the last line in equation (2.6) is identical to transitional probability distribution in their formulation. However, as we show in this section, by adding latent variables $\mathbf{o}_{t-1}$ and introducing the dependency between $\mathbf{o}_t$ and $\mathbf{b}_t$, our formulation is essentially an HMM. It opens the door to introduce many well developed PGM algorithms such as message passing [Forney, 1973] and variational inference [Hoffman et al., 2013] to the reinforcement learning framework. As we show below, the nice conditional independence relationships enjoyed by this model also enable us to prove the equivalence between the option framework's SMDP and MDP formulation.

2.2.3 MDP formulation for the Option Framework

With PGM in hand, we now prove that the HMM formulated MDP option framework has identical value functions with the conventional SMDP option framework[Bacon et al., 2017, Sutton et al., 1999]. In this section, we first show that all value functions defined on our PGM are identical to the SMDP formulation. We will prove that the gradients are also the same in next section.

We follow Sutton and Barto [2018]'s notation in this section and write value functions for MDP below:

$$
\begin{aligned}
V[\mathbf{s}_t] = \mathbb{E}[G_t|\mathbf{s}_t] &= \sum_{G_t} G_t \sum_{\mathbf{o}_t} P(G_t, \mathbf{o}_t|\mathbf{s}_t) \\
&= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) \sum_{G_t} G_t P(G_t|\mathbf{s}_t, \mathbf{o}_t) \\
&= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t] \\
&= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) Q_O[\mathbf{o}_t, \mathbf{s}_t], \qquad (2.9)
\end{aligned}
$$

where $V[\mathbf{s}_t]$ is the state value function[Sutton and Barto, 2018] and $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ is the option value function[Bacon et al., 2017, Sutton et al., 1999]. Note that in deriving equation (2.9) we only use summation rule and production rule, the conditional dependency relationships in PGM (figure 2.2) are not used. The option value function $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ can be further expanded as:

$$
\begin{aligned}
Q_O[\mathbf{o}_t, \mathbf{s}_t] = \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t] &= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
&= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t], \qquad (2.10)
\end{aligned}
$$

where $Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]$ is the option-action value function.

Proposition 2.2.1. *MDP formulation has identical state value function $V[\mathbf{s}_t]$ and option value function $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ to SMDP formulations*

Proof. Note that in derivations above we only use summation and production rules. Both equation (2.9) and (2.10) are identical to the conventional SMDP option framework. □

From now on, we will continue derivations with conditonal independence relationships encoded in PGM (Chapter 8.2.1 [Bishop, 2006a]). Following Bishop [2006a]'s notation, we use A, B and C to denote three non-overlapping sets of arbitrarily many random variables. Sets A and B are conditional independent on set C if $P(A, B|C) = P(A|C)P(B|C)$, denoted as $A \perp\!\!\!\perp B \mid C$. We mainly use head-to-tail conditional independence properties (Chapter 8.2.1 [Bishop, 2006a]) in this section. We have following conditional independence relationships from PGM (figure 2.2):

$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{b}_{t+1}\} \quad | \{\mathbf{o}_{t+1}\}, \tag{2.11}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{s}_t\} \quad | \{\mathbf{s}_{t+1}, \mathbf{o}_t\}, \tag{2.12}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{a}_t\} \quad | \{\mathbf{s}_{t+1}\}, \tag{2.13}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{o}_t\} \quad | \{\mathbf{s}_{t+1}, \mathbf{o}_{t+1}\}, \tag{2.14}$$
$$\{R_{t+1}, G_t, \mathbf{s}_{t+1}\} \perp\!\!\!\perp \{\mathbf{o}_t\} \quad | \{\mathbf{a}_t\}. \tag{2.15}$$

With above conditional independence relationships in hand, we now show that the MDP formulation has identical value functions to the conventional SMDP formulation[Sutton et al., 1999, Bacon et al., 2017].

Proposition 2.2.2. *MDP formulation has identical option-action value function* $Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]$ *to SMDP formulations*

$$Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) U[\mathbf{s}_{t+1}, \mathbf{o}_t]. \tag{2.16}$$

Proof.

$$\begin{aligned}
Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] =& \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
=& \mathbb{E}[R_{t+1} + \gamma G_{t+1}|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
=& \mathbb{E}[R_{t+1}|\mathbf{s}_t, \mathbf{a}_t] + \gamma \sum_{G_{t+1}} G_{t+1} \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{o}_t, \mathbf{a}_t) P(G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t) \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{G_{t+1}} G_{t+1} \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) P(G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t] \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) U[\mathbf{s}_{t+1}, \mathbf{o}_t].
\end{aligned}$$

□

Proposition 2.2.3. *MDP formulation has identical option-value function upon ar-*

rival $U[\mathbf{s}_{t+1}, \mathbf{o}_t]$ *to SMDP formulations*[4]

$$U[\mathbf{s}_{t+1}, \mathbf{o}_t] = (1-\beta_{t+1})Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] + \beta_{t+1}V[\mathbf{s}_{t+1}] \tag{2.17}$$

$$= Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] - \beta_{t+1}A[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}]. \tag{2.18}$$

Proof.

$$\begin{aligned}
U[\mathbf{s}_{t+1}, \mathbf{o}_t] =& \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t] \\
=& \sum_{G_{t+1}} G_{t+1} \\
& \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \\
=& \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \sum_{G_{t+1}} G_{t+1} P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{s}_{t+1}) \\
=& \sum_{\mathbf{o}_{t+1}} \left[(1-\beta_{t+1})\mathbf{1}_{\mathbf{o}_{t+1}=\mathbf{o}_t} + \beta_{t+1}P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1})\right] Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}] \\
=& (1-\beta_{t+1})Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] + \beta_{t+1}V[\mathbf{s}_{t+1}] \\
=& Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] - \beta_{t+1}A[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}].
\end{aligned}$$

from line 3 to line 4 use equation (2.11) and (2.14). From line 4 to line 5 use equation (2.6) and definition of Q_O. The second last line use equation (2.9). The last line use the definition of advantage function A. □

Under our MDP formulation, we also propose proposition 2.2.4. We derive our gradient theorems based on equation (2.19) in section 2.2.4. This important relationship largely simplify derivations than the original paper [Bacon et al., 2017] as well as give rise to the SA.

Proposition 2.2.4. *The option-value function upon arrival* $U[\mathbf{s}_{t+1}, \mathbf{o}_t]$ *is an expectation over option value function* $Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]$ *conditioned on previous option* O_t

$$U[\mathbf{s}_{t+1}, \mathbf{o}_t] = \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]. \tag{2.19}$$

[4]Both equations (2.17) and (2.18) is largely used in the conventional SMDP papers[Sutton et al., 1999, Bacon et al., 2017].

Proof. Following proof of proposition 2.2.3,

$$U[\mathbf{s}_{t+1}, \mathbf{o}_t] = \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \sum_{G_{t+1}} G_{t+1} P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{s}_{t+1})$$
$$= \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}].$$

□

2.2.4 Gradients for the MDP Option Framework

In above sections, we formulate dynamics of the option framework using HMM and prove the MDP build on it has identical value functions to SMDP formulation. In this section we will prove that both MDP and SMDP formulations [Bacon et al., 2017] share same intra-option and termination gradients. Our derivations is largely simplified by equation (2.19) compared to previous work.

Let θ_a denote parameter vector for intra-option policies $P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t; \theta_a)$ and θ_b denote parameter vector for termination policies $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}; \theta_b)$. To keep notation uncluttered, we drop the dependency on parameter vector θ in derivations below.

Proposition 2.2.5. *MDP formulation has identical Intra-Option Policy Gradient with SMDP formulation in [Bacon et al., 2017].*

$$\frac{\partial Q_O[\mathbf{s}_t, \mathbf{o}_t]}{\partial \theta_a} = \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}|\mathbf{s}_t, \mathbf{o}_t) \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \mathbf{o}_{t+k})}{\partial \theta_a} Q_U(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}, \mathbf{a}_{t+k}). \tag{2.20}$$

Proof. This is a direct result by taking gradient of θ_a with respect to equa-

tion (2.10) by using equation (2.16) and (2.19):

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \mathbf{o}_t]}{\partial \theta_a} &= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \frac{\partial Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]}{\partial \theta_a} \\
&= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
&\quad + \gamma \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \frac{\partial U[\mathbf{o}_t, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
&= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
&\quad + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{o}_t) \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
&= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{o}_{t+1}, \mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}, \mathbf{o}_{t+1}|\mathbf{s}_t, \mathbf{o}_t) \frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
&= \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}|\mathbf{s}_t, \mathbf{o}_t) \\
&\quad \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \mathbf{o}_{t+k})}{\partial \theta_a} Q_U(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}, \mathbf{a}_{t+k}).
\end{aligned}$$

□

Proposition 2.2.6. *MDP formulation has identical Termination Policy Gradient with SMDP formulation in [Bacon et al., 2017].*

$$\frac{\partial U[\mathbf{s}_{t+1}, \mathbf{o}_t]}{\partial \theta_b} = -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial \beta_{t+1+k}}{\partial \theta_b} A[\mathbf{s}_{t+k+1}, \mathbf{o}_{t+k+1} = \mathbf{o}_{t+k}]. \tag{2.21}$$

Proof. We first show the gradient of θ_b with respect to equation (2.6) and (2.10) separately:

$$\frac{\partial P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1},\mathbf{o}_t)}{\partial\theta_b} = \big[P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}) - \mathbf{1}_{\mathbf{o}_t=\mathbf{o}_{t-1}}\big]\frac{\partial\beta_{t+1}}{\partial\theta_b} \tag{2.22}$$

$$\begin{aligned}\frac{\partial Q_O[\mathbf{o}_{t+1},\mathbf{s}_{t+1}]}{\partial\theta_b} &= \sum_{\mathbf{a}_{t+1}} P(\mathbf{a}_{t+1}|\mathbf{s}_{t+1},\mathbf{o}_{t+1}) \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1},\mathbf{a}_{t+1})\frac{\partial U[\mathbf{s}_{t+2},\mathbf{o}_{t+1}]}{\partial\theta_b} \\ &= \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1},\mathbf{o}_{t+1})\frac{\partial U[\mathbf{s}_{t+2},\mathbf{o}_{t+1}]}{\partial\theta_b}. \end{aligned} \tag{2.23}$$

The equation (2.21) is a direct result by taking gradient of θ_b with respect to equation (2.19) and using above results:

$$\begin{aligned}\frac{\partial U[\mathbf{s}_{t+1},\mathbf{o}_t]}{\partial\theta_b} &= \sum_{\mathbf{o}_{t+1}} \frac{\partial P(\mathbf{o}_{t+1}|\mathbf{o}_t,\mathbf{s}_{t+1})}{\partial\theta_b} Q_O[\mathbf{o}_{t+1},\mathbf{s}_{t+1}] + \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t,\mathbf{s}_{t+1})\frac{\partial Q_O[\mathbf{o}_{t+1},\mathbf{s}_{t+1}]}{\partial\theta_b} \\ &= \sum_{\mathbf{o}_{t+1}} \big[P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}) - \mathbf{1}_{\mathbf{o}_t=\mathbf{o}_{t-1}}\big] Q_O[\mathbf{o}_{t+1},\mathbf{s}_{t+1}]\frac{\partial\beta_{t+1}}{\partial\theta_b} \\ &\quad + \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t,\mathbf{s}_{t+1})\gamma \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1},\mathbf{o}_{t+1})\frac{\partial U[\mathbf{s}_{t+2},\mathbf{o}_{t+1}]}{\partial\theta_b} \\ &= \big[V[\mathbf{s}_{t+1}] - Q_O[\mathbf{o}_{t+1}=\mathbf{o}_t,\mathbf{s}_{t+1}]\big]\frac{\partial\beta_{t+1}}{\partial\theta_b} \\ &\quad + \gamma \sum_{\mathbf{o}_{t+1},\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2},\mathbf{o}_{t+1}|\mathbf{s}_{t+1},\mathbf{o}_t)\frac{\partial U[\mathbf{s}_{t+2},\mathbf{o}_{t+1}]}{\partial\theta_b} \\ &= -A[\mathbf{o}_{t+1}=\mathbf{o}_t,\mathbf{s}_{t+1}]\frac{\partial\beta_{t+1}}{\partial\theta_b} + \gamma \sum_{\mathbf{o}_{t+1},\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2},\mathbf{o}_{t+1}|\mathbf{s}_{t+1},\mathbf{o}_t)\frac{\partial U[\mathbf{s}_{t+2},\mathbf{o}_{t+1}]}{\partial\theta_b} \\ &= -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+1+k},\mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+1+k},\mathbf{o}_{t+k}|\mathbf{s}_{t+1},\mathbf{o}_t)\frac{\partial\beta_{t+1+k}}{\partial\theta_b} A[\mathbf{s}_{t+k+1},\mathbf{o}_{t+k+1}=\mathbf{o}_{t+k}]. \end{aligned}$$

□

2.2.5 Summary

In this previous sections, we prove that an MDP equivalence to the SMDP. Briefly, we propose a novel MDP "mixture master policy" (Section 2.2.2). Unlike the conventional SMDP master policy that only depends on the current state, the mixture master policy has extra dependencies on the termination function and the previously activated option. We then prove that the MDP has identical optimal properties with the SMDP option framework [Sutton et al.,

1999] and identical policy gradients with the option-critic architecture [Bacon et al., 2017]. We summarize our results as the following theorem:

Theorem 2.2.7. *The SMDP formulated option framework, which employs Markovian options, has an underlying MDP equivalence.*

Proposition 2.2.8. *The MDP formulation has identical optimality (value functions) with the SMDP option framework [Sutton et al., 1999].*

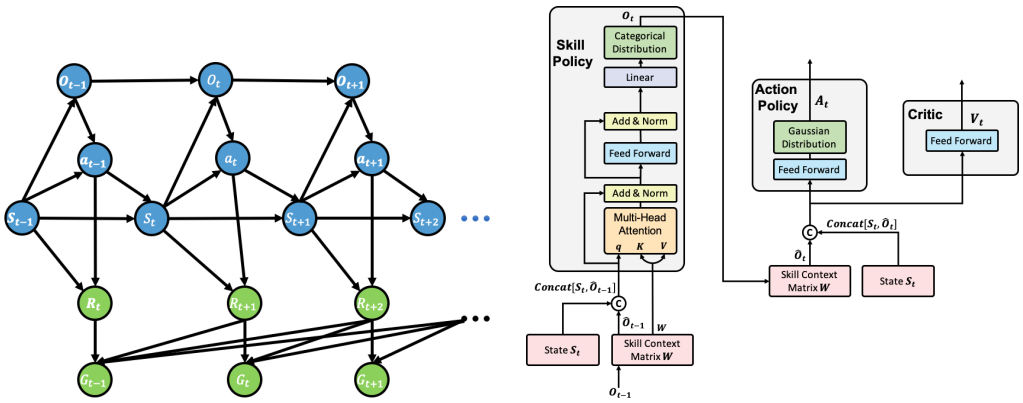
Proposition 2.2.9. *The MDP formulation has identical policy gradients with the option-critic architecture [Bacon et al., 2017].*

2.3 The Skill-Action Architecture

In this section, we propose a simple MDP [Puterman, 1994] implementation of the option framework: the Skill-Action (SA) architecture. Although the mixture master policy (Eq. 2.3) is MDP formulated, the master policy's (Eq. 2.2) gradients are still blocked by its dependency on the termination function. To overcome this, we present a marginalized derivation of the equivalence: the Skill-Action (SA) architecture. SA marginalizes the termination function away and models the marginalized policy (Eq. 2.6) directly with a "skill policy" (Eq. 2.25), which is used to replace both the master policy and termination function while implements their functionalities with the attention mechanism [Vaswani et al., 2017]. Section 2.3.1 describes the dynamics (Markov process) of SA. Section 2.3.2 defines value functions on top of the dynamics, thus formulating the MDP. Policy gradient theorems are then derived. Section 2.3.3 implements SA by employing neural networks and the Multi-Head Attention mechanism [Vaswani et al., 2017], which enables SA to temporally extend skills in the absence of the termination function.

2.3.1 Dynamics of the Skill-Action Architecture

In this section, we define the dynamics (Markov process) of SA. We first introduce MDP notations. A Markov decision process $M = \{\mathbb{S}, \mathbb{A}, r, P, \gamma\}$ consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a state transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t) : \mathbb{S} \to \mathbb{S}$, a reward function $r(\mathbf{s}, \mathbf{a}) : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, and a discount factor $\gamma \in \mathbb{R}$.



(a) PGM of SA (b) Network Architecture of SA

Figure 2.3: The Skill-Action (SA) Architecture

A policy $\pi = P(\mathbf{a}|\mathbf{s}) : \mathbb{S} \to \mathbb{A}$ is a probability distribution defined over actions conditioning on states. An expected discounted return is defined as $G_t = \sum_k^N \gamma^k R_{t+k+1}$, where $R \in \mathbb{R}$ is the actual reward received from the environment. The value function $V[\mathbf{s}_t] = \mathbb{E}_\pi[G_t|\mathbf{s}_t]$ is the expected return starting at state $\mathbf{s}_t$ and following policy π thereafter. The action-value function is defined as $Q[\mathbf{s}_t, \mathbf{a}_t] = \mathbb{E}_{\tau_\pi}[G_t|\mathbf{s}_t, \mathbf{a}_t]$. An Markov decision process together with value functions defined on it are referred to as an MDP [Puterman, 1994].

Having defined notations of MDP, we propose the dynamics of SA. Specifically, a **skill index vector o** $\in \mathbb{Z}_2^M$ is an M-dimensional one-hot vector, where M denotes the total number of skills to learn. Each entry $\mathrm{o} \in \{0, 1\}$ is a binary random variable. $\mathrm{o}_i = 1$ means that the i-th skill is activated. A **skill context matrix** [Kosiorek et al., 2019] $\boldsymbol{W}_S \in \mathbb{R}^{M \times E}$ is a learnable parameter containing M rows of E dimensional real vectors, the i-th row of $\boldsymbol{W}_S$ corresponds to the i-th skill o_i, and different columns encode different properties of a skill. A **skill context vector** $\hat{\mathbf{o}}_t$ is defined as:

$$\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \mathbf{o}_t, \ \hat{\mathbf{o}}_t \in \mathbb{R}^E. \tag{2.24}$$

The **skill policy** is defined as:

$$P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}; \boldsymbol{W}_s) : \mathbb{S} \times \mathbb{R}^E \to \mathbb{R}^E, \tag{2.25}$$

which is a probability distribution over skill context vector $\hat{\mathbf{o}}_t$ conditioned on state $\mathbf{s}_t$ and previous sampled skill context vector $\hat{\mathbf{o}}_{t-1}$, with $\boldsymbol{W}_S$ as its learnable

parameters.

The **action policy** is defined as:

$$P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t) : \mathbb{S} \times \mathbb{R}^E \to \mathbb{A}, \tag{2.26}$$

which is a probability distribution over the action random variable $\mathbf{a}_t \in \mathbb{A}$ conditioned on the skill context vector $\hat{\mathbf{o}}_t$ and state $\mathbf{s}_t$, and decodes them into primary actions.

With both skill and action policies in hand, the dynamics of the SA are defined as a Probabilistic Graphical Model (PGM) [Koller and Friedman, 2009] (Figure 2.3 (a)):

$$\begin{aligned} P(\tau) =& P(\mathbf{s}_0)P(\hat{\mathbf{o}}_0)P(\mathbf{a}_0|\mathbf{s}_0, \hat{\mathbf{o}}_0) \\ & \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t), \end{aligned} \tag{2.27}$$

where $P(\tau) = P(\mathbf{s}_0, \hat{\mathbf{o}}_0, \mathbf{a}_0, \mathbf{s}_1, \hat{\mathbf{o}}_1, \mathbf{a}_1, \ldots)$ denotes the joint distribution of the PGM. Note that under this formulation, $P(\tau)$ is actually an Hidden Markov Model (HMM) with $\mathbf{s}_t$, $\mathbf{a}_t$ as observable random variables and $\hat{\mathbf{o}}_t$ as latent variables.

2.3.2 MDP of the Skill-Action Architecture

With SA's dynamics in hand, in this section, we first propose a novel "skill value upon arrival function" and theoretically prove that it has a smaller variance than the conventional value function. This property is empirically justified in Section 3.1 and further discussed in Section 3.4. Then, we derive the recursive formulation of value functions and formulate the MDP. Based on the MDP, skill and action policies' gradients theorems are finally derived.

Rather than use the conventional value function $V[\mathbf{s}_t]$, we define the **skill value upon arrival function** $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ as:

$$V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] = \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] = \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]. \tag{2.28}$$

Proof. Derivations of Eq. (2.28):

$$\begin{aligned} V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] =& \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\mathbb{E}(G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \hat{\mathbf{o}}_{t-1}) \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t] \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})Q_O[\hat{\mathbf{o}}_t, \mathbf{s}_t], \end{aligned}$$

where from line 2 to line 3 we use the conditional independence property in PGM that $G_t \perp\!\!\!\perp \hat{\mathbf{o}}_{t-1}|\{\mathbf{s}_t, \hat{\mathbf{o}}_t\}$. □

Proposition 2.3.1. *$V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is an unbiased estimation of $V[\mathbf{s}_t]$.*

Proof. By law of total expectation:

$$\mathbb{E}_{\hat{\mathbf{o}}_{t-1}}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]] = \mathbb{E}_{\hat{\mathbf{o}}_{t-1}}[\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]] = \mathbb{E}[G_t|\mathbf{s}_t] = V[\mathbf{s}_t]$$

thus $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is an unbiased estimator of $V[\mathbf{s}_t]$. □

Proposition 2.3.2. *The variance of $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is less than or equal to $V[\mathbf{s}_t]$.*

Proof. By law of total conditional variance:

$$\begin{aligned} \mathrm{Var}(V[\mathbf{s}_t]) &= \mathrm{Var}([\mathbb{E}[G_t|\mathbf{s}_t]]) = \mathbb{E}[\mathrm{Var}(\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}])|\mathbf{s}_t] + \mathrm{Var}(\mathbb{E}[\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t) \\ &= \mathbb{E}[\mathrm{Var}(V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}])|\mathbf{s}_t] + \mathrm{Var}(\mathbb{E}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t) \\ &\geq \mathrm{Var}(\mathbb{E}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t). \end{aligned}$$

□

Eq. (2.28) states that the skill value function upon arrival is an expectation over skill value function $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ conditioned on previous skill $\hat{\mathbf{o}}_{t-1}$. The **skill value function** $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ is defined as:

$$Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] = \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t] = \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t], \tag{2.29}$$

and the **skill-action value function** $Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t]$ is defined as:

$$\begin{aligned} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] &= \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] \\ &= r(s,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t], \end{aligned} \tag{2.30}$$

where $\gamma \in \mathbb{R}$ is a discounting factor.

Proof.

$$\begin{aligned} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] =& \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] \\ =& r(s,o,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t) \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] \\ =& r(s,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t] \\ =& r(s,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t], \end{aligned}$$

where from line 2 to line 3 we use the conditional independence property in PGM that $R_{t+1} \perp\!\!\!\perp \hat{\mathbf{o}}_t | \mathbf{a}_t$, $G_{t+1} \perp\!\!\!\perp \mathbf{s}_t | \{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t\}$ and $G_{t+1} \perp\!\!\!\perp \mathbf{a}_t | \mathbf{s}_{t+1}$. $\gamma \in \mathbb{R}$ is a discounting factor. □

Expanding (Eq. 2.30) with (Eq. 2.28) gives us a recursion formulation from which Bellman equations and policy gradient theorems are derived. To keep notations uncluttered, we use θ_o to denote skill policy's parameters (Eq. 2.25) and θ_a to denote action policy's parameters (Eq. 2.26). The skill and action policies' gradient theorems are:

Theorem 2.3.3. ***Skill Policy Gradient Theorem:*** *Given a stochastic skill policy differentiable in its parameter vector θ_o, the gradient of the expected discounted return with respect to θ_o is:*

$$\frac{\partial V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]}{\partial \theta_o} = \mathbb{E}[\ \frac{\partial P(\hat{\mathbf{o}}'|\mathbf{s}', \hat{\mathbf{o}})}{\partial \theta_o} Q_O[\mathbf{s}', \hat{\mathbf{o}}'] \mid \mathbf{s}_t, \hat{\mathbf{o}}_{t-1}], \tag{2.31}$$

where $\hat{\mathbf{o}}'$ is one time step later than $\hat{\mathbf{o}}$.

Proof.

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_o} &= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)\big[r(s,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o}\big] \\
&= \sum_{\mathbf{s}_{t+1}} \gamma P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t)\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
\frac{\partial V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]}{\partial \theta_o} &= \sum_{\hat{\mathbf{o}}_t} \frac{\partial P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})}{\partial \theta_o} Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] + \gamma \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\frac{Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
&= \sum_{\hat{\mathbf{o}}_t} \frac{\partial P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})}{\partial \theta_o} Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] + \gamma \sum_{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t} P(\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
&= -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1}} \\
&\quad P_\gamma^{(k)}(\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1}|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}) \sum_{\hat{\mathbf{o}}_{t+k}} \frac{\partial P(\hat{\mathbf{o}}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1})}{\partial \theta_o} Q_O[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}] \\
&= \mathbb{E}[\ \frac{\partial P(\mathbf{o}'|\mathbf{s}', \mathbf{o})}{\partial \theta_o} Q_O[\mathbf{s}', \mathbf{o}'] \mid \mathbf{s}_t, \hat{\mathbf{o}}_{t-1}].
\end{aligned}$$

□

Theorem 2.3.4. ***Action Policy Gradient Theorem:*** *Given a stochastic action policy differentiable in its parameter vector θ_a, the gradient of the expected discounted return with respect to θ_a is:*

$$\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_a} = \mathbb{E}[\ \frac{\partial P(\mathbf{a}|\mathbf{s}, \hat{\mathbf{o}})}{\partial \theta_a} Q_A[\mathbf{s}, \hat{\mathbf{o}}, \mathbf{a}] \mid \mathbf{s}_t, \hat{\mathbf{o}}_t]. \tag{2.32}$$

Proof. Similar to the first equation above, continue expanding gradients of $\frac{\partial Q_O}{\partial \theta_a}$

by equations (2.28) (2.29) and (2.30):

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_a} =& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)}{\partial \theta_a} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t) \frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_a} \\
=& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)}{\partial \theta_a} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}} P(\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t) \frac{\partial Q_O[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}]}{\partial \theta_a} \\
=& - \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}} \\
& P_\gamma^{(k)}(\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}|\mathbf{s}_t, \hat{\mathbf{o}}_t) \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k})}{\partial \theta_a} Q_A[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}, \mathbf{a}_{t+k}] \\
=& \mathbb{E}[\frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k})}{\partial \theta_a} Q_A[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}, \mathbf{a}_{t+k}] \mid \mathbf{s}_t, \hat{\mathbf{o}}_t].
\end{aligned}$$

□

Compared to MDP formulated algorithms, SMDP option frameworks are sample inefficient and notoriously unstable to hyperparameters [Zhang and Whiteson, 2019]. The skill and action policies' gradient theorems enable SA to be compatible with both MDP on-policy and off-policy algorithms, and thus has much better stability and convergence. This work is focused on deriving MDPs of SA and its policy gradient theorems. To be comparable with previous work [Zhang and Whiteson, 2019], in this paper we directly apply PPO [Schulman et al., 2017] to our learning algorithm (Algorithm 1). Recent work [Wulfmeier et al., 2020] shows that the off-policy algorithm gives a performance boost to option variants. Designing off-policy algorithms for SA remains open for future work.

2.3.3 Networks Architecture

After deriving the MDP of SA, we present a simple neural network implementation of the Skill-Action Architecture (Figure 2.3). Unlike the conventional SMDP option framework, which employs a termination function and an SMDP master policy to temporally extend the execution of an option, SA implements the temporal extension functionality by employing the Multi-Head Attention (MHA) mechanism [Vaswani et al., 2017].

Specifically, an attention mechanism is described as the mapping from a

query $\mathbf{q} \in \mathbb{R}^E$ and a set of key-value pairs, i.e., $\boldsymbol{K} \in \mathbb{R}^{M \times E}$ and $\boldsymbol{V} \in \mathbb{R}^{M \times E}$ (M and E are total number of skills and embedding dimensions defined in section 2.3.1), to an output:

$$Attention(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\mathbf{q}\boldsymbol{K}^T}{\sqrt{E}})\boldsymbol{V} \tag{2.33}$$

A Multi-Head Attention $\text{MHA}(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V})$ is a linear projection of h (number of heads) concatenated linearly projected $Attention$ outputs:

$$\begin{aligned} \text{MHA}(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}[\text{head}_1, \ldots, \text{head}_h]\boldsymbol{W}^H \\ \text{where head}_i &= Attention(\mathbf{q}\boldsymbol{W}_i^q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \end{aligned} \tag{2.34}$$

where projections are parameter matrices $\boldsymbol{W}_i^q \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^O \in \mathbb{R}^{hE \times E}$. In this paper we use MHA as one building block as illustrated in Figure 2.3.

SA employs MHA as the skill policy. At each time step, the skill policy $P(\hat{\mathbf{o}}_t | \mathbf{s}_t, \hat{\mathbf{o}}_{t-1}; \boldsymbol{W}_s)$ attends to (measures the compatibility of) all skill context vectors in $\boldsymbol{W}_s$ according to $\mathbf{s}_t$ and $\hat{\mathbf{o}}_{t-1}$. If $\hat{\mathbf{o}}_{t-1}$ still fits $\mathbf{s}_t$, then the skill policy assigns a larger attention weight to $\hat{\mathbf{o}}_{t-1}$, thus has a tendency to continue with it. Otherwise, a new skill with better compatibility will be sampled. The action policy is as simple as one decoder to decode $\hat{\mathbf{o}}_t$ and $\mathbf{s}_t$ into primary actions $\mathbf{a}_t$. The attention mechanism together with skill context vectors enable SA to temporally extend skills even in the absence of termination functions.

Specifically, a **skill policy** (Eq. (2.25)) uses a concatenation of current state $\mathbf{s}_t$ and previous skill context vector $\hat{\mathbf{o}}_{t-1}$ as the query for MHA. Both key and value matrices are the skill context matrix $\boldsymbol{W}_S$. In this way, we have:

$$\hat{\mathbf{s}}_{t-1} = linear(\text{Concat}[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]), \tag{2.35}$$

$$\mathbf{d}_t^O = \text{MHA}(\hat{\mathbf{s}}_{t-1}, \boldsymbol{W}_S, \boldsymbol{W}_S), \tag{2.36}$$

$$\mathbf{o}_t \sim Categorical(\mathbf{o}_t | \mathbf{d}_t^O), \tag{2.37}$$

where the *linear* layer simply projects the concatenated vector to E dimension. MHA is employed to attend to (measures the compatibility of) all skills in $\boldsymbol{W}_s$ according to $\mathbf{s}_t$ and $\hat{\mathbf{o}}_{t-1}$. The **skill density vector** $\mathbf{d}_t^O$ is then used as densi-

ties for a Categorical distribution $P(\mathbf{o}_t|\mathbf{d}_t^O)$, from which the new one-hot **skill index vector** $\mathbf{o}_t$ is sampled from. We can retrieve the **skill context vector** by $\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \mathbf{o}_t$. With the skill context vector $\hat{\mathbf{o}}_t$ in hand, the **action policy** can be designed as simple as a multi-layer Feed-Forward Networks (FFN) decoder:

$$\mathbf{d}_t^A = \text{FFN}(\mathbf{s}_t, \hat{\mathbf{o}}_t), \tag{2.38}$$

$$\mathbf{a}_t \sim P(\mathbf{a}_t|\mathbf{d}_t^A), \tag{2.39}$$

where $\mathbf{d}_t^A$ is a density vector and P is an arbitrary probability distribution (works for both discrete and continuous situations).

Similar to Zhang and Whiteson [2019], because of the **skill value upon arrival function** $V(\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})$, (Eq. 2.28) is an expectation of the **skill value function** $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ (Eq. 2.29). It is sufficient for us to model only one critic function:

$$Q_O = \text{FFN}(\mathbf{s}_t, \hat{\mathbf{o}}_t), \tag{2.40}$$

where Q_O is implemented as a multi-layer FFN. We summarize the detailed learning procedures in Algorithm 1.

Since $\hat{\mathbf{o}}_t$ encodes all context of a skill, SA only needs one action policy decoder to decode the activated skill context vectors $\hat{\mathbf{o}}_t$ and current state $\mathbf{s}_t$ into primary actions $\mathbf{a}_t$. This design choice largely improves the scalability of SA: adding one more skill is as cheap as adding a skill context vector. Moreover, unlike the option framework, in which only the activated option's action policy gets updated, the action policy learns to decode each dimension of skill context vectors at every time step. This design choice largely improves sample efficiency and enables SA to converge faster than the conventional option framework.

Algorithm 1 Learning Algorithm for the Skill-Action architecture

Initialize the skill embedding matrix $\boldsymbol{W}_S$ Assign Initial State: $\mathbf{s}_t \leftarrow \mathbf{s}_0$ Assign Initial Skill: $\hat{\mathbf{o}}_{t-1} \leftarrow \hat{\mathbf{o}}_0$

while *Converge* **do**

  # Rollout trajectories and store in replay buffer **repeat**

    Retrieve the skill context vector $\hat{\mathbf{o}}_{t-1} = \boldsymbol{W}_S^T \cdot \hat{\mathbf{o}}_{t-1}$ Sample $\hat{\mathbf{o}}_t \sim P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})$ Retrieve the skill context vector $\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \hat{\mathbf{o}}_t$ Sample $\mathbf{a}_t \sim P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)$ Compute $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ and $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ Take action $\mathbf{a}_t$ in $\mathbf{s}_t$, observe new state $\mathbf{s}_{t+1}$ and reward R_{t+1}

  until *Rollout Length Reached*;

  # Compute Advantages for skill & action policies Assign t reversely, from $RolloutLength - 1$ to 1 **repeat**

    Compute skill Advantage $A_t^O = R_{t+1} + \gamma(V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t] - V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]) + \gamma\lambda A_{t+1}^O$ Compute action Advantage $A_t^A = R_{t+1} + \gamma(Q_O[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}] - Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]) + \gamma\lambda A_{t+1}^A$

  until *Rollout Length Reached*;

  # λ is the GAE coefficient used in PPO.

  # Optimize PPO Obj **while** $i <$ *PPO Optimization Epochs* **do**

    $\theta_o \leftarrow PPO(\frac{\partial P(\mathbf{o}'|\mathbf{s}', \mathbf{o})}{\partial \theta_o}, A^O)$ $\theta_a \leftarrow PPO(\frac{\partial P(\mathbf{a}|\mathbf{s}, \mathbf{o})}{\partial \theta_a}, A^A)$

  end

end

Chapter 3

Application: Multi-timescale Sequence Learning on Robots Simulation Environments

In this section, we design experiments to answer three questions: 1) Can SA outperform other baselines (regarding episodic returns, stability, and scalability)? 2) Can SA temporally extend skills without the termination function? 3) Can skill context vectors be easily interpreted? 4) Does skill embeddings learned by SA have a performance boost over other option variants in transfer learning settings?

For single task learning, experiments are conducted on all OpenAI Gym MuJoCo environments (10 environments) [Brockman et al., 2016]. We follow DAC [Zhang and Whiteson, 2019] and compare our algorithm with five baselines, four of which are option implementations, i.e., DAC+PPO [Zhang and Whiteson, 2019], AHP+PPO [Levy and Shimkin, 2011], PPOC [Klissarov et al., 2017] and OC [Bacon et al., 2017]. The last baseline is PPO [Schulman et al., 2017]. All baselines' parameters used in DAC[1] remain unchanged other than the maximum number of training steps: SA only needs 1 million steps to converge rather than the 2 million used in DAC. For transfer learning, we follow Zhang and Whiteson [2019] and run 6 pairs of transfer learning tasks constructed in DAC based on DeepMind Control Suite [Tassa et al., 2020]. For a fair comparison, we use four skills for SA and four options for other option

[1]All baselines' implementations are from DAC's open source repo https://github.com/ShangtongZhang/DeepRL/tree/DAC. Note that the author list of this paper does not have any overlap with DAC. Our source code is available in supplementary materials.

implementations. All experiments are run on an Intel® Core™ i9-9900X CPU @ 3.50GHz with a single thread and process. Our implementation details are summarized in Section 3.5.

3.1 Single Task Learning

3.1.1 Performance

In Figure 3.1, we report episodic returns on infinite horizon (i.e., HalfCheetah, Swimmer, HumanoidStandup and Reacher) and finite horizon (the other tasks) [2] environments separately. For a fair comparison, we use exactly the same plotting script as used in DAC: curves are averaged over 10 independent runs and smoothed by a sliding window of size 20. Shaded regions indicate standard deviations.

[2] We refer to environments with the game-over condition as finite horizon environments, and infinite vice versa.

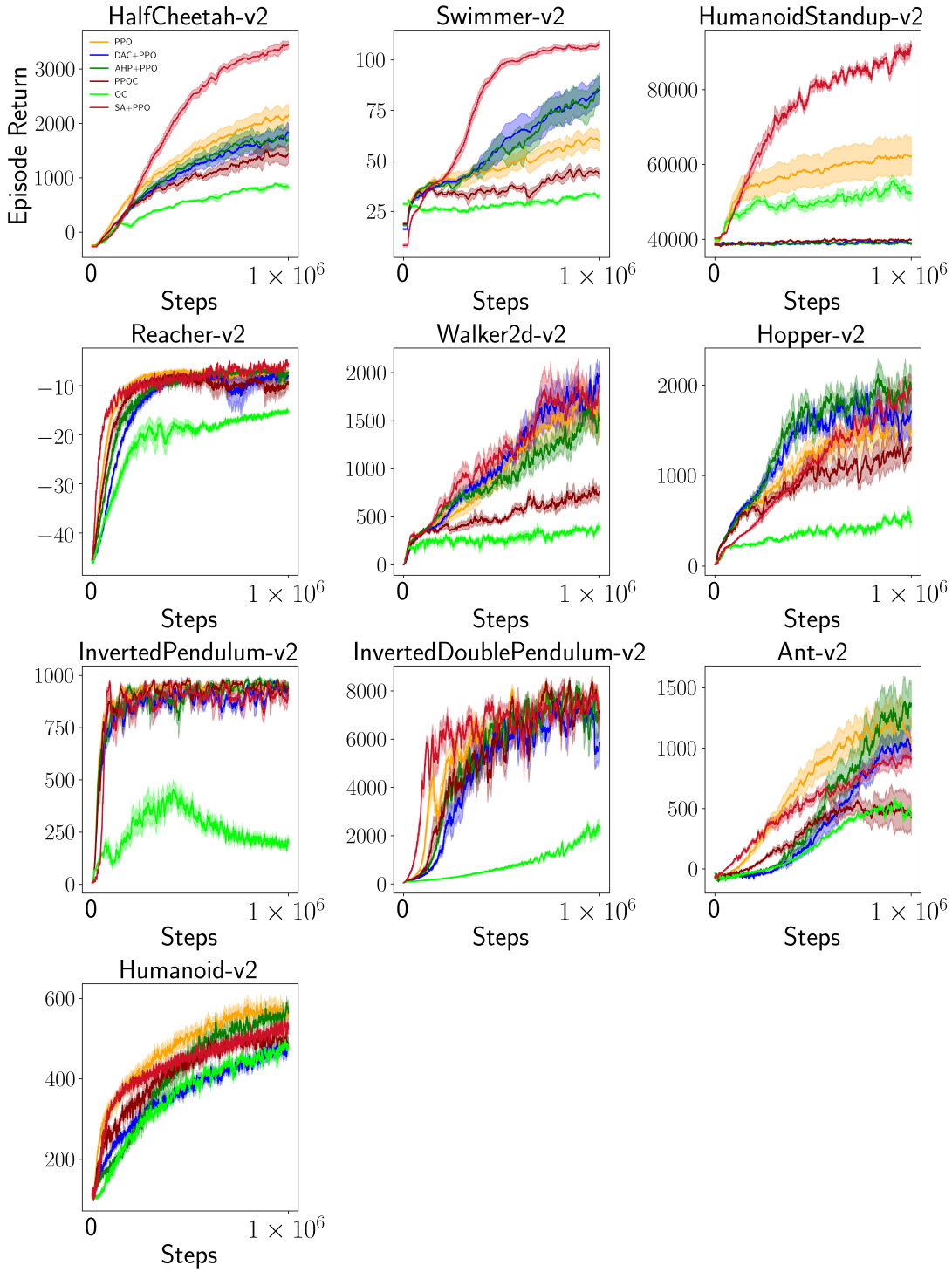


Figure 3.1: Performance of Ten OpenAI Gym MuJoCo Environments.

Table 3.1: Performance of Infinite Horizon Environments

	HalfCheetah	Swimmer	HumanoidStandup	Reacher
PPO	2143.6	59.9	62262.2	-7.5
DAC+PPO	1830.1	85.0	38954.9	-8.1
AHP+PPO	1701.7	86.7	38684.9	-7.3
PPOC	1441.2	43.6	39841.7	-9.4
OC	832.3	33.0	52352.7	-15.3
SA+PPO	**3446.7**	**107.8**	**91654.5**	**-4.6**

Table 3.2: Performance of Finite Horizon Environments

	Walker2d	Hopper	InvertedPendulum	InvertedDoublePendulum	Ant	Humanoid
PPO	1512.5	1489.9	939.9	7112.6	1049.6	562.1
DAC+PPO	**1968.0**	1702.2	**943.7**	5804.5	985.8	487.6
AHP+PPO	1520.6	**1993.6**	940.0	**7120.7**	**1359.3**	**569.3**
PPOC	756.1	1308.1	936.2	7117.6	429.4	483.9
OC	391.9	487.6	207.1	2369.4	433.4	475.1
SA+PPO	1856.9	1955.3	906.5	6884.1	907.4	528.7

It is extremely interesting that SA shows two completely different kinds of behaviors on infinite and finite horizon environments. According to previous option framework implementations [Klissarov et al., 2017, Smith et al., 2018, Harb et al., 2018, Zhang and Whiteson, 2019], on single task environments, option-based algorithms do not have a distinguishable performance boost over hierarchy-free algorithms. SA also has similar behavior and achieves comparable performance to the best baseline algorithm on most finite horizon environments (Figure 3.1). In Section 3.4 we conceptually explain that conventional value functions are insufficient to approximate models which have temporal latent variables dependencies. A concrete deep wide skill-action architecture remains open for the future work.

On infinite horizon environments as shown in Figure 3.1 (a), SA's performance significantly outperforms all baselines by a large margin in various aspects. For episodic return, e.g., HumanoidStandup, all option implementations barely converge, while SA is 240% better than DAC and AHP[3]. For convergence, SA has the fastest convergence speed. On the first two environ-

[3]Even on Reacher, a simple environment on which most algorithms converge to a similar performance, SA is still 38% better than the second best (AHP).

ments, which are also reported in DAC, SA only takes 40% of time steps of DAC and AHP to reach similar episodic returns. This acceleration is because: 1) SA is MDP formulated, the skill policy is updated at each time step; 2) SA only has one action policy decoder; 3) the action decoder learns to decode skill context vectors whichever skill is activated. For stability, all 10 runs of SA converge to a similar level while the other have much larger standard deviations. This property is theoretically justified by Proposition 2.3.2 and further discussed in Section 3.4.

3.1.2 Temporal Extension

It is logical to ask whether SA is capable of temporal extension without the termination function. To illustrate this, we plot the average duration of each skill during training episodes of the HalfCheetah environment in Figure 3.2 (a) and 4 runs of skill activation sequences in Figure 3.3. In Figure 3.2, we plot the average duration of each skill during 430 training episodes (each episode contains a trajectory of 512 time steps) of the HalfCheetah environment. In this environment, the agent learns to run half of a Cheetah by controlling 6 joints: back thigh, back shin, back foot, front thigh, front shin, and front foot. The faster the Cheetah runs forward, the higher return it gets from the environment.

At the start of training, all skills' durations are short. After the 100-th episode, Skill 2's duration quickly grows and dominates the entire episode. This growth of duration proves that SA can still temporally extend a skill.

The dominant skill phenomenon is also reported in other option implementations such as DAC. One explanation for this domination phenomenon is that for some single task environments, primitive actions might be enough to express the optimal policy, in which case extra levels of abstraction (skills) become overhead. However, towards the end of the training, the dominant skill's duration starts to decrease while the duration of a secondary skill (Skill 1) starts to increase. These facts indicate that SA has a better capability of automatically discovering abstract actions from primary actions as well as coordinating between them. The dominant skill phenomenon is also reported in other option implementations such as DAC. However, as shown in the video[4],

[4]https://www.youtube.com/watch?v=QiLVZvI6NJU

SA still learns distinguishable skills. Skill 2 is the running forward skill thus it dominates the whole episode. Skill 1 is only used to recover from falling down thus it has much shorter duration. As discussed in Section 3.4, solution to the dominant skill is actually learning skills at much finer granularity. The SA-style wide value function (Eq. (2.28)) provides an elegant solution to this problem.

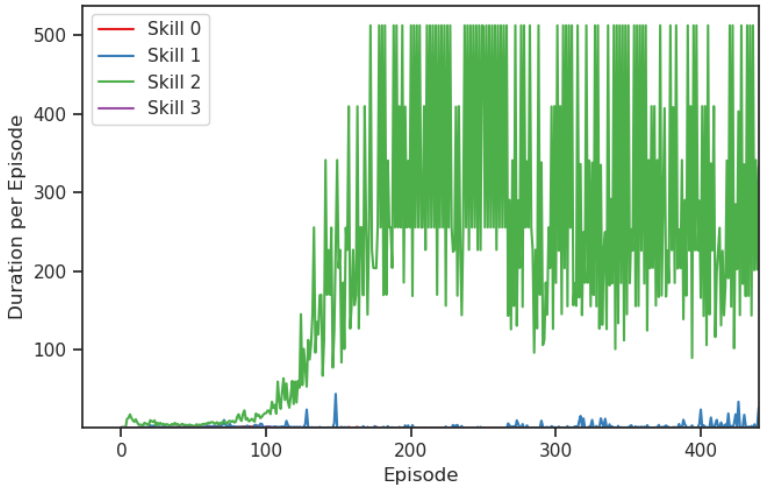


Figure 3.2: Duration of 4 options during 430 training episodes of HalfCheetah.

To illustrate how SA coordinates skills, we take the HalfCheetah model trained after 1 million steps and independently run it 4 times (4 episodes. each episode contains 512 time steps). Skill activation sequences of 4 runs are then plotted in Figure 3.3. As we can see that there are some common patterns

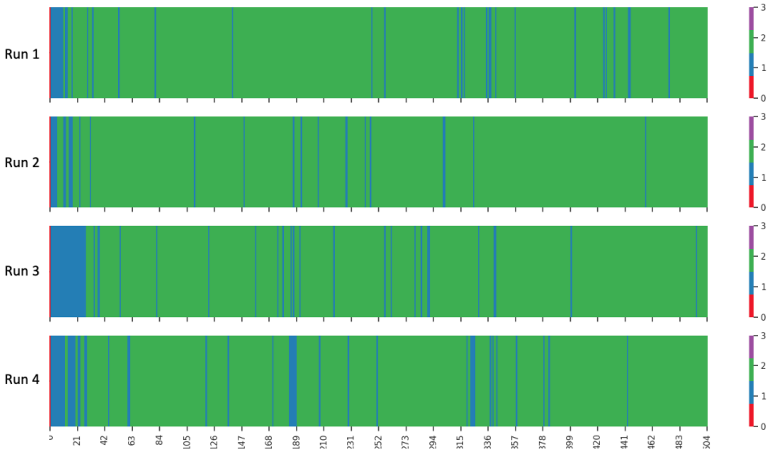


Figure 3.3: Activated option sequences of 4 independent HalfCheetah runs.

between all 4 independent runs. For example, all runs start with Skill 0 and use Skill 1 at the early stage. After executing Skill 1 for a short period, they all switch to Skill 2 which has longest durations in all 4 runs. From time to time they will fall back to Skill 1 for short periods and quickly switch to Skill 2 again. This pattern of coordination indicates that Skill 1 and Skill 2 have completely different functionality and SA has the capability of automatically discovering as well as leveraging those skills.

3.1.3 Interpretation of Skill Context Vectors

In this section we continue with the HalfCheetah model used in Section 3.1.2 and demonstrate how to interpret skill context vectors as well as skill activation sequences (Figure 3.3). In HalfCheetah, the agent learns to run half of a Cheetah by controlling 6 joints: back thigh, back shin, back foot, front thigh, front shin, and front foot. The faster the Cheetah runs forward, the higher return it gets from the environment. We interpret skill context vectors and activation patterns by first inspecting what property each dimension of the skill context vector encodes (Figure 3.5). Once each dimension is understood, skills (Figure 3.4) become straight forward to interpret by simply inspecting on which dimension (property) they have the most significant weights (Figure 3.6). These interpretations can further be taken to explain skill activation patterns in Figure 3.3.

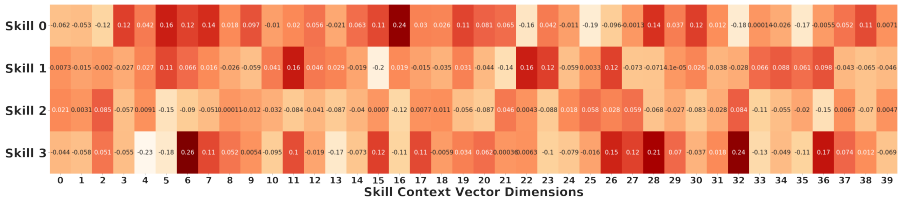


Figure 3.4: Heatmap of all 4 skill context vectors

As the first step, we follow Sabour et al. [2017] to interpret what property each dimension of the skill context vector in Figure 3.4 encodes by perturbing each dimension and decode perturbed skill context vectors into primary actions. Specifically, we perturb one dimension by adding a range of perturbations $[-0.1, 0.09]$ by intervals of 0.01 onto it while keep the other dimensions fixed. After perturbation, each skill context vector dimension has 20 perturbed

vectors. We then use the action policy decoder to decode all those vectors into primary actions and see how the perturbation affects the primary action. As an illustration, we plot Dimension 0's all 20 perturbed results in Figure 3.5.

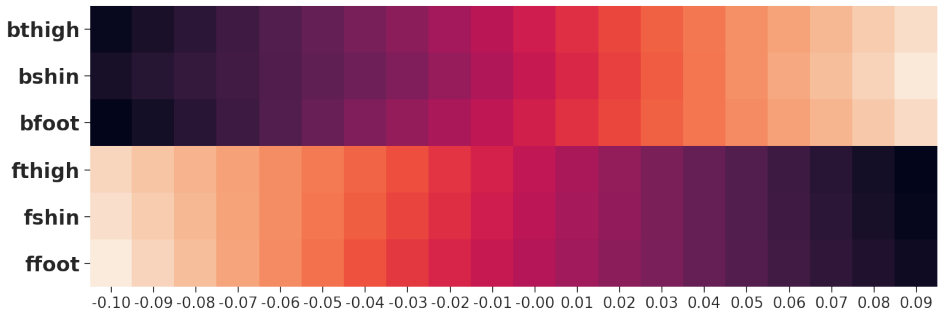


Figure 3.5: Perturbation on the Dim 0

With visualization of perturbation results in hand, we can interpret what property each dimension encode by inspecting relationships between perturbations and primary actions. In Figure 3.5, as an example, it is clear that changes on Dim 0 has opposite effect on the back leg and front leg: a larger value on Dim 0 will assign the back leg a larger torque while the front leg a smaller one, and vice versa. This means Dim 0 is has a focus point property: it focuses torque on only one leg.

Once we know how to interpret one dimension, we can move on to interpret the whole skill context vector. Since Skill 1 and Skill 2 are two main skills employed in Figure 3.3, here we provide an example of how to interpret them. Figure 3.4 shows that Skill 1 has significant values on dimension 11, 15 and 22. Skill 2 is significant on dimension 2, 5 and 36. We demonstrate these dimensions in the same manner as Figure 3.5 below:

Subfigures in Figure 3.6 can be interpreted in the same manner as Figure 3.5. As an example, from Figure 3.4 we can see that Skill 1 has a significant small value on Dim 11. In Figure 3.6, it shows that a smaller Dim 11 will twist the front leg forward and back foot forward while twist back thigh, back shin backward. Composition of these movements is a back leg landing property. Similarly, we can interpret that Dim 15 is a front leg landing property and Dim 22 is a balancing property. Therefore, Skill 1 is focusing on landing from all positions.

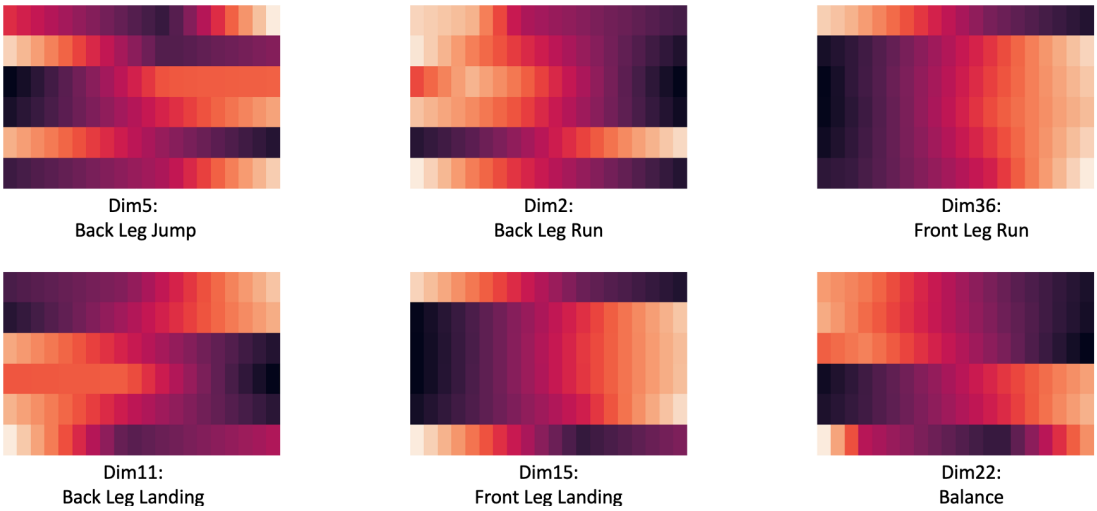


Figure 3.6: Interpretation of Skill 1 and Skill 2

Unlike other skill context vectors which have apparent focusing dimensions, Skill 2 has a rather balanced skill context vector. It has no apparently dominant dimension. It only has slightly more significant values on Dim 2, 5, 36, which are focusing on jumping and running properties. Therefore, Skill 2 is more like an "all-weather" skill: it is a skill having very balanced properties with a slightly demonstration on running and jumping.

Interpretations of Skill 1 and 2 above can then be taken to understand skill activation patterns in Figure 3.3: as an all-weather skill, Skill 2 is the most frequently executed one and has the longest duration. From time to time, when the Cheetah needs to land and balance itself, Skill 1 will be executed. However, since landing skill does not provide power of moving forward and thus has lower returns to continue, once the body is balanced the Cheetah will quickly stop Skill 1's execution and keep running with Skill 2.

3.2 Transfer Learning

We follow Zhang and Whiteson [2019] and run 6 pairs of transfer learning tasks constructed in DAC based on DeepMind Control Suite [Tassa et al., 2020]. Each pair contains two different tasks. To keep consistent with DAC, we train all models one million steps on the first task and switch to the second (SA's skill context matrix is subsequently frozen) to run another one million steps.

Results are reported in Figure 3.7 (Table 3.3). On the first task, SA's performance is among the best algorithms in all environments. This further validates SA's advantages on single task as observed in section 3.1. On the transfer learning (the second) task, SA's performance ranks the first in 5 out of 6 environments. This shows SA's advantages in knowledge reuse tasks.

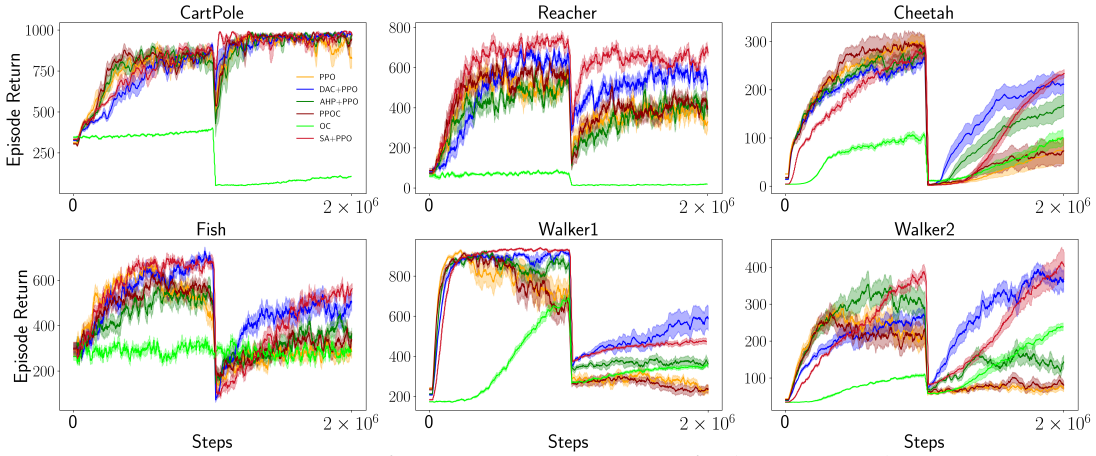


Figure 3.7: Performance on DAC transfer learning tasks

Table 3.3: Performance of Deepmind Control Suite Transfer Learning Environments

	CartPole	Reacher	Cheetah	Fish	Walker1	Walker2
PPO	829.7	327.6	73.0	287.9	231.8	72.2
DAC+PPO	970.8	517.2	211.2	505.4	**590.3**	360.5
AHP+PPO	966.5	395.2	167.4	357.9	362.1	143.2
PPOC	942.1	400.1	72.7	336.7	236.6	80.9
OC	106.1	19.4	100.6	286.6	356.3	238.7
SA+PPO	**974.1**	**675.3**	**233.8**	**562.1**	473.8	**403.0**

3.3 Conclusions

In this paper, we presented a novel MDP equivalence of the SMDP formulated option framework, from which an MDP implementation of the option framework, i.e., the Skill-Action architecture, was derived. We theoretically proved that SA has lower variance than conventional RL models and provided policy gradient theorems for updating SA. Our empirical studies on challenging infinite horizon robot simulation environments demonstrated that SA not only

outperforms all baselines by a large margin, but also exhibits smaller variance, faster convergence, and good interpretability. On transfer learning, SA also outperforms the other models in 5 out of 6 environments and shows its advantages in knowledge reuse tasks.

The final and most important contribution of SA is hierarchically learning explicit abstract actions' representations with "skill context vectors". This design significantly improves the scalability and interpretability of SA. It is straightforward to extend SA to deeper and wider (Section 3.4) architectures, which gives rise to a large-scale pre-training and transfer learning architecture in the reinforcement learning area.

Experiments also show that SA shares two innate limitations with the conventional option framework [Levy and Shimkin, 2011, Klissarov et al., 2017, Smith et al., 2018, Harb et al., 2018, Zhang and Whiteson, 2019]: (1) failure to improve the performance and the sample efficiency on finite horizon environments (section 3.1); (2) "the dominant skill problem" [Zhang and Whiteson, 2019] (section 3.1.2). In Section 3.4 we conceptually discuss that SA-style wide (higher-order dependencies) value functions could be a solution to both limitations. This is mainly because these limitations are caused by the insufficiency of the conventional value functions in approximating values that have temporal latent variables dependencies (discussed in Section 3.4).

3.4 Discussion: Learning Skills at Multi-levels of Granularity

Implementations of the option framework share some common limitations. When proposing the option framework, Sutton et al. [1999] expected that learning at multi-level of temporal abstraction should be in favor of faster convergence and better exploration. On the contrary, significant improvements on single task environments have not been witnessed in most option implementations [Klissarov et al., 2017, Smith et al., 2018, Harb et al., 2018, Zhang and Whiteson, 2019]. To the best of our knowledge, SA is the first option implementation in which these properties are significantly witnessed but only on infinite horizon environments. In this section, we address this problem by first giving a theoretical explanation of why the value function is the main reason

for this deficiency in section 3.4.1 and how **deep wide value functions** could solve this problem. We then thoroughly explain the motivations of SA, and why it is a promising candidate for a deep wide framework, in section 3.4.2. We also give a further explanation of how SA is connected to causality reinforcement learning literature and how a temporal causal reward can be used in objective to further solve this problem in section 3.4.3.

3.4.1 Problem Statement and Evidences

The expectation of improvements of the option framework on single task environment builds on an assumption that, by exploiting hierarchical action and state space, an agent's searching space can be greatly reduced thus accelerates learning and improving exploration. However, as reported in section 3.1.2, most option frameworks including SA suffer from "the dominant skill problem" [Zhang and Whiteson, 2019] which prevents option frameworks from effectively learning hierarchy in action and state space as well as coordinating between skills.

One root reason for this problem is that conventional value functions $V[S_t]$ and $Q[S_t, O_t, A_t]$ make values depend on temporal latent variables indistinguishable (i.e. Although different skills o_1 and o_2 results to different values, such as $V[S_t, O_{t-1} = o_1] = 10$ and $V[S_t, O_{t-1} = o_2] = -10$. Because they arrive at the same state S_t, they have identical values under conventional value function $V[S_t] = 0$). This deficiency makes option frameworks can only learn skills at very coarse level thus fail to exploit hierarchical information. The solution is to use a **deep wide value function**: enabling the framework to learn fine-grained skills at mutli-levels of granularity (deep) and making value functions depend on latent variables with longer (wide) dependencies (e.g. $V[S_t, O_{t-1}]$ and $Q[S_t, O_t, A_t, O_{t-1}]$).

To have a better understanding the importance of the deep wide value function, let us consider a simple environment which can be easily solved by $Q[s_t, a_t, a_{t-1}]$ but not $Q[s_t, a_t]$.

Suppose we are training a robot which only has a camera sensor to cook thanksgiving turkey. In this setting there are only two states:

$$\mathbb{S} = \{\text{Raw Turkey Image}, \text{Cooked Turkey Image}\}$$

The robot's action space only consists of two actions:

$$\mathbb{A} = \{\text{Stuff turkey}, \text{Roast turkey}\}$$

As for reward, if the robot roasted a stuffed turkey, then the reward is 10. However, if the robot roasted an un-stuffed turkey, then the reward is -10. The stuff turkey action receives 0 reward.

The difficulty in this environment is, since the robot only has a camera to capture an image of the turkey, it can only observes either {Raw Turkey Image} or {Cooked Turkey Image}. There is no way to look inside the turkey and see if the turkey is stuffed. Under this setting, a robot can never learn to first stuff a turkey and then roast it because $Q[\text{Raw Turkey Image}, \text{Stuff Turkey}] = Q[\text{Raw Turkey Image}, \text{Roast Turkey}] = 0$. Therefore, the robot can only randomly cook a turkey. However, this problem can be easily solved by using a deep wide value function $Q[S_t, A_t, A_{t-1}]$.

The core problem in this setting is, action has no effect on states, it only affects rewards. At the first glance this is a Partially Observed MDP (POMDP) problem since the state of whether the turkey is stuffed is un-observed. This is true in all reinforcement learning settings without dependencies on latent variables. However, it goes much deeper in HRL settings.

In HRL, a common formulation is to estimate a latent variable O to encode hierarchical information and makes the policy depends on it $P(A_t|S_t, O_t)$. Since O is a latent variable, it is highly likely that at state S_t, different latent variable $P(A_t|S_t, O_t = o_x)$ and $P(A_t|S_t, O_t = o_y)$ emits the same action $A_t = A_1$, and thus makes the conventional value function indistinguishable between o_x and o_y.

This phenomenon is especially common around the switching time step of two skills: around switching point, states are usually compatible with both old and new skills. Conventional value functions will be especially confused at those moments. This is exactly what we observed in Section 3.1.2: overall, skill 2 is executed consistently. However, there are some random switches to skill 1. And the randomization is increased between around switching time steps. To explicitly show this, we visualized "Run4" into a video [5]. The skill selection is very random at the beginning of the episode as well as around the switching

[5] https://www.youtube.com/watch?v=QiLVZvI6NJU

point (the 16th second). These are exactly the most confusing moments of conventional value functions. This is not a cherry-pick result but a common problem. Similar patterns can also be observed here https://youtu.be/xrfxbI3duBM?t=4 in a HumanoidStandUp environment.

Due to the insufficiency of conventional value functions, compatible states have to be different enough to cause distinguishable values of value functions. Therefore, with conventional value functions, SA is only able to learn very coarse skills. For example, as shown in Section 3.1.3 and the video, the HalfCheetah agent is only able learn two skills: one is to run forward, one is to stand up when fall. However it is not able to learn more fine-grained skills such as jump forward and landing. This problem is not limited to SA, but is a common problem in HRL. The solution is to use deep wide value functions.

3.4.2 Motivations behind SA's Architecture

SA is carefully designed to make the most out of deep wide value functions. Compared to other HRL frameworks, SA has following advantages:

1. Stable and unbiased estimation: Thanks to proposition 2.3.1 and 2.3.2, the higher the order of the MDPs, the smaller the variance will be. The deep wide value functions stays unbiased estimations of conventional value functions no matter how many dependencies introduced. The current solution in option framework is a biased estimation [Harb et al., 2018] and adding hyper-parameters to the framework.

2. Easy to incorporate wide value functions: Incorporating a deep wide value function is straightforward, SA's **skill value upon arrival function** is already a wide function. The **skill value function** and the **skill-action value function** can be easily extended to wide function by adding a first-order dependency on $\hat{O}_{t-1}$.

3. Easy to incorporate deep value functions: SA is MDP formulated, extending SA to multiple hierarchies is straightforward.

4. Scalability to long time dependencies: SA is MDP formulated, adding more time dependencies is simply to change the 1st-order MDP to higher-order MDPs while both value functions and gradient theorems stay un-

changed; SA is attention based, SA can easily attends to thousand time steps without adding any extra complexity to neither skill policy nor action policy.

5. Scalability to multiple hierarchies of skills: SA is attention based and embedding based. Adding skills is as simple as adding skill context matrix. In traditional option frameworks [Riemer et al., 2018], the number of option (note that each option is a neural network) grows at $O(N^L)$ complexity of levels (N is the number of options and L is the number of levels).

6. Interpretability. As shown in section 3.1.3, skill context vectors learned under SA-based architectures are straightforward to visualize and interpret. This property is especially useful for investigating multi-level granularity skills.

3.4.3 Causality Discovery Rewards

Although theoretically a DWSA can learn multi-level granularity skills, on-policy optimization algorithm is often insufficient for learning such models especially in sparse reward environments. However, SA has a natural connection with causal reinforcement learning thus can exploits causality as a reward in objective function to further facilitate fine grained skill discovery. In this section we explain how skill embedding vectors learned by SA encodes temporal causality relationships and how to use them to devise causal rewards.

In causal reinforcement learning area, Doshi-Velez and Konidaris [2016] proposed Hidden Parameters MDP (Hi-MDP) in which a skill vector like hidden parameter vector is introduced to learn abstract properties from environments. PEARL [Rakelly et al., 2019] utilizes meta-learning framework to learn a skill representation that encodes abstract properties of a task and updates the framework in an off-policy manner to improve sample efficiency in transfer learning. Killian et al. [2017] extended Hi-MDP by including the hidden parameter vector into transition probability function. Perez et al. [2020] further extended their work by proposing Generalized Hidden Parameter MDPs (GHP-MDPs), a causality discovery framework by including hidden parameter vector into both transition function and value function.

GHP-MDPs is a special case of SA with number of skills $M = 1$. When $M > 1$, SA not only encodes causality relationships between environments and actions but also temporal causality between skills. Since the latent variable is modeled as a skill vector, the distance between different trajectories is straightforward to be calculated and thus can be used as a causal reward to encourage fine-grained and disentangled skills' discovery. To the best of our knowledge, SA is the first RL framework concerns causality in temporal abstraction sequences. We focus this paper on proposing SA, the causality rewarded SA will be discussed in future works.

Another interesting understanding of SA is that, rather than an implementation of the option framework, SA can also be seen as a novel capsule network Kosiorek et al. [2019] trained by policy gradient theorems. In Stacked Capsule Auto-Encoders (SCAE) [Kosiorek et al., 2019], a "capsule" vector encodes a different property (scale, orientation, etc.) of the visual object in each dimension. Kosiorek et al. [2019] proposed to delegate the complexity of part objects detection and part-to-whole objects aggregation by employing the attention mechanism [Lee et al., 2019] on which a generative model is then built to further decode the whole-part relationships. This design choice abstracts the complexity of inference away from the decoder and largely simplified the designation of the generative model.

In this paper, we follow their motivations of learning better representations and utilizing the attention mechanism to simplify the inference problem (sampling new skill without termination function). Moreover, the skill context vector is analogously to a capsule and the skill-action relationship is analogously to the whole-part relationship in the SCAE. Similar to SCAE utilizing the equi-variance property of the whole-part relationship to achieve computing efficiency and better performance, it will be very exciting to investigate potentially "equi-variance" or "invariance" properties existed in skill-action relationships, which might give rise to a novel causal inference architecture in the reinforcement learning area.

3.5 Implementation Details

In this section we summarize our implementation details. For a fair comparison, all baselines: DAC+PPO [Zhang and Whiteson, 2019], AHP+PPO [Levy and Shimkin, 2011], PPOC [Klissarov et al., 2017], OC [Bacon et al., 2017] and PPO [Schulman et al., 2017] are from DAC's open source Github repo: https://github.com/ShangtongZhang/DeepRL/tree/DAC. Hyper-parameters used in DAC [Zhang and Whiteson, 2019] for all these baselines are kept unchanged.

SA Architecture: For all experiments, our implementation of SA is exactly the same as Figure 2.3 (b). We use Pytorch to build neural networks. Specifically, for skill policy module, we use a skill context matrix $\boldsymbol{W}_S \in \mathbb{R}^{4\times 40}$ which has 4 skills (4 rows) and an embedding size of 40 (40 columns). For Multi-Head Attention, we use Pytorch's built-in MultiheadAttention function[6] with $num\_heads = 1$ and $embed\_dim = 40$. For layer normalization we use Pytorch's built-in function LayerNorm [7]. For Feed Forward Networks (FNN), we use a 2 layer FNN with ReLu function as activation function with input size of 40, hidden size of 64, and output size of 64 neurons. For Linear layer, we use built-in Linear function[8] to map FFN's outputs to 4 dimension. Each dimension acts like a logit for each skill and is used as density in Categorical distribution[9]. For both action policy and critic module, FFNs are of the same size as the one used in the skill policy.

Preprocessing: States are normalized by a running estimation of mean and std.

Hyperparameters of PPO: For a fair comparison, we use exactly the same parameters of PPO as DAC. Specifically:

- Optimizer: Adam with $\epsilon = 10^{-5}$ and an initial learning rate 3×10^{-4}
- Discount ratio γ: 0.99
- GAE coefficient: 0.95
- Gradient clip by norm: 0.5

[6] https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html
[7] https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html
[8] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
[9] https://github.com/pytorch/pytorch/blob/master/torch/distributions/categorical.py

- Rollout length: 2048 environment steps
- Optimization epochs: 10
- Optimization batch size: 64
- Action probability ratio clip: 0.2

Computing Infrastructure: We conducted our experiments on an Intel® Core™ i9-9900X CPU @ 3.50GHz with a single thread and process with PyTorch.

Part II

Undirected Probabilistic Graphical Model

Chapter 4

Modeling Higer-order Structural Dependencies with Markov Random Fields (MRFs)

One challenging task in machine learning is recognizing and labeling over complex and structured objects. Many applications, such as image segmentation, motif finding and noun-phrase parsing, involves representing jointly correlated sub-objects and exploiting structural dependencies to identify higher level objects. These objects usually have structural dependencies on sub-objects (smaller components), e.g., the human face has a strong structural dependency on five sensory organs. The Markov Random Fields (MRFs, are also called undirected Probabilistic Graphical Models [Bishop, 2006b]), is one of the most popular framework in modeling structural dependencies. In this Chapter, we propose novel exact inference and learning algorithms, the MRF-LSSVMs (Latent Structural Support Vector Machine) framework, to exploit MRFs' capabilities in modeling higher-order (more than three entities) structural dependencies. The application of MRF-LSSVMs to capture higher-order dynamics on time series is discussed in Chapter 5.

Markov Random Fields (MRFs) are undirected Probabilistic Graphical Models (PGMs). The formulation of MRFs is simply a regularized joint probability distribution. One specialty of MRFs is that they are factorized (conditional independent) over **maximal cliques** (more details in Section 4.1.1) of random variables defined on the undirected graph [Bishop, 2006b]. In many applications, structural information, such as sub-objects to the whole object relationships and relationships between sub-objects, can be well represented in

maximal cliques. By defining each maximal clique's probability distribution and optimizing over them, MRFs provide a powerful framework for modeling complex higher-order dependencies between entities.

Utilizing MRFs usually involves three steps: 1) designing energy functions (un-normalized probability distribution) according to the actual problem, 2) solving inference problem (MAP or energy minimization), and 3) learning parameters from data set. With respect to energy functions, our work focuses on Lower Linear Envelope Potentials (LLEP), a class of higher-order potentials defined as a concave piecewise linear function over a clique of random variables. LLEP has been raising much interest in the image segmentation area, in which the raw image input is used as PGM's graph and pixels are treated as random variables. Maximal cliques of the input image are usually detected at preprocessing stage by using clustering algorithms such as superpixel [Achanta et al., 2012]. Success of LLEP on encoding consistent constraints over large subsets of pixels in image segmentation tasks has been witnessed in many literatures [Kohli et al., 2007, Nowozin and Lampert, 2011, Song et al., 2015]. In this chapter we focus on proposing a novel exact inference algorithm for LLEP and design a learning algorithm under the LLSVM framework. In Chapter 5, we will generalize the MRF-LLSVMs framework into encoding consistent constraints among time-series' entities.

In the second step, in order to solve the inference problem of LLEP, Kohli et al. [2009] proposed a method to represent a class of higher order potentials with lower (upper) linear envelope potentials. By introducing auxiliary variables [Kohli and Kumar, 2010], they reduced the linear representation to a pairwise form and proposed an approximate algorithm with standard linear programming methods. However, they only show an exact inference algorithm on at most three terms. Following their approach, Gould [2015] extended their method to a weighted lower linear envelope with arbitrary many terms solved with an efficient algorithm. They showed that the by introducing auxiliary variables into LLEP, a quadratic pseudo-Boolean form [Boros and Hammer, 2002] can be developed. This psuedo-Boolean form is submodular and can be inferred efficiently and exactly through graph-cuts like algorithms [Boykov and Jolly, 2001]. However, in order to employ Structural Support Vector Machine (SSVM) to solve the learning problem of LLEP, Gould [2015] have to sample the LLEP using a set of fixed space points. Althought this formulation

can be globally optimized by using the SSVM framework, it lost a rich class of representations of energy function due to the fixed space sampling. In this chapter, we an alternative formulation to learn LLEP exactly. We introduce auxiliary variables back to LLEP and propose a graph-cuts algorithm to infer observed variables and auxiliary variables simultaneously. Experiments in Section 4.4 also show that LLEP under this formulation the algorithm can be learned exactly from various different probability distribution configurations.

The third and last difficulty is to design a learning algorithm for our LLEP formulation. In previous work, Gould [2015] sampled the LLEP with fixed space points and solved the learning problem under the Structural Support Vector Machine (SSVM) framework [Tsochantaridis et al., 2005]. However, since we add auxiliary variables back, SSVM does not fit our case anymore. One of our main contribution is that we prove that auxiliary variables introduced in LLEP can be formulated as latent variables in SSVM. With additional convex constraints added to the SSVM object, our formulation results to the Latent SSVM (LSSVM) framework [Yu and Joachims, 2009]. The LSSVM was developed by Felzenszwalb et al. [2008] and Yu and Joachims [2009] independently in different ways. The main idea is introducing a latent variable to extend the feature vector, which results in an arbitrary loss function, e.g. Hinge Loss, with an upper bound. Then the optimization was done by using Concave-Convex Procedure (CCCP) algorithm, which is guaranteed to decrease the objective function to a local minimum. In this thesis, we propose a variant formulation of [Gould, 2015] by rewriting the lower linear envelope function directly into a linear combination with latent feature vectors and developing the learning algorithm using the LSSVM.

The rest of the thesis is structured as follows: Section 4.1 introduces background and related works of MRFs and LSSVM. Section 4.2 proposes our first contribution, the exact inference method of LLEP with auxiliary variables. In Section 4.3.1, we reformulate the LLEP into a linear combination and develop the learning algorithm under the LSSVM framework. Section 4.4 conduct experiments on a synthetic checkerboard image to show the effectiveness of our novel MRF-LLSVMs framework. Application of MRF-LLSVMs on real financial time-series data set is discussed in Chapter 5.

4.1 Background & Related Works

4.1.1 Markov Random Fields

From a PGMs view (figure 4.1) [Bishop, 2007], MRFs' joint probability distribution can be represented as an undirected graph and each random variable can be represented as a node in the graph. A **clique** is a fully connected subset of nodes: there exists a path between any pair of nodes in it. A **maximal clique** is a clique such that it is not possible to include any other nodes from the graph in the set without it ceasing to be a clique. Let C denotes a maximal

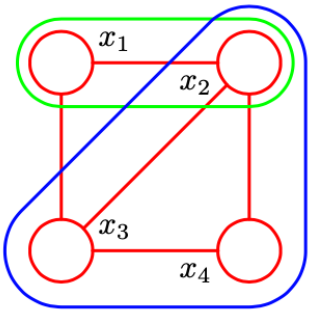


Figure 4.1: An example PGM of MRFs. Each node in this graph corresponds to a random variable in this PGM's joint probability distribution. A clique is outlined in green circle and a maximal clique is outlined in blue circle.

clique in one graph and $\boldsymbol{y}_C$ denotes the set of variables in that clique. Then the joint distribution can be written as:

$$p(\boldsymbol{y}) = \frac{1}{Z} \prod_C \Psi_C(\boldsymbol{y}_C) \tag{4.1}$$

where Ψ is called *potential functions* which can be defined as any non-negative functions and $Z = \sum_{\boldsymbol{y}} \prod_C \Psi_C(\boldsymbol{y}_C)$ which is a normalization constant. To infer labels which best explains input data set, we can find the *maximum a posteriori* (MAP) labels by solving $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y})$. However, potential functions are restricted to be non-negative to ensure it is a probability distribution.

In order to have more flexible representations of probability distributions, by taking exponential of potential terms, MRFs can be represented as a regularized joint log-probability distribution of arbitrary non-negative functions over a set of maximal cliques on the PGM graph [Bishop, 2006b]. Thus the

joint distribution becomes:

$$p(\boldsymbol{y}) = \frac{1}{Z} exp(-\sum_{C} E_C(\boldsymbol{y}_C)) \tag{4.2}$$

where E is called *energy functions* which can be arbitrary functions. Therefore, *maximum a posteriori* problem is equivalent to *energy minimization* problem, which is also known as the *inference* problem:

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}) = \arg\min_{\boldsymbol{y}} (\sum_{C} E_C(\boldsymbol{y}_C)) \tag{4.3}$$

The *inference* problem is computationally expensive. There has been many sub-optimal algorithms such as max-product algorithm [Globerson and Jaakkola, 2007] been proposed to solve the general MRFs' inference problem. However, Boros and Hammer [2002] proved that for submodular energy functions, there exist efficient algorithms based on graph cuts [Ishikawa, 2009, Kohli et al., 2008] and are guaranteed to converge to the global optimum. In Section 4.2 we formulate the MRFs with Lower Linear Envelope Potentials (LLEP) as psuedo-boolean functions and devise a graph cut algorithm for exact inference.

As for the *learning* problem, conventionally, energy functions can be decomposed into three weighted parts: nodes $\mathcal{N}$, edges $\mathcal{E}$ and higher order cliques (any group which has more than 3 fully connected nodes in it) $\mathcal{C}$ [Szummer et al., 2008]. Each term has its own weights. Let $\boldsymbol{w}$ be the vector of parameters and ϕ be arbitrary feature function, then the energy can be decomposed as a set linear combinations of weights and feature vectors:

$$E(\boldsymbol{y}; \boldsymbol{w}) = \sum_{i \in \mathcal{N}} \boldsymbol{w}_i^U \phi^U(\boldsymbol{y}_i) + \sum_{(i,j) \in \mathcal{E}} \boldsymbol{w}_{ij}^P \phi^P(\boldsymbol{y}_i, \boldsymbol{y}_j) + \sum_{\boldsymbol{y}_C \in \mathcal{C}} \boldsymbol{w}_C^H \phi^H(\boldsymbol{y}_C) \tag{4.4}$$

where U denotes *unary* terms, P denotes *pairwise* terms and H denotes *higher order* terms (when $|C| > 2$ namely each clique contains more than two variables).

A weight vector $\boldsymbol{w}$ is more preferable if it gives the ground-truth assignments $\boldsymbol{y}_t$ less than or equal to energy value than any other assignments y:

$$E(y_t, w) \leq E(y, w) \, , \forall y \neq y_t, y \in \mathbb{Y} \tag{4.5}$$

Thus the goal of *learning* MRFs is to learn the parameter vector $\boldsymbol{w}^*$ which returns the lowest energy value for the ground-truth labels y_t relative to any other assignments y [Szummer et al., 2008]:

$$\boldsymbol{w}^* = argmax_{\boldsymbol{w}}(E(y_t, w) - E(y, w)) \, , \forall y \neq y_t, y \in \mathbb{Y} \tag{4.6}$$

Solving the learning problem of MRFs is also computationally expensive. In this thesis, we employ the efficient Latent Structural Support Vector Machines (LSSVMs) algorithm to solve our MRFs.

4.1.2 Latent Structural SVMs

The Structural Support Vector Machines (SSVMs) (also called max-margin framework) [Taskar et al., 2005, Tsochantaridis et al., 2005] is a principled approach to learn weights of pairwise MRFs Szummer et al. [2008], Gould [2011]. Gould [2015] extended this framework with additional linear constraints to enforce concavity on the weights, thus allowing them to be used to learn MRFs with lower linear envelope potentials. However, because SSVM does not include latent variables in its feature vector, such methods only approximately learn higher-order functions. In this thesis, we propose an algorithm to optimize the energy function exactly by introducing auxiliary variables back into the feature vector and solving the learning problem using the Latent Structural SVMs (LSSVMs) framework [Yu and Joachims, 2009]. To include unobserved information, Yu and Joachims [2009] extended the joint feature function in structural SVM with latent variables and re-wrote the objective function of SSVM into a difference of two convex functions. This formulation can be solved using the Concave-Convex Procedure (CCCP)[Yuille et al., 2002] which is two-stages algorithm that guarantee to convergence to a local minimum.

Specifically, given an a linear combination of features vector $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^m$ and weights $\boldsymbol{\theta} \in \mathbb{R}^m$, and a set of n training examples $\{\boldsymbol{y}_i\}_{i=1}^n$ max-margin framework can be used to solve optimized solution $\boldsymbol{\theta}^*$. To include unobserved

information in the model, Yu[Yu and Joachims, 2009] extended the joint feature function[Tsochantaridis et al., 2005] $\phi(\boldsymbol{x}, \boldsymbol{y})$ with a latent variable $\boldsymbol{h} \in \mathcal{H}$ to $\phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h})$. So the inference problem becomes

$$f_\theta(x) = \underset{(\boldsymbol{y} \times \boldsymbol{h}) \in \mathbb{Y} \times \mathbb{H}}{\arg\max} \; \boldsymbol{\theta} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}) \tag{4.7}$$

Accordingly, the loss function can be extended as

$$\Delta((\boldsymbol{y}_i, \boldsymbol{h}_i^*(\theta)), (\hat{\boldsymbol{y}}_i(\boldsymbol{\theta}), \hat{\boldsymbol{h}}_i(\boldsymbol{\theta})))$$

where

$$(\hat{\boldsymbol{y}}_i(\boldsymbol{\theta}), \hat{\boldsymbol{h}}_i(\boldsymbol{\theta})) = \underset{(\boldsymbol{y} \times \boldsymbol{h}) \in \mathcal{Y} \times \mathcal{H}}{\arg\max} \; \theta \cdot \phi(\boldsymbol{x}_i, \boldsymbol{y}, \boldsymbol{h}) \tag{4.8}$$

$$\boldsymbol{h}_i^*(\theta) = \underset{\boldsymbol{h} \in \mathcal{H}}{\arg\max}\, \theta \cdot \phi(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{h}) \tag{4.9}$$

The loss function under this formulation measures difference between the inferred result pair $(\hat{\boldsymbol{y}}_i(\theta), \hat{\boldsymbol{h}}_i(\theta))$ and the pair $(\boldsymbol{y}_i(\theta), \boldsymbol{h}_i^*(\theta))$ which best explains the training data. However, under this formulation the "loss augmented inference" used in structural SVMs[Tsochantaridis et al., 2005] to remove the complexity cannot be performed due to the dependence of loss function Δ on hidden variables $\boldsymbol{h}_i^*(\theta)$. Yu and Joachims [2009] argued that in real world applications hidden variables are usually intermediate results and are not required as an output[Yu and Joachims, 2009]. Therefore, the loss function can only focus on the inferenced hidden variables $\hat{\boldsymbol{h}}_i(\theta)$ which leads to:

$$\Delta((\boldsymbol{y}_i, \boldsymbol{h}_i^*(\theta)), (\hat{\boldsymbol{y}}_i(\theta), \hat{\boldsymbol{h}}_i(\theta))) = \Delta(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i(\theta), \hat{\boldsymbol{h}}_i(\theta))$$

Thus the upper bound used in standard structural SVMs[Tsochantaridis et al., 2005] can be extended to:

$$\Delta((\boldsymbol{y}_i, \boldsymbol{h}_i^*(\theta)), (\hat{\boldsymbol{y}}_i(\theta), \hat{\boldsymbol{h}}_i(\theta))) \leq \left(\max_{(\hat{\boldsymbol{y}} \times \hat{\boldsymbol{h}}) \in \mathcal{Y} \times \mathcal{H}} [\theta \cdot \Psi(\boldsymbol{x}_i, \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}}) + \Delta(\boldsymbol{y}_i, \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}})] \right) - \max_{\boldsymbol{h} \in \mathcal{H}} \theta \cdot \Psi(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{h}) \tag{4.10}$$

Hence the optimization problem for Structural SVMs with latent variables becomes

$$\min_{\theta} \left(\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^{n} \left(\max_{(\hat{\boldsymbol{y}} \times \hat{\boldsymbol{h}}) \in \mathcal{Y} \times \mathcal{H}} [\theta \cdot \Psi(\boldsymbol{x}_i, \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}}) + \Delta(\boldsymbol{y}_i, \hat{\boldsymbol{y}}, \hat{\boldsymbol{h}})] \right) \right) - C \sum_{i=1}^{n} \left(\max_{\boldsymbol{h} \in \mathcal{H}} \theta \cdot \Psi(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{h}) \right) \tag{4.11}$$

which is a difference of two convex functions. Problem of this formulation can be solved using the Concave-Convex Procedure (CCCP)[Yuille et al., 2002] which is guaranteed to converge to a local minimum. Yu and Joachims [2009] proposed a two stages algorithm. In the first step the latent variable $\boldsymbol{h}_i^*$ which best explains training pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is found by solving equation equation 4.9. This step is also called the "latent variable completion" problem. In the second step $\boldsymbol{h}_i^*$ is used as completely observed to substitute $\boldsymbol{h}$ in equation equation 4.11. Therefore, solving equation equation 4.11 is equivalent to solve the standard structural SVM problem.

In contrast to SVM, the latent structural SVM only provides an optimization framework and cannot be directly applied. In order to use it, the inference algorithm, as well as the MRF feature function, loss function, and latent variable completion problem [Yu and Joachims, 2009] must first be specified. Our implementation of these terms are described in section 4.3.

4.2 Markov Random Fields (MRFs) with Lower Linear Envelope Potentials (LLEPs)

Energy functions can be decomposed over nodes $\mathcal{N}$, edges $\mathcal{E}$ and higher order cliques $\mathcal{C}$ [Szummer et al., 2008]. Let w be vector of parameters and ψ be

arbitrary feature function, then the energy can be decomposed as a set of linear combinations of weights and feature vectors:

$$E(\boldsymbol{y};\boldsymbol{w}) = \sum_{i\in\mathcal{N}} \boldsymbol{w}_i^U \psi^U(\boldsymbol{y}_i) + \sum_{(i,j)\in\mathcal{E}} \boldsymbol{w}_{ij}^P \psi^P(\boldsymbol{y}_i,\boldsymbol{y}_j) + \sum_{\boldsymbol{y}_C\in\mathcal{C}} \boldsymbol{w}_C^H \psi^H(\boldsymbol{y}_C) \tag{4.12}$$

where U denotes *unary* terms, P denotes *pairwise* terms, H denotes *higher order* terms. In this section we mainly focus on one class of higher-order potentials ψ^H defined as a concave piecewise linear function which is known as *Lower Linear Envelope Potentials* (LLEP).

LLEP has been studied extensively in Markov Random Fields area for encouraging consistency over large cliques [Kohli et al., 2007, Nowozin and Lampert, 2011, Gould, 2011]. In Section 4.2.1, we begin with developing standard Markov Random Fields (MRFs) (equation equation 4.12) with the LLEP as energy functions. We then show how to perform exact inference under this formulation in Section 4.2.2. The optimization algorithm for our formulation will be discussed in Section 4.3.1.

4.2.1 Higher-order Energy Functions: Weighted Lower Linear Envelope Potentials (LLEP)

Let $\mathcal{C}$ denotes the set of all maximal cliques and $\boldsymbol{y}_c = \{y_i | \text{ for } i \in C_j\}$ denotes set of binary random variables where $y_i \in \{0,1\}$ in clique C_j, a weighted lower linear envelope potential over $\boldsymbol{y}_c$ is defined as the minimum over a set of K linear functions as:

$$\psi_c^H(\boldsymbol{y}_c) = \min_{k=1,\ldots,K} \{a_k W_c(\boldsymbol{y}_c) + b_k\}. \tag{4.13}$$

where $W_c(\boldsymbol{y}_c) = \sum_{i\in c} w_i^c y_i$ with $w_i^c \geq 0$ and $\sum_{i\in c} w_i^c = 1$ which are weights for each clique. $(a_k, b_k) \in \mathbb{R}^2$ are the linear function parameters. We illustrate an example with four linear functions in figure 4.2.

Inference on energy function contains lower linear potentials is the same as

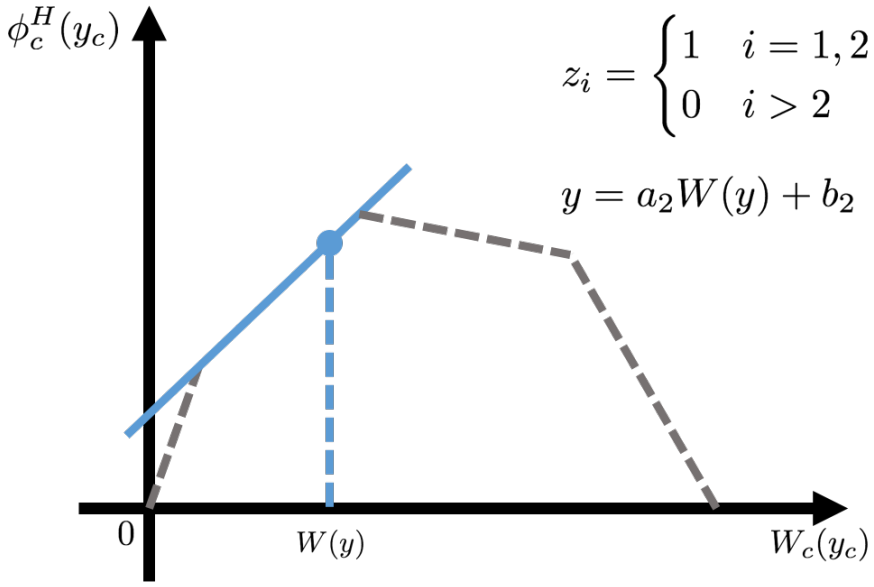


Figure 4.2: Example piecewise-linear concave function of $W_c(\boldsymbol{y}_c) = \sum_{i \in c} w_i^c y_i$. Assume the second linear function is active namely $\boldsymbol{z}^c = (1, 1, 0, 0)$ (equation 4.20). The result of linear combination of parameter vector and feature vector is same as quadratic pseudo-Boolean function.

the standard equation equation 4.12 and is given by:

$$\boldsymbol{y}^* = \arg\min E(\boldsymbol{y}) \tag{4.14}$$

Suppose that parameters $\{(a_k, b_k)\}_{k=1}^K$ are sorted in decreasing order of a_k. From *Definition 3.1* [Gould, 2015] we know that the k-th linear function is said to be *active* if there exists $x \in (0, 1)$ such that the following two inequalities hold

$$\begin{aligned} a_{k-1}x + b_{k-1} &> a_k x + b_k \\ a_{k+1}x + b_{k+1} &> a_k x + b_k \end{aligned} \tag{4.15}$$

The k-th linear function is said to be *redundant* (*Definition 3.2* [Gould, 2015]) if it is not active for any assignment to $\boldsymbol{y}_c$ in any clique $c \in \mathcal{C}$ or is only active whenever another linear function is also active. Figure 4.4 depicts such conditions. As a result, removing redundant functions from the potential does not chang the energy function.

From section 4.1.1 we have already introduced that an *energy function* may

contain *unary*, *pairwise* and *higher-order* potentials (see equation equation 4.12). In this section we mainly focus on one class of higher-order potentials ϕ^H defined as a concave piecewise linear function which is known as *lower linear envelope potentials*. This has been studied extensively in Markov Random Fields area for encouraging consistency over large cliques [Kohli et al., 2007, Nowozin and Lampert, 2011, Gould, 2011].

Let $\mathcal{C}$ denotes the set of all maximal cliques in an image and $\boldsymbol{y}_c = \{y_i | \text{ for } i \in c\}$ denotes set of random variables in the clique c, a weighted lower linear envelope potential [Gould, 2015] over $\boldsymbol{y}_c$ is defined as the minimum over a set of K linear functions as:

$$\psi_c^H(\boldsymbol{y}_c) = \min_{k=1,\ldots,K} \{a_k W_c(\boldsymbol{y}_c) + b_k\}. \tag{4.16}$$

where $W_c(\boldsymbol{y}_c) = \sum_{i \in c} w_i y_i$ with $w_i^c \geq 0$ and $\sum_{i \in c} w_i^c = 1$ which are weights for each clique. $(a_k, b_k) \in \mathbb{R}^2$ are the linear function parameters. We illustrate an example [Gould, 2015] with three linear functions in figure 4.3.

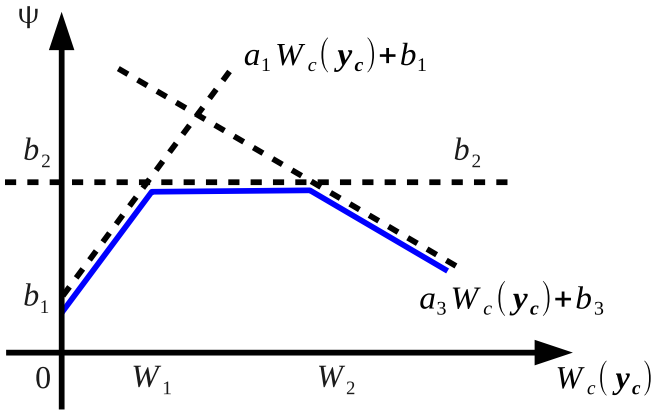


Figure 4.3: Example lower linear envelope $\psi_c^H(\boldsymbol{y}_c)$ (shown solid) with three terms (dashed). When $W_c(\boldsymbol{y}_c) \leq W_1$ the first linear function is active, when $W_1 < W_c(\boldsymbol{y}_c) \leq W_2$ the second linear function is active, otherwise the third linear function is active.

Suppose that parameters $\{(a_k, b_k)\}_{k=1}^K$ are sorted in decreasing order of a_k. Gould [2015] (Definition 3.1) defines that the k-th linear function is said to be *active* if there exists $x \in (0, 1)$ such that the following two inequalities hold

$$\begin{aligned} a_{k-1}x + b_{k-1} &> a_k x + b_k \\ a_{k+1}x + b_{k+1} &> a_k x + b_k \end{aligned} \tag{4.17}$$

The k-th linear function is said to be *redundant* (*Definition 3.2* [Gould, 2015]) if it is not active for any assignment to $\boldsymbol{y}_c$ in any clique $c \in \mathcal{C}$ or is only active whenever another linear function is also active. Figure 4.4 depicts such conditions. As a result, removing redundant functions from the potential does not chang the energy function.

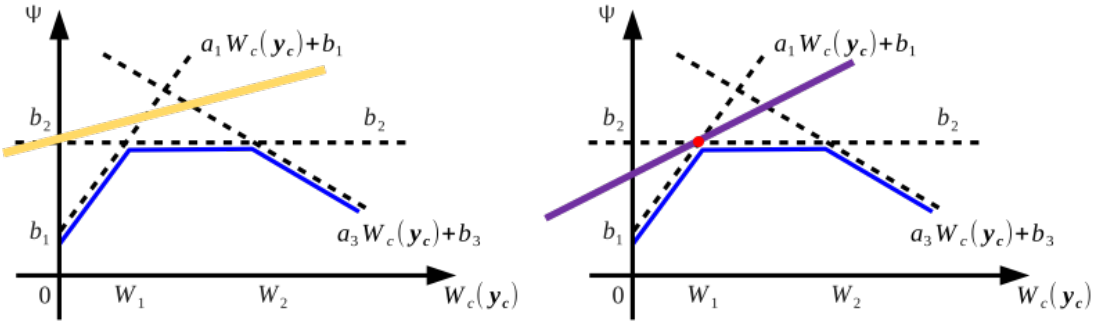


Figure 4.4: Example lower linear envelope with redundant linear functions. On the left figure, the solid yellow line is always inactive. On the right figure, the solid purple line intersects *line* 1 and *line* 2 at the red point. It's only active when *line* 1 and *line* 2 are both active. Both solid lines are redundant linear functions hence can be removed without changing their energy function.

To ensure potentials do not contain redundant linear functions (functions that would never be active), Gould [2015] proposed a constraint on parameters of the envelope. The k-th linear function is not redundant if the following condition is satisfied:

$$0 < \frac{b_k - b_{k-1}}{a_{k-1} - a_k} < \frac{b_{k+1} - b_k}{a_k - a_{k+1}} < 1. \tag{4.18}$$

Another important property of equation equation 4.14 is shift invariant (vertically). We write $\widetilde{\psi}_c^H(\boldsymbol{y}_c)$ by shift equation equation 4.16 vertically with an arbitrary amount $b^{const} \in R$

$$\widetilde{\psi}_c^H(\boldsymbol{y}_c) = \min_{k=1,\ldots,K} \left\{ a_k W_c(\boldsymbol{y}_c) + b_k + b^{\text{const}} \right\}$$

Then we have

$$\arg\min_{\boldsymbol{y}_c} \psi_c^H(\boldsymbol{y}_c) = \arg\min_{\boldsymbol{y}_c} \widetilde{\psi}_c^H(\boldsymbol{y}_c). \tag{4.19}$$

Therefore, in the following discussion without loss of generality we assume

$b_1 = 0$ thus $b_k \geq 0$ for $k = 1, \ldots, n$.

4.2.2 Exact Inference

Exact inference on MRFs has been extensively studied in past years. Researchers found that, energy functions which can be transformed into quadratic pseudo-Boolean functions [Ishikawa, 2003, 2009, Rother et al., 2009] are able to be minimized exactly using *graph-cuts* like algorithms [Freedman and Drineas, 2005, Hammer, 1965] when they satisfy submodularity condition [Boros and Hammer, 2002]. Kohli et al. [2008] and Gould [2011] adapted those results to perform exact inference on lower linear envelope potentials. In this section we mainly focus on describing the minimum *st-cut* graph constructed by Gould [Gould, 2011, 2015] for exact inference of energy function (equation equation 4.14) containing lower linear envelope potentials.

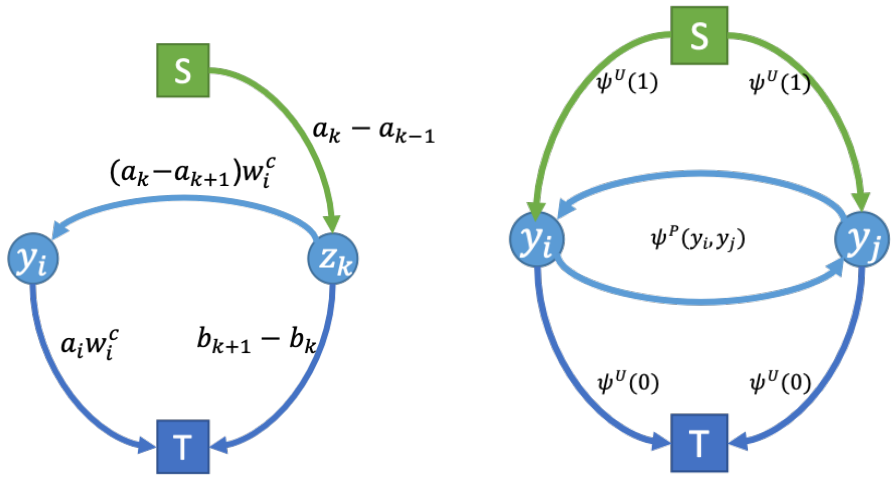
Following the approach of Kohli and Kumar [2010], Gould [2011, 2015] transformed the weighted lower linear envelope potential in equation equation 4.16 into a quadratic pseudo-Boolean function by introducing $K - 1$ auxiliary variables $\boldsymbol{z} = (z_1, \ldots, z_{K-1})$ with $z_k \in \{0, 1\}$:

$$\begin{aligned} E^c(\boldsymbol{y}_c, \boldsymbol{z}) &= a_1 W_c(\boldsymbol{y}_c) + b_1 \\ &\quad + \sum_{k=1}^{K-1} z_k \left((a_{k+1} - a_k) W_c(\boldsymbol{y}_c) + b_{k+1} - b_k \right) \end{aligned} \tag{4.20}$$

for a single clique $c \in \mathcal{C}$. Under this formulation, minimizing the pseudo-Boolean function over $\boldsymbol{z}$ is equivalent to selecting (one of) the active functions(s) from equation equation 4.16. Another important property of optimized $\boldsymbol{z}$ under this formulation is that it automatically satisfies the constraint

$$z_{k+1} \leq z_k$$

This property give rise to further development of parameter vector and feature vector (equation equation 4.24 and equation 4.25) which are used in latent structural SVM. By introducing latent variables within the energy function, we can learn richer energy representations than previous study [Gould, 2015] and solve inference problem exactly within polynomial number of iterations.



(a) (b)

Figure 4.5: *st*-graph construction for equation equation 4.21, unary and pairwise terms. Every cut corresponds to an assignment to the random variables, where variables associated with nodes in the $\mathcal{S}$ set take the value one, and those associated with nodes in the $\mathcal{T}$ set take the value zero. With slight abuse of notation, we use the variables to denote nodes in our graph.

In order to construct the minimum *st-cut* graph, we rewrite equation equation 4.20 into *posiform* [Boros and Hammer, 2002]:

$$\begin{aligned} E^c(\boldsymbol{y}_c, \boldsymbol{z}) = b_1 - (a_1 - a_K) + \sum_{i \in c} a_1 w_i^c y_i \\ + \sum_{k=1}^{K-1} (b_{k+1} - b_k) z_k + \sum_{k=1}^{K-1} (a_k - a_{k+1}) \bar{z}_k \\ + \sum_{k=1}^{K-1} \sum_{i \in c} (a_k - a_{k+1}) w_i^c \bar{y}_i z_k \end{aligned} \tag{4.21}$$

where $\bar{z}_k = 1 - z_k$ and $\bar{y}_i = 1 - y_i$. a_1 is assumed to be greater than 0 so that all coefficients are positive (recall we assume $b_1 = 0$ in section 4.2.1 and we have $a_k > a_{k+1}$ and $b_k < b_{k+1}$). Since the energy function equation 4.21 is submodular, the *st-min-cut* graph can be constructed based on equation equation 4.21.

The construction (including unary and pairwise) is explained in Figure 4.5. Figure (a) denotes construction for equation equation 4.21. For each lower linear envelope potential edges are added as follows: for each $i \in c$, add an

edge from y_i to t with weight $a_1 w_i^c$; for each $i \in c$ and $k = 1, \ldots, K-1$, add an edge from z_k to y_i with weight $(a_k - a_{k+1}) w_i^c$; and for $k = 1, \ldots, K-1$, add an edge from s to z_k with weight $a_k - a_{k+1}$ and edge from z_k to t with weight $b_{k+1} - b_k$. Figure (b) denotes construction for unary and pairwise terms (see [Kolmogorov and Zabih, 2004]). For unary edges (4 edges on both sides), weights on each edge are corresponding to values in input unary terms accordingly. For pairwise edges (2 edges in the middle), both edges share the same weight which equals to the input pairwise term.

4.3 Solving MRFs under the Latent Structrual SVMs (LSSVM) Framework

With the inference algorithm in hand, we now can develop the learning algorithm for weighted lower linear envelope potentials using the Latent Structural SVMs (LSSVMs) framework. In Section 4.3.1, we begin by transforming the equation equation 4.20 into a linear combination of parameter vector and feature vector. A two-step algorithm was developed to solve the latent structural SVM in Section 4.3.2.

4.3.1 Transforming Between Representations

The latent structural SVM formulation requires that the energy function be formulated into a linear combination of features and weights while our higher-order potential is represented as the minimum over a set of linear functions. However, in 4.2.2 we reformulated the piesewise linear functions into a quadratic pseudo-Boolean function in equation equation 4.20 by introducing auxiliary variables. Now we show equation equation 4.20 itself is an inner product of parameter vector and feature vector with latent information. Note that the function can be expanded as a summation of $2K - 1$ terms:

$$E^c(y_c, z) = a_1 W_c(y_c) + \sum_{k=1}^{K-1} (a_{k+1} - a_k) z_k W_c(y_c) + \sum_{k=1}^{K-1} (b_{k+1} - b_k) z_k \tag{4.22}$$

Here we use the fact of equation equation 4.19 and let $b_1 = 0$. Now we can reparameterize the energy function as

$$E^c(\boldsymbol{y}_c, \boldsymbol{z}; \boldsymbol{\theta}^H) = \boldsymbol{\theta}^{H^T} \psi^H(\boldsymbol{y}_c, \boldsymbol{z}) \tag{4.23}$$

where:

$$\theta_k^H = \begin{cases} a_1 & \text{for } k = 1 \\ a_k - a_{k-1} & \text{for } 1 < k \leq K \\ b_{k+1-K} - b_{k-K} & \text{for } K < k \leq 2K - 1 \end{cases} \tag{4.24}$$

$$\psi_k^H = \begin{cases} W_c(\boldsymbol{y}_c) & \text{for } k = 1 \\ W_c(\boldsymbol{y}_c) z_k & \text{for } 1 < k \leq K \\ z_k & \text{for } K < k \leq 2K - 1 \end{cases} \tag{4.25}$$

Under this formulation, similar to [Yu and Joachims, 2009], the inference problem can be given by:

$$(\mathbf{y}_k^*(\boldsymbol{\theta}^H), \mathbf{z}_k^*(\theta^H)) = \underset{(\mathbf{y} \times \mathbf{z}) \in \mathcal{Y} \times \mathcal{Z}}{\arg\min} \; \boldsymbol{\theta}^{H^T} \cdot \psi^H(\mathbf{y}_k, \mathbf{z}_k) \tag{4.26}$$

and

$$\mathbf{z}_k^*(\boldsymbol{\theta}) = \underset{\mathbf{z} \in \mathcal{Z}}{\arg\min} \, \boldsymbol{\theta}^{H^T} \cdot \psi^H(\mathbf{y}_k, \mathbf{z}_k) \tag{4.27}$$

There are two facts worth to mention. The first fact is that in our previous construction of minimum $st - cut$ graph the latent variable z is already included. Therefore, we can apply our inference algorithm directly on our two new formulations. The second fact is that for equation equation 4.27, there ex-

ists a more efficient algorithm. At training stage, the ground-truth labels y_i is an input and is completely observed. Therefore, the term $((a_{k+1} - a_k)W_c(\boldsymbol{y}_c) + b_{k+1} - b_k)$ in equation equation 4.22 becomes constant. So we can infer latent variable $\boldsymbol{z}$ explicitly by:

$$z_k^c = \begin{cases} 0 & \text{if } ((a_{k+1} - a_k)W_c(y_c) + b_{k+1} - b_k) \geq 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.28}$$

To show the equivalence between equation equation 4.20 and equation equation 4.23 we consider the example illustrated in figure 4.2. Assume the inferred latent vector $\boldsymbol{z}^c = (1, 1, 0, 0)$. Plug it into equation equation 4.25 the energy function can be written as:

$$\begin{aligned} E^c(\boldsymbol{y}_c, \boldsymbol{z}; \boldsymbol{\theta}) &= \begin{bmatrix} a_1 \\ a_2 - a_1 \\ a_3 - a_2 \\ a_4 - a_3 \\ b_2 \\ b_3 - b_2 \\ b_4 - b_3 \end{bmatrix}^T \begin{bmatrix} W_c(\boldsymbol{y}_c) \\ W_c(\boldsymbol{y}_c) \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\ &= a_1 W_c(\boldsymbol{y}_c) + (a_2 - a_1)W_c(\boldsymbol{y}_c) + b_2 \\ &= a_2 W_c(\boldsymbol{y}_c) + b_2 \end{aligned}$$

Therefore, assignments inferred by graph-cut algorithm can be directly encoded into a linear combination by using our latent structural SVM formulation for learning purpose. The remaining task is to ensure the concavity of $\boldsymbol{\theta}$. We do this by adding following constraint:

$$A\boldsymbol{\theta} \geq \epsilon, \quad A = \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P} \end{bmatrix} \in \mathbb{R}^{(2K-1)\times(2K-1)} \tag{4.29}$$

where $-\mathbf{1}$ is a matrix of size $(K-1) \times (K-1)$ and $\mathbf{P}$ is an identity matrix of size $(K-1) \times (K-1)$. One subtle problem we found during experiments is that the algorithm can be stuck with small numerical value. To avoid this we

add small slack variables $\epsilon = \mathbf{1}^{-15}$ on those constraints.

$$A\theta \geq \epsilon, \quad A = \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P} \end{bmatrix} \in \mathbb{R}^{(2K-1)\times(2K-1)} \tag{4.30}$$

4.3.2 Latent Structural SVM Learning

With the inner product formulation (equation equation 4.23) of higher order energy function, we are able to derive our latent structural SVM learning algorithm. The energy function (higher order function together with unary and pairwise functions) can be written as:

$$E_{all}(y,z) = \begin{bmatrix} \theta^H \\ \theta^{unary} \\ \theta^{pairwise} \end{bmatrix}^T \cdot \begin{bmatrix} \psi^H \\ \psi^{unary} \\ \psi^{pairwise} \end{bmatrix} = \theta_{all}^T \cdot \psi_{all} \tag{4.31}$$

where $\theta^H \in \mathbb{R}^{2K-1}$ is the parameter vector in higher order equation equation 4.23 of size $2K-1$. θ^{unary} and $\theta^{pairwise}$ are both scalars. $\psi^{\text{unary}} = \sum_i \psi_i^U(y_i)$ and $\psi^{\text{pairwise}} = \sum_{ij} \psi_{ij}^P(y_i, y_j)$. Therefore, the size of θ_{all} is $2K+1$.

Plug equation equation 4.26 and equation equation 4.27 into object function in [Yu and Joachims, 2009], the latent structural SVM object function for our problem can be derived as a difference of two convex functions:

$$\min_\theta \left(\frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{n} \left(\max_{(\hat{\mathbf{y}}\times\hat{\mathbf{z}})\in\mathcal{Y}\times\mathcal{Z}} [\theta\cdot\psi(\hat{\mathbf{y}},\hat{\mathbf{z}}) + \Delta(\mathbf{y}_i,\hat{\mathbf{y}},\hat{\mathbf{z}})] \right) \right) \tag{4.32}$$
$$-C\sum_{i=1}^{n} \left(\max_{\mathbf{z}\in\mathcal{Z}} \theta\cdot\psi(\mathbf{y}_i,\mathbf{z}) \right)$$

Following Yu and Joachims [2009], we use the two stages Concave-Convex Procedure (CCCP) [Yuille et al., 2002] to solve the optimization problem. We first imputes the latent variables z explicitly by equation equation 4.27. Namely solving the "latent variable completion" problem [Yu and Joachims, 2009]:

$$z_i^* = \arg\max_{\mathbf{z}\in\mathcal{Z}} \theta \cdot \psi(\mathbf{y}_i, \mathbf{z}) \tag{4.33}$$

The inference result z_i^* for $i = 1, \ldots, n$ is used as completely observed for later stage. With the latent variable z_i^* which best explains the ground-truth data y_i in hand, updating the parameter vector θ reduces to solve the standard structural SVM problem:

$$\min_{\theta} \left(\frac{1}{2}\|\theta\|^2 + C \sum_{i=1}^{n} \left(\max_{(\hat{\mathbf{y}}\times\hat{\mathbf{z}})\in\mathcal{Y}\times\mathcal{Z}} [\theta \cdot \psi(\hat{\mathbf{y}}, \hat{\mathbf{z}}) + \Delta(\mathbf{y}_i, \hat{\mathbf{y}}, \hat{\mathbf{z}})] \right) \right) \tag{4.34}$$
$$-C \sum_{i=1}^{n} \left(\theta \cdot \psi(\mathbf{y}_i, \mathbf{z}_i^*) \right)$$

The last problem remaining is the initialization method. Because our objective function equation 4.34 is not convex and the CCCP algorithm is only guaranteed to converge to a local minimum or saddle point[Yuille et al., 2002], initialization of θ might affect the performance of our algorithm. Since there are no theoretical solution for this problem, we propose an empirical initialization algorithm in Algorithm 5.

Algorithm 2 Empirical initialization algorithm for θ

1: $gap = \frac{1}{K}$, $a_1 = \mathbb{U}(0, 1e6)$, $b_1 = 0$, $sp_1 = (0, 0)$, $w_0 = 0$, $counter = 2$
2: **for** each clique $c \in \mathcal{C}$ **do**
3: Compute weighted clique value $w_c = W_c(y_C)$
4: **if** $w_c - w_{c-1} > gap$ **then**
5: $upbound = a_{counter} w_c + b_{counter}$
$sp_{counter} = (w_c, \mathbb{U}(upbound - 0.5, upbound))$
Calculate $a_{counter}$ and $b_{counter}$ using $sp_{counter-1}$ and $sp_{counter}$
$counter = counter + 1$
6: **end if**
7: **end for**
8: If $counter < K$, remaining as and bs are all set to be $a_{counter}$ and $b_{counter}$
9: Calculate θ using $\{a_k, b_k\}_{k=1}^{K}$

We assume that the more evenly distributed of $W_c(Y_c)$ where $c \in \mathcal{C}$ on x axis, the more rich representation (number of linear functions) the energy func-

tion should have. In order to initialize θ, we first determine the x-coordinate of sampled points sp. Then we sample its y-coordinate from a uniform distribution $\mathbb{U}(upbound, upbound - 0.5)$ to add some randomness in our initialization as well as maintain concavity. Linear parameters a_k and b_k are later calculated using those sampled points sp_k and sp_{k-1}. At last we encode $\{a_k, b_k\}_{k=1}^{K}$ into θ using equation equation 4.24.

Our optimization algorithm is summarized in algorithm 3.

Algorithm 3 Learning lower linear envelope MRFs with latent variables.

Set $MaxIter = 100$
input training set $\{\boldsymbol{y}_i\}_{i=1}^{n}$, regularization constant $C > 0$, and tolerance $\epsilon \geq 0$
Initialize θ using algorithm 5
repeat
  Set $iter = 0$
  for each training example, $i = 1, \ldots, n$ **do**
    compute $\mathbf{z}_i^* = \arg\max_{\mathbf{z} \in \mathcal{Z}} \theta \cdot \phi(\mathbf{y}_i, \mathbf{z})$
  end for
  initialize active constraints set $\mathcal{C}_i = \{\}$ for all i
  repeat
    solve the quadratic programming problem in equation 4.34 with respect to active constraints set $\mathcal{C}_i$ for all i and concavity constraints $A\theta \geq \epsilon$ to get $\hat{\theta}$ and $\hat{\boldsymbol{x}}_i$
    for each training example, $i = 1, \ldots, n$ **do**
      compute $\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i = \arg\min_{\boldsymbol{y}} E(\boldsymbol{y}, \boldsymbol{z}; \hat{\theta}) - \Delta(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{y}_i)$
      if $\hat{\xi}_i + \epsilon < \Delta(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i, \boldsymbol{y}_i) - E(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i; \hat{\theta}) + E(\boldsymbol{y}_i, \boldsymbol{z}_i^*; \hat{\theta})$ **then**
        $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\boldsymbol{y}_i^\star\}$
      end if
    end for
  until no more violated constraints
  return parameters $\hat{\theta}$
  Set $iter = iter + 1$
until $iter \geq MaxIter$
return parameters $\hat{\theta}$

4.4 Experiments

Since the main contribution of our work is extending our previous approximate formulation of lower linear envelope potentials to an exactly formula-

tion, it is necessary to compare the performance of MRF-LLSVMs to previous work [Gould, 2015]. In this section, we examine our method's effectiveness by comparing our results with [Gould, 2015, 2011] on a synthetic checkerboard. In order to demonstrate that our formulation has the capability to learn a much richer class of energy function's representation, we experiment our method on three different problem instances: checkerboard with squares containing monotonous color 4.4.2, checkerboard with squares containing more pixels of one color over another 4.4.3 and checkerboard with uniformly colored squares containing unbalanced color 4.4.4.

4.4.1 Experiment Settings

An image of synthetic checkerboard contains 8×8 pixel squares. Each square (clique) contains 16×16 (256) pixels. The color of each pixel is either black 0 or white 1. Given a ground-truth checkerboard image $\boldsymbol{y}^* = y_1^*, \ldots, y_{16384}^*$, the observed unary terms $\boldsymbol{y} = y_1, \ldots, y_{16384}$ are generated as followings. Let η_0 and η_1 be the signal-to-noise ratios for the black and white squares, the unary terms are generated by destroying groud-truth label to noisy input

$$y_i = \eta_0 [\![y_i^\star = 0]\!] - \eta_1 [\![y_i^\star = 1]\!] + \delta_i \tag{4.35}$$

where $\delta_i \sim \mathbb{U}(-1, 1)$ is additive i.i.d. uniform noise. $[\![x]\!]$ is an indicator function which equals 1 when x is true and 0 otherwise. The task is to recover the ground-truth checkerboard from the noisy input.

Our MRF is constructed on this image by associating each node in the MRF to each pixel in the image. Thus our MRF contains $8 \times 8 \times 256 = 16,384$ variables. The energy function used in this experiment follows equation equation 4.12 without pairwise terms.

$$E(\boldsymbol{y}; \boldsymbol{\theta}) = \theta^U \sum_{i \in \mathcal{N}} \phi^U(\boldsymbol{y}_i) + \sum_{\boldsymbol{y}_c \in \mathcal{C}} \phi^H(\boldsymbol{y}_c, \boldsymbol{z}_c; \boldsymbol{\theta}^H) \tag{4.36}$$

where $\phi^U(\boldsymbol{y}_i) = \boldsymbol{y}_i$ and θ^U is a scalar weight for unary terms. $\phi^H(\boldsymbol{y}_c, \boldsymbol{z}_c; \boldsymbol{\theta}^H) = \boldsymbol{\theta}^{H\,T} \boldsymbol{\phi}(\boldsymbol{y}_c, \boldsymbol{z}_c)$ is equivalent to equation equation 4.23 and added for each square (clique c) in the checkerboard. The number of linear equations K in equa-

tion equation 4.24 is set to be 10. The parameters θ^U and θ^H are learned using algorithm 3 with $MaxIter = 100$.

4.4.2 Monotonous Colored Squares

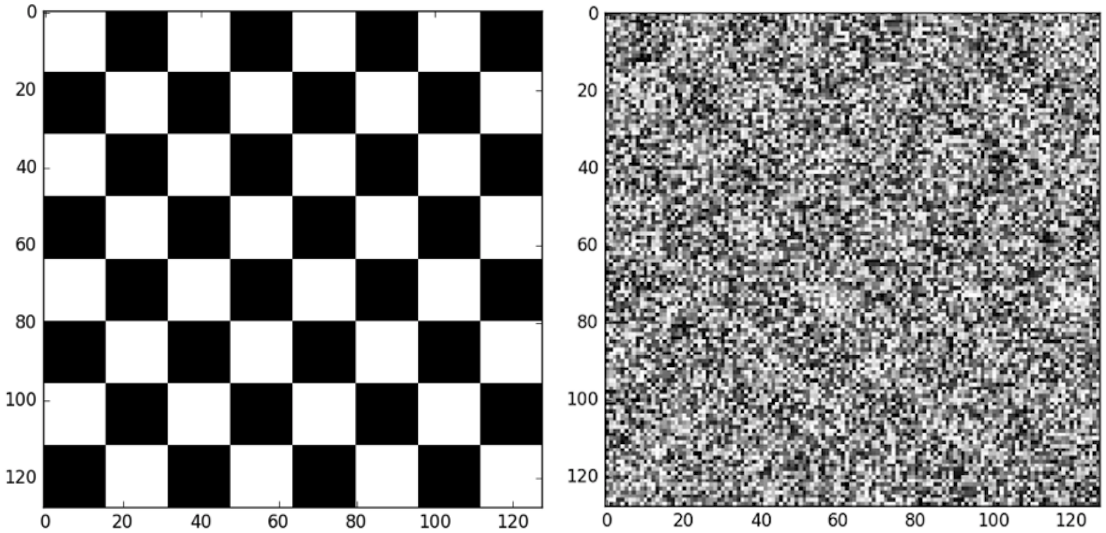


Figure 4.6: Example for monotonous colored squares. figure (a) is the ground-truth checkerboard. Figure (b) is the noisy input (unary terms) destroyed by equation equation 4.35

We first repeat our previous black and white checkerboard experiment [Gould, 2011, 2015] in order to examine the correctness of our new formulation. Each clique (square) $c \in \mathcal{C}$ in the checkerboard contains either all white pixels $y_i = 1,\ \forall i \in c$ or all black pixels $y_i = 0,\ \forall i \in c$. Figure 4.6 illustrates the ground-truth checkerboard and the noisy input destroyed by equation equation 4.35 with $\eta_0 = \eta_1 = 0.1$. Figure 4.7 shows the results of our new method (on the bottom) together with our previous method [Gould, 2015] (on the top).

From figure 4.7 we conclude that both formulations can recover checkerboard perfectly so our new formulation's accuracy is as good as previous one. However, there are significant differences between structural SVM formulation (previous method) and latent structural SVM formulation. There are 10 active linear functions in figure 4.7 (b) while there are only 2 active linear functions in figure 4.7 (d). Shapes learned by each formulation are also significantly different.

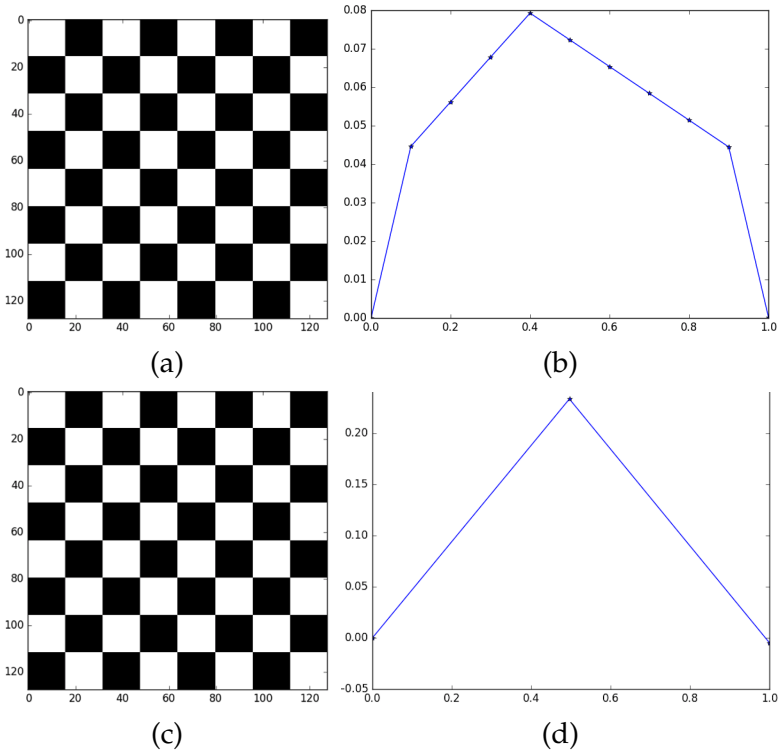


Figure 4.7: Results comparison for monotonous colored squares. Figure (a) and Figure (c) are inferred checkerboard from our previous and current formulation separately. Figure (b) and Figure (d) are lower linear envelopes learned by each formulation.

In general, the second result is more preferable than the first one. The reason is despite the image contains 64 cliques, there are only two kinds of squares in the image: completely black and completely white. Accordingly, our model only see two kinds of cliques: completely 0s (black) and completely 1s (white). In this case, a lower linear envelope contains two linear functions is enough for encoding consistency information. This is reflected in figure 4.7 (d) which gives least penalty (0) when the clique value $W_C(y_c)$ equals either 0 or 1. It gives the highest penalty when $W_C(y_c)$ is in the middle because our model has least probability seen that in training data. The results certificates that our latent structural SVM formulation can learn lower linear envelope exactly. Therefore, we say that our new method learns more preferable lower linear envelope.

In terms of computational performance, because our initial point are generated randomly using algorithm 5, the performance various between runnings.

On average it takes 2 *outer loops* and 47 *inner loops* to converge. Which means the latent structural SVM formulation spends 3.5 times iterations to converge than previous one (27 iterations). Each *inner loop* took under 1s with inference taking about 120ms on a 2.7GHz dual-core Intel CPU, which is the same as our previous method.

4.4.3 Unbalanced Colored Squares

Experiment in section 4.4.2 proves that our latent structural SVM formulation can learn the lower linear envelope exactly. In this section we conduct further experiment to investigate its capability of representing unbalanced input. The desirable result of this experiment should be the shape of the lower linear envelope shifting along with the changing of input data.

We design our checkerboards contain unbalanced colored squares as shown in figure 4.8.

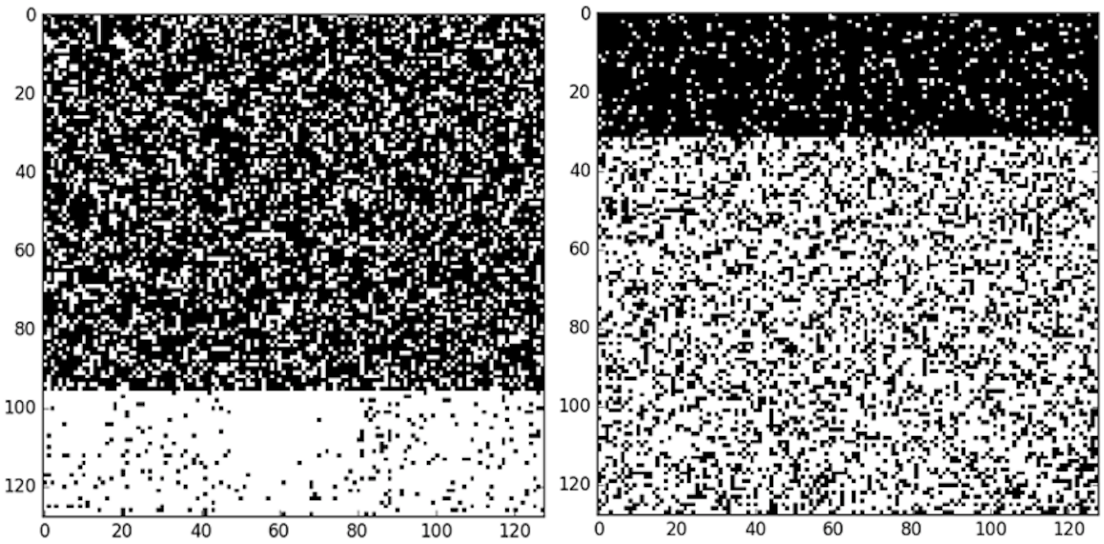


Figure 4.8: Example for unbalanced colored squares. In figure (a) 75% cliques contain more than 85% black pixels while 25% cliques contain more than 85% white pixels. Figure (b) is the opposite of figure (a)

As before, figure 4.9 shows results learned by structural SVM (top row) and latent structural SVM (bottom row). The accuracy performance of both methods are almost the same. Both methods are able to recover $45\% - 50\%$ pixels.

The shape of each formulations' results are both preferable and very similar when compared to each other. The most significant difference is the number of linear functions (10 active linear functions v.s. 2). In terms of computational performance, our previous method only takes 10 iterations to converge while the latent structural SVM formulation takes 89 iterations. Our new method is much more computational expensive than our previous method.

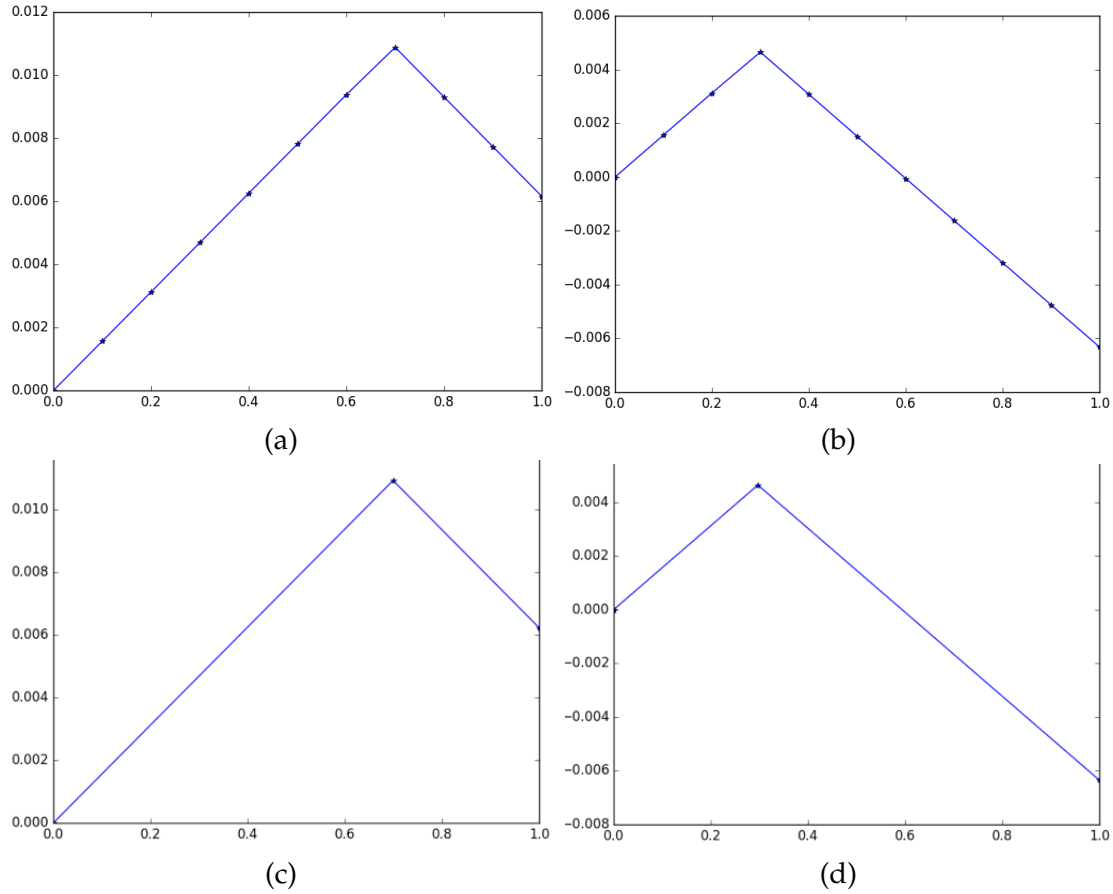


Figure 4.9: Results comparison for unbalanced colored squares. Figure (a) and Figure (b) are lower linear (more black and more white) envelopes learned by structural SVM. Figure (c) and Figure (d) are learned by latent structural SVM.

4.4.4 Uniformly Colored Squares

All of the above experiments show that our new method can significantly simplify the shape of the lower linear envelope function while maintaining the inference performance at the same level. However, one significant cost is the

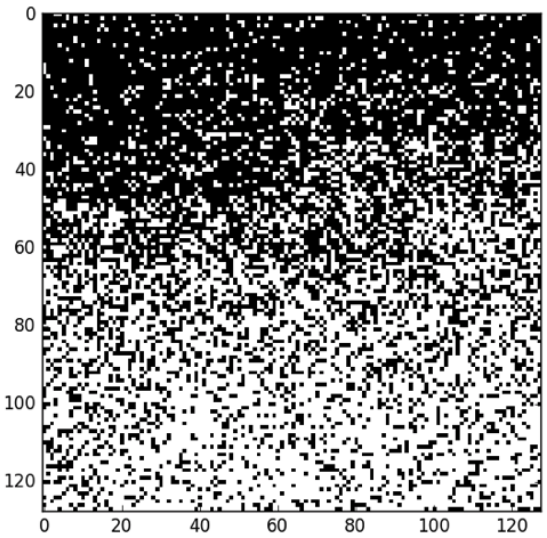


Figure 4.10: Uniformly colored squares example. $W_c(\boldsymbol{y}_c) = \sum_{i \in c} w_i^c y_i$ is uniformly distributed from 0 to 1.

computational performance. It still remains obscure if there exists any other advantages. In this section we design a much harder problem. $W_c(y_c)$ is uniformly distributed from 0 to 1. Figure 4.10 shows the result. The preferable shape of the lower linear envelope should contain a line which is parallel to the x-axis.

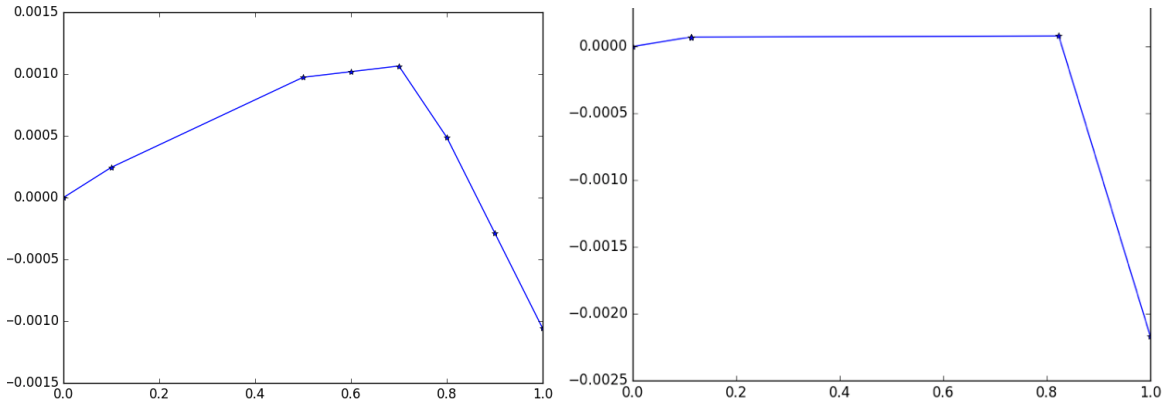
Results are shown in figure 4.11. As we can see that shapes are very different between two formulations. Our latent structural formulation (figure 4.11 (b)) learned a very flat representation of the lower linear envelope function, which is much preferable, while the structural SVM formulation preserves much concavity in the shape. This might because in previous work [Gould, 2015, 2011] we imposed strict concave constraints on parameter vector $\boldsymbol{\theta}$.

The performance of accuracy also various significantly. Under this formulation our new method is still able to recover $45\% - 50\%$ pixels while our previous can only recover $25\% - 30\%$ pixels on average. Therefore, our new formulation finally outperforms previous one. In terms of computational performance, the new formulation takes 129 *inner loops* in total (2 *outer loops*) while our previous formulation takes 75 iterations to converge. Although the new formulation is still more computational expensive than previous one, the gap decreases significantly.

We consider all of those improvements are due to our new method is able

to learn the lower linear envelope exactly.

One subtle thing is that the linear function on the right side in figure 4.11 (b) decreases sharply which seems abnormally at first glance. The reason is that we assume $b_1 = 0$ in section 4.2.1 which fixes the y-intercept of the first linear function to be zero. Therefore the last linear function can be arbitrarily deep while the first linear function is fixed at the original point.



(a) (b)

Figure 4.11: Results of uniformly colored squares experiment. Figure (a) is the result learned by structural SVM formulation. Figure (b) is the result learned by latent structural SVM formulation.

4.4.5 Conclusions

From above experiments we conclude our findings as followings:

- All of those experiments verified that our latent structural formulation is able learn the lower linear envelope exactly.
- In general (see section 4.4.2 and section 4.4.3), our new method have equivalent accuracy performance to our old method (structural SVM formulation[Gould, 2011, 2015]).
- In terms of computational performance, the new formulation

  during training. However, it is more efficient during testing due to it simplicity for the lower linear envelope potentials.

- For harder problem (see section 4.4.4), the new method outperforms the previous one significantly. The gap of computational performance also decreases a significant amount.

Chapter 5

Application: Learning Higher-Order Dynamics on China Securities Index (CSI) 300

In Chapter 4 we proposed a novel framework, the MRF-LSSVMs, to optimize Markov Random Fields with higher-order Lower Linear Envelope Potentials under the Latent Structural Support Vector Machine framework. On synthetic checkerboard experiments, MRF-LSSVMs has shown its efficiency in representing higher-order dependencies and encouraging consistency among large group of random variables. In this Chapter, we continue our experiment on the real financial market stock price data set and show how MRF-LSSVMs can be used to model dependency dynamics between time series. In order to do that, we first employ Recurrent Neural Networks (RNNs) as unary energy functions. Each stock is treated as a unary node in MRFs and RNNs are used to extract feature from each stock's historical market price time series. We then layer MRFs on top of RNNs extractor and optimize the entire framework with the LSSVMs algorithm proposed in Section 4.3.

Specifically, it is well known that single price movement of an individual stock not only depends on historical records but also highly correlated to other stocks [Lo and MacKinlay, 1990, Mech, 1993] and may change in a non-synchronous manner [Lo and MacKinlay, 1990, Brennan et al., 1993]. This correlated yet asynchronous price movement is sometimes referred to as the lead-lag relationship [Hou, 2007] between a group of stocks and is thought to arise from the different speed of information diffusion[Lo and MacKinlay, 1990, Badrinath et al., 1995, McQueen et al., 1996]. When new information

hits the market, some stocks react faster than others and identification of these leading stocks and their lead-lag relationships to other lagging stocks provides strong predictive evidence to the latter's price movement.

However, there are three key challenges in utilizing the lead-lag relationship: (1) discovering which stock will be affected by newly arriving information (such as news); (2) identifying the group (*e.g.*, industry, supply chain, *etc*.) it belongs to along with the leading and lagging stocks in this group and modeling their relationships; (3) predicting the price movement of each stock by jointly considering knowledge in the correlated group and an individual stock'price movement at that moment.

The first challenge is extremely difficult, not only because it requires an expert level of understanding of the finance system and market dynamics and the stock price, but also due to a lack of training data. However, according to the efficient market hypothesis [Malkiel and Fama, 1970], stock price reflects all available market information. Economists hitherto to used patterns hidden inside historical trading prices and volume to predict future price movements [Fama and Blume, 1966, Jensen, 1967]. As a result, hundreds of handcrafted features, known as technical analysis indicators [Kirkpatrick II and Dahlquist, 2010], have been designed. However, most of these models have stopped generating profitable signals since the early 1990s [Park and Irwin, 2007].

To overcome these problems and address the first challenge, here we employ an end-to-end hierarchical multi-task [Caruana, 1993] RNN to extract informative changes from raw market prices without using hand-crafted features such as technical analysis indicators. Good price prediction relies on rich representations and a multi-task framework that can leverage complementary aspects from diverse tasks [Søgaard and Goldberg, 2016]. Specifically, given raw market price data, which only contains six features (opening price, low price, high price, closing price, volume, and amount) at each time interval, we leverage a hierarchical multi-task network to first extract features on different tasks and then concatenate those complementary feature vectors to make the final prediction.

The MRF-LSSVMs framework can be employed to solve the other challenges. In our implementation, we treat each stock as a node in MRFs and each stock's group with lead-lag relationships as a maximum clique in MRFs.

The complexity of modeling dynamics between leading and lagging stocks becomes encouraging consistency over large cliques under weighted lower linear envelope potentials. Logits from hierarchical RNN networks are used as unary features in MRFs. By minimizing the energy function which contains both unary and higher order features, we can predict each stock's future price movement by jointly considering individual market price trends together with lead-lag relationships.

Unlike the first challenge trying to avoid prior knowledge, we consider being able to embed prior knowledge as an advantage. Definitions of sectors as well as leading and lagging stocks in each sector require solid financial industry research. Statistical evidence learned automatically from market price data are usually insufficient for determining such relationships.

We demonstrate the effectiveness of the proposed technique using three popular Chinese stock market indexes, and the proposed method outperforms baseline approaches. To our best knowledge, the proposed technique is the first one to investigate intra-clique relationships with higher-order MRFs on stock price movement prediction.

To summarize, the main contributions of this Chapter are:

- We propose a hierarchical multi-task RNN architecture to learn stock price patterns without hand-crafted features. To our best knowledge, this is the first work proposing a multi-task neural networks for stock price movement prediction.

- We propose the first model that encode lead-lag relationships between stocks using higher-order MRFs.

5.1 Related Works

Lead-lag relationships: Lead-lag relationships have long been recognized in the stock market. They can arise for many reasons such as information diffusion, sector (industry) rotation, investment style rotation, event-driven trading, and asynchronous trading [Lo and MacKinlay, 1990, Chordia and Swaminathan, 2000, Conrad and Kaul, 1988, Hameed, 1997]. It is generally believed that lead-lag relationships are more prevalent in firms in the same industry

[Hou, 2007], justifying our use of pre-defined industry classification list [ths] as prior domain knowledge of each stock's maximum clique. Several studies [Brennan et al., 1993, Hou, 2007, Badrinath et al., 1995, McQueen et al., 1996] have shown that stocks with larger capital size and higher liquidity tend to be leading stocks and vice versa. To replicate potential lead-lag relationships, we assign each stock a different weight from its corresponding indexes created by the China Securities Index Company, Ltd. More complicated dynamics hidden behind a clique of stocks are learned by higher-order MRFs.

Multi-task learning: Caruana [1993] showed that inductive knowledge learned from multiple tasks can transfer between tasks and help improving generalization of all tasks. Many Natural Language Processing (NLP) tasks take advantage of multi-task frameworks and achieve state-of-the-art performance while using simple models for each of these tasks [Søgaard and Goldberg, 2016, Hashimoto et al., 2016]. However, as noted elsewhere [Caruana, 1993, Ruder, 2017], there is a lack of theory on underpinning a diverse set of tasks and the hierarchical architecture of the chosen tasks. Recent works [Søgaard and Goldberg, 2016, Hashimoto et al., 2016] apply the principle that the task complexity should increase according to hierarchical level, and we do likewise. Because technical analysis indicators can be categorized into trend, momentum, volatility and volume [Kirkpatrick II and Dahlquist, 2010], and volume is included in market price data, we propose an architecture that uses trend and volatility tasks as lower level tasks and price movement prediction (upward or downward) as a higher level task. Other task selection and hierarchical designations remain open for further research.

5.2 Methods

In this section, we first introduce the multi-task RNN-MRFs architecture which is constructed with two parts. The detailed architecture is shown in Figure 5.1.

The first part is a "Multi-task Market Price Learner", which consists of three dual stage attention based recurrent neural network (DARNN) [Qin et al., 2017] modules. The goal of the first part is to tackle the first challenge, i.e., automatically extracting informative representations of the raw market price without considering any hand-crafted feature and technical indicator.

The second part is an "Intra-clique Predictor" which is the adaption of MRF-LSSVMs on stock market. In Intra-clique Predictor, higher order functions are applied to sector lists (used as maximum cliques) defined by financial index companies. The domain knowledge about leading stocks and lagging stocks are assigned as higher and lower weights in energy function accordingly. The goal of this part is to tackle the second and third challenges. Unary features learned by DARNN modules are jointly employed to maintain higher order consistency among stocks belonging to the same sector.

5.2.1 Multi-task Market Price Learner

Stock price movement can be interpreted from many aspects such as investors sentiment, temporal patterns and cycles, flow of funds and market strength, *etc*. Ideal features should incorporate as many aspects as possible. Multi-task learning has shown its effectiveness to learn inductive knowledge among tasks and improve performance as well as generalization capability [Caruana, 1993]. Therefore, we propose a multi-task RNN framework entitled "Multi-task Market Price Learner (MMPL)" to tackle the first challenge: extracting informational representations from raw market price.

However, as noted elsewhere [Caruana, 1993, Ruder, 2017], there is a lack of theory on underpinning a diverse set of tasks and the hierarchical architecture of the chosen tasks. We follow this intuition to construct our model. Most technical indicators fall into four categories: trend, momentum, volatility and volume [Kirkpatrick II and Dahlquist, 2010]. Since volume is included in input for all low-level tasks and we assume that momentum information can be learned by a high-level task, we propose an architecture that using trend and volatility tasks as our low-level tasks and price movement prediction (upward or downward) as the high-level task.

Multi-task Market Price Learner(MMPL) contains two levels, three modules of DARNNs. DARNNs [Qin et al., 2017] are used as our basic module not only because of its capability of selecting relevant deriving series as well as temporal features, but also due to its superior performance for time series prediction compared to LSTM [Hochreiter and Schmidhuber, 1997] and attention based LSTM [Bahdanau et al., 2014]. Specifically, the bottom level contains two separate DARNN modules. They are supervised by low-level tasks which aim

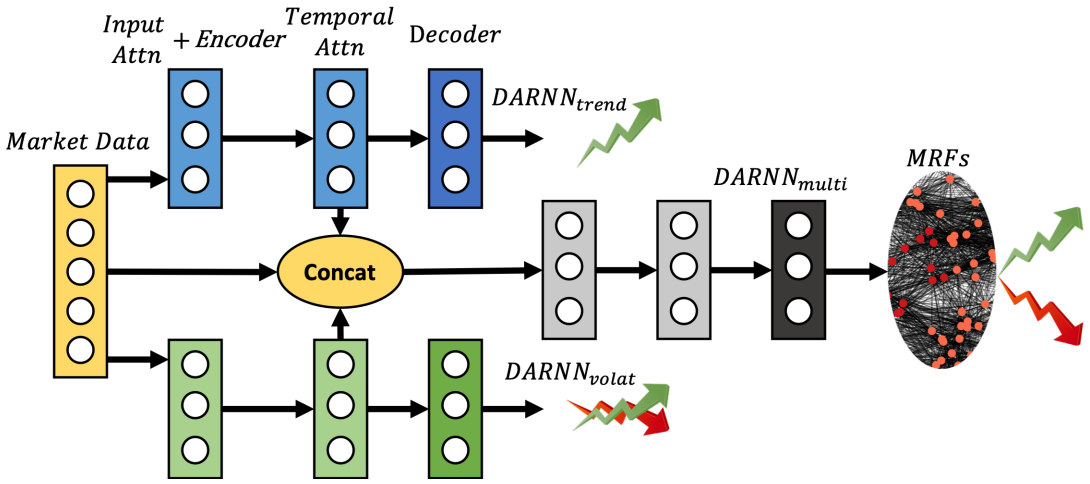


Figure 5.1: Multi-task RNN-MRFs architecture. Note that the output of $\text{DARNN}_{\text{multi}}$ only corresponds to one node's unary feature in MRFs.

to predict future price as well as volatility based upon the raw market price data. The key difference among those modules is the loss function. At the top level, it is supervised by a high-level task that learns to use representations extracted by two low-level modules as well as raw market price data to predict positive / negative price movement of stocks. Logits of the last layer are passed to Intra-clique Predictor described in section 5.2.2 as unary features.

All three DARNN modules share the same raw market price data. Here, we denote the time-series dataset as $\boldsymbol{X}$ where $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{N \times T}$. We use $\boldsymbol{x}^n = (x_1^n, x_2^n, \ldots, x_T^n) \in \mathbb{R}^T$ to denote a driving series of T time-steps and $\boldsymbol{x}_t = (x_t^1, x_t^2, \ldots, x_t^N) \in \mathbb{R}^N$ to denote a snapshot at time-step t of all N features.

For both DARNN modules at the low level, the input is $\boldsymbol{X} \in \mathbb{R}^{5 \times T}$ which contains 5 exogenous driving series, *i.e.*, opening price, low price, high price, volume, amount and 1 target series $\boldsymbol{y} = (y_1, y_2, \ldots, y_T) \in \mathbb{R}^T$. These two modules aim to predict target series y_{t+p} in the next p time steps:

$$\hat{y}_{t+p} = \text{DARNN}(y_1, \ldots, y_t, x_1, \ldots, x_t)$$

The target series $\boldsymbol{y}_{\text{trend}}$ of $\text{DARNN}_{\text{trend}}$ is the closing price. The target series $\boldsymbol{y}_{\text{volat}}$ of $\text{DARNN}_{\text{volat}}$ is the standard deviation of closing price over M constant time-steps. In our implementation we set $M = 10$. We use Mean Squared Error (MSE) as the loss function to train those two modules separately.

To construct the high level DARNN module, which aims to predict the price movement, we concatenate context vectors $\boldsymbol{c}_T$ from each of low level DARNN module's encoder and raw market price matrix as the input. The target series $\boldsymbol{y}^{\text{binary}}$ is constructed by the sign function $y_t^{\text{binary}} = sign(y_{t+p} - y_t)$ where y_t denotes closing price at time-step t. We use cross-entropy as loss function to train the final $\text{DARNN}_{\text{multi}}$. Logits (outputs before going through *softmax*) of $\text{DARNN}_{\text{multi}}$ are then passed to Intra-clique Predictor as unary features.

In order to train MMPL together with MRFs in an end-to-end manner, we follow the subgradient method proposed by Witoonchart and Chongstitvatana [2017]. Since our inner loop proposed in section 4.3.2 is actually a latent structural SVM. Only gradients of parameters and feature functions need to be updated. In our framework, outputs of MMPL (Logits of $\text{DARNN}_{\text{multi}}$) are only used as unary features in MRFs' energy functions, our back-propagation rules can be defined by taking derivative of the objective function *w.r.t* $\boldsymbol{w}^U$ defined in equation 4.34:

$$\frac{\partial L}{\partial \boldsymbol{w}^U} = \psi^U(y) - \psi^U(y^*) \tag{5.1}$$

where y is the ground-truth label and y^* is inferenced label. ψ^U is unary feature function described in section 4.2.1, here it denotes logits calculated from $\text{DARNN}_{\text{multi}}$. $\boldsymbol{w}^U$ is unary parameter defined in energy function equation 4.12. Equations equation 5.1 can be directly plugged into sub-gradient algorithm proposed in [Witoonchart and Chongstitvatana, 2017]. Other configurations stay the same with their algorithm.

5.2.2 Intra-clique Predictor

In this section, we show how to construct an "Intra-clique Predictor" with MRF-LSSVMs to model lead-lag relationships and address the other two challenges as mentioned in the introduction. Specifically, each stock is treated as a node in MRFs and each stock's group with lead-lag relationships is treated as a maximum clique in MRFs. We use a pre-defined industry classification list [ths] as the prior domain knowledge of each maximum clique for each

stock. By using a weighted version of higher order functions, stocks have higher weights in the above list can be seen as leading stocks, and vice versa. However, in our implementation, we use logits from MMPL as unary function and weighted lower linear envelopes as higher order function to encode lead-lag relationships. Pairwise features are excluded. Finally, the complexity of modeling dynamics between leading and lagging stocks becomes encouraging consistency over large cliques under weighted lower linear envelope potentials. Log- its from hierarchical RNN networks are used as unary features in MRFs. By minimizing the energy function which contains both unary and higher order features, we can predict each stock's future price movement by jointly considering individual market price trends together with lead-lag relationships.

The detail of the optimization algorithm is summarized in algorithm 4. As we mentioned in Appendix 5.6, although we proposed an end-to-end subgradient algorithm is section 5.2.1, MRFs updated by such algorithm take too many iterations to converge. Therefore, we propose a two-stage training procedure. At first stage, MMPL and MRFs are trained separately. Therefore, MRFs can take advantage of the efficient latent structural SVM and converge in a polynomial number of iterations. After all those models are converged, we then combine them together to conduct end-to-end training. Note that the CCCP Inner Loop in algorithm 4 is actually solving standard structural SVM problem. Therefore, at the second stage, we use subgradient algorithm proposed in section 5.2.1 to replace the CCCP Inner Loop. Other settings remain the same.

Empirically, the more evenly distributed of $W_c(Y_c)$ where $c \in \mathcal{C}$ on x axis, the more rich representation (number of linear functions) the energy function should have. In order to initialize θ, we first determine the x-coordinate of sampled points sp. Then we sample its y-coordinate from a uniform distribution $\mathcal{U}(\text{upbound}, \text{upbound} - 0.5)$ to add some randomness in our initialization as well as maintain concavity. Linear parameters a_k and b_k are later calculated using those sampled points sp_k and sp_{k-1}. At last we encode $\{a_k, b_k\}_{k=1}^{K}$ into θ using equation equation 4.24. This algorithm is summarized in algorithm 5.

Algorithm 4 Learning lower linear envelope MRFs with latent variables.

Set $MaxIter = 100$
input training set $\{\boldsymbol{y}_i\}_{i=1}^n$, regularization constant $C > 0$, and tolerance $\epsilon \geq 0$
Initialize $\boldsymbol{\theta}$ using algorithm 5
repeat
  CCCP Outer Loop
  Set $iter = 0$
  for each training example, $i = 1, \ldots, n$ **do**
    compute $\mathbf{z}_i^* = \arg\max_{\mathbf{z} \in \mathcal{Z}} \theta \cdot \psi(\mathbf{y}_i, \mathbf{z})$
  end for
  initialize active constraints set $\mathcal{C}_i = \{\}$ for all i
  repeat
    CCCP Inner Loop
    solve the quadratic programming problem in equation 4.34 with respect to active constraints set $\mathcal{C}_i$ for all i and concavity constraints $A\boldsymbol{\theta} \geq \epsilon$ to get $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{x}}_i$
    for each training example, $i = 1, \ldots, n$ **do**
      compute $\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i = \arg\min_{\boldsymbol{y}} E(\boldsymbol{y}, \boldsymbol{z}; \hat{\boldsymbol{\theta}}) - \Delta(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{y}_i)$
      if $\hat{\xi}_i + \epsilon < \Delta(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i, \boldsymbol{y}_i) - E(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{z}}_i; \hat{\boldsymbol{\theta}}) + E(\boldsymbol{y}_i, \boldsymbol{z}_i^*; \hat{\boldsymbol{\theta}})$ **then**
        $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\boldsymbol{y}_i^\star\}$
      end if
    end for
  until no more violated constraints
  return parameters $\hat{\boldsymbol{\theta}}$
  Set $iter = iter + 1$
until $iter \geq MaxIter$
return parameters $\hat{\boldsymbol{\theta}}$

5.3 Dataset and Model Settings

In this section, we first introduce 3 stock datasets. Then, we introduce the parameter settings for our model and training details. Finally, we select four evaluation metrics and use them to demonstrate the effectiveness of our proposed model by comparing to several baseline approaches.

To demonstrate the effectiveness of higher order consistency, we choose three exclusive and the most famous stock indexes on Chinese stock market to build our input datasets. Their index codes are: CSI (China Securities Index) 200, CSI 300 and CSI 500 which contain 200, 500 and 300 constituent stocks respectively. The CSI 300 index selects most liquid A-share stocks. It aims to

Algorithm 5 Empirical initialization algorithm for θ

$gap = \frac{1}{K}, a_1 = \mathcal{U}(0, 1e6), b_1 = 0, sp_1 = (0, 0), w_0 = 0, counter = 2$
for each clique $c \in \mathcal{C}$ **do**
  Compute weighted clique value $w_c = W_c(y_C)$
  if $w_c - w_{c-1} > gap$ **then**
    $upbound = a_{counter} w_c + b_{counter}$
    $sp_{counter} = (w_c, \mathcal{U}(upbound - 0.5, upbound))$
    Calculate $a_{counter}$ and $b_{counter}$ using $sp_{counter-1}$ and $sp_{counter}$
    $counter = counter + 1$
  end if
end for
If $counter < K$, remaining as and bs are all set to be $a_{counter}$ and $b_{counter}$
Calculate θ using $\{a_k, b_k\}_{k=1}^{K}$

reflect the overall performance of China A-share market. The CSI 200 and 500 indexes aim to reflect the overall performance of mid-to-large and small-to-mid capital A-shares respectively.

All these indexes are exclusive and are refined on a yearly basis. In this paper, we use fixed versions on 30-JAN-2015. We then collect their constituent stocks' minute-level data from 05-JAN-2015 to 29-DEC-2017. On Chinese stock market each trading day has 4 trading hours. So there are 240 samples (minutes) for each normally traded stock on each day. Each sample contains 6 features: opening price, high price, low price, closing price, volume, and amount. [1] For each stock, the first 80% days are used to construct the training set and the last 20% days are used as test set. Approximately training set and test set contain 33.6 million and 4.2 million samples, respectively. 49.5% of them are positive movements, 0.3% of them stay unchanged and 50.2% of them are negative movements. For binary classification task, we follow Mitchell and Pulvino [2001]'s approach and label all positive movement samples 1 and 0 for the other samples. More labeling details are described in 5.6.

[1] During this period, there are some stocks de-listed (SZ000024, SH600485, SH600832 in CSI 200; SZ000693, SZ000748, SZ000982 in CSI 500; SH600485, SH600832, SZ000024, SH601299 in CSI 300). Therefore, in total we collect 197, 497 and 296 stocks during this period respectively.

Table 5.1: Technical Indicators Selection

Category	Indicator Name
Momentum	Awesome Oscillator, Money Flow Index
Volume	Chaikin Money Flow On-balance volume mean
Volatility	Bollinger Bands (Upper and Lower Bands)
Trend	Average Directional Movement Index Moving Average Convergence Divergence

To demonstrate the benefits of multi-task RNN over manually designed technical indicators, we construct technical indicators datasets. We select 8 most popular indicators, 2 from each category [Kirkpatrick II and Dahlquist, 2010] shown in Table 5.1. In implementation, we use open source package *Technical Analysis Library in Python*[2] to calculate those indicators and all hyperparameters are using package's default settings without any prior expert knowledge involved with. After technical indicators calculation, these 8 new features are concatenated to above market price dataset (5 features at each minute). So the final input dataset for each single task model contains 13 features in total. Before feeding into models, we normalize each stock with *z-score* function using standard deviation and mean calculated in the training set.

For brevity, we denote market price dataset which only contains 5 features as **Market** and the concatenated 13 features dataset as **Indicator**. As discussed in section 5.2.1, closing price at time t can be directly used as regression target for $\text{DARNN}_{\text{trend}}$. Standard deviation of closing price with a window size of 10 is used as regression target for $\text{DARNN}_{\text{volat}}$. The dimensions of hidden state and cell state are fixed as 32 for $\text{DARNN}_{\text{trend}}$ as well as $\text{DARNN}_{\text{volat}}$, and 128 for $\text{DARNN}_{\text{multi}}$. More training details are described in 5.6.

5.4 Results

In order to demonstrate the effectiveness of our framework, we compare 3 baseline methods, *i.e.*, LSTM [Hochreiter and Schmidhuber, 1997], attention based LSTM Encoder_Decoder [Bahdanau et al., 2014], and DARNN [Qin et al.,

[2]https://github.com/bukosabino/ta

Data Set	Models	Chinese Securities Index (CSI)											
		CSI200				CSI500				CSI300			
		Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
Indicator	LSTM	62.30	73.82	70.70	72.23	60.35	68.56	70.03	69.29	60.16	71.39	68.50	69.91
	LSTM Encoder_Decoder	64.26	72.41	**74.34**	73.36	61.13	75.63	66.67	70.86	64.26	75.54	70.57	72.97
	DARNN	63.09	72.08	73.55	72.81	66.60	78.98	**74.13**	**76.48**	65.82	76.68	73.46	75.04
Market	LSTM	57.62	67.57	67.37	67.47	55.86	68.10	64.53	66.27	56.25	67.17	65.98	66.57
	LSTM Encoder_Decoder	59.57	71.60	66.86	69.15	58.40	69.53	68.12	68.81	61.33	71.87	68.91	70.36
	DARNN	61.13	71.26	71.47	71.37	63.09	77.04	67.87	72.16	63.87	72.09	73.59	72.83
	$DARNN_{trend}$ + $DARNN_{multi}$	63.67	74.77	71.38	73.04	62.89	75.30	69.38	72.22	62.69	73.84	66.56	70.00
	$DARNN_{volat}$ + $DARNN_{multi}$	62.50	73.97	71.06	72.49	62.30	74.46	69.20	71.74	61.91	72.98	68.51	70.67
	MMPL	65.04	74.04	73.39	73.72	65.43	76.04	72.80	74.38	66.60	71.67	**78.90**	75.11
	MMPL+MRFs	**67.97**	**77.51**	73.91	**75.67**	**66.80**	**79.65**	72.78	76.06	**68.95**	**78.55**	74.71	**76.58**

Figure 5.2: Results: baselines and ablation studies. All models have a window size (lag steps) of 20 and predict price movement label at the next time step.

2017] on 3 different Chinese Securities Indexes with and without technical analysis indicators as inputs. Results are summarized in Table 5.2. All results are reported over the test sets. We select four metrics (Accuracy, Precision, Recall and F1 Score) as evaluation metrics to justify the effectiveness of the proposed approach. They are calculated by collecting all predicted labels of constituent stocks in each CSI index.

5.4.1 Effectiveness of multi-task framework

As mentioned earlier, to demonstrate effectiveness of multi-task framework, we use **Indicator** dataset, which contains both market price data and technical analysis indicators as inputs for baseline approaches and **Market** dataset which only contains market price data as inputs for MMPL (multi-task RNN) as well as baseline methods. For DARNN, we use a hidden size of 128. MMPL's configuration is described in section 5.6.1. As we can see in Table 5.2, single task models (LSTM, LSTM Encoder_Decoder, DARNN) tested on **Market** dataset (without technical analysis indicators as inputs) generally have worse performance with all 4 metrics. In particular, performance of DARNN models tested on **Indicator** dataset is consistently better than the ones on **Market** dataset. This proves that even with hand-crafted features, deep learning models can still benefit from diversified and complementary features.

To test the effectiveness of multi-task framework, we conduct ablation study with only one low-level task ($DARNN_{trend}$ or $DARNN_{volat}$) together with the high-level task module $DARNN_{multi}$. Results indicate that these two variants have comparable or slightly worse result than

DARNN on **Market**. This may because single task model does not provide diversified features while have more parameters than DARNN. Finally, MMPL outperforms all single task models and baseline methods on **Market**. This suggests that diversified and complementary tasks can help MMPL extract effective features. Specifically, by comparing MMPL and DARNN on **Market** as well as **Indicator**, we can see that MMPL generally outperforms DARNN on CSI200 and CSI300 indexes and is slightly worse than DARNN on **Indicator** of CSI500 index. We can conclude that by using multi-task RNNs, we can extract better or at least comparable features compared with hand-crafted features.

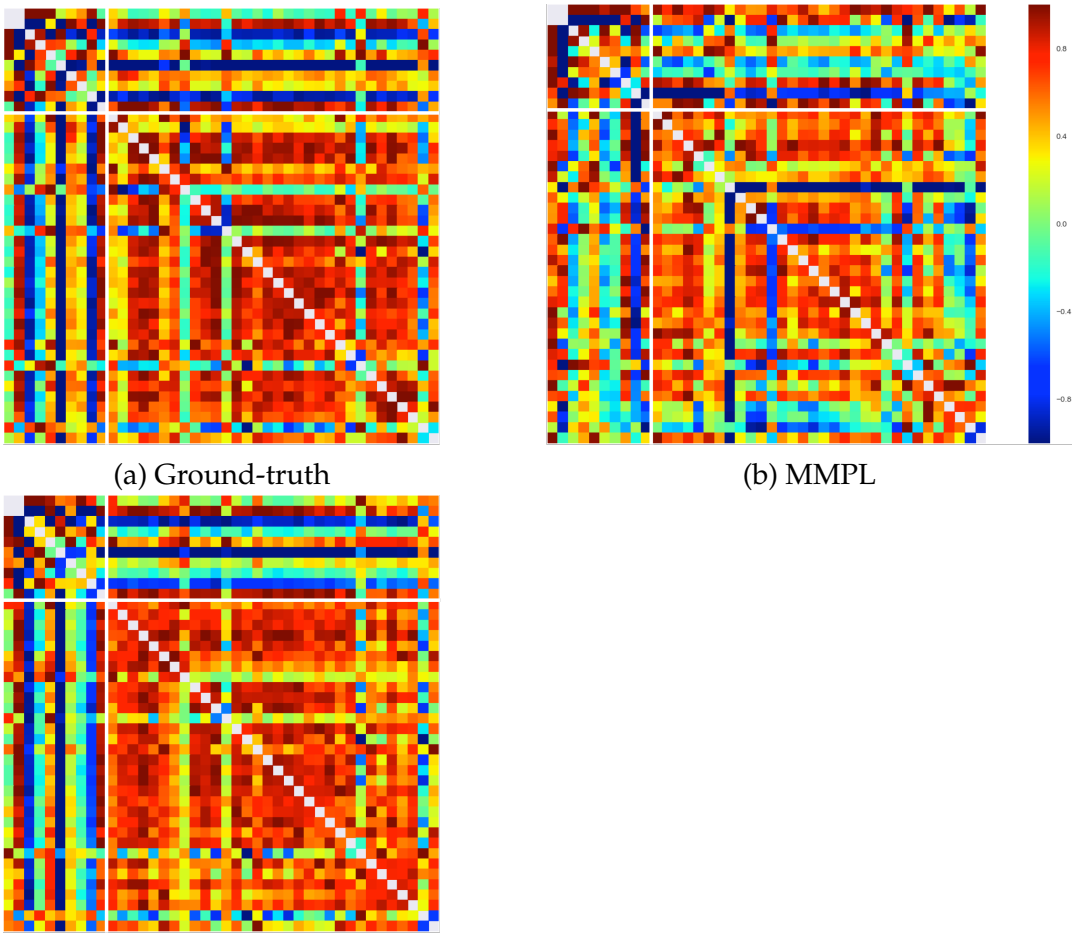
5.4.2 Effectiveness of higher-order MRFs

In Table 5.2, we can observe that MMPL-MRFs framework consistently outperforms other baselines on all 3 CSI index constituent stocks. It shows evidence that higher-order energy function can help with encoding clique level consistency thus improving overall prediction performance. One interesting point to note is that the recall rate of MMPL-MRFs is constantly lower than other baselines. This can be seen as a trade off between accuracy and recall rate. However, it is worth to mention that for stock price movement prediction, high accuracy and precision are much preferred than recall rate. Another interesting phenomenon is that MMPL-MRFs gives more improvement on CSI200 and CSI300 while less improvement over DARNN trained with technical analysis indicators on CSI500. One possible reason is that CSI200 and CSI300 select most liquid and representative stocks in Chinese stock market. Those stocks exhibit much stronger and higher order consistency than illiquid stocks. CSI500 selects small-mid capital stocks which are less liquid and contains much more noisy movements.

In the training stage, our algorithm converges in 4 to 19 CCCP outer loops. The average inference time of graph-cut algorithm is 34 seconds.

5.4.3 Visualization of higher-order consistency

In order to further investigate higher-order MRFs' effectiveness, we design a heat-map to visualize CSI300 index intra-clique higher-order relationship in figure 5.3.



(a) Ground-truth

(b) MMPL

(c) MMPL-MRFs

Figure 5.3: Higher order consistency visualization. (a) is calculated directly from ground truth labels on test set. (b) is calculated using predicted labels of MMPL without MRFs on the test set. (c), we use predicted labels of MMPL-MRFs on test set as inputs.

We first select two sectors: nonferrous metal sector, which contains 10 constituent stocks, and infrastructure sector, which contains 35 constituent stocks from CSI300 index [3]. We then measure consistency level between each two of these constituent stocks. In order to capture their temporal relationship, we propose a novel consistency measure which is calculated on temporal intervals.

Let $\boldsymbol{y}_i^T = \{y_i^1, y_i^2, ..., y_i^T\}$ denotes time-series for stock i. $y_i^t \in \{0, 1\}$ is

[3]These two sectors are selected only because painting many sectors in one figure would be too messy to interpret and those two sectors have appropriate clique size (number of stocks) for visualization. Conclusions from these two sectors also apply to other sectors

the binary price movement label at time t. We segment time-series $\boldsymbol{y}_i^T$ into $N = \lceil \frac{T}{P} \rceil$ non-overlapping intervals $\{y_i^n, y_i^{n+1}, ..., y_i^{n+P}\}$ with fixed length P. For any two stocks i and j, we calculate the difference $d_{ij}^n = \sum_n^{n+P} y_i^n - \sum_n^{n+P} y_j^n$ of how many times positive price movement happen in the n-th time interval in each stock. Then the consistency level c_{ij} between stocks i and j can be calculated via a $\ell_1$norm:

$$c_{ij} = -\|\mathbf{d}_{ij}\|_1$$

where $\mathbf{d}_{ij} = \{d_{ij}^1, d_{ij}^2, ..., d_{ij}^N\}$. We normalize c_{ij} into interval $[-1, 1]$. Each entry in figure 5.3 denotes a consistency level measure c_{ij}. The larger the c_{ij} is, the higher of consistency level between stock i and stock j, the color of corresponding entry is closer to red, and vice versa. As we mentioned, the average duration of information arrival-conduction-integration-release process is 4.04 minutes [Yan, 2012]. Since which stock is leading at each time interval is elusive, we set $P = 9$ when calculating consistency measures.

As we can see in Figure 4(a), there is a significant red square area, which means ground-truth heat-map shows strong intra-clique consistency. This is an evidence that higher-order relationships do exist within clique of stocks. However, in Figure 4(b), the red square area is fragmented into many little pieces. The whole area's color is closer to blue when compared to ground-truth heat-map, which means that MMPL captures little higher-order consistency. The reason we still can observe a shape of red square is that the accuracy of MMPL model on CSI300 is 66.6%. However, we can still conclude that the accuracy of single MMPL model mainly comes from unary features and it fails to capture higher order consistency of different stocks belonging to the same clique. On the contrary, even though MMPL-MRFs model's accuracy on CSI300 index is only 2.35% better than MMPL model, we can observe that heat-map Figure 4(c) is more close to ground-truth heat-map than heat-map Figure 4(b). There is a much clear red square and the number of small fragments in red area is also less than Figure 4(b). We can conclude that MMPL-MRFs models learn to utilize both unary features from MMPL as well as higher-order relationships encoded in MRFs.

5.5 Conclusions

Here we show how to model individual stock price predictions without hand-crafted features and encode lead-lag relationships between stocks using weighted higher-order MRFs. A multi-task neural network framework: Multi-task Market Price Learner (MMPL), is proposed to automatically extract diversified and complementary features from individual stock price sequences. Features learned by MMPL are passed to a binary MRF with a weighted lower linear envelope energy function to utilize intra-clique higher-order consistency between stocks. An efficient latent structural SVM algorithm is designed to learn MRFs in polynomial time. Finally, the MRFs and MMPL are trained end-to-end using the sub-gradient algorithm. Extensive experiments are conducted on three major Chinese stock market indexes, and the proposed MMPL-MRFs achieve the best accuracy on all three indexes.

Our work provides a number of directions for future research. In this work we proposed a multi-task recurrent neural network for stock price prediction. While we directly use DARNN as a proof of concept, other, more dedicated architectures are worthy of exploration. As well as time series tasks, we can also investigate how the latent SSVM framework performs on computer vision tasks. Another interesting direction is to investigate the implicit relationship between the expert-defined index list and graph RNN [You et al., 2018], which could further help to reduce the domain knowledge required by our framework.

5.6 Training Details

5.6.1 Multi-task training

To improve accuracy and reduce over-fitting, we add a drop out layer between input layer and LSTM layer with a ratio of 0.2. We also clip and normalize gradients during back-propagation stage with a maximum norm of 5.0 to prevent gradient exploding issue. As pointed out by Lample et al. [2016], the question of "when should the training schedule switch from one task to another task?" or "should each task be weighted equally?" remains open. In our implementation, we follow the proportional sampling approach described by

Søgaard and Goldberg [2016]. After a backward pass completed, we randomly sample a new task as well as its batch data as the next task to be trained. In practice, we use a proportion of $[0.25, 0.25, 0.5]$ for three tasks respectively. This mechanism helps multi-task model to avoid *Catastrophic Forgetting* phenomenon which means lower level model forgets learned knowledge during higher level model back-propagation pass.

Even though we propose an end-to-end training algorithm for MMPL and MRFs in section 5.2.1, MRFs inference stage is still too slow to be trained jointly with MMPL. To overcome this difficulty, we implement a two stages training procedure. We first add a *softmax* layer on top of $\text{DARNN}_{\text{class}}$ and train MMPL separately from MRFs. We use *Negative Log-likelihood* as the loss function. At the second stage, after MMPL converge, we remove the *softmax* layer and retrain it together with MRFs. One issue we must mention is that, even though we use binary MRFs which can only predict positive / negative price movement, we find there is a significant amount of time when stock price remains no change. We find it benefits the performance a lot if we treat the classification as a three classes problem rather than a binary classification problem during the first stage. Therefore, at the first stage, the *softmax* layer will output probability for three labels: *negative movement*, *no changes* and *positive movement*. Since binary MRFs still needs a two dimension input as part of unary energy function, after the *softmax* layer is removed, we add an additional linear mapping layer between logits of MMPL and MRFs at the second stage.

5.6.2 End-to-end multi-task RNN-MRFs training

With converged MMPL and MRFs at hand, now we can go forward to train them in an end-to-end manner. We only include pairwise energy function through section 5.2.2 and section 4.3 to show a general application of our proposed algorithm. In the case of Chinese stock market, to our best knowledge there is no public available definition of pairwise relationship between stocks. Therefore, in our implementation we only use unary and higher order energy function. Each stock is then treated as a node in MRFs and each stocks group which has lead-lag relationships is treated as a maximum clique in MRFs. One benefit of MRFs clique is that we can embed domain expert knowledge about industry classification as maximum cliques into our model. We choose to use

Tonghuashun industry classification [ths] in our model. One subtle but crucial detail about modeling lead-lag effect lies in equation (4.16). Recall that $W_c(\boldsymbol{y}_c) = \sum_{i \in c} w_i y_i$ with $w_i^c \geq 0$ and $\sum_{i \in c} w_i^c = 1$ which are weights for stocks in each clique. Therefore, leading stocks should have a higher weights while lagging stocks should have lower weights. In our implementation, we use constituents' weight defined in CSI200, CSI500 and CSI300 as their weights in equation (4.16) and normalize them to ensure the summation equals 1.

Part III

Conclusion

Chapter 6

Conclusion and Future Work

We summarize our work in this chapter. We will also conclude its advantages and disadvantages basing on our synthetic (section 4.4) experiment's results. Our work also provide some insights to our future work which we will briefly discuss in this chapter.

6.1 Conclusion

Lower linear envelope binary MRFs are raising interests due to its capability for encoding higher-order consistency constraints over large sets of random variables. Gould [2015] has shown how to perform exact inference and learning of this problem under the max margin framework. In order to transform the lower linear envelope function to a linear combination formulation, they interpolates it with a set of fixed space sample points. Thus their algorithm is only able to learn the shape of the lower linear envelope function approximately.

The main goal of our research is to learn the lower linear envelope function exactly. Based on their work, we explore a variant of their formulation by introducing auxiliary variables back to the energy function to formulate an exact representation. We find that the lower linear envelope function under the quadratic pseudo-Boolean formulation equation 4.22 itself is an inner product of parameters and features thus can be written into a linear combination directly. Under this formulation the inference algorithm (*st min cut* construction) developed by Gould [2015] still adapts to our problem. Therefore, we are still able to conduct exact inference on our problem. We developed the learning algorithm 3 using an extension of the max margin framework which is known as

latent structural SVM. However, this algorithm is only guaranteed to decrease the objective function to a local minimum thus the initial point will affect the overall performance. In order to overcome this issue we also proposed an empirical initialization algorithm 5.

In order to examine the effectiveness of our new algorithm, we repeat two experiments Gould [2015] conducted in their research and compare both results. In the first synthetic checkerboard experiment, we found that in general the new algorithm's accuracy is at least as well as the previous one. But on harder problem 4.4.4 the new method outperforms previous one significantly. The new method is much more computationally expensive during the training period. However it is more efficient during the testing stage because of the simplicity of the shape of the lower linear envelope. We also found that the shape learned by the new method can shift along with the changes of the input data which proves that we can learn the lower linear envelope exactly.

At last we summarize the advantages and disadvantages as following.

Advantages of the new method (compared to the previous method [Gould, 2015, 2011]):

- Able to learn the lower linear envelope exactly.
- Performs better (higher accuracy) on harder problems.
- Efficient to compute during testing due to the simplicity of the shape of the lower linear envelope function.

Disadvantages of the new method:

- Only guaranteed to decrease to the local minimum.
- Computationally expensive during training.
- Generalization various significantly.

6.2 Future Work

As we suggested in the conclusion, our new method seems to have some generalization issues. It will be our primal goal to keep investigating into this

problem. We also proposed an empirical initialization method in section 4.3.2. For future work we would compare this method to others.

Our research also provide insights to further directions. Extending our approach to multi-label MRFs seems to be very promising. Other straightforward extensions include the introduction of features for modulating the higher-order terms and the use of dynamic graph cuts [Kohli and Torr, 2007] for accelerating loss-augmented inference within our learning framework. Other optimization algorithms for solving our learning problem may also be considered, e.g., the subgradient method [Nowozin and Lampert, 2011, Bertsekas, 2004].

Bibliography

Tonghuashun industry classification. http://q.10jqka.com.cn/thshy/. Accessed: 2019-01-15.

R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *PAMI*, 34(11): 2274–2282, 2012.

P.-L. Bacon. *Temporal Representation Learning*. PhD thesis, McGill University Libraries, 2018.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

S. G. Badrinath, J. R. Kale, and T. H. Noe. Of shepherds, sheep, and the cross-autocorrelations in equity returns. *The Review of Financial Studies*, 8(2):401–430, 1995.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2004.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006a.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006b.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123:155–225, 2002.

Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, 2001.

M. J. Brennan, N. Jegadeesh, and B. Swaminathan. Investment analysis and the adjustment of stock prices to common information. *The Review of Financial Studies*, 6(4):799–824, 1993.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

R. A. Caruana. Multitask connectionist learning. In *In Proceedings of the 1993 Connectionist Models Summer School*. Citeseer, 1993.

T. Chordia and B. Swaminathan. Trading volume and cross-autocorrelations in stock returns. *The Journal of Finance*, 55(2):913–935, 2000.

J. Conrad and G. Kaul. Time-variation in expected returns. *Journal of business*, pages 409–425, 1988.

C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3): 337–357, 2016.

F. Doshi-Velez and G. Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, page 1432. NIH Public Access, 2016.

E. F. Fama and M. E. Blume. Filter rules and stock-market trading. *The Journal of Business*, 39(1):226–241, 1966.

P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

D. Freedman and P. Drineas. Energy minimization via graph cuts: Settling what is possible. In *CVPR*, 2005.

A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, 2007.

S. Gould. Max-margin learning for lower linear envelope potentials in binary Markov random fields. In *ICML*, 2011.

S. Gould. Learning weighted lower linear envelope potentials in binary markov random fields. *PAMI*, 37(7):1336–1346, 2015.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

A. Hameed. Time-varying factors and cross-autocorrelations in short-horizon stock returns. *Journal of Financial Research*, 20(4):435–458, 1997.

P. L. Hammer. Some network flow problems solved with psuedo-boolean programming. *Operations Research*, 13:388–399, 1965.

J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI*

Conference on Artificial Intelligence, 2018.

K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

K. Hou. Industry information diffusion and the lead-lag effect in stock returns. *The Review of Financial Studies*, 20(4):1113–1138, 2007.

H. Ishikawa. Exact optimization for Markov random fields with convex priors. *PAMI*, 25:1333–1336, 2003.

H. Ishikawa. Higher-order clique reduction in binary graph cut. In *CVPR*, 2009.

M. C. Jensen. Random walks: reality or myth—comment. *Financial Analysts Journal*, 23(6):77–85, 1967.

T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in Neural Information Processing Systems*, pages 6250–6261, 2017.

C. D. Kirkpatrick II and J. A. Dahlquist. *Technical analysis: the complete resource for financial market technicians*. FT press, 2010.

M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

P. Kohli and M. P. Kumar. Energy minimization for linear envelope MRFs. In *CVPR*, 2010.

P. Kohli and P. H. S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *PAMI*, 2007.

P. Kohli, M. P. Kumar, and P. H. S. Torr. P3 & beyond: Solving energies with higher order cliques. In *CVPR*, 2007.

P. Kohli, L. Ladicky, and P. H. S. Torr. Graph cuts for minimizing higher order potentials. Technical report, Microsoft Research, 2008.

P. Kohli, P. H. Torr, et al. Robust higher order potentials for enforcing label

consistency. *IJCV*, 82(3):302–324, 2009.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26:65–81, 2004.

A. Kolobov, D. S. Weld, et al. Discovering hidden structure in factored mdps. *Artificial Intelligence*, 189:19–47, 2012.

A. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pages 15512–15522, 2019.

G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753, 2019.

K. Y. Levy and N. Shimkin. Unified inter and intra options learning using policy gradient methods. In *European Workshop on Reinforcement Learning*, pages 153–164. Springer, 2011.

Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822, 2017.

A. W. Lo and A. C. MacKinlay. When are contrarian profits due to stock market overreaction? *The review of financial studies*, 3(2):175–205, 1990.

B. G. Malkiel and E. F. Fama. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417, 1970.

G. McQueen, M. Pinegar, and S. Thorley. Delayed reaction to good news and the cross-autocorrelation of portfolio returns. *The Journal of Finance*, 51(3): 889–919, 1996.

T. S. Mech. Portfolio return autocorrelation. *Journal of Financial Economics*, 34 (3):307–344, 1993.

M. Mitchell and T. Pulvino. Characteristics of risk and return in risk arbitrage. *the Journal of Finance*, 56(6):2135–2175, 2001.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level

control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

S. Nowozin and C. H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4): 185–365, 2011.

C.-H. Park and S. H. Irwin. What do we know about the profitability of technical analysis? *Journal of Economic Surveys*, 21(4):786–826, 2007.

C. F. Perez, F. P. Such, and T. Karaletsos. Generalized hidden parameter mdps transferable model-based rl in a handful of trials. *arXiv preprint arXiv:2002.03072*, 2020.

M. L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *IJCAI*, 2017.

K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, pages 5331–5340, 2019.

M. Riemer, M. Liu, and G. Tesauro. Learning abstract options. In *Advances in Neural Information Processing Systems*, pages 10424–10434, 2018.

C. Rother, P. Kohli, W. Feng, and J. Jia. Minimizing sparse higher order energy functions of discrete variables. In *CVPR*, 2009.

S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

M. Smith, H. Hoof, and J. Pineau. An inference-based policy gradient method for learning options. In *International Conference on Machine Learning*, pages 4703–4712, 2018.

A. Søgaard and Y. Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 231–235, 2016.

D. Song, W. Liu, T. Zhou, D. Tao, and D. A. Meyer. Efficient robust conditional random fields. *IEEE Transactions on Image Processing (T-IP)*, pages 3124–3136, 2015.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. 2018.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

M. Szummer, P. Kohli, and D. Hoiem. Learning CRFs using graph-cuts. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2008.

B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *ICML*, 2005.

Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dmcontrol: Software and tasks for continuous control, 2020.

D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, G. Wayne, R. Pascanu, Y. W. Teh, and N. Heess. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

P. Witoonchart and P. Chongstitvatana. Application of structured support vector machine backpropagation to a convolutional neural network for human pose estimation. *Neural Networks*, 92:39–46, 2017.

M. Wulfmeier, D. Rao, R. Hafner, T. Lampe, A. Abdolmaleki, T. Hertweck, M. Neunert, D. Tirumala, N. Siegel, N. Heess, et al. Data-efficient hindsight off-policy option learning. *arXiv preprint arXiv:2007.15588*, 2020.

F. Yan. *The Research of Information Integration Under Chinese Stock Market Impact*. PhD thesis, Tianjin University, 2012.

J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, pages 5694–5703, 2018.

C.-N. J. Yu and T. Joachims. Learning structural svms with latent variables. In *ICML*, pages 1169–1176. ACM, 2009.

A. L. Yuille, A. Rangarajan, and A. Yuille. The concave-convex procedure (cccp). *NIPS*, 2:1033–1040, 2002.

S. Zhang and S. Whiteson. Dac: The double actor-critic architecture for learning options. In *Advances in Neural Information Processing Systems*, pages 2012–2022, 2019.