ABSTRACT

WANG, LONGSHAOKAN. Sufficient Markov Decision Processes. (Under the direction of Dr. Eric Laber.)

Advances in mobile computing technologies have made it possible to monitor and apply data-driven interventions across complex systems in real time. Recent and high-profile examples of data-driven decision making include autonomous vehicles, intelligent power grids, and precision medicine through mobile health. Markov decision processes are the primary mathematical model for sequential decision problems with a large or indefinite time horizon; existing methods for estimation and inference rely critically on the correctness of this model. Mathematically, this choice of model incurs little loss in generality as any decision process evolving in discrete time with observable process states, decisions, and outcomes can be represented as a Markov decision process. However, in some application domains, e.g., mobile health, choosing a representation of the underlying decision process that is both Markov and low-dimensional is non-trivial; current practice is to select a representation using domain expertise. We propose an automated method for constructing a low-dimensional representation of the original decision process for which: (P1) the Markov decision process model holds; and (P2) a decision strategy that leads to maximal mean utility when applied to the low-dimensional representation also leads to maximal mean utility when applied to population of interest. Our approach uses a novel deep neural network to define a class of potential process representations and then searches within this class for the representation of lowest dimension which satisfies (P1) and (P2). We illustrate the proposed method using a suite of simulation experiments and application to data from a mobile health intervention targeting smoking and heavy episodic drinking among college students. Chapter 1 lays the theoretical framework for the desired dimension
reduction in Markov decision processes; Chapter 2 describes and illustrates the specific dimension reduction technique proposed; Chapter 3 relates to Chapter 2 in methodology, but tackles a different problem in text-based ordinal regression with an application in human trafficking detection.
DEDICATION

To my family.
BIOGRAPHY

Longshaokan Wang was born on October 27, 1990 in Changsha, Hunan, China to parents Heping Wang and Huanghong Long. After finishing high school in Qingdao, China, he moved to the United States to pursue higher education. He obtained his bachelors degree in Honors Mathematics from University of Notre Dame in 2013. Then he joined the Ph.D. program in Statistics at North Carolina State University where he conducted research in dimension reduction, reinforcement learning, deep learning, and artificial intelligence under the guidance of Dr. Eric Laber. He obtained an en-route masters degree in Statistics in 2015, and will graduate with his doctoral degree in Statistics in 2018.
ACKNOWLEDGEMENTS

I would like to thank my committee members for their support and insights. In particular, I would like to thank my advisor Dr. Eric Laber for providing guidance on my research, pushing me to realize my potentials, tolerating my shenanigans, and being a good friend.

I would like to thank my parents for their encouragement and understanding on my studying abroad, as I was only able to come home once a year.

I would like to thank my friends for keeping me sane on my quest for knowledge.

I would like to thank the Laber Lab members for all the enlightening brainstorming sessions.

I would like to thank Praveen Bodigutla for introducing me to the exciting world of Natural Language Processing during my internship with Amazon Conversation AI Team. The experience proved to be very helpful when I worked on the human trafficking detection project.

I would like to thank Katie Witkiewitz, Yeng Saanchi, and Cara Jones for providing the datasets used in my research.
# TABLE OF CONTENTS

**LIST OF TABLES** ................................................................. vii

**LIST OF FIGURES** .............................................................. viii

**Chapter 1** Theoretical Framework of Sufficient Markov Decision Processes ........................................... 1
  1.1 Introduction .......................................................... 1
  1.2 Setup and Notation .................................................. 3
  1.3 Theoretical Results ................................................ 6
    1.3.1 Variable Screening ........................................... 11
  1.4 Discussion .......................................................... 13

**Chapter 2** Constructing Sufficient Markov Decision Processes with Alternating Deep Neural Networks ......................... 16
  2.1 Introduction ........................................................ 16
  2.2 Method ............................................................... 19
  2.3 Simulation Experiments ........................................... 24
  2.4 Application to BASICS-Mobile ................................... 28
  2.5 Discussion .......................................................... 31
  2.6 Extension to Multi-Dimensional Arrays – Convolutional Alternating Deep Neural Networks .................. 32
    2.6.1 Background ..................................................... 33
    2.6.2 Related Work .................................................. 36
    2.6.3 Method .......................................................... 38
    2.6.4 Simulation Experiments ...................................... 41
    2.6.5 Discussion .......................................................... 46

**Chapter 3** Human Trafficking Detection with Ordinal Regression Neural Networks 47
  3.1 Introduction ........................................................ 47
  3.2 Related Work ........................................................ 49
  3.3 Method ............................................................... 52
    3.3.1 Word Embeddings ............................................. 52
    3.3.2 Gated-Feedback Recurrent Neural Network ................. 55
    3.3.3 Multi-Labeled Logistic Regression Layer .................. 58
  3.4 Experiments .......................................................... 60
    3.4.1 Datasets .......................................................... 61
    3.4.2 Comparison to Baselines ..................................... 62
    3.4.3 Ablation Test .................................................. 64
    3.4.4 Emoji Analysis ................................................ 66
  3.5 Discussion .......................................................... 68
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table 2.1</th>
<th>Comparison of feature map estimators under linear transition across different numbers of noise variables (nNoise) in terms of: marginal mean outcome using Q-learning with linear function approximation (Linear Q); Q-learning with neural network function approximation (NN Q); the number of selected variables (nVar); and the dimension of the feature map (nDim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.2</td>
<td>Comparison of feature map estimators under quadratic transition</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Comparison of feature map estimators under exponential transition</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Comparison of feature map estimators in terms of cumulative utility using DDDQ. Standard errors are included in parentheses.</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Description and distribution of labels in Trafficking10k.</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Comparison of our ordinal regression neural network (ORNN) against Immediate-Threshold ordinal logistic regression (IT), All-Threshold ordinal logistic regression (AT), Least Absolute Deviation (LAD), multi-class logistic regression (MC), and the Human Trafficking Deep Network (HTDN) in terms of Mean Absolute Error (MAE), macro-averaged Mean Absolute Error (MAEM), binary classification accuracy (Acc.) and weighted binary classification accuracy (Wt. Acc.). The results are averaged across 10-fold CV on Trafficking10k except for HTDN. The best result is highlighted in green with the 2nd best in blue.</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>Ablation test. Except for models everything is the same as Table 3.2.</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2.1</td>
<td>A basic artificial neural network with 2 hidden layers.</td>
</tr>
<tr>
<td>2.2</td>
<td>Schematic for alternating deep neural network (ADNN) model. The term “alternating” refers to the estimation algorithm which cycles over the networks for each treatment ( a \in \mathcal{A} ).</td>
</tr>
<tr>
<td>2.3</td>
<td>Relationship between ( S^t ) and ( Y^{t+1} ) in the generative model, which depends on the action. First 16 variables determine the next state. First 4 variables determine the utility.</td>
</tr>
<tr>
<td>2.4</td>
<td>Weights of the original variables in the first two components of the estimated feature map.</td>
</tr>
<tr>
<td>2.5</td>
<td>The convolution operation in ConvNets. In this example, each input feature map is ( 4 \times 4 ); each output feature map is ( 2 \times 2 ); the kernel's first two dimensions are ( 3 \times 3 ); the stride size is 1; there is no padding; the convolution takes 4 steps.</td>
</tr>
<tr>
<td>2.6</td>
<td>The architecture of Action-Conditional Video Prediction Network (Oh et al., 2015). The blue, orange, and purple arrows indicate convolution, linear transformation, and transposed convolution respectively.</td>
</tr>
<tr>
<td>2.7</td>
<td>The architecture of Convolutional Alternating Deep Neural Network.</td>
</tr>
<tr>
<td>2.8</td>
<td>The simplified architecture of Convolutional Alternating Deep Neural Network.</td>
</tr>
<tr>
<td>2.9</td>
<td>Screenshots from the games used as testbeds. From left to right, top to bottom: Catcher, Pong, Pixelcopter, FlappyBird.</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the ordinal regression neural network for text input. ( h ) represents hidden state (details explained in the following subsections).</td>
</tr>
<tr>
<td>3.2</td>
<td>The Skip-gram model architecture.</td>
</tr>
<tr>
<td>3.3</td>
<td>A basic multi-layered RNN.</td>
</tr>
<tr>
<td>3.4</td>
<td>A Gated-feedback RNN.</td>
</tr>
<tr>
<td>3.5</td>
<td>Ordinal regression layer.</td>
</tr>
<tr>
<td>3.6</td>
<td>Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis that appeared most frequently in the escort ads we scraped are shown out of the total 968 emojis that appeared. See Appendix C.5 for maps of other emojis.</td>
</tr>
<tr>
<td>C.1</td>
<td>Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 201–400 are shown; outliers were removed for better zoom.</td>
</tr>
</tbody>
</table>
Figure C.2  Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 401–600 are shown; outliers were removed for better zoom. .......................................................... 107

Figure C.3  Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 601–800 are shown; outliers were removed for better zoom. .......................................................... 108

Figure C.4  Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 801–968 are shown; outliers were removed for better zoom. .......................................................... 109
CHAPTER 1

THEORETICAL FRAMEWORK OF
SUFFICIENT MARKOV DECISION
PROCESSES

1.1 Introduction

Sequential decision problems arise in a wide range of application domains including au-
tonomous vehicles (Bagnell and Schneider, 2001), finance (Bäuerle and Rieder, 2011),
logistics (Zhang and Dietterich, 1995), robotics (Kober et al., 2013), power grids (Riedmiller
et al., 2000), and healthcare (Chakraborty and Moodie, 2013). Markov decision processes
(MDPs) (Bellman, 1957; Puterman, 2014) are the primary mathematical model for rep-
resenting sequential decision problems with an indefinite time horizon (Bertsekas and
Tsitsiklis, 1996; Sutton and Barto, 1998; Bather, 2000; Si, 2004; Powell, 2007; Wiering and Van Otterlo, 2012). This class of models is quite general as almost any decision process can be made into an MDP by concatenating data over multiple decision points (see Section 1.2 for a precise statement); however, coercing a decision process into the MDP framework in this way can lead to high-dimensional system state information that is difficult to model effectively. One common approach to construct a low-dimensional decision process from a high-dimensional MDP is to create a finite discretization of the space of possible system states and to treat the resultant process as a finite MDP (Gordon, 1995; Murao and Kita-mura, 1997; Sutton and Barto, 1998; Kamio et al., 2004; Whiteson et al., 2007). However, such discretization can result in a significant loss of information and can be difficult to apply when the system state information is continuous and high-dimensional. Another common approach to dimension reduction is to construct a low-dimensional summary of the underlying system states, e.g., by applying principal components analysis (Jolliffe, 1986), multidimensional scaling (Borg and Groenen, 1997), or by constructing a local linear embedding (Roweis and Saul, 2000). These approaches can identify a low-dimensional representation of the system state but, as we shall demonstrate, they need not retain salient features for making good decisions.

The preceding methods seek to construct a low-dimensional representation of a high-dimensional MDP with the goal of using the low-dimensional representation to estimate an optimal decision strategy, i.e., one that leads to maximal mean utility when applied to the original process; however, they offer no guarantee that the resulting process is an MDP or that a decision strategy estimated using data from the low-dimensional process will perform well when applied to the original process. We derive sufficient conditions under which a low-dimensional representation is an MDP, and that an optimal decision strategy for this low-dimensional representation is optimal for the original process. We develop a
hypothesis test for this sufficient condition based on the Brownian distance covariance (Székely et al., 2007; Székely and Rizzo, 2009) and use this test as the basis for evaluating a dimension reduction function.

In Section 1.2, we review the MDP model for sequential decision making and define an optimal decision strategy. In Section 1.3, we derive conditions under which a low-dimensional representation of an MDP is sufficient for estimating an optimal decision strategy for the original process. In Section 1.4, we discuss the challenges for designing a dimension reduction function that can produce the aforementioned low-dimensional representation, which will motivate our new deep learning algorithm described in Chapter 2.

1.2 Setup and Notation

We assume that the observed data are \( \{(S^i_1, A^i_1, U^i_1, S^i_2, \ldots, A^i_T, U^i_T, S^i_{T+1})\}_{i=1}^n \) which comprise \( n \) independent and identically distributed copies of the trajectory \( (S^1, A^1, U^1, S^2, \ldots, A^T, U^T, S^{T+1}) \) where: \( T \in \mathbb{N} \) denotes the observation time; \( S^t \in \mathbb{R}^{pt} \) denotes a summary of information collected up to time \( t = 1, \ldots, T \); \( A^t \in \mathcal{A} = \{1, \ldots, K\} \) denotes the decision made at time \( t = 1, \ldots, T \); and \( U^t = U^t(S^t, A^t, S^{t+1}) \) is a real-valued deterministic function of \( (S^t, A^t, S^{t+1}) \) that quantifies the momentary “goodness” of being in state \( S^t \), making decision \( A^t \), and subsequently transitioning to state \( S^{t+1} \). We assume throughout that \( \sup_t |U^t| \leq M \) with probability one for some fixed constant \( M \). In applications like mobile health, the observed data might be collected in a pilot study with a preset time horizon \( T \) (Maahs et al., 2012; Witkiewitz et al., 2014); however, the intent is to use these data to estimate an intervention strategy that will maximize some measure of cumulative utility

\[1\text{ often referred to as “reward” in the reinforcement learning literature} \]
when applied over an indefinite time horizon (Ertefaie, 2014; Liao et al., 2015; Luckett et al., 2016). Thus, we view the observed trajectory \((S^1, A^1, U^1, S^2, \ldots, A^T, U^T, S^{T+1})\) as a truncation of the infinite process \((S^1, A^1, U^1, S^2, \ldots)\). Furthermore, we assume (A0) that this infinite process is Markov and homogeneous in that it satisfies

\[
P\left( S^{t+1} \in G^{t+1} \mid A^t, S^t, \ldots, A^1, S^1 \right) = P\left( S^{t+1} \in G^{t+1} \mid A^t, S^t \right), \tag{1.1}
\]

for all (measurable) subsets \(G^{t+1} \subseteq \text{dom} S^{t+1}\) and \(t \in \mathbb{N}\) and that the probability measure in (1.1) does not depend on \(t\). For any process \((S^1, A^1, U^1, S^2, \ldots)\) one can define \(\tilde{S}' = (S', U^{t-1}, A^{t-1}, \ldots, S^{t-\ell})\), where \(\ell\) is chosen so that process \((\tilde{S}', A^1, U^1, \tilde{S}', \ldots)\) satisfies (A0); to see this, note that the result holds trivially for \(\ell = t - 1\). Furthermore, by augmenting the state with a variable for time, i.e., defining the new state at time \(t\) to be \((\tilde{S}', t)\), one can ensure that the probability measure in (A0) does not depend on \(t\). In practice, \(\ell\) is typically chosen to be a constant, as letting the dimension of the state grow with time makes extrapolation beyond the observed time horizon, \(T\), difficult. Thus, hereafter we assume that the domain of the state is constant over time, i.e., \(\text{dom} S^t = \mathcal{S} \subseteq \mathbb{R}^p\) for all \(t \in \mathbb{N}\). Furthermore, we assume that the utility is homogeneous in time, i.e., \(U^t = U(S^t, A^t, S^{t+1})\) for all \(t \in \mathbb{N}\).

A decision strategy, \(\pi : \mathcal{S} \rightarrow \mathcal{A}\), is a map from states to decisions so that, under \(\pi\), a decision maker presented with \(S^t = s^t\) at time \(t\) will select decision \(\pi(s^t)\). We define an optimal decision strategy using the language of potential outcomes (Rubin, 1978). We use an overline to denote history so that \(\bar{a}^t = (a^1, \ldots, a^t)\) and \(\bar{s}^t = (s^1, \ldots, s^t)\). The set of potential outcomes is \(O^* = \{S^t(\bar{a}^{t-1})\}_{t \geq 1}\) where \(S^t(\bar{a}^{t-1})\) is the potential state under \(\bar{a}^{t-1}\) and we have defined \(S^1(\bar{a}^0) = S^1\). Thus, the potential utility at time \(t\) under \(\bar{a}^t\) is \(U\left\{S^t(\bar{a}^{t-1}), a^t, S^{t+1}(\bar{a}^t)\right\}\). The
potential state under a decision strategy, \( \pi \), is

\[
S^*_{t}^{\pi} = \sum_{a_{t-1}} S^*_{t-1}^{(\overline{a}_{t-1})} \prod_{\nu=1}^{t-1} \mathbb{1}_{\nu\{S^\nu(\overline{a}^\nu)=a^\nu\}},
\]

and the potential utility under \( \pi \) is

\[
U^*_{t}^{\pi} = U\left[ S^*_{t}^{\pi}, \pi \{ S^*_{t}^{\pi} \}, S^*_{t+1}^{\pi} \right].
\]

Define the discounted mean utility under a decision strategy, \( \pi \), as

\[
V(\pi) = \mathbb{E}\left\{ \sum_{t \geq 1} \gamma^{t-1} U^*_{t}^{\pi} \right\},
\]

where \( \gamma \in (0, 1) \) is a discount factor that balances the trade-off between immediate and long-term utility. Given a class of decision strategies, \( \Pi \), an optimal decision strategy, \( \pi^{opt} \in \Pi \), satisfies

\[
V(\pi^{opt}) \geq V(\pi)
\]

for all \( \pi \in \Pi \).

Define the action selection probability during data generation as

\[
\mu^t(a^t; \overline{s}_t, \overline{a}^{t-1}) = P\left( A^t = a^t \mid \overline{S}^t = \overline{s}_t, \overline{A}^{t-1} = \overline{a}^{t-1} \right).
\]

To characterize \( \pi^{opt} \) in terms of the data-generating model, we make the following assumptions for all \( t \in \mathbb{N} \):

(C1) consistency, \( S^t = S^*_{t}(\overline{A}^{t-1}) \);

(C2) positivity, there exists \( \epsilon > 0 \) such that \( \mu^t(a^t; \overline{s}_t, \overline{A}^{t-1}) \geq \epsilon \) with probability one for all \( a^t \in \mathcal{A}^t \);

(C3) sequential ignorability, \( O^t \perp A^t \mid \overline{S}^t, \overline{A}^{t-1} \).
These assumptions are standard in data-driven decision making (Robins, 2004; Schulte et al., 2014). Assumptions (C2) and (C3) hold by design in a randomized trial (Liao et al., 2015; Klasnja et al., 2015) but are not verifiable in the data for observational studies. Under these assumptions, the joint distribution of \( \{S^* t(\pi)\}_{t=1}^T \) is non-parametrically identifiable under the data-generating model for any decision strategy \( \pi \) and time horizon \( T \). In our application, these assumptions will enable us to construct low-dimensional features of the state that retain all relevant information for estimating \( \pi^{opt} \) without having to solve the original MDP as an intermediate step.

1.3 Theoretical Results

If the states \( S^t \) are high-dimensional it can be difficult to construct a high-quality estimator of the optimal decision strategy; furthermore, in applications like mobile health, storage and computational resources on the mobile device are limited, making it desirable to store only as much information as is needed to inform decision making. For any map \( \phi : \mathcal{S} \to \mathbb{R}^q \) define \( S^t_{\phi} = \phi(S^t) \). We say that \( \phi \) induces a sufficient MDP for \( \pi^{opt} \) if \( (A^t, S^{t+1}_{\phi}, U^t) \) contains all relevant information in \( (A^t, S^{t+1}, U^t) \) about \( \pi^{opt} \). Given a policy \( \pi_{\phi} : \text{dom} S^t_{\phi} \to \mathcal{A} \) define the potential utility under \( \pi_{\phi} \) as

\[
U^{st}_{\phi}(\pi_{\phi}) = \sum_{\bar{a}^t} U \{ S^{st}(\bar{a}^{t-1}), a^t, S^{st+1}(\bar{a}^t) \} \prod_{v=1}^t \pi_{\phi} \{ S^v_{\phi}(\bar{a}^{v-1}) = a^v \}.
\]

The following definition formalizes the notion of inducing a sufficient MDP.

**Definition 1.3.1.** Let \( \Pi \subseteq \mathcal{A}^{\mathcal{S}} \) denote a class of decision strategies defined on \( \mathcal{S} \) and \( \Pi_{\phi} \subseteq \mathcal{A}^{\mathcal{S}_{\phi}} \) a class of decision strategies defined on \( \mathcal{S}_{\phi} = \text{dom} S^t_{\phi} \subseteq \mathbb{R}^q \). We say that the pair \( (\phi, \Pi_{\phi}) \) induces a sufficient MDP for \( \pi^{opt} \) within \( \Pi \) if the following conditions hold for all
$t \in \mathbb{N}$: 

(S1) the process $(\tilde{A}_t, \tilde{S}_t^\phi, \tilde{U}_t)$ is Markov and homogeneous, i.e.,

$$
P(S_{t+1}^\phi \in G_{t+1}^\phi | \tilde{S}_t^\phi, \tilde{A}_t) = P(S_{t+1}^\phi \in G_{t+1}^\phi | S_t^\phi, A_t)$$

for any (measurable) subset $G_{t+1}^\phi \subseteq \mathbb{R}^q$ and this probability does not depend on $t$;

(S2) there exists $\pi_{opt} \in \arg \max_{\pi \in \Pi} V(\pi)$ which can be written as $\pi_{opt} = \pi_{opt}^\phi \circ \phi$, where

$$
\pi_{opt}^\phi \in \arg \max_{\pi \in \Pi_{\phi}} \mathbb{E} \left\{ \sum_{t \geq 1} r_{t-1} U_{t-1}^{\phi}(\pi_{\phi}) \right\} .
$$

Thus, given the observed data, $\{ (\tilde{S}_t^\phi, \tilde{A}_t, \tilde{U}_t) \}_{i=1}^n$ and class of decision strategies, $\Pi$, if one can find a pair $(\phi, \Pi_{\phi})$ which induces a sufficient MDP for $\pi_{opt}^\phi$ within $\Pi$, then it suffices to store only the reduced process $\{ (\tilde{S}_i^\phi, \tilde{A}_i^T, \tilde{U}_i) \}_{i=1}^n$. Furthermore, existing reinforcement learning algorithms (see Appendix A.1 and e.g., Sutton and Barto, 1998; Szepesvári, 2010) can be applied to this reduced process to construct an estimator of $\pi_{opt}^\phi$ and hence $\pi_{opt} = \pi_{opt}^\phi \circ \phi$.

If the dimension of $S_t^\phi$ is substantially smaller than that of $S_t$, then using the reduced process can lead to smaller estimation error as well as reduced storage and computational costs. In some applications, it may also be desirable to have $\phi$ be a sparse function of $S_t$ in the sense that it only depends on a subset of the components of $S_t$. For example, in the context of mobile health, one may construct the state, $S_t$, by concatenating measurements taken at time points $t, t-1, \ldots, t-m$, where the look-back period, $m$, is chosen conservatively based on clinical judgement to ensure that the process is Markov; however, a data-driven sparse feature map might identify that a look-back period of $m' \ll m$ is sufficient thereby reducing computational and memory requirements but also generating new knowledge that may be
of clinical value. The remainder of this section will focus on developing verifiable conditions for checking that \((\phi, \Pi_{\phi})\) induces a sufficient MDP. These conditions are used to build a data-driven, low-dimensional, and potentially sparse sufficient MDP.

Define \(Y_{t+1} = \{U^t, (S^{t+1})^T\}^T\) for all \(t \in \mathbb{N}\). The following result provides a conditional independence criterion that ensures a given feature map induces a sufficient MDP; this criterion can be seen as an MDP analog of nonlinear sufficient dimension reduction in regression (Cook, 2007; Li et al., 2011). A proof is provided in Appendix A.2.

**Theorem 1.3.2.** Let \((S^1, A^1, U^1, S^2, \ldots)\) be an MDP that satisfies (A0) and (C1)-(C3). Suppose that there exists \(\phi : \mathcal{S} \rightarrow \mathbb{R}^q\) such that

\[
Y_{t+1} \perp \perp S_t|S_\phi^t, A',
\]

then, \((\phi, \Pi_{\phi, \text{msbl}})\) induces a sufficient MDP for \(\pi^{\text{opt}}\) within \(\Pi_{\text{msbl}}\), where \(\Pi_{\text{msbl}}\) is the set of measurable maps from \(\mathcal{S}\) into \(\mathcal{A}\) and \(\Pi_{\phi, \text{msbl}}\) is the set of measurable maps from \(\mathbb{R}^q\) into \(\mathcal{A}\).

The preceding result could be used to construct an estimator for \(\phi\) so that \((\phi, \Pi_{\phi, \text{msbl}})\) induces a sufficient MDP for \(\pi^{\text{opt}}\) within \(\Pi_{\text{msbl}}\) as follows. Let \(\Phi\) denote a potential class of vector-valued functions on \(\mathcal{S}\). Let \(\hat{p}_n(\phi)\) denote a p-value for a test of the conditional independence criterion (1.2) based on the mapping \(\phi\), e.g., one might construct this p-value using conditional Brownian distance correlation (Wang et al., 2015) or kernel-based tests of conditional independence (Fukumizu et al., 2007). Then, one could select \(\hat{\phi}_n\) to be the transformation of lowest dimension among those within the set \(\{\phi \in \Phi : \hat{p}_n(\phi) < \tau\}\), where \(\tau\) is a fixed significance level, e.g., \(\tau = 0.10\). However, such an approach can be computationally burdensome especially if the class \(\Phi\) is large. Instead, we will develop a procedure based on a series of unconditional tests that is computationally simpler and allows for a flexible class of potential transformations. Before presenting this approach, we
first describe how the conditional independence criterion in the above theorem can be applied recursively to potentially produce a sufficient MDP of lower dimension.

The condition $Y_{t+1} \perp S_t | S_{\phi}^t, A^t$ is overly stringent in that it requires $S_{\phi}^t$ to capture all the information about $Y_{t+1}$ contained within $S_t$ regardless of whether or not that information is useful for decision making. However, given a sufficient MDP $(S_1^1, A_1^1, U_1, S_2^2, \ldots)$, one can apply the above theorem to this MDP to obtain further dimension reduction; this process can be iterated until no further dimension reduction is possible. For any map $\phi : \mathcal{S} \rightarrow \mathbb{R}^q$, define $Y_{\phi}^t = \{U^t, (S_{\phi}^{t+1})^T\}^T$. The following result is proved in Appendix A.3.

**Corollary 1.3.3.** Let $(S^1, A^1, U^1, S^2, \ldots)$ be an MDP that satisfies (A0) and (C1)-(C3). Assume that there exists $\phi_0 : \mathcal{S} \rightarrow \mathbb{R}^{q_0}$ such that $(\phi_0, \Pi_{\phi_0, \text{msrbl}})$ induces a sufficient MDP for $\pi_{\text{opt}}$ within $\Pi_{\text{msrbl}}$. Suppose that there exists $\phi_1 : \mathbb{R}^{q_0} \rightarrow \mathbb{R}^{q_1}$ such that for all $t \in \mathbb{N}$

$$Y_{\phi_0}^t \perp S_t^t | S_{\phi_0}^t, A^t,$$  \hspace{1cm} (1.3)

then $(\phi_1 \circ \phi_0, \Pi_{\phi_1 \circ \phi_0, \text{msrbl}})$ induces a sufficient MDP for $\pi_{\text{opt}}$ within $\Pi_{\text{msrbl}}$. Furthermore, for $k \geq 2$, denoting $\phi_k \circ \phi_{k-1} \circ \cdots \circ \phi_0$ by $\overline{\phi_k}$, if there exists $\phi_k : \mathbb{R}^{q_{k-1}} \rightarrow \mathbb{R}^{q_k}$ such that

$$Y_{\overline{\phi_k}}^{t+1} \perp S_t^t | S_{\overline{\phi_k}}^t, A^t,$$

then $(\overline{\phi_k}, \Pi_{\overline{\phi_k}, \text{msrbl}})$ induces a sufficient MDP for $\pi_{\text{opt}}$ within $\Pi_{\text{msrbl}}$.

We now state a simple condition involving the residuals of a multivariate regression that can be used to test the conditional independence required in each step of the preceding corollary. In our implementation we use residuals from a variant of deep neural networks that is suited to sequential decision problems (see Section 4). The following result is proved in Appendix A.4.
Lemma 1.3.4. Let \((S^1, A^1, U^1, S^2, \ldots)\) be an MDP that satisfies (A0) and (C1)-(C3). Suppose that there exists \(\phi : \mathcal{S} \rightarrow \mathbb{R}^q\) such that at least one of the following conditions holds:

\[
(i) \left\{ Y^{t+1} - E \left( Y^{t+1} | S^t, A^t \right) \right\} \perp \perp S^t | A^t,
\]

\[
(ii) \left\{ S^t - E \left( S^t | S^t_\phi \right) \right\} \perp \perp \left( Y^{t+1}, S^t_\phi \right) | A^t,
\]

then \(Y^{t+1} \perp \perp S^t | S^t_\phi, A^t\).

The preceding result can be used to verify the conditional independence condition required by Theorem (1.3.2) and Corollary (1.3.3) using unconditional tests of independence within levels of \(A^t\); in our simulation experiments, we used Brownian distance covariance for continuous states (Székely et al., 2007; Székely and Rizzo, 2009) and a likelihood ratio test for discrete states, though other choices are possible (Gretton et al., 2005a,b). Application of these tests requires modification to account for dependence over time within each subject. One simple approach, the one we follow here, is to compute a separate test at each time point and then to pool the resultant p-values using a pooling procedure that allows for general dependence. For example, let \(G^t = g(S^t, A^t, S^{t+1}) \in \mathbb{R}^{d_1}\) and \(H^t = h(S^t) \in \mathbb{R}^{d_2}\) be known features of \((S^t, A^t, S^{t+1})\) and \(S^t\). Let \(\mathbb{P}_n\) denote the empirical measure. To test \(G^t \perp \perp H^t\) using the Brownian distance covariance, we compute the test statistic

\[
\hat{T}_n = \| \mathbb{P}_n \exp \{ i (\zeta^T G^t + \varphi^T H^t) \} - \mathbb{P}_n \exp \{ i \zeta^T G^t \} \mathbb{P}_n \exp \{ i \varphi^T G^t \} \|_\omega^2
\]

\[
= \int \frac{\left[ \mathbb{P}_n \exp \{ i (\zeta^T G^t + \varphi^T H^t) \} - \mathbb{P}_n \exp \{ i \zeta^T G^t \} \mathbb{P}_n \exp \{ i \varphi^T H^t \} \right]^2 \Gamma \left( \frac{1+d_1}{2} \right) \Gamma \left( \frac{1+d_2}{2} \right) d \varphi d \zeta}{\| \zeta \|^{d_1+1} \| \varphi \|^{d_2+1} \pi^{(d_1+d_2+2)/2}}
\]

and subsequently compute the p-value, say \(\tilde{p}_n\), using the null distribution of \(\hat{T}_n\) as estimated by permutation (see Appendix A.7 for the algorithm, and Székely et al., 2007; Székely and Rizzo, 2009, for more details). For each \(u = 1, \ldots, T\), let \(\tilde{p}_n^{(u)}\) denote the \(u\)th order statistic of.
\(\hat{p}_n^1, \ldots, \hat{p}_n^T\) and define the pooled \(p\)-value

\[
\hat{p}_{n, \text{pooled}}^u = T \frac{\hat{p}_n^{(u)}}{u}.
\]

For each \(u = 1, \ldots, T\) it can be shown that \(\hat{p}_{n, \text{pooled}}^u\) is valid \(p\)-value (Rüger, 1978), e.g., \(u = 1\) corresponds to the common Bonferroni correction. In our simulation experiments, we set \(u = \lceil T/20 + 1 \rceil\) across all settings (see Appendix A.8 for more details).

### 1.3.1 Variable Screening

The preceding results provide a pathway for constructing sufficient MDPs. However, while the criteria given in Theorem 1.3.2 and Lemma 1.3.4 can be used to identify low-dimensional structure in the state, they cannot be used to eliminate certain simple types of noise variables. For example, let \(\{B_t^{(i)}\}_{t \geq 1}\) denote a homogeneous Markov process that is independent of \((S_1^1, A_1^1, U_1^1, S_2^2, \ldots)\), and consider the augmented process \((\tilde{S}_1^1, A_1^1, U_1^1, \tilde{S}_2^2, \ldots)\), where \(\tilde{S}_t^i = \{(S_t^i)^\top, (B_t^i)^\top\}^\top\). Clearly, the optimal policy for the augmented process does not depend on \(\{B_t^i\}_{t \geq 1}\), yet, \(Y_{t+1}\) need not be conditionally independent of \(\tilde{S}_t^i\) given \(S_t^i\). To remove variables of this type, we develop a simple screening procedure that can be applied prior to constructing nonlinear features as described in the next section.

The proposed screening procedure is based on the following result which is proved in Appendix A.5.

**Theorem 1.3.5.** Let \((S^1, A^1, U^1, S^2, \ldots)\) be an MDP that satisfies (A0) and (C1)-(C3). Suppose that there exists \(\phi : \mathcal{X} \to \mathbb{R}^q\) such that

\[
Y_{t+1}^{i+1} \perp \perp S_t^i | S_{\phi t}^i, A_t^i,
\]

(1.4)
then, \((\phi, \Pi_{\phi, \text{msbl}})\) induces a sufficient MDP for \(\pi^{\text{opt}}\) within \(\Pi_{\text{msbl}}\), where \(\Pi_{\text{msbl}}\) is the set of measurable maps from \(\mathcal{S}\) into \(\mathcal{A}\) and \(\Pi_{\phi, \text{msbl}}\) is the set of measurable maps from \(\mathbb{R}^q\) into \(\mathcal{A}\).

This result can be viewed as a stronger version of Theorem 1.3.2 in that the required conditional independence condition is weaker; indeed, in the example stated above, it can be seen that \(\phi(\hat{S}^t) = S^t\) satisfies (1.4). However, because \(\phi\) appears in both \(Y_{\phi}^{t+1}\) and \(S_\phi^t\), constructing nonlinear features using this criterion is more challenging as the residual-based conditions stated in Lemma 1.3.4 can no longer be applied. Nevertheless, this criterion turns out to be ideally suited to screening procedures wherein the functions \(\phi: \mathbb{R}^p \to \mathbb{R}^q\) are of the form \(\phi(s)_j = s_{k_j}^j\) for \(j = 1, \ldots, q\), where \(\{k_1, \ldots, k_q\}\) is a subset of \(\{1, \ldots, p\}\).

For any subset \(J \subseteq \{1, \ldots, p\}\), define \(S'_j = \{S'_j\}_{j \in J}\) and \(Y'_j = \{Y_t, (S'_j)^\top\}\). Let \(J_1\) denote the smallest set of indices such that \(U^t\) depends on \(S^t\) and \(S^{t+1}\) only through \(S'_j\) and \(S^{t+1}_j\) conditioned on \(A^t\). For \(k \geq 2\), define \(J_k = \{1 \leq j \leq p : S'_j \not\perp \perp Y_{j-1}^{t+k-1} | A^t\}\). Let \(K\) denote the smallest value for which \(J_{K-1} = J_K\), such a \(K\) must exist as \(J_{k-1} \subseteq J_k\) for all \(k\), and define \(\phi_{\text{screen}}(S^t) = S'_{J_K}\). The following result (see Appendix A.6 for a proof) shows that \(\phi_{\text{screen}}\) induces a sufficient MDP; furthermore, Corollary 1.3.3 shows that such screening can be applied before nonlinear feature construction without destroying sufficiency.

**Theorem 1.3.6.** Let \((S^1, A^1, U^1, S^2, \ldots)\) be an MDP that satisfies \((A0)\) and \((C1)-(C3)\), and let \(J_1, \ldots, J_K, \phi_{\text{screen}}\) be as defined above. Assume that for any two non-empty subsets, \(J, J' \subseteq \{1, \ldots, p\}\), if \(S'_j \not\perp \perp Y_{j'}^{t+1} | A^t\) then there exists \(j \in J\) such that \(S'_j \not\perp \perp Y_{j'}^{t+1} | A^t\). Then,

\[
Y_{\phi_{\text{screen}}}^{t+1} \perp S^t | S'_{\phi_{\text{screen}}}, A^t.
\]

The condition that joint dependence implies marginal dependence (or equivalently, marginal independence implies joint independence) ensures that screening one variable at a time will identify the entire collection of important variables; this condition could be weakened.
by considering sets of multiple variables at a time though at the expense of additional computational burden. Algorithm 1 gives a schematic for estimating $\phi_{\text{screen}}$ using the Brownian distance covariance to test for dependence. The inner for-loop (lines 4-7) of the algorithm can be executed in parallel and thereby scaled to large domains.

Algorithm 1 Screening with Brownian Distance Covariance

**Input:** p-value threshold $\tau$; max number of iterations $N_{\text{max}}$; data $\left\{\left(\mathbf{S}_i^{T+1}, \mathbf{A}_i^T, \mathbf{U}_i^T\right)\right\}_{i=1}^n$; set of all indices $D = \{1, 2, \ldots, p = \dim(S')\}$.

1: Set $J_0 = \emptyset$, and $Y_{T+1}^{\tau} = \{U^T\}$
2: for $k = 1, \ldots, N_{\text{max}}$ do
3: Set $J_k = J_{k-1}$
4: for each $j \in D \setminus J_{k-1}$ do
5: Perform dCov test on $S^T_j$ and $Y_{T+1}^{\tau}^{J_{k-1}}$ within levels of $A^T$
6: if p-value $\leq \tau$ then
7: Set $J_k = J_k \cup \{j\}$
8: if $J_k = J_{k-1}$ then
9: Set $K = k$, stop.

**Output:** $J_K$

### 1.4 Discussion

Theorem 1.3.2, Corollary 1.3.3, and Theorem 1.3.5 provide general conditions that a dimension reduction function $\phi$ should meet to retain all relevant information regardless of the implementations of hypothesis tests for those conditions or how $\hat{\phi}$ is constructed.

We presented a Brownian distance covariance based test for checking the conditions in Section 1.3, and we will present our deep neural network algorithm for constructing $\hat{\phi}$ in Chapter 2. But before we present the algorithm, it is worth discussing both the challenges of constructing a $\hat{\phi}$ that can yield a sufficient MDP and the desired properties such a $\hat{\phi}$
Condition 1.2 suggests that in the context of MDPs, a sufficient dimension reduction needs to retain the information in \( S_t \) relevant to \( Y_{t+1} \). Thus, the goal is very different from the goal of common dimension reduction techniques, which is usually to retain information about the original input, \( S_t \) in this case. Common techniques like principal components analysis (PCA) (Jolliffe, 1986) and autoencoder (Bengio, 2009; Vincent et al., 2010) can discard redundant information from \( S_t \), e.g., when \( S_t \) has some highly correlated variables, but they are not designed to discard noisy information irrelevant to \( Y_{t+1} \). Imagine that \( S_t \) has a few independent variables that don't affect \( Y_{t+1} \) at all. PCA and autoencoder will keep the information of these variables, because they contribute to the variance of \( S_t \) and cannot be reconstructed from the other variables in \( S_t \). In general, any dimension reduction technique that doesn't evaluate the relationship between \( S_t \) and \( Y_{t+1} \) is unlikely to be efficient.

Therefore, it seems the most reasonable approach to perform supervised learning and build a model that predicts \( Y_{t+1} \) from \( S_t \), and then obtain \( \phi \) from the predictive model. For example, if the model is a deep neural network (DNN) (LeCun et al., 2015) whose input is \( S_t \) and output is \( \mathbb{E}[Y_{t+1} \mid S_t] \), then \( \phi(S_t) \) could be the the output from an intermediate/hidden layer with lower dimension than \( S_t \).

However, another challenge arises: The relationship between \( S_t \) and \( Y_{t+1} \) often depends on \( A_t \), and the relevant information for \( Y_{t+1} \) in \( S_t \) could be different for different levels of \( A_t \). So a good predictive model will predict \( \mathbb{E}[Y_{t+1} \mid S_t, A_t] \) differently for each level of \( A_t \). In the mean time, the computation of \( \hat{\phi}(S_t) \) cannot depend on \( A_t \). \( \hat{\phi}(S_t) \) should contain the union of relevant information for \( Y_{t+1} \) for all possible levels of \( A_t \), because eventually we wish to apply the estimated optimal policy to new data, and use \( a^* = \arg\max_a Q^*(\phi(s'), a) \) for action selection. Naturally, we need \( \phi(S_t) \) before knowing \( A_t \).

In short, the dilemma is: We wish to obtain \( \phi(S_t) \) from a predictive model for \( Y_{t+1} \);
the predictive model should capture the relationship between $S^t$ and $Y^{t+1}$ unique to $A^t$ by making different predictions accordingly; $\phi(S^t)$ should be the same and contain the relevant information for $Y^{t+1}$ no matter what $A^t$ turns out to be. Going back to the DNN example where we let $\hat{\phi}(S^t)$ be the output from a hidden layer, if we train a different network for each level of $A^t$ or include $A^t$ in the input, the computation of $\hat{\phi}(S^t)$ will depend on $A^t$; if we completely ignore $A^t$, then the network will try to learn the potentially conflicting $A^t$-specific relationships between $S^t$ and $Y^{t+1}$ all at once, making the network intrinsically unstable. This motivates us to design a new deep learning algorithm, Alternating Deep Neural Networks (see Chapter 2), that can learn $A^t$-specific relationships separately while constructing an $A^t$-indifferent $\hat{\phi}$ which aggregates all the relevant information for $Y^{t+1}$. 
2.1 Introduction

Deep neural networks (DNNs) have recently become a focal point in machine learning research because of their ability to identify complex and nonlinear structure in high-dimensional data (see Anthony and Bartlett, 2009; LeCun et al., 2015; Goodfellow et al., 2016, and references therein). Thus, such models are ideally suited for nonlinear feature
construction. To address the challenges mentioned in Section 1.4, we propose a novel variation of DNNs. Before explaining the details of our algorithm, we will briefly introduce DNNs.

DNNs are essentially artificial neural networks (ANNs) with many hidden layers and various improvements (see Goodfellow et al., 2016, and references therein). ANNs are inspired by biological neural networks. An ANN model is usually composed of an input layer, one or several hidden layers, and an output layer (Figure 2.1). Data are propagated forward through the network via linear transformations between layers followed by possibly non-linear activation functions within each layer to obtain the output. More precisely, for the $l$-th layer, let $O^{l-1} \in \mathbb{R}^{p \times 1}$ be the input, $w^l \in \mathbb{R}^{q \times p}$ be the weight matrix, $b^l \in \mathbb{R}^{q \times 1}$ be the intercept vector\(^1\), $\xi^l$ be the element-wise activation function which is typically sigmoid or rectified linear unit, and $O^l \in \mathbb{R}^{q \times 1}$ be the output, then

$$O^l = \xi^l(w^l \cdot O^{l-1} + b^l).$$

Let $X_i = O^1_i$ be the $i$-th initial input, $\hat{Y}_i = O^L_i$ be the $i$-th final output, $\theta$ be the set of all parameters. The network is trained to minimize a cost function, typically the root mean squared error of predictions:

$$C(\theta) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \|Y_i - \hat{Y}_i\|^2_2}.$$

The gradient of the cost function is propagated backwards through the network to update all the parameters using Stochastic Gradient Descent or its variations.

\(^1\)also referred to as the bias vector in the literature.
Figure 2.1 A basic artificial neural network with 2 hidden layers.

...tion and better performance than “shallow” ANNs (Bengio, 2009). However, because the gradients of early layers are products of the gradients of the subsequent layers by the chain rule, they tend to vanish or explode when the number of layers is increased naively, making DNNs difficult to train (Bengio, 2009). Researchers have developed a variety of methods to enable the training and enhance the performance of DNNs, such as layer by layer pretraining (Hinton and Salakhutdinov, 2006; Vincent et al., 2010), using Convolutional Network for image data (Nielsen, 2015), performing $L_1$ or $L_2$ regularizations, adding drop-out to hidden layers (Srivastava et al., 2014) and so on. The advancement of computer hardware, especially Graphical Processing Units, also played a big role in the rise of popularity of DNNs (LeCun et al., 2015). On top of some of the aforementioned enhancements, we add a new structure and training procedure to DNNs to construct the dimension reduction function $\hat{\phi}$, so that we can solve the dilemma that $\hat{\phi}$ should not depend on $A^t$ while the prediction $\hat{E}[Y^{t+1}|S^t, A^t]$ should.

In Section 2.2, we describe our proposed algorithm that is designed to produce a low-dimensional representation that satisfies the proposed sufficient condition from Theorem 1.3.2. In Section 2.3, we evaluate the performance of the proposed method in a suite of...
simulation experiments. In Section 2.4, we illustrate the proposed method using data from a study of a mobile health intervention targeting smoking and heavy episodic drinking among college students (Witkiewitz et al., 2014). In Section 2.5, we discuss some shortcomings of the main algorithm and several directions it can be extended. In Section 2.6, we complete one such extension that allows our algorithm to handle states in the form of multi-dimensional arrays such as image frames in video games.

2.2 Method

We inherit the notations on Markov decision processes (MDPs) from Chapter 1. Section 2.1 describes deep neural networks (DNNs) on a high level with notations commonly used in the literature; in this section we will re-describe DNNs with slightly different notations more suitable in our feature construction/dimension reduction framework. For simplicity, we assume that $\mathcal{S} = \mathbb{R}^p$. We consider summary functions $\phi: \mathbb{R}^p \to \mathbb{R}^q$ that are representable as DNNs. Here, we present a novel neural network architecture for estimating sufficient MDPs.

We use criterion (i) in Lemma 1.3.4 to construct a data-driven summary function $\phi$, therefore we also require a model for the regression of $Y_{t+1}$ on $S_t^\phi$ and $A^t$; we also use a DNN for this predictive model. Thus, the model can be visualized as two DNNs: one that composes the feature map $\phi$ and another that models the regression of $Y_{t+1}$ on $S_t^\phi$ and $A^t$. A schematic for this model is displayed in Figure 2.2. Let $\Phi: \mathbb{R} \to [0, 1]$ denote a continuous and monotone increasing function and write $\Phi^\circ$ to denote the vector-valued function obtained by elementwise application of $\Phi$, i.e., $\Phi^\circ(v) = \Phi(v_j)$ where $v \in \mathbb{R}^d$. The neural network for the feature map is parameterized as follows. Let $r_1, \ldots, r_M, \in \mathbb{N}$ be such
that \( r_1 = p \). The first layer of the feature map network is

\[
\mathcal{L}_1(s; \Sigma_1, \eta_1) = \Phi^\circ (\Sigma_1 s + \eta_1),
\]

where \( \Sigma_1 \in \mathbb{R}^{r_2 \times r_1} \) and \( \eta_1 \in \mathbb{R}^{r_2} \). Recursively, for \( k = 2, \ldots, M_1 \), define

\[
\mathcal{L}_k(s; \Sigma_k, \eta_k, \ldots, \Sigma_1, \eta_1) = \Phi^\circ \left( \Sigma_k \mathcal{L}_{k-1}(s; \Sigma_{k-1}, \eta_{k-1}, \ldots, \Sigma_1, \eta_1) + \eta_k \right),
\]

where \( \Sigma_k \in \mathbb{R}^{r_k \times r_{k-1}} \) and \( \eta_k \in \mathbb{R}^{r_k} \). Let \( \theta_1 = (\Sigma_{M_1}, \eta_{M_1}, \ldots, \Sigma_1, \eta_1) \), then the feature map under \( \theta_1 \) is

\[
\phi(s; \theta_1) = \mathcal{L}_{M_1}(s; \theta_1) = \mathcal{L}_{M_1}(s; \Sigma_{M_1}, \eta_{M_1}, \ldots, \Sigma_1, \eta_1).
\]

Thus, the dimension of the feature map is \( r_{M_1} \). The neural network for the regression of \( Y_{t+1} \) on \( S_t^\phi \) and \( A_t \) is as follows. Let \( r_{M_1+1}, \ldots, r_{M_1+M_2} \in \mathbb{N} \) be such that \( r_{M_1+M_2} = p + 1 \). For each \( a \in \mathcal{A} \) define

\[
\mathcal{L}_{M_1+1,a}(s; \theta_1, \Sigma_{M_1+1,a}, \eta_{M_1+1,a}) = \Phi^\circ \left( \Sigma_{M_1+1,a} \phi(s; \theta_1) + \eta_{M_1,a} \right),
\]

where \( \Sigma_{M_1+1,a} \in \mathbb{R}^{r_{M_1+1} \times r_{M_1}} \) and \( \eta_{M_1,a} \in \mathbb{R}^{r_{M_1+1}} \). Recursively, for \( k = 2, \ldots, M_2 \) and each \( a \in \mathcal{A} \) define

\[
\mathcal{L}_{M_1+k,a}(s; \theta_1, \Sigma_{M_1+k,a}, \eta_{M_1+k,a}, \ldots, \Sigma_{M_1+1,a}, \eta_{M_1+1,a}) = \Phi^\circ \left( \Sigma_{M_1+k,a} \mathcal{L}_{M_1+k-1,a}(s; \theta_1, \Sigma_{M_1+k-1,a}, \eta_{M_1+k-1,a}, \ldots, \Sigma_{M_1+1,a}, \eta_{M_1+1,a}) + \eta_{M_1+k,a} \right),
\]

where \( \Sigma_{M_1+k,a} \in \mathbb{R}^{r_{M_1+k} \times r_{M_1+k-1}} \) and \( \eta_{M_1+k} \in \mathbb{R}^{r_{M_1+k}} \). For each \( a \in \mathcal{A} \), define

\[
\theta_{2,a} = \left( \Sigma_{M_1+M_2,a}, \eta_{M_1+M_2,a}, \ldots, \Sigma_{M_1+1,a}, \eta_{M_1+1,a} \right),
\]

20
and write \(\theta_2 = \{\theta_{2,a}\}_{a \in \mathcal{A}}\). The postulated model for \(E(Y_{t+1}^{\phi} | S_t^{\phi} = s^{\phi}_t, A^t = a^t)\) under parameters \((\theta_1, \theta_2)\) is \(\mathcal{L}_{M_1 + M_2}(s; \theta_1, \theta_{2,a})\).

We use penalized least squares to construct an estimator of \((\theta_1, \theta_2)\). Let \(\mathbb{P}_n\) denote the empirical measure and define

\[
C_n^\lambda(\theta_1, \theta_2) = \mathbb{P}_n \sum_{t=1}^T \|\mathcal{L}_{M_1 + M_2}(S^t_\phi, \theta_1, \theta_{2,A^t}) - Y_t^{\phi+1}\|_2^2 + \lambda \sum_{j=1}^{r_1} \left( \sum_{\ell=1}^{r_2} \sum_{j=1}^{\Sigma_1^{2,\ell,j}} \right), \tag{2.1}
\]

and subsequently \((\hat{\theta}_1^\lambda, \hat{\theta}_2^\lambda) = \arg\min_{\theta_1, \theta_2} C_n^\lambda(\theta_1, \theta_2)\), where \(\lambda > 0\) is a tuning parameter. The term \(\sqrt{\sum_{\ell=1}^{r_2} \sum_{j=1}^{\Sigma_1^{2,\ell,j}}}\) is a group-lasso penalty (Yuan and Lin, 2006) on the \(\ell\)th column of \(\Sigma_1\); if the \(\ell\)th column of \(\Sigma_1\) shrunk to zero then \(S_\phi^t\) does not depend on the \(\ell\)th component of \(S^t\). Computation of \((\hat{\theta}_1^\lambda, \hat{\theta}_2^\lambda)\) also requires choosing values for \(\lambda, M_1, M_2,\) and \(r_2, r_3, r_{M_1-1}, r_{M_1+1}, \ldots, r_{M_1+M_2-1}\), (recall that \(r_1 = p, r_{M_1+M_2} = p+1\), and \(r_{M_1}\) is the dimension of the feature map and is therefore considered separately). Tuning each of these parameters individually can be computationally burdensome, especially when \(M_1 + M_2\) is large. In our implementation, we assumed \(r_2 = r_3 = \cdots = r_{M_1-1} = r_{M_1+1} = \cdots = r_{M_1+M_2-1} = K_1\) and \(M_1 = M_2 = K_2\); then, for each fixed value of \(r_{M_1}\) we selected \((K_1, K_2, \lambda)\) to minimize cross-validated cost. Algorithm 2 shows the process for fitting this model; the algorithm uses subsampling to improve stability of the underlying sub-gradient descent updates (this is also known as taking minibatches, see Lab, 2014; Goodfellow et al., 2016, and references therein).

To select the dimension of the feature map we choose the lowest dimension for which the Brownian distance covariance test of dependence between \(Y_{t+1}^{\phi} - \mathcal{L}_{M_1 + M_2}(S^t_\phi; \hat{\theta}_1^\lambda, \hat{\theta}_{2,A^t,n})\) and \(S^t\) fails to reject at a pre-specified error level \(\tau \in (0, 1)\). Let \(\hat{\phi}_n^1\) be the estimated feature
**Algorithm 2** Alternating Deep Neural Networks

**Input:** Tuning parameters $K_1, K_2 \in \mathbb{N}$, $\lambda \geq 0$; feature map dimension $r_1$; data $\left\{ \left( \mathbf{S}^T_{i}, \mathbf{A}^T_{i}, \mathbf{U}^T_{i} \right) \right\}_{i=1}^{n}$; batch size proportion $\nu \in (0, 1)$; gradient-descent step-size $\{\alpha_b\}_{b \geq 1}$; error tolerance $\varepsilon > 0$; max number of iterations $N_{\text{max}}$; and initial parameter values $\hat{\theta}_{1,n}^{(1)}, \hat{\theta}_{2,n}^{(1)}$.

1: Set $D_a = \{(i, t) : A_i^t = a\}$ and $n_a = \#D_a$ for each $a \in \mathcal{A}$ and $t = 1, \ldots, T$

2: for $b = 1, \ldots, N_{\text{max}}$ do

3: for each $a \in \mathcal{A}$ do

4: Draw a random batch $B_a$ of size $\lfloor \nu n_a \rfloor$ without replacement from $D_a$

5: Compute a sub-gradient of the cost on batch $B_a$

$$\Lambda_a^{(b)} = \nabla \left[ \frac{1}{\nu n_a} \sum_{(i, t) \in B_a} \left\| \mathcal{L}_{M_1+M_2} \left( \mathbf{S}^t_i; \hat{\theta}_{1,n}^{(b)}, \hat{\theta}_{2,a,n}^{(b)} \right) - Y_i^{t+1} \right\|^2 + \lambda \sum_{j=1}^{r_1} \sum_{\ell=1}^{r_2} \Sigma_{1,\ell,j}^2 \right]$$

6: Compute a sub-gradient descent update

$$\left( \hat{\theta}_{1,n}^{(b+1)}, \hat{\theta}_{2,a,n}^{(b+1)} \right) = \left( \hat{\theta}_{1,n}^{(b)}, \hat{\theta}_{2,a,n}^{(b)} \right) + \alpha_b \Lambda_a^{(b)}$$

7: Set $\hat{\theta}_{2,a',n}^{(b+1)} = \hat{\theta}_{2,a',n}^{(b)}$ for all $a' \neq a$

8: If $\max_a \left| C_n^\lambda \left( \hat{\theta}_{1,n}^{(b+1)}, \hat{\theta}_{2,a,n}^{(b+1)} \right) - C_n^\lambda \left( \hat{\theta}_{1,n}^{(b)}, \hat{\theta}_{2,a,n}^{(b)} \right) \right| \leq \varepsilon$ stop.

**Output:** $\left( \hat{\theta}_{1,n}^{(b+1)}, \hat{\theta}_{2,n}^{(b+1)} \right)$
Figure 2.2 Schematic for alternating deep neural network (ADNN) model. The term “alternating” refers to the estimation algorithm which cycles over the networks for each treatment $a \in \mathcal{A}$.

map $s \mapsto \mathcal{L}_{M_1}(s; \theta_1)$. Define

$$\tilde{R}_n = \left\{ j \in \{1, \ldots, r_1\} : \sum_{\ell, j}^2 \neq 0 \text{ for some } \ell \in \{1, \ldots, r_2\} \right\}$$

to be the elements of $S^I_\phi$ that dictate $S^L_\phi$; write $S^I_\phi$ as shorthand for $\left\{ S^I_j \right\}_{j \in \tilde{R}_n}$. One may wish to iterate the foregoing estimation procedure as described in Corollary 1.3.3. However, because the components of $S^L_\phi$ are each a potentially nonlinear combination of the elements of $S^I_\phi$, therefore a sparse feature map defined on the domain of $S^L_\phi$ may not be any more sparse in terms of the original features. Thus, when iterating the feature map construction algorithm, we recommend using the reduced process $\left\{ S^T_{\tilde{R}_n, i}, \tilde{A}_i^T, \tilde{U}_i^T \right\}_{i=1}^n$ and the input; because the sigma-algebra generated by $S^T_{\tilde{R}_n}$ contains the sigma-algebra generated by $S^T_\phi$, this does not incur any loss in generality. The above procedure can be iterated until no further dimension reduction occurs.
2.3 Simulation Experiments

We evaluate the finite sample performance of the proposed method (pre-screening with Brownian distance covariance + iterative alternating deep neural networks, which we will simply refer to as ADNN in this section) using a series of simulation experiments. To form a basis for comparison, we consider two alternative feature construction methods: (PCA) principal components analysis, so that the estimated feature map \( \hat{\phi}_{PCA}(s) \) is the projection of \( s \) onto the first \( k \) principal components of \( T^{-1} \sum_{t=1}^{T} \{ S_t - P_n S_t \} \{ S_t - P_n S_t \}^\top \); and (tNN) a traditional sparse neural network, which can be seen as a special case of our proposed alternating deep neural network estimator where there is only 1 action. In our implementation of PCA, we choose the number of principal components, \( k \), corresponding to 90% of variance explained. We do not compare with sparse PCA for variable selection, because based on preliminary runs, the principal components that explain 50% of variance already use all the variables in our generative model. In our implementation of tNN, we build a separate tNN for each \( a \in \mathcal{A} \), where \( (\lambda, K_1, K_2, r_1) \) are tuned using cross-validation, and take the union of selected variables and constructed features. Note that there is no other obvious way to join the constructed features from tNN but to simply concatenate them, which will lead to inefficient dimension reduction especially when \( |\mathcal{A}| \) is large, whereas we will see that ADNN provides a much more efficient way to aggregate the useful information across actions.

We evaluate the quality of a feature map, \( \phi \), in terms of the marginal mean outcome under the estimated optimal regime constructed from the reduced data \( \{ \tilde{S}^{T+1}, \tilde{A}_{\phi,i}, \tilde{U}_{i}^T \}^n_{i=1} \) using Q-learning with function approximation (Bertsekas and Tsitsiklis, 1996; Murphy, 2005); we use both linear function approximation and non-linear function approximation with neural networks. A description of Q-learning is in Appendix A.1.
We consider data from the following class of generative models, as illustrated in Figure 2.3:

\[
S^1 \sim \text{Normal}_{64}(0, 0.25I_{64}); \quad A^1, \ldots, A^T \sim \text{i.i.d. Bernoulli}(0.5); \\
S_{4i-3}^{t+1}, S_{4i-2}^{t+1} \sim \text{i.i.d. Normal}\{(1-A^t)g(S^t_i), 0.01(1-A^t)+0.25A^t\}; \\
S_{4i-1}^{t+1}, S_{4i}^{t+1} \sim \text{i.i.d. Normal}\{A^tg(S^t_i), 0.01A^t+0.25(1-A^t)\}; \\
U^t \sim \text{Normal}\{(1-A^t)[2\{g(S^t_1)+g(S^t_2)\} - \{g(S^t_3)+g(S^t_4)\}] + A^t[2\{g(S^t_1)+g(S^t_2)\} - \{g(S^t_3)+g(S^t_4)\}], 0.01\};
\]

for \(i = 1, 2, \ldots, 16.\)

**Figure 2.3** Relationship between \(S^t\) and \(Y^{t+1}\) in the generative model, which depends on the action. First 16 variables determine the next state. First 4 variables determine the utility.

The above class of models is indexed by \(g : \mathbb{R} \to \mathbb{R}\) which we vary across the following maps: identity \(g(u) = u\), truncated quadratic \(g(u) = \min\{u^2, 3\}\), and truncated exponential \(g(u) = \min\{e^u, 3\}\), where the truncation is used to keep all variables of relatively the same scale across time points. Additionally, we add 3 types of noise variables, each taking up about \(\frac{1}{3}\) of total noises added:

(i) dependent noise variables \(D_j^t\), which are generated the same way as above except that they don’t affect the utility;

(ii) white noises \(W_k^t\), which are sampled independently from \(\text{Normal}(0, 0.25)\) at each
time point; and

(iii) constants $C_t^I$, which are sampled independently from \(\text{Normal}(0, 0.25)\) at \(t = 1\) and remain constant over time.

More precisely, let \(m\) be the total number of noise variables, then

\[
D^1_j, W^1_k, C^1_l \sim \text{i.i.d. Normal}(0, 0.25);
\]

\[
D^{t+1}_{4i-3}, D^{t+1}_{4i-2} \sim \text{i.i.d. Normal} \{(1 - A^t)g(D^t_i), 0.01(1 - A^t) + 0.25A^t\};
\]

\[
D^{t+1}_{4i-1}, D^{t+1}_{4i} \sim \text{i.i.d. Normal} \{A^tg(D^t_i), 0.01A^t + 0.25(1 - A^t)\};
\]

\[
W^t_k \sim \text{i.i.d. Normal}(0, 0.25); \quad C^t_l = C^1_l;
\]

for \(j = 1, 2, \ldots, \lfloor m/3 \rfloor; \quad k = 1, 2, \ldots, \lfloor m/3 \rfloor; \quad l = 1, 2, \ldots, \lfloor m/3 \rfloor\).

It can be seen that the first 16 variables, the first 4 variables, and \(\{g(S^1_i), g(S^2_i), g(S^3_i) + g(S^4_i)\}^T\) all induce a sufficient MDP. The foregoing class of models is designed to evaluate the ability of the proposed method to identify low-dimensional and potentially nonlinear features of the state in the presence of action-dependent transitions and various noises. For each Monte Carlo replication, we sample \(n = 30\) i.i.d. trajectories of length \(T = 90\) from the above generative model.

The results based on 500 Monte Carlo replications are reported in Table 2.1 - 2.3. In addition to reporting the marginal mean outcome under the policy estimated using Q-learning with both function approximations, we also report: (nVar) the number of selected variables; and (nDim) the dimension of the feature map. It can be seen that

(i) ADNN produces substantially smaller nVar and nDim compared with PCA or tNN in all cases;

(ii) ADNN is robust to the 3 types of noises;
(iii) when fed into the Q-learning algorithm, ADNN leads to considerably better marginal mean outcome than PCA and the original states with non-linear models;

(iv) ADNN is able to construct features suitable for Q-learning with linear function approximation even when the utility function and transition between states are non-linear;

(v) even when Q-function is a flexible function approximator like neural nets, marginal mean outcome could still be poor when input features are high dimensional (Table 2.2), suggesting the necessity of efficient dimension reduction.

<table>
<thead>
<tr>
<th>Model</th>
<th>nNoise</th>
<th>Feature map</th>
<th>Linear Q</th>
<th>NNQ</th>
<th>nVar</th>
<th>nDim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$s_t$</td>
<td>3.36(0.012)</td>
<td>3.31(0.012)</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_{1t}^t, s_{2t}^t, s_{3t}^t, s_{4t}^t)^T$</td>
<td>3.34(0.012)</td>
<td>3.31(0.012)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{ADNN}(s^t)$</td>
<td>3.21(0.018)</td>
<td>3.34(0.013)</td>
<td>4.1(0.01)</td>
<td>3.1(0.02)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{NN}(s^t)$</td>
<td>3.38(0.012)</td>
<td>3.30(0.012)</td>
<td>16.0(0.00)</td>
<td>34.4(0.13)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{PCA}(s^t)$</td>
<td>3.34(0.012)</td>
<td>3.30(0.012)</td>
<td>64</td>
<td>50.0(0.00)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>$s_t$</td>
<td>3.31(0.012)</td>
<td>3.27(0.012)</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_{1t}^t, s_{2t}^t, s_{3t}^t, s_{4t}^t)^T$</td>
<td>3.31(0.012)</td>
<td>3.29(0.013)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{ADNN}(s^t)$</td>
<td>3.26(0.014)</td>
<td>3.32(0.013)</td>
<td>5.6(0.08)</td>
<td>4.6(0.08)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{NN}(s^t)$</td>
<td>3.32(0.012)</td>
<td>3.29(0.013)</td>
<td>37.0(0.00)</td>
<td>86.0(0.19)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{PCA}(s^t)$</td>
<td>3.34(0.012)</td>
<td>3.28(0.013)</td>
<td>114</td>
<td>85.8(0.02)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>$s_t$</td>
<td>2.17(0.016)</td>
<td>2.98(0.035)</td>
<td>264</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_{1t}^t, s_{2t}^t, s_{3t}^t, s_{4t}^t)^T$</td>
<td>3.33(0.012)</td>
<td>3.31(0.013)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{ADNN}(s^t)$</td>
<td>3.29(0.013)</td>
<td>3.32(0.012)</td>
<td>10.2(0.12)</td>
<td>7.5(0.09)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{NN}(s^t)$</td>
<td>3.34(0.012)</td>
<td>3.27(0.013)</td>
<td>87.4(0.11)</td>
<td>157.8(0.48)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\hat{\phi}_{PCA}(s^t)$</td>
<td>3.33(0.013)</td>
<td>3.11(0.028)</td>
<td>264</td>
<td>166.0(0.02)</td>
</tr>
</tbody>
</table>
Table 2.2 Comparison of feature map estimators under quadratic transition

<table>
<thead>
<tr>
<th>Model</th>
<th>nNoise</th>
<th>Feature map</th>
<th>Linear Q</th>
<th>NNQ</th>
<th>nVar</th>
<th>nDim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>$s_f$</td>
<td>3.08(0.062)</td>
<td>2.64(0.073)</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_1^f, s_2^f, s_3^f, s_4^f)^T$</td>
<td>2.54(0.056)</td>
<td>6.75(0.046)</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{ADNN}(s')$</td>
<td>6.63(0.038)</td>
<td>6.97(0.034)</td>
<td>4.1(0.02)</td>
<td>2.4(0.04)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{tNN}(s')$</td>
<td>6.94(0.027)</td>
<td>6.54(0.068)</td>
<td>15.3(0.04)</td>
<td>37.1(0.22)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{PCA}(s')$</td>
<td>2.97(0.064)</td>
<td>2.50(0.067)</td>
<td>64</td>
<td>51.2(0.02)</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>$s_f$</td>
<td>2.96(0.054)</td>
<td>1.69(0.064)</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_1^f, s_2^f, s_3^f, s_4^f)^T$</td>
<td>2.58(0.057)</td>
<td>6.76(0.042)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{ADNN}(s')$</td>
<td>6.76(0.032)</td>
<td>6.99(0.030)</td>
<td>6.4(0.06)</td>
<td>5.3(0.09)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{tNN}(s')$</td>
<td>6.98(0.031)</td>
<td>6.53(0.064)</td>
<td>36.5(0.03)</td>
<td>88.3(0.22)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{PCA}(s')$</td>
<td>3.09(0.061)</td>
<td>2.00(0.067)</td>
<td>114</td>
<td>87.1(0.03)</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>$s_f$</td>
<td>1.28(0.030)</td>
<td>0.88(0.031)</td>
<td>264</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(s_1^f, s_2^f, s_3^f, s_4^f)^T$</td>
<td>2.52(0.056)</td>
<td>6.68(0.050)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{ADNN}(s')$</td>
<td>6.87(0.034)</td>
<td>6.92(0.033)</td>
<td>14.3(0.14)</td>
<td>12.5(0.23)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{tNN}(s')$</td>
<td>6.76(0.044)</td>
<td>6.03(0.075)</td>
<td>84.1(0.11)</td>
<td>152.4(0.36)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\phi_{PCA}(s')$</td>
<td>3.09(0.062)</td>
<td>0.96(0.033)</td>
<td>264</td>
<td>167.4(0.03)</td>
</tr>
</tbody>
</table>

2.4 Application to BASICS-Mobile

We illustrate the proposed methodology using data on the effectiveness of BASICS-Mobile, a behavioral intervention delivered via mobile device, targeting heavy drinking and smoking among college students (Witkiewitz et al., 2014). Mobile interventions are appealing because of their 24-hour availability, anonymity, portability, increased compliance, and accurate data recording (Heron and Smyth, 2010). BASICS-Mobile enrolled 30 students and lasted for 14 days. On the afternoon and evening of each day, subjects were asked to complete a list of self-report questions and then were provided with either an informational module or a treatment module. A treatment module contains 1-3 mobile phone screens of interactive
### Table 2.3 Comparison of feature map estimators under exponential transition

<table>
<thead>
<tr>
<th>Model</th>
<th>nNoise</th>
<th>Feature map</th>
<th>Linear Q (n=15)</th>
<th>NNQ (n=15)</th>
<th>nVar</th>
<th>nDim</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>0</td>
<td>s, (s_1, s_2, s_3, s_4)^T, \phi_{ADNN}(s'), \phi_{tNN}(s'), \phi_{PCA}(s')</td>
<td>8.73(0.008)</td>
<td>8.78(0.012)</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.20(0.006)</td>
<td>9.43(0.004)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.30(0.018)</td>
<td>9.45(0.004)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.44(0.005)</td>
<td>9.29(0.009)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.10(0.016)</td>
<td>9.02(0.023)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exp</td>
<td>50</td>
<td>s, (s_1, s_2, s_3, s_4)^T, \phi_{ADNN}(s'), \phi_{tNN}(s'), \phi_{PCA}(s')</td>
<td>8.78(0.008)</td>
<td>8.77(0.012)</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.19(0.006)</td>
<td>9.43(0.005)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.32(0.018)</td>
<td>9.43(0.005)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.43(0.005)</td>
<td>9.18(0.012)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8.89(0.014)</td>
<td>8.99(0.020)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exp</td>
<td>200</td>
<td>s, (s_1, s_2, s_3, s_4)^T, \phi_{ADNN}(s'), \phi_{tNN}(s'), \phi_{PCA}(s')</td>
<td>8.71(0.008)</td>
<td>8.73(0.012)</td>
<td>264</td>
<td>264</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.19(0.006)</td>
<td>9.44(0.004)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.37(0.016)</td>
<td>9.41(0.006)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.41(0.005)</td>
<td>9.06(0.016)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8.66(0.014)</td>
<td>9.02(0.022)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

content, such as comparing the subject's current smoking level with those of their peers, or guiding the subject to manage their smoking urges. A treatment module is generally more burdensome than an informational module and may be less effective if, for example, the subject's stress level is high; furthermore, excessive treatment can cause habituation and disengagement from the intervention. An optimal intervention will assignment a treatment module if and when it is needed without diminishing engagement.

In our original formulation of this decision problem as an MDP, the state comprises 15 variables capturing baseline information, current answers to the self-report questions, a weekend indicator, age, past attempts to quit smoking, current smoking urge, and current stress level (see Appendix B.1 for details); the action is whether a treatment module is assigned; the utility is the negative of the number of cigarettes smoked at the next time...
point; the goal is to find a strategy that minimizes the cumulative cigarette rate.

Two subjects with large amounts of missing data were excluded. All other missing values are imputed with the fitted value from a local polynomial regression of the state variable on time $t$. We treat all the variables as ordinal, partitioning some of them (see Appendix B.2 for a complete description). We estimate $\hat{\phi}_{ADNN}(s_t)$ wherein conditional independence is checked via condition (i) in Lemma 3.4 using a likelihood ratio test (LRT). (Recall that we use distance covariance test for continuous states and LRT for categorical/ordinal states. See Appendix B.3 for more details on the LRT.) The dimension of $\hat{\phi}_{ADNN}(s_t)$ is set to be the smallest dimension for which $\hat{\phi}_{ADNN}(s_t)$ fails to reject this independence condition at level $\tau = 0.05$; this procedure resulted in a feature set of dimension six. To increase the interpretability of the constructed feature map, we constrained the dimension reduction network to have no hidden layers. Under this constraint, $\hat{\phi}_{ADNN}(s_t)$ is a linear transformation of $s_t$ followed by application of $\Phi^e$ which was set to be the arctangent function. A plot of the weights of the 15 original variables in the linear transformation for each component of the feature map is useful in interpreting the learned feature map; see Figure 2.4 for an example.

We estimate the optimal strategy using Q-learning applied to the learned feature map. Comparing the estimated parameters for treatment and no treatment, while examining the plots of weights, we can get a sense of how the original variables impact the optimal treatment assignment. For instance, the 1st parameter in the Q-function for treatment is smaller than the one for no treatment, which suggests that the 1st new variable contributes to the decision to apply treatment by being small, i.e., if it is the weekend and a subject’s stress level is low then the estimated policy is more likely to provide treatment. This agrees with the intuition that a treatment module would be more effective when the subject is not busy or stressed. Similarly, it can be seen that previous attempts to quit smoking is positively associated with providing treatment, with the possible explanation that individuals with
prior quit attempts tend to be more cooperative and receptive of frequent treatment.

2.5 Discussion

Data-driven decision support systems are being deployed across a wide range of application domains including medicine, engineering, and business. MDPs provide the mathematical underpinning for most data-driven decision problems with an infinite or indefinite time horizon. While the MDP model is extremely general, choosing a parsimonious representation of a decision process that fits the MDP model is non-trivial. We introduced the notion of a feature map which induces a sufficient MDP and provided an estimator of such a feature map based on a variant of deep neural networks.

There are several important ways in which this work can be extended; we mention two of the most pressing below.

We considered estimation from a batch of i.i.d. replicates; however, in some applications it may be desirable to estimate a feature map online as data accumulate. In such cases,
a data-driven, and hence evolving, feature map of the state will be stored, complicating estimation. Furthermore, because the proposed algorithm sweeps through the observed data multiple times it is not suitable for real-time estimation. One possible solution is to use an experience replay system (Mnih et al., 2013, 2015): Store new observations in the form of the tuple \((S', A', R', S'^{t+1})\) in a database, and sample batches of tuples from the database to update the feature map periodically. But if there is an online Q-learning algorithm performing on the constructed features, there will be stability issues when both the feature map and the Q-function are simultaneously updated. One needs to be careful with how often (in comparison to the update schedule of the Q-function) and how much (in terms of learning rate and number of iterations) the feature map gets updated.

Another important extension is to tailor the feature map to states with complex data structures, for example, images and text; such data are increasingly common in health, engineering, and security applications. Existing neural network architectures designed for such data (Krizhevsky et al., 2012; Dahl et al., 2012; Simonyan and Zisserman, 2014) could potentially be integrated into the proposed feature map construction algorithm. In the next section we will explore one such architecture.

2.6 Extension to Multi-Dimensional Arrays – Convolutional Alternating Deep Neural Networks

Multi-dimensional\(^2\) data such as images often have spatial correlations among the features. Squeezing the data into 1D could lead to the loss of the spatial structures. In this section we will explore ways to tailor our Alternating Deep Neural Networks (ADNN) algorithm to

\(^2\)Here the dimension refers to the modality of the input arrays, not number of elements. E.g., a 2D sample is a matrix.
multi-dimensional data with spatial structures. For simplicity, from now on we will assume that the data are images or stacks of images (e.g., consecutive frames in a video), but the methods to be discussed apply to data of an arbitrary dimension.

### 2.6.1 Background

Dominating the computer-vision field since 2012, Convolutional Neural Networks (ConvNets) (see Nielsen, 2015; LeCun et al., 2015; Goodfellow et al., 2016; Dumoulin and Visin, 2016, and references therein) are a variation of neural networks that has been widely adopted in tasks involving image data. The key ideas of ConvNets include local connections, shared weights, and pooling (LeCun et al., 2015), which distinguish them from “typical” neural networks (see Section 2.1).

In ConvNets, the constructed features in each layer are organized as a 3D array, or equivalently, a stack of 2D arrays where each 2D array is called a feature map. The convolution operation is essentially running a local neural network with no hidden layers, called a filter or kernel, on the sub-matrices with the same location of all the feature maps from the previous layer to generate a single column (in the 3rd dimension) of features in the next layer; the kernel “slides” across the input feature maps with a pre-specified stride size to generate the complete output features; the input feature maps are sometimes padded with 0’s on the edges; the kernel size, stride size, and padding scheme together determine the output dimension (Figure 2.5). As a concrete example, for simplicity assume that the input and output both contain a single square feature map (so we can ignore the 3rd dimension), there is no padding, the kernel is square, and the stride size is 1 along both axes, then we can let $O^{l-1} \in \mathbb{R}^{p \times p}$ denote the input, $w^l \in \mathbb{R}^{q \times q}$ denote the kernel, $O^l \in \mathbb{R}^{(p-q+1) \times (p-q+1)}$ denote $O^{l-1} \in \mathbb{R}^{p \times p}$ denote the input, $w^l \in \mathbb{R}^{q \times q}$ denote the kernel, $O^l \in \mathbb{R}^{(p-q+1) \times (p-q+1)}$ denote

---

In fact, 1D data can also have spatial correlations, such as audio signals.

---

The term “convolution” used here does not correspond precisely to the definition used in engineering or mathematics, but instead “cross-correlation” (Goodfellow et al., 2016).
the output, and compute the convolution (\(*\)) as follows:

\[
O^l_{i,j} = (O^{l-1} * w^l)_{i,j} = \sum_{m=1}^{q} \sum_{n=1}^{q} O^{l-1}_{i+m-1,j+n-1} w^l_{m,n}.
\]

A convolutional layer replaces the typical affine transformation with such a convolution operation (see Dumoulin and Visin, 2016; Goodfellow et al., 2016, for more details on the arithmetic of convolution) and is the defining component of ConvNets. Note that all elements in a feature map are generated by the same kernel, hence they are connected to different subsets of features from the previous layer but share the same parameters. As a result of this architecture, each feature map can potentially extract a particular type of local feature (such as edges, circles, etc.), which is also desirably translation invariant (i.e., the location of the feature is unimportant). The shared weights also allow a convolutional layer to have drastically less parameters than fully-connected layers and scale better to high dimensional input. The typical non-linear activation function applied to a convolutional layer is Rectified Linear Unit (ReLU), which is element-wise defined as \(\xi(x) = \max\{x, 0\}\).

Another type of layer commonly used in ConvNets is a pooling layer, which simply computes the maximum or average values over sub-matrices of feature maps from the previous layer. The pooling layer functions as coarse-graining the previous features in order to be robust to small perturbations in the input and reduce the variation of learned representations (LeCun et al., 2015). ConvNets usually use convolutional layers (with ReLU) and pooling layers alternately before adding a few fully-connected layers and a softmax layer in the end (Krizhevsky et al., 2012; LeCun et al., 2015). The use of ReLU and regularization techniques such as Dropout (Srivastava et al., 2014), Batch normalization (Ioffe and Szegedy, 2015), and Residual connection (He et al., 2016) allows efficient training of very deep ConvNets (Szegedy et al., 2015; He et al., 2016)
Figure 2.5 The convolution operation in ConvNets. In this example, each input feature map is 4 × 4; each output feature map is 2 × 2; the kernel’s first two dimensions are 3 × 3; the stride size is 1; there is no padding; the convolution takes 4 steps.

When the dimension of the output of a convolutional layer is higher than that of the input, a transposed convolution\(^5\) is used for upsampling. A transposed convolution can be viewed as swapping the forward and backward passes of a convolution, or transposing the unrolled matrix multiplication of a convolution, or applying convolution to appropriately 0-padded input (Dumoulin and Visin, 2016).

Convolutions and transposed convolutions will be the building blocks of Alternating

\(^5\)It is sometimes called *fractionally strided convolution* or *deconvolution*, although some argue that deconvolution is a misleading name (Dumoulin and Visin, 2016).
Deep Neural Networks with image input.

2.6.2 Related Work

A common dimension reduction technique using neural networks is called autoencoder (Bengio, 2009; Vincent et al., 2010). A typical autoencoder\(^6\) is composed of fully-connected layers with a hidden layer of smaller dimension than that of the input, serving as the bottleneck. The autoencoder is trained to reconstruct the original input, making the features in the bottleneck layer a dimension reduced representation of the input. A convolutional autoencoder (Zeiler et al., 2010; Masci et al., 2011) uses convolutions to construct a 2D dimension reduced representation and transposed convolutions to reconstruct the original 2D input. But as discussed in Section 1.4, autoencoder (and for the same reason convolutional autoencoder) is not a suitable dimension reduction technique in the context of Markov Decision Processes, as it does not account for action, future state, or utility.

A more closely related method is the Action-Conditional Video Prediction Network (ACVPN) (Oh et al., 2015). It uses convolutions to encode the 2D states, transforms the reduced representation with an action-conditional matrix multiplication, then decodes the transformed representation with transposed convolutions to predict the future state (Figure 2.6). Besides the obvious difference of having 2D input, ACVPN differs from Alternating Deep Neural Network (ADNN) in the following ways: 1. The goal of ACVPN is to predict action-conditional future states to guide exploration during reinforcement learning, while the goal of ADNN is dimension reduction and variable selection with no information loss. Hence the complete ADNN algorithm includes pre-screening with distance covariance (Algorithm 1), variable selection with group lasso (Equation 2.1), iterative dimension re-

---

\(^{6}\)Some literature (Vincent et al., 2010) refers to an autoencoder with more than one hidden layer as stacked autoencoders, but for simplicity we will just call it autoencoder no matter the number of hidden layers
duction (Corollary 1.3.3), and sufficiency check (Lemma 1.3.4), which are not as important for ACVPN’s purpose. 2. ADNN uses utility as part the action-conditional output, which is necessary for constructing a sufficient dimension reduction. 3. ACVPN trades off predictive flexibility with parameter parsimony. ADNN gives each action a separate branch of prediction network, while ACVPN only gives each action a separate matrix multiplication, which is equivalent to giving each action a separate fully-connected layer before merging the action-conditional layers to a single branch of prediction network. When the states are huge, ACVPN’s approach of incorporating actions might be more suitable as it keeps the model size reasonable at the cost of some prediction accuracy.

Our convolutional ADNN presented in the next sub-section will merge the elements of ADNN and ACVPN to produce a dimension reduction of states that are multi-dimensional arrays.

**Figure 2.6** The architecture of Action-Conditional Video Prediction Network (Oh et al., 2015). The blue, orange, and purple arrows indicate convolution, linear transformation, and transposed convolution respectively.
2.6.3 Method

Combining Alternating Deep Neural Network (ADNN) and Action-Conditional Video Prediction Network (ACVPN), our proposed Convolutional Alternating Deep Neural Network (ConvADNN) has four major components and follows the overall encode-transform-decode architecture. As mentioned, for convenience we assume that the input is a 3D array, such as a stack of images, but the model applies to input of arbitrary dimension as long as it has spatial structure to be taken advantage of. Let $\xi_k$ denote an element-wise activation function\(^7\). Let $\ast$ and $\ast^T$ denote the convolution and transposed convolution respectively. Then the four components are defined as (Figure 2.7):

(i) **Dimension Reduction Network**: The dimension reduction network uses a sequence of convolutions to construct a low-dimensional 3D representation of the 3D input state. Given $s^t \in \mathbb{R}^{p_1 \times p_2 \times p_3}$, define

$$\mathcal{L}_1(s^t) = \xi_1(s^t \ast w_1),$$

and recursively,

$$\mathcal{L}_k(s^t) = \xi_k \{ \mathcal{L}_{k-1}(s^t) \ast w_k \}$$

for $k = 2, \ldots, M_1$. $\phi(s^t) := \mathcal{L}_{M_1}(s^t) \in \mathbb{R}^{q_1 \times q_2 \times q_3}$ will be our dimension reduced state, where $p_1 p_2 p_3 > q_1 q_2 q_3$. Note that it does not depend on the action.

(ii) **Action Transformation Layer**: The action transformation layer applies an action-

---

\(^7\)In case there is no activation, $\xi_k$ will just be the identity map.
conditional affine transformation to the dimension reduced state. Define

\[
L_{M_1+1}(s^t, a^t) = \sum_{a \in \mathcal{A}} 1_{a = a^t} \left\{ w_{M_1+1,a} \cdot L_{M_1}(s^t) + b_{M_1+1,a} \right\}.
\]

Essentially, we give each possible action a separate fully-connected layer; all \(|\mathcal{A}|\) such layers share the same previous layer and next layer. Note that it is possible to give each action multiple fully-connected layers to form an action transformation network, increasing flexibility but also model size.

(iii) **State Prediction Network**: The state prediction network upsamples the transformed reduced state with a sequence of transposed convolutions to predict the next state. Define

\[
\mathcal{L}_{M_1+2}^s(s^t, a^t) = \xi_{M_1+2}^s \left\{ \mathcal{L}_{M_1+1}(s^t, a^t)^\top w_{M_1+2}^s \right\},
\]

and recursively,

\[
\mathcal{L}_{M_1+1+k}^s(s^t, a^t) = \xi_{M_1+1+k}^s \left\{ \mathcal{L}_{M_1+k}^s(s^t, a^t)^\top w_{M_1+k}^s \right\}
\]

for \(k = 2, \ldots, M_2\). The postulated model for \(E\left(S^{t+1} | S_\phi^t = s_\phi^t, A_\phi^t = a_\phi^t\right)\) is \(\mathcal{L}_{M_1+1+M_2}^s(s^t, a^t)\).

(iv) **Utility Prediction Network**: The utility\(^8\) prediction network uses a sequence of fully-connected layers to predict the utility. Define

\[
\mathcal{L}_{M_1+2}^u(s^t, a^t) = \xi_{M_1+2}^u \left\{ w_{M_1+2}^u \cdot \mathcal{L}_{M_1+1}(s^t, a^t) + b_{M_1+2}^u \right\},
\]

\(^8\)As mentioned in Chapter 1, the utility is often referred to as “reward” in the reinforcement learning literature.
and recursively,

$$
\mathcal{L}^u_{M_1+1+k}(s^t, a^t) = \xi_{M_1+1+k} \left\{ w^u_{M_1+1+k} \cdot \mathcal{L}^u_{M_1+k}(s^t, a^t) + b^u_{M_1+1+k} \right\}
$$

for \( k = 2, \ldots, M_3 \). The postulated model for \( \mathbb{E} \left( U^t | S^t = s^t, A^t = a^t \right) \) is \( \mathcal{L}^u_{M_1+1+M_3}(s^t, a^t) \).

Figure 2.7 The architecture of Convolutional Alternating Deep Neural Network.

Let \( \theta \) denote the set of parameters of ConvADNN, \( \delta \) denote the weight on the utility prediction, and \( \mathbb{P}_n \) denote the empirical measure, without group lasso the cost function is

$$
C^\delta_n(\theta) = \mathbb{P}_n \sum_{t=1}^T \left\{ \left\| \mathcal{L}^s_{M_1+M_2}(s^t, A^t) - s^{t+1} \right\|^2 + \delta \left\| \mathcal{L}^u_{M_1+M_3}(s^t, A^t) - U^t \right\|^2 \right\}.
$$
Similar to ADNN, ConvADNN can apply group lasso on the first layer for variable selection. But due to the arithmetic of convolution (think of the kernel “sliding” across the input), when both the input and output of the first layer are 3D, group lasso should be applied to 3D slices of the 4D kernel to select/remove entire feature maps from the input, instead of being applied to the columns.

Except for the notational differences, the training algorithm of ConvADNN is the same as that of ADNN (Algorithm 2) and can be applied iteratively (Corollary 1.3.3). However, the variable pre-screening with Brownian Distance Covariance (Algorithm 1) might not be suitable as it does not account for spatial structures in the input and can be too computationally expensive for input data like video segments.

If it is known that a certain variable from the input is unimportant (such as a corner pixel of one frame of a video segment), one can simplify the architecture of ConvADNN by replacing the unimportant variable in $S^{t+1}$ with $U^t$ during training, and using a sequence of transposed convolutions to predict both the next state and the utility instead of having separate state and utility prediction networks (Figure 2.8).

## 2.6.4 Simulation Experiments

Video games have become popular test beds for reinforcement learning algorithms (see for example Mnih et al., 2015, 2016; Wang et al., 2017). Variations of q-learning with ConvNet q-functions have been the predominant methods and achieved great success in the video game domain (Henderson et al., 2017). Dimension reduction with Alternating Deep Neural Network (ADNN) or its convolutional variation (ConvADNN) is particularly useful when the full states are expensive to collect or store, policy estimation is off-line, and reinforcement learning on the full states is ineffective (see earlier sections). Hence ConvADNN and perhaps
dimension reduction in general are not best suited for the video game domain, where unlimited full states can be cheaply generated and discarded, policy estimation is better done online with an adaptive exploration strategy, and q-learning on the full states is very successful\textsuperscript{9}. Nevertheless, we can still compare ConvADNN with alternative dimension reduction techniques on video games to obtain some preliminary evidence supporting ConvADNN as a superior dimension reduction technique for states with spatial structures.

We use the following four games with increasing complexity from PyGame Learning Environment (Tasfi, 2016) to evaluate the dimension reduction techniques: Catcher, Pong, Pixelcopter, and FlappyBird (Figure 2.9). See Appendix B.4 for a brief description of the games.

We initialize the screen size for all the games to be $84 \times 84$ (pixels). During preprocessing, the RGB channels of each frame are converted to grayscale using relative luminance: $\text{GRAY} = \frac{0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B}{\text{max}(R, G, B)}$.\footnote{Note that image and video data from real life scenarios often don't have all these conveniences and might need dimension reduction before reinforcement learning}
Figure 2.9 Screenshots from the games used as testbeds. From left to right, top to bottom: Catcher, Pong, Pixelcopter, FlappyBird.

0.21 · RED + 0.72 · GREEN + 0.07 · BLUE. As per convention (Mnih et al., 2015), the four most recent grayscale frames are concatenated to form the current state\(^\text{10}\), so \(S^t \in \mathbb{R}^{84 \times 84 \times 4}\). \(U^t\) is determined by the game settings (Appendix B.4).

To estimate the optimal strategies from the process with full states (no dimension reduction), we apply one of the state-of-the-art reinforcement learning algorithms in the video game domain, dueling double deep q-learning (DDDQ) (Hasselt et al., 2016; Wang et al., 2016), to the original process. The dimension reduction techniques to be evaluated are as follows (note that for a fair comparison the total number of features in the reduced state remains the same):

---

\(^{10}\)A single frame shows the positions of the objects but not their velocity or acceleration, and hence multiple frames are needed to form a (approximate) Markov Decision Process.
(i) **ConvADNN:** We use the simplified architecture of ConvADNN (Figure 2.8), restrict the output of the dimension reduction network so that \( \hat{\phi}_{CADNN}(S^t) \in \mathbb{R}^{40 \times 40 \times 1} \), and train the whole ConvADNN with \( \{[S_i^1, A_i^1, U_i^1, S_i^2, \ldots, A_i^{T_i}, U_i^{T_i}, S_i^{T_i+1}]\}_{i=1}^n \) generated from random (discrete uniform) action selection.

(ii) **Single-Downsizing:** When the screen size of a game is larger than 84 × 84, each frame is typically downsized to 84 × 84 during preprocessing in the experiments in the literature (Henderson et al., 2017). So it is natural to use further downsizing as a dimension reduction method. Recall that \( S^t \) contains the four most recent frames from the game. Single-downsizing downsizes the last frame in \( S^t \) to 40 × 40 using nearest neighbor (Jones et al., 2001–) and deletes the previous three frames, so \( \hat{\phi}_{SD}(S^t) \in \mathbb{R}^{40 \times 40 \times 1} \).

(iii) **Multi-Downsizing:** Multi-downsizing downsizes all four frames in \( S^t \) to 20 × 20 using nearest neighbor (Jones et al., 2001–), so \( \hat{\phi}_{MD}(S^t) \in \mathbb{R}^{20 \times 20 \times 4} \).

(iv) **PCA:** We apply principal component analysis (PCA) to \( \{[S_i^1, S_i^2, \ldots, S_i^{T_i+1}]\}_{i=1}^n \) generated from random action selection and keep the first 1600 principal components. After reformatting, \( \hat{\phi}_{PCA}(S^t) \in \mathbb{R}^{40 \times 40 \times 1} \).

We again apply DDDQ (but with smaller kernel and stride) to the process with reduced states to estimate the optimal strategies. At the end of DDDQ, we test the learned strategy and report the cumulative utility averaged over 100 episodes (Table 2.4). See Appendix B.5 for the hyperparameters used in ConvADNN and DDDQ.

The results show that policies learned from full states perform the best. This is somewhat expected because 1. we mentioned earlier that dimension reduction might not be suitable or necessary in the video game domain; and 2. the hyperparameters for DDDQ on the full states have been optimized and borrowed from other papers (Hasselt et al., 2016;
Table 2.4 Comparison of feature map estimators in terms of cumulative utility using DDDQ. Standard errors are included in parentheses.

<table>
<thead>
<tr>
<th>Game</th>
<th>Feature map</th>
<th>Cumulative utility</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s_t$</td>
<td>219.27 (0.13)</td>
<td>28,224</td>
</tr>
<tr>
<td>Catcher</td>
<td>$\hat{\phi}_{\text{CADNN}}(s')$</td>
<td>219.27 (0.13)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{SD}}(s')$</td>
<td>218.90 (0.11)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{MD}}(s')$</td>
<td>-5.26 (0.12)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{PCA}}(s')$</td>
<td>219.27 (0.13)</td>
<td>1,600</td>
</tr>
<tr>
<td>Pong</td>
<td>$s_t$</td>
<td>15.09 (0.10)</td>
<td>28,224</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{CADNN}}(s')$</td>
<td>13.41 (0.26)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{SD}}(s')$</td>
<td>-14.67 (0.10)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{MD}}(s')$</td>
<td>-15.91 (0.04)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{PCA}}(s')$</td>
<td>-12.64 (0.20)</td>
<td>1,600</td>
</tr>
<tr>
<td>Pixelcopter</td>
<td>$s_t$</td>
<td>106.74 (10.87)</td>
<td>28,224</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{CADNN}}(s')$</td>
<td>77.51 (7.39)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{SD}}(s')$</td>
<td>5.06 (0.70)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{MD}}(s')$</td>
<td>11.58 (1.28)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{PCA}}(s')$</td>
<td>8.42 (0.91)</td>
<td>1,600</td>
</tr>
<tr>
<td>FlappyBird</td>
<td>$s_t$</td>
<td>35.06 (5.27)</td>
<td>28,224</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{CADNN}}(s')$</td>
<td>-3.61 (0.21)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{SD}}(s')$</td>
<td>2.46 (0.22)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{MD}}(s')$</td>
<td>5.35 (1.14)</td>
<td>1,600</td>
</tr>
<tr>
<td></td>
<td>$\hat{\phi}_{\text{PCA}}(s')$</td>
<td>-2.17 (0.37)</td>
<td>1,600</td>
</tr>
</tbody>
</table>

Wang et al., 2016), but only roughly tuned on the reduced states. However, among the dimension reduction techniques, ConvADNN substantially outperforms the others and its cumulative utility is close to that of the original process on all games except for FlappyBird. We suspect that the reasons that FlappyBird poses trouble for ConvADNN are 1. FlappyBird changes the color scheme randomly every time the game restarts; and 2. random walk action selection does not reveal enough game states to train ConvADNN. The second issue would’ve likely been avoided if online versions of ADNN and ConvADNN were designed,
which will be left for future work. Overall we have obtained preliminary evidence in favor of ConvADNN over the alternative dimension reduction techniques as well as some insight into its shortcomings.

2.6.5 Discussion

Markov decision processes where states are multi-dimensional arrays with spatial structures arise in many applications, such as self-driving vehicles, robotics, forest fire controls, and infectious disease containments. We developed a convolutional variation of Alternating Deep Neural Networks (ADNN), ConvADNN, to handle data of such format. Preliminary testing using a suite of video games shows a clear advantage for ConvADNN over image downsizing and principal component analysis.

As mentioned in Section 2.5, an online dimension reduction algorithm that can be combined with online reinforcement learning will be very useful in scenarios where the state space needs to be explored with continuously improving policies, such as in certain video games. So designing online versions of ADNN and ConvADNN will be the main goal of future work.
CHAPTER 3

HUMAN TRAFFICKING DETECTION
WITH ORDINAL REGRESSION NEURAL NETWORKS

3.1 Introduction

Human trafficking is one of the fastest growing crimes and the third largest crime overall in profit behind drug and arms (Amin, 2010). Sex trafficking is a form of human trafficking that involves sexual exploitation through multiple forms of coercion. There were an estimated 3.8 million adult sex trafficking victims and 1.0 million child victims on any given day of 2016, 99 percent of which are females (International Labour Organization et al., 2017). Escort websites have become an increasingly popular vehicle for selling the services of
trafficking victims: According to a recent survivor survey, 38% of underage trafficking victims who were enslaved prior to 2004 were advertised online, and the number rose to 75% for those enslaved after 2004 (THORN and Bouché, 2018). The most frequently used online advertising platform is Backpage, followed by Craigslist, Redbook, SugarDaddy, and Facebook (THORN and Bouché, 2018).

Consequently, escort websites are a rich resource for anti-trafficking operations. However, the law enforcement does not have the time or resource to sift through millions of escort ads to identify those coming from potential traffickers. A scalable and efficient solution is to build a statistical model to predict the likelihood of an ad coming from a trafficker using a dataset annotated by anti-trafficking experts.

We propose an ordinal regression neural network tailored for text input. It has three components: a Word2Vec model (Mikolov et al., 2013b) that maps each word from the text input to a numeric vector, a gated-feedback recurrent neural network (Chung et al., 2015) that sequentially processes the word vectors, and finally an ordinal regression layer (Cheng et al., 2008) that produces an ordinal label as the prediction. We also leverage several recent advancements in deep neural networks such as residual connection (He et al., 2016) and batch normalization (Ioffe and Szegedy, 2015). We conduct our experiments on Trafficking-10k (Tong et al., 2017), a dataset of escort ads for which anti-trafficking experts assigned each sample one of seven ordered labels ranging from “1: Very Unlikely (to come from traffickers)” to “7: Very Likely”. Our proposed model significantly outperforms previously published models (Tong et al., 2017) on Trafficking-10k as well as a variety of baseline ordinal regression models. In addition, we conduct an emoji analysis over the emoji’s used in escort ads using skip-gram model (Mikolov et al., 2013b) and t-SNE (van der Maaten and Hinton, 2008), and we show that the lexicon of trafficking-related emoji’s can be subsequently expanded.
In Section 3.2 we discuss related work on human trafficking detection as well as on ordinal regression in general. In Section 3.3 we present in detail our proposed model by components. In Section 3.4 we show the experiment results on Trafficking-10k. In Section 3.5 we summarize our findings and discuss future work.

3.2 Related Work

**Trafficking detection:** There have been several softwares developed to combat human trafficking using statistics and machine learning on online data. To name a few, Memex\(^1\) focuses on search functionalities in the dark web; Spotlight\(^2\) can identify suspicious ads and match images across ads; Traffic Jam\(^3\) finds patterns that can connect multiple ads to the same trafficking organization; TraffickCam\(^4\) is building a crowdsourced database of hotel room images to help identify where an image is taken. However, research efforts have largely been isolated as very few research articles have been published. Closest to our work is the Human Trafficking Deep Network (HTDN) by E. Tong et al. (2017). HTDN has 3 main components: a language network that uses pretrained word embeddings and a long short-term memory network (LSTM) to process text input; a vision network that uses a convolutional network to process image input; and another convolutional network to combine the output of the previous two networks and produce a binary classification. Compared to the language network in HTDN, our model replaces LSTM with a gated-feedback recurrent neural network along with a few other enhancements and uses an ordinal regression layer in the end. We also pretrain word embeddings using skip-gram model (Mikolov et al., 2013b) on unlabeled data from escort ads like E. Tong et al. (2017) did,

\(^1\)darpa.mil/program/memex
\(^2\)htspotlight.com
\(^3\)marinusanalytics.com/trafficjam
\(^4\)traffickcam.com
but we go one step further and conduct an emoji analysis on the learned word embeddings to uncover interesting patterns.

**Ordinal regression:** We will briefly introduce the basic settings of ordinal regression before discussing the previous work. Suppose we have training data 

\[ D_{\text{train}} = \{ (X_i, Y_i) \}_{i=1}^{n} \],

where \( X_i \) are the features and \( Y_i \) is the response. \( Y_i \) is one of \( k \) ordered labels \( \{1, 2, \ldots, k\} \) where \( 1 < 2 \ldots < k \). In the general form, ordinal regression learns a composite map \( \eta = g \circ h \), where \( g : \text{dom}(X) \rightarrow \mathbb{R} \) and \( h : \mathbb{R} \rightarrow \{1, 2, \ldots, k\} \), by minimizing some loss function \( C\{g(X), Y\} \) averaged over the training data. For example, a naive ordinal regression could construct \( \hat{g} \) with linear regression and set \( \hat{h} \) to be a bounded rounding function.

A common evaluation measure for an ordinal regression model is Mean Absolute Error (MAE) (Baccianella et al., 2009), though it may not necessarily correspond to the loss function used. MAE is defined as

\[
\text{MAE}(\hat{\eta}, D_{\text{test}}) = \frac{1}{|D_{\text{test}}|} \sum_{(X_i, Y_i) \in D_{\text{test}}} |\hat{\eta}(X_i) - Y_i|.
\]

To account for rank imbalance in the data, macro-averaged MAE (MAE\textsuperscript{M}) can be used instead (Baccianella et al., 2009). Let \( D_{\text{test}}^i = \{ (X_j, Y_j) \in D_{\text{test}} : Y_j = i \} \), then

\[
\text{MAE}^\text{M}(\hat{\eta}, D_{\text{test}}) = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|D_{\text{test}}^i|} \sum_{(X_j, Y_j) \in D_{\text{test}}^i} |\hat{\eta}(X_j) - Y_j| = \frac{1}{k} \sum_{i=1}^{k} \text{MAE}(\hat{\eta}, D_{\text{test}}^i).
\]

A common approach to ordinal regression is threshold-based methods. Consider binary classification as having a single threshold, say,

\[
\hat{Y} = \begin{cases} 
0 & \text{if } \hat{g}(X) < 0.5 \\
1 & \text{if } \hat{g}(X) \geq 0.5 
\end{cases}
\]
Generalizing binary classification, threshold-based ordinal regression methods partition the real line into \( k \) segments with \( k - 1 \) thresholds: \(-\infty < \theta_1 < \theta_2 < \ldots < \theta_{k-1} < \infty\), and \( \bar{Y} = i \) if \( \theta_{i-1} \leq \bar{g}(X) < \theta_i \). \( \theta_{Y-1} \) and \( \theta_Y \) are the target thresholds defining the “correct” interval for \( \bar{g}(X) \). J. Rennie & N. Srebro (2005) proposed two ways to construct the loss function. Let \( \phi \) be some margin penalty function. Immediate-Threshold penalizes \( g(X) \) falling to the wrong side of the target thresholds:

\[
C \left\{ g(X), Y \right\} = \phi \left\{ g(X) - \theta_{Y-1} \right\} + \phi \left\{ \theta_Y - g(X) \right\}.
\]

In comparison, All-Threshold penalizes \( g(X) \) falling to the wrong side of each threshold:

\[
C \left\{ g(X), Y \right\} = \sum_{i=1}^{Y-1} \phi \left\{ g(X) - \theta_i \right\} + \sum_{i=Y}^{k-1} \phi \left\{ \theta_i - g(X) \right\}.
\]

Minimizing \( C \left\{ g(X), Y \right\} \) under \( L_2 \)-regularization using either construction, the logistic loss \( \phi(t) = \log(1 + e^{-t}) \) corresponds to ordinal logistic regression, and the hinge loss \( \phi(t) = \max(0, 1 - t) \) corresponds to ordinal support vector machine (Rennie and Srebro, 2005).

Least Absolute Deviation (LAD) makes the simplifying assumption that \( \theta_i = i + 0.5 \), i.e., \( \bar{Y} = \text{Round}\{\bar{g}(X)\} \). LAD minimizes \( C \left\{ g(X), Y \right\} = |Y - g(X)| \) for a linear \( g \) under \( L_2 \)-regularization, and can be considered a special case of support vector regression (Fan et al., 2008; Ho and Lin, 2012; Pedregosa-Izquierdo, 2015).

Another common approach to ordinal regression is to transform it into a series of binary classification sub-problems (Frank and Hall, 2001; Li and Lin, 2006). Inspired by K. Crammer & Y. Singer’s threshold-based perceptron model, J. Cheng et al. (2008) introduced the ordinal regression neural network: The network has \( k - 1 \) output, with the \( i \)-th output performing the binary classification of whether \( \bar{Y} > i \). More specifically, the output has
the form $\tilde{g}(X) = \tilde{Z} = [\tilde{Z}_0, \ldots, \tilde{Z}_{k-1}]^T$; $\tilde{Z}_i = \tilde{P}(Y > i | X)$; and $\tilde{Y} = 1 + \sum_{i=1}^{k-1} \text{Round}(\tilde{Z}_i)$. However, as acknowledged by J. Cheng et al. (2008), there is no guarantee that the individual binary classifications won’t conflict with one another, i.e., the predictions might violate the desired monotonic relation ($\tilde{Z}_0 \geq \tilde{Z}_1 \geq \ldots \geq \tilde{Z}_{k-1}$). The ordinal regression neural network was later adapted to process image data (Niu et al., 2016) and text data (Irsoy and Cardie, 2015; Ruder et al., 2016), but to our knowledge, our proposed model is the first to address the conflicting predictions problem.

### 3.3 Method

Our ordinal regression model is a generalization and improvement over the language network used by E. Tong et al. (2017). It has three components (Figure 3.1): Word embeddings pre-trained by a Skip-gram model, a gated-feedback recurrent neural network that constructs abstract features from sentences, and a multi-labeled logistic regression layer tailored for ordinal regression. The details of the components and their respective alternatives will be discussed below.

#### 3.3.1 Word Embeddings

Vector representations of words, or word embeddings, can be obtained through unsupervised learning on a large text corpus so that certain linguistic regularities and patterns are encoded. Compared to Latent Semantic Analysis (Dumais, 2004), embedding algorithms using neural networks are particularly good at preserving linear regularities among words in addition to grouping similar words together (Mikolov et al., 2013a). For example, in the word embeddings trained by Mikolov’s neural networks model on Google News, $\text{vector} (“France”) - \text{vector} (“Paris”) + \text{vector} (“Rome”)$ is closer to $\text{vector} (“Italy”)” than
Figure 3.1 Overview of the ordinal regression neural network for text input. $h$ represents hidden state (details explained in the following subsections).

to any other word vector. Such embeddings can in turn help other algorithms achieve better performances in various natural language processing tasks (Mikolov et al., 2013b), hence the embedding learning can be viewed as a pre-training step.

Unfortunately, the escort ads contain a plethora of emoji’s, acronyms, and (sometimes intentional) typos that are not encountered in more standard text data, which suggests that it is necessary to learn word embeddings from scratch on a large collection of escort ads instead of using previously published embeddings (Tong et al., 2017). We use 168,337 ads scraped from Backpage as our dataset and the Skip-gram model with Negative sampling (Mikolov et al., 2013b) as our model for its state-of-the-art performance and computational efficiency.
The Skip-gram model aims to capture the context of each word by predicting the surrounding words given the current word (Figure 3.2). Let \( W_1, W_2, \ldots, W_T \) be the sequence of words from tokenizing the training text corpus and discarding low-frequency words, the objective is to maximize the average log probability

\[
\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(W_{t+j} | W_t)
\]

where the window size \( c \) determines the number of surrounding words to predict. Each word \( W \) in the vocabulary \( \mathcal{S} \) has an “input” vector representation \( u_W \) and an “output” vector representation \( v_W \), which are learnable parameters. The final input representations will be the desired word embeddings. The basic Skip-gram formulation treats the surrounding word prediction as a multi-class classification task with \( |\mathcal{S}| \) classes and defines \( P(W_{t+j} | W_t) \) using the softmax function:

\[
P(W_{t+j} | W_t) = \frac{\exp(v_{W_{t+j}}^T u_{W_t})}{\sum_{W \in \mathcal{S}} \exp(v_W^T u_{W_t})}.
\]

Note that the cost of computing \( \nabla \log P(W_{t+j} | W_t) \) is proportional to \( |\mathcal{S}| \), which is expensive. In comparison, Negative sampling treats each prediction as a binary logistic regression where the goal is to distinguish the target word \( W_{t+j} \) from \( m \) incorrect words sampled from the noise distribution \( f_n(w) \). More specifically, Negative sampling replaces every \( \log P(W_{t+j} | W_t) \) in the Skip-gram objective by

\[
\log \sigma(v_{W_{t+j}}^T u_{W_t}) + \sum_{W_i \sim f_n(w)} \log \sigma(-v_{W_i}^T u_{W_t})
\]

where \( \sigma \) is the sigmoid function. Now the cost of computing gradient is proportional to \( m \),
which can be as small as 2-5 for large datasets (Mikolov et al., 2013b).

Figure 3.2 The Skip-gram model architecture.

3.3.2 Gated-Feedback Recurrent Neural Network

With word embeddings we can vectorize each word in an escort ad. But to process entire sentences and paragraphs, we need a model to handle sequential data. Recurrent neural networks (RNNs) have recently seen great success at modeling sequential data, especially in natural language processing tasks (LeCun et al., 2015). On a high level, an RNN is a neural network that processes a sequence of inputs one at a time, taking the summary of the sequence seen so far from the previous time point as an additional input and producing a summary for the next time point. The parameters of the network are shared across time points, enabling an RNN to process sequences of arbitrary lengths. One of the most widely used variations of RNNs, a Long short-term memory network (LSTM), uses various gates to control the information flow and is able to better preserve long-term dependencies in the running summary compared to a basic RNN (see Goodfellow et al., 2016, and references...
therein). As an improvement over multi-layered LSTM, a Gated-feedback recurrent neural network (GF-RNN) allows the layers to capture dependencies of different timescales more easily (Chung et al., 2015) and will be our model of choice.

More formally, let $X_1, X_2, \ldots, X_T$ be the sequence of word vectors for an escort ad, where $X_t$ is initialized with the pretrained word embedding when available or randomly otherwise. At timestep $t$, a basic single-layered RNN computes its current hidden states as the summary of information:

$$H_t = \xi (w X_t + u H_{t-1})$$

where $w$ is the input-to-hidden weight matrix, $u$ is the hidden-to-hidden recurrent weight matrix, $H_0 = 0$, and $\xi$ is the activation function. Multiple RNN layers can be stacked on top of one another to form a multi-layered RNN (Figure 3.3):

$$H^j_t = \xi (w^j H^{j-1}_t + u^j H^j_{t-1})$$

where the superscript indicates the layer number, $j = 1, 2, \ldots, m$, and $H^0_t = X_t$. 

![Figure 3.3](image-url) A basic multi-layered RNN.

56
Compared to the above basic multi-layered RNN, besides adding the gates used by LSTM, a GF-RNN also adds cross connections among layers and global reset gates (Figure 3.4). The full computations for the $j$-th layer of a GF-RNN at time $t$ are given by:

\[
\begin{align*}
\text{input gates:} & \quad I^j_t = \sigma(w^j_i H^j_{t-1} + u^j_i H^j_t) \\
\text{forget gates:} & \quad F^j_t = \sigma(w^j_f H^j_{t-1} + u^j_f H^j_t) \\
\text{output gates:} & \quad O^j_t = \sigma(w^j_o H^j_{t-1} + u^j_o H^j_t) \\
\text{global reset gates:} & \quad G^{r-j}_t = \sigma(w^{r-j}_g H^j_{t-1} + u^{r-j}_g H^r_{t-1}) \\
& \text{where } \mathbf{H}^r_{t-1} = \{ (\mathbf{H}^1_{t-1})^T, \ldots, (\mathbf{H}^m_{t-1})^T \}^T, r = 1, \ldots, m; \\
\text{new memory content:} & \quad \tilde{C}^j_t = \tanh(w^j_c H^j_{t-1} + \sum_{r=1}^m G^{r-j}_t u^{r-j}_c H^r_{t-1}) \\
\text{cell states:} & \quad C^j_t = F^j_t \odot C^j_{t-1} + I^j_t \odot \tilde{C}^j_t; \\
\text{hidden states:} & \quad H^j_t = O^j_t \odot \tanh(C^j_t).
\end{align*}
\]

In the above equations, $\sigma$ and $\tanh$ are element-wise sigmoid and hyperbolic tangent functions respectively, and $\odot$ is the element-wise product.

**Figure 3.4** A Gated-feedback RNN.
Regularization techniques for neural networks such as Dropout (Srivastava et al., 2014), Batch normalization (Ioffe and Szegedy, 2015), and Residual connection (He et al., 2016) are added to GF-RNN for further improvements (see Appendix C.1 for details).

After GF-RNN processes an entire escort ad, the average of the hidden states of the last layer, $\overline{H}_m = 1/T \sum_{t=1}^{T} H_t^m$, will become the input for the multi-labeled logistic regression layer. Note that by design the dimension of $\overline{H}_m$ is fixed even though the length $T$ of an ad can vary.

### 3.3.3 Multi-Labeled Logistic Regression Layer

The ordinal regression problem can be cast into a series of binary classification problems to utilize the well-studied classification algorithms (Frank and Hall, 2001; Li and Lin, 2006; Niu et al., 2016). One formulation is as such: Given $k$ total ranks, the $i$-th binary classifier is trained to predict the probability that a sample $X$ has rank larger than $i$: $\hat{f}_i(X) = \hat{P}(Y > i|X)$. Then the predicted rank is

$$\hat{Y} = 1 + \sum_{i=1}^{k-1} \text{Round}\{\hat{f}_i(X)\}.$$ 

In a classification task, the final layer of a deep neural network is typically a softmax layer of dimension the number of classes (Goodfellow et al., 2016). Using the ordinal-regression-to-binary-classifications formulation described above, J. Cheng et al. (2008) replaced the softmax layer in their neural network with a $(k-1)$-dimensional sigmoid layer, where each neuron serves as a binary classifier (Figure 3.5). With sigmoid as the activation function, the output of the $i$-th neuron can be viewed as the predicted probability that the sample has rank greater than $i$\(^5\). Alternatively, the entire sigmoid layer can be considered to perform

---

\(^5\)Actually, in J. Cheng et al.’s original formulation, the final layer is $k$-dimensional with the $i$-th neuron predicting the probability that the sample has rank greater than or equal to $i$. This is redundant because the first neuron should always be equal to 1. Hence we make the slight adjustment of using only $k-1$ neurons.
multi-labeled logistic regression, where the \( i \)-th label is the indicator of the sample's rank being greater than \( i \). The training data are re-formatted accordingly: The response variable for a sample with rank \( i \) becomes \((1^\top_{i-1}, 0^\top_{k-i})\). The \( i-1 \) binary classifiers share the features constructed by the earlier layers of the neural network and can be trained jointly with mean squared error loss. Notice a key difference between the multi-labeled logistic regression and the naive classification (ignoring the order and treating all ranks as separate classes): The loss for \( \hat{Y} \neq Y \) is constant in the naive classification but proportional to \(|\hat{Y} - Y|\) in the multi-labeled logistic regression.

J. Cheng et al.’s (2008) final layer was preceded by a simple feed-forward network. In our case, word embeddings and GF-RNN allow us to construct a feature vector of fixed length from the texts of any escort ad, so we can simply attach the multi-labeled logistic regression layer to the output of GF-RNN to complete an ordinal regression neural network for texts.

Besides adapting the ordinal regression neural network to text data, we resolve the conflicting predictions problem of J. Cheng et al.’s model: J. Cheng et al. and researchers who later adopted their ideas (Irsoy and Cardie, 2015; Niu et al., 2016) acknowledged that the predictions from the binary classifiers could sometimes be conflicting with one another.
E.g., the output of the $i$-th classifier, $\hat{f}_i(X) = \hat{P}(Y > i | X)$, could be smaller than the output of the $(i + 1)$-th classifier, $\hat{f}_{i+1}(X) = \hat{P}(Y > i + 1 | X)$. Z. Niu et al. (2016) claimed that eliminating the conflicts would significantly increase the training complexity and opted to ignore them. We propose an effective and computationally efficient solution to avoid the conflicting predictions: Penalize such conflicts in the training phase by adding

$$C(X; \lambda) = \lambda \sum_{i=1}^{k-2} \max \{ \hat{f}_{i+1}(X) - \hat{f}_i(X), 0 \}$$

to the loss function for a sample $X$, where $\lambda$ is the penalty parameter. We will see in the next section that adding this penalty makes the predictions from the binary classifiers more realistic/interpretable, although the improvement in the model performance is not significant.

All three components of our model (word embeddings, GF-RNN, and multi-labeled logistic regression layer) can be trained jointly, with word embeddings optionally held fixed or given a smaller learning rate for fine-tuning. The hyperparameters for all components are given in Appendix C.2. They were selected according to either literature or grid-search.

### 3.4 Experiments

We will first describe the datasets we use to train and evaluate our models. Then we will show a detailed comparison of our proposed model against commonly used ordinal regression models as well as the previous state-of-the-art classification model by E. Tong et al. (2017). To assess the effect of each component in our model, we will perform an ablation test where the components are swapped by their more standard alternatives one by one. Finally, we will conduct an emoji analysis using the word embeddings trained on raw escort ads.
3.4.1 Datasets

We use raw texts scraped from Backpage and TNABoard to pre-train the word embeddings, and use the same labeled texts E. Tong et al. (2017) used to conduct model comparisons.

The raw text dataset consists of 44,105 ads from TNABoard and 124,220 ads from Backpage. Data cleaning/preprocessing includes joining the title and the body of an ad; adding white space around every emoji so that it can be tokenized properly; stripping tabs, line breaks, punctuations, and extra white space; removing phone numbers; and converting all letters to lower case. We have checked the raw dataset to ensure that it has no overlap with the labeled dataset to avoid bias in test accuracy. While it is possible to scrape more raw data, we did not observe significant improvements in model performances when the size of raw data increased from ~70,000 to ~170,000, hence we will assume that the current raw dataset is sufficiently large.

The labeled dataset is called Trafficking10k. It consists of 12,350 ads from Backpage labeled by experts in human trafficking detection (Tong et al., 2017). Each label is one of seven ordered levels of likelihood that the corresponding ad comes from a human trafficker. The descriptions and distributions of labels are in Table 3.1. The original Trafficking10k includes both texts and images, but as mentioned in Section 3.1, only the texts will be used in this case. We apply the same preprocessing to Trafficking10k as we did to raw data.

Table 3.1 Description and distribution of labels in Trafficking10k.

<table>
<thead>
<tr>
<th>Label Description</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>1,977</td>
<td>1,904</td>
<td>3,619</td>
<td>796</td>
<td>3,515</td>
<td>457</td>
<td>82</td>
</tr>
</tbody>
</table>
3.4.2 Comparison to Baselines

We compare our proposed ordinal regression neural network (ORNN) to Immediate-Threshold ordinal logistic regression (IT) (Rennie and Srebro, 2005), All-Threshold ordinal logistic regression (AT) (Rennie and Srebro, 2005), Least Absolute Deviation (LAD) (Bloomfield and Steiger, 1980; Narula and Wellington, 1982), and multi-class logistic regression (MC) that ignores the ordering. The primary evaluation metrics are Mean Absolute Error (MAE) and macro-averaged Mean Absolute Error (MAE_M) (Baccianella et al., 2009). Refer to Section 3.2 for a brief description of the baseline models and evaluation metrics. To compare our model with the previous state-of-the-art classification model for escort ads, the Human Trafficking Deep Network (HTDN) (Tong et al., 2017), we will also polarize the true and predicted labels into two classes, “1-4: Unlikely” and “5-7: Likely”; then we compute the binary classification accuracy (Acc.) as well as the weighted binary classification accuracy (Wt. Acc.) given by

\[
Wt. \text{ Acc.} = \frac{1}{2} \left( \frac{\text{True Positives}}{\text{Total Positives}} + \frac{\text{True Negatives}}{\text{Total Negatives}} \right).
\]

Note that for applications in human trafficking detection, MAE and Acc. are of primary interest. Whereas for a more general comparison among the models, the class imbalance robust metrics, MAE_M and Wt. Acc., might be more suitable. Bootstrapping or increasing the weight of samples in smaller classes can improve MAE_M and Wt. Acc. at the cost of MAE and Acc..

The text data need to be vectorized before they can be fed into the baseline models (whereas vectorization is built into ORNN). The standard practice is to tokenize the texts using n-grams and then create weighted term frequency vectors using the term frequency...
(TF)-inverse document frequency (IDF) scheme (Beel et al., 2016; Manning et al., 2009). The specific variation we use is the recommended unigram + sublinear TF + smooth IDF (Manning et al., 2009; Pedregosa et al., 2011) (see Appendix C.3 for details). Dimension reduction techniques such as Latent Semantic Analysis (Dumais, 2004) can be optionally applied to the frequency vectors, but B. Schuller et al. (2015) concluded from their experiments that dimension reduction on frequency vectors actually hurts model performance, which our preliminary experiments agree with.

All models are trained and evaluated using the same (w.r.t. data shuffle and split) 10-fold cross-validation (CV) on Trafficking10k, except for HTDN, whose result is read from the original paper (Tong et al., 2017). During each train-test split, 2/9 of the training set is further reserved as the validation set for tuning hyperparameters such as L2-penalty in IT, AT and LAD, and learning rate in ORNN. So the overall train-validation-test ratio is 70%-20%-10%. We report the mean metrics from the CV in Table 3.2. Previous research has pointed out that there is no unbiased estimator of the variance of CV statistics (Bengio and Grandvalet, 2004), and bootstrap confidence intervals are prohibitively expensive for deep learning models. So we will just report the point estimates.

We can see that ORNN has the best MAE, MAE$^M$ and Acc. as well as a close 2nd best Wt. Acc. among all models. Its Wt. Acc. is a substantial improvement over HTDN despite the fact that the latter use both text and image data. It is important to note that HTDN is trained using binary labels, whereas the other models are trained using ordinal labels and then have their ordinal predictions converted to binary predictions. This is most likely the reason that even the baseline models except for LAD can yield better Wt. Acc. than HTDN, confirming our earlier claim that polarizing the ordinal labels during training may lead to

---

6The authors of HTDN used a single train-validation-test split instead of CV.

7Reporting only point estimates with a single train-test split or CV is common in deep learning literature due to the computational cost.
Table 3.2 Comparison of our ordinal regression neural network (ORNN) against Immediate-Threshold ordinal logistic regression (IT), All-Threshold ordinal logistic regression (AT), Least Absolute Deviation (LAD), multi-class logistic regression (MC), and the Human Trafficking Deep Network (HTDN) in terms of Mean Absolute Error (MAE), macro-averaged Mean Absolute Error (MAE^M), binary classification accuracy (Acc.) and weighted binary classification accuracy (Wt. Acc.). The results are averaged across 10-fold CV on Trafficking10k except for HTDN. The best result is highlighted in green with the 2nd best in blue.

<table>
<thead>
<tr>
<th>Model</th>
<th>MAE</th>
<th>MAE^M</th>
<th>Acc.</th>
<th>Wt. Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORNN</td>
<td>0.769</td>
<td>1.238</td>
<td>0.818</td>
<td>0.772</td>
</tr>
<tr>
<td>IT</td>
<td>0.807</td>
<td>1.244</td>
<td>0.801</td>
<td>0.781</td>
</tr>
<tr>
<td>AT</td>
<td>0.778</td>
<td>1.246</td>
<td>0.813</td>
<td>0.755</td>
</tr>
<tr>
<td>LAD</td>
<td>0.829</td>
<td>1.298</td>
<td>0.786</td>
<td>0.686</td>
</tr>
<tr>
<td>MC</td>
<td>0.794</td>
<td>1.286</td>
<td>0.804</td>
<td>0.767</td>
</tr>
<tr>
<td>HTDN</td>
<td>-</td>
<td>-</td>
<td>0.800</td>
<td>0.753</td>
</tr>
</tbody>
</table>

information loss.

3.4.3 Ablation Test

To ensure that we did not unnecessarily complicate our ORNN model, and to assess the impact of each component on the final model performance, we will perform an ablation test. Using the same CV and evaluation metrics, we make the following replacements separately and re-evaluate the model:

1. Replace word embeddings pre-trained from skip-gram model with randomly initialized word embeddings;

2. replace gated-feedback recurrent neural network with long short-term memory network (LSTM);

3. disable batch normalization;
4. disable residual connection;

5. replace the multi-labeled logistic regression layer with a softmax layer (i.e., let the model perform classification, treating the ordinal response variable as a categorical variable with \( k \) classes);

6. replace the multi-labeled logistic regression layer with a 1-dimensional linear layer (i.e., let the model perform regression, treating the ordinal response variable as a continuous variable) and round the prediction to the nearest integer during testing;

7. set the order penalty to 0.

The results are shown in Table 3.3

**Table 3.3** Ablation test. Except for models everything is the same as Table 3.2.

<table>
<thead>
<tr>
<th>Model</th>
<th>MAE</th>
<th>MAE(^M)</th>
<th>Acc.</th>
<th>Wt. Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Original ORNN</td>
<td>0.769</td>
<td>1.238</td>
<td>0.818</td>
<td>0.772</td>
</tr>
<tr>
<td>1. Random Embeddings</td>
<td>0.789</td>
<td>1.254</td>
<td>0.810</td>
<td>0.757</td>
</tr>
<tr>
<td>2. LSTM</td>
<td>0.778</td>
<td>1.261</td>
<td>0.815</td>
<td>0.764</td>
</tr>
<tr>
<td>3. No Batch Norm.</td>
<td>0.780</td>
<td>1.311</td>
<td>0.815</td>
<td>0.754</td>
</tr>
<tr>
<td>4. No Res. Connect.</td>
<td>0.775</td>
<td>1.271</td>
<td>0.816</td>
<td>0.766</td>
</tr>
<tr>
<td>5. Classification</td>
<td>0.785</td>
<td>1.253</td>
<td>0.812</td>
<td>0.780</td>
</tr>
<tr>
<td>6. Regression</td>
<td>0.850</td>
<td>1.279</td>
<td>0.784</td>
<td>0.686</td>
</tr>
<tr>
<td>7. No Order Penalty</td>
<td>0.769</td>
<td>1.251</td>
<td>0.818</td>
<td>0.769</td>
</tr>
</tbody>
</table>

The original ORNN once again has all the best metrics except for Wt. Acc. which is the 2nd best. This shows that each improvement we add indeed has its contributions. Note that if we disregard the ordinal labels and perform classification or regression, MAE falls off by a large margin. Setting order penalty to 0 does not deteriorate the performance too
much, however, the percent of conflicting binary predictions (see Section 3.3.3) rises from 1.4% to 5.2%. So adding an order penalty helps produce more interpretable results\textsuperscript{8}.

### 3.4.4 Emoji Analysis

The fight against human traffickers is dynamically evolving with adversarial opponents. Traffickers often avoid using explicit keywords when advertising victims, but instead use acronyms, intentional typos, and emojis. The law enforcement can compile a lexicon of trafficking flags mapping certain emojis to their potential true meanings (e.g., the cherry emoji can indicate an underaged victim), but compiling such a lexicon manually is expensive, incomprehensive, and may require insider information hard to obtain. To make matters worse, traffickers change their dictionaries over time and keep switching to new emojis to replace certain keywords. In such a dynamic and adversarial environment, the need for a data driven approach in updating the existing lexicon is evident.

As mentioned in Section 3.3.1, training a skip-gram model on the raw texts can map words (including emojis) used in similar contexts to similar numeric vectors. Besides using the vectors learned from the raw escort ads to train ORNN, we can directly visualize the vectors for the emojis to help identify their relationships, by mapping the vectors to a 2-dimensional space using t-SNE\textsuperscript{9} (van der Maaten and Hinton, 2008) (Figure 3.6). See Appendix C.4 for the algorithm of t-SNE.

We can first empirically assess the quality of the emoji map by noting that similar emojis do seem clustered together: the smileys near the coordinate (2, 3), the flowers near (-6, -1), the heart shapes near (-8, 1), the phones near (-2, 4) and so on. It is worth emphasizing that

---

\textsuperscript{8}It is possible to increase the order penalty to further reduce or eliminate conflicting predictions, but we found that a large order penalty harmed model performance.

\textsuperscript{9}t-SNE is reported to produce better 2-dimensional visualizations than other dimension reduction techniques such as Principal Component Analysis, Multi-dimensional Scaling, and Local Linear Embedding (van der Maaten and Hinton, 2008).
Figure 3.6 Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis that appeared most frequently in the escort ads we scraped are shown out of the total 968 emojis that appeared. See Appendix C.5 for maps of other emojis.

The skip-gram model learns the vectors of these emojis based on their contexts in escort ads and not their visual representations, so the fact that the visually similar emojis are close to one another in the map suggests that the vectors have been learned as desired.

The emoji map can assist anti-trafficking experts in expanding the existing lexicon of trafficking flags. For example, according to the lexicon we obtained from Global Emancipation Network\(^\text{10}\), the cherry emoji and the lollipop emoji are both flags for underaged victims. In the map, right next to these two emojis are the porcelain dolls emoji, the grapes emoji, the strawberry emoji, the candy emoji, and the ice cream emojis, indicating that they are all used in similar contexts and perhaps should all be flags for underaged victims

\(^{10}\)Global Emancipation Network is a non-profit organization dedicated to combating human trafficking. For more information see https://www.globalemancipation.ngo.
in the updated lexicon.

If we re-train the skip-gram model and update the emoji map periodically on new escort ads, when traffickers switch to new emojis, the map can link the new emojis to the old ones, assisting anti-trafficking experts in expanding the lexicon of trafficking flags.

3.5 Discussion

Human trafficking is a form of modern day slavery that victimizes millions of people. It has become a norm for sex traffickers to use escort websites to openly advertise the victims. We designed an ordinal regression neural network (ORNN) to predict the likelihood that an escort ad comes from a trafficker, which can drastically narrow down the suspicious ads for the law enforcement. Our ORNN achieved the state-of-the-art performance on Trafficking10k (Tong et al., 2017), beating all of the baseline ordinal regression models as well as improving the classification accuracy over the Human Trafficking Deep Network (Tong et al., 2017). We also conducted an emoji analysis and showed how to use word embeddings learned from raw texts to help expand the lexicon of trafficking flags.

There are several ways to extend our work on ordinal regression. We used gated-feedback recurrent neural network as the 2nd component in our model (Section 3.3.2), which proved to be an improvement over long short-term memory network (LSTM) according to the ablation test (Section 3.4.3), but other options are possible, such as bi-directional LSTM (Graves et al., 2005), convolutional neural network (Kim, 2014), recursive neural network (Socher et al., 2013), or an ensemble of these models. A comprehensive evaluation of these alternatives could be of interest. We may also apply our ORNN to other domains such as sentiment analysis in Twitter (Rosenthal et al., 2017), after re-training or importing word embeddings accordingly.
As for future work in trafficking detection, we can design multi-modal ordinal regression networks that utilize both image and text data, with the current ORNN as a sub-model. To aid that effort, we will coordinate with Raleigh Police Department to create a larger labeled dataset of escort ads. But given the time and resources required to label escort ads, we may explore more unsupervised learning algorithms or supervised models that can be transferred from other tasks, such as using object detection (Ren et al., 2015) and matching algorithms to match hotel rooms in the images.


THORN and Bouché, V. Survivor insights: The role of technology in domestic minor sex trafficking. THORN (2018).


One of the most common reinforcement learning algorithms is Q-learning, which itself has many variations. Here we will present the setup and the classic variation.

Let $(A_t, S_{t+1}, U_t)$ be an MDP. The value of a state-action pair under a policy $\pi$, referred to as the Q-function, is the discounted mean utility if the current state is $s$, current action is $a$, and the agent follows $\pi$ afterwards:

$$Q^\pi(s, a) = \mathbb{E}\left\{ \sum_{t \geq 1} \gamma^{t-1} U^*(\pi)|S^1 = s, A^1 = a \right\},$$

where $\gamma \in (0, 1)$ is the discount factor. An optimal policy $\pi^{opt}$ yields the largest value for
every state-action pair. Denote the corresponding Q-function as

\[ Q_{\text{opt}}(s, a) = Q_{\pi_{\text{opt}}}^\pi(s, a) = \max_{\pi} Q^\pi(s, a). \]

If \( Q_{\text{opt}}(s, a) \) is known for all \((s, a)\), then an optimal policy can be defined as

\[ \pi_{\text{opt}}(s) = \arg\max_{a \in \mathcal{A}} Q_{\text{opt}}(s, a). \]

\( Q_{\text{opt}} \) satisfies the Bellman Optimality Equations (BOE):

\[ Q_{\text{opt}}(s, a) = \mathbb{E}\left\{ U' + \gamma \max_{a' \in \mathcal{A}} Q_{\text{opt}}(s_{t+1}, a') \mid S_t = s, A_t = a \right\}. \]

In practice, one often cannot obtain \( Q_{\text{opt}} \) by solving the above equations, because computing the right-hand-side requires the underlying model of the MDP, which is often unknown. Besides, solving a huge linear system can be costly.

Q-learning is a stochastic optimization algorithm that doesn’t require knowing the transition model or solving a linear system. The update step for the classic Q-learning, Watkin’s Q-learning (Sutton and Barto, 1998), for finite state and action spaces, is as follows:

\[ Q^{k+1}(s', a') \leftarrow Q^k(s', a') + \alpha \left\{ u' + \gamma \max_a Q^k(s_{t+1}, a) - Q^k(s', a') \right\}, \]

where \( \alpha \) is the learning rate. If the state space is continuous, one may approximate \( Q(s, a) \) with a parametric function \( F(s, a; \theta) \) and update the parameters instead:

\[ \theta^{k+1} \leftarrow \theta^k + \alpha \left\{ u' + \gamma \max_a F(s_{t+1}, a; \theta^k) - F(s', a'; \theta^k) \right\} \cdot \nabla_{\theta} F(s', a'; \theta^k). \]
In the simulation experiments from Section 2.3, we evaluate the mean marginal outcome for both linear and neural net function approximations of \( Q(s, a) \). As the size of data is not huge, instead of implementing Watkin’s Q-learning, we simply minimize the sum of squares of the temporal difference errors with a robust optimizer:

\[
\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} \sum_{t=1}^{T-1} \left\{ u_i^t + \gamma \max_a F(s_i^{t+1}, a; \theta) - F(s_i^t, a^t; \theta) \right\}^2.
\]

The choices of optimizers considered were Nelder-Mead (Nelder and Mead, 1965; Wright, 1996), BFGS (Nocedal and Wright, 2006), Powell (Powell, 1964; Press et al., 2007), CG (Nocedal and Wright, 2006), and SLSQP (Kraft, 1988), all implemented in the Python library SciPy (Jones et al., 2001–). We selected SLSQP based on performance from preliminary experiments.

### A.2 Proof of Theorem 1.3.2

**Proof.** First we show that the process \((A_t, S_{\phi_t}^{t+1}, \bar{U}^t)\) induced by \((\phi, \Pi_{\phi,m_{sb}})\) satisfies (S1). For any \( t \in \mathbb{N} \) and measurable subset \( G \in \mathbb{R}^q \),

\[
P(S^t_{\phi} \in G^{t+1} | S^t_{\phi}, A^t)
= E\{P(S^t_{\phi} \in G^{t+1} | S^t_{\phi}, S^t_{\phi}, A^t) | S^t_{\phi}, A^t\} (by \text{Markov property of the original process})
= E\{P(S^t_{\phi} \in G^{t+1} | S^t_{\phi}, A^t) | S^t_{\phi}, A^t\} (by (1.2))
= P(S^t_{\phi} \in G^{t+1} | S^t_{\phi}, A^t).
\]
Also note that \( E[P(S_{\phi}^{t+1} \in \mathcal{G}_{\phi}^{t+1} | S^t, S_{\phi}^t, \bar{A}^t | S_{\phi}^t, \bar{A}^t) \) does not depend on \( t \) by homogeneity of the original process. Thus the induced process is Markov and homogeneous.

Next we show that the induced process satisfies (S2). Let \( Q^{\text{opt}}(s, a) \) be defined as before. Then we have

\[
Q^{\text{opt}}(s, a) = E[U^t + \gamma \max_{a'} Q^{\text{opt}}(S^{t+1}, a') | S^t = s, A^t = a] \quad \text{(by BOE)}
\]

\[
= E[U^t + \gamma \max_{a'} Q^{\text{opt}}(S^{t+1}, a') | S^t = s, S_{\phi}^t = s_{\phi}, A^t = a]
\]

\[
= E[U^t + \gamma \max_{a'} Q^{\text{opt}}(S^{t+1}, a') | S_{\phi}^t = s_{\phi}, A^t = a] \quad \text{(by (2))}
\]

\[
= Q_{\phi}^{\text{opt}}(s_{\phi}, a), \quad \text{and}
\]

\[
\pi^{\text{opt}}(s) = \arg\max_a Q^{\text{opt}}(s, a) = \arg\max_a Q_{\phi}^{\text{opt}}(s_{\phi}, a) = \pi_{\phi}^{\text{opt}}(s_{\phi}).
\]

\[\square\]

### A.3 Proof of Corollary 1.3.3

**Proof.** By assumption \((\phi_0, \Pi_{\phi_0, \text{msrbl}})\) induces a sufficient MDP for \( \pi^{\text{opt}} \) within \( \Pi_{\text{msrbl}} \), then by definition the process \((A^t, S_{\phi_0}^{t+1}, U^t)\) is Markov and homogeneous, and there exists \( \pi^{\text{opt}} = \pi_{\phi_0}^{\text{opt}} \circ \phi_0 \).

Define \( \overline{\phi}_k = \phi_k \circ \cdots \circ \phi_0 \). By (3) and Theorem 3.2, \((\phi_1, \Pi_{\phi_1, \text{msrbl}})\) induces a sufficient MDP for \( \pi_{\phi_0}^{\text{opt}} \) within \( \Pi_{\phi_0, \text{msrbl}} \). Then the process \((\overline{A}^t, S_{\phi_1}^{t+1}, \overline{U}^t)\) is Markov and homogeneous, and there exists \( \pi_{\phi_0}^{\text{opt}} = \pi_{\overline{\phi}_1}^{\text{opt}} \circ \phi_1 \). Thus,

\[
\pi^{\text{opt}} = \pi_{\phi_0}^{\text{opt}} \circ \phi_0 = \pi_{\overline{\phi}_1}^{\text{opt}} \circ \phi_1 \circ \phi_0 = \pi_{\overline{\phi}_1}^{\text{opt}} \circ \overline{\phi}_1.
\]

Therefore \((\phi_1, \Pi_{\overline{\phi}_1, \text{msrbl}})\) induces a sufficient MDP for \( \pi^{\text{opt}} \) within \( \Pi_{\text{msrbl}} \). The 2nd part of the
corollary follows from induction. 

\textbf{A.4 Proof of Lemma 1.3.4}

\textit{Proof.} We show that (i) $\Rightarrow Y^{t+1} \perp S^t | S^t_{\phi}, A^t$:

\[
\{Y^{t+1} - \mathbb{E}(Y^{t+1} | S^t_{\phi}, A^t)\} \perp S^t | A^t \\
\Rightarrow \{Y^{t+1} - \mathbb{E}(Y^{t+1} | S^t_{\phi}, A^t)\} \perp (S^t, S^t_{\phi}) | A, \\
\Rightarrow \{Y^{t+1} - \mathbb{E}(Y^{t+1} | S^t_{\phi}, A^t)\} \perp S^t | S^t_{\phi}, A^t \\
\Rightarrow Y^{t+1} \perp S^t | S^t_{\phi}, A^t.
\]

The 1st implication follows from the fact that $S^t_{\phi}$ is a transformation of $S^t$. The 2nd implication follows from the fact that $X \perp Y, Z \Rightarrow f(X|Y, Z) = f(X) = f(X|Z) \Rightarrow X \perp Y|Z$

for random variables $X, Y,$ and $Z$. The 3rd implication follows from the fact that $\mathbb{E}(Y^{t+1} | S^t_{\phi}, A^t)$ is constant conditional on $S^t_{\phi}$ and $A^t$.

Similarly, one can show that (ii) $\Rightarrow Y^{t+1} \perp S^t | S^t_{\phi}, A^t$. 

\[\square\]
A.5 Proof of Theorem 1.3.5

Proof. First we show that the process \((\overline{A}^t, \overline{S}^t, \overline{U}^t)\) induced by \((\phi, \Pi, m_{sbi})\) satisfies (S1). For any \(t \in \mathbb{N}\) and measurable subset \(G \in \mathbb{R}^q\),

\[
P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t)
= \mathbb{E}\{P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t) | \overline{S}^t, \overline{A}^t\}
= \mathbb{E}\{P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t) | \overline{S}^t, \overline{A}^t\} \quad \text{(by Markov property of the original process)}
= \mathbb{E}\{P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t) | \overline{S}^t, \overline{A}^t\} \quad \text{(by (1.4))}
= P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t).
\]

Also note that \(\mathbb{E}\{P(\overline{S}^{t+1} \in G | \overline{S}^t, \overline{A}^t) | \overline{S}^t, \overline{A}^t\}\) does not depend on \(t\) by homogeneity of the original process. Thus the induced process is Markov and homogeneous.

Next we show that the induced process satisfies (S2). Define

\[
Q^{opt,1}(s^t, a^t) := \mathbb{E}\{U(s^t, A^t, S^{t+1}) | S^t = s^t, A^t = a^t\}
= \mathbb{E}\{U(s^t, A^t, S^{t+1}) | S^t = s^t, A^t = a^t\} \quad \text{(by (1.4))}
= \mathbb{E}\{U(s^t, A^t, S^{t+1}) | S^t = s^t, A^t = a^t\}
= Q^{opt,1}(s^t, a^t).
\]
For $T \geq 2$, define

$$Q^{\text{opt}, T}(s', a') = \mathbb{E}\{U(S', A', S^{t+1}) + \gamma \max_{a'} Q^{\text{opt}, T-1}(S^{t+1}, a') \mid S' = s', A' = a'\}$$

$$= \mathbb{E}\{U(S', A', S^{t+1}) + \gamma \max_{a'} Q^{\text{opt}, T-1}(S^{t+1}, a') \mid S' = s', A' = a'\} \quad \text{(by induction)}$$

$$= \mathbb{E}\{U(S', A', S^{t+1}) + \gamma \max_{a'} Q^{\text{opt}, T-1}(S^{t+1}, a') \mid S' = s', A' = a'\} \quad \text{(by (1.4))}$$

$$= Q^{\text{opt}, T}(s', a').$$

From now on we use $U^t = U(S', A', S^{t+1}) = U(S'_t', A'_t', S^{t+1})$ for short.

**Claim:** $\sup_{a^t, s^t} |Q^{\text{opt}, T}(s^t, a^t) - Q^{\text{opt}}(s^t, a^t)| = O(\gamma^T)$.

Given that the utilities are bounded, we have $\sup_{s^t, a^t, S^{t+1}} |u^t| \leq C_1$, and consequently, $\sup_{s^t, a^t} |Q^{\text{opt}}(s^t, a^t)| \leq C_2$, for some constants $C_1$ and $C_2$. Thus,

$$\sup_{a^t, s^t} \left| Q^{\text{opt}, 1}(s^t, a^t) - Q^{\text{opt}}(s^t, a^t) \right|$$

$$= \sup_{a^t, s^t} \left| \mathbb{E}\{U^t \mid S' = s', A' = a'\} - \mathbb{E}\{U^t + \gamma \max_{a'} Q^{\text{opt}}(S^{t+1}, a') \mid S' = s', A' = a'\} \right|$$

$$= \sup_{a^t, s^t} \gamma \left| \mathbb{E}\{\max_{a'} Q^{\text{opt}}(S^{t+1}, a') \mid S' = s', A' = a'\} \right|$$

$$\leq \gamma C_2 = O(\gamma).$$

For $T \geq 2$, assume that $\sup_{a^t, s^t} |Q^{\text{opt}, T-1}(s^t, a^t) - Q^{\text{opt}}(s^t, a^t)| \leq \gamma^{T-1} C_2$, then

$$\sup_{s^t} \left| \max_{a'} Q^{\text{opt}, T-1}(s^t, a') - \max_{a'} Q^{\text{opt}}(s^t, a') \right| \leq \gamma^{T-1} C_2,$$
and

\[
\sup_{a', s'} \left| Q_{\text{opt}, T}(s', a') - Q_{\text{opt}}(s', a') \right| \\
= \sup_{a', s'} \left| \mathbb{E}\left[U + \gamma \max_{a'} Q_{\text{opt}, T-1}(s'^{t+1}, a') - U - \gamma \max_{a'} Q_{\text{opt}}(s'^{t+1}, a') \mid s'^t = s', A'^t = a' \right] \right| \\
= \sup_{a', s'} \gamma \left| \mathbb{E}\left[\max_{a'} Q_{\text{opt}, T-1}(s'^{t+1}, a') - \max_{a'} Q_{\text{opt}}(s'^{t+1}, a') \mid s'^t = s', A'^t = a' \right] \right| \\
\leq \gamma (\gamma^{T-1} C_2) = \Theta(\gamma^T), \quad \text{(by induction)}
\]

which proves the claim. Therefore, \( \lim_{T \to \infty} Q_{\text{opt}, T}(s, a) = Q_{\text{opt}}(s, a) \) for all \( s \) and \( a \). Similarly, \( \lim_{T \to \infty} Q_{\phi, T}(s, a) = Q_{\phi}(s, a) \) for all \( s \) and \( a \). And we have

\[
\pi_{\text{opt}}(s) = \arg\max_a Q_{\text{opt}}(s, a) = \arg\max_a \lim_{T \to \infty} Q_{\text{opt}, T}(s, a) \\
= \arg\max_a \lim_{T \to \infty} Q_{\phi, T}(s, a) = \arg\max_a Q_{\phi}(s, a) = \pi_{\phi}(s).
\]

\[ \square \]

### A.6 Proof of Theorem 1.3.6

**Proof.** Let \( D \) be the set of all indices. Under the assumption that joint dependence implies marginal dependence, by construction \( Y_{j_k-1}^{t+1} \perp D \setminus j_k \mid A^t \). Thus \( Y_{j_k-1}^{t+1} \perp S^t \mid S_{j_k}^t, A^t \). Because \( J_{k-1} = J_K \), the result follows. \[ \square \]
A.7 Distance Covariance Test of Dependence

We use the distance covariance (dCov) test of dependence (Székely et al., 2007; Székely and Rizzo, 2009) to verify the conditions in Lemma 1.3.4. See Algorithm 3 for how the p-value is computed.

Algorithm 3 Computing the p-value for Distance Covariance Test of Dependence

Input: data \( \{(X_i, Y_i)\}_{i=1}^n \) where \( (X_i, Y_i) \) are i.i.d. pairs of random vectors from a joint distribution; number of permutation runs \( B \).

1. for \( b = 0, \ldots, B \) do
2. if \( b = 0 \) then
3. Set \( \{(X'_i, Y_i)\}_{i=1}^n = \{(X_i, Y_i)\}_{i=1}^n \) (no permutation)
4. else
5. Randomly shuffle the order of \( X_i \) and get \( \{(X'_i, Y_i)\}_{i=1}^n \)
6. Compute the Euclidean distance matrices \( (a_{ij}) = (\|X'_i - X'_j\|_2^2) \) and \( (b_{ij}) = (\|Y_i - Y_j\|_2^2) \)
7. Compute \( \bar{a}_i = \frac{1}{n} \sum_{j=1}^n a_{ij} \) for \( i = 1, \ldots, n \)
8. Compute \( \bar{a}_{ij} = \frac{1}{n} \sum_{i=1}^n a_{ij} \) for \( j = 1, \ldots, n \)
9. Compute \( \bar{a}_i = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \)
10. Compute \( A_{ij} = a_{ij} - \bar{a}_i - \bar{a}_j + \bar{a} \) for \( i, j = 1, \ldots, n \)
11. Similarly, compute \( B_{ij} \) for \( i, j = 1, \ldots, n \)
12. Compute dCov \( C_b = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n A_{ij} B_{ij} \)
13. Compute the p-value for the null hypothesis that \( X \) and \( Y \) are independent:

\[
p\text{-value} = \frac{|\{C_b : b > 0, C_b \geq C_0\}|}{B}
\]

Output: p-value

1For dCov test, the null hypothesis is that the variables of interest are independent. We will accept that the null is true if dCov test fails to reject. Even though failing to reject the null doesn't prove that the null is true in general, dCov test is very powerful so we will use failing to reject as evidence that supports the null when the sample size is reasonably large.
In our implementation we set \( B = 200 \).

Note that the data across time points are not independent in Markov Decision Processes. See A.8 for how to create a pooled p-value.

### A.8 Hypothesis Test with Dependent Observations

To apply Lemma 1.3.4, we wish to check the dependence between \( \left\{ Y_{t+1} - E(Y_{t+1}|S_t^t, A_t^t) \right\} \) and \( S^t \) within levels of \( A^t \) across time points. For Markov Decision Processes (MDP) data, the observations are i.i.d. across subjects at a fixed time point, but are dependent across time points for a single subject. To resolve the issue of dependent observations, for each fixed time point \( t \) and within levels of \( A^t \), we will perform a separate dCov test (see Appendix A.7) on \( H_0 : \left\{ Y_{t+1} - E(Y_{t+1}|S_t^t, A_t^t) \right\} \perp \perp S^t \) and obtain \( \hat{p}_{H_t}^t \). Then pool all the p-values based on Theorem A.8.1 (Rüger, 1978; Vovk, 2012):

**Theorem A.8.1.** Let \( p_1, p_2, \ldots, p^K \) be \( K \geq 2 \) p-values of different statistical tests on the same hypothesis. Then for any fixed \( k \in \{ 1, \ldots, K \} \),

\[
  p_{\text{pooled}}(p_1, \ldots, p^K) := \frac{K}{k} p^{(k)}
\]

is a valid p-value for the same hypothesis, where \( p^{(k)} \) is \( k \)-th order statistic of \( p_1, \ldots, p^K \).

We summarize the steps in Algorithm 4.

\( u = 1 \) would correspond to the Bonferroni correction. We don’t set \( u = 1 \) because dCov test is a permutation test and sometimes 0-valued p-values would appear, making the Bonferroni correction too aggressive to reject the null (note that if \( \hat{p}^{(u)} = 0 \), \( \hat{p}_{\text{pooled}} = 0 \), making the adjustment multiplier \( T/u \) useless). To avoid having \( \hat{p}^{(u)} \) by chance, we set \( u = \lceil T/20 \rceil + 1 \) across all settings.
Algorithm 4 Testing for Condition (i) in Lemma 1.3.4 With MDP Data

Input: data \( \{ \mathbf{S}_i^{T+1}, \mathbf{A}_i^T, \mathbf{U}_i^T \}_{i=1}^n \); significance level \( \alpha \); order level \( u \); dimension reduction function \( \phi \); \( \{ \mathbb{E}[\mathbf{Y}_t^{i+1}|\mathbf{S}_{\phi}^t, A^t] \}_{i=1}^n \) (in practice both this and \( \phi \) will be estimated).  

1. Within levels of \( A^t \) (i.e., subset data for each level of \( A^t \), but for simplicity we will omit the subscript \( \{ A^t = a \} \) for all the variables below) do 

2. for Fixed \( t = 1, \ldots, T \) do

3. Perform dCov test on \( \mathbb{E}[\mathbf{Y}_t^{i+1} - \mathbb{E}[\mathbf{Y}_t^{i+1}|\mathbf{S}_{\phi}^t, A^t]] \) and \( \mathbf{S}^t \) and obtain p-value \( \hat{p}_t \)

4. Compute pooled p-value 

\[
\hat{p}_{\text{pooled}}(\hat{p}_1, \ldots, \hat{p}_T) := \frac{T}{u} \hat{p}^{(u)}. 
\]

5. if \( \hat{p}_{\text{pooled}}(\hat{p}_1, \ldots, \hat{p}_T) < \alpha \) then

6. Reject \( H_0 : \{ \mathbf{Y}_t^{i+1} - \mathbb{E}[\mathbf{Y}_t^{i+1}|\mathbf{S}_{\phi}^t, A^t] \} \perp \mathbf{S}^t | A^t \), stop.
B.1 Description of State Variables in BASICS-Mobile

- wkend: indicator of whether it is currently weekend
- age: age of the subject
- male: indicator of whether the subject is male
- past90Smk: how many days in the past 90 days has the subject smoked
- quitSmk: indicator of whether the subject has tried to quit smoking before
- baseCigs: baseline (before the study) number of cigarettes/day
- urge: self-reported current level of urge to smoke
• wouldSmk: self-reported indicator of whether the subject would smoke right now if possible
• smkMood: self-reported indicator of whether smoking right now would make the subject feel less depressed/tense/anxious
• fixMind: self-reported current level of need to control thoughts
• stress: self-reported current level of stress
• tired: self-reported current level of fatigue
• cigs: self-reported number of cigarettes smoked since the last report
• drinks: self-reported number of drinks had since the last report
• lastCigs: cigs from the previous time point

B.2 How the Variables from BASICS-Mobile Are Partitioned

The variable that records the number of cigarettes smoked in between reports, CIGS, ranges from 0 to 20. The values imputed by local polynomial regression have decimals and are rounded to the nearest integers. CIGS has 18 unique values after rounding, which is still the most among all variables. We divide CIGS by 20 to rescale it to [0, 1], and the rescaled unique values of CIGS will be used as the levels for all variables. All other variables are rescaled to [0, 1] and rounded to the nearest level.
B.3 Testing for Dependence When State Variables Are Discrete or Mixed

So far Chapter 1 and 2 have mostly focused on continuous states: When checking Condition (i) in Lemma 1.3.4, we obtain \( \mathbb{E}(Y_{t+1}^t | S_{\phi}^t, A^t) \) directly from ADNN, then perform dCov test between \( Y_{t+1}^t - \mathbb{E}(Y_{t+1}^t | S_{\phi}^t, A^t) \) and \( S^t \) within levels of \( A^t \). Here we will fill in the details of how to handle discrete or mixed states.

For now let us assume that the states contain only ordinal variables, which is the case for BASICS-Mobile. First, we fit a multivariate probit model (Model 1) using \( S^t \) and \( A^t \) as features and \( Y_{t+1}^t \) as responses and estimate the joint likelihood with pairwise likelihood inference (Pagui and A., 2015), which can be conveniently achieved through the R package \textit{PLordprob} (Pagui et al., 2014). Next, we re-fit the model (Model 2) substituting \( S^t \) with \( \hat{\phi}_{ADNN}(S^t) \) (or in general \( \hat{\phi}(S^t) \) for any dimension reduction function \( \hat{\phi} \) to be evaluated) and estimate its likelihood. Given the likelihoods and parameter counts of Model 1 and Model 2, we can test if Model 2 is a lack-of-fit compared to Model 1. If we restrict \( \hat{\phi}_{ADNN} \) to be linear (use no non-linear activations in the dimension reduction portion of ADNN), then Model 2 can be considered a nested model of Model 1 \(^1\), and the standard Likelihood Ratio Test (LRT) using \( \chi^2 \) distribution can be performed (Wilks, 1938). If a non-linear \( \hat{\phi}_{ADNN} \) is used, we can use a test for non-nested models such as Vuong Test (Vuong, 1989), Cox Test (Cox, 1961), or Clarke Test (Clarke, 2003). If there is no lack of fit, we conclude that Condition (i) in Lemma 1.3.4 is true, otherwise we increase the dimension of \( \hat{\phi}_{ADNN} \) and repeat the process.

\(^1\)Two models are nonnested if one model cannot be reduced to the other model by imposing a set of linear restrictions on the parameter vector" (Clarke, 2003). A pre-constructed linear \( \hat{\phi}_{ADNN} \) essentially imposes a set of linear restrictions on the parameters of Model 1, making Model 2 nested in Model 1.
An obvious drawback of this approach is that the multivariate probit model may not be as flexible as ADNN in predicting \( \mathbb{E}(Y_{t+1}^t \mid S^t, A^t) \), but we need the former to provide the likelihoods used in the tests (in comparison, dCov test for continuous states doesn’t require likelihoods).

If the state variables are categorical but not ordinal, we can replace the multivariate probit model with a multivariate logistic regression model and everything else is similar.

If the states contain a mixture of discrete and continuous variables, we can decompose the conditional joint density

\[
f(Y_{t+1}^t \mid S^t, A^t) = f(Y_{\text{disc.}}_{t+1}^t \mid Y_{\text{cont.}}_{t+1}^t, S^t, A^t) \cdot f(Y_{\text{cont.}}_{t+1}^t \mid S^t, A^t).
\]

Then we can check whether

\[
f(Y_{\text{cont.}}_{t+1}^t \mid S^t, A^t) = f(Y_{\text{cont.}}_{t+1}^t \mid S^t, A^t)
\]

using dCov test and

\[
f(Y_{\text{disc.}}_{t+1}^t \mid Y_{\text{cont.}}_{t+1}^t, S^t, A^t) = f(Y_{\text{disc.}}_{t+1}^t \mid Y_{\text{cont.}}_{t+1}^t, S^t, A^t)
\]

using LRT (or its alternatives for non-nested models).

### B.4 Games Used for Evaluating ConvADNN

Refer to Figure 2.9 to better understand the following descriptions. See the original repository (Tasfi, 2016) for more details and code.

- **Catcher**: The agent controls the paddle (white rectangle) to catch the fruits (red
squares). Possible actions are \{LEFT, RIGHT, NONE\}. The agent gains +1 utility for catching a fruit and −6 for missing one. The game ends when the agent misses a fruit or after 10,000 frames.

- **Pong**: The agent controls the left paddle to catch the ball and score by making the right paddle miss. Possible actions are \{UP, DOWN, NONE\}. The agent gains +1 utility for scoring, +6 for scoring 11 times, −1 for missing the ball, and −6 for missing 11 times. The game ends when the agent scores or misses 11 times.

- **Pixelcopter**: The agent controls the helicopter (white square) to navigate through the tunnel. Possible actions are \{JUMP, NONE\}. The agent gains +1 utility for every segment of terrain it passes and −5 for colliding into any object. The game ends with any collision.

- **FlappyBird**: The agent controls the bird to fly forward in between the pipes. Possible actions are \{JUMP, NONE\}. The agent gains +1 utility for flying through a pair of pipes and −5 for colliding into any object. The game ends with any collision. The color scheme changes randomly as the game restarts.

## B.5 Hyperparameters Used in the Simulation Study

Refer to our source code\(^2\) and the original papers (Hasselt et al., 2016; Wang et al., 2016) for detailed descriptions. Most of the hyperparameters are borrowed from the literature on closely related models (Oh et al., 2015; Wang et al., 2016) and can be assumed to be near optimal.

\(^2\)https://gitlab.com/BlazingBlade/Atari-ADNN
**ConvADNN**: As we are using the simplified architecture (Figure 2.8), $U'$ is placed at the bottom right corner of the first frame in $S'$.

Dimension Reduction Network: convolution 1: output feature maps: 64, kernel size: 6, stride: 2, padding size: 0, activation: ReLU, batch normalization: True; convolution 2: output feature maps: 64, kernel size: 5, stride: 1, padding size: 2, activation: ReLU, batch normalization: True; convolution 3: output feature maps: 1, kernel size: 5, stride: 1, padding size: 2, activation: ReLU, batch normalization: True.

Action Transformation Layer: A separate fully connected layer of size 1600 (which is the same as the number of features constructed by Dimension Reduction Network) for each possible action. The output preserves the format of the input ($\mathbb{R}^{40 \times 40 \times 1}$) for the follow-up transposed convolutions.

Prediction Network: 3 transposed convolutions mirroring the 3 convolutions used in Dimension Reduction Network. E.g., transposed convolution 1 mirrors convolution 3.

Training ConvADNN: random walk sample size: 3e10, initial learning rate: 0.1, learning rate decay patience: 2, decay factor: 2, early stop patience: 5, max epoch: 45, gradient clipping: 0.25, optimizer: Stochastic Gradient Descent, batch size: 30.

**Dueling Double Deep Q-Network** (DDDQN): The convolutional layers for the original input (no dimension reduction) and dimension reduced input differ slightly, as we find that smaller kernel and stride work better when the input is compressed.

For the original input: convolution 1: output feature maps: 32, kernel size: 8, stride: 4, padding size: 0, activation: None; convolution 2: output feature maps: 64, kernel size: 4, stride: 2, padding size: 0, activation: None; convolution 3: output feature maps: 64, kernel size: 3, stride: 1, padding size: 0, activation: None.

For the dimension reduced input: convolution 1: output feature maps: 32, kernel size: 5,
stride: 1, padding size: 0, activation: None; convolution 2: output feature maps: 64, kernel size: 4, stride: 1, padding size: 0, activation: None; convolution 3: output feature maps: 64, kernel size: 3, stride: 1, padding size: 0, activation: None.

Regardless of the input type, the convolution layers are followed by a fully connected layer of size 512, which branches into a state value network of size 1 and advantage network of size $|\mathcal{A}|$ (number of actions in the game).

Training DDDQN: random steps before training begins: 50,000, max random steps when a new episode starts: 30, training steps per epoch: 55,000, validation steps per epoch: 25,000, max number of epochs: 80, batch size: 32, validation epsilon (chance for random action selection): 0.001, starting training epsilon: 1.0, final training epsilon: 0.005, epsilon annealing steps: 1,000,000, experience replay size: 500,000, number of action repeat: 1, online network update frequency: 4, target network update frequency: 5,000, optimizer: ADAM, reward adjustment: None, learning rate: 0.00025, discount factor: 0.99.
C.1 Regularization Techniques for Neural Networks

The three main regularization techniques used in our model are Dropout (Srivastava et al., 2014), Residual connection (He et al., 2016), and Batch normalization (Ioffe and Szegedy, 2015). We briefly describe them below:

**Dropout**: An ensemble of models often has better generalization results compared to an individual model in the ensemble, as an individual model is more susceptible to overfitting (Srivastava et al., 2014) (e.g., a random forest often performs better than a single tree model). However, training an ensemble of neural networks is computationally infeasible.
Dropout allows a single neural network to perform like an ensemble of “thinned” sub-neural networks by masking each neuron with probability\(^1\) \(p\) during each training step, and re-weight the output of each neuron by \(p\) during testing time. More precisely, using notations from Section 2.1, during training Dropout replaces the typical between-layer computation

\[
O^l = \xi^l(w^l \cdot O^{l-1} + b^l)
\]

with

\[
O^l = \xi^l(w^l \cdot O^{l-1} \odot D^l + b^l), \quad \text{where} \quad D^l \sim \text{Bernoulli}(p).
\]

Dropout has become one of the most frequently used techniques to avoid overfitting due to its effectiveness (Goodfellow et al., 2016).

**Batch Normalization**: When training a neural network, the input distribution of each layer changes constantly as the parameters of earlier layers get updated. To reduce this internal distribution shift between layers, Batch Normalization adds a normalization procedure after each layer for each training mini-batch. Let \(\mathcal{B}\) be a mini-batch of input vectors for a given layer, \(X^{(k)}_i\) the value in dimension \(k\) of sample \(i\) in this batch, \(\mu^{(k)}_{\mathcal{B}}\) the mini-batch mean in dimension \(k\), and \(\sigma^{2(k)}_{\mathcal{B}}\) the mini-batch variance in dimension \(k\), then Batch Normalization makes the following replacement:

\[
X^{(k)}_i \leftarrow \gamma \frac{X^{(k)}_i - \mu^{(k)}_{\mathcal{B}}}{\sqrt{\sigma^{2(k)}_{\mathcal{B}} + \epsilon}} + \beta,
\]

which is simply a normalization over the mini-batch followed by re-scaling with learnable parameters \(\gamma\) and \(\beta\). A small \(\epsilon\) is added for numerical stability. It is common practice to

\(^1\)The pre-specified dropout probability could vary from layer to layer, but is typically set close to 1 for the input layer and close to 0.5 for all hidden layers (Srivastava et al., 2014)
normalize the input data for a neural network, and Batch Normalization essentially does it between layers. The resulting neural network is reported to be more robust to initial weights and learning rate, train much faster, and have better performance (Ioffe and Szegedy, 2015). Subsequent research (Laurent et al., 2015; Ba et al., 2016) has suggested that Batch Normalization is not suitable for the input-to-hidden transitions in a recurrent neural network, so in our implementation we apply Batch Normalization to every layer except for the recurrent layers.

**Residual Connection**: Residual Connection was first introduced in the 2015 ImageNet classification winner (He et al., 2016) and later adopted by Google’s machine translation system (Wu et al., 2016) among other applications. Consider a single layer in a neural network as an approximator of some desired underlying map \( \mathcal{H}(X) \). He et al. (2016) hypothesize that it is easier for the layer to learn the residual map \( \mathcal{F}(X) := \mathcal{H}(X) - X \) than to learn the original map \( \mathcal{H}(X) \) directly. As a simple example, if \( \mathcal{H}(X) \) is the identity map, the hypothetically easier \( \mathcal{F}(X) \) would be the zero map. Residual Connection adds the input of a layer to its output, allowing the layer to learn the residual map instead of the original map. Much evidence has shown that Residual Connection helps improve the model performance, especially for very deep neural networks (He et al., 2016).

The extent of improvement gained from these techniques may still depend on the model structure and task/dataset, and can be tuned on validation data.
C.2 Hyperparameters of Ordinal Regression Neural Network

The hyperparameters of our ordinal regression model are as follows (refer to our source code\(^2\) for detailed descriptions):

**Word Embeddings**: pretraining model type: Skip-gram; speedup method: negative sampling; number of negative samples: 100; noise distribution: unigram distribution raised to \(3/4\); batch size: 16; window size: 5; minimum word count: 5; number of epochs: 50; embedding size: 128; pretraining learning rate: 0.2; fine-tuning learning rate scale: 1.0.

**GF-RNN**: hidden size: 128; dropout: 0.2; number of layers: 3; gradient clipping norm: 0.25; L2 penalty: 0.00001; learning rate decay factor: 2.0; learning rate decay patience: 3; early stop patience: 9; batch size: 200; batch normalization: true; residual connection: true; output layer type: mean-pooling; minimum word count: 5; maximum input length: 120.

**Multi-labeled logistic regression layer**: task weight scheme: uniform; conflict penalty: 0.5.

C.3 Term Frequency-Inverse Document Frequency

How the term frequency-inverse document frequency (tf-idf) matrix is created (Beel et al., 2016; Manning et al., 2009) in our implementation:

Assume that the escort ads have been preprocessed as described in Section 3.4.1. They are further tokenized by unigrams. So in this case each unigram is a term, and each escort ad is a document. The term frequency (tf) for a term \(t\) in a document \(d\), denoted as \(tf(t, d)\),

\(^2https://gitlab.com/BlazingBlade/TrafficKill\)
is simply the number of occurrences of $t$ in $d$. Heuristically, the importance of a term does not necessarily scale linearly with its frequency, so sublinear tf, denoted as $tf_s(t, d)$, makes the following adjustment:

$$
    tf_s(t, d) = \begin{cases} 
    1 + \log tf(t, d) & \text{if } tf(t, d) > 0 \\
    0 & \text{otherwise}
\end{cases}
$$

If a term (such as “a”, “the”, etc.) appears very frequently in most of the documents, then it has little relevance in distinguishing documents. So the importance of term should be weighted by the inverse document frequency (idf) of this term. Let $df(t)$ be the number of documents in the dataset that contains the term $t$ and $N$ be the total number of documents in the dataset, then the idf of $t$ is given by

$$
    \text{idf}(t) = \log \frac{N}{df(t)}.
$$

Smooth idf, denoted as $idf_s(t)$, adds one to the df to avoid zero division:

$$
    \text{idf}_s(t) = \log \frac{N}{1 + df(t)}.
$$

Finally, the importance of a term $t$ in a document $d$ can be represented by the weighted tf:

$$
    \text{tf-idf}(t, d) = tf_s(t, d) \cdot \text{idf}_s(t).
$$

Given the tf-idf$(t, d)$ $\forall$ $t, d$, each document $d$ in the dataset can be converted to a numeric vector containing tf-idf$(t, d)$ $\forall$ $t$.

As a term that appears too infrequently or too frequently across documents will have little impact on the tf-idf matrix of the dataset, it is customary to ignore these terms all
together which can help reduce the dimension. In our implementation, any term that appears in less than 2 documents or in more than 95% of documents is ignored; furthermore, only the top 20,000 frequently appeared terms across the dataset are retained.

C.4 t-Distributed Neighbor Embedding

t-Distributed Neighbor Embedding (t-SNE) is good at preserving both the local and global structures when mapping data points from high dimension to low dimension; it demonstrates superior visualization performances over other state-of-the-art dimension reduction techniques on a variety of datasets (van der Maaten and Hinton, 2008). The main algorithm is summarized in Algorithm 5. Refer to the original paper (van der Maaten and Hinton, 2008) for the motivation behind each computation.

C.5 Additonal Emoji Maps

Figure 3.6 shows the top 200 most frequently appeared emojis from the escort ads with some outliers left out for better zoom. Here we show the emoji maps of the less frequent emojis, (approximately) 200 per map: Figure C.1, C.2, C.3, and C.4. Again we see many clusters of visually similar emojis, suggesting the high quality of learned embeddings. One particularly interesting cluster not shown in Section 3.4.4 is the cluster of transportation related emojis (near (2, -10) across the maps): airplane, bus, train, helicopter, bike, race car, etc. The airplane emoji is an indicator of movement in the trafficking lexicon, and here we see all its alternatives.
Algorithm 5 t-Distributed Neighbor Embedding

**Input:** data \( \{X_i\}_{i=1}^n \) where \( X_i \in \mathbb{R}^a \); perplexity \( \xi \); number of iterations \( T \), learning rate \( \eta \), momentum \( \alpha(t) \); desired low dimension \( b < a \).

1. Compute the pairwise affinities
   \[ p_{ji} = \frac{\exp \left\{-\|X_i - X_j\|^2/(2\sigma_i^2)\right\}}{\sum_{k \neq i} \exp \left\{-\|X_i - X_k\|^2/(2\sigma_i^2)\right\}} \]
   where \( \sigma_i \) is determined by satisfying
   \[ \xi = 2^{-\sum_i p_{ji} \log p_{ji}} . \]

2. Set \( p_{ij} = \frac{p_{ji} + p_{ij}}{2n} \).

3. Sample initial low-dimensional representations
   \[ \mathcal{Y}^{(0)} := \{Y_i\}_{i=1}^n \sim \text{Normal}(0, 10^{-4}I_b) . \]

4. for \( t = 1, \ldots, T \) do
   5. Compute low-dimensional affinities
      \[ q_{ij} = \frac{(1 + \|Y_i - Y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|Y_k - Y_l\|^2)^{-1}} . \]
   6. Compute the gradient \( \frac{\partial C}{\partial \mathcal{Y}} \) of the Kullback-Leibler divergence
      \[ C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \]
      by
      \[ \frac{\partial C}{\partial Y_i} = 4 \sum_j (p_{ij} - q_{ij})(Y_i - Y_j)(1 + \|Y_i - Y_j\|^2)^{-1} . \]
   7. Set \( \mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathcal{Y}} + \alpha(t) \left( \mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right) \)

**Output:** \( \mathcal{Y}^{(T)} \)
**Figure C.1** Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 201–400 are shown; outliers were removed for better zoom.
Figure C.2  Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 401–600 are shown; outliers were removed for better zoom.
Figure C.3  Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 601–800 are shown; outliers were removed for better zoom.
Figure C.4 Emoji map produced by applying t-SNE to the emojis’ vectors learned from escort ads using skip-gram model. For visual clarity, only the emojis with frequency ranking 801–968 are shown; outliers were removed for better zoom.