Université de Montréal

Coordination in Generative Modeling, Automatic Differentiation and Multi-Agent Learning

par

Tim M.E. Cooijmans

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

August 31, 2023

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Coordination in Generative Modeling, Automatic Differentiation and Multi-Agent Learning

présentée par

Tim M.E. Cooijmans

a été évaluée par un jury composé des personnes suivantes :

Ioannis Mitliagkas
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Pascal Vincent
(membre du jury)

Jakob Foerster
(examinateur externe)

Ioannis Mitliagkas
(représentant du doyen de la FESP)

Résumé

Cette thèse présente quatre articles dans trois domaines différents : la modélisation générative de la musique, l'attribution de crédits pour les réseaux de neurones récurrents (RNN) et l'apprentissage par renforcement multi-agents (MARL) dans des dilemmes sociaux. Un thème commun est la coordination, c'est-à-dire la prise en compte des dépendances entre des éléments modélisés comme indépendants.

Le premier article concerne la coordination des *états informationnels*. Nous développons une approche de raffinement itératif de la composition musicale. Notre modèle produit des prédictions conditionnellement indépendantes pour les variables dépendantes, qui sont réconciliées par un processus de raffinement itératif. Notre processus est plus proche de celui des compositeurs humains que des approches existantes qui produisent de la musique du début à la fin, et plus résistant aux entrées corrompues.

Dans le deuxième article, nous considérons la coordination des *communications* entre des nœuds des graphes. L'algorithme d'entraînement UORO pour les réseaux neuronaux récurrents en mode direct simplifie le graphe de calcul des dérivées en introduisant des fausses connexions, ce qui transforme les communications ciblées en simples diffusions. Les véritables connexions sont rétablies en espérance, grâce à l'utilisation de secrets partagés sous forme de bruit corrélé. Nous étudions la variance de cette approximation. Sur la base de nos connaissances, nous introduisons une variation pratique qui réduit considérablement la variance au prix d'une augmentation des ressources de calcul. Nous établissons également un lien théorique entre REINFORCE et UORO, avec égalité jusqu'à un terme dont l'espérance est zéro mais qui contribue à une variance arbitraire.

Dans les troisième et quatrième articles, nous coordonnons des processus. D'abord, nous proposons un algorithme pour trouver des politiques fortes dans des dilemmes sociaux en supposant un adversaire rationnel. Notre agent vise à trouver des politiques pour lesquelles la meilleure réponse est favorable, coordonnant ainsi de manière unilatérale les deux agents. Cette méthode est plus évolutive que les travaux antérieurs basés sur la même idée, et elle est performante sur un benchmark exigeant impliquant des politiques de réseaux neuronaux.

Ensuite, nous proposons de traiter l'apprentissage multi-agents comme un jeu et d'y appliquer value learning. Il en résulte une fonction de méta-valeur ; un jeu modifié dans lequel l'apprentissage naïf est coordonné. Nous démontrons une manière efficace d'apprendre la méta-valeur en *Q-learn*, sans représenter explicitement l'espace d'actions continues et sans utiliser d'estimateurs REINFORCE. La méthode trouve des politiques fortes dans des dilemmes sociaux et se compare favorablement à une méthode similaire basée sur une méthode *policy-gradient* dans la littérature.

Mots clés: reseaux neuronaux, modélisation générative, attribution des crédits, dérivation automatique, apprentissage par renforcement multi-agents, dilemmes sociaux

Abstract

This dissertation presents four articles in three different domains: generative modeling of music, credit assignment for recurrent neural networks (RNNs), and multi-agent reinforcement learning (MARL) on social dilemmas. A common theme is *coordination*, by which we mean accounting for dependencies between things that were modeled as independent.

The first article concerns coordination of *information states* of disjoint variables. We develop a model that produces conditionally independent predictions for dependent variables, which are reconciled through an iterative refinement process. Our process is closer to that of human composers than existing approaches which produce music from beginning to end, and more robust to corrupted inputs.

In the second article, we consider coordination of *communications* between graph nodes. The forward-mode RNN training algorithm UORO simplifies the derivative computation graph by introducing spurious connections, turning targeted communications into simple broadcasts. The true connections are restored in expectation through the use of shared secrets in the form of correlated noise. We study the variance of this approximation. Based on our insights we introduce a practical variation that drastically reduces variance at increased computational expense. We also establish a theoretical connection between REINFORCE and UORO, with equality up to a term that is zero in expectation but contributes arbitrary variance.

In the third and fourth articles, we coordinate *processes*. First, we propose an algorithm for finding strong policies in social dilemmas by assuming a rational opponent. Our agent aims to find policies for which the best response is favorable, thus unilaterally coordinating both agents. This method scales better than prior work based on the same idea, and is competitive on a tough benchmark involving neural network policies.

We then proceed to treat multi-agent learning as a game and apply value learning to it. This results in a meta-value function that measures how the return improves as learning progresses, which can be seen as a modified game in which naive learning is coordinated. We demonstrate an efficient way to Q-learn the meta-value, without explicitly representing the continuous action space of policy updates, and without the

use of REINFORCE estimators. The method finds strong policies in social dilemmas and compares favorably to a related policy gradient-based method from the literature.

Keywords: neural networks, generative modeling, credit assignment, automatic differentiation, multi-agent reinforcement learning, social dilemmas

Contents

Résumé 5

Abstract 7

List of tables 15

List of figures 17

List of acronyms and abbreviations 21

Acknowledgements 23

Chapter 1. Introduction 25

Chapter 2. Background 29

2.1. Supervised Learning 29

2.2. Unsupervised and Self-Supervised Learning 30

2.3. Generative Modeling 31

2.4. Reinforcement Learning 33

2.4.1. Temporal Difference Learning 35

2.4.2. Policy Gradient Methods 36

2.5. Multi-Agent Reinforcement Learning 37

2.6. Neural Networks 40

2.7. Optimization 45

Chapter 3. Counterpoint by Convolution 49

Prologue 49

3.1. Introduction 50

3.2. Related Work 53

3.3. Model . . . 54
3.4. Evaluation . . . 57
3.5. Sampling . . . 58
3.5.1. Orderless NADE Sampling . . . 58
3.5.2. Gibbs Sampling . . . 59
3.6. Experiments . . . 60
3.6.1. Data Log-likelihood . . . 60
3.6.2. Sample Quality . . . 60
3.6.3. Human Evaluations . . . 62
3.7. Conclusion . . . 63
Acknowledgments . . . 63
References . . . 63
Chapter 4. On the Variance of Unbiased Online Recurrent Optimization . . . 67
Prologue . . . 67
4.1. Introduction . . . 68
4.2. Outline of the Paper . . . 69
4.3. Automatic Differentiation in Recurrent Neural Networks . . . 69
4.4. Other Approaches to Credit Assignment . . . 71
4.5. Unbiased Online Recurrent Optimization . . . 72
4.5.1. Derivation . . . 72
4.5.2. Greedy Iterative Rescaling . . . 74
4.6. Variance Analysis . . . 75
4.6.1. Greedy Iterative Rescaling is Greedy . . . 75
4.6.2. Greedy Iterative Rescaling Optimizes an Inappropriate Objective . . . 77
4.6.3. Generalized Recursions . . . 77
4.6.4. A Simple Expression for the Gradient Estimate . . . 78
4.6.5. Computing the Variance of the Total Gradient Estimate . . . 79
4.7. Variance Reduction . . . 80
4.7.1. Optimizing Q subject to restrictions on its form . . . 80

4.7.2. Variance Reduction Experiments 83
4.8. Projection in the Space of Preactivations 86
4.9. REINFORCE as Approximate Real-Time Recurrent Learning 89
4.10. Conclusions 92
Acknowledgements 92
References 93
4.A. Supporting Results for Variance Computations 95
4.B. Variance of a Single Jacobian Estimate 96
4.C. Optimizing α given Q_0 97
4.D. Online optimization of α coefficients 99
4.E. Minimization of the Product of Traces 100
4.F. Estimating B online 103
4.G. Hyperparameter Settings for Variance Reduction Experiments 105
4.H. Variance of Preactivation-Space Projection 105
Chapter 5. Best Response Shaping 107
Prologue 107
5.1. Introduction 108
5.2. Background 110
5.2.1. Multi Agent Reinforcement Learning 110
5.2.2. Social Dilemmas and the Iterated Prisoner's Dilemma 110
5.3. Related Work 111
5.4. Best Response Shaping 112
5.4.1. Best Response Agent to the Best Response Opponent 113
5.4.2. Detective Opponent Training 114
5.4.3. Agent training 115
5.5. Experiments 116
5.5.1. Iterated Prisoner's Dilemma 116
5.5.2. The Coin Game 118

5.6. Limitations . . . 119
5.7. Conclusion . . . 120
References . . . 121
5.A. Experimental Details . . . 122
5.A.1. IPD . . . 122
5.A.2. Coin Game . . . 122
5.B. Reproducing Results . . . 124
5.B.1. IPD . . . 124
5.B.2. Coin Game . . . 125
5.C. League Results . . . 125
5.D. Self-Play . . . 125
5.E. Tree Search Detective . . . 129
5.F. Evaluation Metrics of Various Agents . . . 129
Chapter 6. Meta-Value Learning: a General Framework for Learning with Learning Awareness . . . 135
Prologue . . . 135
6.1. Introduction . . . 136
6.2. Background . . . 138
6.2.1. Naive Learning . . . 138
6.2.2. Looking Ahead . . . 138
6.2.3. Going Meta . . . 139
6.3. Meta-Value Learning . . . 140
6.3.1. The Meta-Value Function . . . 140
6.3.2. Learning Meta-Values . . . 141
6.3.3. Q-learning interpretation . . . 141
6.4. Practical Considerations . . . 142
6.4.1. Reformulation as a Correction . . . 143
6.4.2. Variable Discount Rates . . . 143
6.4.3. Exploration . . . 143

6.5. Experiments . . . 144
6.5.1. Logistic Game . . . 144
6.5.2. Matrix Games . . . 145
6.5.3. Ablation . . . 148
6.6. Limitations . . . 148
6.7. Conclusion . . . 148
Acknowledgements . . . 150
References . . . 151
6.A. Normalized Bellman Equation . . . 154
6.B. LOLA, HOLA, COLA . . . 154
6.C. Detailed Algorithm Description . . . 155
6.D. Logistic Game Details . . . 155
6.E. Matrix Game Details . . . 157
6.F. ZD-extortion on the IPD . . . 159
Conclusion . . . 161

List of tables

2.1 Iterated Prisoner's Dilemma payoffs 38

3.1 Framewise negative log-likelihoods (NLLs) on the Bach corpus. We compare against (Boulanger-Lewandowski et al., 2012), who used quarter-note resolution. We also compare on higher temporal resolutions (eighth notes, sixteenth notes), against our own reimplementation of RNN-NADE. COCONET is an instance of orderless NADE, and as such we evaluate it on random orderings. However, the baselines support only chronological frame ordering, and hence we evaluate our model in this setting as well. 57

3.2 Mean (± SEM) NLL under model of unconditioned samples generated from model by various schemes. 61

5.1 Payoff matrix for the prisoner's dilemma game 122

5.2 Hyperparameter search options 124

6.1 Payoffs for the matrix games considered. 146

6.2 Head-to-head comparison of learning rules on Iterated Prisoner's Dilemma. MeVa extorts the naive learner, and appears to exploit LOLA to a small extent. 147

6.4 Head-to-head comparison of learning rules on Iterated Prisoner's Dilemma. . . 158

6.5 Head-to-head comparison of learning rules on Iterated Matching Pennies. 158

6.6 Head-to-head comparison of learning rules on the Chicken Game. 158

List of figures

3.1 Blocked Gibbs inpainting of a corrupted Bach chorale by COCONET. At each step, a random subset of notes is removed, and the model is asked to infer their values. New values are sampled from the probability distribution put out by the model, and the process is repeated. Left: annealed masks show resampled variables. Colors distinguish the four voices. Middle: grayscale heatmaps show predictions $p(\mathbf{x}_j \mid \mathbf{x}_C)$ summed across instruments. Right: complete pianorolls after resampling the masked variables. Bottom: a sample from NADE (left) and the original Bach chorale fragment (right). 51

3.2 Likelihood under the model for ancestral Gibbs samples obtained with various context distributions $p(C)$. NADE (Bernoulli(0.00)) is included for reference. . 61

3.3 Human evaluations from MTurk on how many times a sampling procedure or Bach is perceived as more Bach-like. Error bars show the standard deviation of a binomial distribution fitted to each's binary win/loss counts. 62

4.1 Training curves on the row-wise sequential MNIST task. For each setting we have run 10 trials and plotted the mean of the classification loss and the 95% confidence interval of the mean. For clarity of presentation, these curves have been aggressively smoothed by a median filter prior to the computation of their statistics. 84

4.2 Theoretical predictions and empirical measurements of quantities contributing to total gradient variance. The "intrinsic" variance measures the expected norm of the total gradient $\mathcal{J}_\theta^L$, estimated by averaging across the minibatch. The "expected" variance is a theoretical prediction of $V(Q)$ according to Equation 4.7.6. The "actual" variance measures $V(Q)$ empirically by the expected norm of the total gradient estimate. 86

4.3 Evolution of $\log \alpha_s$ for some time steps s as training proceeds. At each training step, the $\log \alpha_s$ are centered so that $\min_s \log \alpha_s = 0$; this eliminates irrelevant constant factors. 87

4.4 Training curves on the queue task showing interpolation between RTRL and UORO by ablation of the spatial and temporal approximations. "neither" denotes exact computation of the gradient using RTRL, "spatial" denotes RTRL with $\mathcal{J}^{h_t}_{z_t} \nu_t \nu_t^\top \mathcal{J}^{z_t}_{\theta_t}$ standing in for $\mathcal{J}^{h_t}_{\theta_t}$, "temporal" denotes PREUORO computed by Equation 4.8.1, "both" denotes UORO . Where applicable, the cut vertex $z_t \equiv W_t a_t$ is the preactivations. 88

5.1 The detective is trained using agents sampled from a replay buffer, which contains agents encountered during training. Additional noise is incorporated to broaden the range of agents encountered by the detective. The detective's training focuses on conditioning based on the agent's policy, aiming to estimate the optimal response to each policy to maximize its own return. Following this, the agent's training is conducted through backpropagation, through the detective's conditioning mechanism. 113

5.2 Illustration of the policies of agents trained with BRS and BRS-SP in a finite Iterated Prisoner's Dilemma game of length 6. The agents are trained against a tree search detective maximizing its own return. BRS-SP agents learn tit-for-tat, a policy that cooperates initially and mirrors the opponent's behavior thereafter. BRS agents learn cynic-tit-for-tat (CTFT), they defect initially but mirror the opponent's behavior thereafter. Therefore, at the initial state, BRS agents exploit the rationality of the opponent. 117

5.3 Comparison of Agent's trained with BRS, BRS-SP, and POLA on a 3×3 sized Coin Game. We evaluate the agent's returns versus different opponents: Always Defect opponent (AD) that takes the shortest path to all coins; Always Cooperate opponent (AC) that takes the shortest path to its own coin but does not take the agent's coin; A Monte Carlo Tree Search opponent (MCTS) that do one thousand rollouts using the agent's policy at each state; a trained opponent (Trained) that is trained vs the agent to maximize its own return; and agent's performance against itself (Self). The POLA agents are exploited by the MCTS, which approximates the best response. It means, a rational opponent maximizing its return, harms POLA's return unintentionally. In contrast, BRS and BRS-SP get a higher return against the MCTS. Although BRS agents are not inherently cooperative among themselves, they effectively exploit the AC opponent in a rational manner. On the other hand, BRS-SP agents exhibit greater levels of cooperation among themselves but refrain from exploiting the AC opponent. 119

5.4 Evaluating BRS, BRS-SP, and POLA agents against each other. The red dot shows the average return of each agent against the average return of the other agent. Left: Both agents get a positive return on average. However, POLA gets a higher return than BRS-SP agent on average, indicating exploitation. Middle: POLA agents and BRS agents do not cooperate. BRS is able to get a higher return than POLA on average indicating that POLA is exploited. Right: BRS agents get a higher return than the BRS-SP agents. Effectively BRS agent is exploiting the BRS-SP agent. 120

5.5 The presented figure illustrates the outcomes of 1-vs-1 Coin games lasting 50 rounds, involving a range of agents. The return achieved by each agent is documented within the corresponding cell. The reported returns are an average across 32 independent games. The numerical suffixes following the agent names signify agents trained using distinct initialization seeds. It is important to note that there are no games recorded between the MTCS agent and the trained agent due to a lack of clarity regarding its meaning. 126

5.6 This figure illustrates the training of the IPD agent against the TSD. TSD samples from the agent's policy, represented by red arrows in the plot, while exploring all possible actions when considering its own actions, represented by black arrows in the plot. The agent treats the TSD as a black-box algorithm and differentiates through it via REINFORCE. Note that the summation is over all log probabilities and not only over the log probabilities presnet in the path. 130

5.7 This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against the evaluation metrics. 131

5.8 This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against each other. 132

6.1 The Logistic Game. The left panel displays the contours of player 1's objective $f_1(x)$, the right panel similarly for player 2. Player 1's policy x_1 is a horizontal position, player 2's policy x_2 is a vertical position. Both players prefer solution B over solution A, but cannot unilaterally go there. Naive learning converges to whichever solution is closest upon initialization. 145

6.2 Logistic Game behaviors of different algorithms (rows) with different settings (columns). 146

6.3 Ablation experiment on the Iterated Prisoner's Dilemma. We show the effect of using the V formulation over the U formulation, and disabling exploration, target networks, λ-returns, and distributional RL. For each configuration we train 3 models for 300 outer loops. We show short-term TD error (over k steps, as in training) and long-term TD error (over 100 steps, as a validation); the difference between these is due to bootstrapping. The horizontal axis measures number of outer loops performed. We also show the returns $f(x)$ of agents that are being trained on the model (with $\gamma = 0.95$), and are reset every 10 outer loops. 149

6.4 Meta-value agents extorting naive agents on the Iterated Prisoner's Dilemma. We train $\hat{U}$ from four different initializations (columns). Then, we initialize a pair of policies and show their behavior as they learn (rows). Each panel shows the polytope of possible game returns (gray outline), the subset of possible game returns given the current MeVa policy x_1 (blue scatterplot of return pairs obtained by pitting random policies against x_1), and the actual return pair given x_1 and x_2 (orange, with lines to help tell the angle of the rightmost front of the blue region). 160

List of acronyms and abbreviations

A2C	Advantage Actor-Critic
AC	Always Cooperate
AD	Always Defect
AI	Artificial Intelligence
BN	Batch Normalization
BPTT	Back-Propagation Through Time
BRS	Best Response Shaping
BRS-SP	Best Response Shaping with Self-Play
CNN	Convolutional Neural Network
COLA	Consistent LOLA
CTFT	Cynic Tit-For-Tat
DBN	Deep Belief Network
GAE	Generalized Advantage Estimation
GELU	Gaussian Error Linear Unit
GIR	Greedy Iterative Rescaling
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GSN	Generative Stochastic Network
HMM	Hidden Markov Model
HOLA	Higher-Order LOLA
iid	Independently Identically Distributed
IPD	Iterated Prisoner's Dilemma
LOLA	Learning with Opponent Learning Awareness
LSTM	Long Short-Term Memory
MARL	Multi-Agent Reinforcement Learning
MCMC	Markov Chain Monte-Carlo
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process

MeVa	Meta-Value Learning
M-FOS	Model-Free Opponent Shaping
MH	Metropolis-Hastings
MIDI	Musical Instrument Digital Interface
MLP	Multi-Layer Perceptron
MTurk	Amazon's Mechanical Turk
NADE	Neural Autoregressive Distribution Estimator
NLL	Negative Log-Likelihood
NN	Neural Network
POLA	Proximal LOLA
PSD	Positive Semidefinite
PVN	Policy Evaluation Networks
QA	Question Answering
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTRL	Real-Time Recurrent Learning
SEM	Standard Error of the Mean
SFP	Stable Fixed Point
SGD	Stochastic Gradient Descent
TD	Temporal Difference
TFT	Tit-For-Tat
TSD	Tree Search Detective
UORO	Unbiased Online Recurrent Optimization

Acknowledgements

I am grateful for the support I've received from loved ones; my partner in crime Cheng-Zhi Anna Huang, my parents Rien and Wilma, my brother Geert and his family, my local cousin Krystle, and other family and friends back home that I don't see often enough.

I want to thank my advisor, Aaron Courville, for providing a great environment for research, and for being invested in and engaged with my projects. I also very much enjoyed my brief collaboration with James Martens, whose deep, practical understanding of so many topics in machine learning inspired me to be bold in my own explorations. I'm grateful to numerous fellow students in the LISA, MILA and Mila labs as well as the staff running them over the years, as well as the administrative staff at UdeM for putting up with us.

I'm happy to have worked with many and varied coauthors; among them Milad Aghajohari, Faruk Ahmed, Shunichi Akatsuka, Amjad Almahairi, Akilesh Badrinaaraayanan, Nicolas Ballas, Pablo Samuel Castro, Yi Deng, Monica Dinculescu, Chris Donahue, Juan Augustin Duque, Douglas Eck, Jesse Engel, Çağlar Gülçehre, Vishal Gupta, Curtis Hawthorne, Natasha Jaques, David Krueger, Rithesh Kumar, Anirban Laha, Hugo Larochelle, César Laurent, Tegan Maharaj, Ethan Manilow, Amartya Mitra, Michael Noukhovitch, Shayegan Omidshafiei, Adam Roberts, Alexander Scarlatos, Ian Simon, Rigel Swavely, Cassie Tarakajian, Christos Tsirigotis, Yusong Wu, Tianyu Zhang, and Yin Zheng.

Lastly, I want to acknowledge how heavily our academic research efforts depend on the continued cooperation among huge numbers of people. These massive investments in research of unclear practical gain are possible only in a state of luxury, where our basic needs are met and our quibbles are comparatively minor. If we can stay away from major wars, figure out a sustainable source of energy, keep the natural environment in shape, and mitigate inequality just a touch, we can go a long way.

Chapter 1

Introduction

A common theme in computer science is the decomposition of large problems into smaller, independent subproblems. This allows computation to be parallelized, distributed without communication, or sometimes massively simplified by use of linearity. We can often still gainfully do this when the subsystems are not independent, and restore or simulate the effects of important dependencies by other means.

Coordination (Malone and Crowston, 1994) refers to the process of inducing a dependency structure in (apparently) independent systems. Examples of this dynamic can be found in biomechanics (Turvey, 1990), classical counterpoint (Fux, 1965) and jazz improvisation (Bastien and Hostager, 1988), swarm intelligence (Kennedy, 2006), encryption (Merkle, 1978) and quantum entanglement (Einstein et al., 1935). We will say that two systems are coordinated when their states are in correspondence without a direct causal link.

In this work we will focus on specific instances of coordination in three areas. We coordinate *information states* of disjoint variables in the domain of music generation, we coordinate *communications* to distill broadcasts into directed messages in the context of credit assignment, and we coordinate *processes* to attain cooperation in social dilemmas.

In generative modeling of music, we develop a model that composes music by repeatedly rewriting it, rather than the prevailing method of writing it in one pass from beginning to end. Existing autoregressive models predict variables one-by-one and never revisit them; mistakes made along the way tend to compound and lead this process off the rails. We instead predict many variables simultaneously (conditionally-independently), and use iterative refinement to coordinate the variables, smoothing out mistakes and inconsistencies. Our process is closer to that of human composers, and explicitly robust to corruptions in its input.

In our work on credit assignment for RNNs we study the approximate forward-mode differentiation algorithm UORO (Tallec and Ollivier, 2018). Forward-mode differentiation computes gradients forward in time, which is more appropriate for streaming settings

(such as language modeling) than the usual reverse-mode computation. In the context of RNNs, the forward mode is known as Real-Time Recurrent Learning (RTRL).The main drawback of RTRL is its space cost: it stores a parameter differential for each unit in the current hidden state. UORO approximates RTRL by introducing spurious connections into the computation graph, which allows simplifications that drastically reduce the space requirement. The spurious connections derive from noise that is specifically correlated (*coordinated*) to restore the original graph in expectation. In essence, UORO replaces targeted communications between connected nodes by simple broadcasts to all nodes, using shared secrets to mitigate cross-communication. While UORO is much more space-efficient than RTRL, this comes at the cost of introducing variance. We develop a deep theoretical analysis of the sources of this variance and how they may be reduced. Based on these insights we introduce a practical variation, PreUORO, which drastically reduces variance at increased computational expense. Moreover, we establish a deep theoretical connection between REINFORCE and UORO.

Our work in MARL revolves around social dilemmas. These are games in which selfish agents tend to converge to lose-lose outcomes, even though a win-win outcome is available. The win-win outcome is unstable and requires coordination and threats to arrive at and maintain. A real-world example is environmental pollution: we all derive value from living in a clean environment, but no one wishes to shoulder the burden of undoing or even just preventing pollution. In game theory, these dynamics are captured in simple repeated matrix games, such as the well-known Iterated Prisoner's Dilemma (IPD; Axelrod and Hamilton, 1981). Done naively, multi-agent learning on such problems leads to lose-lose scenarios. Foerster et al. (2018) made a breakthrough with LOLA, which considers the effects of the policy update and actively attempts to influence the opponent's learning process. LOLA was the first method to systematically find the famous tit-for-tat strategy in the IPD.

We extend the ideas of LOLA in two directions. With Best Response Shaping (Chapter 5), we look infinitely far ahead to consider how a rational opponent would respond to us, and seek a policy for which that response is favorable. The practical implementation of this process takes the form of a conditional opponent policy that observes our current policy and is trained to play the best response to it. Prior work that has explored the same idea uses an explicit optimization process to find the best response, which is expensive to compute and differentiate through (Zhang et al., 2020; Balaguer et al., 2022). Our use of a trained conditional policy amortizes the cost of this optimization. Best Response Shaping scales to policies parameterized by neural networks and is competitive on a tough benchmark that requires them.

Our other work, Meta-Value Learning (Chapter 6), introduces the meta-value function: it scores policies by how well they do on the game, now and after future optimization.

We propose to follow naive gradients of the meta-value function rather than those of the game itself, and rather than those of LOLA's surrogate, which is short-sighted and inconsistent. The meta-value function follows from a fairly straightforward interpretation of joint learning as a meta-game, featuring policy pairs as meta-states, policy changes as meta-actions, and policy returns as meta-rewards (Al-Shedivat et al., 2017; Kim et al., 2021). When used as a surrogate for the original game, it can be seen as a mollified game where naive learning is coordinated. We apply Q-learning on this meta-game to approximate the meta-value function, in a way that sidesteps the complications of the continuous action space, and avoids the need for REINFORCE estimators. We visualize the method's behavior on a toy game, and quantify its performance on repeated matrix games.

Chapter 2

Background

We begin by discussing some basic machine learning topics that are foundational to the articles in the dissertation. For more details we recommend the books *Pattern Recognition and Machine Learning* by Bishop and Nasrabadi (2006), *Deep Learning* by Goodfellow et al. (2016) and *Reinforcement Learning: an Introduction* by Sutton and Barto (2018).

2.1. Supervised Learning

In supervised learning, we generally wish to predict some information y given some other information x. The classic example is image classification, where x are the pixels of an image and the label y is one of a finite number of categories, e.g. cat or dog. We draw a finite set of empirical examples $\{(x^{(i)}, y^{(i)})\}$ iid from the joint distribution $p(x, y)$ of images and their labels, and use these examples to "learn" a model $q_\theta(y \mid x) \approx p(y \mid x)$ that approximates the conditional distribution of y given x.

We will restrict ourselves to parametric functions q_θ where the parameter vector $\theta \in \mathbb{R}^d$ spans a range of possible models. For now, let us assume a log-linear model class:

$$q_\theta(y \mid x) \propto \exp(Wx + b)_y$$

with parameters $\theta = \begin{pmatrix} \operatorname{vec}(W)^\top & b^\top \end{pmatrix}^\top$. We will later discuss neural networks (Section 2.6), a very powerful model class that is nevertheless easy to work with.

The model $\theta^\star$ that best fits the data distribution is defined to be the one that minimizes some prediction error, say the KL divergence:

$$\begin{aligned} \theta^\star &= \operatorname{argmin}_\theta \mathbb{E}_{x\sim p(x)} D_{\mathrm{KL}}(p(\,\cdot \mid x) \| \, q_\theta(\,\cdot \mid x)) \\ &= \operatorname{argmin}_\theta -\mathbb{E}_{x\sim p(x)} \mathbb{E}_{y\sim p(y|x)} \log \frac{q_\theta(y \mid x)}{p(y \mid x)} \\ &= \operatorname{argmax}_\theta \mathbb{E}_{x,y\sim p(x,y)} \log q_\theta(y \mid x). \end{aligned}$$

In practice we use a finite training set to estimate the expectation, and we will typically use gradient descent to find the minimizer $\theta^\star$, repeatedly updating a candidate θ according to

$$\theta \leftarrow \theta + \eta \nabla \mathbb{E}_{x,y \sim p(x,y)} \log q_\theta(y \mid x),$$

until convergence. We will discuss this algorithm in more detail later (Section 2.7).

This approach is known as maximum likelihood estimation (Bishop and Nasrabadi, 2006). According to the classical understanding of machine learning, the model will suffer from overfitting and underfitting. Overfitting stems from the restriction to a finite sample from $p(x, y)$; a given sample may have exaggerated or even entirely accidental correlations between x and y, so that the model θ that maximizes the likelihood of our sample would not maximize the likelihood of another sample from the same distribution. Underfitting stems from the restriction to a model class q_θ that does not contain the true function $p(x \mid y)$, and whose closest fitting candidate is not particularly close.

We typically monitor overfitting and underfitting by measuring the error on the *training* data as well as a held-out *validation* or *test* set. The performance on the test set indicates how well the model will do on unseen data from $p(x, y)$. A model is said to overfit to the extent that the test error exceeds the training error. A model is said to underfit to the extent that it fails to reduce even the training error.

It is worth noting that before neural networks took off in the 2010s, the conventional wisdom was that there is a fundamental trade-off between overfitting and underfitting. Reducing model capacity (expressiveness of the model class) to reduce overfitting at the cost of underfitting, and for a long time this was the main regularization tool in the box. Neural networks have significantly changed this story, thanks in part to stochastic gradient descent, which has several implicit regularization effects (Wilson and Martinez, 2003; Keskar et al., 2016; Zhang et al., 2021; Arpit et al., 2017). In fact, it is well established that neural networks work best when made large (Krizhevsky et al., 2012; Belkin et al., 2019; Nakkiran et al., 2021).

2.2. Unsupervised and Self-Supervised Learning

Sometimes supervised learning is only a means to an end. Training an image classifier, for example, can be just a way to force a model to understand images in general, an understanding that we can later harness for other purposes (Simonyan and Zisserman, 2014; Szegedy et al., 2015; Bommasani et al., 2021). However, training an image classifier requires paired data $(x, y) \sim p(x, y)$, which can be costly to obtain. However, we can learn to understand images $x \sim p(x)$ without labels y.

For example, autoencoders (Goodfellow et al., 2016) are latent-variable models that consist of an encoder f_θ and a decoder g_θ, and are trained to minimize

$$\mathbb{E}_{x\sim p(x)}\|g_\theta(f_\theta(x)) - x\|^2.$$

This error can be seen as the negative log-likelihood under a Gaussian model $\mathcal{N}(g_\theta(f_\theta(x)); I)$. The purpose of an autoencoder is to learn useful latent codes $z = f_\theta(x)$ that describe x in terms of abstract features (entities, relationships, events) rather than surface-level properties (pixels). Regularization is required to avoid learning a trivial identity $g_\theta \circ f_\theta = \text{Id}$ and instead force the model to understand the data and extract useful features. We can do this for example by restricting the dimensionality of the latent code z, encouraging piecewise-constant codes (Rifai et al., 2011), penalizing the latent codes to fit some prior distribution, or adding noise to the codes (Kingma and Welling, 2013).

We highlight the *denoising* autoencoder (Vincent et al., 2008), which is trained to reverse a chosen corruption process $p(\tilde{x} \mid x)$. Each time we train on an example x, we produce a corrupted version $\tilde{x} \sim p(\tilde{x} \mid x)$, and ask the model to predict x given $\tilde{x}$. We minimize

$$\mathbb{E}_{x\sim p(x)}\mathbb{E}_{\tilde{x}\sim p(\tilde{x}|x)}\|g_\theta(f_\theta(\tilde{x})) - x\|^2.$$

Common choices are elementwise iid additive or multiplicative noise. The corruption destroys information, and the model must infer that information from what remains. The only way to do this is to understand the correlations in the data: reconstructing half a picture of a cat requires understanding cats. The idea of reconstructing corrupted idea is central to our work *Counterpoint by Convolution* (Chapter 3). It also underlies the current wave of diffusion models, which can be seen as denoising autoencoders that can operate at multiple severities of corruption (Dieleman, 2022).

2.3. Generative Modeling

In generative modeling we are interested in modeling the data distribution $p(x)$, with a particular emphasis on convenience of drawing samples x. We will assume $x \in \{1 \dots k\}^n$ is a vector of n categorical variables (e.g. text tokens, or discretized pixel values).

In this dissertation we will focus on autoregressive models, which model individual elements of x given information about other elements. When the n elements of x are independent, the model $q(x)$ can be factored like $p(x)$:

$$q(x) = q_1(x_1)q_2(x_2)\cdots q_n(x_n) \approx p_1(x_1)p_2(x_2)\cdots p_n(x_n) = p(x).$$

The problem becomes more interesting when the elements of x correlate, in which case we need to model the joint distribution $p(x)$. Different situations call for different factorizations, but the most common one is to treat x as a sequence and factor the model so that we

can generate from beginning to end:

$$q(x) = q_1(x_1)q_2(x_2 \mid x_1)q_3(x_3 \mid x_1, x_2) \cdots q_n(x_n \mid x_1 \cdots x_{n-1}).$$

The benefit of factoring the model is that we break up the problem into easier subproblems. Instead of having to predict $x_1, x_2 \cdots x_n$ as a whole, we predict the variables one at a time and get to condition on true values at training time. We minimize the expected NLL

$$\mathbb{E}_{x\sim p(x)}[-\log q(x)] = -\mathbb{E}_{x\sim p(x)} \log \prod_{i=1}^{n} q_i(x_i \mid x_{<i}) = -\mathbb{E}_{x\sim p(x)} \sum_{i=1}^{n} \log q_i(x_i \mid x_{<i}),$$

which in the simplest case boils down to training n models on n independent prediction problems. Generally it makes sense to share parameters between these models (as in NADE; Larochelle and Murray, 2011).

The extreme case of parameter sharing is that of recurrent models, where the variable-length sequence $x_{<i}$ is recursively aggregated into a fixed-length latent vector h_i:

$$h_i = f_\theta(h_{i-1}, x_i) \tag{2.3.1}$$

$$q_i(x_{i+1} \mid x_{<i}) \triangleq g_\theta(x_{i+1} \mid h_i). \tag{2.3.2}$$

Now the functions f_θ, g_θ do not depend on the subscript i, so they can be applied repeatedly with the same parameters. Jointly, they form a language model.

The generative procedure for autoregressive models is to predict variables x_i one at a time, each time conditioning on the variables $x_{<i}$ already predicted. A major problem with this approach is compounding of errors: a mistaken prediction or unlucky random draw for x_i will deteriorate the predictions $x_{>i}$ that follow it (Venkatraman et al., 2015). The process quickly goes off the rails as the observed variables $x_{<i}$ become less and less like the data on which the model was trained (Bengio et al., 2015).

In Chapter 3, we discuss a modeling approach in which the model observes the values of an arbitrary subset $x_C = \{x_j \mid j \in C\}$ of variables, and makes conditionally independent predictions for the unobserved variables $x_{\neg C}$:

$$\tilde{q}_{\neg C}(x_{\neg C} \mid x_C) \triangleq \prod_{i \notin C} q_i(x_i \mid x_C).$$

This structure is cheap to train, but it forms a poor model of the true distribution $p(x_{\neg C} \mid x_C)$. However, we can reinterpret these conditionally independent predictions: each of the conditionals $\prod_{i\notin C} q_i(x_i \mid x_C)$ forms part of autoregressive factorizations with variables reordered such that the observed variables x_C come before x_i and x_i comes before the other unobserved variables $x_{\neg C\{i\}}$. In fact, this model provides an autoregressive factorization along any ordering o:

$$q(x; o) = \prod_i q_{o_i}(x_{o_i} \mid x_{o_{<i}}).$$

One way to draw coordinated samples then, is to randomly choose an ordering o, and sample autoregressively along the ordering. However, this process still suffers from compounding errors.

Gibbs sampling (Geman and Geman, 1984) provides an alternative approach, that coordinates variables by gradually reconciling them with one another. Formally, we start with some initial value $x^{(0)}$ and then repeatedly resample variables using full conditionals:

$$x_i^{(t+1)} \sim p_i(x_i \mid x_{\neg\{i\}}^{(t)}).$$

It can be shown that this is a special case of Metropolis-Hastings (Metropolis et al., 1953; Hastings, 1970) with acceptance rate 1, and thus eventually converges to producing samples from the joint distribution $p(x)$.

The drawback is that the process tends to *mix* slowly: consecutive samples are highly correlated, so samples that are intended to be iid must be taken many time steps apart. This can be alleviated through block-Gibbs sampling (Liu, 1994), which resamples several variables at once according to their conditional joint distribution. In our work we applied block-Gibbs sampling with a conditional independence assumption within the block. This lets us use the cheap conditionally independent predictions $\tilde{q}_{\neg C}(x_{\neg C} \mid x_C)$. By annealing the block size down to one over time, we mitigate the downside of the assumption without losing the computational and statistical gains of block-Gibbs sampling. Moreover, the process is inherently robust to errors, as the repeated resampling allows them to be corrected over time.

2.4. Reinforcement Learning

So far we have considered problems in which we learn to predict targets on a fixed data distribution. In reinforcement learning (RL; Sutton and Barto, 2018), we consider an agent interacting with an environment, emitting actions and observing rewards and changes in the environment. The rewards provide a learning signal, but the agent does not a priori know which actions it took earned it the reward. Sometimes rewards are misleading, for example in investment, where an agent must forego reward in the short term in order to obtain much higher reward in the long term. A central theme in RL is *credit assignment*: taking all the rewards obtained and linking them to the actions that were causally responsible.

For the sake of demonstration, we will consider a fully-observable environment. Using the notation of Agarwal et al. (2021), this is a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \mu)$ where $\mathcal{S}$ is the state space of the environment, $\mathcal{A}$ is the action space, $\mu \in \Delta(\mathcal{S})$ and $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ are respectively the initial and transition distributions, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is a discount rate. An agent acting in

an MDP gives rise to a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$. We will denote prefixes by $\tau_t = (s_0, a_0, r_0, \ldots, s_t, a_t, r_t)$, and suffixes by $\tau^t = (s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \ldots)$. The set of all prefixes τ_t (finite trajectories) will be denoted $\mathcal{H}$. A policy $\pi : \mathcal{H} \to \Delta(\mathcal{A})$ observes a finite trajectory and emits a mixed action.

Given an initial state $s_0 \sim \mu$, we obtain the following process:

$$\begin{aligned} a_t &\sim \pi(\tau_t) \\ s_{t+1} &\sim P(s_{t+1} \mid s_t, a_t), \end{aligned}$$

which induces a probability distribution over trajectories

$$\Pr_\mu^\pi(\tau) = \mu(s_0)\pi(a_0 \mid \tau_0)P(s_1 \mid s_0, a_0)\pi(a_1 \mid \tau_1)\cdots$$

An agent generally seeks to adopt a policy π that maximizes the expected return

$$\mathbb{E}_{\tau \sim \Pr_\mu^\pi} R(\tau) \equiv \mathbb{E}_{\tau \sim \Pr_\mu^\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t),$$

and the discount rate γ controls the extent to which the agent values future reward.

If we view the expected return as a function of the initial state distribution μ, we obtain the state-value function:

$$V^\pi(\mu) = \mathbb{E}_{\tau \sim \Pr_\mu^\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t). \tag{2.4.1}$$

Thanks to the Markov property, the value of a given state is independent of the trajectory through which it was encountered. When applied to a pure state (i.e. a deterministic state distribution) encountered on a trajectory, the value function gives the expected return for the rest of the trajectory. It satisfies the Bellman equation, around which reinforcement learning revolves:

$$V^\pi(s) = \mathbb{E}_{a|s\sim\pi}\mathbb{E}_{s'|s,a\sim P}[r(s,a) + \gamma V^\pi(s')]. \tag{2.4.2}$$

Closely related to the state-value function is the action-value function with Bellman equation

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s'|s,a\sim P}\mathbb{E}_{a'|s'\sim\pi} Q^\pi(s',a').$$

The two are related by

$$\begin{aligned} V^\pi(s) &= \gamma \mathbb{E}_{a|s\sim\pi} Q^\pi(s,a) \\ Q^\pi(s,a) &= r(s,a) + \mathbb{E}_{s'|s,a\sim P} V^\pi(s). \end{aligned}$$

The appeal of Q-functions is the ability to convert them into policies, particularly the optimal policy $\pi(\tau_t) = \operatorname{argmax}_a Q^\pi(s_t, a)$.

2.4.1. Temporal Difference Learning

Given a policy π, the straightforward way to learn value function approximations $\hat{V}_\phi(s) \approx V^\pi(s)$ or $\hat{Q}_\phi(s,a) \approx Q^\pi(s,a)$ with parameters ϕ, is to collect finite trajectories τ and minimize the error between both sides of Equation 2.4.1.

$$\begin{aligned}&\text{argmin}_\phi \, \mathbb{E}_{\tau \sim \text{Pr}_\mu^\pi}(R(\tau) - \hat{V}_\phi(s_0))^2, \quad \text{or} \\ &\text{argmin}_\phi \, \mathbb{E}_{\tau \sim \text{Pr}_\mu^\pi}(R(\tau) - \hat{Q}_\phi(s_0, a_0))^2.\end{aligned}$$

However, a more sample-efficient approach is to learn to satisfy the Bellman equation (2.4.2), effectively using our own predictions to provide biased but low-variance estimates of $R(\tau^t)$ for all steps t:

$$\begin{aligned}&\text{argmin}_\phi \, \mathbb{E}_{\tau \sim \text{Pr}_\mu^\pi} \sum_t (r_t + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t))^2 \quad \text{or,} \\ &\text{argmin}_\phi \, \mathbb{E}_{\tau \sim \text{Pr}_\mu^\pi} \sum_t (r_t + \gamma \hat{Q}_\phi(s_{t+1}, a_{t+1}) - \hat{Q}_\phi(s_t, a_t))^2.\end{aligned}$$

This is called *bootstrapping*, and it greatly speeds up learning.

Usually, an RL agent does not have the luxury of learning upfront from an idly collected set of trajectories – it has to learn *while* acting on the environment and incurring rewards and punishments. In this setting there is a crucial tradeoff between exploration and exploitation: do I do what I think is best, or do I try something new in hopes of discovering something even better?

A classic approach to retain exploration is ϵ-greedy: take the greedy action $\text{argmax}_a \hat{Q}_\phi(s,a)$ most of the time, but with vanishing probability take a uniformly random action instead. Sarsa provides an algorithm for learning the values for such an ergodic policy:

$$\phi_{t+1} \;=\; \phi_t + \eta(r_t + \gamma \hat{Q}_{\phi_t}(s_{t+1}, a_{t+1}) - \hat{Q}_{\phi_t}(s_t, a_t)) \nabla_\phi \hat{Q}_{\phi_t}(s_t, a_t).$$

Sarsa learns values for the noisy policy; as the noise is annealed away, its values converge to those of the optimal policy.

A related algorithm is Q-learning, which allows learning the values of the optimal policy directly, while acting according to an exploratory policy:

$$\phi_{t+1} \;=\; \phi_t + \eta(r_t + \gamma \max_a \hat{Q}_{\phi_t}(s_{t+1}, a) - \hat{Q}_{\phi_t}(s_t, a_t)) \nabla_\phi \hat{Q}_{\phi_t}(s_t, a_t).$$

When the model $\hat{Q}_{\phi_t}$ is a deep neural network, the algorithm is not guaranteed to converge. However, it generally converges in practice with some engineering tricks (Mnih et al., 2015; Hessel et al., 2018).

2.4.2. Policy Gradient Methods

In Q-learning, we implicitly learned a policy by learning its action values. We can also directly learn a policy by maximizing

$$\mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} \sum_{t=0}^{\infty} R(\tau)$$

with gradient descent. Whereas in supervised learning we had an expectation over a fixed data distribution $p(x)$, now we have an expectation over a trajectory distribution $\Pr_\mu^{\pi_\theta}(\tau)$ that depends on our policy parameter θ. In taking the gradient of the expectation, we can no longer simply take the expectation of the gradient.

REINFORCE (Williams, 1992) gives us the gradient of the expected return in the form of an expected gradient which we can sample as usual:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} R(\tau) &= \int R(\tau) \nabla_\theta \Pr_\mu^{\pi_\theta}(\tau) d\tau \\
&= \int R(\tau) \frac{\Pr_\mu^{\pi_\theta}(\tau)}{\Pr_\mu^{\pi_\theta}(\tau)} \nabla_\theta \Pr_\mu^{\pi_\theta}(\tau) d\tau \\
&= \int R(\tau) \Pr_\mu^{\pi_\theta}(\tau) \frac{\nabla_\theta \Pr_\mu^{\pi_\theta}(\tau)}{\Pr_\mu^{\pi_\theta}(\tau)} d\tau \\
&= \int R(\tau) \Pr_\mu^{\pi_\theta}(\tau) \nabla_\theta \log \Pr_\mu^{\pi_\theta}(\tau) d\tau \\
&= \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} R(\tau) \nabla_\theta \log \Pr_\mu^{\pi_\theta}(\tau).
\end{aligned}$$

It is worth unpacking $R(\tau)$ and $\Pr_\mu^{\pi_\theta}(\tau)$ to omit noncausal terms $r(s_{t'}, a_{t'}) \log \pi_\theta(a_t \mid s_t)$ with $t' < t$; these terms have mean zero but contribute variance to the estimator. We are left with

$$\nabla_\theta \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} R(\tau) = \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} \nabla_\theta \sum_{t=0}^{\infty} \gamma^t R(\tau^t) \log \pi_\theta(a_t \mid s_t).$$

The mechanism of REINFORCE is intuitive: reward or punish actions according to the returns they achieve down the line. Unfortunately, the estimator has high variance. A typical variance reduction technique is to subtract a baseline prediction $\hat{V}(s_t)$ of the future return $R(\tau^t)$:

$$\nabla_\theta \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} R(\tau) = \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_\theta}} \nabla_\theta \sum_{t=0}^{\infty} \gamma^t (R(\tau^t) - \hat{V}(s_t)) \log \pi_\theta(a_t \mid s_t).$$

As the notation suggests, the ideal baseline is $\hat{V}(s_t) = V^{\pi_\theta}(s_t)$, and we saw in the previous section how we may learn to predict it. We may further reduce variance by writing $R(\tau^t)$ in terms of $R(\tau^{t+1})$ and replacing the noisy sample $R(\tau^{t+1})$ by a deterministic prediction

$\hat{V}(s_{t+1})$:

$$\begin{aligned}
\mathbb{E}_{\tau\sim\Pr_\mu^\pi} R(\tau^t) &= \mathbb{E}_{\tau\sim\Pr_\mu^\pi} r_t + \gamma R(\tau^{t+1}) \\
&= \mathbb{E}_{\tau_t\sim\Pr_\mu^\pi} r_t + \gamma \mathbb{E}_{\tau^{t+1}|\tau_t\sim\Pr_\mu^\pi} R(\tau^{t+1}) \\
&= \mathbb{E}_{\tau_t\sim\Pr_\mu^\pi} \mathbb{E}_{s_{t+1}|s_t,a_t\sim P} r_t + \gamma V^\pi(s_{t+1}) \\
&\approx \mathbb{E}_{\tau_t\sim\Pr_\mu^\pi} \mathbb{E}_{s_{t+1}|s_t,a_t\sim P} r_t + \gamma \hat{V}(s_{t+1}).
\end{aligned}$$

While this reduces variance, it does so at the cost of introducing a bias insofar as the prediction $\hat{V}(s_{t+1})$ deviates from the true value $V^{\pi_\theta}(s_{t+1})$.

This substitution, a form of bootstrapping, brings us to the advantage actor-critic estimator (A2C; Mnih et al., 2016):

$$\nabla_\theta \mathbb{E}_{\tau\sim\Pr_\mu^{\pi_\theta}} R(\tau) \approx \mathbb{E}_{\tau\sim\Pr_\mu^{\pi_\theta}} \nabla_\theta \sum_{t=0}^{\infty} \gamma^t (r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \log \pi_\theta(a_t \mid s_t).$$

This estimator is routinely used in practice and works well with neural network actors π_θ and critics $\hat{V}_\phi$.

2.5. Multi-Agent Reinforcement Learning

When multiple agents interact with and learn from a shared environment, the single-agent perspective considered so far becomes problematic. Multi-Agent Reinforcement Learning (MARL) lies at the intersection of game theory and reinforcement learning.

The generalization of the earlier MDP formalism is straightforward: we now have a *joint* action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, the transition $P : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ operator takes *joint* actions, and the reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^n$ emits a *vector* of rewards, one for each of the n agents. The agents follow a *joint* policy $\pi = (\pi_1, \ldots, \pi_n)$, producing conditionally independent actions. Each agent seeks a policy π_i that maximizes its own expected return $\mathbb{E}_\tau R_i(\tau)$.

A multi-agent learning problem is similar to a multi-objective optimization problem in that there exists no notion of *joint* maximum. The closest notion is that of a Pareto optimum (Pareto, 1919), a solution $\pi^\star$ for which there is no other solution π that makes some player better off without making others worse off, i.e.

$$\mathbb{E}_{\tau\sim\Pr_\mu^{\pi^\star}} R_i(\tau) \geqslant \mathbb{E}_{\tau\sim\Pr_\mu^{\pi}} R_i(\tau)$$

for all π, i. However, Pareto optima were developed in an economics context where the joint policy π can be imposed by a central authority.

In multi-agent systems, the joint policy instead comes about through agents choosing their own policies. If some agent can improve their outcome at the expense of others, then they will do so. Effectively, this restricts us to joint policies where no agent wishes

to change their policy. This is the notion of a Nash equilibrium (Nash Jr, 1950), a choice of joint policy $(\pi_1, \ldots, \pi_n)$ for which it holds that each π_i maximizes $\mathbb{E}_\tau R_i(\tau)$ given the other $\pi_j, j \neq i$. For example, in two-player systems:

$$\begin{aligned} \pi_1 &\in \operatorname{argmax}_x \mathbb{E}_{\tau \sim \Pr_\mu^{x,\pi_2}} R_1(\tau) \\ \pi_2 &\in \operatorname{argmax}_y \mathbb{E}_{\tau \sim \Pr_\mu^{\pi_1,y}} R_2(\tau). \end{aligned}$$

The problem with Nash equilibria is that they are generally not unique, and different equilibria will provide different utility to different agents.

We will take the Iterated Prisoner's Dilemma (Axelrod and Hamilton, 1981) as a running example. The IPD is a repeated matrix game where, each round, players choose to either cooperate or defect, after observing the joint action from the prior round. Formally, the state space $\mathcal{S} = \{\emptyset, \text{CC}, \text{CD}, \text{DC}, \text{DD}\}$ consists of a special initial state $\emptyset$ and the set of joint actions. The payoff matrix in Table 2.1 specifies the joint reward function as a function of the joint action.

	C	D
C	$(-1, -1)$	$(-3, -0)$
D	$(-0, -3)$	$(-2, -2)$

Table 2.1. Iterated Prisoner's Dilemma payoffs

In this game, players have a tendency to defect, even though they both prefer CC to DD. Arriving at and maintaining CC requires coordination. The famous tit-for-tat policy (Axelrod and Hamilton, 1981) uses threats of defection to induce cooperation in its opponents. Tit-for-tat cooperates, but it punishes defection with defection. This can be used as a way to coordinate joint policies without a central authority.

In IPD, the solution where both players defect all the time is a Nash equilibrium, but both players would prefer the Nash equilibrium where both players play tit-for-tat. The latter is both a Pareto optimum and a (weak) Nash equilibrium at once, which is ideal. In general, however, these sets of solutions are disjoint, and it is unclear how to characterize what solution is desirable. Nevertheless, it should be one goal in MARL to avoid equilibria that are dominated (in the Pareto sense) by other equilibria. The IPD, as simple as it is, provides a basic testbed.

There is one other complication to be discussed: we will be *learning* policies, that is, *gradually* improving them using gradient descent. The characterizations of Nash equilibrium and Pareto optimality instead are *global*, speaking of argmaxes and existences without regard for the topology of the joint policy space and the path by which an improved solution may be reached.

We can view the multi-agent MDP as a differentiable game, a function $f(x)$ that maps joint policies x to joint expected returns. The joint gradient (better known as the *naive* or

simultaneous gradient) will be written as

$$\bar{\nabla} f(x) = \begin{pmatrix} \nabla_{x_1} f_1(x) \\ \nabla_{x_n} f_n(x) \end{pmatrix}.$$

The local analog of a Nash equilibrium is a stable fixed point, a point at which the gradient is zero and the curvature is negative (roughly; see Letcher (2018) for the intricate details). Basically, a point where, if you start close enough to it, gradient descent with small enough learning rate will converge to it.

Naive learning is the simplest way to apply RL to multi-agent systems; we simply let each agent learn independently, treating other agents as part of the environment. In RL terms, naive learning makes the environment nonstationary – the transition probabilities P in each agent's MDP would come to depend on the policies of other agents, which change over time.

In optimization terms, the gradient becomes less reliable as an improvement direction *even locally*. The gradient derives from a first-order approximation of f, which measures *independent* contributions of each element of x to $f(x)$. This is already violated in single-objective optimization, e.g. neural network training, but at least there the different elements of x are being updated in service of a shared objective f. When different elements optimize for different objectives, the gradient may lose its meaning entirely.

Several approaches address the issue by "looking ahead" (Zhang and Lesser, 2010), effectively performing naive learning on a modified game $\tilde{f}$ that evalutes f after an imagined naive update:

$$\tilde{f}(x) = f(x + \alpha \bar{\nabla} f(x)).$$

Learning with Opponent Learning Awareness (LOLA; Foerster et al., 2018) proposed to capture the second-order dependency on x through the imagined update. It was the first general learning algorithm to consistently find tit-for-tat on the IPD. However, it has an inconsistency: each player assumes its opponent to be naive, just like naive learning assumes its opponent to stand still.

A consistent version of LOLA would use the definition

$$\tilde{f}(x) = f(x + \alpha \bar{\nabla} \tilde{f}(x)),$$

however this is an implicit equation that is not useful in practice. COLA (Willi et al., 2022) proposed to approximate $\tilde{f}$ with a model trained to minimize the inconsistency.

POLA (Zhao et al., 2022) improved the basic LOLA algorithm by replacing all gradient updates by proximal updates, which are improvement directions subject to function-space step size penalties. The result is an approximately parameterization-invariant LOLA that can be applied in the case where policies x are represented by neural network parameters.

In Chapter 5 we will discuss an approach that essentially takes LOLA to an extreme of looking infinitely many steps ahead. The lookahead process is amortized by a learned parametric model, which at the same time serves to smooth the result, as it will typically be almost piecewise constant with respect to x.

In order to look farther ahead in optimization time, some have turned to the meta-game in which joint policies are meta-states and returns $f(x)$ are meta-rewards. Now we seek to maximize the expected meta-return

$$\mathbb{E}_\Omega \sum_{t=0}^{\infty} \gamma^t f(x^{(t)}), \tag{2.5.1}$$

under the optimization process

$$\begin{aligned} x_1^{(t+1)} &\sim \Omega_1(\cdot \mid x^{(t)}) \\ x_2^{(t+1)} &\sim \Omega_2(\cdot \mid x^{(t)}) \end{aligned}$$

governed by the meta-policies Ω_1, Ω_2.

Meta-PG (Al-Shedivat et al., 2017) first considered this meta-game, and Meta-MAPG (Kim et al., 2021) specialized it to the multi-agent setting. Both propose to use (simultaneous) policy gradients to maximize (2.5.1) with respect to x. However, both estimate the gradient of (2.5.1) using naive rollouts (i.e. Ω being a deterministic naive gradient update) and hence are not consistent with the true process by which x is being updated.

M-FOS (Lu et al., 2022) solves the inconsistency by allowing arbitrary parameteric meta-policies Ω_θ, and using (simultaneous) policy gradients to maximize (2.5.1) with respect to θ. Now the meta-policy can be used both in the rollouts to estimate (2.5.1) *and* as the true meta-policy.

We propose in Chapter 6 our own method based on the meta-game formalism. Our method uses a model of (2.5.1) to solve the inconsistency in Meta-MAPG, while retaining a gradient-based meta-policy Ω. In essence, we replace the original game f by a modified game

$$V(x) = f(x) + \gamma V(x + \alpha \bar{\nabla} V(x)),$$

on which naive gradient descent is coordinated.

2.6. Neural Networks

Neural networks are an ancient and venerable order of extremely flexible differentiable functions that, despite some false starts, turns out to be easy to optimize. The nature of this apparently free lunch is still under active study (Zhang et al., 2021; Belkin et al., 2019; Nakkiran et al., 2021).

The simplest example is a single-layer network – an affine transformation followed by a nonlinear activation function:

$$f(x;\theta) = \sigma(Wx + b).$$

The weight matrix W and bias vector b make up the parameters $\theta = \begin{pmatrix} \mathrm{vec}(W)^\top & b^\top \end{pmatrix}^\top$ of the model. By adjusting their values, we can express different functions $f(\cdot;\theta)$. The activation is typically either a threshold function (e.g. the ReLU $\sigma(x) = \max\{x, 0\}$) or a squashing function (e.g. the hyperbolic tangent $\sigma(x) = \tanh(x)$).

In modern times we treat this basic structure as a building block. Networks get much more interesting when we arrange multiple blocks in sequence. For example, a deep neural network with L layers:

$$\begin{aligned} h^{(0)} &= x \\ h^{(l)} &= f^{(l)}(h^{(l-1)};\theta^{(l)}) = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}) \\ f(x;\theta) &= h^{(L)}, \end{aligned}$$

with parameters $\theta = \begin{pmatrix} \theta^{(1)\top} & \cdots & \theta^{(L)\top} \end{pmatrix}^\top$.

The expressivity of neural networks grows exponentially in the number of layers (Montufar et al., 2014), which in part explains the success of deep learning. Each layer has its own hidden units $h^{(l)}$; functions of the input x that it is free to choose and that will often come to represent useful abstract features of the input (except for the last layer, whose values will be constrained to match data). If x represents an image of a face, for example, such features include the pose, the lighting, the expression, the presence or absence of facial hair and glasses, and so on. These are features that would be very hard to extract by manually writing a program, but neural networks often learn to extract them just as a means to some downstream end. The power of neural network depth stems from the ability to *reuse* features for multiple purposes.

In order to train a neural network, we construct a performance metric that tells us how good (or bad) the model is, and then iteratively improve it. Typically, we construct a per-example loss $\mathcal{L}(\theta; x, y)$ as a function of the parameters and minimize its expectation using some form of gradient descent:

$$\theta \leftarrow \theta - \eta \mathbb{E}_{x,y} \nabla \mathcal{L}(\theta; x, y).$$

The expectation here is over a large set of training examples, but in practice we often use small randomized subsets to estimate the expectation. This is stochastic gradient descent, and it exploits a common theme in optimization, which is that taking many cheap but approximate steps is more efficient than taking few costly but accurate steps.

Either way, we need to compute the gradient $\nabla \mathcal{L}(\theta; x, y)$ to proceed. For the sake of demonstration, we let the per-example loss be the squared error between the network's output and the target y:

$$\mathcal{L}(\theta; x, y) = \mathcal{D}(y, f(x; \theta)), \qquad \mathcal{D}(y, \hat{y}) = \frac{1}{2}\|\hat{y} - y\|^2.$$

By the chain rule of differentiation, we can relate the error to the model output:

$$\begin{aligned} (\nabla \mathcal{L}(\theta; x, y))^\top &= \frac{\partial \mathcal{D}}{\partial \hat{y}}(y, f(x; \theta)) \frac{\partial f}{\partial \theta}(x; \theta) \\ &= (f(x; \theta) - y)^\top \frac{\partial f}{\partial \theta}(x; \theta). \end{aligned}$$

This takes the form of a vector-Jacobian product $u^\top \frac{\partial f}{\partial \theta}(x; \theta)$, which we can compute cheaply by backpropagation Rumelhart et al., 1986:

$$\begin{aligned} u^\top \frac{dh^{(L)}}{dh^{(L)}} &= u^\top \\ u^\top \frac{dh^{(L)}}{dh^{(l-1)}} &= u^\top \frac{dh^{(L)}}{dh^{(l)}} \frac{\partial f^{(l)}}{\partial h^{(l-1)}}(h^{(l-1)}; \theta^{(l)}) \\ &= u^\top \frac{dh^{(L)}}{dh^{(l)}} \sigma'(W^{(l)} h^{(l-1)} + b^{(l)}) W^{(l)}. \end{aligned}$$

We iterate over the layers in reverse order, maintaining the projected derivatives $u^\top \frac{dh^{(L)}}{dh^{(l)}}$ with respect to the current layer's output. We could compute the entire Jacobian $\frac{\partial f}{\partial \theta}(x; \theta)$ using similar recursions, before finally reducing it with the vector u. What makes backpropagation efficient is that we do not need to explicitly compute any Jacobians, and only ever work with vectors shaped like $h^{(l)}$.

Given $u^\top \frac{dh^{(L)}}{dh^{(l)}}$, we can compute the layer's contribution to the gradient using its input $h^{(l-1)}$ only:

$$\begin{aligned} u^\top \frac{dh^{(L)}}{d\theta^{(l)}} &= u^\top \frac{dh^{(L)}}{dh^{(l)}} \frac{\partial f^{(l)}}{\partial \theta^{(l)}}(h^{(l-1)}; \theta^{(l)}) \\ &= \begin{pmatrix} \operatorname{vec}\left(\left(u^\top \frac{dh^{(L)}}{dh^{(l)}} \sigma'(z^{(l)}) \right)^\top (h^{(l-1)})^\top \right)^\top & u^\top \frac{dh^{(L)}}{dh^{(l)}} \sigma'(z^{(l)}) \end{pmatrix}, \end{aligned}$$

where $z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$. In practice we rarely need to derive these gnarly expressions by hand – we use automatic differentiation software instead that can handle general directed acyclic computation graphs, such as Theano (Al-Rfou et al., 2016), Tensorflow (Abadi et al., 2016), PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018).

What we have described is the simple case: a series of fully-connected layers. It is often useful to restrict the connectivity W between units in consecutive layers, e.g. to enforce domain knowledge. A prominent example is a convolutional neural network (CNN;

LeCun, Bengio, et al., 1995), in which we have sequences (or 2D arrays as in images) of hidden vectors $h_t^{(l)}$ and a short sequence of weight matrices $W_k^{(l)}$ to aggregate information from neighboring hidden vectors $h_{t-k}^{(l)}$:

$$h_t^{(l)} = \sum_k W_k^{(l)} h_{t-k}^{(l-1)} + b^{(l)}.$$

With an appropriate flattening of dimensions, this can be seen as a regular layer with a Toeplitz structure in W. This restriction makes the transformation equivariant to translations; in images for example, it means that if a network is able to recognize a cat in the bottom left, it will also be able to recognize it anywhere else. This alleviates the need to collect a large variety of training data. The notion of convolution can be generalized to graphs that aren't uniform grids (Bronstein et al., 2021), where it brings similar benefits.

Next, we will consider a network structure that is central to the work discussed in Chapter 4: recurrent neural networks (RNNs; Elman, 1990). Like CNNs, RNNs map sequences to sequences, but they can effectively aggregate information over much longer horizons (infinitely long in principle, although we still struggle to train them). An RNN has the structure

$$h_t = f_\theta(h_{t-1}, x_t),$$

for example

$$h_t = \sigma(W h_{t-1} + U x_t + b),$$

a simple nonlinear transformation, repeatedly applied over time t. RNNs are special in that the computational path from h_1 to h_t goes through $t-1$ activation functions, producing a very nonlinear dependency. Moreover, as the time t grows, the dynamics of the backpropagation procedure to compute $u^\top \frac{dh_t}{dh_1}$ become hard to control, mainly due to the spectral norm of W. The backpropagated vector $u^\top \frac{dh_t}{dh_1}$ will tend to either grow or shrink exponentially; this is known as *exploding gradients* and *vanishing gradients* respectively (Bengio et al., 1994). Best practice is to use a saturating nonlinearity with contractive Jacobian, such as the hyperbolic tangent, to avoid exploding in favor of vanishing. Vanishing can subsequently be alleviated through the use of skip-connections, as in Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997).

RNNs are often used for streaming tasks such as language modeling, where a text comes in one word at a time and the model is tasked with predicting the next word. In this case, at each time step, the model's input is the current word, the output is a categorical distribution $q(x_{t+1} \mid h_t)$ over the next word (e.g. from a classifier on top of the hidden state h_t), and the loss is the negative log-likelihood of correctly predicting the next word:

$$\mathcal{L}(\theta; x) = \sum_t \mathcal{L}_t(h_t; x) = -\sum_t \log q(x_{t+1} \mid h_t).$$

The causal structure of the gradient $\frac{d\mathcal{L}(\theta;x)}{d\theta}$ admits two convenient factorizations:

$$\frac{d\mathcal{L}}{d\theta} = \underbrace{\sum_t \frac{d\mathcal{L}}{dh_t}\frac{\partial h_t}{\partial \theta}}_{\text{reverse mode}} = \underbrace{\sum_t \frac{\partial \mathcal{L}_t}{\partial h_t}\frac{dh_t}{d\theta}}_{\text{forward mode}},$$

where we have used the simplified notation

$$\frac{d\mathcal{L}}{dh_t} = \frac{d\mathcal{L}(\theta;x)}{dh_t}, \quad \frac{\partial \mathcal{L}_t}{\partial h_t} = \frac{\partial \mathcal{L}_t}{\partial h}(h_t; x_{t+1}), \quad \frac{\partial h_t}{\partial \theta} = \frac{\partial f_\theta}{\partial \theta}(h_{s-1}, x_s).$$

The reverse-mode factorization can be computed cheaply by Back-Propagation Through Time (BPTT; Werbos, 1990), using the recursions

$$\begin{aligned} \frac{d\mathcal{L}}{dh_t} &= \frac{d\mathcal{L}}{dh_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial \mathcal{L}_t}{\partial h_t} \\ g_t &= g_{t+1} + \frac{d\mathcal{L}}{dh_t}\frac{\partial h_t}{\partial \theta}. \end{aligned}$$

to compute $g_0 = \frac{d\mathcal{L}}{d\theta}$. These recursions go backward in time, so the practical implementation on (infinite) streaming tasks is to take a short chunk, run the model forward and then back up to compute the gradient on the chunk.

Real-Time Recurrent Learning (RTRL; Williams and Zipser, 1989) exploits the forward-mode factorization to compute $g_{t_{\max}} = \frac{d\mathcal{L}}{d\theta}$:

$$\begin{aligned} \frac{dh_t}{d\theta} &= \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{d\theta} + \frac{\partial h_t}{\partial \theta} \\ g_t &= g_{t-1} + \frac{\partial \mathcal{L}_t}{\partial h_t}\frac{dh_t}{d\theta}. \end{aligned}$$

RTRL is computed forward in time in lockstep with the model, and the terms accumulated into g_t can be directly used to update the parameters to enable online learning. However, whereas BPTT maintains a vector $\frac{d\mathcal{L}}{dh_t}$, RTRL maintains a *matrix* $\frac{dh_t}{d\theta}$, which makes it very expensive in terms of both time and space to compute.

UORO (Tallec and Ollivier, 2018) and its predecessor NoBackTrack (Ollivier et al., 2015) address this computational expense through a rank-one approximation:

$$\frac{dh_t}{d\theta} = \sum_{s\leqslant t}\frac{dh_t}{dh_s}\frac{\partial h_s}{\partial \theta} \approx \left(\sum_{s\leqslant t}\frac{dh_t}{dh_s}\nu_s\right)\left(\sum_{s\leqslant t}\nu_s^\top \frac{\partial h_s}{\partial \theta}\right) \triangleq \tilde{h}_t \tilde{w}_t^\top,$$

where the ν_s are iid random vectors that satisfy $\mathbb{E}[\nu_t \nu_t^\top] = I$, for example $\nu_t \sim \mathcal{N}(0, I)$.

The projection onto random vectors makes the approximation cheap to maintain:

$$\begin{aligned} \tilde{h}_t &= \frac{\partial h_t}{\partial h_{t-1}}\tilde{h}_{t-1} + \nu_t \\ \tilde{w}_t^\top &= \tilde{w}_{t-1}^\top + \nu_t^\top \frac{\partial h_t}{\partial \theta}. \end{aligned}$$

Effectively, it destroys information about the spatial (between units) and temporal (between time steps) connectivity of the model. However, the coordinated structure of these projections serves to restore the true graph in expectation.

BPTT, RTRL and UORO will be discussed extensively in Chapter 4.

2.7. Optimization

In the previous section, we introduced (stochastic) gradient descent:

$$\theta \leftarrow \theta - \eta g,$$

where $g = \mathbb{E}_{x,y} \nabla \mathcal{L}(\theta; x, y)$ is the (stochastic) gradient of the task loss. Much of deep learning revolves around improving the speed and quality of this optimization process. However, we are not ultimately interested solely in reducing the training loss; we wish in particular to find solutions that generalize well beyond the training set.

If we think of the task loss $\mathbb{E}_{x,y} \mathcal{L}(\theta; x, y)$ as the height of a landscape at different spatial locations θ, then gradient descent is the algorithm that aims to find the lowest point by always going down in the steepest direction available, at a rate proportional to the slope. This is a very naive way of finding the lowest point, and it has some issues:

- It can get stuck in local optima – solutions that are better than their immediate neighborhood, but worse than some other more remote solutions. Bowls, for example, provide local minima for soup to get stuck in.
- It can spend a lot of time traversing plateaus, where there is a definite direction of improvement but the slope is minimal. Increasing the learning rate helps, but hurts when the end of the plateau is reached.
- It can overshoot the lowest point and spend time oscillating around it. Reducing the learning rate helps.
- It can spend a lot of time around saddle points – solutions of mixed curvature, where the gradient is very small.

One of the most important techniques to improve optimization is momentum (Sutskever et al., 2013). Momentum uses an exponentially decaying sum of past gradients to update the parameter, which effectively adapts the learning rate for each parameter element:

$$\begin{aligned} \mu &\leftarrow \beta\mu - \eta g \\ \theta &\leftarrow \theta + \mu. \end{aligned}$$

Intuitively, momentum accelerates on plateaus and saddles and dampens oscillations in narrow valleys.

Another set of techniques is based on the idea of taking the sign of each gradient element, discarding its scale (Riedmiller and Braun, 1993; Lydia and Francis, 2019). The most popular of these is RMSProp (Tieleman, Hinton, et al., 2012), which keeps an exponential moving estimate of the second moment of each gradient element, and divides the gradient by the square root of it:

$$\begin{aligned} \nu &\leftarrow \beta\nu + (1-\beta)g^2 \\ \theta &\leftarrow \theta + \frac{g}{\epsilon + \sqrt{\nu}}. \end{aligned}$$

This is helpful because large gradient elements correspond to sensitive parameters which we would like to change carefully, and vice versa. AdaGrad (Duchi et al., 2011) combines the ideas of momentum and sign methods; its successor Adam (Kingma and Ba, 2014) is currently the most popular optimization method by far.

In deep learning, we can improve optimization not just by adjusting the optimization algorithm; we may also adjust the optimization problem. We are free to modify (*reparameterize*) the neural network architecture, as long as the function class f_θ remains sufficiently flexible. Several techniques have been discovered to combat the tendency of gradients to vanish as they propagate through many layers.

The earliest breakthrough in this direction is the skip-connection: a shortcut between layers that would otherwise be far apart. Such shortcuts reduce the effective depth, while maintaining the expressiveness of the network. The idea is used to great effect in Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997), an RNN architecture. It was later carried over to feed-forward neural networks to enable training neural networks with hundreds of layers (Srivastava et al., 2015; He et al., 2016).

Another advance was to switch to rectified linear units (ReLUs) as activation functions, away from squashing functions such as the logistic function and hyperbolic tangent which easily saturate and yield small gradients (Krizhevsky et al., 2012). Much work has gone into drawing good values θ from which to start optimization, to avoid starting out on such plateaus in the first place (Glorot and Bengio, 2010; He et al., 2015).

Batch Normalization (Ioffe and Szegedy, 2015) pioneered the idea of standardizing activations by subtracting their means and dividing by their standard deviations, before passing to the next step in the computation. Typical use is to standardize before the activation function, with a parametric scale and shift to enable the network to control the distribution going into the activation function. The use of normalization makes networks stable by default: the means and variances of hiddens coming out of a layer no longer depend on those of the hiddens coming into the layer. Batch Normalization was a true breakthrough that sped up training by an order of magnitude. It inspired several variations

(Ba et al., 2016; Salimans and Kingma, 2016; Ulyanov et al., 2016), including an extension to recurrent neural networks by myself (Cooijmans et al., 2016).

Finally, we must mention Dropout (Srivastava et al., 2014), a fully general regularization technique that makes optimization slower but greatly improves generalization. Dropout simply randomly zeroes out a subset of the activations going into the next layer, thus breaking the model's ability to rely on all features being available at all times. The overall procedure can be interpreted as training an ensemble of randomly pruned networks with shared parameters.

All of these ideas are considered best practice today, and deep learning would not be possible without them.

Chapter 3

Counterpoint by Convolution

Cheng-Zhi Anna Huang[*†] Tim Cooijmans[*†] Adam Roberts[♯]
Aaron Courville[†] Douglas Eck[♯]

[*] Equal contribution [†] Mila & Université de Montréal [♯] Google Brain

Machine learning models of music typically break up the task of composition into a chronological process, composing a piece of music in a single pass from beginning to end. On the contrary, human composers write music in a nonlinear fashion, scribbling motifs here and there, often revisiting choices previously made. In order to better approximate this process, we train a convolutional neural network to complete partial musical scores, and explore the use of blocked Gibbs sampling as an analogue to rewriting. Neither the model nor the generative procedure are tied to a particular causal direction of composition. Our model is an instance of orderless NADE (Uria et al., 2014), which allows more direct ancestral sampling. However, we find that Gibbs sampling greatly improves sample quality, which we demonstrate to be due to some conditional distributions being poorly modeled. Moreover, we show that even the cheap approximate blocked Gibbs procedure from Yao et al. (2014) yields better samples than ancestral sampling, based on both log-likelihood and human evaluation.

Prologue

This project was conceived during an internship at Google Brain in the summer of 2016. Anna Huang, Kyle Kastner, Natasha Jaques and myself interned with the Magenta project, focused on the intersection of machine learning and music. Fresh on our minds was the recent work on PixelCNN (Oord et al., 2016a), an autoregressive model of images that predicts pixels one by one in scanline order (left to right, top to bottom). The results were impressive at the time, but the prediction ordering felt awkward: whether the model generates a cat or a dog is decided somewhere along the way from the top to the bottom. In theory, any ordering can lead to a valid model of the data distribution, but it seemed like the ordering could matter quite a bit in practice. Moreover, predicting variables one

by one without ever revisiting past predictions struck us as unnatural regardless of the ordering, especially in our chosen domain of music composition.

Anna initiated the project and came up with the original idea of an "inpainting" model that could fill in arbitrary blanks in musical scores. She also came up with the idea of using a block-Gibbs sampling procedure to let this model revisit its own predictions. I implemented generative procedures, fast batched evaluation, and batch normalization. After our internships ended, we continued the work at Mila (then LISA) with my advisor Aaron. Anna and I worked together closely; we both ran many experiments, and sat down to listen to samples multiple times per day. I worked out a formal interpretation of the training procedure as learning to predict along all possible orderings, and from it derived a few modifications to ensure uniform training. We ultimately discovered that this analysis had already been done by Uria et al. (2014), and credit that work accordingly. In the end, we found our iterative, orderless approach produced considerably higher-quality music than a single-pass ordered approach.

3.1. Introduction

Counterpoint is the process of placing notes against notes to construct a polyphonic musical piece (Fux, 1965). This is a challenging task, as each note has strong musical influences on its neighbors and notes beyond. Human composers have developed systems of rules to guide their compositional decisions. However, these rules sometimes contradict each other, and can fail to prevent their users from going down musical dead ends. Statistical models of music, which is our current focus, is one of the many computational approaches that can help composers try out ideas more quickly, thus reducing the cost of exploration (Fernández and Vico, 2013).

Whereas previous work in statistical music modeling has relied mainly on sequence models such as Hidden Markov Models (HMMs; Baum and Petrie, 1966) and Recurrent Neural Networks (RNNs Rumelhart et al., 1988), we instead employ convolutional neural networks due to their invariance properties and emphasis on capturing local structure. Nevertheless, they have also been shown to successfully model large-scale structure (Oord et al., 2016a; Oord et al., 2016b). Moreover, convolutional neural networks have shown to be extremely versatile once trained, as demonstrated by a variety of creative uses such as DeepDream (Mordvintsev et al., 2015) and style transfer (Gatys et al., 2015).

We introduce COCONET, a deep convolutional model trained to reconstruct partial scores. Once trained, COCONET provides direct access to all conditionals of the form $p(\mathbf{x}_i \mid \mathbf{x}_C)$ where C selects a fragment of a musical score $\mathbf{x}$ and $i \notin C$ is in its complement. COCONET is an instance of deep orderless NADE (Uria et al., 2014), which learns an ensemble of factorizations of the joint $p(\mathbf{x})$, each corresponding to a different ordering. A

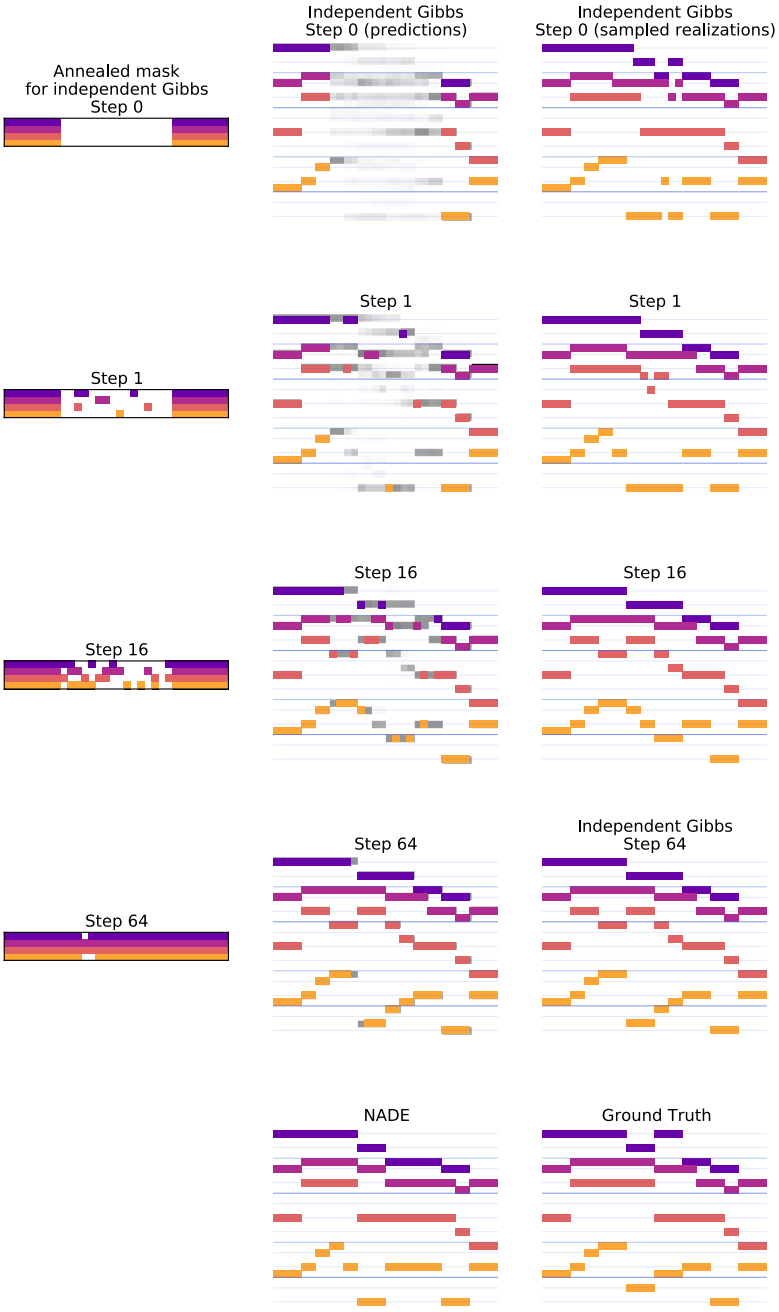


Fig. 3.1. Blocked Gibbs inpainting of a corrupted Bach chorale by COCONET. At each step, a random subset of notes is removed, and the model is asked to infer their values. New values are sampled from the probability distribution put out by the model, and the process is repeated. Left: annealed masks show resampled variables. Colors distinguish the four voices. Middle: grayscale heatmaps show predictions $p(\mathbf{x}_j \mid \mathbf{x}_C)$ summed across instruments. Right: complete pianorolls after resampling the masked variables. Bottom: a sample from NADE (left) and the original Bach chorale fragment (right).

related approach is multi-prediction training of deep Boltzmann machines (Goodfellow et al., 2013), which also gives a model that can predict any subset of variables given its complement.

However, the sampling procedure for orderless NADE treats the ensemble as a mixture and relies heavily on ordering. Sampling from an orderless NADE involves (randomly) choosing an ordering, and sampling variables one by one according to the chosen ordering. This process is called *ancestral sampling*, as the order of sampling follows the directed structure of the model. We have found that this produces poor results for the highly structured and complex domain of musical counterpoint.

Instead, we propose to use blocked-Gibbs sampling, a Markov Chain Monte-Carlo (MCMC) method to sample from a joint probability distribution by repeatedly resampling subsets of variables using conditional distributions derived from the joint probability distribution. An instance of this was previously explored by Yao et al. (2014) who employed a NADE in the transition operator for a Markov chain, yielding a Generative Stochastic Network (GSN). The transition consists of a corruption process that masks out a subset $\mathbf{x}_{\neg C}$ of variables, followed by a process that independently resamples variables $\mathbf{x}_i$ (with $i \notin C$) according to the distribution $p_\theta(\mathbf{x}_i \mid \mathbf{x}_C)$ emitted by the model with parameters θ. Crucially, the effects of independent sampling are amortized by annealing the probability with which variables are masked out. Whereas Yao et al. (2014) treat their procedure as a cheap approximation to ancestral sampling, we find that it produces superior samples. Intuitively, the resampling process allows the model to iteratively *rewrite* the score, giving it the opportunity to correct its own mistakes.

COCONET addresses the general task of completing partial scores; special cases of this task include "bridging" two musical fragments, and temporal upsampling and extrapolation. Figure 3.1 shows an example of COCONET populating a partial piano roll using blocked-Gibbs sampling. Code and samples are publically available.[1] Our samples on a variety of generative tasks such as rewriting, melodic harmonization and unconditioned polyphonic music generation show the versatility of our model. In this work we focus on Bach chorales, and assume four voices are active at all times. However, our model can be easily adapted to the more general, arbitrarily polyphonic representation as used by Boulanger-Lewandowski et al. (2012).

Section 3.2 discusses related work in modeling music composition, with a focus on counterpoint. The details of our model and training procedure are laid out in Section 3.3. We discuss evaluation under the model in Section 3.4, and sampling from the model in Section 3.5. Results of quantitative and qualitative evaluations are reported in Section 3.6.

[1] Code: https://github.com/czhuang/coconet
Data: https://github.com/czhuang/JSB-Chorales-dataset
Samples: https://coconets.github.io/

3.2. Related Work

Computers have been used since their early days for experiments in music composition. A notable composition is Hiller and Isaacson's string quartet Illiac Suite (Hiller Jr and Isaacson, 1957), which experiments with statistical sequence models such as Markov chains. One challenge in adapting such models is that music consists of multiple interdependent streams of events. Compare this to typical sequence domains such as speech and language, which involve modeling a single stream of events: a single speaker or a single stream of words. In music, extensive theories in counterpoint have been developed to address the challenge of composing multiple streams of notes that coordinate. One notable theory is due to Fux (Fux, 1965) from the Baroque period, which introduces *species counterpoint* as a pedagogical scheme to gradually introduce students to the complexity of counterpoint. In first species counterpoint only one note is composed against every note in a given fixed melody (*cantus firmus*), with all notes bearing equal durations and the resulting vertical intervals consisting of only consonances.

Computer music researchers have taken inspiration from this pedagogical scheme by first teaching computers to write species counterpoint as opposed to full-fledged counterpoint. Farbood and Schoner (2001) use Markov chains to capture transition probabilities of different melodic and harmonic transitions rules. Herremans and Sörensen (2012; 2013) take an optimization approach by writing down an objective function that consists of existing rules of counterpoint and using a variable neighbourhood search to optimize it.

J.S. Bach chorales has been the main corpus in computer music that serves as a starting point to tackle full-fledged counterpoint. A wide range of approaches have been used to generate music in the style of Bach chorales, for example rule-based and instance-based approaches such the recombinancy method from Cope (1991). This method involves first segmenting existing Bach chorales into smaller chunks based on music theory, analyzing their function and stylistic signatures and then re-concatenating the chunks into new coherent works. Other approaches range from constraint-based (Pachet and Roy, 2001) to statistical methods (Conklin, 2003). Fernández and Vico (2013) give a comprehensive survey of AI methods used not just for generating Bach chorales, but also algorithmic composition in general.

Sequence models such as HMMs and RNNs are natural choices for modeling music. Successful application of such models to polyphonic music often requires serializing or otherwise re-representing the music to fit the sequence paradigm. For instance, Liang (BachBot 2016) serializes four-part Bach chorales by interleaving the parts, while Allan and Williams (2005) construct a chord vocabulary. Boulanger-Lewandowski et al. (2012) adopt a piano roll representation, a binary matrix $\mathbf{x}$ where $\mathbf{x}_{it} = 1$ iff some instrument is playing pitch i at time t. To model the joint probability distribution of the multi-hot pitch vector $\mathbf{x}_t$,

they employ a Restricted Boltzmann Machine (RBM; Smolensky, 1986; Hinton et al., 2006) or Neural Autoregressive Distribution Estimator (NADE; Larochelle and Murray, 2011) at each time step. Similarly Goel et al. (2014) employ a Deep Belief Network (Hinton et al., 2006) on top of an RNN.

Hadjeres et al. (2016) instead employ an undirected Markov model to learn pairwise relationships between neighboring notes up to a specified number of steps away in a score. Sampling involves Markov Chain Monte-Carlo (MCMC) using the model as a Metropolis-Hastings (MH) objective. The model permits constraints on the state space to support tasks such as melody harmonization. However, the Markov assumption can limit the expressivity of the model.

DeepBach (Hadjeres and Pachet, 2016) models note predictions by breaking down its full context into three parts, with the past and the future modeled by stacked LSTMs going in the forward and backward directions respectively, and the present harmonic context modeled by a third neural network. The three are then combined by a fourth neural network and used in Gibbs sampling for generation.

Lattner et al. (2016) imposes higher-level structure by interleaving selective Gibbs sampling on a convolutional RBM and gradient descent that minimizes cost to template piece on features such as self-similarity. This procedure itself is wrapped in simulated annealing to ensure steps do not lower the solution quality too much.

We opt for an orderless NADE training procedure which enables us to train a mixture of all possible directed models simultaneously. Finally, an approximate blocked Gibbs sampling procedure from Yao et al. (2014) allows fast generation from the model.

3.3. Model

We employ machine learning techniques to obtain a generative model of musical counterpoint in the form of piano rolls. Given a dataset of observed musical pieces $\mathbf{x}^{(1)} \dots \mathbf{x}^{(n)}$ posited to come from some true distribution $p(\mathbf{x})$, we introduce a model $p_\theta(\mathbf{x})$ with parameters θ. In order to make $p_\theta(\mathbf{x})$ close to $p(\mathbf{x})$, we maximize the data log-likelihood $\sum_i \log p_\theta(\mathbf{x}^{(i)})$ (an approximation of $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \log p_\theta(\mathbf{x})$) by stochastic gradient descent.

The joint distribution $p(\mathbf{x})$ over D variables $\mathbf{x}_1 \dots \mathbf{x}_D$ is often difficult to model directly and hence we construct our model $p_\theta(\mathbf{x})$ from simpler factors. In the NADE (Larochelle and Murray, 2011) framework, the joint $p_\theta(\mathbf{x})$ is factorized autoregressively, one variable at a time, according to some ordering $o = o_1 \dots o_D$, such that

$$p_\theta(\mathbf{x}) = \prod_d p_\theta(\mathbf{x}_{o_d} \mid \mathbf{x}_{o_{<d}}). \tag{3.3.1}$$

For example, it can be factorized in chronological order:

$$p_\theta(\mathbf{x}) = p_\theta(\mathbf{x}_1)p_\theta(\mathbf{x}_2|\mathbf{x}_1)\dots p_\theta(\mathbf{x}_D|\mathbf{x}_{D-1}\dots\mathbf{x}_1) \tag{3.3.2}$$

In general, NADE permits any one fixed ordering, and although all orderings are equivalent from a theoretical perspective, they differ in practice due to effects of optimization and approximation.

Instead, we can train NADE for all orderings o simultaneously using the orderless NADE (Uria et al., 2014) training procedure. This procedure relies on the observation that, thanks to parameter sharing, computing $p_\theta(\mathbf{x}_{o_{d'}} \mid \mathbf{x}_{o_{<d}})$ for all $d' \geq d$ is no more expensive than computing it only for $d' = d$.[2] Hence for a given o and d we can simultaneously obtain partial losses for all orderings that agree with o up to d:

$$\mathcal{L}(\mathbf{x}; o_{<d}, \theta) = -\sum_{o_d} \log p_\theta(\mathbf{x}_{o_d} \mid \mathbf{x}_{o_{<d}}, o_{<d}, o_d) \tag{3.3.3}$$

An orderless NADE model offers direct access to all distributions of the form $p_\theta(\mathbf{x}_i|\mathbf{x}_C)$ conditioned on any set of contextual variables $\mathbf{x}_C = \mathbf{x}_{o_{<d}}$ that might already be known. This gives us a very flexible generative model; in particular, we can use these conditional distributions to complete arbitrarily partial musical scores.

To train the model, we sample a training example $\mathbf{x}$ and context C such that $|C| \sim U(1, D)$, and update θ based on the gradient of the loss given by Equation 3.3.3. This loss consists of $D - d + 1$ terms, each of which corresponds to one ordering. To ensure all orderings are trained evenly we must reweight the gradients by $1/(D - d + 1)$. This correction, due to Uria et al. (2014), ensures consistent estimation of the joint negative log-likelihood $\log p_\theta(\mathbf{x})$.

In this work, the model $p_\theta(x)$ is implemented by a deep convolutional neural network (Krizhevsky et al., 2012). This choice is motivated by the locality of contrapuntal rules and their near-invariance to translation, both in time and in pitch space.

We represent the music as a stack of piano rolls encoded in a binary three-dimensional tensor $\mathbf{x} \in \{0, 1\}^{I \times T \times P}$. Here I denotes the number of instruments, T the number of time steps, P the number of pitches, and $\mathbf{x}_{i,t,p} = 1$ iff the ith instrument plays pitch p at time t. We will assume each instrument plays exactly one pitch at a time, that is, $\sum_p \mathbf{x}_{i,t,p} = 1$ for all i, t.

Our focus is on four-part Bach chorales as used in prior work (Allan and Williams, 2005; Boulanger-Lewandowski et al., 2012; Goel et al., 2014; Liang, 2016; Hadjeres et al., 2016). Hence we assume $I = 4$ throughout. We constrain ourselves to only the range that appears in our training data (MIDI pitches 36 through 88). Time is discretized at the level

[2] Here $\mathbf{x}_{o_{<d}}$ is used as shorthand for variables $\mathbf{x}_{o_1} \dots \mathbf{x}_{o_{d-1}}$ that occur earlier in the ordering.

of 16th notes for similar reasons. To curb memory requirements, we enforce $T = 128$ by randomly cropping the training examples.

Given a training example $\mathbf{x} \sim p(\mathbf{x})$, we present the model with values of only a strict subset of its elements $\mathbf{x}_C = \{\mathbf{x}_{(i,t)} \mid (i,t) \in C\}$ and ask it to reconstruct its complement $\mathbf{x}_{\neg C}$. The input $\mathbf{h}^0 \in \{0,1\}^{2I \times T \times P}$ is obtained by masking the piano rolls $\mathbf{x}$ to obtain the context $\mathbf{x}_C$ and concatenating this with the corresponding mask:

$$\mathbf{h}^0_{i,t,p} = \mathbb{1}_{(i,t) \in C}\, \mathbf{x}_{i,t,p} \tag{3.3.4}$$

$$\mathbf{h}^0_{I+i,t,p} = \mathbb{1}_{(i,t) \in C} \tag{3.3.5}$$

where the time and pitch dimensions are treated as spatial dimensions to convolve over. Each instrument's piano roll $\mathbf{h}^0_i$ and mask $\mathbf{h}^{\mathbf{0}}_{\mathbf{I+i}}$ is treated as a separate channel and convolved independently.

With the exception of the first and final layers, all convolutions preserve the size of the hidden representation. That is, we use "same" padding throughout and all activations have the same number of channels H, such that $\mathbf{h}^l \in \mathbb{R}^{H \times T \times P}$ for all $1 < l < L$. Throughout our experiments we used $L = 64$ layers and $H = 128$ channels. After each convolution we apply batch normalization (Ioffe and Szegedy, 2015) (denoted by $\mathrm{BN}(\cdot)$) with statistics tied across time and pitch. Batch normalization rescales activations at each layer to have mean β and standard deviation γ, which greatly improves optimization. After every second convolution, we introduce a skip connection from the hidden state two levels below to reap the benefits of residual learning (He et al., 2016).

$$\mathbf{a}^l = \mathrm{BN}(\mathbf{W}^l * \mathbf{h}^{l-1}; \gamma^l, \beta^l) \tag{3.3.6}$$

$$\mathbf{h}^l = \begin{cases} \mathrm{ReLU}(\mathbf{a}^l + \mathbf{h}^{l-2}) \\ \qquad\qquad \text{if } 3 < l < L-1 \text{ and } l \bmod 2 = 0 \\ \mathrm{ReLU}(\mathbf{a}^l) \text{ otherwise} \end{cases}$$

$$\mathbf{h}^L = \mathbf{a}^L \tag{3.3.7}$$

The final activations $\mathbf{h}^L \in \mathbb{R}^{I \times T \times P}$ are passed through the softmax function to obtain predictions for the pitch at each instrument/time pair:

$$p_\theta(\mathbf{x}_{i,t,p} \mid \mathbf{x}_C, C) = \frac{\exp\left(h^L_{i,t,p}\right)}{\sum_p \exp\left(h^L_{i,t,p}\right)} \tag{3.3.8}$$

Model	Temporal resolution		
	quarter	eighth	sixteenth
NADE (Boulanger-Lewandowski et al.)	7.19		
RNN-RBM (Boulanger-Lewandowski et al.)	6.27		
RNN-NADE (Boulanger-Lewandowski et al.)	5.56		
RNN-NADE (our implementation)	5.03	3.78	2.05
COCONET (chronological)	7.79 ± 0.09	4.21 ± 0.05	2.22 ± 0.03
COCONET (random)	5.03 ± 0.06	1.84 ± 0.02	0.57 ± 0.01

Table 3.1. Framewise negative log-likelihoods (NLLs) on the Bach corpus. We compare against (Boulanger-Lewandowski et al., 2012), who used quarter-note resolution. We also compare on higher temporal resolutions (eighth notes, sixteenth notes), against our own reimplementation of RNN-NADE. COCONET is an instance of orderless NADE, and as such we evaluate it on random orderings. However, the baselines support only chronological frame ordering, and hence we evaluate our model in this setting as well.

The loss function from Equation 3.3.3 is then given by

$$\begin{aligned} \mathcal{L}(\mathbf{x}; C, \theta) &= -\sum_{(i,t)\notin C} \log p_\theta(\mathbf{x}_{i,t} \mid \mathbf{x}_C, C) \\ &= -\sum_{(i,t)\notin C} \sum_p \mathbf{x}_{i,t,p} \log p_\theta(\mathbf{x}_{i,t,p} \mid \mathbf{x}_C, C) \end{aligned} \tag{3.3.9}$$

where p_θ denotes the probability under the model with parameters $\theta = \mathbf{W}^1, \gamma^1, \beta^1, \ldots, \mathbf{W}^{L-1}, \gamma^{L-1}, \beta^{L-1}$. We train the model by minimizing

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim p(\mathbf{x})} \mathop{\mathbb{E}}_{C\sim p(C)} \frac{1}{|\neg C|} \mathcal{L}(\mathbf{x}; C, \theta) \tag{3.3.10}$$

with respect to θ using stochastic gradient descent with step size determined by Adam (Kingma and Ba, 2014). The expectations are estimated by sampling piano rolls $\mathbf{x}$ from the training set and drawing a single context C per sample.

3.4. Evaluation

The log-likelihood of a given example is computed according to Algorithm 1 by repeated application of Equation 3.3.8. Evaluation occurs one frame at a time, within which the model conditions on its own predictions and does not see the ground truth. Unlike notewise teacher-forcing, where the ground truth is injected after each prediction, the framewise evaluation is thus sensitive to accumulation of error. This gives a more representative measure of quality of the generative model. For each example, we repeat the evaluation process a number of times to average over multiple orderings, and finally average across frames and examples. For chronological evaluation, we draw only orderings that have the t_ls in increasing order.

Algorithm 1 Framewise log-likelihood evaluation

Given a piano roll $\mathbf{x}$
$L_{m,i,t} \leftarrow 0$ for all m,i,t
for multiple orderings $m = 0 \dots M$ **do**
  $C \leftarrow \emptyset$, $\widehat{\mathbf{x}} \leftarrow \mathbf{x}$
  Sample an ordering $t_1, t_2 \dots t_T$ over frames
  for $l = 0 \dots T$ **do**
    Sample an ordering $i_1, i_2 \dots i_I$ over instruments
    for $k = 0 \dots I$ **do**
      $\pi_p \leftarrow p_\theta(\mathbf{x}_{i_k,t_l,p} \mid \widehat{\mathbf{x}}_C, C)$ for all p
      $L_{m,i_k,t_l} \leftarrow \sum_p \mathbf{x}_{i_k,t_l,p} \log \pi_p$
      $\widehat{\mathbf{x}}_{i_k,t_l} \sim \mathrm{Cat}(P, \pi)$
      $C \leftarrow C \cup (i_k, t_l)$
    end for
    $\widehat{\mathbf{x}}_C \leftarrow \mathbf{x}_C$
  end for
end for
return $-\frac{1}{T} \sum_t \log \frac{1}{M} \sum_m \exp \sum_i L_{m,i,t}$

3.5. Sampling

We can sample from the model using the orderless NADE ancestral sampling procedure, in which we first sample an ordering and then sample variables one by one according to the ordering. However, we find that this yields poor samples, and we propose instead to use Gibbs sampling.

3.5.1. Orderless NADE Sampling

Sampling according to orderless NADE involves first randomly choosing an ordering and then sampling variables one by one according to the chosen ordering. We use an equivalent procedure in which we arrive at a random ordering by at each step randomly choosing the next variable to sample. We start with an empty (zero everywhere) piano roll $\mathbf{x}^0$ and empty context C^0 and populate them iteratively by the following process. We feed the piano roll $\mathbf{x}^s$ and context C^s into the model to obtain a set of categorical distributions $p_\theta(\mathbf{x}_{i,t} | \mathbf{x}^s_{C^s}, C^s)$ for $(i,t) \notin C^s$. As the $\mathbf{x}_{i,t}$ are not conditionally independent, we cannot simply sample from these distributions independently. However, if we sample from one of them, we can compute new conditional distributions for the others. Hence we randomly choose one $(i,t)^{s+1} \notin C^s$ to sample from, and let $\mathbf{x}^{s+1}_{i,t}$ equal the one-hot realization. Augment the context with $C^{s+1} = C^s \cup (i,t)^{s+1}$ and repeat until the piano roll is populated. This procedure is easily generalized to tasks such as melody harmonization and partial score completion by starting with a nonempty piano roll.

Unfortunately, samples thus generated are of low quality, which we surmise is due to accumulation of errors. This is a well-known weakness of autoregressive models (Venkataraman et al., 2015; Bengio et al., 2015; Huszár, 2015; Lamb et al., 2016). While the model provides conditionals $p_\theta(\mathbf{x}_{i,t}|\mathbf{x}_C, C)$ for all $(i,t) \notin C$, some of these conditionals may be better modeled than others. We suspect in particular those conditionals used early on in the procedure, for which the context C consists of very few variables. Moreover, although the model is trained to be order-agnostic, different orderings invoke different distributions, which is another indication that some conditionals are poorly learned. We test this hypothesis in Section 3.6.2.

3.5.2. Gibbs Sampling

To remedy this, we allow the model to revisit its choices: we repeatedly mask out some part of the piano roll and then repopulate it. This is a form of blocked Gibbs sampling (Liu, 1994). Blocked sampling is crucial for mixing, as the high temporal resolution of our representation causes strong correlations between consecutive notes. For instance, without blocked sampling, it would take many steps to snap out of a long-held note. Similar considerations hold for the Ising model from statistical mechanics, leading to the Swendsen-Wang algorithm (Swendsen and Wang, 1987) in which large clusters of variables are resampled at once.

We consider two strategies for resampling a given block of variables: *ancestral* sampling and *independent* sampling. Ancestral sampling invokes the orderless NADE sampling procedure described in Section 3.5.1 on the masked-out portion of the piano roll. Independent sampling simply treats the masked-out variables $\mathbf{x}_{\neg C}$ as independent given the context $\mathbf{x}_C$.

Using independent blocked Gibbs to sample from a NADE model has been studied by Yao et al. (2014), who propose to use an annealed masking probability $\alpha_n = \max(\alpha_{\min}, \alpha_{\max} - n(\alpha_{\max} - \alpha_{\min})/(\eta N))$ for some minimum and maximum probabilities $\alpha_{\min}, \alpha_{\max}$, total number of Gibbs steps N and fraction η of time spent before settling onto the minimum probability $\alpha_{\min}$. Initially, when the masking probability is high, the chain mixes fast but samples are poor due to independent sampling. As α_n decreases, the blocked Gibbs process with independent resampling approaches standard Gibbs where one variable at a time is resampled, thus amortizing the effects of independent sampling. N is a hyperparameter which as a rule of thumb we set equal to IT; it can be set lower than that to save computation at a slight loss of sample quality.

Yao et al. (2014) treat independent blocked Gibbs as a cheap approximation to ancestral sampling. Whereas plain ancestral sampling (3.5.1) requires $O(IT)$ model evaluations, ancestral blocked Gibbs requires a prohibitive $O(ITN)$ model evaluations and independent Gibbs requires only $O(N)$, where N can be chosen to be less than IT. Moreover, we find

that independent blocked Gibbs sampling in fact yields *better* samples than plain ancestral sampling.

3.6. Experiments

We evaluate our approach on a popular corpus of four-part Bach chorales. While the literature features many variants of this dataset (Allan and Williams, 2005; Boulanger-Lewandowski et al., 2012; Liang, 2016; Hadjeres et al., 2016), we report results on that used by Boulanger-Lewandowski et al. (2012). As the quarter-note temporal resolution used by Boulanger-Lewandowski et al. (2012) is frankly too coarse to accurately convey counterpoint, we also evaluate on eighth-note and sixteenth-note quantizations of the same data.

It should be noted that quantitative evaluation of generative models is fundamentally hard (Theis et al., 2015). The gold standard for evaluation is qualitative comparison by humans, and we therefore report human evaluation results as well.

3.6.1. Data Log-likelihood

Table 3.1 compares the framewise log-likelihood of the test data under variants of our model and those reported in Boulanger-Lewandowski et al. (2012). We find that the temporal resolution has a dramatic influence on the performance, which we suspect is an artifact of the performance metric. The log-likelihood is evaluated by teacher-forcing, that is, the prediction of a frame is conditioned on the ground truth of all previously predicted frames. As temporal resolution increases, chord changes become increasingly rare, and the model is increasingly rewarded for simply holding notes over time.

We evaluate COCONET on both chronological and random orderings, in both cases averaging likelihoods across an ensemble of $M = 5$ orderings. The chronological orderings differ only in the ordering of instruments within each frame. We see in Table 3.1 that fully random orderings lead to significantly better performance. We believe the members of the more diverse random ensemble are more mutually complementary. For example, a forward ordering is uncertain at the beginning of a piece and more certain toward the end, whereas a backward ordering is more certain at the beginning and less certain toward the end.

3.6.2. Sample Quality

In Section 3.5 we conjectured that the low quality of NADE samples is due to poorly modeled conditionals $p_\theta(\mathbf{x}_{i,t} \mid \mathbf{x}_C, C)$ where C is small. We test this hypothesis by evaluating the likelihood under the model of samples generated by the ancestral blocked Gibbs procedure with C chosen according to independent Bernoulli variables. When we set the

Sampling scheme	Framewise NLL
Ancestral Gibbs, $\rho = 0.00$ (NADE)	1.09 ± 0.06
Ancestral Gibbs, $\rho = 0.05$	1.08 ± 0.06
Ancestral Gibbs, $\rho = 0.10$	0.97 ± 0.05
Ancestral Gibbs, $\rho = 0.25$	0.80 ± 0.04
Ancestral Gibbs, $\rho = 0.50$	0.74 ± 0.04
Independent Gibbs (Yao et al., 2014)	0.52 ± 0.01

Table 3.2. Mean ($\pm$ SEM) NLL under model of unconditioned samples generated from model by various schemes.

inclusion probability ρ to 0, we obtain NADE. Increasing ρ increases the expected context size $|C|$, which should yield better samples if our hypothesis is true. The results shown in Table 3.2 confirm that this is the case. For these experiments, we used sample length $T = 32$ time steps and number of Gibbs steps $N = 100$.

Figure 3.2 shows the convergence behavior of the various Gibbs procedures, averaged over 100 runs. We see that for low values of ρ (small C), the chains hardly make progress beyond NADE in terms of likelihood. Higher values of ρ (large C) enable the model to get off the ground and reach significantly better likelihood.

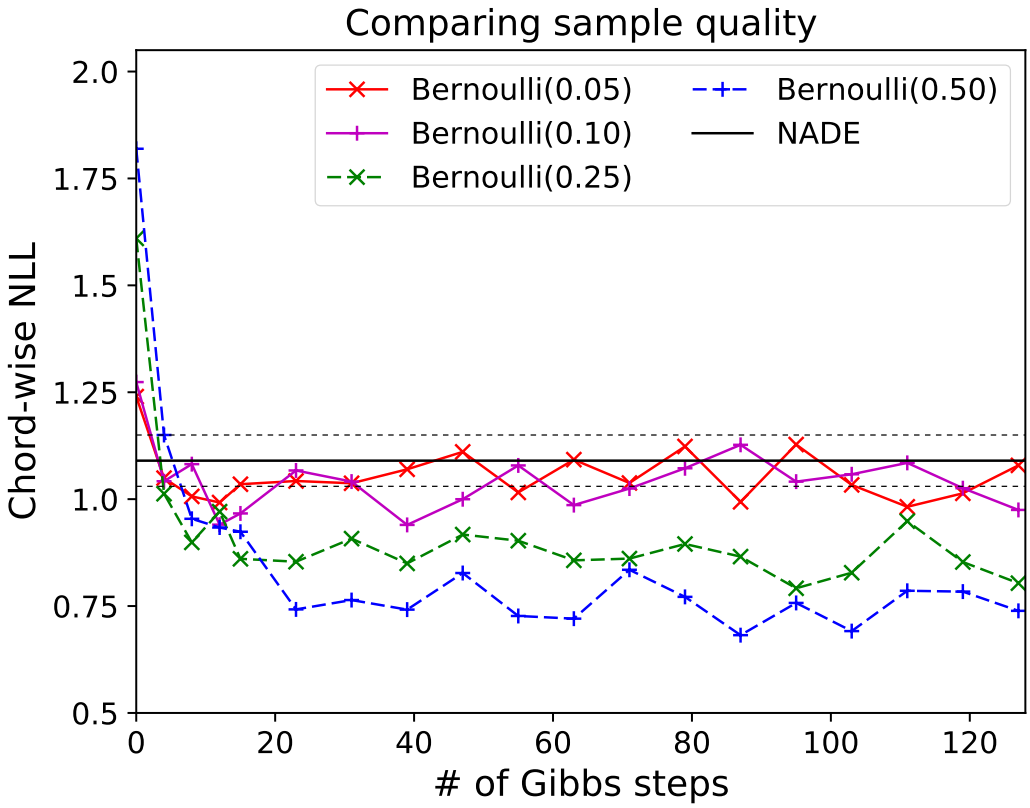


Fig. 3.2. Likelihood under the model for ancestral Gibbs samples obtained with various context distributions $p(C)$. NADE (Bernoulli(0.00)) is included for reference.

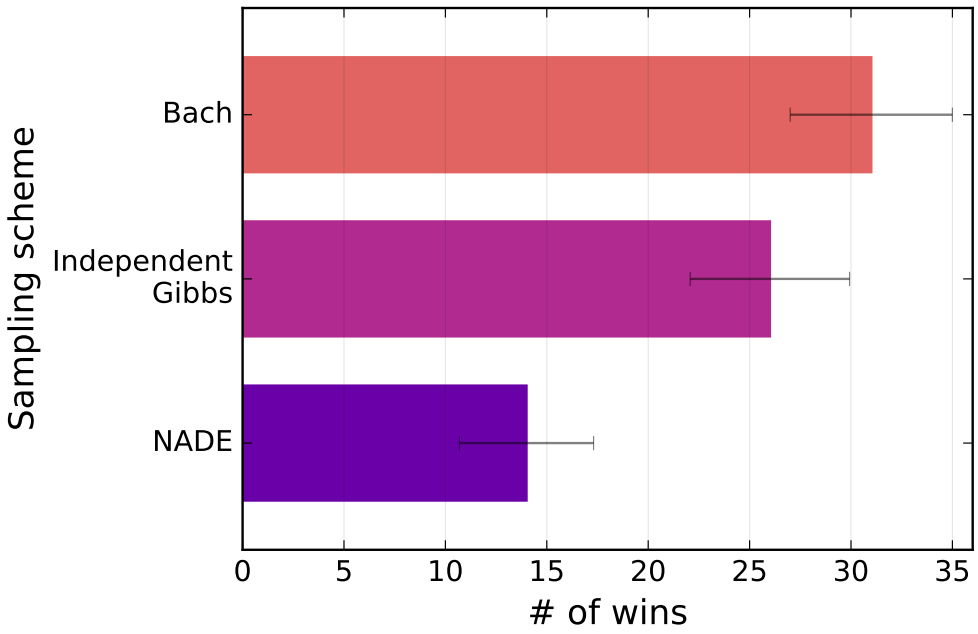


Fig. 3.3. Human evaluations from MTurk on how many times a sampling procedure or Bach is perceived as more Bach-like. Error bars show the standard deviation of a binomial distribution fitted to each's binary win/loss counts.

3.6.3. Human Evaluations

To further compare the sample quality of different sampling procedures, we carried out a listening test on Amazon's Mechanical Turk (MTurk). The procedures include orderless NADE ancestral sampling and independent Gibbs (Yao et al., 2014), with each we generate four unconditioned samples of eight-measure lengths from empty piano rolls. To have an absolute reference for the quality of samples, we include first eight measures of four random Bach chorale pieces from the validation set. Each fragment lasts thirty-four seconds after synthesis.

For each MTurk hit, participants are asked to rate on a Likert scale which of the two random samples they perceive as more Bach-like. A total of 96 ratings were collected, with each source involved in 64 (=96*2/3) pairwise comparisons. Figure 3.3 shows the number of times each source was perceived as closer to Bach's style. We perform a Kruskal-Wallis H test on the ratings, $\chi^2(2) = 12.23, p < 0.001$, showing there are statistically significant differences between models. A post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction showed that participants perceived samples from independent Gibbs as more Bach-like than ancestral sampling (NADE), $p < 0.05/3$. This confirms the loglikelihood comparisons on sample quality in 3.6.2 that independent Gibbs produces better samples. There was also a significance difference between Bach and ancestral samples but not between Bach and independent Gibbs.

3.7. Conclusion

We introduced a convolutional approach to modeling musical scores based on the orderless NADE (Uria et al., 2016) framework. Our experiments show that the NADE ancestral sampling procedure yields poor samples, which we have argued is because some conditionals are not captured well by the model. We have shown that sample quality improves significantly when we use blocked Gibbs sampling to iteratively rewrite parts of the score. Moreover, annealed independent blocked Gibbs sampling as proposed by Yao et al. (2014) is not only faster but in fact produces better samples.

Acknowledgments

We thank Kyle Kastner, Guillaume Alain, Gabriel Huang, Curtis (Fjord) Hawthorne, the Google Brain Magenta team, as well as Jason Freidenfelds for helpful feedback, discussions, suggestions and support. We also thank Calcul Québec and Compute Canada for computational support.

References

Allan, Moray and Christopher KI Williams (2005). "Harmonising chorales by probabilistic inference". In: *Advances in neural information processing systems* 17, pp. 25–32.

Baum, Leonard E and Ted Petrie (1966). "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6, pp. 1554–1563.

Bengio, Samy et al. (2015). "Scheduled sampling for sequence prediction with recurrent neural networks". In: *Advances in Neural Information Processing Systems*, pp. 1171–1179.

Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent (2012). "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription". In: *International Conference on Machine Learning*.

Conklin, Darrell (2003). "Music generation from statistical models". In: *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*. Citeseer, pp. 30–35.

Cope, David (1991). "Computers and musical style". In.

Farbood, Mary and Bernd Schoner (2001). "Analysis and synthesis of Palestrina-style counterpoint using Markov chains". In: *Proceedings of the International Computer Music Conference*, pp. 471–474.

Fernández, Jose D and Francisco Vico (2013). "AI methods in algorithmic composition: A comprehensive survey". In: *Journal of Artificial Intelligence Research* 48, pp. 513–582.

Fux, Johann Joseph (1965). *The study of counterpoint from Johann Joseph Fux's Gradus ad Parnassum*. 277. WW Norton & Company.

Gatys, Leon A, Alexander S Ecker, and Matthias Bethge (2015). "A neural algorithm of artistic style". In: *arXiv preprint arXiv:1508.06576*.

Goel, Kratarth, Raunaq Vohra, and JK Sahoo (2014). "Polyphonic music generation by modeling temporal dependencies using a RNN-DBN". In: *International Conference on Artificial Neural Networks*. Springer, pp. 217–224.

Goodfellow, Ian et al. (2013). "Multi-prediction deep Boltzmann machines". In: *Advances in Neural Information Processing Systems*, pp. 548–556.

Hadjeres, Gaëtan and François Pachet (2016). "DeepBach: a Steerable Model for Bach chorales generation". In: *arXiv preprint arXiv:1612.01010*.

Hadjeres, Gaëtan, Jason Sakellariou, and François Pachet (2016). "Style Imitation and Chord Invention in Polyphonic Music with Exponential Families". In: *arXiv preprint arXiv:1609.05152*.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Herremans, Dorien and Kenneth Sörensen (2012). "Composing first species counterpoint with a variable neighbourhood search algorithm". In: *Journal of Mathematics and the Arts* 6.4, pp. 169–189.

Herremans, Dorien and Kenneth Sörensen (2013). "Composing fifth species counterpoint music with a variable neighborhood search algorithm". In: *Expert systems with applications* 40.16, pp. 6427–6437.

Hiller Jr, Lejaren A and Leonard M Isaacson (1957). "Musical composition with a high speed digital computer". In: *Audio Engineering Society Convention 9*. Audio Engineering Society.

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh (2006). "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7, pp. 1527–1554.

Huszár, Ferenc (2015). "How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?" In: *arXiv preprint arXiv:1511.05101*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167*.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Lamb, Alex M et al. (2016). "Professor forcing: A new algorithm for training recurrent networks". In: *Advances In Neural Information Processing Systems*, pp. 4601–4609.

Larochelle, Hugo and Iain Murray (2011). "The Neural Autoregressive Distribution Estimator." In: *AISTATS*. Vol. 1, p. 2.

Lattner, Stefan, Maarten Grachten, and Gerhard Widmer (2016). "Imposing higher-level Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints". In: *arXiv preprint arXiv:1612.04742*.

Liang, Feynman (2016). "BachBot: Automatic composition in the style of Bach chorales". In: *Masters thesis, University of Cambridge*.

Liu, Jun S (1994). "The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem". In: *Journal of the American Statistical Association* 89.427, pp. 958–966.

Mordvintsev, Alexander, Christopher Olah, and Mike Tyka (2015). *Inceptionism: Going Deeper into Neural Networks*. URL: `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html` (visited on 11/04/2016).

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016a). "Pixel Recurrent Neural Networks". In: *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1747–1756.

Oord, Aaron van den et al. (2016b). "WaveNet: A Generative Model for Raw Audio". In: *arXiv preprint arXiv:1609.03499*.

Pachet, François and Pierre Roy (2001). "Musical harmonization with constraints: A survey". In: *Constraints* 6.1, pp. 7–19.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1988). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3, p. 1.

Smolensky, Paul (1986). *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. DTIC Document.

Swendsen, Robert H and Jian-Sheng Wang (1987). "Nonuniversal critical dynamics in Monte Carlo simulations". In: *Physical review letters* 58.2, p. 86.

Theis, Lucas, Aäron van den Oord, and Matthias Bethge (2015). "A note on the evaluation of generative models". In: *arXiv preprint arXiv:1511.01844*.

Uria, Benigno, Iain Murray, and Hugo Larochelle (2014). "A Deep and Tractable Density Estimator." In: *ICML*, pp. 467–475.

Uria, Benigno et al. (2016). "Neural Autoregressive Distribution Estimation". In: *arXiv preprint arXiv:1605.02226*.

Venkatraman, Arun, Martial Hebert, and J Andrew Bagnell (2015). "Improving Multi-Step Prediction of Learned Time Series Models." In: *AAAI*, pp. 3024–3030.

Yao, Li et al. (2014). "On the equivalence between deep nade and generative stochastic networks". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 322–336.

Chapter 4

On the Variance of Unbiased Online Recurrent Optimization

Tim Cooijmans*† James Martens†

* Mila & Université de Montréal † DeepMind

The recently proposed Unbiased Online Recurrent Optimization (UORO) algorithm (Tallec and Ollivier, 2018) uses an unbiased approximation of RTRL to achieve fully online gradient-based learning in RNNs. In this work we analyze the variance of the gradient estimate computed by UORO, and propose several possible changes to the method which reduce this variance both in theory and practice. We also contribute significantly to the theoretical and intuitive understanding of UORO (and its existing variance reduction technique), and demonstrate a fundamental connection between its gradient estimate and the one that would be computed by REINFORCE if small amounts of noise were added to the RNN's hidden units.

Prologue

This work was done as part of my internship at DeepMind in 2017. I worked with James Martens, who has worked extensively on optimization and automatic differentiation. We had a common interest in recurrent neural networks (RNNs) in particular, and both felt that the dominant approach, backpropagation through time (BPTT), was awkward. BPTT computes gradients in reverse order, which means the RNN must pause every so many steps to work backwards. Real-Time Recurrent Learning (RTRL), which computes gradients in forward order, has a much more suitable computational structure, but scales poorly as the number of parameters grows. We set out to find an efficient approximation to RTRL.

I initially developed a stochastic approximation to RTRL based on random linear compression and decompression of the Jacobian carry. James later showed that we could skip several compression-decompression steps without biasing the estimator, at which point

we obtained the UORO algorithm by Tallec and Ollivier (2018). From there we studied the algorithm's sources of variance, and ways to reduce them. The pragmatic design choices made in our variance reduction scheme are due to James. I discovered PREUORO, which drastically reduces variance simply by choosing to put the random projections elsewhere along the computational graph. I also formalized the link to REINFORCE, although I relied on James to show what happens in the limit as the noise vanishes. All implementations and experiments were my work. James and I contributed about equally to writing the paper.

4.1. Introduction

All learning algorithms are driven by some form of credit assignment—identification of the causal effect of past actions on a learning signal (Minsky, 1961; Sutton, 1984). This enables agents to learn from experience by amplifying behaviors that lead to success, and attenuating behaviors that lead to failure. The problem of performing efficient and precise credit assignment, especially in temporal agents, is a central one in artificial intelligence.

Knowledge of the inner workings of the agent can simplify the problem considerably, as we can trace responsibility for the agent's decisions back to its parameters. In this work, we consider credit assignment in recurrent neural networks (RNNs; Elman, 1990; Hochreiter and Schmidhuber, 1997), where the differentiability of the learning signal with respect to past hidden units allows us to assign credit using derivatives. But even with this structure, *online* credit assignment across long or indefinite stretches of time remains a largely unsolved problem.

Typically, differentiation occurs by Backpropagation Through Time (BPTT; Rumelhart et al., 1986; Werbos, 1990), which requires a "forward pass" in which the network is evaluated for a length of time, followed by a "backwards pass" in which gradient with respect to the model's parameters is computed. This is impractical for very long sequences, and a common trick is to "truncate" the backwards pass after some fixed number of iterations (Williams and Peng, 1990). As a consequence, parameter updates are infrequent, expensive, and limited in the range of temporal dependencies they reflect.

BPTT's more natural dual, Real-Time Recurrent Learning (RTRL; Williams and Zipser, 1989), carries gradient information forward rather than backward. It runs alongside the model and provides parameters updates at every time step. To do so, however, it must retain a large matrix relating the model's internal state to its parameters. Even when this matrix can be stored at all, updating it is prohibitively expensive. Various approximations to RTRL have been proposed (e.g. Mak et al., 1999) in order to obtain cheaper gradient estimates at the cost of reducing their accuracy.

In this paper we consider Unbiased Online Recurrent Optimization (UORO; Ollivier et al., 2015; Tallec and Ollivier, 2018), an unbiased stochastic approximation to RTRL that compresses the gradient information through random projections. We analyze the variance of the UORO gradient estimator, relate it to other gradient estimators, and propose various modifications to it that reduce its variance both in theory and practice.

4.2. Outline of the Paper

We begin with a detailed discussion of the relationship and tradeoffs between RTRL and BPTT in Section 4.3. Before narrowing our focus to approximations to RTRL, we briefly review other approaches to online credit assignment in Section 4.4. We then contribute a novel and arguably more intuitive derivation of the UORO algorithm in Section 4.5.

In Sections 4.6 and 4.7 we give our main contribution in the form of a thorough analysis of UORO and the variance it incurs., and derive a new variance reduction method based on this analysis. Sections 4.6.1 and 4.6.2 discuss limitations of the variance reduction scheme of Tallec and Ollivier (2018), and in Section 4.6.3 propose to augment its scalar coefficients with matrix-valued transformations. We develop a framework for analysis of UORO-style estimators in Sections 4.6.4 and 4.6.5, which allows us to determine the total variance incurred when accumulating consecutive gradient estimates over time. Working within this framework, we derive a formula for matrices that gives the optimal variance reduction subject to certain structural constraints (Section 4.7.1). We evaluate our theory in a tractable empirical setting in Section 4.7.2, and explore avenues toward a practical algorithm in Section 4.7.1.

Section 4.8 introduces a variant of UORO that avoids one of its two levels of approximation. It exploits the fact that gradients with respect to weight matrices are naturally rank-one. We show this reduces the variance by a factor on the order of the number of hidden units, at the cost of increasing computation time by the same factor.

Finally, we study the relationship between UORO and REINFORCE (Williams, 1992) in Section 4.9. The analysis uncovers a close connection when REINFORCE is used to train RNNs with perturbed hidden states. We show that when this noise is annealed, the REINFORCE estimator converges to the UORO estimator plus an additional term that has expectation zero but unbounded variance.

4.3. Automatic Differentiation in Recurrent Neural Networks

Recurrent Neural Networks (RNNs; Elman, 1990; Hochreiter and Schmidhuber, 1997) are a general class of nonlinear sequence models endowed with memory. Given a sequence

of input vectors x_t, and initial state vector h_0, an RNN's state evolves according to

$$h_t = F(h_{t-1}, x_t; \theta_t)$$

where F is an arbitrary continuously differentiable transition function parameterized by θ_t that produces the next state h_t given the previous state h_{t-1} and the current observation x_t. Typically, F will take the form of an affine map followed by a nonlinear function:

$$\begin{aligned} a_t &= \begin{pmatrix} h_{t-1}^\top & x_t^\top & 1 \end{pmatrix}^\top \\ h_t &= f(W_t a_t). \end{aligned} \tag{4.3.1}$$

Here $f(\cdot)$ is the "activation function", which is assumed to be continuously differentiable (and is typically nonlinear and coordinate-wise), and W_t is a square matrix parameter whose vectorization is θ_t.

The defining feature of recurrent neural networks as compared to feed-forward neural networks is the fact that their weights are tied over time. That is, we have $\theta_t = \theta$. However, we will continue to distinguish the different θ_t's in the recurrence, as this allows us to refer to individual "applications" of θ in the analysis (which will be useful later).

Although we will treat the sequence as finite, i.e. $1 \leqslant t \leqslant T$ for some sequence length T, we are interested mainly in *streaming* tasks for which T may as well be infinite.

At each time step t, we incur a loss L_t which is some differentiable function of h_t. In order to minimize the aggregate loss $L = \sum_{t=1}^{T} L_t$ with respect to θ, we require an estimate of its gradient with respect to θ. We will write $\mathcal{J}_x^y$ (or occasionally $\mathcal{J}_x(y)$) for the Jacobian of y with respect to x. We can express the gradient as a double sum over time that factorizes in two interesting ways:

$$\mathcal{J}_\theta^L = \sum_{t=1}^{T}\sum_{s=1}^{T} \mathcal{J}_{\theta_s}^{L_t} = \underbrace{\sum_{s=1}^{T}\left(\sum_{t=s}^{T} \mathcal{J}_{h_s}^{L_t}\right)\mathcal{J}_{\theta_s}^{h_s}}_{\text{reverse accumulation}} = \underbrace{\sum_{t=1}^{T} \mathcal{J}_{h_t}^{L_t}\left(\sum_{s=1}^{t} \mathcal{J}_{\theta_s}^{h_t}\right)}_{\text{forward accumulation}} \tag{4.3.2}$$

Each of the terms $\mathcal{J}_{\theta_s}^{L_t}$ indicates how the use of the parameter θ at time s affected the loss at time t. This is a double sum over time with $\mathcal{O}(T^2)$ terms, but since future parameter applications do not affect past losses, we have $\mathcal{J}_{\theta_s}^{L_t} = 0$ for $s > t$. Both factorizations exploit this triangular structure and allow the gradient to be computed in $\mathcal{O}(T)$ by recursive accumulation.

By far the most popular strategy for breaking down this computation goes by the name of Back-Propagation Through Time (BPTT; Werbos, 1990). It is an instance of what is known as *reverse-mode accumulation* in the autodifferentiation community, and relies on the reverse factorization in Equation 4.3.2. BPTT computes gradients of total future loss $\mathcal{J}_{h_t}^L$ with respect to states h_t in reverse chronological order by the recursion

$$\mathcal{J}_{h_t}^L = \mathcal{J}_{h_{t+1}}^L \mathcal{J}_{h_t}^{h_{t+1}} + \mathcal{J}_{h_t}^{L_t}. \tag{4.3.3}$$

At each step, a term $\mathcal{J}_{\theta_t}^{L} = \mathcal{J}_{h_t}^{L} \mathcal{J}_{\theta_t}^{h_t}$ of the gradient is accumulated.

Since the quantities $\mathcal{J}_{h_t}^{h_{t+1}}$, $\mathcal{J}_{h_t}^{L_t}$ and $\mathcal{J}_{\theta_t}^{h_t}$ generally depend on h_t and L_t, the use of BPTT in practice implies running the model forward for T steps to obtain the sequence of hidden states h_t and losses L_t, and subsequently running backward to compute the gradient.

Its converse, Real-Time Recurrent Learning (RTRL; Williams and Zipser, 1989), is an instance of *forward-mode accumulation*. It exploits the forward factorization of the gradient in Equation 4.3.2, computing Jacobians $\mathcal{J}_{\theta}^{h_t}$ of hidden states h_t with respect to past applications of the parameter θ recursively according to

$$\mathcal{J}_{\theta}^{h_t} = \mathcal{J}_{h_{t-1}}^{h_t} \mathcal{J}_{\theta}^{h_{t-1}} + \mathcal{J}_{\theta_t}^{h_t}. \tag{4.3.4}$$

What RTRL provides over BPTT is that we can run it forward alongside our model, and at each time-step t update the model parameters θ immediately (using $\mathcal{J}_{\theta}^{L_t} = \mathcal{J}_{h_t}^{L} \mathcal{J}_{\theta}^{h_t}$), thus performing fully online learning. This is to be contrasted with BPTT, where we must run the model forward for T time-steps before we can make a parameter update, thus introducing a long delay between the reception of a learning signal L_t and the parameter update that takes it into account.

There is a caveat to the above, which is that as soon as we update our parameter θ, the Jacobian $\mathcal{J}_{\theta}^{h_t}$ accumulated by RTRL is no longer quite correct, as it is based on previous values of θ. However, as argued by Williams and Zipser (1995) and Ollivier et al. (2015) this problem can be mostly ignored as long as the learning rate is small enough in relation to the rate of the natural decay of the Jacobian (which occurs due to the vanishing gradient phenomenon).

The main drawback of RTRL is that the accumulated quantity $\mathcal{J}_{\theta}^{h_t}$ is a large matrix. If the size of the parameters θ is $\mathcal{O}(H^2)$ where H is the hidden state size, then this matrix requires $\mathcal{O}(H^3)$ space to store. This is typically much larger than BPTT's $\mathcal{O}(TH)$ space. Moreover, the RTRL recursions involve propagating a matrix forward by the matrix-matrix product $\mathcal{J}_{h_{t-1}}^{h_t} \mathcal{J}_{\theta}^{h_{t-1}}$, which takes $\mathcal{O}(H^4)$ time. BPTT on the other hand only propagates a vector through time at a cost of $\mathcal{O}(H^2)$. Although RTRL frees us to grow T and capture arbitrarily-long-term dependencies, the algorithm is grossly impractical for models of even modest size.

4.4. Other Approaches to Credit Assignment

A number of techniques have been proposed to reduce the memory requirements of BPTT. Storage of past hidden states may be traded for time by recomputing the states on demand, in the extreme case resulting in a quadratic-time algorithm. Better choices for this tradeoff are explored by Chen et al. (2016) and Gruslys et al. (2016). Reversible Recurrent Neural Networks (MacKay et al., 2018; Gomez et al., 2017) allow the on-demand

computation of past states to occur in reverse order, restoring the linear time complexity while limiting the model class. Stochastic Attentive Backtracking (Ke et al., 2018) sidesteps the storage requirements of backprop through long periods of time by retaining only a sparse subset of states in the distant past. This subset is selected based on an attention mechanism that is part of the model being trained. Gradient from future loss is propagated backwards to these states only through the attention connections. Synthetic gradients (Jaderberg et al., 2017) approximates BPTT by use of a predictive model of the total future gradient $\mathcal{J}^L_{h_s}$, which is trained online based on BPTT.

Instead of transporting derivatives through time, we may assign credit by transporting value over time. For example, actor-critic architectures (Konda and Tsitsiklis, 2000; Barto et al., 1983) employ Temporal Difference Learning (Sutton, 1988) to obtain a predictive model of the total future loss. By differentiation, the estimated total future loss may be used to estimate the total future gradient. More commonly, such estimates are used directly as a proxy for the total future loss, or as a REINFORCE baseline. Along similar lines as our analysis of REINFORCE in Section 4.9, we may interpret these methods as effectively differentiating the estimate in expectation. RUDDER (Arjona-Medina et al., 2018) redistributes the total loss L over time, replacing the immediate losses L_s by surrogates L'_s determined by a process similar to backpropagation through a critic. These surrogates preserve the total loss but in an RL setting may better reflect the long-term impact of the action taken at time s. Temporal Value Transport (Hung et al., 2018) relies on attention weights to determine which past time steps were relevant to which future time steps, and injects the estimated total future loss from the future time steps into the immediate loss for the associated past time steps.

4.5. Unbiased Online Recurrent Optimization

The recently proposed Unbiased Online Recurrent Optimization algorithm (UORO; Tallec and Ollivier, 2018) and its predecessor NoBackTrack (Ollivier et al., 2015) approximate RTRL by maintaining a rank-one estimate $\tilde{h}_t \tilde{w}_t^\top$ of the Jacobian $\mathcal{J}^{h_t}_\theta$. We now briefly derive the basic algorithm.

4.5.1. Derivation

First, we note that $\mathcal{J}^{h_t}_\theta$ can be written as $\mathcal{J}^{h_t}_\theta = \sum_{s \leqslant t} \mathcal{J}^{h_t}_{h_s} \mathcal{J}^{h_s}_{\theta_s}$. We then perform a rank-one projection of each term in this sum using a random vector ν_s (which is chosen to satisfy $\mathbb{E}[\nu_s \nu_s^\top] = I$). This gives us the estimator

$$\mathcal{J}^{h_t}_\theta \approx \sum_{s \leqslant t} \mathcal{J}^{h_t}_{h_s} \nu_s \nu_s^\top \mathcal{J}^{h_s}_{\theta_s}.$$

Unbiasedness follows from a simple application of linearity of expectation:

$$\mathbb{E}\Big[\sum_{s\leqslant t} \mathcal{J}_{h_s}^{h_t} \nu_s \nu_s^\top \mathcal{J}_{\theta_s}^{h_s}\Big] = \sum_{s\leqslant t} \mathcal{J}_{h_s}^{h_t} \mathbb{E}[\nu_s \nu_s^\top] \mathcal{J}_{\theta_s}^{h_s} = \sum_{s\leqslant t} \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{\theta_s}^{h_s}.$$

We will refer to this projection as the *spatial projection* to distinguish it from the *temporal projection* that is to follow.

It is interesting to note that $\mathcal{J}_{h_s}^{h_t} \nu_s$ can be interpreted as a "directional Jacobian", which measures the instantaneous change in h_t as a function of h_s's movement along the direction ν_s. Similarly $\nu_s^\top \mathcal{J}_{\theta_s}^{h_s}$ is essentially the gradient of $\nu_s^\top h_s$ with respect to θ_s, and thus measures the instantaneous change of h_s along the direction of ν_s, as a function of the change in θ_s. Thus the intuition behind this first approximation is that we are guessing the relevant direction of change in h_s and performing the gradient computations only along that direction.

We can generalize the spatial projection from the standard UORO method by projecting in the space of any cut vertex z_s on the computational path from θ_s to h_s. For UORO, $z_s \equiv h_s$; other choices include $z_s \equiv \theta_s$ for projection in parameter space, and $z_s \equiv W_s a_s$ for projection in preactivation space. We will make extensive use of this choice in later Sections.

This gives the generalized estimator

$$\mathcal{J}_\theta^{h_t} \approx \sum_{s\leqslant t} \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{z_s}^{h_s} \nu_s \nu_s^\top \mathcal{J}_{\theta_s}^{z_s},$$

which is unbiased following a similar argument as before.

The random projections serve to reduce the large $\mathcal{J}_{\theta_s}^{h_s}$ matrix into the more manageable vector quantities $\mathcal{J}_{z_s}^{h_s} \nu_s$ and $\nu_s^\top \mathcal{J}_{\theta_s}^{z_s}$. But because the sum of rank-one matrices is not itself rank one, the resultant estimator will still be too expensive to maintain and update online.

In order to obtain a practical algorithm we make a second rank-one approximation, now across time instead of z-space. To this end we introduce random scalar coefficients τ_s satisfying $\mathbb{E}[\tau_s \tau_r] = \delta_{sr}$ (where δ_{sr} is the Kronecker delta which is 1 if $s = r$ and 0 otherwise) and define the following rank-one estimator:

$$\mathcal{J}_\theta^{h_t} \approx \tilde{h}_t \tilde{w}_t^\top \triangleq \Big(\sum_{s\leqslant t} \tau_s \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{z_s}^{h_s} \nu_s\Big)\Big(\sum_{s\leqslant t} \tau_s \nu_s^\top \mathcal{J}_{\theta_s}^{z_s}\Big) = \sum_{r\leqslant t}\sum_{s\leqslant t} \tau_s \tau_r \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{z_s}^{h_s} \nu_s \nu_r^\top \mathcal{J}_{\theta_r}^{z_r}.$$

By linearity of expectation this is an unbiased estimate of the previous spatially projected estimator $\sum_{r\leqslant t} \mathcal{J}_{z_s}^{h_t} \nu_s \nu_s^\top \mathcal{J}_{\theta_s}^{z_s}$, and is thus also an unbiased estimator of $\mathcal{J}_\theta^{h_t}$, although with potentially much higher variance.

Going forward we will assume that $\tau_s \sim \mathcal{U}\{-1, +1\}$ are iid random signs and $\nu_s \sim \mathcal{N}(0, I)$ are iid standard normal vectors, so that we may treat the product $\tau_s \nu_s$ as a single Gaussian-distributed random vector $u_s \sim \mathcal{N}(0, I)$, which will simplify our analysis.

The two factors $\tilde{h}_t$ and $\tilde{w}_t$ of the rank-one approximation are maintained by the following pair of recursions:

$$\begin{aligned}\tilde{h}_t &= \gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} + \beta_t \mathcal{J}_{z_t}^{h_t} u_t \\ \tilde{w}_t^\top &= \gamma_t^{-1} \tilde{w}_{t-1}^\top + \beta_t^{-1} u_t^\top \mathcal{J}_{\theta_t}^{z_t},\end{aligned} \tag{4.5.1}$$

with $\tilde{h}_0, \tilde{w}_0$ initialized to zero vectors. Notably these recursions are similar in structure to that used by RTRL to compute the exact Jacobian $\mathcal{J}_\theta^{h_t}$ (c.f. Equation 4.3.4). As with the RTRL equations, their validity follows from the fact that $\mathcal{J}_{h_s}^{h_t} = \mathcal{J}_{h_{t-1}}^{h_t} \mathcal{J}_{h_{t-2}}^{h_{t-1}} \cdots \mathcal{J}_{h_s}^{h_{s+1}}$.

In these recursions we have introduced coefficients γ_t and β_t to implement the variance reduction technique from Tallec and Ollivier (2018) and Ollivier et al. (2015), which we will refer to as *greedy iterative rescaling* (GIR). We will discuss GIR in detail in the next subsection.

Finally, at each step we estimate $\mathcal{J}_\theta^{L_t} = \mathcal{J}_{h_t}^{L_t} \mathcal{J}_\theta^{h_t}$ using the estimator $\mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top$. This is a small deviation from the one given by Tallec and Ollivier (2018), which uses backpropagation to compute $\mathcal{J}_{\theta_t}^{L_t}$ exactly, and the remaining part of the gradient, $\sum_{s<t} \mathcal{J}_{\theta_s}^{L_t}$, is estimated as $\mathcal{J}_{h_{t-1}}^{L_t} \tilde{h}_{t-1} \tilde{w}_{t-1}^\top$. Although our version has slightly higher variance, it is conceptually simpler.

The projected Jacobians that appear in Equation 4.5.1 can be computed efficiently without explicitly handling the full Jacobians. Specifically, $u_t^\top \mathcal{J}_{\theta_t}^{z_t}$ can be computed by reverse-mode differentiating z_t with respect to θ_t, and substituting $u_t^\top$ in place of the adjoint $\mathcal{J}_{z_t}^L$. By a similar trick, one can compute $\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1}$ and $\mathcal{J}_{z_t}^{h_t} u_t$ using forward-mode differentiation. The resulting algorithm has the same $\mathcal{O}(H^2)$ time complexity as backpropagation through time, but its $\mathcal{O}(H^2)$ storage does not grow with time.

4.5.2. Greedy Iterative Rescaling

This subsection explains GIR and the role of the coefficients $\gamma_t, \beta_t > 0$ in Equation 4.5.1.

Whereas our above derivation of the algorithm introduced a temporal projection, Ollivier et al. (2015) and Tallec and Ollivier (2018) interpret the algorithm given by Equation 4.5.1 as implementing a *series* of projections. Under this view, $\tilde{h}_t \tilde{w}_t^\top$ is a rank-one estimate of the rank-two matrix that is the sum of the forwarded previous Jacobian estimate $\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} \tilde{w}_{t-1}^\top$ and the approximate contribution $\mathcal{J}_{z_t}^{h_t} u_t u_t^\top \mathcal{J}_{\theta_t}^{z_t}$:

$$\begin{aligned}\tilde{h}_t \tilde{w}_t^\top &= (\gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} + \beta_t \mathcal{J}_{z_t}^{h_t} u_t)(\gamma_t^{-1} \tilde{w}_{t-1}^\top + \beta_t^{-1} u_t^\top \mathcal{J}_{\theta_t}^{z_t}) \\ &= \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} \tilde{w}_{t-1}^\top + \mathcal{J}_{z_t}^{h_t} u_t u_t^\top \mathcal{J}_{\theta_t}^{z_t} + \tau_t \gamma_t \beta_t^{-1} \boxed{\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} \nu_t^\top \mathcal{J}_{\theta_t}^{z_t}} + \tau_t \beta_t \gamma_t^{-1} \boxed{\mathcal{J}_{z_t}^{h_t} \nu_t \tilde{w}_{t-1}^\top}.\end{aligned}$$

The temporal "cross-terms" $\tau_t \gamma_t \beta_t^{-1} \boxed{\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} \nu_t^\top \mathcal{J}_{\theta_t}^{z_t}}$ and $\tau_t \beta_t \gamma_t^{-1} \boxed{\mathcal{J}_{z_t}^{h_t} \nu_t \tilde{w}_{t-1}^\top}$, which are zero in expectation (but contribute variance), constitute the error introduced in the transition from time $t-1$ to t. The coefficients γ_t and β_t provide an extra degree of freedom with which we can minimize this error. As shown by Ollivier et al. (2015), the minimizers

ensure the terms $\gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1}, \beta_t \mathcal{J}_{z_t}^{h_t} u_t$ and their $\tilde{w}_t$ counterparts have small norm, so that their contribution to the variance is small as well.

The total (trace) variance of $\tilde{h}_t \tilde{w}_t^\top$ with respect to τ_t is given by the expected squared Frobenius norm $\|\cdot\|_F^2$ of the error:

$$\mathbb{E}_{\tau_t}\left[\left\|\tilde{h}_t\tilde{w}_t^\top - \mathbb{E}_{\tau_t}[\tilde{h}_t\tilde{w}_t^\top]\right\|_F^2\right] = \mathbb{E}_{\tau_t}\left[\left\|\tau_t\gamma_t\beta_t^{-1}\boxed{\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}} + \tau_t\beta_t\gamma_t^{-1}\boxed{\mathcal{J}_{z_t}^{h_t}\nu_t\tilde{w}_{t-1}^\top}\right\|_F^2\right].$$

As the common sign τ_t does not affect the norm, this is simply

$$\gamma_t^2\beta_t^{-2}\left\|\boxed{\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}}\right\|_F^2 + \beta_t^2\gamma_t^{-2}\left\|\boxed{\mathcal{J}_{z_t}^{h_t}\nu_t\tilde{w}_{t-1}^\top}\right\|_F^2 + 2\left\langle\boxed{\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}}, \boxed{\mathcal{J}_{z_t}^{h_t}\nu_t\tilde{w}_{t-1}^\top}\right\rangle_F,$$

where $\langle\cdot,\cdot\rangle_F$ denotes the Frobenius inner product.

The coefficients γ_t and β_t affect the error through the single degree of freedom $\gamma_t^2\beta_t^{-2}$. By differentiation and use of the identity $\|xy^\top\|_F^2 = \|x\|^2\|y\|^2$ we find that the optimal choices satisfy

$$\gamma_t^2\beta_t^{-2}\|\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\|^2\|\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}\|^2 = \beta_t^2\gamma_t^{-2}\|\mathcal{J}_{z_t}^{h_t}\nu_t\|^2\|\tilde{w}_{t-1}\|^2.$$

This includes the solution $\gamma_t^2 = \|\tilde{w}_{t-1}\|/\|\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\|, \beta_t^2 = \|\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}\|/\|\mathcal{J}_{z_t}^{h_t}\nu_t\|$ from Ollivier et al. (2015).

Examining their use in Equation 4.5.1 we can see that for this particular solution γ_t plays the important role of contracting $\tilde{w}_t$, which would otherwise grow indefinitely (being a sum of independent random quantities). While division by γ_t in the recursion for $\tilde{h}_t$ causes an expansive effect, this is more than counteracted by the natural contractive property of the Jacobian $\mathcal{J}_{h_{t-1}}^{h_t}$ (which is due to gradient vanishing in well-behaved RNNs). Thus we can interpret the role of γ_t as distributing this contraction evenly between $\tilde{h}_t$ and $\tilde{w}_t$, which limits the growth of both quantities and thus keeps the variance of their product under control. A formal treatment of the growth of variance over time is given by Massé (2017).

4.6. Variance Analysis

In this section we analyze the variance behavior of UORO-style algorithms. We first discuss limitations of the GIR variance reduction scheme discussed in Section 4.5.2, namely that it is greedy (Section 4.6.1) and derives from a somewhat inappropriate objective (Section 4.6.2). We then generalize the algorithm and develop a more holistic theoretical framework for its analysis (Sections 4.6.3 through 4.6.5).

4.6.1. Greedy Iterative Rescaling is Greedy

In Section 4.5.2 we discussed how GIR can be interpreted as minimizing the variance of a rank-one estimate $\tilde{h}_t\tilde{w}_t^\top$ of a rank-two matrix $\mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1}\tilde{w}_{t-1}^\top + \mathcal{J}_{z_t}^{h_t}\nu_t\nu_t^\top\mathcal{J}_{\theta_t}^{z_t}$ (which is a

stochastic approximation that occurs at each step in UORO). Here we unify this sequence of approximations into a single temporal rank-one estimation (as introduced in Section 4.5.1), which helps us reveal the inherent limitations of GIR.

Recall that the UORO recursions (Equation 4.5.1) maintain past contributions in the form of sums $\tilde{h}_t$ and $\tilde{w}_t$, and at each step GIR applies respective scaling factors γ_{t+1} and γ_{t+1}^{-1} (resp.) to these sums. This gives rise to an overall scaling $\alpha_s^{(t)} = \beta_s \gamma_{s+1} \gamma_{s+2} \dots \gamma_t$ (and similarly $(\alpha_s^{(t)})^{-1}$) of contributions made at time step s and propagated forward through time step t. We can write the estimates $\tilde{h}_t \tilde{w}_t^\top$ produced by UORO in terms of $\alpha_s^{(t)}$ as follows:

$$\mathcal{J}_\theta^{h_t} \approx \tilde{h}_t \tilde{w}_t^\top = \Big(\sum_{s \leqslant t} \alpha_s^{(t)} \mathcal{J}_{z_s}^{h_t} u_s\Big)\Big(\sum_{r \leqslant t} \frac{1}{\alpha_r^{(t)}} u_r^\top \mathcal{J}_{\theta_r}^{z_r}\Big) = \sum_{r \leqslant t} \sum_{s \leqslant t} \frac{\alpha_s^{(t)}}{\alpha_r^{(t)}} \tau_s \tau_r \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{z_s}^{h_s} \nu_s \nu_r^\top \mathcal{J}_{\theta_r}^{z_r}.$$

Note that each such estimate is but one element in a *sequence* of estimates. In the next section, we will establish a notion of the variance for this sequence, so that we may speak meaningfully about its minimization. For now, we will consider the minimization of the variance of $\tilde{h}_t \tilde{w}_t^\top$ at each time step t as an independent problem, with independent decision variables $\alpha_s^{(t)}$. The optimal coefficients given by $(\alpha_s^{(t)})^2 = \|\nu_s^\top \mathcal{J}_{\theta_s}^{z_s}\| / \|\mathcal{J}_{z_s}^{h_t} \nu_s\|$ (derived in Appendix 4.B) minimize the variance of $\tilde{h}_t \tilde{w}_t^\top$ with respect to τ_s.

This solution is generally different from that of GIR, which is constrained to have the form $\alpha_s^{(t+1)} = \gamma_{t+1} \alpha_s^{(t)}$ for $s \leqslant t$ (where γ_{t+1} is independent of s). This relationship between $\alpha_s^{(t+1)}$ and $\alpha_s^{(t)}$ breaks the independence of consecutive variance minimization problems, and therefore the resulting coefficients cannot in general be optimal for all t.

We can see this by writing the optimal coefficients $\alpha_s^{(t+1)}$ for $s \leqslant t$ that minimize the variance of $\tilde{h}_{t+1} \tilde{w}_{t+1}^\top$ in terms of the coefficients $\alpha_s^{(t)}$ that minimize the variance of $\tilde{h}_t \tilde{w}_t^\top$:

$$\begin{aligned}(\alpha_s^{(t+1)})^2 &= \frac{\|\nu_s^\top \mathcal{J}_{\theta_s}^{z_s}\|}{\|\mathcal{J}_{z_s}^{h_{t+1}} \nu_s\|} = \frac{\|\nu_s^\top \mathcal{J}_{\theta_s}^{z_s}\|}{\|\mathcal{J}_{z_s}^{h_t} \nu_s\|} \frac{\|\mathcal{J}_{z_s}^{h_t} \nu_s\|}{\|\mathcal{J}_{z_s}^{h_{t+1}} \nu_s\|} \\ &= (\alpha_s^{(t)})^2 \frac{\|\mathcal{J}_{z_s}^{h_t} \nu_s\|}{\|\mathcal{J}_{z_s}^{h_{t+1}} \nu_s\|} = (\alpha_s^{(t)})^2 \left\| \mathcal{J}_{h_t}^{h_{t+1}} \frac{\mathcal{J}_{z_s}^{h_t} \nu_s}{\|\mathcal{J}_{z_s}^{h_t} \nu_s\|} \right\|^{-1}.\end{aligned}$$

We see that in order to minimize the variance of $\tilde{h}_{t+1} \tilde{w}_{t+1}^\top$ given coefficients $\alpha_s^{(t)}$ that minimize the variance of $\tilde{h}_t \tilde{w}_t^\top$, we should *divide* each contribution $\alpha_s^{(t)} \mathcal{J}_{z_s}^{h_t} \nu_s$ by the square root of its contraction due to forward-propagation through $\mathcal{J}_{h_t}^{h_{t+1}}$, and *multiply* each $(\alpha_s^{(t)})^{-1} \nu_s^\top \mathcal{J}_{\theta_s}^{z_s}$ by the same factor. Crucially, this factor depends on s and therefore cannot be expressed by GIR, which is constrained to rescale all past contributions by a constant factor y_{t+1} independent of s. This is true of any algorithm that maintains past contributions in a reduced form such as $\tilde{h}_t, \tilde{w}_t$.

4.6.2. Greedy Iterative Rescaling Optimizes an Inappropriate Objective

In the previous subsection, we saw a sense in which GIR is greedy: its ability to minimize the variance of $\tilde{h}_t\tilde{w}_t^\top$ is hampered by its own past decisions. To see this, we took a holistic view of the sequence of variance minimization problems solved by GIR, and showed that the choice of coefficients γ_s, β_s at time s constrains the choice of future coefficients. Here we take a further step back, and argue that the variance of $\tilde{h}_t\tilde{w}_t^\top$ is not the right objective in light of the downstream application of these estimates.

The Jacobian estimates $\tilde{h}_t\tilde{w}_t^\top \approx \mathcal{J}_\theta^{h_t}$ are used to determine a sequence of *gradient* estimates $\mathcal{J}_{h_t}^{L_t}\tilde{h}_t\tilde{w}_t^\top \approx \mathcal{J}_\theta^{L_t}$, which are accumulated by a gradient descent process. We argue that the quantity of interest is the variance of the **total gradient estimate** $\sum_{t\leqslant T}\mathcal{J}_{h_t}^{L_t}\tilde{h}_t\tilde{w}_t^\top \approx \mathcal{J}_\theta^L$ incurred during T steps of optimization (which estimates the **total gradient** $\mathcal{J}_\theta^L$).

Since consecutive gradient contributions depend largely on the same stochastic quantities, the variance of this sum is not simply the sum of the individual variances. Hence even if we *could* independently minimize the variances of the Jacobian estimates, doing so is not equivalent to minimizing the variance of the total gradient estimate.

4.6.3. Generalized Recursions

Before proceeding with the variance computation we will generalize the UORO recursions by replacing the γ_t and β_t coefficients by an invertible matrix Q_t as follows:

$$\begin{aligned}\tilde{h}_t &= \mathcal{J}_{h_{t-1}}^{h_t}\tilde{h}_{t-1} + \mathcal{J}_{z_t}^{h_t}Q_t u_t \\ \tilde{w}_t^\top &= \tilde{w}_{t-1}^\top + u_t^\top Q_t^{-1}\mathcal{J}_{\theta_t}^{z_t}\end{aligned} \tag{4.6.1}$$

Q_t can be interpreted as modifying the covariance of the noise vector u_t (although differently for either recursion). Analogously to the standard UORO recursions, our generalized recursions compute the following sums:

$$\tilde{h}_t = \sum_{s\leqslant t}\mathcal{J}_{h_s}^{h_t}\mathcal{J}_{z_s}^{h_s}Q_s u_s \qquad \text{and} \qquad \tilde{w}_t = \sum_{s\leqslant t} u_s^\top Q_s^{-1}\mathcal{J}_{\theta_s}^{z_s}.$$

We can view Q_s as a matrix-valued generalization of the GIR coefficients, with equivalence when $Q_s = \beta_s\gamma_{s+1}\gamma_{s+2}\ldots\gamma_T I$. The extra degrees of freedom allow more fine-grained control over the norms of cross-terms,[1] as can be seen when we expand both the temporal *and* the spatial projections in the estimator $\tilde{h}_t\tilde{w}_t^\top$:

$$\mathcal{J}_\theta^{h_t} \approx \tilde{h}_t\tilde{w}_t^\top = \sum_{r\leqslant t}\sum_{s\leqslant t}\sum_{ijkl}\mathcal{J}_{z_{ri}}^{h_t}(Q_r)_{ik}u_{rk}u_{sl}(Q_s^{-1})_{lj}\mathcal{J}_{\theta_s}^{z_{sj}}$$

[1] By "cross-term" we mean a term that appears in the expanded sum which is zero in expectation but contributes variance.

Each term's scaling depends not just on temporal indices r, s but now also on the indices i, j of units. As we shall see, *in expectation*, terms where both the temporal indices $r = s$ and units $i = j$ correspond remain unaffected, and it is only the undesired cross-terms for which $r \neq s$ or $i \neq j$ that are affected.

Tallec and Ollivier (2018) hint at a related approach which would correspond to choosing $Q_s = \alpha_s \operatorname{diag}(q_s)$ to be diagonal matrices. However, they derive their choice $q_{si}^2 \propto \|\mathcal{J}_{\theta_s}^{z_{si}}\| / \|\mathcal{J}_{z_{si}}^{h_s}\|$ by optimizing the norms of only temporally corresponding terms for which $r = s$, and ignoring temporal cross terms $r \neq s$ which make up the bulk of the error. We instead consider a class of Q_s matrices that is not constrained to be diagonal, and whose value minimizes a measure of variance that is more relevant to the optimization process.

Thus our recursion in Equation 4.6.1 is a strict generalization of the UORO recursion in Equation 4.5.1. The Q_s matrices can express a broad class of variance reduction mechanisms, including GIR. That said, our analysis of this system will be limited to cases where the Q_s are independent of the noise vectors u_t for all s, t. Notably, this precludes GIR because of its complex nonlinear interaction with the noise.

4.6.4. A Simple Expression for the Gradient Estimate

In this subsection we will derive a simple expression for the gradient estimate $\mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top$ which will prove useful in our subsequent computations.

To reduce visual clutter we define the following notational aliases, which we will make heavy use of throughout the rest of the manuscript:

$$b_s^{(t)} = \mathcal{J}_{z_s}^{L_t} \qquad \text{and} \qquad J_s = \mathcal{J}_{\theta_s}^{z_s}.$$

First, we observe that that $\mathbb{1}_{s \leqslant t} b_s^{(t)} = b_s^{(t)}$, as derivatives of past losses with respect to future activations are zero. Next we observe that

$$\mathcal{J}_{h_t}^{L_t} \tilde{h}_t = \sum_{s \leqslant t} \mathcal{J}_{h_t}^{L_t} \mathcal{J}_{h_s}^{h_t} \mathcal{J}_{z_s}^{h_s} Q_s u_s = \sum_{s \leqslant t} b_s^{(t)\top} Q_s u_s.$$

Given these observations we may express the estimate of each gradient contribution $\mathcal{J}_\theta^{L_t}$ as

$$\begin{aligned}
\mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top &= \Big(\sum_{s \leqslant t} b_s^{(t)\top} Q_s u_s \Big) \Big(\sum_{s \leqslant t} u_s^\top Q_s^{-1} J_s \Big) \\
&= \Big(\sum_{s \leqslant T} \mathbb{1}_{s \leqslant t} b_s^{(t)\top} Q_s u_s \Big) \Big(\sum_{s \leqslant T} \mathbb{1}_{s \leqslant t} u_s^\top Q_s^{-1} J_s \Big) \\
&= \Big(\sum_{s \leqslant T} b_s^{(t)\top} Q_s u_s \Big) \Big(\sum_{s \leqslant T} \mathbb{1}_{s \leqslant t} u_s^\top Q_s^{-1} J_s \Big) \\
&= b^{(t)\top} Q u u^\top Q^{-1} S^{(t)} J,
\end{aligned}$$

where in the last step we have:

- consolidated the temporal and spatial projections by concatenating the $b_s^{(t)}$ into a single vector $b^{(t)}$, and the noise vectors u_s into a single vector u,
- stacked the J_s's into the matrix J,
- defined Q to be the block-diagonal matrix $\operatorname{diag}(Q_1, Q_2, \ldots, Q_T)$, and
- introduced the "truncated identity matrix" $S^{(t)}$ with diagonal blocks $S_s^{(t)} = \mathbb{1}_{s \leqslant t} I$.

Finally, the total gradient estimate is given by

$$\sum_{t \leqslant T} \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top = \sum_{t \leqslant T} b^{(t)\top} Q u u^\top Q^{-1} S^{(t)} J. \tag{4.6.2}$$

The $S^{(t)}$ matrix accounts for the fact that at time t of the algorithm, contributions $\mathcal{J}_{\theta_s}^{z_s}$ from future steps $s > t$ are not included in $\tilde{w}_t^\top$. Omitting this matrix would introduce terms that are zero in expectation and hence would not bias the total gradient estimate, but they would still contribute to the variance of the estimator (to a degree which would adversely affect the usefulness of our subsequent analysis).

It is easy to see that this estimator is unbiased as long as $\mathbb{E}\left[Q u u^\top Q^{-1}\right] = I$. This can happen, for example, when Q and u are independent with $\mathbb{E}[u u^\top] = I$. We will focus our analysis on this case.

4.6.5. Computing the Variance of the Total Gradient Estimate

In this section we derive the variance of the total gradient estimate. We assume that Q is independent of u, so that we may use the general results from Appendix 4.A.

By bilinearity, the covariance matrix of the total gradient estimate is

$$\operatorname{Var}\Big[\sum_{t \leqslant T} \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top\Big] = \sum_{t \leqslant T} \sum_{s \leqslant T} \operatorname{Cov}\Big[\mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top, \mathcal{J}_{h_s}^{L_s} \tilde{h}_s \tilde{w}_s^\top\Big].$$

Combining this with the identity $\mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top = b^{(t)\top} Q u u^\top Q^{-1} S^{(t)} J$ from the previous subsection and applying Corollary 3 (with $\kappa = 0$) yields the following expression for the same quantity:

$$\sum_{s \leqslant T} \sum_{t \leqslant T} \operatorname{tr}\Big(b^{(s)} b^{(t)\top} Q Q^\top\Big) J^\top S^{(s)} (Q Q^\top)^{-1} S^{(t)} J + J^\top b^{(s)} b^{(t)\top} J.$$

Corollary 3 also yields the following expression for the *total variance*[2] of the total gradient estimate:

$$\sum_{s \leqslant T} \sum_{t \leqslant T} \operatorname{tr}\Big(b^{(s)} b^{(t)\top} Q Q^\top\Big) \operatorname{tr}\Big(J^\top S^{(s)} (Q Q^\top)^{-1} S^{(t)} J\Big) + \operatorname{tr}\Big(J^\top b^{(s)} b^{(t)\top} J\Big). \tag{4.6.3}$$

[2] We define the "total variance" to be the trace of the covariance matrix.

4.7. Variance Reduction

We now turn to the problem of reducing the variance given in Equation 4.6.3. In Sections 4.7.1 through 4.7.1 we develop an improved (though as yet impractical) variance reduction scheme. Finally, we evaluate our theory in Section 4.7.2.

4.7.1. Optimizing Q subject to restrictions on its form

Denote by $V(Q)$ the part of the total variance (Equation 4.6.3) that depends on Q. Making use of the cyclic property of the trace, and the fact that Q is block-diagonal, we can write this as

$$V(Q) = \sum_{s \leqslant T} \sum_{t \leqslant T} \operatorname{tr}\Big(\sum_{r \leqslant T} b_r^{(s)} b_r^{(t)\top} Q_r Q_r^\top \Big) \operatorname{tr}\Big(\sum_{r \leqslant T} S_r^{(t)} J_r J_r^\top S_r^{(s)} (Q_r Q_r^\top)^{-1} \Big). \tag{4.7.1}$$

We wish to optimize $V(Q)$ with respect to Q in a way that leads to a practical online algorithm. To this end, we require that Q_s be of the form $Q_s = \alpha_s Q_0$, with α_s a scalar and Q_0 a constant matrix. This restriction makes sense from a practical standpoint; we envision an algorithm that maintains a statistical estimate of the optimal value of Q_0. The stationarity assumption enables us to amortize over time both the sample complexity of obtaining this estimate, and the computational cost associated with inverting it.

We furthermore assume projection occurs in preactivation space, that is, $z_r \equiv W_r a_r$. This assumption gives $J_r = \mathcal{J}_{z_r}^{h_r} = I \otimes a_r^\top$, which is a convenient algebraic structure to work with. Even given this restricted form we cannot find the jointly optimal solution for Q_0 and α. Instead, we will consider optimizing Q_0 while holding the α_s's fixed, and vice versa.

Optimizing α_s coefficients given Q_0.

Let us first simplify the expression for $V(Q)$. Given the restricted form $Q_s = \alpha_s Q_0$ we may write

$$V(Q) = \sum_{r \leqslant T} \sum_{q \leqslant T} \frac{\alpha_r^2}{\alpha_q^2} C_{qr}, \tag{4.7.2}$$

where we have collected the factors that do not depend on α into the matrix C with elements

$$\begin{aligned} C_{qr} &= \sum_{s \leqslant T} \sum_{t \leqslant T} \operatorname{tr}\Big(b_r^{(s)} b_r^{(t)\top} Q_0 Q_0^\top \Big) \operatorname{tr}\Big(S_q^{(t)} J_q J_q^\top S_q^{(s)} (Q_0 Q_0^\top)^{-1} \Big) \\ &= \operatorname{tr}\Big(\sum_{s=q}^{T} \sum_{t=q}^{T} b_r^{(s)} b_r^{(t)\top} Q_0 Q_0^\top \Big) \operatorname{tr}\Big(J_q J_q^\top (Q_0 Q_0^\top)^{-1} \Big) \\ &= \Big\| \sum_{t=q}^{T} b_r^{(t)\top} Q_0 \Big\|^2 \Big\| Q_0^{-1} J_q \Big\|_F^2. \end{aligned} \tag{4.7.3}$$

Now we wish to solve

$$\alpha^\star = \operatorname*{argmin}_{\alpha>0} \sum_{r\leqslant T}\sum_{q\leqslant T} \frac{\alpha_r^2}{\alpha_q^2} C_{qr}. \tag{4.7.4}$$

The optimization problem considered here differs from that given in Section 4.6.1. Although the objective considered there can similarly be written in terms of a matrix like C, that matrix would have rank one (see Appendix 4.B). This difference is a consequence of $V(Q)$ being the variance of the *total* gradient estimate rather than that of a single contribution $\mathcal{J}_{h_t}^{L_t}\tilde{h}_t\tilde{w}_t^\top$. In particular, the rank-one property is lost due to our inclusion of the $S^{(t)}$ matrix that discards noncausal terms (see Section 4.6.4).

We analyze the problem in Appendix 4.C, and find that it is an instance of matrix equilibration (see e.g. Idel, 2016, for a review), for which no closed-form solution is known. Instead, we give a second-order steepest-descent update rule that solves for α numerically, which we use in our experiments. (Empirically, first-order updates routinely get stuck in cycles on this problem.)

However, solving Equation 4.7.4 directly does not lead to a practical algorithm. Along the lines of the discussion in Section 4.6.1, any algorithm that maintains past contributions as a single sum must take α_s to be $\beta_s\gamma_{s+1}\gamma_{s+2}\dots\gamma_T$ for some coefficient sequences $\{\beta_s\}$ and $\{\gamma_s\}$. In principle, if C were known upfront, one could choose $\beta_s = \alpha_s^\star$ with $\gamma_s = 1$, and hence this parameterization appears to be degenerate. However, C is not known; it depends on gradients $b_r^{(t)} = \mathcal{J}_{z_r}^{L_t}$ and Jacobians $J_t = \mathcal{J}_{\theta_t}^{z_t}$ from future time steps $t > s$. In light of this, we can view β_s as merely an estimate of $\alpha_s^\star$, to be corrected by future γ_t's as more information becomes available.

One way of formalizing this idea of "incomplete information" is as follows. Suppose C were the final element $C^{(T)}$ of a sequence of matrices $C^{(1)}\dots C^{(T)}$, where each $C^{(s)}$ incorporates all "information" available up to time s. Then a natural way to choose β_s and γ_s at time s would be solve the following optimization problem based on $C^{(s)}$:

$$\beta_s^\star, \gamma_s^\star = \operatorname*{argmin}_{\beta_s,\gamma_s} \min_{\beta_{>s},\gamma_{>s}} \sum_{r\leqslant T}\sum_{q\leqslant T} \frac{\alpha_r^2}{\alpha_q^2} C_{qr}^{(s)}. \tag{4.7.5}$$

Past coefficients $\beta_{<s}, \gamma_{<s}$ are known (and fixed), and the unknown future coefficients $\beta_{>s}, \gamma_{>s}$ are estimated by the inner minimization.

In Appendix 4.D we explore a natural choice for $C^{(s)}$ where future gradients/Jacobians are treated as though they were 0, which leads to formulas for the coefficients that are similar to GIR's, although not identical. This approach can be improved by incorporating statistical predictions or estimates of unknown future information in $C^{(s)}$. We leave further exploration of such schemes to future work.

Optimizing Q_0 given the α_s coefficients.

Given our assumption that $z_r \equiv W_r a_r$ we have $J_r = I \otimes a_r^\top$ and $J_r J_r^\top = (I \otimes a_r^\top)(I \otimes a_r) = (I \otimes a_r^\top a_r) = \|a_r\|^2 I$. Thus,

$$S_r^{(t)} J_r J_r^\top S_r^{(s)} (Q_r Q_r^\top)^{-1} = \mathbb{1}_{r \leqslant t} \mathbb{1}_{r \leqslant s} \|a_r\|^2 \alpha_r^{-2} (Q_0 Q_0^\top)^{-1},$$

and $V(Q)$ becomes

$$\sum_{s \leqslant T} \sum_{t \leqslant T} \operatorname{tr}\Big(\sum_{r \leqslant T} \alpha_r^2 b_r^{(s)} b_r^{(t)\top} Q_0 Q_0^\top \Big) \operatorname{tr}\Big(\sum_{r=1}^{\min(s,t)} \|a_r\|^2 \alpha_r^{-2} (Q_0 Q_0^\top)^{-1} \Big).$$

Now we can move the scalar $\sum_{r=1}^{\min(s,t)} \|a_r\|^2 \alpha_r^{-2}$ leftward and group the terms that depend on s and t, giving

$$V(Q) = \operatorname{tr}(B Q_0 Q_0^\top) \operatorname{tr}((Q_0 Q_0^\top)^{-1}), \tag{4.7.6}$$

where

$$B = \sum_{s \leqslant T} \sum_{t \leqslant T} \Big(\sum_{q=1}^{\min(s,t)} \alpha_q^{-2} \|a_q\|^2 \Big) \Big(\sum_{r \leqslant T} \alpha_r^2 b_r^{(s)} b_r^{(t)\top} \Big) = \sum_{q \leqslant T} \sum_{r \leqslant T} \frac{\alpha_r^2}{\alpha_q^2} \|a_q\|^2 \Big(\sum_{s=q}^{T} b_r^{(s)} \Big) \Big(\sum_{t=q}^{T} b_r^{(t)} \Big)^\top. \tag{4.7.7}$$

The matrix B is PSD (it is a sum of PSD matrices), and we will further assume it is invertible. By Theorem 5 (which is stated and proved in Appendix 4.E) any choice of Q_0 satisfying $\eta B Q_0 Q_0^\top = (Q_0 Q_0^\top)^{-1}$ for some constant $\eta > 0$ will be a global minimizer of $V(Q)$. One such choice is

$$Q_0 = B^{-1/4}.$$

This solution, or any other globally optimal one, gives us

$$V(Q) = \operatorname{tr}(B^{1/2})^2,$$

where λ is the vector of eigenvalues of $B^{1/2}$. We can compare this to the variance attained by temporal scaling only ($Q_0 = I$):

$$V(Q) = \operatorname{tr}(B) \operatorname{tr}(I).$$

Writing $\operatorname{tr}(B^{1/2})^2 = (\vec{1}^\top \lambda)^2$ and $\operatorname{tr}(B) \operatorname{tr}(I) = \|\vec{1}\|^2 \|\lambda\|^2$, where $\vec{1}$ is the vector of ones and λ is the vector of eigenvalues of $B^{1/2}$, we have by the Cauchy-Schwarz inequality that

$$\operatorname{tr}(B^{1/2})^2 = (\vec{1}^\top \lambda)^2 \leqslant \|\vec{1}\|^2 \|\lambda\|^2 = \operatorname{tr}(B) \operatorname{tr}(I).$$

This approaches equality as λ approaches a multiple of $\vec{1}$, or in other words, as the spectrum of $B^{1/2}$ becomes flat. Conversely, the inequality will be more extreme when the spectrum is lopsided, indicating improved variance reduction when using $Q_0 = B^{-1/4}$ over the default choice $Q_0 = I$.

Practical Considerations. In practice, the proposed choice of Q_0 requires computing the B matrix and its eigendecomposition. Computing B involves four levels of summations over time and seemingly cannot be computed online. However, we can estimate it using quantities similar to the ones we use to estimate the gradient. Appendix 4.F derives the following unbiased estimator of B:

$$B \approx \frac{1}{2}(\tilde{m}_T \tilde{n}_T^\top + \tilde{n}_T \tilde{m}_T^\top)$$

where $\tilde{m}_t$ is given by

$$\sum_{s \leqslant t} \Big(\sum_{q \leqslant s} \sigma_q \alpha_q^{-1} \|a_q\| \Big) \Big(\sum_{r \leqslant s} \tau_r \alpha_r b_r^{(s)\top} \nu_r \Big) \Big(\sum_{r \leqslant s} \nu_r \Big)$$

and $\tilde{n}_t$ is like $\tilde{m}_t$ except with spatial noise μ_r instead of and independent of ν_r. In these expressions, σ, μ are temporal and spatial noise vectors distributed identically to τ, ν. This extra layer of stochastic approximation severely degrades the quality of the estimates. Additionally, the estimator depends on unknown future quantities, such as the total future gradient with respect to all time steps. As detailed in Appendix 4.F, we may compute intermediate estimates based on $\tilde{m}_t, \tilde{n}_t$ for $t < T$. To the extent that B is stationary, a moving average of these intermediate estimates can serve as a good approximation to B.

Empirically however, computing Q_0 based on this kind of estimator does not seem to improve optimization performance, due to its high variance. We leave a broader exploration of approximation algorithms for B to future work, while noting that an estimator for B need not be unbiased in order for us to obtain an unbiased estimate of the gradient. Indeed, *any* invertible choice of Q_0 will result in an unbiased estimate of the gradient, as was shown in Section 4.6.4. Unbiasedness may not even be a particularly desirable property for the B estimator to have, compared to other reasonable-sounding properties such as positive-semidefiniteness.

Once we have our estimate $\hat{B}$ of B and wish to compute its fourth root, the $\mathcal{O}(H^3)$ cost of factorization could be amortized by only performing it every so often or maintaining the estimate in factored form. It is often advisable to "dampen" or "regularize" the estimate by adding a multiple of the identity, i.e.

$$Q_0 = (\hat{B} + \lambda I)^{1/4}$$

where the hyperparameter λ serves to control the amount of trust placed in the estimate by biasing it towards a flat eigenvalue spectrum (i.e. towards $Q_0 \propto I$).

4.7.2. Variance Reduction Experiments

We empirically evaluate four settings for $Q_s = \alpha_s Q_0$ in a controlled setting based on the sequential MNIST task (Le et al., 2015). We choose this task because it is episodic; it gives

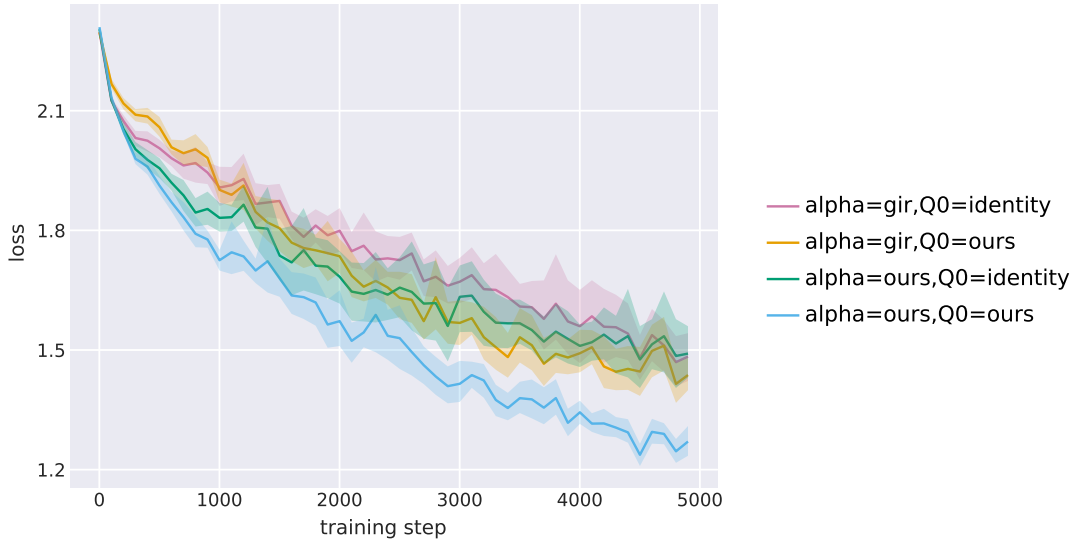


Fig. 4.1. Training curves on the row-wise sequential MNIST task. For each setting we have run 10 trials and plotted the mean of the classification loss and the 95% confidence interval of the mean. For clarity of presentation, these curves have been aggressively smoothed by a median filter prior to the computation of their statistics.

us access to gradients $b_s^{(t)}$ and Jacobians J_s for all s, t by BPTT. Thus we can compute the matrices B and C from Section 4.7.1 exactly. In order to curb the cost of these computations, we simplify the task to be row-by-row instead of pixel-by-pixel (i.e. $T = 28$ as opposed to $T = 784$). Moreover, the model is tasked with classifying the digit at every step rather than only at the end, as otherwise $L_t = 0$ and therefore $b_s^{(t)} = 0$ for $t < T$, trivializing the total gradient estimate (Equation 4.6.2).

For α_s, we compare the GIR-style coefficients

$$\gamma_s^2 = \frac{\|\tilde{w}_{s-1}\|}{\|\mathcal{J}_{h_{s-1}}^{h_s} \tilde{h}_{s-1}\|} \qquad \text{and} \qquad \beta_s^2 = \frac{\left\|u_s^\top Q_0^{-1} \mathcal{J}_{\theta_s}^{z_s}\right\|}{\|\mathcal{J}_{z_s}^{h_s} Q_0 u_s\|}$$

against the ones prescribed by our analysis. In the latter case, we use the algorithm described in Appendix 4.C to solve Equation 4.7.4 for α. Given α, we derive a sequence of γ, β coefficients by setting γ_s equal to the geometric average ratio of consecutive α_s's, and solving for β such that $\alpha_s = \beta_s \gamma_{s+1} \dots \gamma_T$ for all s.[3]

For Q_0, we consider the naive choice $Q_0 = I$ as well as the solution $Q_0 = B^{-1/4}$ from Section 4.7.1. Recall that the optimal Q_0 depends on the choice of α and both choices of α depend on the choice of Q_0. We break this circularity by maintaining an exponential

[3]The simpler choice $\gamma_s = 1, \beta_s = \alpha_s$ may run into numerical issues but is otherwise equivalent, as the distribution of the total scaling α across γ, β does not affect the variance.

moving average $\bar{B}$ of B across episodes, which we use to compute Q_0 according to

$$Q_0 = \left(\bar{B} + \lambda \frac{\mathrm{tr}(\bar{B})}{\mathrm{tr}(I)} I\right)^{1/4},$$

where the amount of damping/regularization is controlled by the hyperparameter λ. Given Q_0, we compute α exactly, process the episode and update the parameters by the total gradient estimate (Equation 4.6.2). At the end of the episode, we compute B exactly based on the α used in the episode, average it across the minibatch, and use the result to update $\bar{B}$.

The model consists of an LSTM (Hochreiter and Schmidhuber, 1997) with 50 hidden units. At each step, the digit is classified by softmax regression based on the hidden state h_t. As the classifier parameters do not affect h_t, their gradient is obtained by backprop. The gradients are averaged across a minibatch of 50 examples and across the duration of each episode, before being passed to the Adam (Kingma and Ba, 2014) optimizer. The settings of the learning rate, momentum and $\bar{B}$ decay and dampening hyperparameters are detailed in Appendix 4.G.

Figure 4.1 shows the training curves for each of the four configurations. While there is a clear advantage to using both our proposed α and Q_0 choices, that advantage appears to be lost when only one of the two is used.

In order to test our variance analysis, we show in Figure 4.2 predictions and measurements of several quantities that contribute to the variance, recorded during optimization. Recall from Section 4.6.5 that the variance of the total gradient estimate takes the form

$$\mathrm{Var}\Big[\sum_{t \leqslant T} \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top\Big] = V(Q) + \big\|\mathcal{J}_\theta^L\big\|^2.$$

The actual variance in Figure 4.2 measures $V(Q)$ empirically by computing

$$\mathbb{E}\Big[\big\|\sum_{t \leqslant T} \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_{t-1}^\top - \mathcal{J}_\theta^L\big\|^2 - \big\|\mathcal{J}_\theta^L\big\|^2\Big],$$

where the expectation is estimated by averaging across the minibatch. The intrinsic variance is similarly computed as $\mathbb{E}\big[\|\mathcal{J}_\theta^L\|^2\big]$. The expected variance measures the theoretical prediction of $V(Q)$ by plugging the corresponding choice of Q_0 into Equation 4.7.6.

We see that the theoretical predictions of $V(Q)$ are correct when alpha=ours, but that they overestimate $V(Q)$ when alpha=GIR. When we derived $V(Q)$ in Section 4.6.5, we started with the assumption that Q and u be independent; this assumption is violated by the GIR coefficients, which depend on the noise u. Finally, we see that our proposals indeed reduce the actual variance; significantly so when both Q0=ours, alpha=ours.

We furthermore highlight in Figure 4.3 the difference in behavior of the α coefficients under the four configurations. The GIR coefficients appear to take on more extreme values, especially early on in training. Presumably, poor initialization causes increased levels of

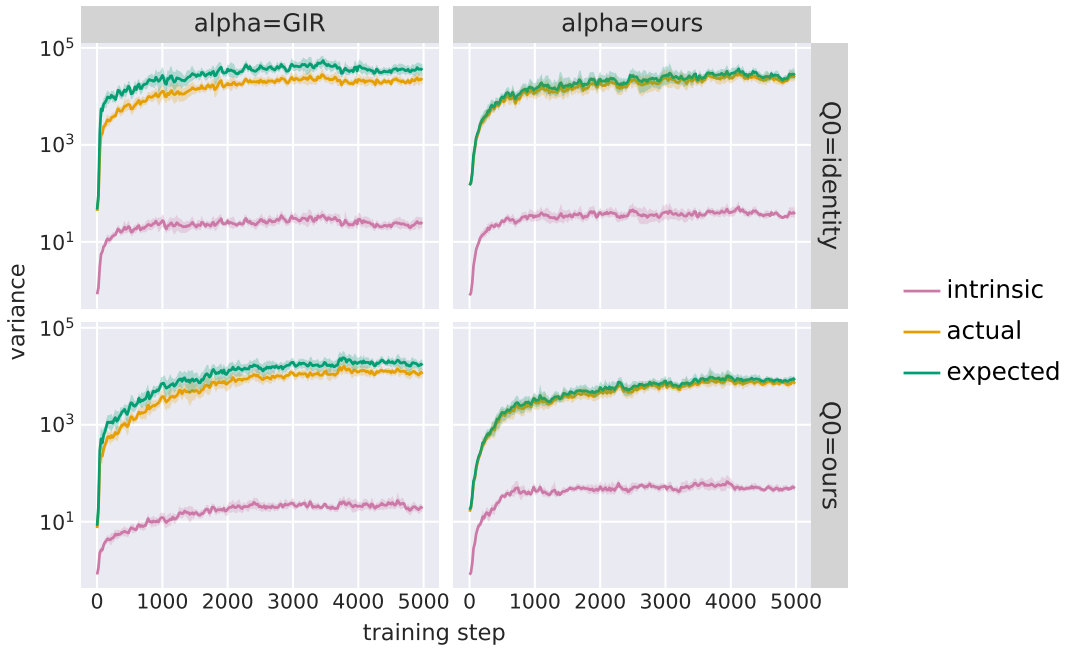


Fig. 4.2. Theoretical predictions and empirical measurements of quantities contributing to total gradient variance. The "intrinsic" variance measures the expected norm of the total gradient $\mathcal{J}_\theta^L$, estimated by averaging across the minibatch. The "expected" variance is a theoretical prediction of $V(Q)$ according to Equation 4.7.6. The "actual" variance measures $V(Q)$ empirically by the expected norm of the total gradient estimate.

gradient vanishing, which subsequently causes γ_s to be large in order to compensate. However, when we combine the GIR coefficients with our choice of Q_0, the effect is exacerbated. This may be because the GIR coefficients and our Q_0 optimize for conflicting objectives. Curiously, when both Q0=ours, alpha=ours, the relative ordering of the coefficients is reversed, so that $\alpha_s < \alpha_t$ for $s < t$.

4.8. Projection in the Space of Preactivations

Recall from Section 4.5 how the spatial rank-one approximation breaks down the Jacobian $\mathcal{J}_{\theta_t}^{h_t} = \mathcal{J}_{z_t}^{h_t} \nu_t \nu_t^\top \mathcal{J}_{\theta_t}^{z_t}$ into more manageable quantities $\mathcal{J}_{z_t}^{h_t} \nu_t$ and $\nu_t^\top \mathcal{J}_{\theta_t}^{z_t}$ by projecting in the space of some cut vertex z_t. Assuming the transition function F takes the form given in Equation 4.3.1, we observe that the Jacobian can be factored as $\mathcal{J}_{\theta_t}^{h_t} = \mathcal{J}_{W_t a_t}^{h_t} (I \otimes a_t^\top)$ where $\otimes$ denotes the Kronecker product, i.e. it is already rank-one. By choosing z_t to be the preactivations $W_t a_t$, we can avoid the projection, and we obtain the following recursion:

$$\begin{aligned} \tilde{h}_t &= \gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} + \beta_t \tau_t \mathcal{J}_{W_t a_t}^{h_t} \\ \tilde{w}_t &= \gamma_t^{-1} \tilde{w}_{t-1} + \beta_t^{-1} \tau_t a_t \end{aligned} \tag{4.8.1}$$

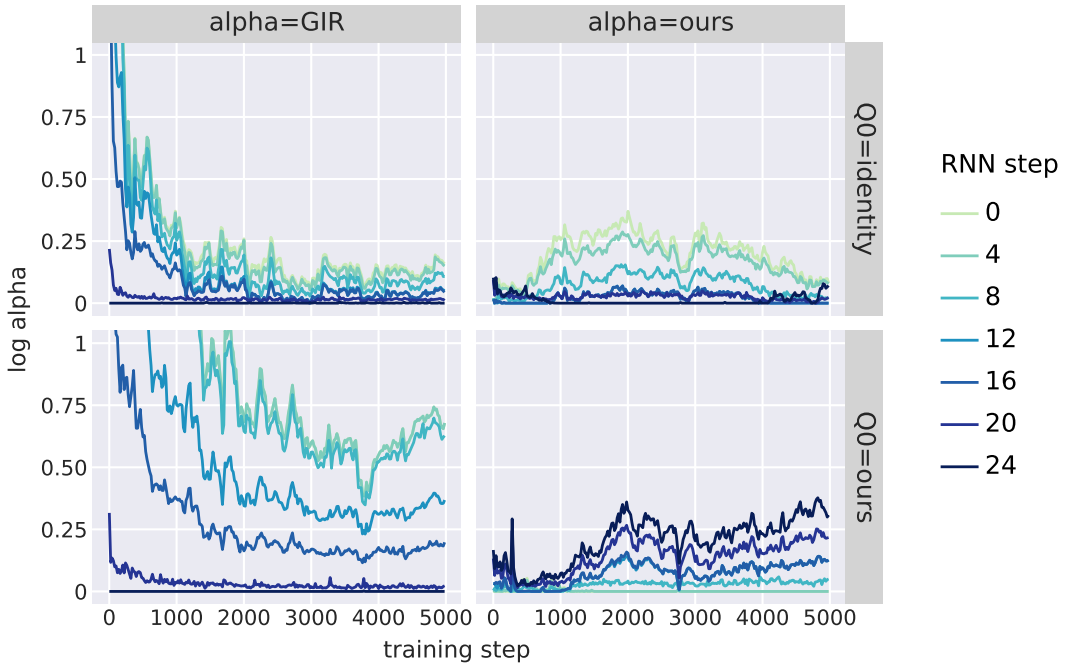


Fig. 4.3. Evolution of $\log \alpha_s$ for some time steps s as training proceeds. At each training step, the $\log \alpha_s$ are centered so that $\min_s \log \alpha_s = 0$; this eliminates irrelevant constant factors.

The vector-valued $\tilde{h}_t$ has been replaced by a matrix $\tilde{h}_t$, and the contributions $\mathcal{J}_{W_t a_t}^{h_t}$ and a_t are multiplied by scalar noise $\tau_s \sim \mathcal{N}(0,1)$ rather than projected down. At each step, the gradient contribution $\mathcal{J}_\theta^{L_t}$ is computed as $\mathrm{vec}((\mathcal{J}_{h_t}^{L_t} \tilde{h}_t)^\top \tilde{w}_t^\top)$. The GIR coefficients

$$\gamma_t^2 = \|\tilde{w}_{t-1}\| / \|\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1}\|_F, \qquad \beta_t^2 = \|a_t\| / \|\mathcal{J}_{W_t a_t}^{h_t}\|_F$$

can be derived like in Section 4.5. We will refer to this variant of UORO as "PREUORO". This algorithm has also been discovered by Mujika et al. (2018).

Define $b_s^{(t)} = \mathcal{J}_{z_s}^{L_t}$, the gradient of the loss at time t with respect to the projection variable at time s. Then the total gradient $\mathcal{J}_\theta^L$ can be expressed as

$$\mathcal{J}_\theta^L = \sum_{t \leqslant T} \mathcal{J}_\theta^{L_t} = \sum_{t \leqslant T} \sum_{s \leqslant t} b_s^{(t)\top} (I \otimes a_s^\top) = \sum_{t \leqslant T} \sum_{s \leqslant t} \mathrm{vec}(b_s^{(t)} a_s^\top),$$

where vec is the vectorization operator that serializes its matrix argument into a row vector in row-major order. We can express the total gradient estimate as

$$\begin{aligned} \mathrm{vec}\Big(\sum_{t \leqslant T} (\mathcal{J}_{h_t}^{L_t} \tilde{h}_t)^\top \tilde{w}_t^\top\Big) &= \mathrm{vec}\Big(\sum_{t \leqslant T} \big(\sum_{s \leqslant t} \tau_s \alpha_s b_s^{(t)}\big) \big(\sum_{r \leqslant t} \tau_r \alpha_r^{-1} a_r^\top\big)\Big) \\ &= \mathrm{vec}\Big(\sum_{t \leqslant T} \bar{B}^{(t)\top} \bar{Q} \tau \tau^\top \bar{Q}^{-1} \bar{S}^{(t)} \bar{J}\Big), \end{aligned} \tag{4.8.2}$$

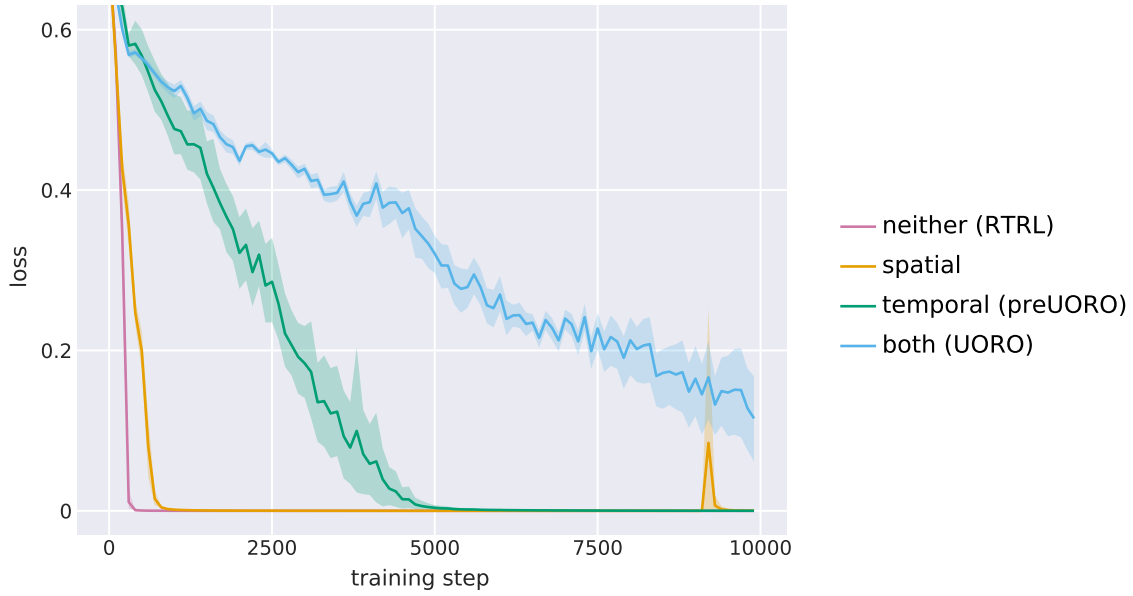


Fig. 4.4. Training curves on the queue task showing interpolation between RTRL and UORO by ablation of the spatial and temporal approximations. "neither" denotes exact computation of the gradient using RTRL, "spatial" denotes RTRL with $\mathcal{J}_{z_t}^{h_t} \nu_t \nu_t^\top \mathcal{J}_{\theta_t}^{z_t}$ standing in for $\mathcal{J}_{\theta_t}^{h_t}$, "temporal" denotes PREUORO computed by Equation 4.8.1, "both" denotes UORO . Where applicable, the cut vertex $z_t \equiv W_t a_t$ is the preactivations.

where we have defined the matrices

$$\bar{B}^{(t)\top} = \begin{pmatrix} b_1^{(t)} & \cdots & b_T^{(t)} \end{pmatrix}, \bar{Q} = \mathrm{diag}(\alpha), \bar{S}_{ij}^{(t)} = \delta_{ij} \mathbb{1}_{i \geqslant t}, \bar{J} = \begin{pmatrix} a_1 & \cdots & a_T \end{pmatrix}^\top.$$

that mirror similarly-named quantities from Section 4.6.4. The expression in Equation 4.8.2 is analogous to that in Equation 4.6.2, but with the crucial difference that no summation across space is involved. Hence the noise vector τ has much smaller dimension T rather than TN (with N being the dimension of the projection space).

We show in Appendix 4.H that the variance contribution $V(Q)$ of PREUORO can be written

$$\sum_{s \leqslant T} \sum_{t \leqslant T} \mathrm{tr}\left(\bar{B}^{(s)} \bar{B}^{(t)\top} \bar{Q} \bar{Q}^\top\right) \mathrm{tr}\left(\bar{S}^{(t)} \bar{J} \bar{J}^\top \bar{S}^{(s)} (\bar{Q} \bar{Q}^\top)^{-1}\right)$$

and the variance contribution $V(Q)$ of UORO 's total gradient estimate from Section 4.6.4 (Equation 4.6.2) can be written:

$$\sum_{s \leqslant T} \sum_{t \leqslant T} \mathrm{tr}\left(\bar{B}^{(t)} Q_0 Q_0^\top \bar{B}^{(s)\top} \bar{Q} \bar{Q}^\top\right) \mathrm{tr}\left(\bar{S}^{(t)} \bar{J} \bar{J}^\top \bar{S}^{(s)} (\bar{Q} \bar{Q}^\top)^{-1}\right) \mathrm{tr}\left((Q_0 Q_0^\top)^{-1}\right)$$

The latter has an extra factor $\mathrm{tr}((Q_0 Q_0^\top)^{-1})$. If $Q_0 = I$, then this factor is equal to $tr(I)$. Spatial projection thus causes the dominant term of the variance to be multiplied by the dimension of the preactivations, which typically ranges in the thousands. Avoiding the spatial projection avoids this multiplication and hence achieves drastically lower variance.

Figure 4.4 confirms the corresponding improvement in optimization performance. This figure shows training curves of four variations on RTRL: RTRL, RTRL plus spatial projection (UORO minus temporal projection), PREUORO (UORO minus spatial projection), and UORO which performs both spatial and temporal projection. The task under consideration is the queue task, in which the model is trained to emit its input stream with a delay. Effectively, the model learns to implement a queue.

The model is similar to that described in 4.7.2, except with 50 hidden units. The model observes a random binary input stream and has to predict a binary output stream that is equal to the input stream but with a delay of 4 time steps. The $\mathcal{J}_\theta^{L_t}$ estimates are averaged across a minibatch of 100 examples, and applied to the parameters by Adam (Kingma and Ba, 2014) with momentum 0.5 and learning rate set to 0.008 for "neither", 0.008 for "spatial", 0.0008 for "temporal", 0.002 for "both" (found by grid search).

The main drawback of this method is its computational complexity: the algorithm involves propagating multiple vectors forward, which increases the computation time by the same factor N that we removed from the variance. The dominant operation is the matrix-matrix multiplication $\mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1}$, which has computational cost $\mathcal{O}(N^3)$ (recall N is the dimension of the projection space). This is better than RTRL's $\mathcal{J}_{h_{t-1}}^{h_t} \mathcal{J}_\theta^{h_{t-1}}$ which costs $\mathcal{O}(N^4)$, but worse than UORO and BPTT which propagate vectors at a cost of $\mathcal{O}(N^2)$. The space complexity is $\mathcal{O}(N^2)$, which matches that of UORO.

4.9. REINFORCE as Approximate Real-Time Recurrent Learning

In this section we show a fundamental connection between REINFORCE (Williams, 1992) and UORO. The REINFORCE algorithm provides gradient estimates for systems with stochastic transitions. It can also be used to train recurrent neural networks if we artificially induce stochasticity by adding Gaussian noise to the hidden states. We will show that in this setting, the REINFORCE estimator is closely related to the UORO estimator.

REINFORCE aims to estimate the gradient of the expected loss $\mathbb{E}_{\chi \sim p(\chi;\theta)}[L(\chi)]$ which depends on the parameter θ through some distribution $p(\chi;\theta)$ over stochastic context χ that determines the loss $L(\chi)$. Conceptually, $\chi = (\chi_t)$ is the trajectory of the state of an agent and its external environment, and θ parameterizes a stochastic policy over actions, which induces a distribution $p(\chi;\theta)$ on χ.

The gradient of the expected loss can be rewritten as an expected gradient as follows:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\chi \sim p(\chi;\theta)}[L(\chi)] = \nabla_\theta \int L(\chi) p(\chi;\theta) d\chi &= \int L(\chi) \nabla_\theta p(\chi;\theta) d\chi \\ &= \int L(\chi) \nabla_\theta (\log p(\chi;\theta)) p(\chi;\theta) d\chi,\end{aligned}$$

where we have used the fact that $\nabla_\theta \log p(\chi;\theta) = {}^{\nabla_\theta p(\chi;\theta)}/_{p(\chi;\theta)}$. With this modified expression, we can estimate $\nabla_\theta \mathbb{E}_{\chi\sim p(\chi;\theta)}[L(\chi)]$ by sampling from $p(\chi;\theta)$.

In our case, χ will be the trajectory of the stochastic hidden states of the RNN, and sampling from $p(\chi;\theta)$ will correspond to the following recursions:

$$\begin{aligned} h_t &= F(\bar{h}_{t-1}, x_t; \theta_t) \\ \bar{h}_t &= h_t + \sigma u_t, \end{aligned} \tag{4.9.1}$$

with additive Gaussian noise $u_t \sim \mathcal{N}(0, I)$. The stochastic hidden state $\bar{h}_t$ is effectively sampled from a state transition policy $p(\bar{h}_t|\bar{h}_{t-1}, \theta_t) \propto \exp\Big(-\frac{1}{2\sigma^2}\|\bar{h}_t - h_t\|^2\Big)$.

For each state $\bar{h}_t$ so visited, we compute the score $\nabla_\theta \log p(\bar{h}_{\leqslant t};\theta)$ of the trajectory $\bar{h}_{\leqslant t} = (\bar{h}_0, \bar{h}_1, \ldots, \bar{h}_t)$ that brought us there, and multiply it by an immediate loss L_t so obtained. Intuitively, higher rewards "reinforce" directions in parameter space that bring them about. We will assume L_t is a differentiable function of $\bar{h}_t$.

By the chain rule of probability, the score $\nabla_\theta \log p(\bar{h}_{\leqslant t};\theta)$ of the trajectory is simply the sum $\nabla_\theta \sum_{s=1}^t \log p(\bar{h}_s|\bar{h}_{s-1}, \theta_s)$, which we can recursively maintain according to

$$\begin{aligned} \bar{w}_t^\top &= \bar{w}_{t-1}^\top + \nabla_\theta \log p(\bar{h}_t|\bar{h}_{t-1}, \theta_t) = \bar{w}_{t-1}^\top - \frac{1}{2\sigma^2} \mathcal{J}_{h_t}^{\|\bar{h}_t - h_t\|^2} \mathcal{J}_{\theta_t}^{h_t} \\ &= \bar{w}_{t-1}^\top + \frac{1}{\sigma^2}(\bar{h}_t - h_t)^\top \mathcal{J}_{\theta_t}^{h_t} = \bar{w}_{t-1}^\top + \frac{1}{\sigma} u_t^\top \mathcal{J}_{\theta_t}^{h_t}. \end{aligned}$$

Note that in the above computations, "$\bar{h}_t$" and "$\bar{h}_{t-1}$" are not the variables themselves but particular values. (This is a consequence of our adoption of the standard abuse of notation for random variables.) Thus they are treated as constants with respect to differentiation. The only quantity that depends on θ is h_t, which when we condition on the value of $\bar{h}_{t-1}$, only depends on θ via θ_t.

This recursion is very similar to UORO's recursion for $\tilde{w}_t^\top$, and it computes a similar type of sum:

$$\bar{w}_t^\top = \frac{1}{\sigma} \sum_{s\leqslant t} u_s^\top \mathcal{J}_{\theta_s}^{h_s}. \tag{4.9.2}$$

Once we have $\nabla_\theta \log p(\bar{h}_{\leqslant t};\theta)$, we need to multiply it by the loss L_t to obtain a REINFORCE gradient estimate of $\mathcal{J}_\theta^{L_t}$. We can express the loss by its Taylor series around the point $u = 0$ where the noise is zero, as follows:

$$\begin{aligned} L_t &= L_t|_{u=0} + \Big(\sum_{s\leqslant t} \mathcal{J}_{u_s}^{L_t}|_{u=0} u_s\Big) + \frac{1}{2}\Big(\sum_{r\leqslant t}\sum_{s\leqslant t} u_r^\top \mathcal{H}_{u_r,u_s}^{L_t}\Big|_{u=0} u_s\Big) + \cdots \\ &= L_t|_{u=0} + \sigma\Big(\sum_{s\leqslant t} \mathcal{J}_{h_s}^{L_t}|_{u=0} u_s\Big) + \mathcal{O}(\sigma^2), \end{aligned}$$

where $\mathcal{H}^{L_t}_{u_r,u_s}$ denotes the Hessian of L_t with respect to u_r and u_s. The last step uses the fact that σu_s affects L_t in exactly the same way that h_s does, so that $\mathcal{J}^{L_t}_{u_s} = \sigma \mathcal{J}^{L_t}_{h_s}$ and $\mathcal{H}^{L_t}_{u_r,u_s} = \sigma^2 \mathcal{H}^{L_t}_{h_r,h_s}$.

Plugging the Taylor series for L_t into the REINFORCE gradient estimate and using Equation 4.9.2, we get:

$$L_t \bar{w}_t^\top = L_t|_{u=0} \bar{w}_t^\top + \Big(\sum_{s \leqslant t} \mathcal{J}^{L_t}_{h_s}|_{u=0} u_s\Big)\Big(\sum_{s \leqslant t} u_s^\top \mathcal{J}^{h_s}_{\theta_s}\Big) + \mathcal{O}(\sigma). \tag{4.9.3}$$

Here we see the UORO gradient estimator appear in the second term, but with an important difference: the $\mathcal{J}^{h_s}_{\theta_s}$'s are evaluated in the noisy system, whereas the $\mathcal{J}^{L_t}_{h_s}|_{u=0}$ are evaluated with zero noise. Thus this term doesn't estimate $\mathcal{J}^{L_t}_{\theta}$ for any value of u. However, the equivalence becomes exact when we let the noise go to zero by taking the limit $\sigma \to 0$.

To see this we first observe that letting σ go to 0 is equivalent to letting u go to 0 in the recursions for h_s (Equation 4.9.1). Furthermore, since F is continuously differentiable, so is h_s (w.r.t. all of its dependencies). Therefore $\mathcal{J}^{h_s}_{\theta_s}$ is a continuous function of u, and it follows that

$$\lim_{\sigma \to 0} \mathcal{J}^{h_s}_{\theta_s} = \lim_{u \to 0} \mathcal{J}^{h_s}_{\theta_s} = \mathcal{J}^{h_s}_{\theta_s}|_{u=0}.$$

And therefore we have

$$\lim_{\sigma \to 0}\Big[\Big(\sum_{s \leqslant t} \mathcal{J}^{L_t}_{h_s}|_{u=0} u_s\Big)\Big(\sum_{s \leqslant t} u_s^\top \mathcal{J}^{h_s}_{\theta_s}\Big) + \mathcal{O}(\sigma)\Big] = \Big(\sum_{s \leqslant t} \mathcal{J}^{L_t}_{h_s}|_{u=0} u_s\Big)\Big(\sum_{s \leqslant t} u_s^\top \mathcal{J}^{h_s}_{\theta_s}|_{u=0}\Big),$$

which is identical to the standard UORO estimate $\mathcal{J}^{L_t}_{h_t} \tilde{h}_t \tilde{w}_t^\top$ of $\mathcal{J}^{L_t}_{\theta}$ (without any variance reduction).

Thus we can see that in the limit as $\sigma \to 0$, REINFORCE becomes equivalent to UORO (sans variance reduction), except that it includes the additional term:

$$L_t|_{u=0} \bar{w}_t^\top = \frac{1}{\sigma} L_t|_{u=0} \sum_{s \leqslant t} u_s^\top \mathcal{J}^{h_s}_{\theta_s}.$$

From the right-hand side we see that this term has mean zero, and thus the limiting behavior of REINFORCE is to give an unbiased estimate of the gradient of the noise-free model. However, the variance of the additional term goes to infinity as $\sigma \to 0$. For models where the noise is bounded away from zero this term represents the main source of variance for REINFORCE estimators. It can however be addressed by subtracting an estimate of $L_t|_{u=0}$ from L_t before multiplying by the score function. This is known as a "baseline" in the REINFORCE literature (Williams, 1992).

The appearance of the UORO estimator as part of the REINFORCE estimator suggests an additional opportunity for variance reduction in REINFORCE. If in Equation 4.9.1 we had

instead defined

$$\bar{h}_t = h_t + \sigma Q_t u_t,$$

that is, the noise added to h_t has covariance $\sigma^2 Q_t^\top Q_t$, then we would have found

$$\bar{w}_t^\top = \frac{1}{\sigma} \sum_{s \leqslant t} u_s^\top Q^{-1} \mathcal{J}_{\theta_s}^{h_s} \quad \text{and} \quad \sum_{s \leqslant t} \mathcal{J}_{u_s}^{L_t}|_{u=0} = \sigma \sum_{s \leqslant t} \mathcal{J}_{h_s}^{L_t}|_{u=0} Q_s u_s.$$

Putting these two together as in Equation 4.9.3 and passing to the limit $\sigma \to 0$ as before, we get

$$\lim_{\sigma \to 0} L_t \bar{w}_t^\top = L_t|_{u=0} \bar{w}_t^\top + \Big(\sum_{s \leqslant t} \mathcal{J}_{h_s}^{L_t}|_{u=0} Q_s u_s \Big) \Big(\sum_{s \leqslant t} u_s^\top Q_s^{-1} \mathcal{J}_{\theta_s}^{h_s}|_{u=0} \Big),$$

where now the second term is identical to UORO *with* the generalized variance reduction described in Section 4.6.3. Thus the Q_s matrices that enable variance reduction in UORO correspond directly to a choice of covariance on the exploration noise in REINFORCE.

4.10. Conclusions

We have contributed a thorough analysis of UORO-style approximate differentiation algorithms and their variance behavior. The theory takes a holistic view of the algorithm as part of an optimization process, where the sequence of mutually dependent gradient estimates $\mathcal{J}_\theta^{L_t} \approx \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{w}_t^\top$ produced by UORO are accumulated as per gradient descent. Our analysis considers the variance of this total gradient estimate. This is in contrast to UORO's variance reduction scheme (GIR) which minimizes the variance of individual Jacobian estimates $\mathcal{J}_\theta^{h_t} \approx \tilde{h}_t \tilde{w}_t^\top$, without accounting for the way in which they are used. We have developed a generalization of GIR, and suggested avenues toward a practical implementation. Empirical evaluation confirms our theoretical claims.

Furthermore we have described a variation on UORO that avoids "spatial" projection, greatly reducing the variance at the cost of increased computational complexity. Finally, we have drawn a deep connection between UORO and REINFORCE when the latter is used to train an RNN with perturbed hidden states.

Acknowledgements

The authors thank Max Jaderberg, David Sussillo, David Duvenaud and Aaron Courville for helpful discussion, and Chris Maddison and Grzegorz Swirszcz for reviewing drafts of this paper. This research was enabled by computational resources courtesy of Compute Canada.

References

Arjona-Medina, Jose A et al. (2018). "RUDDER: Return Decomposition for Delayed Rewards". In: *arXiv preprint arXiv:1806.07857*.

Barto, Andrew G, Richard S Sutton, and Charles W Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics*, pp. 834–846.

Chen, Tianqi et al. (2016). "Training deep nets with sublinear memory cost". In: *arXiv preprint arXiv:1604.06174*.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Gomez, Aidan N et al. (2017). "The reversible residual network: Backpropagation without storing activations". In: *Advances in Neural Information Processing Systems*, pp. 2214–2224.

Gruslys, Audrunas et al. (2016). "Memory-efficient backpropagation through time". In: *Advances in Neural Information Processing Systems*, pp. 4125–4133.

Heath, Michael T (2018). *Scientific computing: an introductory survey*. Vol. 80.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hung, Chia-Chun et al. (2018). "Optimizing Agent Behavior over Long Time Scales by Transporting Value". In: *arXiv preprint arXiv:1810.06721*.

Idel, Martin (2016). "A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps". In: *arXiv preprint arXiv:1609.06349*.

Jaderberg, Max et al. (2017). "Decoupled Neural Interfaces using Synthetic Gradients". In: *International Conference on Machine Learning*, pp. 1627–1635.

Ke, Nan Rosemary et al. (2018). "Sparse Attentive Backtracking: Temporal credit assignment through reminding". In: *Advances in Neural Information Processing Systems*, pp. 7651–7662.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Konda, Vijay R and John N Tsitsiklis (2000). "Actor-critic algorithms". In: *Advances in neural information processing systems*, pp. 1008–1014.

Le, Quoc V, Navdeep Jaitly, and Geoffrey E Hinton (2015). "A simple way to initialize recurrent networks of rectified linear units". In: *arXiv preprint arXiv:1504.00941*.

MacKay, Matthew et al. (2018). "Reversible Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems*, pp. 9042–9053.

Mak, Man-Wai, Kim-Wing Ku, and Yee-Ling Lu (1999). "On the improvement of the real time recurrent learning algorithm for recurrent neural networks". In: *Neurocomputing* 24.1-3, pp. 13–36.

Massé, Pierre-Yves (2017). "Around the Use of Gradients in Machine Learning". PhD thesis. Université Paris-Saclay. URL: https://tel.archives-ouvertes.fr/tel-01744761.

Minsky, Marvin (1961). "Steps toward artificial intelligence". In: *Proceedings of the IRE* 49.1, pp. 8–30.

Mujika, Asier, Florian Meier, and Angelika Steger (2018). "Approximating Real-Time Recurrent Learning with Random Kronecker Factors". In: *Advances in Neural Information Processing Systems*. Vol. 31, pp. 6594–6603.

Ollivier, Yann, Corentin Tallec, and Guillaume Charpiat (2015). "Training recurrent networks online without backtracking". In: *arXiv preprint arXiv:1507.07680*.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, p. 533.

Sutton, Richard S (1984). "Temporal credit assignment in reinforcement learning". PhD thesis. University of Massachusetts Amherst.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

Tallec, Corentin and Yann Ollivier (2018). "Unbiased Online Recurrent Optimization". In: *International Conference on Learning Representations*.

Werbos, Paul J (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement Learning*, pp. 5–32.

Williams, Ronald J and Jing Peng (1990). "An efficient gradient-based algorithm for on-line training of recurrent network trajectories". In: *Neural computation* 2.4, pp. 490–501.

Williams, Ronald J and David Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2, pp. 270–280.

Williams, Ronald J and David Zipser (1995). "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: *Backpropagation: Theory, architectures, and applications* 1, pp. 433–486.

4.A. Supporting Results for Variance Computations

In this section we prove several technical results supporting our variance computations in the main text.

Definition 1 (Standard random vector)**.** A standard random vector is any real vector u whose elements u_i are drawn iid from a distribution that is symmetric around zero and has unit variance.

Standard random vectors u satisfy $\mathbb{E}[u] = 0$ and $\mathbb{E}[uu^\top] = I$, which is required for our algorithms to be unbiased. Moreover, by symmetry the odd moments of their elements u_i are zero. The results below will involve the "excess kurtosis" $\mathbb{E}[u_1^4] - 3$ of the distribution of the elements of u. The standard normal distribution $\mathcal{N}(0,1)$ has excess kurtosis 0, whereas the uniform distribution on signs $\mathcal{U}\{-1,+1\}$ has excess kurtosis -2.

Proposition 2. *Suppose A, B, C, D are constant matrices, and u is a standard random vector with excess kurtosis κ. Then we have*

$$\mathbb{E}[Auu^\top BCuu^\top D] = \operatorname{tr}(BC)AD + 2ABCD + \kappa A((BC) \odot I)D.$$

PROOF. By linearity of expectation,

$$\mathbb{E}[Auu^\top BCuu^\top D] = \sum_{jklm} \mathbb{E}[u_j u_k u_l u_m] A_{ij} (BC)_{kl} D_{mn}.$$

In order to evaluate the expectation $\mathbb{E}[u_j u_k u_l u_m]$, we make use of the fact that u_i and u_j are independent unless $i = j$. This allows us to express the product inside the expectation as a product of powers $u_i^{p(i)}$, with the power $p(i)$ equal to the multiplicity of i in (j,k,l,m). By independence, the expectation of this product then factors into a product $\prod_i \mathbb{E}[u_i^{p(i)}] = \prod_i \mu_{p(i)}$ of moments $\mu_p \triangleq \mathbb{E}[u_1^p]$ of the elements u_i. Moreover, since by symmetry the odd moments of u_i are zero, we need only consider cases in which all indices have even multiplicity. Thus we get

$$\sum_{jklm} \mathbb{E}[u_j u_k u_l u_m] A_{ij} (BC)_{kl} D_{mn} = \sum_{jklm} \begin{cases} \mu_4 A_{ij} (BC)_{jj} D_{jn} & \text{if } j = k = l = m \\ \mu_2^2 A_{ij} (BC)_{jl} D_{ln} & \text{if } j = k \neq l = m \\ \mu_2^2 A_{ij} (BC)_{kj} D_{kn} & \text{if } j = l \neq k = m \\ \mu_2^2 A_{ij} (BC)_{kk} D_{jn} & \text{if } j = m \neq k = l \\ 0 & \text{else} \end{cases}.$$

Casting this back into matrix form, we have

$$\begin{aligned} \mathbb{E}[Auu^\top BCuu^\top D] &= \mu_2^2 \operatorname{tr}(BC)AD + 2\mu_2^2 ABCD + (\mu_4 - 3\mu_2^2) A((BC) \odot I)D \\ &= \operatorname{tr}(BC)AD + 2ABCD + \kappa A((BC) \odot I)D, \end{aligned}$$

where $\mu_2^2 = 1$ follows from the fact that u is a standard random vector, and $\mu_4 - 3\mu_2^2 = \mu_4 - 3 = \kappa$ is its excess kurtosis.

□

Corollary 3. *Suppose x and y are constant vectors, V and W are constant matrices, and u is a standard random vector with excess kurtosis κ. Then*

$$\mathrm{Cov}[x^\top uu^\top V, y^\top uu^\top W] = (x^\top y)V^\top W + V^\top xy^\top W + \kappa V^\top((xy^\top)\odot I)W$$

and

$$\begin{aligned}\operatorname{tr}\big(\mathrm{Cov}[x^\top uu^\top V, y^\top uu^\top W]\big) = (x^\top y)\operatorname{tr}(V^\top W) + 2\operatorname{tr}(V^\top xy^\top W)\\ + \kappa\operatorname{tr}\big(V^\top((xy^\top)\odot I)W\big).\end{aligned}$$

PROOF. $x^\top uu^\top V$ and $y^\top uu^\top W$ are row vectors and so their covariance is given by

$$\begin{aligned}\mathrm{Cov}[x^\top uu^\top V, y^\top uu^\top W] &= \mathbb{E}[(x^\top uu^\top V)^\top (y^\top uu^\top W)] - \mathbb{E}[x^\top uu^\top V]^\top \mathbb{E}[y^\top uu^\top W]\\ &= \mathbb{E}[V^\top uu^\top xy^\top uu^\top W] - \mathbb{E}[V^\top uu^\top x]\mathbb{E}[y^\top uu^\top W].\end{aligned}$$

By Proposition 2,

$$\begin{aligned}\mathbb{E}[V^\top uu^\top xy^\top uu^\top W] &= \operatorname{tr}(xy^\top)V^\top W + 2V^\top xy^\top W + \kappa V^\top((xy^\top)\odot I)W\\ &= (x^\top y)V^\top W + 2V^\top xy^\top W + \kappa V^\top((xy^\top)\odot I)W.\end{aligned}$$

And by linearity of expectation we have $\mathbb{E}[y^\top uu^\top W] = y^\top \mathbb{E}[uu^\top]W = y^\top W$ and similarly $\mathbb{E}[V^\top uu^\top x] = V^\top x$, so that $\mathbb{E}[V^\top uu^\top x]\mathbb{E}[y^\top uu^\top W] = V^\top xy^\top W$. Combining these equations yields

$$\mathrm{Cov}[x^\top uu^\top V, y^\top uu^\top W] = (x^\top y)V^\top W + V^\top xy^\top W + \kappa V^\top((xy^\top)\odot I)W.$$

The formula for $\operatorname{tr}\big(\mathrm{Cov}[x^\top uu^\top V, y^\top uu^\top W]\big)$ follows immediately. □

4.B. Variance of a Single Jacobian Estimate

Section 4.6.1 discusses the following expression for the UORO Jacobian estimate at time t in terms of the overall coefficients $\alpha_r^{(t)}$:

$$\mathcal{J}_\theta^{h_t} \approx \tilde{h}_t\tilde{w}_t^\top = \Big(\sum_{s\leqslant t}\alpha_s^{(t)}\mathcal{J}_{z_s}^{h_t}u_s\Big)\Big(\sum_{r\leqslant t}\frac{1}{\alpha_r^{(t)}}u_r^\top\mathcal{J}_{\theta_r}^{z_r}\Big) = \sum_{r\leqslant t}\sum_{s\leqslant t}\frac{\alpha_s^{(t)}}{\alpha_r^{(t)}}\tau_s\tau_r\mathcal{J}_{h_s}^{h_t}\mathcal{J}_{z_s}^{h_s}\nu_s\nu_r^\top\mathcal{J}_{\theta_r}^{z_r}. \quad (4.B.1)$$

This section concerns the variance of this estimate and the coefficients $\alpha_s^{(t)}$ that minimize it. We will omit the superscript on $\alpha_s^{(t)}$ to avoid notational clutter.

Defining $R^\top = \begin{pmatrix} \mathcal{J}_{z_1}^{h_t}\nu_1 & \cdots & \mathcal{J}_{z_t}^{h_t}\nu_t \end{pmatrix}$, $\tilde{J}^\top = \begin{pmatrix} \tilde{J}_1 & \cdots & \tilde{J}_t \end{pmatrix}$ for $\tilde{J}_s^\top = \nu_s^\top \mathcal{J}_{\theta_s}^{z_s}$, and the diagonal matrix $A = \operatorname{diag}(\alpha)$, we can write Equation 4.B.1 as

$$\tilde{h}_t \tilde{w}_t^\top = R^\top A \tau \tau^\top A^{-1} \tilde{J}.$$

Its variance with respect to the temporal noise τ is given by

$$\begin{aligned} &\mathbb{E}_\tau\big[\|R^\top A\tau\tau^\top A^{-1}\tilde{J}\|_F^2\big] - \big\|\mathbb{E}_\tau[R^\top A\tau\tau^\top A^{-1}\tilde{J}]\big\|_F^2 \\ =\ &\operatorname{tr}\big(\mathbb{E}_\tau[R^\top A\tau\tau^\top A^{-1}\tilde{J}\tilde{J}^\top A^{-1}\tau\tau^\top AR]\big) - \|R^\top\tilde{J}\|_F^2 \\ =\ &\operatorname{tr}(RR^\top A^2)\operatorname{tr}(\tilde{J}\tilde{J}^\top A^{-2}) + \|R^\top\tilde{J}\|_F^2 - 2\operatorname{tr}\big(R^\top((\tilde{J}\tilde{J}^\top)\odot I)R\big), \end{aligned}$$

where in the last step we have made use of Proposition 2 (with $\kappa = -2$) to evaluate the second moment. The part that depends on α is

$$\operatorname{tr}(RR^\top A^2)\operatorname{tr}(\tilde{J}\tilde{J}^\top A^{-2}) = \sum_{q\leqslant t}\sum_{r\leqslant t}\frac{\alpha_r^2}{\alpha_q^2}\|\mathcal{J}_{z_r}^{h_t}\nu_r\|^2\|\nu_q^\top\mathcal{J}_{\theta_q}^{z_q}\|^2 = \sum_{q\leqslant t}\sum_{r\leqslant t}\frac{\alpha_r^2}{\alpha_q^2}C_{qr}$$

where $C_{qr} \triangleq m_q n_r \triangleq \|\mathcal{J}_{z_r}^{h_t}\nu_r\|^2\|\nu_q^\top\mathcal{J}_{\theta_q}^{z_q}\|^2$. From the analysis in Appendix 4.C we know that this is minimal iff

$$e_k^\top A^2 C A^{-2}\vec{1} = e_k^\top A^{-2}C^\top A^2\vec{1},$$

where e_k is the kth column of the identity matrix and $\vec{1}$ is the vector of ones. Using the rank-one structure of C, we have

$$\alpha_k^2 m_k n^\top A^{-2}\vec{1} = \alpha_k^{-2} n_k m^\top A^2\vec{1},$$

which leads to the solution

$$\alpha_k^4 = \frac{n_k}{m_k}\frac{m^\top A^2\vec{1}}{n^\top A^{-2}\vec{1}} \propto \frac{n_k}{m_k} = \frac{\|\nu_k^\top\mathcal{J}_{\theta_k}^{z_k}\|^2}{\|\mathcal{J}_{z_k}^{h_t}\nu_k\|^2}.$$

4.C. Optimizing α given Q_0

Section 4.7.1 introduced the following optimization problem (Equation 4.7.4):

$$\alpha^\star = \operatorname*{argmin}_{\alpha>0}\sum_{r\leqslant T}\sum_{q\leqslant T}\frac{\alpha_r^2}{\alpha_q^2}C_{qr}$$

Here we analyze this problem in terms of a logarithmic parameterization $\alpha_i^2 = \exp(\zeta_i)$. The coefficients $\exp(\zeta_i)$ give rise to diagonal column- and row-scaling matrices Z, Z^{-1} with $Z_{ij} = \delta_{ij}\exp(\zeta_i)$. These matrices act on C to produce a modified matrix $\bar{C} = Z^{-1}CZ$, of which $V(Q)$ is the elementwise sum:

$$V(Q) = \sum_{r\leqslant T}\sum_{q\leqslant T}\exp(-\zeta_q)C_{qr}\exp(\zeta_r) = \vec{1}^\top Z^{-1}CZ\vec{1} = \vec{1}^\top\bar{C}\vec{1}.$$

By $\vec{1}$ we denote the vector of ones.

We will make use of the matrix differential

$$\frac{\mathrm{d}\bar{C}}{\mathrm{d}\zeta_k} = e_k e_k^\top \bar{C} - \bar{C} e_k e_k^\top$$

which measures the first-order change in $\bar{C}$ with respect to ζ_k. Here e_k is the kth column of the identity matrix. From this we get the derivative of $V(Q)$ with respect to ζ_k:

$$\frac{\mathrm{d}V(Q)}{\mathrm{d}\zeta_k} = \vec{1}^\top \frac{\mathrm{d}\bar{C}}{\mathrm{d}\zeta_k}\vec{1} = e_k^\top \bar{C}\vec{1} - \vec{1}^\top \bar{C} e_k.$$

The stationary points of $V(Q)$ satisfy $\bar{C}\vec{1} = \bar{C}^\top\vec{1}$, i.e. the modified matrix $\bar{C}$ has equal column and row sums.

Using the matrix differential $\frac{\mathrm{d}\bar{C}}{\mathrm{d}\zeta_k}$ twice, we find the elements of the Hessian H:

$$\begin{aligned}
H_{ij} &= \frac{\mathrm{d}}{\mathrm{d}\zeta_i}\frac{\mathrm{d}}{\mathrm{d}\zeta_j}\vec{1}^\top\bar{C}\vec{1} = \frac{\mathrm{d}}{\mathrm{d}\zeta_i}\vec{1}^\top\left(e_j e_j^\top\bar{C} - \bar{C}e_j e_j^\top\right)\vec{1} \\
&= \vec{1}^\top\left(e_j e_j^\top e_i e_i^\top\bar{C} - e_j e_j^\top\bar{C}e_i e_i^\top - e_i e_i^\top\bar{C}e_j e_j^\top + \bar{C}e_i e_i^\top e_j e_j^\top\right)\vec{1} \\
&= \delta_{ij} e_i^\top\bar{C}\vec{1} - e_j^\top\bar{C}e_i - e_i^\top\bar{C}e_j + \delta_{ij}\vec{1}^\top\bar{C}e_i
\end{aligned}$$

which in matrix form is

$$H = \mathrm{diag}(\bar{C}\vec{1}) - \bar{C} + \mathrm{diag}(\bar{C}^\top\vec{1}) - \bar{C}^\top = \mathrm{diag}((\bar{C}+\bar{C}^\top)\vec{1}) - (\bar{C}+\bar{C}^\top).$$

It is easily shown that the Hessian is positive semidefinite everywhere and hence $V(Q)$ is convex in ζ for all real vectors v:

$$\begin{aligned}
v^\top H v &= v^\top\,\mathrm{diag}((\bar{C}+\bar{C}^\top)\vec{1})v - v^\top(\bar{C}+\bar{C}^\top)v = \sum_{ij}(v_i^2 - v_i v_j)(\bar{C}+\bar{C}^\top)_{ij} \\
&= \frac{1}{2}\sum_{ij}(2v_i^2 - 2v_i v_j)(\bar{C}+\bar{C}^\top)_{ij} = \frac{1}{2}\sum_{ij}(v_i^2 + v_j^2 - 2v_i v_j)(\bar{C}+\bar{C}^\top)_{ij} \\
&= \frac{1}{2}\sum_{ij}(v_i - v_j)^2(\bar{C}+\bar{C}^\top)_{ij}.
\end{aligned}$$

As $(v_i - v_j)^2 \geqslant 0$ and $(\bar{C}+\bar{C}^\top)_{ij} \geqslant 0$ due to positivity of the entries of C, each term in the sum is nonnegative and therefore the whole sum is nonnegative. This implies H is positive semidefinite and hence $V(Q)$ is convex.

Given that $V(Q)$ is smooth and convex, its stationary points are global minimizers. In our experiments we solve for the stationary points by Newton's method, according to the update

$$\zeta \leftarrow \zeta - \eta(H + \lambda I)^{-1}(\bar{C} - \bar{C}^\top)\vec{1}$$

where η is a learning rate and λ is a damping factor on H, which is necessary because one of its eigenvalues is zero. In our experiments, we use $\eta = 1$ and $\lambda = 10^{-8}$.

4.D. Online optimization of α coefficients

This Appendix demonstrates how the incremental formulation of the optimization with respect to α from Section 4.7.1 (Equation 4.7.5) may be used to derive practical values for the γ, β coefficients. Recall that the optimization problem is defined in terms of a matrix $C^{(s)}$ that stands in for the unknown C. We will work with a naive choice that assumes future gradients and Jacobians are zero:

$$C^{(s)}_{qr} \triangleq \big\| \sum_{t=q}^{s} b_r^{(t)\top} Q_0 \big\|^2 \big\| Q_0^{-1} J_q \big\|_F^2.$$

Note that $\| \sum_{t=q}^{s} b_r^{(t)\top} Q_0\|^2$ is zero unless $q \leqslant s$ and $r \leqslant s$, and thus $C^{(s)}_{qr} = \mathbb{1}_{q\leqslant s}\mathbb{1}_{r\leqslant s} C^{(s)}_{qr}$. Using this property, we can rewrite the problem (Equation 4.7.5) as

$$\beta_s^\star, \gamma_s^\star = \operatorname*{argmin}_{\beta_s,\gamma_s} \sum_{r\leqslant s} \sum_{q\leqslant s} \frac{\alpha_r^2}{\alpha_q^2} C^{(s)}_{qr}.$$

Expanding

$$\frac{\alpha_r^2}{\alpha_q^2} = \frac{\beta_r^2\gamma_{r+1}^2 \cdots \gamma_T^2}{\beta_q^2\gamma_{q+1}^2 \cdots \gamma_T^2} = \frac{\beta_r^2\gamma_{r+1}^2 \cdots \gamma_q^2}{\beta_q^2\gamma_{q+1}^2 \cdots \gamma_r^2}$$

reveals that only terms with either $q = s$ or $r = s$ depend on β_s and/or γ_s, and thus

$$\beta_s^\star, \gamma_s^\star = \operatorname*{argmin}_{\beta_s,\gamma_s} \frac{\beta_s^2}{\gamma_s^2} \sum_{q<s} \beta_q^{-2}\gamma_{q+1}^{-2} \cdots \gamma_{s-1}^{-2} C^{(s)}_{qs} + \frac{\gamma_s^2}{\beta_s^2} \sum_{r<s} \beta_r^2\gamma_{r+1}^2 \cdots \gamma_{s-1}^2 C^{(s)}_{sr}.$$

Note that β_s and γ_s appear through the single degree of freedom β_s^2/γ_s^2, and by differentiation we find the stationary points

$$\frac{\gamma_s^4}{\beta_s^4} = \frac{\sum_{q<s} \beta_q^{-2}\gamma_{q+1}^{-2} \cdots \gamma_{s-1}^{-2} C^{(s)}_{qs}}{\sum_{r<s} \beta_r^2\gamma_{r+1}^2 \cdots \gamma_{s-1}^2 C^{(s)}_{sr}}$$

From our definition of $C^{(s)}$ we have that

$$C^{(s)}_{qs} = \|b_s^{(s)\top} Q_0\|^2 \|Q_0^{-1} J_q\|_F^2 \quad \text{and} \quad C^{(s)}_{sr} = \|b_r^{(s)\top} Q_0\|^2 \|Q_0^{-1} J_s\|_F^2,$$

which leads to the natural solution

$$\beta_s^4 = \frac{\|Q_0^{-1} J_s\|_F^2}{\|b_s^{(s)\top} Q_0\|^2} \quad \text{and} \quad \gamma_s^4 = \frac{\sum_{q<s} \|\beta_q^{-1}\gamma_{q+1}^{-1} \cdots \gamma_{s-1}^{-1} Q_0^{-1} J_q\|_F^2}{\sum_{r<s} \|\beta_r \gamma_{r+1} \cdots \gamma_{s-1} b_r^{(s)\top} Q_0\|^2}.$$

It can be shown that the above solution can be expressed in terms of ratios of expectations of familiar quantities:

$$\beta_s^4 = \frac{\mathbb{E}_u \|u_s^\top Q_0^{-1} \mathcal{J}_{\theta_s}^{z_s}\|_F^2}{\mathbb{E}_u \|\mathcal{J}_{h_s}^{L_s} \mathcal{J}_{z_s}^{h_s} Q_0 u_s\|^2} \quad \text{and} \quad \gamma_s^4 = \frac{\mathbb{E}_u \|\tilde{w}_{s-1}\|_F^2}{\mathbb{E}_u \|\mathcal{J}_{h_s}^{L_s} \mathcal{J}_{h_{s-1}}^{h_s} \tilde{h}_{s-1}\|^2}$$

These coefficients are closely related to those of GIR as derived in Section 4.5.2. In fact, had we defined

$$C_{qr}^{(s)} \triangleq \|\mathcal{J}_{z_r}^{h_s} Q_0\|^2 \|Q_0^{-1} J_q\|_F^2,$$

the projection onto $\mathcal{J}_{z_s}^{L_s}$ would disappear from the coefficients, making the similarity even more striking. However, this choice is not consistent with our objective of minimizing the variance $V(Q)$ of the total gradient estimate.

Note that we were able to solve for the coefficients in closed form thanks to the property $C_{qr}^{(s)} = \mathbb{1}_{q \leqslant s} \mathbb{1}_{r \leqslant s} C_{qr}^{(s)}$. In general, solving Equation 4.7.5 involves joint optimization of $\beta_{\geqslant s}, \gamma_{\geqslant s}$, which requires a numerical approach similar to the one described in Appendix 4.C. Moreover, β_s and γ_s will in general be independent parameters.

4.E. Minimization of the Product of Traces

Minimizing the total variance of our estimators involves minimizing a product of traces by choice of a noise-shaping matrix. Here we characterize the optimal choice of such a matrix in a general setting.

Definition 4. Define $c(A) = \operatorname{tr}(XA) \operatorname{tr}(YA^{-1})$ for PD matrices X and Y.

The goal of this section will be to prove the following theorem.

Theorem 5. *A PD matrix A is a global minimizer of $c(A)$ over the set of PD matrices if and only if*

$$XA = \gamma A^{-1} Y$$

for some scalar $\gamma > 0$.

Note that a similar result to Theorem 5 one was used implicitly by Ollivier et al. (2015), but wasn't given rigorous justification. It is relatively easy to characterize the critical points of $c(A)$, but proving that any critical point is a global minimizer is much more involved. It would be tempting to use convexity to prove such a result but unfortunately $c(A)$ is not convex in general.

We begin by stating and proving some basic technical claims.

Claim 6. *Let U be a matrix and V be a PD matrix with $V = CC^\top$ for some C. Then the eigenvalues of UV are the same as the eigenvalues of $C^\top UC$.*

PROOF. Observe that

$$C^\top (UV) C^{-\top} = C^\top (UCC^\top) C^{-\top} = C^\top UC.$$

Thus UV is similar to the matrix $C^\top UC$ and so has the same eigenvalues. □

Corollary 7. *If U and V are PD matrices then they have all positive eigenvalues and $\operatorname{tr}(UV) > 0$.*

Claim 8. *A is a critical point of c if and only if $XA = \gamma A^{-1} Y$ for some $\gamma > 0$.*

PROOF. Differentiating $c(A)$ with respect to A, we find

$$\begin{aligned}\frac{dc(A)}{dA} &= \frac{d}{dA}\operatorname{tr}(XA)\operatorname{tr}(YA^{-1}) \\ &= \operatorname{tr}(YA^{-1})\frac{d}{dA}\operatorname{tr}(XA) + \operatorname{tr}(XA)\frac{d}{dA}\operatorname{tr}(YA^{-1}) \\ &= \operatorname{tr}(YA^{-1})X^{\top} - \operatorname{tr}(XA)A^{-\top}Y^{\top}A^{-\top}.\end{aligned}$$

Setting this to zero and rearranging terms gives

$$XA = \frac{\operatorname{tr}(XA)}{\operatorname{tr}(A^{-1}Y)}A^{-1}Y.$$

Because A, A^{-1}, X, and Y are all PD matrices, and the trace of a product of PD matrices is positive by the previous claim, the result follows. □

Claim 9. *$c(A) = \operatorname{tr}((XY)^{1/2})^2$ for critical points A, where $(XY)^{1/2}$ is the (unique) positive square root of XY.*

PROOF. Let A be a critical point. By Claim 8 we have $XA = \gamma A^{-1}Y$ for some $\gamma > 0$. This implies that $X = \gamma A^{-1}YA^{-1}$ and thus $XY = \gamma A^{-1}YA^{-1}Y = \gamma(A^{-1}Y)^2$. Because A^{-1} and Y are PD matrices we have by Claim 6 that $A^{-1}Y$ has all positive eigenvalues. Thus $A^{-1}Y = \gamma^{-1/2}(XY)^{1/2}$ where $(XY)^{1/2}$ is the (unique) positive square root of XY. We also have that $Y = \gamma^{-1}AXA$ which implies that $XY = \gamma^{-1}(XA)^2$, and so by a similar argument to the one above we have that $XA = \gamma^{1/2}(XY)^{1/2}$. Thus

$$c(A) = \operatorname{tr}(XA)\operatorname{tr}(YA^{-1}) = \operatorname{tr}(\gamma^{1/2}(XY)^{1/2})\operatorname{tr}(\gamma^{-1/2}(XY)^{1/2}) = \operatorname{tr}((XY)^{1/2})^2.$$

□

Observation. Note that $\operatorname{tr}((XY)^{1/2})^2$ depends only on the eigenvalues of XY and so for any other matrix V with the same eigenvalues $\operatorname{tr}(V^{1/2})^2$ would also give us the value of $c(A)$ at critical points. By Claim 6 such choices include $X^{1/2}YX^{1/2}$ and $Y^{1/2}XY^{1/2}$.

Definition 10. Let $\lambda_{\min}(V)$ denote the minimum eigenvalue of V.

Note that we may restrict our analysis of c to the following domain:

$$\mathcal{A}_1 = \{A : A \text{ is PD and } \lambda_{\min}(A) = 1\}.$$

This is because $c(\alpha A) = \operatorname{tr}(X(\alpha A))\operatorname{tr}(Y(\alpha A)^{-1}) = \alpha\operatorname{tr}(XA)\alpha^{-1}\operatorname{tr}(YA^{-1}) = c(A)$, and so we can always replace A with $A/\lambda_{\min}(A) \in \mathcal{A}_1$ without changing the objective function value. (Note that $\lambda_{\min}(A) > 0$ since A is PD, so the new matrix $A/\lambda_{\min}(A)$ remains PD.)

The remainder of this section will be devoted to showing that c, when restricted to $\mathcal{A}_1$, attains its minimum on that set. Combining this with the fact that c is continuously differentiable on the (larger) set of PD matrices, we will thus have that some critical point is a global minimizer of c on the set of all PD matrices.

And since all critical points have the same objective function value by Claim 9, it will follow that all critical points are global minimizers. And so by Claim 8, we will have that A is a global minimizer if and only if

$$XA = \gamma A^{-1} Y$$

for some $\gamma > 0$.

Claim 11. *Let A be some PD matrix with eigendecomposition given by $A = U \operatorname{diag}(d) U^\top$. Then we have*

$$c(A) = \Big(\sum_i d_i (u_i^\top X u_i)\Big)\Big(\sum_i d_i^{-1} (u_i^\top Y u_i)\Big),$$

where u_i is the i-th column of U (i.e. the i-th eigenvector of A) and d_i is the i-th entry of the vector d.

PROOF. Observe that

$$\operatorname{tr}(XA) = \operatorname{tr}(XU \operatorname{diag}(d) U^\top) = \operatorname{tr}(U^\top X U \operatorname{diag}(d)).$$

Noting that the i-th diagonal element of $U^\top X U$ is $u_i^\top X u_i$, so that the i-th diagonal element of $U^\top X U \operatorname{diag}(d)$ is $d_i (u_i^\top X u_i)$, it follows that

$$\operatorname{tr}(XA) = \sum_i d_i (u_i^\top X u_i).$$

Observing that $A^{-1} = U \operatorname{diag}(d)^{-1} U^\top$ so that $A^{-1} U = U \operatorname{diag}(d)^{-1}$, and that the i-th diagonal element of $\operatorname{diag}(d)^{-1}$ is just d_i^{-1} we can apply a similar argument to the above to show that

$$\operatorname{tr}(YA^{-1}) = \sum_i d_i^{-1} (u_i^\top Y u_i).$$

Combining these equation equations establishes the claim. □

Claim 12. *$\mathcal{A}_1$ is a closed set.*

PROOF. Note that $\lambda_{\min}(A)$ is a continuous function of A, and $\{1\}$ is a closed set. Morever the set of PSD matrices is a closed set. We therefore have that the intersection of the preimage of $\lambda_{\min}(A)$ on $\{1\}$ and the set of PSD matrices is a closed set (i.e. the set $\{A : A \text{ is PSD and } \lambda_{\min}(A) = 1\}$) is closed. But this set is precisely $\mathcal{A}_1$ since any PSD matrix with $\lambda_{\min}(A) = 1$ is also clearly PD. □

Definition 13. A function $f : S \to \mathbb{R}$ defined on a set $S \subset \mathbb{R}^n$ is called "coercive" if we have

$$f(v) \to \infty \qquad \text{as} \qquad \|v\| \to \infty.$$

The following is a standard result in finite dimensional analysis Heath, 2018, Chapter 6:

Theorem 14. *If $f : S \to \mathbb{R}$ is coercive and continuous and $S \subset \mathbb{R}^n$ is a closed set then f obtains in minimum on S.*

Note that the theorem applies equally to the space of finite dimensional real-valued matrices where the norm is any valid matrix norm, including the standard spectral norm (which our notation will assume).

Claim 15. *$c(A)$ is coercive on the set $\mathcal{A}_1$.*

PROOF. Let $A \in \mathcal{A}_1$ with eigendecomposition given by $A = U \operatorname{diag}(d) U^\top$. By Claim 11 we have that

$$c(A) = \Big(\sum_i d_i (u_i^\top X u_i)\Big)\Big(\sum_i d_i^{-1} (u_i^\top Y u_i)\Big).$$

Since $A \in \mathcal{A}_1$ we can assume without loss of generality that $d_1 = 1$. We can also assume without loss of generality that d_2 is the largest eigenvalue of A so that $d_2 = \|A\|$.

Because A, X and Y are all PD we have that $u_i^\top X u_i \geqslant \lambda_{\min}(X) > 0$ and $u_i^\top Y u_i \geqslant \lambda_{\min}(Y) > 0$ for each i. And thus

$$\begin{aligned} c(A) &\geqslant \lambda_{\min}(X)\lambda_{\min}(Y)\Big(1 + \|A\| + \sum_{i>2} d_i\Big)\Big(1 + \|A\|^{-1} + \sum_{i>2} d_i^{-1}\Big) \\ &\geqslant \lambda_{\min}(X)\lambda_{\min}(Y)(1 + \|A\|)(1 + \|A\|^{-1}). \end{aligned}$$

Clearly this goes to infinity as $\|A\|$ does, which establishes the claim. □

Claim 16. *$c(A)$ attains its minimum on the set $\mathcal{A}_1$.*

PROOF. This follows directly from Claims 12 and 15 and Theorem 14. □

4.F. Estimating B online

The optimal Q_0 derived in Section 4.7.1 depends on the matrix B (Equation 4.7.7) which is unknown due to its dependence on future coefficients and gradients. To obtain a practical algorithm, we must approximate it online. We will consider B to be the final element in a sequence $B^{(1)} \dots B^{(T)}$ of matrices that accumulate information observed so far:

$$B^{(k)} = \sum_{s \leqslant k} \sum_{t \leqslant k} \Big(\sum_{q=1}^{\min(s,t)} \alpha_q^{-2} \|a_q\|^2 \Big) \Big(\sum_{r=1}^{\min(s,t)} \alpha_r^2 b_r^{(s)} b_r^{(t)\top} \Big).$$

The remainder of this section develops an online algorithm that produces an unbiased estimate of $B^{(s)}$ at each step s. Although this will not yield an unbiased estimate of $B^{(T)}$ until the final time step T, to the extent that $B^{(s)}$ is stationary we may use its intermediate estimates $B^{(s)}$ for $s < T$ as approximations to $B^{(T)}$.

First, we factorize the sums over q and r using the now-familiar random projections onto independent temporal noise vectors σ and τ:

$$\sum_{q=1}^{\min(s,t)} \alpha_q^{-2} \|a_q\|^2 = \mathbb{E}_\sigma \Big[\Big(\sum_{q \leqslant s} \sigma_q \alpha_q^{-1} \|a_q\| \Big) \Big(\sum_{q \leqslant t} \sigma_q \alpha_q^{-1} \|a_q\| \Big) \Big]$$

and

$$\sum_{r=1}^{\min(s,t)} \alpha_r^2 b_r^{(s)} b_r^{(t)\top} = \mathbb{E}_\tau\Big[\Big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)}\Big)\Big(\sum_{r\leqslant t} \tau_r \alpha_r b_r^{(t)}\Big)^\top\Big].$$

By doing so we have broken up the dependency on $\min(s,t)$ into separate factors. Defining $\tilde{a}_s = \sum_{q\leqslant s} \sigma_q \alpha_q^{-1} \|a_q\|$, we may now express $B^{(k)}$ as

$$\begin{aligned} B^{(k)} &= \mathbb{E}_{\sigma,\tau}\Big[\sum_{s\leqslant k}\sum_{t\leqslant k} \tilde{a}_s \tilde{a}_t \Big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)}\Big)\Big(\sum_{r\leqslant t} \tau_r \alpha_r b_r^{(t)}\Big)^\top\Big] \\ &= \mathbb{E}_{\sigma,\tau}\Big[\Big(\sum_{s\leqslant k} \tilde{a}_s \big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)}\big)\Big)\Big(\sum_{t\leqslant k} \tilde{a}_t \big(\sum_{r\leqslant t} \tau_r \alpha_r b_r^{(t)}\big)^\top\Big)\Big] \\ &= \mathbb{E}_{\sigma,\tau}[m_k m_k^\top], \end{aligned}$$

the expectation of a rank-one estimator given by the outer product of the vector

$$m_k \triangleq \sum_{s\leqslant k} \tilde{a}_s \Big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)}\Big)$$

with itself. As this vector has zero mean, $B^{(k)}$ is its covariance.

The scalar $\tilde{a}_s$ is readily accumulated online, but the vector $\sum_{r\leqslant s} \tau_q \alpha_r b_r^{(s)}$ requires approximate forward differentiation. We can estimate m_k by

$$\tilde{m}_k \triangleq \sum_{s\leqslant k} \tilde{a}_s \Big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)\top} \nu_r\Big)\Big(\sum_{r\leqslant s} \nu_r\Big)$$

which can be computed efficiently according to the recursions

$$\begin{aligned} \tilde{a}_t &= \gamma_t^{-1} \tilde{a}_{t-1} + \sigma_t \beta_t^{-1} \|a_t\| \\ \tilde{h}_t &= \eta_t \gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1} + \zeta_t \tau_t \beta_t \mathcal{J}_{z_t}^{h_t} \nu_t \\ \tilde{v}_t &= \eta_t^{-1} \tilde{v}_{t-1} + \zeta_t^{-1} \nu_t \\ \tilde{m}_t &= \tilde{m}_{k-1} + \tilde{a}_t \mathcal{J}_{h_t}^{L_t} \tilde{h}_t \tilde{v}_t. \end{aligned}$$

The coefficients η_t, ζ_t can be used to reduce the variance of $\tilde{h}_t \tilde{v}_t^\top$, e.g. by the GIR choice $\eta_t^2 = \|\tilde{v}_{t-1}\| / \|\gamma_t \mathcal{J}_{h_{t-1}}^{h_t} \tilde{h}_{t-1}\|$, $\zeta_t^2 = \|\nu_t\| / \|\tau_t \beta_t \mathcal{J}_{z_t}^{h_t} \nu_t\|$.

Although $\mathbb{E}_\nu[\tilde{m}_k] = m_k$ (i.e. $\tilde{m}_k$ is an unbiased estimator of m_k), $\mathbb{E}_\nu[\tilde{m}_k \tilde{m}_k^\top] \neq \mathbb{E}_\nu[\tilde{m}_k]\mathbb{E}_\nu[\tilde{m}_k^\top]$ and therefore $\tilde{m}_k \tilde{m}_k^\top$ is not an unbiased estimator of $B^{(k)}$. To estimate $B^{(k)}$, we require a replication $\tilde{n}_k$ of $\tilde{m}_k$ with independent spatial noise μ in place of ν:

$$\tilde{n}_k \triangleq \sum_{s\leqslant k} \tilde{a}_s \Big(\sum_{r\leqslant s} \tau_r \alpha_r b_r^{(s)\top} \mu_r\Big)\Big(\sum_{r\leqslant s} \mu_r\Big)$$

computed by similar recursions as $\tilde{m}_k$. Now

$$\mathbb{E}_{\sigma,\tau,\nu,\mu}[\tilde{m}_k \tilde{n}_k^\top] = \mathbb{E}_{\sigma,\tau}\big[\mathbb{E}_\nu[\tilde{m}_k]\mathbb{E}_\mu[\tilde{n}_k^\top]\big] = \mathbb{E}_{\sigma,\tau}[m_k m_k^\top] = B^{(k)}.$$

It should be noted that although $B^{(k)}$ is symmetric PSD, the estimates $\tilde{m}_k\tilde{n}_k^\top$ are not. Symmetry may however be restored by use of the estimator $1/2(\tilde{m}_k\tilde{n}_k^\top + \tilde{n}_k\tilde{m}_k^\top)$.

4.G. Hyperparameter Settings for Variance Reduction Experiments

The following table lists the hyperparameter settings used for the experiments in Section 4.7.2:

Q_0	α	Learning rate	Momentum	$\bar{B}$ decay	$\bar{B}$ dampening
identity	GIR	0.005	0.8		
identity	ours	0.005	0.5		
ours	GIR	0.005	0.5	0.9	0.008
ours	ours	0.003	0.8	0.9	0.005

These settings were found by grid search on learning rate in {0.001,0.003,0.005,0.007,0.009}, momentum in {0.5,0.8}, $\bar{B}$ decay rate in $\{0.8, 0.9, 0.95\}$ and $\bar{B}$ dampening coefficient in $\{5\times10^{-3}, 5\times10^{-4}, 5\times10^{-5}\}$.

4.H. Variance of Preactivation-Space Projection

This appendix explores the variance of the estimator from Section 4.8 and its relationship to the variance of the usual total gradient estimator from Section 4.6.4 (Equation 4.6.4). The former is given by Equation 4.8.2:

$$\operatorname{vec}\Big(\sum_{t\leqslant T} \bar{B}^{(t)\top}\bar{Q}\tau\tau^\top\bar{Q}^{-1}\bar{S}^{(t)}\bar{J}\Big)$$

with matrices $\bar{B}^{(t)\top} = \begin{pmatrix} b_1^{(t)} & \cdots & b_T^{(t)}\end{pmatrix}, \bar{Q} = \operatorname{diag}(\alpha), \bar{S}_{ij}^{(t)} = \delta_{ij}\mathbb{1}_{i\geqslant t}, \bar{J} = \begin{pmatrix} a_1 & \cdots & a_T\end{pmatrix}^\top$ that mirror similarly-named quantities from Section 4.6.4.

Noting that for a matrix X,

$$\begin{aligned}\operatorname{tr}\big(\operatorname{Var}[\operatorname{vec}(X)]\big) &= \mathbb{E}\big[\|\operatorname{vec}(X)\|^2\big] - \big\|\mathbb{E}[\operatorname{vec}(X)]\big\|^2 \\ &= \mathbb{E}\big[\|X\|_F^2\big] - \big\|\mathbb{E}[X]\big\|_F^2 = \operatorname{tr}\big(\mathbb{E}[XX^\top]\big) - \big\|\mathbb{E}[X]\big\|_F^2\end{aligned}$$

we have by Proposition 2 (with $\kappa = 0$) that

$$\begin{aligned}
&\operatorname{tr}\Big(\operatorname{Var}\Big[\operatorname{vec}\Big(\sum_{t\leqslant T} \bar{B}^{(t)\top}\bar{Q}\tau\tau^\top\bar{Q}^{-1}\bar{S}^{(t)}\bar{J}\Big)\Big]\Big)\\
&= \sum_{s\leqslant T}\sum_{t\leqslant T} \operatorname{tr}\big(\mathbb{E}[\bar{J}^\top\bar{S}^{(s)}\bar{Q}^{-\top}\tau\tau^\top\bar{Q}^\top\bar{B}^{(s)}\bar{B}^{(t)\top}\bar{Q}\tau\tau^\top\bar{Q}^{-1}\bar{S}^{(t)}\bar{J}]\big) - \|\mathcal{J}_\theta^L\|_F^2\\
&= \sum_{s\leqslant T}\sum_{t\leqslant T} \operatorname{tr}(\bar{B}^{(s)}\bar{B}^{(t)\top}\bar{Q}\bar{Q}^\top)\operatorname{tr}(\bar{S}^{(t)}\bar{J}\bar{J}^\top\bar{S}^{(s)}(\bar{Q}\bar{Q}^\top)^{-1}) + \|\mathcal{J}_\theta^L\|_F^2. \qquad (4.H.1)
\end{aligned}$$

The variance of the usual estimator (Section 4.6.4) is given by Equation 4.6.3:

$$\begin{aligned}
&\operatorname{tr}\Big(\operatorname{Var}\big[\sum_{t\leqslant T} b^{(t)\top}Quu^\top Q^{-1}S^{(t)}J\big]\Big)\\
&= \sum_{s\leqslant T}\sum_{t\leqslant T} \operatorname{tr}(b^{(s)}b^{(t)\top}QQ^\top)\operatorname{tr}(S^{(t)}JJ^\top S^{(s)}(QQ^\top)^{-1}) + \|\mathcal{J}_\theta^L\|_F^2. \qquad (4.H.2)
\end{aligned}$$

We will now relate Equation 4.H.2 to Equation 4.H.1 using the following observations:

$$b^{(t)\top} = \operatorname{vec}(\bar{B}^{(t)}), Q = \bar{Q}\otimes Q_0, S^{(t)} = \bar{S}^{(t)}\otimes I, \text{ and } JJ^\top = \bar{J}\bar{J}^\top\otimes I.$$

Thus $\operatorname{tr}(b^{(s)}b^{(t)\top}QQ^\top)$ can be written

$$\begin{aligned}
\operatorname{tr}(b^{(s)}b^{(t)\top}QQ^\top) &= \operatorname{vec}(\bar{B}^{(t)})^\top(\bar{Q}\bar{Q}^\top\otimes Q_0Q_0^\top)\operatorname{vec}(\bar{B}^{(s)})\\
&= \operatorname{vec}(\bar{B}^{(t)})^\top\operatorname{vec}(Q_0Q_0^\top\bar{B}^{(s)\top}\bar{Q}\bar{Q}^\top)\\
&= \operatorname{tr}(\bar{B}^{(t)}Q_0Q_0^\top\bar{B}^{(s)\top}\bar{Q}\bar{Q}^\top)
\end{aligned}$$

and $\operatorname{tr}(S^{(t)}JJ^\top S^{(s)}(QQ^\top)^{-1})$ can be written

$$\begin{aligned}
\operatorname{tr}(S^{(t)}JJ^\top S^{(s)}(QQ^\top)^{-1}) &= \operatorname{tr}\big((\bar{S}^{(t)}\otimes I)(\bar{J}\bar{J}^\top\otimes I)(\bar{S}^{(s)}\otimes I)(\bar{Q}^{-2}\otimes(Q_0Q_0^\top)^{-1})\big)\\
&= \operatorname{tr}\big(\bar{S}^{(t)}\bar{J}\bar{J}^\top\bar{S}^{(s)}(\bar{Q}\bar{Q}^\top)^{-1}\big)\otimes(Q_0Q_0^\top)^{-1}\big)\\
&= \operatorname{tr}\big(\bar{S}^{(t)}\bar{J}\bar{J}^\top\bar{S}^{(s)}(\bar{Q}\bar{Q}^\top)^{-1}\big)\operatorname{tr}\big((Q_0Q_0^\top)^{-1}\big).
\end{aligned}$$

This results in the following expression for the dominant term of the variance in Equation 4.H.2:

$$\sum_{s\leqslant T}\sum_{t\leqslant T}\operatorname{tr}\big(\bar{B}^{(t)}Q_0Q_0^\top\bar{B}^{(s)\top}\bar{Q}\bar{Q}^\top\big)\operatorname{tr}\big(\bar{S}^{(t)}\bar{J}\bar{J}^\top\bar{S}^{(s)}(\bar{Q}\bar{Q}^\top)^{-1}\big)\operatorname{tr}\big((Q_0Q_0^\top)^{-1}\big).$$

Chapter 5

Best Response Shaping

Milad Aghajohari[*†] Tim Cooijmans[*†] Juan Augustin Duque[†]
Shunichi Akatsuka[‡] Aaron Courville[†]

[*] Equal contribution [†] Mila & Université de Montréal [‡] Hitachi

We investigate the challenge of multi-agent deep reinforcement learning in partially competitive environments, where traditional methods struggle to foster non-exploitable cooperation. LOLA and POLA agents learn non-exploitable cooperative policies by differentiation through look-ahead optimization steps of their opponent. However, there is a key limitation in these techniques as they are susceptible to exploitation by further optimization. In response, we introduce a novel approach, Best Response Shaping (BRS), which differentiates through an opponent approximating the best response, termed the "detective". To condition the detective on the agent's policy for complex games we propose a state-aware differentiable conditioning mechanism, facilitated by a question answering (QA) method that extracts a representation of the agent based on its behaviour on specific environment states. To empirically validate our method, we showcase its enhanced performance against a Monte Carlo Tree Search (MCTS) opponent, which serves as an approximation to the best response in the Coin Game. This work expands the applicability of multi-agent RL in partially competitive environments and provides a new pathway towards achieving improved social welfare in general sum games.

Prologue

Best Response Shaping is one of two projects that came out of our multi-agent learning group. We spent several years developing an understanding of the problem of multi-agent coordination and the proposals in the literature. We were especially interested in learning reciprocity, which naive learning generally does not do. Milad and Aaron came up with the idea of training against the best response as a function of the current policy, taking account of the gradient through the best response. The agent thus tries to choose a policy that will incentivize the opponent to cooperate – tit-for-tat is one such policy. We iterated over many months to find an effective way to approximate that ideal.

Initially, we explored the possibility of using Monte-Carlo Tree Search to approximate the best response. I implemented Monte-Carlo Tree Search in JAX, in such a way that we could use REINFORCE to (approximately) propagate gradients through it with respect to the agent's policy. Unfortunately, the variance on these gradients turned out to be too high. I also briefly investigated a Neural ODE approach, which would have allowed us to explicitly find the best response through optimization, without having to differentiate through this optimization trajectory. Milad experimented with a variety of policy fingerprinting approaches, and ultimately found success by observing behavior versus a random agent. Finally, I took responsibility for understanding and training the POLA baseline, helping Zhao et al. (2022a) find and fix several bugs in their implementation. The paper was written primarily by Milad, Juan and Shunichi, with some proofreading and math checks by myself.

5.1. Introduction

Reinforcement Learning (RL) algorithms have enabled agents to perform well in complex high-dimensional games like Go (Silver et al., 2016) and StarCraft (Vinyals et al., 2019). The end goal of RL is to train agents that can help humans solve challenging problems. Inevitably, these agents will need to integrate in real-life scenarios that require interacting with humans and other learning agents. While multi-agent RL training shines in fully cooperative or fully competitive environments, it often fails to find non-exploitable cooperation in partially competitive environments. One such example is the failure of multi-agent RL (MARL) agents to learn non-exploitable cooperative policies like tit-for-tat (TFT) in the Iterated Prisoner's Dilemma (IPD) (Foerster et al., 2018b).

Despite the toy-ish character of common general-sum games such as IPD, these sorts of problems are ubiquitous in both society and nature. Consider a scenario where two countries (agents), strive to maximize their industrial output while also ensuring a suitable climate for production by limiting carbon emissions. On the one hand, each country (agent) would like to see the other country fulfill it's obligations to limit carbon emissions. Yet on the other hand, each one is motivated to emit more carbon themselves to achieve higher industrial yields. An effective climate treaty would compel each country – likely through the threat of penalties – to abide by the agreed limits to carbon emissions. If these agents fail to develop such tit-for-tat-like strategies they will likely converge to an unfortunate mutual escalation of consumption and carbon emission.

Foerster et al. (2018b) proposed Learning with Opponent-Learning Awareness (LOLA), an algorithm that successfully learns TFT behavior in the IPD setting by differentiating through an assumed single naive gradient step taken by the opponent. Building upon this, Zhao et al. (2022b) introduced proximal LOLA (POLA), which further enhances LOLA

by assuming a proximal policy update for the opponent. This improvement allows for the training of Neural Network (NN) policies in more complex games, such as the Coin Game (Foerster et al., 2018b). To the best of our knowledge, POLA is the only method that reliably trains non-exploitable cooperative agents in the Coin Game.

Despite its success on the Coin Game, POLA has its limitations. POLA assumes a limited number of look-ahead optimization steps by the opponent, which renders it vulnerable to exploitation by opponents engaging in additional optimization. Additionally, this constraint may hinder POLA's scalability, as the agent cannot differentiate through additional stages of opponent optimization. In particular, our analysis of POLA agents trained on the Coin Game demonstrates that POLA is susceptible to exploitation by the best response opponent. When the opponent is specifically trained to maximize its own return against a fixed policy trained by POLA, the first exploits the former.

In this paper, we present a novel approach called Best Response Shaping (BRS). Our method is based on the construction of an opponent that approximates the best response policy against any agent. We refer to this opponent as the "detective." The overall concept is depicted in Figure 5.1: the detective undergoes training against agents sampled from a diverse distribution. To train the agent, we differentiate through the detective opponent. Unlike approaches such as LOLA and POLA, which assume few look-ahead optimization steps, our method relies on the detective issuing the best response to the current agent through policy conditioning.

The detective conditions on an embedding of the agent's policy that effectively captures its behavior across various states of the environment. Extracting such a representation is a non-trivial task (Harb et al., 2020). A straightforward approach of concatenating all policy parameters into a single representation results in a loss of architectural information and requires a large number of samples to be effective. Alternatively, conditioning the representation on the agent's behavior in specific query states, as done in Harb et al. (2020), can be attempted. However, learning these query states to enable generalization of the agent's behavior is, by itself, a difficult problem.

To address this, we introduce a question-answering (QA) mechanism dependent on the current state of the environment, which serves as a means to extract a representation of the agent policy. The detective evaluates the agent's policy (answers) based on specific environment states (questions) given the current state.

BRS gives us a strong agent against the best response opponent. However, it also has a tendency to produce agents that exploit policies that exhibit excessive levels of cooperation. If more cooperative behavior is desired, we propose self-play with reward sharing as a regularization technique. Throughout the paper, we refer to this approach as Best Response Shaping with Self-Play (BRS-SP).

We empirically validate our method on Iterated Prisoner's Dilemma (IPD) and the Coin Game. Given the dependency on the opponent's policy for an agent's outcomes, it is not always straightforward to evaluate and compare policies of different agents in games. This is especially true in non-zero-sum games that exhibit both cooperative and competitive aspects. In this paper, we advocate that a reasonable point of comparison is the agent's outcome when facing a best response opponent, which we approximate by Monte Carlo Tree Search (MCTS). We show that while the MCTS best response opponents learn to exploit POLA agents, they also learn to cooperate with our BRS agents. To the best of our knowledge, Best Response Shaping is the only method that enables training non-exploitable cooperative agents parameterized by neural networks.

Main Contributions. We summarize our main contributions below:

- We identify that POLA agents are exploited by the best response opponent, approximated by the Monte Carlo Tree Search (MCTS).
- To address this vulnerability, we introduce the BRS method, which trains an agent by differentiating through an opponent approximating the best response (referred to as the 'detective opponent'). We empirically validate our method and demonstrate that agents trained with BRS are not exploited by the MCTS as shown in Figure 5.3.
- Additionally, we propose a state-aware differentiable conditioning mechanism for the detective opponent, enabling it to condition on the agent's policy.

5.2. Background

5.2.1. Multi Agent Reinforcement Learning

An N-agent Markov Games is denoted by a tuple $(N, \mathcal{S}, \{\mathcal{A}^i\}_{i=1}^N, \mathbb{P}, \{r^i\}_{i=1}^N, \gamma)$. Here, N represents the number of agents, $\mathcal{S}$ the state space of the environment, and $\mathcal{A} := \mathcal{A}^1 \times \cdots \times \mathcal{A}^N$ the set of actions for each agent. Transition probabilities are denoted by $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ and the reward function by $r^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Lastly, $\gamma \in [0,1]$ is the discount factor. In a multi-agent reinforcement learning problem each agent attempts to maximize their return $R^i = \sum_{t=0}^{\infty} \gamma^t r_t^i$. The policy of agent i is denoted by $\pi_{\theta_i}^i$ where θ_i are policy parameters. In Deep RL these policies are neural networks. These policies will be trained via gradient estimators such as REINFORCE (Sutton et al., 1999).

5.2.2. Social Dilemmas and the Iterated Prisoner's Dilemma

In the context of general sum games, social dilemmas emerge when individual agents striving to optimize their personal rewards inadvertently undermine the collective outcome or social welfare. This phenomenon is most distinct when the collective result is inferior to the outcome that could have been achieved through full cooperation. Theoretical

studies, such as the Prisoner's Dilemma, illustrate scenarios where each participant, though individually better off confessing, collectively achieves a lower reward compared to remaining silent.

However, in the Iterated Prisoner's Dilemma (IPD), unconditional defection ceases to be the dominant strategy. For instance, against an opponent following a tit-for-tat (TFT) strategy, perpetual cooperation results in a higher return for the agent. It might be expected that MARL, designed to maximize each agent's return, would discover the TFT strategy, as it enhances both collective and individual returns, and provides no incentive for policy change, embodying a Nash Equilibrium. Yet, empirical observations reveal that standard RL agents, trained to maximize their own return, typically converge to unconditional defection.

This exemplifies one of the key challenges of multi-agent RL in general sum games: during training, agents often neglect the fact that other agents are also in the process of learning. To address this issue, and if social welfare is the primary consideration, one could share the rewards among the agents during training. For instance, training both agents in an IPD setup to maximize the collective return would lead to a constant cooperation. However, this approach is inadequate if the goal is to foster reciprocation-based cooperation. A policy is sought that incites the opponent to cooperate in order to maximize their own return. While TFT is one such policy, manually designing a similar TFT policies in other domains is neither desirable nor feasible, underscoring the necessity to develop novel training algorithms that can discover these policies.

5.3. Related Work

LOLA (Foerster et al., 2018b) attempts to shape the opponent by taking the gradient of the value with respect to a one-step look ahead of the opponent's parameters. Instead of considering the expected return under the current policy parameter pair, $V^1(\theta_i^1, \theta_i^2)$, LOLA optimizes $V^1(\theta_i^1, \theta_i^2 + \Delta\theta_i^2)$ where $\Delta\theta_i^2$ denotes a naive learning step of the opponent. To make a gradient calculation of the update $\Delta\theta_i^2$, LOLA considers the surrogate value given by the first order Taylor approximation of $V^1(\theta_i^1, \theta_i^2 + \Delta\theta_i^2)$. Since for most games the exact value cannot be calculated analytically, the authors introduce a policy gradient formulation that relies on environment roll-outs to approximate it. This method is able to find tit-for-tat strategies on the Iterated Prisoner's Dilemma.

POLA (Zhao et al., 2022b) introduces an idealized version of LOLA that is invariant to policy parameterization. To do so, each player attempts to increase the probability of actions that lead to higher returns while penalizing the Kullback-Leibler divergence in policy space relative to their policies at the previous time step. Similar to the proximal point method, each step of POLA constitutes an optimization problem that is solved

approximately through gradient descent. Like LOLA, POLA uses trajectory roll-outs to estimate the value of each player and applies the reinforce estimator to compute gradients. POLA effectively achieves non exploitable cooperation on the IPD and the Coin Game improving on the shortcomings of its predecessor.

Lu et al. (2022) consider a meta-game where at each meta-step a full game is played and the meta-reward is the return of that game. The agent is then a meta-policy that learns to influence the opponent's behaviour over these rollouts. M-FOS changes the game and is not comparable to our method which considers learning a single policy. Baker (2020) changes the structure of the game where each agent is sharing reward with other agents. The agents are aware of this grouping of rewards via a noisy version of the reward sharing matrix. In the test time, the representation matrix is set to no reward sharing and no noise is added to this matrix.

Diplomacy is a zero-sum game, but locally the agents need to form a retaliatory alliance in order to gain support for winning the game. Previous approaches like Paquette et al. (2019) and FAIR et al. (2022) utilize human data to learn this behavior. However, self-play without human data, as shown by Bakhtin et al. (2021), leads to significantly different policies where agents struggle to cooperate with human players effectively.

Policy Evaluation Networks (PVN) conditions a neural network on a policy by considering the policy's behavior on a set of learned states from the environment (Harb et al., 2020). This aligns closely with our QA idea for conditioning the detective opponent on the agent's policy. However, the PVN representation is not dependent on the current state.

5.4. Best Response Shaping

Our Best Response Shaping (BRS) algorithm trains an agent by differentiating through an approximation to the best response opponent (as described in Section 5.4.1). This opponent, called the *detective*, conditions on the agent's policy via a question answering mechanism to select its actions (Section 5.4.2). Subsequently, we train the agent by differentiating through the detective using the REINFORCE gradient estimator (Sutton et al., 1999) (Section 5.4.3). For situations where a cooperative policy is desired, we propose Self-Play with reward sharing as a regularization method, encouraging the agent to explore cooperative policies. We refer to this method as BRS with Self-Play (BRS-SP). The pseudo-code for BRS and BRS-SP is provided in Algorithm 2.

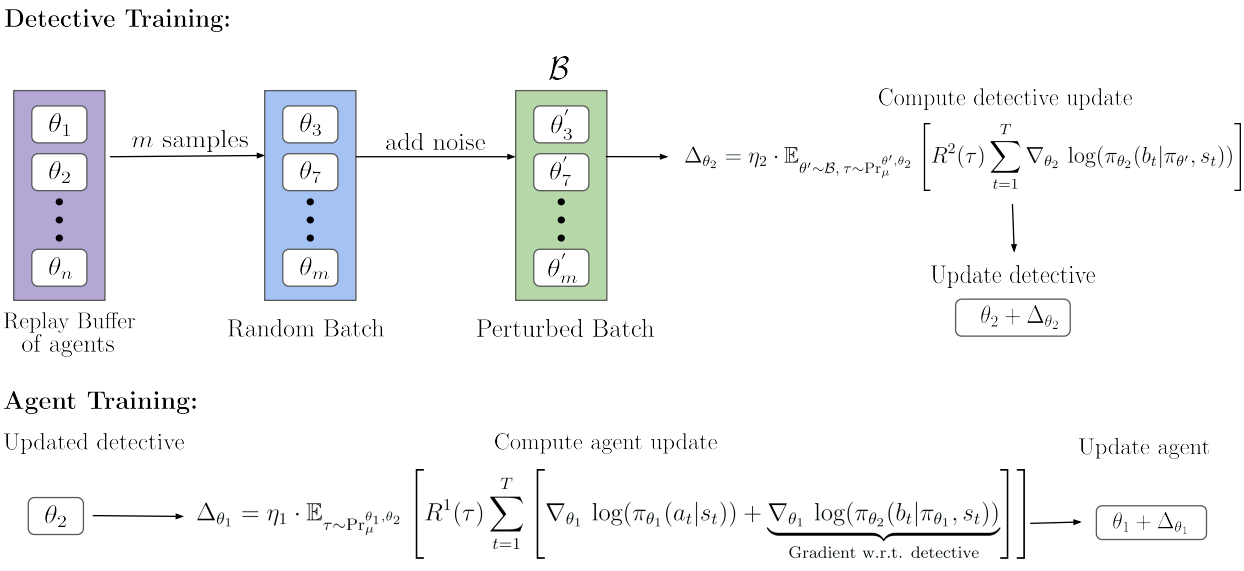


Fig. 5.1. The detective is trained using agents sampled from a replay buffer, which contains agents encountered during training. Additional noise is incorporated to broaden the range of agents encountered by the detective. The detective's training focuses on conditioning based on the agent's policy, aiming to estimate the optimal response to each policy to maximize its own return. Following this, the agent's training is conducted through backpropagation, through the detective's conditioning mechanism.

5.4.1. Best Response Agent to the Best Response Opponent

Our notation and definitions follow from Agarwal et al., 2021, we denote τ as a trajectory whose distribution, $\Pr_\mu^{\theta_1,\theta_2}(\tau)$, with initial state distribution μ is given by

$$\Pr_\mu^{\theta_1,\theta_2}(\tau) = \mu(s_0)\pi_{\theta_1}(a_0|s_0)\pi_{\theta_2}(b_0|\pi_{\theta_1}, s_0)P(s_1|s_0, a_0, b_0)\dots$$

The best response opponent is the policy that gets the highest expected return against a given agent. Formally, given θ_1, the best response opponent policy θ_2^* solves for the following:

$$\theta_2^* = \arg\max_{\theta_2} \mathbb{E}_{\tau \sim \Pr_\mu^{\theta_1,\theta_2}} \left[R^2(\tau) \right] \tag{5.4.1}$$

Subsequently, we train the agent's policy to get the highest expected return against the best response agent. This training of the agent's policy is solving for the following:

$$\theta_1^{**} = \arg\max_{\theta_1} \mathbb{E}_{\tau \sim \Pr_\mu^{\theta_1,\theta_2^*}} \left[R^1(\tau) \right] \tag{5.4.2}$$

We hypothesize that the agent $\pi_{\theta_1}^{**}$ exhibits characteristics of a non-exploitable agent, as it learns retaliatory strategies in response to a defecting opponent, thereby creating incentives for a rational opponent to cooperate. Note that $(\theta_1^{**}, \theta_2^*)$ is a nash equilibrium by definition.

5.4.2. Detective Opponent Training

In deep reinforcement learning, the training of agents relies on the utilization of gradient-based optimization. Consequently, we need a differentiable opponent approximating a best response opponent. We call this opponent the *detective*. The detective's policy conditions on the agent's policy in addition to the state of the environment, which we denote $\pi_{\theta_2}(a|\pi_{\theta_1}, s)$. We train the detective to maximize its own return against various agents. Formally, the detective is trained by the following gradient step:

$$\nabla_{\theta_2} \mathop{\mathbb{E}}_{\theta_1 \sim \mathcal{B}} \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_2}} \left[R^2(\tau) \right] \tag{5.4.3}$$

where $\mathcal{B}$ represents a distribution of diverse policies for agent 1.

Conditioning on Agent's Policy. To effectively train the agent's policy against the detective using gradient ascent on the agent's return, it is essential to establish a differentiable mechanism for the detective's conditioning. In scenarios involving toy environments with simple policy spaces, a straightforward approach of directly incorporating the agent's parameters as an input to the detective's policy works. However, it proves to be infeasible for larger policy spaces. This becomes particularly challenging when the agent's policy is represented by a neural network, as conditioning on the parameters would require an impractical number of samples. To address this limitation in more complex cases, we employ two strategies:

- *State aware conditioning* Extracting a general representation that encapsulates the whole agent behavior is a complex task. Instead, the detective extracts a representation for the current state of the game.
- *Conditioning on behavior* The detective queries the behaviour of the agent on various states of the game. To do so, the detective evaluates the agent's action probabilities (the answer) on a state of the game (the question). Formally, let $\mathcal{Q}_\psi(\theta_1, s)$ be the function used by the detective to extract a state-aware representation of the agent. We call $\mathcal{Q}$ a question answering (QA) function if $\mathcal{Q}$ can be expressed as only having access to the policy function, i.e. $\mathcal{Q}_\psi(\pi_{\theta_1}, s)$. There are many possible ways to architect a QA function. Next, we outline a method that has shown significant success in the context of the Coin Game.

Simulation Based Question Answering. The behavior of the agent in possible continuations of the game starting from state s holds valuable information. More specifically, we can assess the behavior of the agent against a random agent starting from game state s. Formally Let δ_A be defined as the following where τ is a trajectory starting from state s at

time t:

$$\delta_A := \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_r, \theta_2}} \Big[R^2(\tau) | s_t = s \Big] \tag{5.4.4}$$

where π_{θ_r} is an opponent that chooses action A at time t and afterwards samples from a uniform distribution over all possible actions:

$$\pi_{\theta_r}(a_i = A | s_i) = \begin{cases} \frac{1}{|\mathcal{A}|} & \text{if } i > t \\ \mathbb{1}_{\{a_i = A\}} & \text{if } i = t \end{cases} \tag{5.4.5}$$

Detective estimates δ_A by monte-carlo rollouts of the game to a certain length between the agent and the random opponent, π_{θ_r}. We denote the estimate of δ_A by $\hat{\delta}_A$. Then we define $\mathcal{Q}^{\text{simulation}} = [\hat{\delta}_{A_1}, \hat{\delta}_{A_2}, \cdots, \hat{\delta}_{A_{|\mathcal{A}|}}]$. The number of samples used to estimate the returns of the game and the length of the simulated games are considered hyperparameters of $\mathcal{Q}^{\text{simulation}}$ QA. Note that the $\mathcal{Q}^{\text{simulation}}$ can be differentiated with respect to agent's policy parameters via REINFORCE (Sutton et al., 1999) term. Specifically, we use the DiCE operator (Foerster et al., 2018a).

5.4.3. Agent training

Differentiating Through the Detective. The agent's policy is trained to maximize its return against the detective opponent via REINFORCE gradient estimator. However, because the detective's policy is taking the agent's policy as input, the REINFORCE term will include an additional detective-backpropagation term over the usual REINFORCE term:

$$\mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1, \theta_2}} \left[R^1(\tau) \sum_{t=1}^{T} \left[\nabla_{\theta_1} \log\big(\pi_{\theta_1}(a_t | s_t)\big) + \underbrace{\nabla_{\theta_1} \log\big(\pi_{\theta_2}(b_t | \pi_{\theta_1}, s_t)\big)}_{\text{detective-backpropagation term}} \right] \right] \tag{5.4.6}$$

This extra term can be thought of as the direction in policy space in which changing the agent's parameters encourages the detective to take actions that increase the agent's own return.

Cooperation Regularization via Self-Play with Reward Sharing. Agents that are trained against rational opponents tend to rely on the assumption that the opposing agent is lenient towards their non-cooperative actions. This reliance on rational behavior allows them to exploit the opponent to some extent. Consequently, they may not effectively learn to cooperate with their own selves. In scenarios where the objective is to foster more cooperative behavior, particularly encouraging the agent to cooperate with itself, a straightforward approach is to train the agent in a self-play setting, assuming that the opponent's policy mirrors the agent's policy. Formally, we update the agent using the

Algorithm 2 BRS/BRS-SP pseudo code: a single iteration

Input: Replay Buffer of Agent Parameters $\mathcal{B}$, Agent parameters θ^1, Detective parameters θ_2, learning rates $\alpha_1, \alpha_2, \alpha_3$, Do Self-Play with Reward Sharing flag SP, Standard Error of Noise σ
Detective Training Step:
Sample agent parameter $\theta_1\prime$ from $\mathcal{B}$
$\theta_1 \leftarrow \theta_1 + z$, where $z \sim \mathcal{N}(0, \sigma)$
Rollout trajectory τ_2 using policies $(\pi_{\theta_1\prime}, \pi_{\theta_2})$
$\theta_2 \leftarrow \theta_2 + \alpha_2 R^2(\tau_2) \sum_{t=1}^{T} \nabla_{\theta_2} \log\big(\pi_{\theta_2}(a_t|\pi_{\theta_1\prime}, s_t)\big)$
Agent Training Step:
Rollout trajectory τ_1 using policies $(\pi_{\theta_1}, \pi_{\theta_2})$
$\theta_1 \leftarrow \theta_1 + \alpha_1 R^1(\tau) \sum_{t=1}^{T} \nabla_{\theta_1} \log\big(\pi_{\theta_1}(a_t|s_t)\big) + \nabla_{\theta_1} \log\big(\pi_{\theta_2}(b_t|\pi_{\theta_1}, s_t)\big)$
Self Play Reward Sharing Step:
if SP = True **then**
  Rollout trajectory τ_3 using policies $(\pi_{\theta_1}, \pi_{\theta_1})$
  $\theta_1 \leftarrow \theta_1 + \alpha_3 R^1(\tau_3) \sum_{t=1}^{T} \nabla_{\theta_1} \big[\log\big(\pi_{\theta_1}(a_t|s_t)\big) + \log\big(\pi_{\theta_1}(b_t|s_t)\big)\big]$
end if
Push θ_1 to $\mathcal{B}$
Output: θ_1, θ_2

following update rule:

$$\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \Big[R^1(\tau)\Big] \tag{5.4.7}$$

We prove that in symmetric games like IPD and Coin Game, this is equivalent to training an agent with self-play with reward sharing (see proof in §5.D). This training brings out the cooperative element of general-sum games. In zero-sum games, this update will have no effect as the gradient would be zero (see proof in §5.D). We refer to this regularization loss term as Self-Play with reward sharing throughout the paper.

5.5. Experiments

5.5.1. Iterated Prisoner's Dilemma

Following Foerster et al. (2018b), we study Iterated Prisoner's Dilemma (IPD) game where the agents observe the last actions taken by the agents. Therefore, all possible agent observations would be $\mathcal{S} = \{\mathrm{C}, \mathrm{CC}, \mathrm{CD}, \mathrm{DC}, \mathrm{DD}\}$, where C is the initial state, and each agent's policy can be described by indicating the probability of cooperation for each $s \in \mathcal{S}$. We consider the IPD game that is six steps long. As shown by Foerster et al. (2018b) and Zhao et al. (2022b), training two naïve-learning agents leads to strategies that always defect. Although this is a Nash Equilibrium, both agents will receive negative returns.

We test our method by training the agent against a tree search detective. The tree search detective constructs a tree, commencing from the current state. During this process, the agent's actions are sampled from the agent's policy, while the tree branches explore all possible choices for the detective's actions. The detective selects the actions that maximize its return, i.e. the actions that construct the best response path within the tree. The agent receives the return that corresponds to this particular path (see §5.E for details). Our agent is a two-layer MLP that receives the five possible states and outputs the probability of cooperation. We choose an MLP to showcase the possibility of training neural networks via BRS. We update our agent policy via policy gradient.

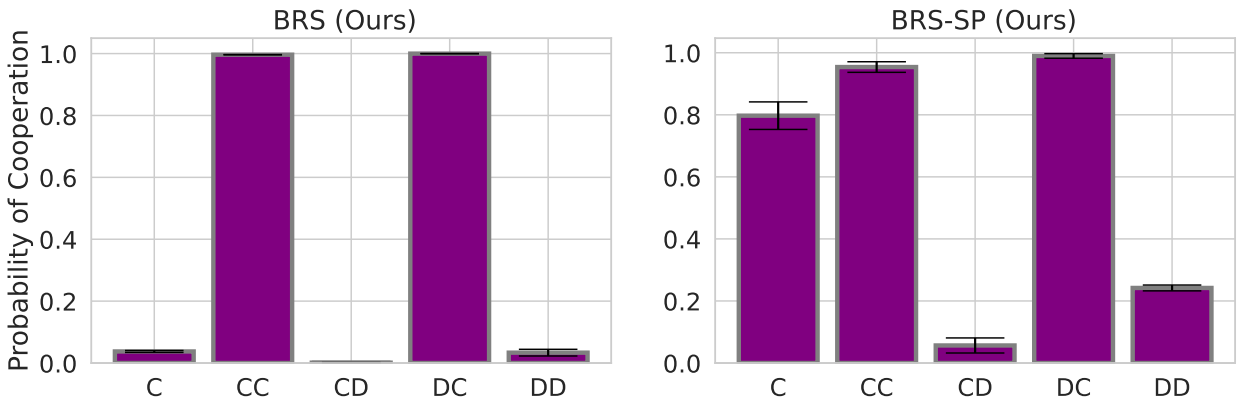


Fig. 5.2. Illustration of the policies of agents trained with BRS and BRS-SP in a finite Iterated Prisoner's Dilemma game of length 6. The agents are trained against a tree search detective maximizing its own return. BRS-SP agents learn tit-for-tat, a policy that cooperates initially and mirrors the opponent's behavior thereafter. BRS agents learn cynic-tit-for-tat (CTFT), they defect initially but mirror the opponent's behavior thereafter. Therefore, at the initial state, BRS agents exploit the rationality of the opponent.

In Figure 5.2, we show the result of training the BRS agent against a tree search detective opponent. The agents learn a variant of tit-for-tat that defects initially but has the same probability of cooperation as tit-for-tat in $\{CC, CD, DC, DD\}$. We name this policy cynic-tit-for-tat (CTFT). This is because the agent assumes the opponent is a rational opponent that chooses the best response w.r.t to the agent's policy. The best response to a cynic-tit-for-tat in an infinite IPD game is always cooperating because if the opponent defects initially, the agent will defect in the next turn. In other words, the BRS agent learns to exploit the fact that the opponent (detective) is rational.

As shown in Figure 5.2 we also train the BRS with Self-Play (BRS-SP) agent against the same opponent. The trained agents learn tit-for-tat policy. Unlike CTFT, TFT cooperates in the first step, thus it cooperates with the best response opponent and with itself. The downside is that it is exploited by the always-defect opponent at the first step.

5.5.2. The Coin Game

The Coin Game, introduced by Lerer and Peysakhovich (2017), is a two-player general sum game that takes place in a grid. The game involves two players: the red player and the blue player. At each episode, a coin, either red or blue, spawns somewhere in the grid and players compete to pick it up. The color of the spawning coin changes after each episode. If a player picks any coin, they receive a positive reward of $+1$. If the coin corresponding to their color is picked by the other player, they are punished with a negative reward of -2. Ideally, players should cooperate by taking only the coins of their associated color in fear of future retaliation from the other agent.

We follow Zhao et al. (2022b) in training a GRU (Cho et al., 2014) agent on a 3×3 sized Coin Game with a game length of 50 and a discount factor of 0.96. The detective opponent is also a GRU agent with an MLP that conditions on the result of the QA (for more details see §5.A).

We evaluate our agent against four policies: an opponent that always takes the shortest path towards the coin regardless of the coin's color (Always Defect), an opponent that takes the shortest path towards its associated coin but never picks up the agent's associated coin (Always Cooperate), a Monte Carlo Tree Search opponent that evaluates multiple rollouts of the game against the agent in order to take an action (MCTS), an opponent that is trained to maximize its own return against the current agent (Trained), and itself (Self). Note that the MCTS will approximate the best response opponent given enough samples. Figure 5.3 visually presents the evaluation metrics for the BRS, BRS-SP, and POLA agents. In the subsequent paragraphs, we present a comprehensive analysis and interpretation of these results.

Is cooperation the best response?. MCTS opponent and the Trained opponent approximate the best response opponent. The cooperation exhibited by these opponents towards the agents indicates the extent to which the agents have fostered retaliatory behavior. Comparing returns, BRS and BRS-SP outperform POLA against MCTS, and BRS-SP surpasses both BRS and POLA against the Trained opponent (Figure 5.3). [1]

Does Always Cooperate regret its cooperation with the agent?. Ideally, an agent should be retaliatory in such a way that the best response would be always cooperation. In other words, the Always-Cooperate should not regret cooperating with the agent. As shown in Figure 5.3, the Always Cooperate's return is close the MCTS opponent's return against the BRS-SP. However, the MCTS return against the BRS and POLA is significantly higher than the Always-Cooperate's return.

[1] Note that our MCTS opponent, with 1000 rollouts at each state, provides a significantly stronger approximation of the best response compared to the Trained agent which is a GRU agent trained using policy gradient for 3000 steps.

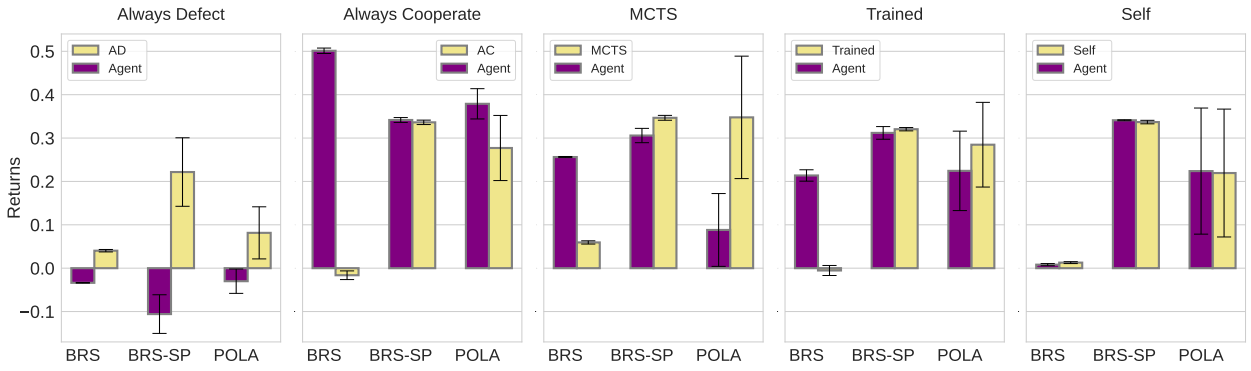


Fig. 5.3. Comparison of Agent's trained with BRS, BRS-SP, and POLA on a 3×3 sized Coin Game. We evaluate the agent's returns versus different opponents: Always Defect opponent (AD) that takes the shortest path to all coins; Always Cooperate opponent (AC) that takes the shortest path to its own coin but does not take the agent's coin; A Monte Carlo Tree Search opponent (MCTS) that do one thousand rollouts using the agent's policy at each state; a trained opponent (Trained) that is trained vs the agent to maximize its own return; and agent's performance against itself (Self). The POLA agents are exploited by the MCTS, which approximates the best response. It means, a rational opponent maximizing its return, harms POLA's return unintentionally. In contrast, BRS and BRS-SP get a higher return against the MCTS. Although BRS agents are not inherently cooperative among themselves, they effectively exploit the AC opponent in a rational manner. On the other hand, BRS-SP agents exhibit greater levels of cooperation among themselves but refrain from exploiting the AC opponent.

Does always defect exploit the agent?. As shown in Figure 5.3 BRS agents and POLA agents get near zero return against Always Defect. However, the return of the Always Defect against POLA is higher. BRS-SP agent gets a lower return compared to BRS and POLA indicating it is exploited more by the AD policy (Note that BRS-SP still retaliates but to a lesser degree).

Does the agent cooperate with itself?. As shown in Figure 5.3 BRS agents get near zero return against themselves. Indicating that they are not cooperating with themselves. BRS-SP agents always cooperate with themselves. The POLA agents cooperate with themselves less than BRS-SP and more than BRS and also with higher variance.

BRS, BRS-SP, and POLA. As shown in Figure 5.4, BRS and POLA agents exploit BRS-SP agents and BRS exploits POLA agents on average.

5.6. Limitations

This paper focuses on the implementation of our proposed idea in two-player games. However, extending this approach to games with more than two players presents a non-trivial challenge. Additionally, the detective agent approximates the best response opponent, and the accuracy of this approximation is crucial for obtaining correct gradients

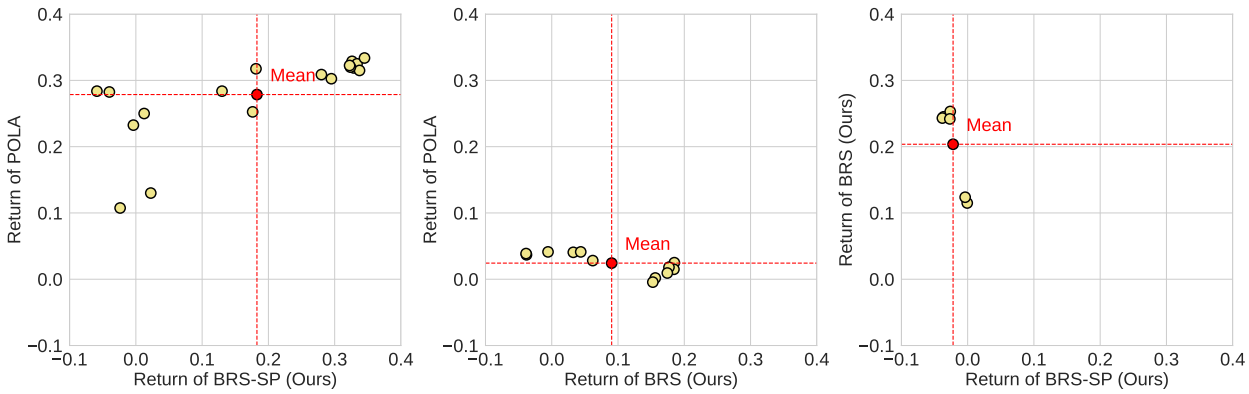


Fig. 5.4. Evaluating BRS, BRS-SP, and POLA agents against each other. The red dot shows the average return of each agent against the average return of the other agent. Left: Both agents get a positive return on average. However, POLA gets a higher return than BRS-SP agent on average, indicating exploitation. Middle: POLA agents and BRS agents do not cooperate. BRS is able to get a higher return than POLA on average indicating that POLA is exploited. Right: BRS agents get a higher return than the BRS-SP agents. Effectively BRS agent is exploiting the BRS-SP agent.

during the agent's training process. It is particularly important to train the detective against a diverse set of agents in order to ensure the correct gradient estimation. In this study, we introduce a replay buffer that contains a noisy version of agents encountered during training, which serves as a proxy for a diverse agent set. Nevertheless, for more complex settings, this level of diversity may be insufficient. While we employ simulation-based question answering against a random agent to condition the detective in the Coin Game, it should be noted that a random agent might not provide adequate conditioning for more intricate games.

5.7. Conclusion

Motivated by differentiating through optimization steps of our opponent in order to achieve non-exploitable cooperation, we introduced BRS that differentiates through an approximated best response opponent. We evaluated BRS agents in detail, examining the implications of such an assumption. In particular, we show that BRS agents are not exploited by an MCTS agent. We also introduced BRS-SP for situations in which more cooperative policies are needed. The BRS-SP agent reaches a policy where always cooperate is the best response to that policy. We hope this work helps improving the scalability and non-exploitability of agents in Multi Agent Reinforcement Learning enabling agents that learn reciprocation-based cooperation in complex games.

References

Agarwal, Alekh et al. (2021). *Reinforcement Learning: Theory and Algorithms*.

Baker, Bowen (2020). *Emergent Reciprocity and Team Formation from Randomized Uncertain Social Preferences*. arXiv: 2011.05373 [cs.LG].

Bakhtin, Anton et al. (2021). "No-press diplomacy from scratch". In: *Advances in Neural Information Processing Systems* 34, pp. 18063–18074.

Cho, Kyunghyun et al. (Oct. 2014). "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches". In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, pp. 103–111. DOI: 10.3115/v1/W14-4012.

FAIR, Meta Fundamental AI Research Diplomacy Team et al. (2022). "Human-level play in the game of Diplomacy by combining language models with strategic reasoning". In: *Science* 378.6624, pp. 1067–1074.

Foerster, Jakob et al. (2018a). "Dice: The infinitely differentiable monte carlo estimator". In: *International Conference on Machine Learning*. PMLR, pp. 1529–1538.

Foerster, Jakob et al. (2018b). "Learning with Opponent-Learning Awareness". In: *International Conference on Autonomous Agents and Multiagent Systems*.

Harb, Jean et al. (2020). "Policy evaluation networks". In: *arXiv preprint arXiv:2002.11833*.

Lerer, Adam and Alexander Peysakhovich (2017). "Maintaining cooperation in complex social dilemmas using deep reinforcement learning". In: *arXiv preprint arXiv:1707.01068*.

Lu, Chris et al. (2022). *Model-Free Opponent Shaping*. arXiv: 2205.01447 [cs.AI].

Paquette, Philip et al. (2019). "No-press diplomacy: Modeling multi-agent gameplay". In: *Advances in Neural Information Processing Systems* 32.

Schulman, John et al. (2018). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv: 1506.02438 [cs.LG].

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Sutton, Richard S et al. (1999). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12.

Vinyals, Oriol et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354.

Zhao, Stephen et al. (2022a). "Proximal Learning With Opponent-Learning Awareness". In: *Advances in Neural Information Processing Systems* 35, pp. 26324–26336.

Zhao, Stephen et al. (2022b). "Proximal Learning With Opponent-Learning Awareness". In: *Advances in Neural Information Processing Systems* 35, pp. 26324–26336.

5.A. Experimental Details

5.A.1. IPD

In IPD experiments, we are experimenting on IPD with 6 steps and discount factor of 1., i.e. no discount factor. The payoff matrix of the IPD game is shown in 5.1.

Player 2 \ Player 1	Cooperate	Defect
Cooperate	-1 / -1	0 / -3
Defect	-3 / 0	-2 / -2

Table 5.1. Payoff matrix for the prisoner's dilemma game

Our agent's policy is parameterized by a two-layer MLP (Multi-Layer Perceptron) with a tanh non-linearity. The choice of tanh non-linearity is motivated by its smoothing effect and its ability to prevent large gradient updates.

During training, the agent is trained against the Tree Search Detective (TSD) (see Appendix 5.E) using a policy gradient estimator. We employ a learning rate of `3e-4` with the SGD (Stochastic Gradient Descent) optimizer. In the BRS-SP experiments, the Self-Play with reward sharing loss is optimized using SGD with the same learning rate of `3e-4`. To reduce variance, the policy gradients incorporate a baseline.

For replicating the exact results presented in the paper, we provide the code in Appendix 5.B. Running the code on an A100 GPU is expected to take approximately an hour. The plots and error bars are averaged over 10 seeds for both BRS and BRS-SP. The hyperparameter search was conducted by iterating over various learning rates including `1e-4,3e-4,1e-3`, and the optimizers were explored between SGD and Adam.

5.A.2. Coin Game

The game. Our coin game implementation exactly follows the POLA implementation (Zhao et al., 2022b). Similar to POLA, we also experiment with the game length of 50 and a discount factor of 0.96.

Agent's architecture. In the coin game, we have an actor-critic setup. The policy of our agent is parameterized by a GRU (Gated Recurrent Unit) architecture, following the approach outlined in the POLA repository (source). However, we introduce a modification compared to POLA by including a two-layer MLP on top of the observations before they are fed into the GRU instead of a single-layer MLP. Additionally, we utilize two linear heads to facilitate separate learning for policy and value estimation.

Detective's architecture. The architecture of the detective is as follows: The sequence of observations is fed into a GRU (Gated Recurrent Unit), which is the same architecture used by the agent. At each time step, the agent's representation is extracted using the QA (Question-Answering) module of the detective. In our experiments, we employed 16 samples of continuing the game for the next 4 steps from the current state.

Environment Simulations. The QA for the BRS experiments changes the environment seed each time simulating the environment. However, in the BRS-SP runs we are using the same environment seed for the simulation which reduces the variance in simulations further but fixes the place that the next coin will appear given the same sequence of events.

Subsequently, the output of the QA module and the GRU are concatenated and passed through a two-layer MLP with ReLU non-linearities. The resulting output from this MLP is then fed into a linear layer for estimating the value (critic), and a linear layer for determining the policy (actor).

Separate optimizers for the two terms. The agent uses separate optimizers for the two terms in the policy gradient. That is, it uses two separate optimizers for the two terms indicated in 5.A.1.

$$\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_2}}\left[R^1(\tau)\sum_{t=1}^{T}\left[\underbrace{\nabla_{\theta_1}\log\big(\pi_{\theta_1}(a_t|s_t)\big)}_{\text{Term 1}}+\underbrace{\nabla_{\theta_1}\log\big(\pi_{\theta_2}(b_t|\pi_{\theta_1},s_t)\big)}_{\text{Term 2}}\right]\right] \tag{5.A.1}$$

Losses and optimizers. The value functions in our setup are trained using the Huber loss. On the other hand, the policies are trained using the standard policy gradient loss with Generalized Advantage Estimation (GAE) (Schulman et al., 2018). However, it is important to note that our hyperparameter search led us to set the GAE parameter, λ, to 1, which results in an equivalent estimation of the advantage using the Monte-Carlo estimate. This choice is similar to the hyperparameters reported by POLA (source).

In the BRS experiments, the agent's policy is trained using a learning rate of `1e-3`, while in the BRS-SP experiments, an Adam optimizer with a learning rate of `3e-4` is utilized. The value functions of both the agent and the detective in all experiments are trained using Adam with a learning rate of `3e-4`. Similarly, the detective's policy is trained using Adam with a learning rate of `3e-4` in all experiments.

Replay buffer of previous agents. During the training, we keep a replay buffer of previous agents seen during the training. In BRS experiments we keep 2048 previous agents and in BRS-SP experiments we keep the last 512 agents. For training the detective, we sample a batch from this replay buffers uniformly. We add a normal noise with variance of 0.01 to the parameters of these agents to ensure the detective is trained against a diverse set of agents.

Hyperparameter search. We conducted a hyperparameter search using random search over the configurations explained Table 5.2. the entropy coefficient β, which is multiplied by the entropy of the log probabilities associated with the actions of the corresponding player, is added to the policy gradient loss of the corresponding player for controlling the exploration-exploitation trade-off.
Plots and error bars. The results on the paper are computed over two seeds for the BRS, three seeds for the BRS-SP and six seeds for POLA.[2] It is worth noting that the error bars are calculate over seeds, i.e. checkpoints. The result of games between each pair of agents is averaged over 32 independent games between those two agents.

Hyperparameter	**Values**
inner game length in QA	4, 8, 12, 16
samples in QA	16, 64, 256, 1024
replay buffer of agent's size	10, 512, 4096, 16384
value learning algorithm	TD-0, Monte-Carlo
GAE λ	0.9, 0.96, 0.99, 0.999, 1.0
agent policy gradient learning rate	0.001, 0.0003
agent entropy β	0.0, 1.0, 2.0, 5.0, 10.0
detective entropy β	0.0, 1.0, 2.0, 5.0, 10.0

Table 5.2. Hyperparameter search options

Compute. Our BRS/BRS-SP runs are run for 48 hours on a single A100 GPU with 40 Gigabytes of memory[3].
Batch size. We use a batch size of 128.
POLA agent's training. To evaluate the POLA agents, we trained them by executing the POLA repository here (Zhao et al., 2022b).

5.B. Reproducing Results

5.B.1. IPD

To replicate the results on IPD (Iterated Prisoner's Dilemma), please refer to the instructions available at here. By running the provided Colab notebook, you will obtain the IPD plot that is included in the paper.

[2]The reason we have fewer seeds for BRS and BRS-SP than POLA is computational constraints due to conference deadlines. Especially, one of our BRS checkpoints was not pickled properly and could not be loaded in the last minute. That is why we don't have three BRS seeds equal to BRS-SP
[3]A single A100 GPU is 80 Gigabytes, but it can be split into two 40 Gigabyte equivalents and we train on one of these splits

5.B.2. Coin Game

To replicate the outcomes of the coin game, please refer to the instructions available at here. In essence, the provided guidelines encompass training scripts designed for the purpose of training agent checkpoints. Subsequently, there is an exporting phase in which these checkpoints are transformed into their lightweight counterparts. Finally, a script is provided to facilitate the execution of a league involving multiple agents.

5.C. League Results

In order to visualize the results of our training in complete detail, in Figure 5.5 we visualize a matrix, in the format of a heatmap, of the returns of various agents against each other. All the results are averaged over 32 independent games between the corresponding agents. The game is the Coin game of length 50. [4]

5.D. Self-Play

Lemma D.1. *Denote $o \in \mathcal{S}$ to be the state $s \in \mathcal{S}$ from the perspective of the opponent. For a symmetric game, if it holds that $\mu(s_0) = \mu(o_0)$ for all $s_0, o_0 \in \mathcal{S}$, then*

$$\mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \left[R^1(\tau) \right] = \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \left[R^2(\tau) \right]$$

where $R^2 := \sum_{t=0}^{\infty} \gamma^t r^2(o_t, b_t, a_t)$ and r^2 denotes r^1 from the perspective of the opponent.

Proof. Denote $\bar{\tau} = o_0, b_0, a_0, o_1, \ldots$, then notice that

$$\mu(s_0)\pi^1_{\theta_1}(a_0|s_0)\pi^1_{\theta_1}(b_0|o_0)P(s_1|s_0,a_0,b_0)\ldots = \mu(o_0)\pi^1_{\theta_1}(b_0|o_0)\pi^1_{\theta_1}(a_0|s_0)P(o_1|o_0,b_0,a_0)\ldots$$
$$\iff \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}(\tau) = \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}(\bar{\tau})$$

[4] Note that there is no meaning to test the trained agent against a trained agent. It is because the trained agent assumes a fixed agent and trains against it. Similarly, there is no meaning to train a trained agent against MCTS. Because the MCTS is not a fixed policy assigning probability to actions given different trajectories. Also, there is no meaning to train MCTS against MCTS because the MCTS needs to roll-out the agent's policy to choose an action. However, MCTS against MCTS implies an infinite loop of rolling out the other agent's policy

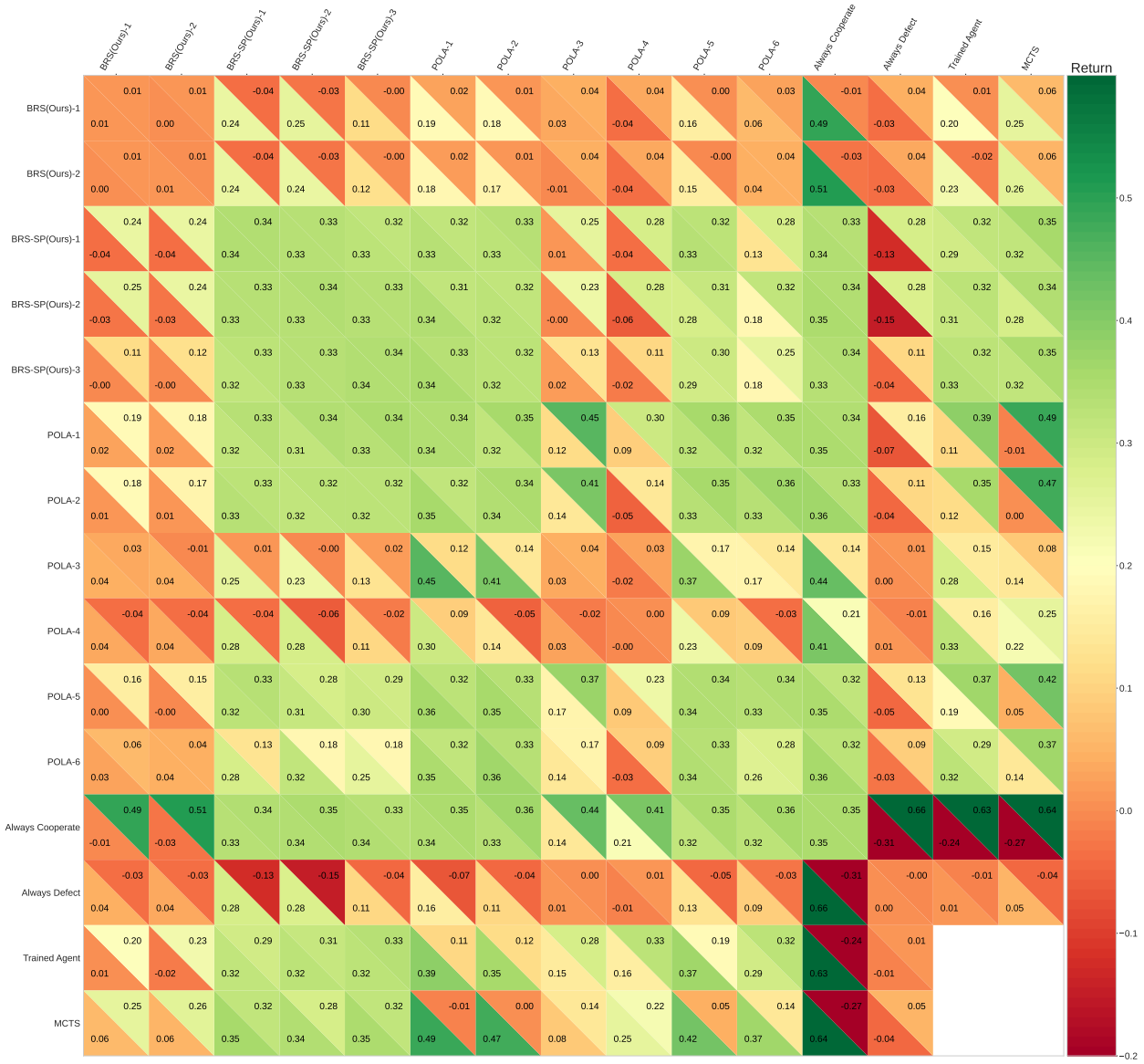


Fig. 5.5. The presented figure illustrates the outcomes of 1-vs-1 Coin games lasting 50 rounds, involving a range of agents. The return achieved by each agent is documented within the corresponding cell. The reported returns are an average across 32 independent games. The numerical suffixes following the agent names signify agents trained using distinct initialization seeds. It is important to note that there are no games recorded between the MTCS agent and the trained agent due to a lack of clarity regarding its meaning.

now by symmetry we have that $r^1(s_t, a_t, b_t) = r^2(o_t, b_t, a_t)$, therefore

$$\begin{aligned}
\mathbb{E}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] &= \mathbb{E}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[\sum_{t=0}^{\infty}\gamma^t r^1(s_t, a_t, b_t)\right] \\
&= \sum_{\tau}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\sum_{t=0}^{\infty}\gamma^t r^1(s_t, a_t, b_t) \\
&= \sum_{\bar{\tau}}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\bar{\tau})\sum_{t=0}^{\infty}\gamma^t r^2(o_t, b_t, a_t) \\
&= \mathbb{E}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^2(\tau)\right]
\end{aligned}$$

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] &= \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[\sum_{t=0}^{\infty}\gamma^t r^1(s_t, a_t, b_t)\right] \\
&= \sum_{\tau}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\sum_{t=0}^{\infty}\gamma^t r^1(s_t, a_t, b_t) \\
&= \sum_{\bar{\tau}}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\bar{\tau})\sum_{t=0}^{\infty}\gamma^t r^2(o_t, b_t, a_t) \\
&= \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^2(\tau)\right]
\end{aligned}$$

where we just rename $\bar{\tau}$ in the last equality. ■

Proposition D.2 states that the gradient in Equation 5.4.7 is equivalent to that of self-play with reward-sharing.

Proposition D.2. *For a symmetric game,*

$$\nabla_{\theta_1}\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] \propto \left[\nabla_{\theta_1}\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\left[R^1(\tau)+R^2(\tau)\right] + \nabla_{\theta_2}\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\left[R^1(\tau)+R^2(\tau)\right]\right]_{\theta_2=\theta_1}.$$

Proof. We write the gradient as follows:

$$\begin{aligned}
\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)\Big] &= \sum_\tau R^1(\tau)\nabla_{\theta_1}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau) \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\nabla_{\theta_1}\log \mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau) \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\nabla_{\theta_1}\log \mu(p_0)\pi^1_{\theta_1}(a_0|s_0)\pi^1_{\theta_1}(b_0|o_0)\dots \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log \pi^1_{\theta_1}(a_t|s_t) + \nabla_{\theta_1}\log \pi^1_{\theta_1}(b_t|o_t) \\
&= \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log \pi^1_{\theta_1}(a_t|s_t) + \nabla_{\theta_1}\log \pi^1_{\theta_1}(b_t|o_t)\right].
\end{aligned}$$

Now by symmetry and Lemma D.1. we have

$$\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)\Big] = \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^2(\tau)\Big],$$

and by linearity of expectation,

$$\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)\Big] \propto \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)+R^2(\tau)\Big].$$

Hence

$$\begin{aligned}
\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)\Big] &\propto \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[\Big(R^1(\tau)+R^2(\tau)\Big)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log \pi^1_{\theta_1}(a_t|s_t) + \nabla_{\theta_1}\log \pi^1_{\theta_1}(b_t|o_t)\right] \\
&= \left[\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\left[\Big(R^1(\tau)+R^2(\tau)\Big)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log \pi^1_{\theta_1}(a_t|s_t) + \nabla_{\theta_2}\log \pi^2_{\theta_2}(b_t|o_t)\right]\right]_{\theta_2=\theta_1} \\
&= \left[\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\Big[\Big(R^1(\tau)+R^2(\tau)\Big)\Big(\nabla_{\theta_1}\log \mathrm{Pr}_\mu^{\theta_1,\theta_2}(\tau) + \nabla_{\theta_2}\log \mathrm{Pr}_\mu^{\theta_1,\theta_2}(\tau)\Big)\Big]\right]_{\theta_2=\theta_1} \\
&= \left[\nabla_{\theta_1}\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\Big[R^1(\tau)+R^2(\tau)\Big] + \nabla_{\theta_2}\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}}\Big[R^1(\tau)+R^2(\tau)\Big]\right]_{\theta_2=\theta_1},
\end{aligned}$$

which was to be shown. ■

Corollary D.3. *For a symmetric, zero-sum game it holds that*

$$\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_\mu^{\theta_1,\theta_1}}\Big[R^1(\tau)\Big] = 0$$

Proof. By definition of zero-sum game, we have that

$$
\begin{aligned}
& r^1(s_t, a_t, b_t) + r^2(s_t, b_t, a_t) = 0 \\
\implies & \sum_{t=0}^{\infty} \gamma^t \Big(r^1(s_t, a_t, b_t) + r^2(s_t, b_t, a_t) \Big) = 0 \\
\iff & R^1(\tau) = -R^2(\tau) \text{ for all } \tau
\end{aligned}
$$

From proposition D.2. we get

$$
\begin{aligned}
\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1, \theta_1}} \Big[R^1(\tau) \Big] \quad \propto \quad & \left[\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1, \theta_2}} \underbrace{\Big[R^1(\tau) + R^2(\tau) \Big]}_{=0} + \nabla_{\theta_2} \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1, \theta_2}} \underbrace{\Big[R^1(\tau) + R^2(\tau) \Big]}_{=0} \right]_{\theta_2 = \theta_1} \\
= \quad & \big[\nabla_{\theta_1} 0 + \nabla_{\theta_2} 0 \big]_{\theta_2 = \theta_1} \\
= \quad & 0,
\end{aligned}
$$

completing the proof. ■

5.E. Tree Search Detective

In this section, we describe the Tree Search Detective (TSD) used in the IPD experiments.The intuition behind TSD is that by simulating all possible trajectories based on the agent's policy, the opponent can select the path that maximizes its own returns. Consequently, the agent achieves the return associated with that specific path.

TSD implements this idea. TSD builds a tree structure in which the agent's actions are directly sampled from its policy. When it comes to TSD's action, a branch is formed for each action to explore the potential outcomes of that specific action.

The agent will treat TSD as a black-box algorithm that queries the agent's policy on a set of states and returns a single return, i.e. the return that corresponds to the agent's return in the path that yielded the highest return for the TSD. This black-box can be differentiated through via policy gradient estimators. It is worth noting that when calculating the policy gradient loss, the sum of all log probabilities should be considered, not just the ones present in the chosen path. This is crucial because the agent's actions in states outside of the selected path are significant in TSD's decision-making process for selecting that particular path. This idea has been depicted in Figure 5.6.

5.F. Evaluation Metrics of Various Agents

In Figure 5.7 we plotted the evaluation metrics for all the agents against the evaluation opponents. Note that for BRS, we used two seeds. For BRS-SP we used three seeds, and for POLA we used six seeds. Indeed, the POLA agents have more variance in their

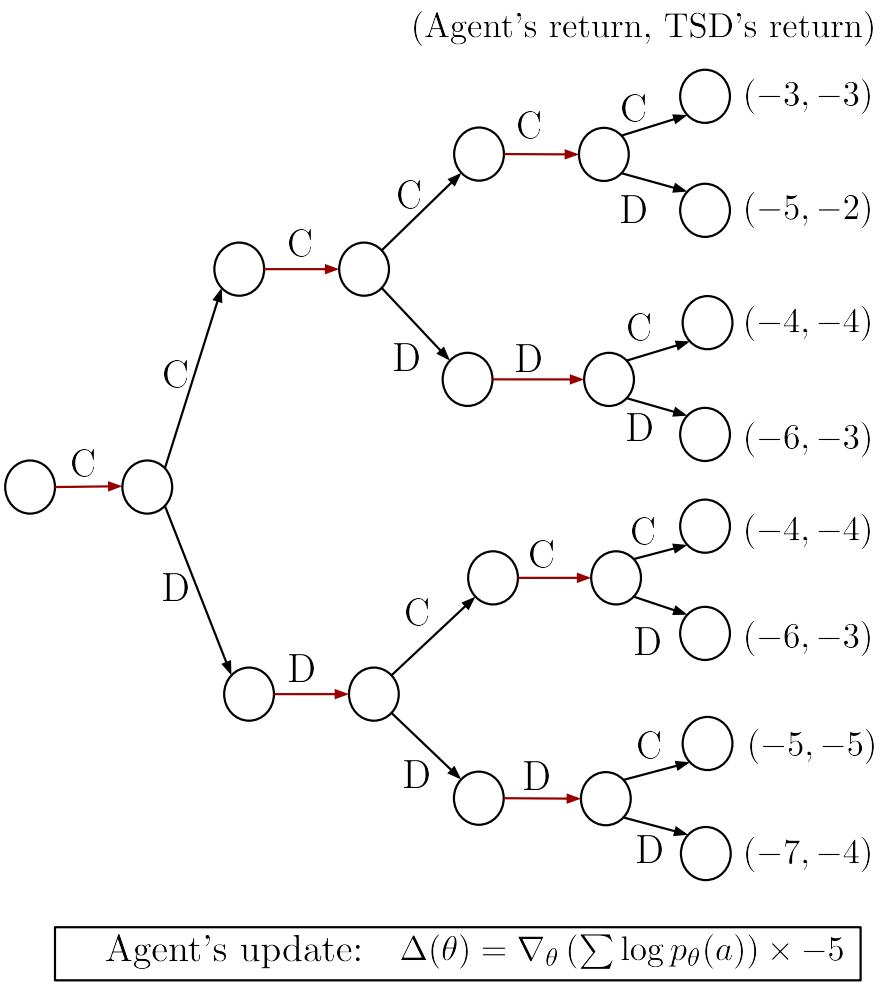


Fig. 5.6. This figure illustrates the training of the IPD agent against the TSD. TSD samples from the agent's policy, represented by red arrows in the plot, while exploring all possible actions when considering its own actions, represented by black arrows in the plot. The agent treats the TSD as a black-box algorithm and differentiates through it via REINFORCE. Note that the summation is over all log probabilities and not only over the log probabilities presnet in the path.

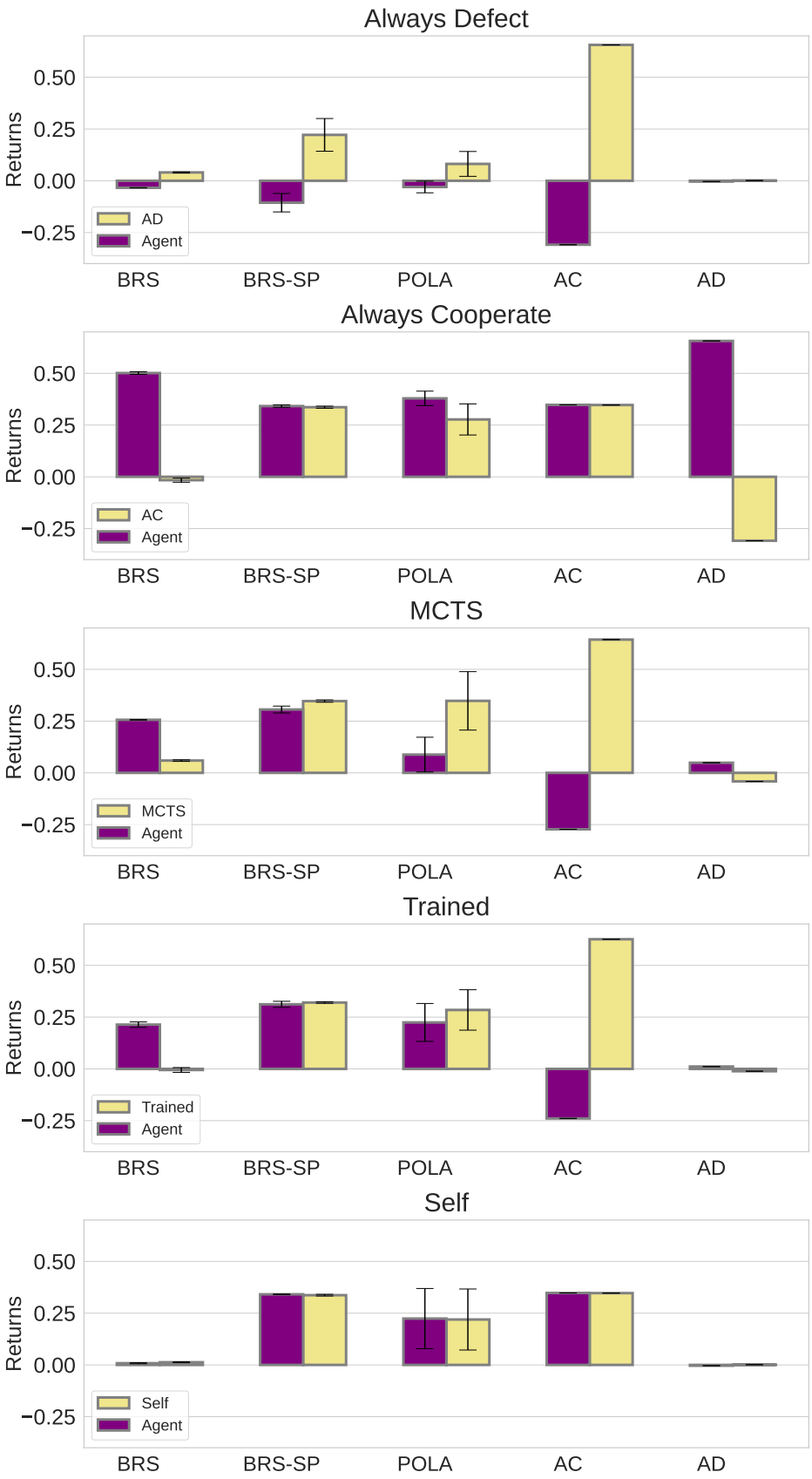


Fig. 5.7. This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against the evaluation metrics.

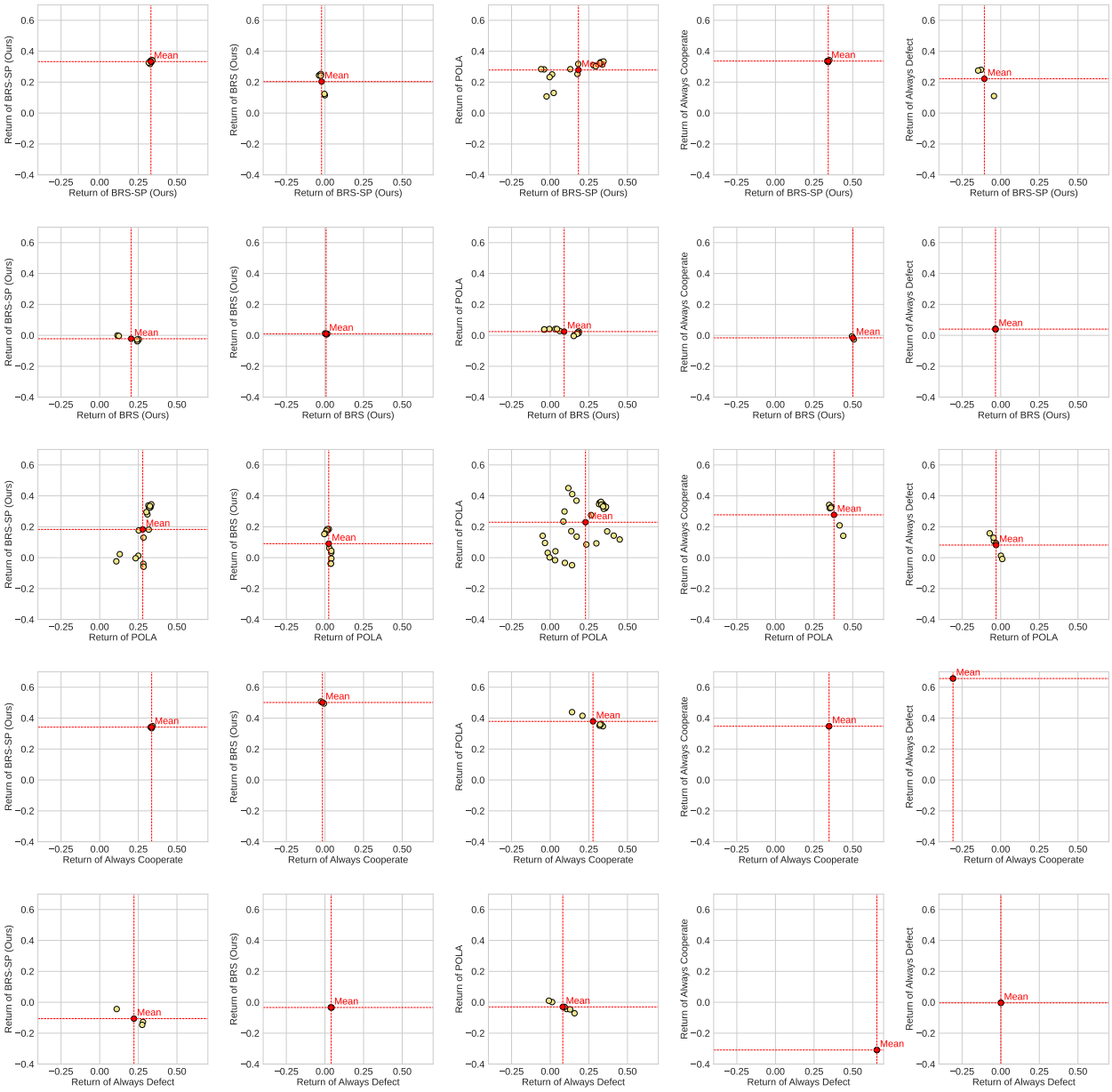


Fig. 5.8. This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against each other.

performance in contrary to the fact that we used more seeds to compute the error bars for them. In Figure 5.8 we visualized the average result of 32 games between different agents.

Chapter 6

Meta-Value Learning: a General Framework for Learning with Learning Awareness

Tim Cooijmans* Milad Aghajohari* Aaron Courville*

* Mila & Université de Montréal

Gradient-based learning in multi-agent systems is difficult because the gradient derives from a first-order model which does not account for the interaction between agents' learning processes. LOLA (Foerster et al., 2018c) accounts for this by differentiating through one step of optimization. We propose to judge joint policies by their long-term prospects as measured by the meta-value, a discounted sum over the returns of future optimization iterates. We apply a form of Q-learning to the meta-game of optimization, in a way that avoids the need to explicitly represent the continuous action space of policy updates. The resulting method, MeVa, is consistent and far-sighted, and does not require REINFORCE estimators. We analyze the behavior of our method on a toy game and compare to prior work on repeated matrix games.

Prologue

Meta-Value Learning is the other work to come out of our multi-agent learning project. Recall that the main problem with naive learning is that each agent assumes it is the only learning agent, effectively treating the other agents as static aspects of the environment. This makes the environment nonstationary; the transition probabilities for example now depend on the policies of the other agents, which change as these agents learn. From an optimization perspective, the gradient becomes uninformative, as the first-order model from which it derives ignores interactions between the agents' derivatives.

LOLA was at the time the only method that would reliably find tit-for-tat on the Iterated Prisoner's Dilemma. LOLA's inconsistency seemed like an important shortcoming – solving it might lead us to game theory of mind. I eventually came up with the idea of using a *learned* surrogate trained to satisfy an implicit, consistent equation (like COLA, although this was well before COLA was published). Around the same time, we were

discussing the ideas that ultimately became Best Response Shaping, and this is where I started thinking about using a Bellman equation to look farther ahead. I developed the Q-learning interpretation, the efficient parameterization, the correction term formulation and the exploration scheme. Milad suggested using the full gradient instead of semigradient, and training a single model for different values of γ. The code, experiments and paper are all my own work.

6.1. Introduction

Multi-agent reinforcement learning (Busoniu et al., 2008) has found success in two-player zero-sum games (Mnih et al., 2015; Silver et al., 2017), cooperative settings (Lauer, 2000; Matignon et al., 2007; Foerster et al., 2018a; Panait and Luke, 2005), and mixed settings with intra-team cooperation and inter-team competition (Lowe et al., 2017). General-sum games, however, have proven to be a formidable challenge.

The classic example is the Prisoner's Dilemma, a matrix game in which two players must decide simultaneously whether to cooperate or defect. Both players prefer cooperate-cooperate over defect-defect, but unilaterally both players prefer to defect. The game becomes qualitatively different when it is infinitely repeated and players can see what their opponents did in previous rounds: this is known as the Iterated Prisoner's Dilemma (IPD). This gives players the ability to retaliate against defection, which allows cooperation with limited risk of exploitation. The most well-known retaliatory strategy is tit-for-tat (Axelrod and Hamilton, 1981), which cooperates initially and from then on copies its opponent. It was only recently discovered that there exist ZD-extortion policies (Press and Dyson, 2012) which force rational opponents to accept considerable losses. Such exortionate policies, however, perform worse against themselves than tit-for-tat does.

The problem of learning to coordinate in general-sum games has received considerable attention over the years (Busoniu et al., 2008; Gronauer and Diepold, 2022). The naive application of gradient descent (see §6.2.1) fails to find tit-for-tat on the IPD unless initialized sufficiently close to it (Foerster et al., 2018c). Instead, it converges to always-defect, to the detriment of both players. Several works approach the problem through modifications of the objective, e.g. to share reward (Baker, 2020), to explicitly encourage interaction (Jaques et al., 2019) or encourage fairness (Hughes et al., 2018). While these approaches may achieve cooperation, they could well do so for the wrong reasons. We instead are interested in achieving cooperation only where *self-interest* warrants it. Otherwise our policies may end up exploitable by other self-interested agents.

We take inspiration from the recent work Learning with Opponent-Learning Awareness (LOLA; Foerster et al., 2018c; Foerster et al., 2018b), the first general learning algorithm to find tit-for-tat on IPD. LOLA mitigates the problems of simultaneous gradient descent

by looking ahead: it simulates a naive update and evaluates the objective at the resulting point. By differentiating through the naive update, LOLA effectively uses second-order information to account for the opponent's learning process.

Much of the work following LOLA focused on *opponent shaping*: intentionally influencing the opponent's learning process to our benefit. This typically refers to dynamical exploitation of the opponent's learning process, but similar effects can in some cases be achieved by taking on a fixed policy with a threat (such as tit-for-tat or a ZD-extortion policy).

Our proposal improves on LOLA in two aspects. It is self-consistent: it does not assume its opponent is naive, and it is aware of its own learning as well. Moreover, it looks more than one step ahead, through a discounted sum formulation. The discounted sum corresponds to the value function of a meta-game in which policies are meta-states and policy changes are meta-actions. We are not the first to consider this meta-value. Meta-PG (Al-Shedivat et al., 2017) and Meta-MAPG (Kim et al., 2021) use policy gradients to find policies with high meta-value. However, they estimate the meta-value by extrapolation using naive learning, and hence are inconsistent with the true process by which policies are being updated.

M-FOS (Lu et al., 2022) uses policy gradients to find arbitrary parametric meta-policies. This solves the inconsistency, as the learned meta-policy can be used for extrapolation. However, we argue that this throws out the baby with the bathwater: the meta-policy no longer takes the form of gradual policy improvement that is characteristic of learning. M-FOS does not learn to *learn* with learning awareness so much as learn to *act* with learning awareness. Nevertheless, its ability to derail naive and LOLA learners through arbitrarily abrupt policy changes does provide an interesting upper bound on what can be achieved through opponent shaping.

We make the following contributions:

- We propose Meta-Value Learning (MeVa), a general framework for learning with learning awareness. In §6.3 we propose our objective, which is naturally consistent and readily accounts for longer-term and higher-order interactions.
- Unlike prior work, our approach is based on value learning and does not require policy gradients anywhere. However, we provide a variant (§6.4.1) that mitigates the downsides of bootstrapping, generally at the cost of requiring policy gradients on the inner game.
- Our Q-function is implemented implicitly through a state-value function V (§6.3.3). We approximate the greedy (argmax) action as the gradient of V, sidestepping the need to explicitly represent the continuous meta-action space.

– We demonstrate the importance of looking far ahead in §6.5.1, and show qualitatively how the gradient of the meta-value leads to the Pareto-efficient solution regardless of initialization.
– In a tournament on repeated matrix games (§6.5.2), MeVa exhibits opponent-shaping behavior that nearly matches that of M-FOS. In particular, we find ZD-extortion (Press and Dyson, 2012) on the IPD, and dynamical exploitation on Iterated Matching Pennies.

Code is available at `https://github.com/MetaValueLearning/MetaValueLearning`.

6.2. Background

We consider differentiable games $f : \mathbb{R}^{2\times n} \mapsto \mathbb{R}^2$ that map pairs of policies to pairs of expected returns,

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

or $y = f(x)$ for short. For simplicity, we assume $x_1, x_2 \in \mathbb{R}^n$ are real-valued parameter vectors that represent policies through some fixed parametric class (e.g. a lookup table or a fixed neural network architecture).

6.2.1. Naive Learning

Under naive learning (also known as "simultaneous gradient descent"), agents update their policies according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} f(x^{(t)}) \tag{6.2.1}$$

where α is a learning rate and $\bar{\nabla} f(x) = \text{blockdiag}(f'(x)) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) \end{pmatrix}^\top$ denotes the simultaneous gradient.

Naive learning is the straightforward application of standard gradient descent which works extremely well when optimizing a single objective, e.g. a single-agent system or a supervised learning problem. However, the gradient derives from a local first-order model – for each element of x, it reflects the change in f that can be attained by modifying that element, but only if all other elements remain constant. For optimization problems with a single objective, this assumption can profitably be violated, but when different elements of x optimize different objectives, the gradient generally fails to be a reliable direction of improvement.

6.2.2. Looking Ahead

A number of approaches in the literature aim to address these issues by "looking ahead", considering not just the current parameter values $x^{(t)}$ but also an extrapolation

based on an imagined update. They essentially replace the game f with a surrogate game $\tilde{f}$ that evaluates f after an imagined naive update with learning rate α:[1]

$$\tilde{f}(x) = f(x + \alpha \bar{\nabla} f(x)). \tag{6.2.2}$$

This surrogate was to our knowledge first suggested with LookAhead (Zhang and Lesser, 2010), though in computing the associated update $\bar{\nabla}\tilde{f}$, the authors considered $\bar{\nabla} f(x)$ constant with respect to x. LOLA (Foerster et al., 2018c) introduced the idea of differentiating through the imagined update $\bar{\nabla} f(x)$, thus incorporating second-order information that accounts for interactions between the learning of agents.

Unfortunately, this surrogate trades one assumption for another: while it no longer assumes players to stand still, it now assumes players to update according to naive learning. The LOLA authors also proposed Higher-Order LOLA (HOLA (Foerster et al., 2018c)), where HOLA0 assumes opponents are fixed, HOLA1 assumes opponents are naive learners, HOLA2 assumes opponents use LOLA, and so on. Nevertheless there is always a gap where each player assumes it looks one step further ahead than their opponent. To avoid such an assumption, we should like to use a consistent surrogate such as

$$\tilde{f}^\star(x) = f(x + \alpha \bar{\nabla} \tilde{f}^\star(x)), \tag{6.2.3}$$

however this is an implicit equation; it is unclear how to obtain the associated update $\bar{\nabla}\tilde{f}^\star(x)$.

COLA (Willi et al., 2022) solves the implicit equation by replacing the surrogate with a model $\hat{g}(x;\theta) \approx \bar{\nabla}\tilde{f}^\star(x)$. The model is trained to satisfy

$$\hat{g}(x;\theta) = \bar{\nabla} f(x + \alpha \hat{g}(x;\theta)) \tag{6.2.4}$$

by minimizing the squared error between both sides. When the equation is tight, $\hat{g}(x;\theta) = \bar{\nabla}\tilde{f}^\star(x)$ which lets us train policies according to the consistent surrogate (6.2.3).

6.2.3. Going Meta

Several approaches address LOLA's short-sightedness through a meta-game, where meta-states are policy pairs x and meta-rewards are policy returns $f(x)$. The meta-value is generally defined as the infinitely discounted sum

$$V^\pi(x^{(t)}) = \mathop{\mathbb{E}}_{x^{(>t)}} \sum_{\tau=t}^{\infty} \gamma^{\tau-t} f(x^{(\tau)}), \tag{6.2.5}$$

[1] LookAhead (Zhang and Lesser, 2010), LOLA (Foerster et al., 2018c) and COLA (Willi et al., 2022) differ slightly from (6.2.2) in that each player extrapolates only their opponent and not themselves. We provide an exact formulation of LOLA and COLA in Appendix 6.B.

where the optimization process

$$\begin{aligned} x_1^{(t+1)} &\sim \pi_1(\,\cdot \mid x^{(t)}) \\ x_2^{(t+1)} &\sim \pi_2(\,\cdot \mid x^{(t)}) \end{aligned}$$

comes about through a pair of (assumed Markov) meta-policies π_1,π_2. In this context, gradient methods like naive learning and LOLA can be seen as deterministic meta-policies, although they become stochastic when policy gradients are involved, and non-Markov when path-dependent information like momentum is used.

Meta-PG (Al-Shedivat et al., 2017) was the first to consider such a meta-game, applying policy gradients to maximize V^π with respect to x, with π assumed to be naive learning on f. Meta-MAPG (Kim et al., 2021) tailored Meta-PG to multi-agent learning, taking the learning process of other agents into account. However, Meta-MAPG (like Meta-PG) assumes all agents use naive learning, and hence is inconsistent like LOLA.

M-FOS (Lu et al., 2022) allows arbitrary parametric meta-policies π_θ, training θ to maximize V^{π_θ}, thus solving the inconsistency. However, the move to arbitrary meta-policies, away from gradient methods, discards the gradual dynamics that are characteristic of learning. M-FOS disrupts naive and LOLA learners through arbitrarily fast policy changes.

6.3. Meta-Value Learning

We now describe our method. First we introduce the meta-value function, a consistent and far-sighted surrogate, to be used in place of the original game f. Next, we make a connection to reinforcement learning, which yields a straightforward way of approximating the surrogate.

6.3.1. The Meta-Value Function

We propose to use the surrogate

$$V(x) = f(x) + \gamma V(x + \alpha \bar{\nabla} V(x)) \tag{6.3.1}$$

which consists of the original objective f plus a discounted sum of objective values at future optimization iterates. Like the popular surrogate from (6.2.2), it looks ahead in optimization time, but it does so in a way that is consistent and naturally covers multiple steps. Our meta-value function is a special case of (6.2.5), with a deterministic meta-policy pair θ_1,θ_2 that updates x according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} V(x^{(t)}). \tag{6.3.2}$$

This can be viewed as a consistent version of Meta-MAPG (Kim et al., 2021), and a version of M-FOS (Lu et al., 2022) with gradient-based meta-policy π. Unlike those approaches, we do not require any REINFORCE-style gradient approximations.

6.3.2. Learning Meta-Values

We do not have direct access to the meta-value function, but we can learn an approximation $\hat{V}(x;\theta)$ with parameters θ. In its simplest form, the learning process follows a nested loop (see Algorithm 3). In the inner loop, we collect a policy optimization trajectory according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} \hat{V}(x^{(t)};\theta). \tag{6.3.3}$$

Then in the outer loop, we train $\hat{V}$ by minimizing with respect to θ the TD error

$$\sum_t \|\hat{f}(x^{(t)}) + \gamma \hat{V}(x^{(t+1)};\bar{\theta}) - \hat{V}(x^{(t)};\theta)\|^2 \tag{6.3.4}$$

along the trajectory. Here $\hat{f}(x^{(t)})$ is in general an empirical estimate of the expected return $f(x^{(t)})$ based on a batch of Monte-Carlo rollouts, although in our experiments we use the exact expected return. The target involves $\bar{\theta}$, typically a target network that lags behind θ (Mnih et al., 2015).

We write $\hat{V}(x;\theta)$ for convenience. However, in order to ensure self-interest, we maintain a separate model for each agent, with disjoint parameters:

$$\hat{V}(x;\theta) = \begin{pmatrix} \hat{V}_1(x;\theta_1) \\ \hat{V}_2(x;\theta_2) \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}.$$

As training progresses, $\hat{V} \to V$ and hence $\bar{\nabla}\hat{V} \to \bar{\nabla}V$, and so (6.3.3) approaches the proposed update (6.3.2). Note that rather than emitting entire gradients (as COLA does) or emitting entire policies (as M-FOS does), we model scalars, and estimate the gradient of the scalar by the gradient of the estimated scalar. The resulting algorithm is related to Value-Gradient Learning (Fairbank and Alonso, 2012), but we do not directly enforce a Bellman equation on the gradients $\bar{\nabla}\hat{V}$.

6.3.3. Q-learning interpretation

In this section we establish a theoretical link between the gradient $\bar{\nabla}V$ and the action that greedily (if locally) maximizes the state-action value Q. Focusing on player 1, we have

$$Q_1(x,x_1+\Delta_1) = \mathop{\mathbb{E}}_{\Delta_2 \sim \pi_2} V_1(x+\Delta),$$

for some proposed updates Δ_1,Δ_2. We construct a first-order Taylor approximation $\tilde{Q}$ of Q around x:

$$\tilde{Q}_1(x,x_1+\Delta_1) = V_1(x) + \Delta_1^\top \nabla_{x_1} V_1(x) + \Delta_2^\top \nabla_{x_2} V_2(x).$$

Algorithm 3 Meta-Value Learning.

Require: Learning rates η,α, rollout length T, fixed discount rate γ.
  Initialize models θ_1,θ_2.
  while θ has not converged **do**
    Initialize policies $x^{(0)}$.
    for t=0,...,T **do**
      $x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} \hat{V}(x^{(t)}; \theta)$
    end for
    $\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{T} \sum_t \| f(x^{(t)}) + \gamma V(x^{(t+1)}; \theta) - V(x^{(t)}; \theta) \|^2$
  end while

This approximation is justified when $\|\Delta\|^2$ is small, i.e. both agents are making small updates.

We now proceed to maximize (6.3.3) to find the argmax of $\tilde{Q}$, as a proxy for the argmax of Q. If we use a soft norm penalty, we arrive at exactly our update:

$$\text{argmax}_{\Delta_1} \tilde{Q}(x, x_1 + \Delta_1) - \frac{1}{2\alpha} \|\Delta_1\|^2 = \alpha \nabla_{x_1} V_1(x).$$

Alternatively, we may apply a hard norm constraint, which would admit a treatment from the perspective of a game with a well-defined *local* action space. Either way, the argmax update will be proportional to $\nabla_{x_1} V_1(x)$, which lends an interpretation to our use of the gradient in (6.3.2) as choosing actions greedily in Q.

Our method is thus related to independent Q-learning (Watkins and Dayan, 1992; Busoniu et al., 2008), which we must point out is not known to converge in general-sum games. It nevertheless does appear to converge reliably in practice, and we conjecture that applying it on the level of optimization effectively simplifies the interaction between the agents' learning processes.

6.4. Practical Considerations

We use a number of established general techniques to improve the dynamics of value function approximation (Hessel et al., 2018). The prediction targets in (6.3.4) are computed with a target network (Mnih et al., 2015) that is an exponential moving average of the parameters θ. We use distributional reinforcement learning with quantile regression (Dabney et al., 2018). Instead of the fully-bootstrapped TD(0) error, we use λ-returns (Sutton and Barto, 2018) as the targets, computed individually for each quantile.

The rest of this section describes a number of additional techniques designed to mitigate the negative effects of bootstrapping and to encourage generalization. Algorithm 4 in Appendix 6.C lays out the complete learning process in detail.

6.4.1. Reformulation as a Correction

We provide a variant of the method that provides a correction to the naive gradient $\bar{\nabla} f$ rather than replacing it entirely. If we define $U(x) = V(x')$ with $x' = x + \alpha \bar{\nabla} V(x))$ then we have the Bellman equation

$$U(x) = V(x') = f(x') + \gamma U(x').$$

Now agents follow the gradient field $\bar{\nabla} f(x) + \gamma \bar{\nabla} U(x)$, and we minimize

$$\sum_t \|\hat{f}(x^{(t+1)}) + \gamma \hat{U}(x^{(t+1)}; \theta) - \hat{U}(x^{(t)}; \theta)\|^2$$

with respect to the parameters θ of our model $\hat{U}(x; \theta)$.

This variant is more strongly grounded in the game f and helps avoid the detachment from reality that plagues bootstrapped value functions. A drawback of this approach is that now the naive gradient term $\bar{\nabla} f(x)$ will usually have to be estimated by REINFORCE (Williams, 1992).

6.4.2. Variable Discount Rates

We also set up the model (be it $\hat{U}$ or $\hat{V}$) to condition on discount rates γ_1, γ_2, so that we can train it for different rates and even rates that differ between the players. This is helpful because it forces the model to better understand the given policies, in order to distinguish policies that would behave the same under some fixed discount rate but differently under another. During training, we draw $\gamma_1, \gamma_2 \sim \text{Beta}(1/2, 1/2)$ from the standard arcsine distribution to emphasize extreme values.

Varying γ affects the scale of V,U and hence the scale of our approximations to them. This in turn results in a change in the effective learning rate when we take gradients. To account for this, we can normalize the outputs and gradients of V,U by scaling by $1 - \gamma$ before use. However, we instead choose to multiply the meta-reward term $f(x)$ in the Bellman equation by $1 - \gamma$. This ensures our models learn the normalized values instead, which will fall in the same range as $f(x)$. Appendix 6.A has a derivation.

When the context calls for it, we will make this structure explicit by writing $(1 - \gamma) \odot f(x) + \gamma \odot \hat{U}(x; \theta, \gamma)$, where γ is now a vector and $\odot$ denotes the elementwise product.

6.4.3. Exploration

Our model $\hat{V}$ provides a deterministic (meta-)policy for changing the inner policies x. For effective value learning, however, we need exploration as well as exploitation. A straightforward way to introduce exploration into the system is to perturb the greedy transition in (6.3.3) with some additive Gaussian noise (Heess et al., 2015). However,

this leads to a random walk that fails to systematically explore the state space. Instead of perturbing the actions, we perturb the policy by applying noise to the parameters θ, and hold the perturbed policy fixed over the course of an entire optimization trajectory. Specifically, we randomly flip signs on parameters in the final layer of $\hat{V}$; this results in a perturbed value function that incentivizes different high-level characteristics of the inner policies x. The trajectories so collected are entirely off-policy and serve only to provide a variety of states. In order to train our model on a given state x, we do a short on-policy rollout with the unperturbed parameters θ and minimize TD error there. In Algorithm 4 we implement this in a strided fashion: the off-policy exploration is halted every k steps to perform k steps of on-policy rollout.

6.5. Experiments

The method is evaluated on four environments. First, we demonstrate the advantage of being able to look farther ahead on a two-dimensional game that is easy to visualize. Next, we evaluate opponent shaping on the IPD, IMP and Chicken games by pitting MeVa head-to-head against naive and LOLA agents.

6.5.1. Logistic Game

We analyze the behavior of several algorithms on the Logistic Game (Letcher, 2018), a two-player game where each player's policy is a single scalar value. Thus the entire joint policy space is a two-dimensional plane, which we can easily visualize. The game is given by the function[2]

$$f(x) = -\begin{pmatrix} 4\sigma(x_1)(1-2\sigma(x_2)) \\ 4\sigma(x_2)(1-2\sigma(x_1)) \end{pmatrix} - \frac{x_1^2 x_2^2 + (x_1 - x_2)^2 (x_1 + x_2)^2}{10000}. \tag{6.5.1}$$

Figure 6.1 shows the structure of the game. There are two stable fixed points – one in the lower left (A) and one in the upper right (B). Both players prefer B to A, however to get from A to B requires coordination: the horizontal player prefers left if the vertical player plays low and vice versa.

We look at this game through the lens of basins of attraction, and how different algorithms affect them (Figure 6.2b). Following naive gradients (LOLA/HOLA2 with $\alpha = 0$), players converge to whichever solution is nearest; the basins of attraction meet at a diagonal line through the origin. LOLA grows the basin of the preferred solution B, but only slightly and increasing the extrapolation step size α does not help much. HOLA2 grows the basin of B around the edges, but suffers from instabilities around the origin (a saddlepoint).

[2] Letcher (2018) use the divisor 1000 in Eqn (6.5.1), however it does not match their plots. Moreover we have flipped the sign to turn this into a maximization problem in accordance with our notation.

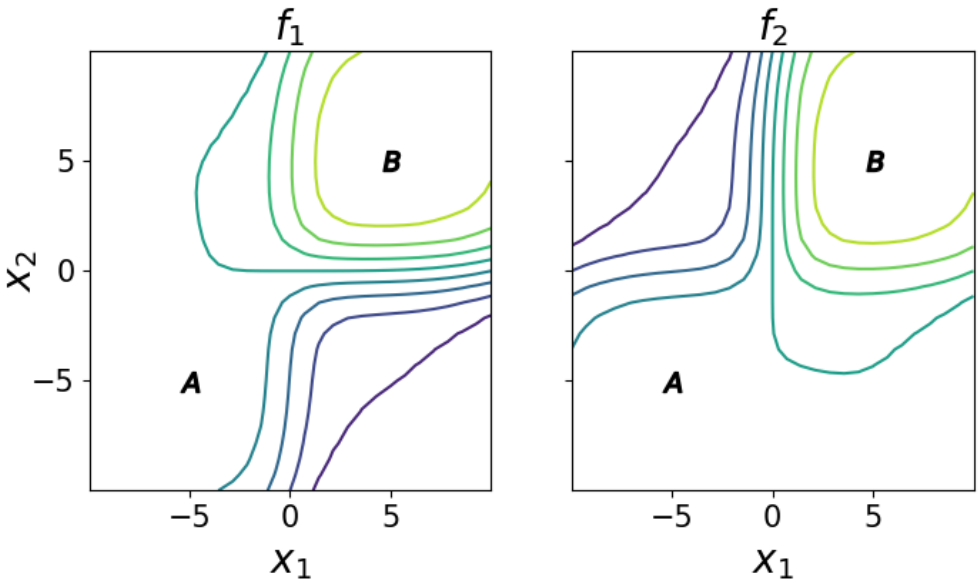


Fig. 6.1. The Logistic Game. The left panel displays the contours of player 1's objective $f_1(x)$, the right panel similarly for player 2. Player 1's policy x_1 is a horizontal position, player 2's policy x_2 is a vertical position. Both players prefer solution B over solution A, but cannot unilaterally go there. Naive learning converges to whichever solution is closest upon initialization.

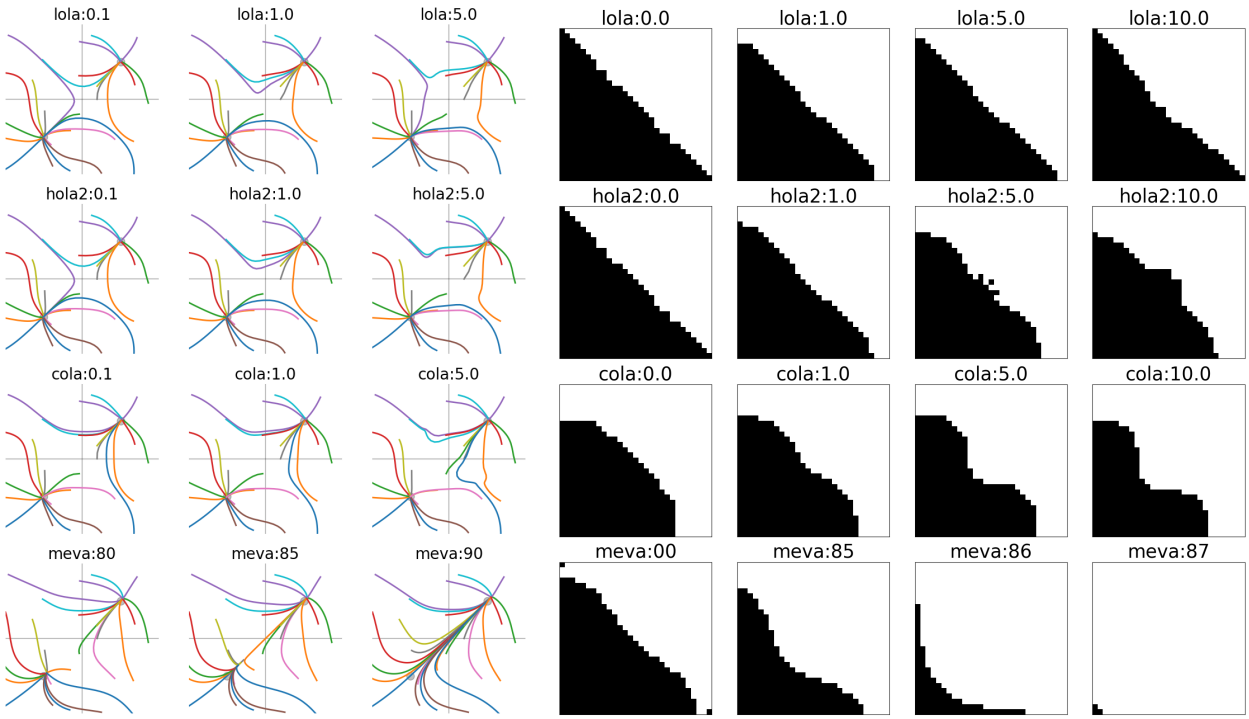
We found HOLA3 to be significantly worse than HOLA2 and did not pursue that direction further. COLA (our implementation) makes significant improvements around the edges and around the origin. Finally, MeVa is able to make the basin of B arbitrarily large. When $\gamma > 0.9$, it converges to the preferred solution B from anywhere in the surveyed area. We also show some actual optimization trajectories in Figure 6.2a.

The meta-value approach gives us an additional hyperparameter γ to control the extent to which we look ahead. By increasing γ, we can make the basin of B arbitrarily large. Even if agents initialize close to A, it is worth moving in a direction that immediately decreases f, because they know (through the gradient of the meta-value) that doing so will eventually increase f.

Experiment details can be found in Appendix 6.D.

6.5.2. Matrix Games

We evaluate our method on several repeated matrix games by going head-to-head with naive learners and LOLA. M-FOS is the closest method from the literature. However, we do not provide head-to-head comparison with M-FOS as doing so would require training M-FOS and MeVa jointly, which is prohibitively complex. Instead, we compare our performance versus Naive and LOLA with that of M-FOS.



(a) Optimization trajectories. We took a random set of policy pairs and, for each panel, optimized them according to the algorithm under consideration. Each curve shows an optimization trajectory, typically finishing in or close to either A or B.

(b) Basins of attraction. For each panel, we took a grid of policy space points and optimized them according to the algorithm under consideration. White cells indicate that the corresponding point ended up in the positive quadrant x_1,$x_2 > 0$, black cells ended up in other quadrants (typically the negative quadrant).

Fig. 6.2. Logistic Game behaviors of different algorithms (rows) with different settings (columns).

Table 6.1. Payoffs for the matrix games considered.

	A	B
A	$(-1,-1)$	$(-3,\ 0)$
B	$(0,-3)$	$(-2,-2)$

(a) Iterated Prisoner's Dilemma

	A	B
A	$(+1,-1)$	$(-1,+1)$
B	$(-1,+1)$	$(+1,-1)$

(b) Iterated Matching Pennies

	A	B
A	$(0,\ 0)$	$(-1,+1)$
B	$(+1,-1)$	$(-100,-100)$

(c) Chicken Game

We use the same setup as Lu et al. (2022): policies $x_i \in \mathbb{R}^5$ consist of five binary logits, corresponding to the probability of playing action A or B in each of five possible states (the initial state and the previous joint action AA, AB, BA, BB). We use the exact value function given in Foerster et al. (2018c) with discount rate 0.96 (not to be confused with our meta-discount rate γ). Unlike Foerster et al. (2018c) and Lu et al. (2022), we work with the *normalized* value, and hence our learning rates must be $1/(1-0.96) = 25$ times larger to match theirs. The games differ only in their payoff matrices, which are shown in Table 6.1.

	Naive	LOLA	M-FOS	MeVa
Naive	-1.99	-1.56	-2.02	-1.98
LOLA	-1.49	-1.09	-1.02	-1.08
M-FOS	**-0.56**	-1.02	**-1.01**	
MeVa	-0.57	**-0.98**		**-1.06**

Table 6.2. Head-to-head comparison of learning rules on Iterated Prisoner's Dilemma. MeVa extorts the naive learner, and appears to exploit LOLA to a small extent.

	Naive	LOLA	M-FOS	MeVa
Naive	0.00	0.02	-0.20	-0.23
LOLA	-0.02	-0.02	-0.19	-0.16
M-FOS	0.20	**0.19**	**0.00**	
MeVa	**0.23**	0.16		**0.00**

(a) Head-to-head comparison of learning rules on Iterated Matching Pennies. Like M-FOS, MeVa is able to significantly (dynamically) exploit both the naive learner and LOLA.

	Naive	LOLA	M-FOS	MeVa
Naive	0.01	**-0.71**	-1.03	-0.93
LOLA	0.70	-5.32	**0.79**	**0.86**
M-FOS	**0.97**	-1.16	-0.01	
MeVa	0.92	-1.11		-0.05

(b) Head-to-head comparison of learning rules on the Chicken Game. MeVa exploits the naive learner while avoiding disasters against itself. LOLA exploits every opponent but does poorly against itself, an observation also made by Lu et al. (2022).

For each game, we train 10 models from different seeds, and produce 2048 policy pairs from each model. We use meta-discount rate $\gamma = 0.95$, and learning rates $\alpha = 25$ (except on the Chicken Game, where we use $\alpha = 1$ to accommodate the large payoffs). For M-FOS we found different numbers than those reported in Lu et al. (2022), so to be sure we trained 30 models with 4096 policy pairs each, using their code and hyperparameters ($\alpha = 25$ on IPD and the Chicken Game, $\alpha = 2.5$ on IMP). Further detail, including standard errors on these results, can be found in Appendix 6.E.

On the **Iterated Prisoner's Dilemma** (Table 6.2), MeVa extorts the naive learner (details in Appendix 6.F), and LOLA to a small extent. The behavior is similar to that of M-FOS, although M-FOS leads the naive agent to accept returns below -2, indicating dynamical exploitation.

On **Iterated Matching Pennies** (Table 6.3a), M-FOS and MeVa both significantly exploit naive and LOLA learners. ZD-extortion is not possible in zero-sum games, so MeVa exhibits dynamical exploitation just like M-FOS.

On the **Chicken Game** (Table 6.3b), LOLA exploits every opponent but does poorly against itself (as also observed in (Lu et al., 2022)). MeVa exploits the naive learner while avoiding disasters against itself. However, it yields to LOLA, more so than the naive learner. This is consistent with M-FOS's results.

6.5.3. Ablation

To evaluate the impact of the practical techniques from § 6.4, we perform an ablation experiment on the Iterated Prisoner's Dilemma. In Figure 6.3, we show the effect of using the V formulation over the U formulation, disabling exploration, disabling target networks, disabling λ-returns, and disabling distributional RL in favor of least squares regression.

The configurations differ in the extent to which they reduce the true error (estimated by the long-term TD error), while minimizing the loss (measured by the short-term TD error). The difference between these two is due to bootstrapping. Moreover, there are differences in the rate at which the agents find cooperation, and the stability of that state.

We found the U formulation to be the most impactful; abandoning it hurts the convergence and stability of the process. Target networks and λ-returns appear detrimental to the speed of learning as part of their stabilizing effect. Exploration appears to have the least marginal impact.

6.6. Limitations

The meta-value function is a scalar function over (joint) policies. In practice, policies will often take the form of neural networks, and so will our approximations to the meta-value function. Conditioning neural networks on other neural networks is a major challenge (Harb et al., 2020). In addition, the large parameter vectors associated with neural networks will quickly prohibit handling batched optimization trajectories.

During training and opponent shaping, we assume opponent parameters to be visible to our agent. This is not necessarily unrealistic – we learn and use the meta-value only as a means to the end of finding good policies x for the game f that can then be deployed in the wild without further training. Nevertheless, the algorithm could be extended to work with opponent models, or more directly, the model could observe policy behaviors instead of parameters.

The meta-discount rate γ, like LOLA's step size α, is hard to interpret. Its meaning changes significantly depending on the learning rate α and the parameterization of both the model $\hat{V}$ and the policies x. Future work could explore the use of proximal updates, like POLA (Zhao et al., 2022) did for LOLA.

Finally, it is well known that LOLA fails to preserve the Nash equilibria of the original game f. The method presented here shares this property.

6.7. Conclusion

We have introduced Meta-Value Learning (MeVa), a naturally consistent and far-sighted approach to learning with learning awareness. MeVa derives from a meta-game similar to

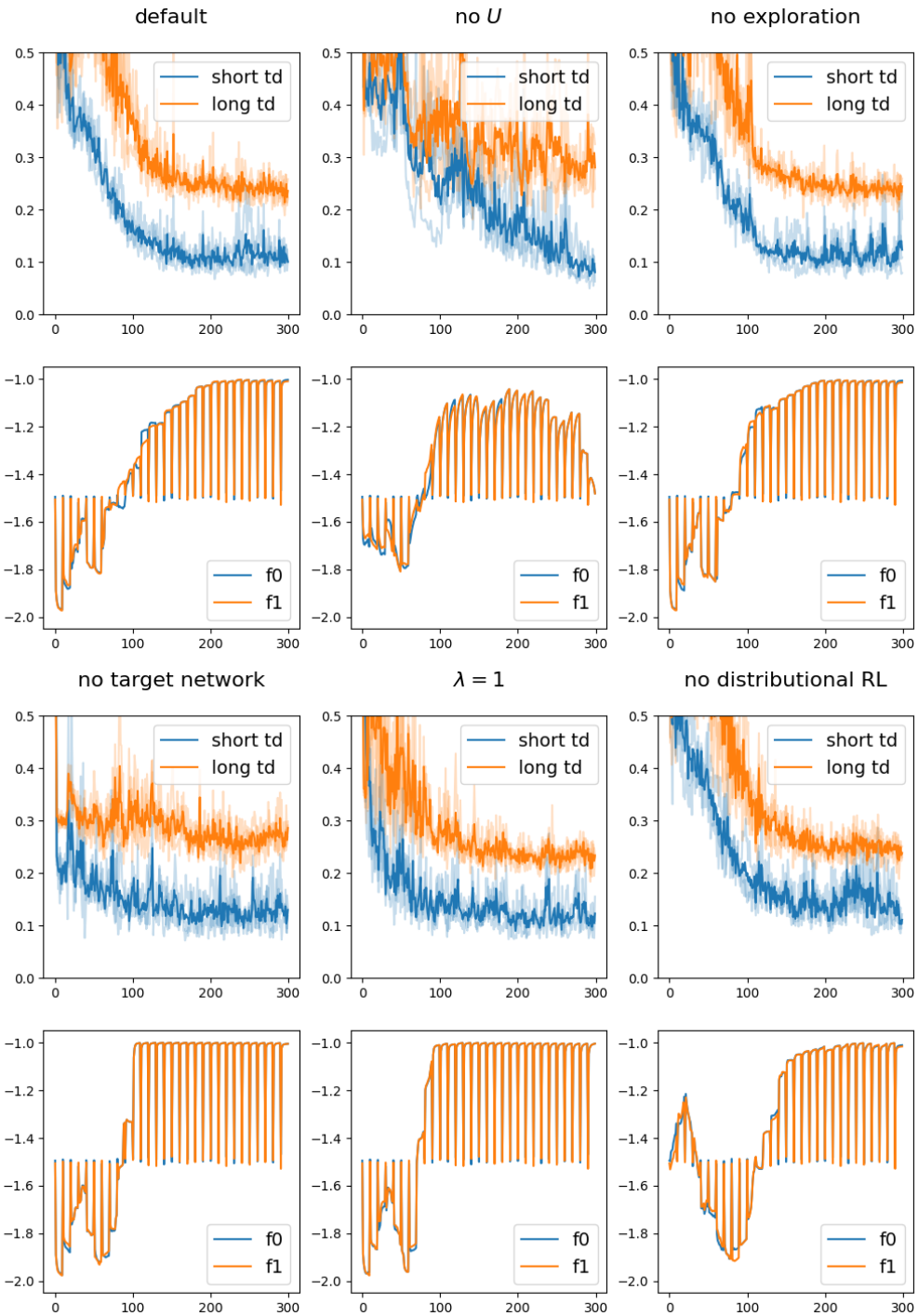


Fig. 6.3. Ablation experiment on the Iterated Prisoner's Dilemma. We show the effect of using the V formulation over the U formulation, and disabling exploration, target networks, λ-returns, and distributional RL. For each configuration we train 3 models for 300 outer loops. We show short-term TD error (over k steps, as in training) and long-term TD error (over 100 steps, as a validation); the difference between these is due to bootstrapping. The horizontal axis measures number of outer loops performed. We also show the returns $f(x)$ of agents that are being trained on the model (with $\gamma = 0.95$), and are reset every 10 outer loops.

that considered in prior work (Al-Shedivat et al., 2017; Kim et al., 2021; Lu et al., 2022), and can be seen as the Q-learning complement to the policy gradient of M-FOS (Lu et al., 2022), although we choose different meta-action spaces. MeVa requires no REINFORCE-style gradient approximations, and avoids explicitly modeling the continuous action space by parameterizing the action-value function implicitly through a state-value function.

The opponent-shaping capabilities of our method are similar to those of M-FOS, although we are strictly dominated in that respect. We do find ZD-extortion on the general-sum IPD, and dynamical exploitation on the zero-sum IMP.

The main weakness of the method as it stands is scalability, particularly to policies that take the form of neural networks. We aim to address this in future work using policy fingerprinting (Harb et al., 2020).

Finally, we note that although we develop our method in the context of multi-agent reinforcement learning, it is a general meta-learning approach that readily applies to optimization problems with a single objective.

Acknowledgements

This research was enabled in part by compute resources provided by Mila. We used the JAX (Bradbury et al., 2018) library for scientific computing. We thank Cheng-Zhi (Anna) Huang, Kyle Kastner, David Krueger, Christos Tsirigotis, Michael Noukhovitch, Amartya Mitra, Juan Augustin Duque and Shunichi Akatsuka for helpful discussion.

References

Axelrod, Robert and William D Hamilton (1981). "The evolution of cooperation". In: *science* 211.4489, pp. 1390–1396.

Baker, Bowen (2020). "Emergent reciprocity and team formation from randomized uncertain social preferences". In: *Advances in Neural Information Processing Systems* 33, pp. 15786–15799.

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: `http://github.com/google/jax`.

Busoniu, Lucian, Robert Babuska, and Bart De Schutter (2008). "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2, pp. 156–172.

Dabney, Will et al. (2018). "Distributional reinforcement learning with quantile regression". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Fairbank, Michael and Eduardo Alonso (2012). "Value-gradient learning". In: *The 2012 international joint conference on neural networks (ijcnn)*. IEEE, pp. 1–8.

Foerster, Jakob et al. (2018a). "Counterfactual multi-agent policy gradients". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1.

Foerster, Jakob et al. (2018b). "Dice: The infinitely differentiable monte carlo estimator". In: *International Conference on Machine Learning*. PMLR, pp. 1529–1538.

Foerster, Jakob et al. (2018c). "Learning with Opponent-Learning Awareness". In: *International Conference on Autonomous Agents and Multiagent Systems*.

Gronauer, Sven and Klaus Diepold (2022). "Multi-agent deep reinforcement learning: a survey". In: *Artificial Intelligence Review*, pp. 1–49.

Harb, Jean et al. (2020). "Policy evaluation networks". In: *arXiv preprint arXiv:2002.11833*.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Heess, Nicolas et al. (2015). "Learning continuous control policies by stochastic value gradients". In: *Advances in neural information processing systems* 28.

Hendrycks, Dan and Kevin Gimpel (2016). "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415*.

Hessel, Matteo et al. (2018). "Rainbow: Combining improvements in deep reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1.

Hughes, Edward et al. (2018). "Inequity aversion improves cooperation in intertemporal social dilemmas". In: *Advances in neural information processing systems* 31.

Jaques, Natasha et al. (2019). "Social influence as intrinsic motivation for multi-agent deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 3040–3049.

Kim, Dong Ki et al. (2021). "A policy gradient algorithm for learning to learn in multiagent reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 5541–5550.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Lauer, Martin (2000). "An algorithm for distributed reinforcement learning in cooperative multiagent systems". In: *Proc. 17th International Conf. on Machine Learning*.

Letcher, Alistair (2018). "Stability and Exploitation in Differentiable Games". MA thesis.

Loshchilov, Ilya and Frank Hutter (2017). "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101*.

Lowe, Ryan et al. (2017). "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/68a9750337a418a86fe06c1991a1d64c-Paper.pdf.

Lu, Christopher et al. (2022). "Model-free opponent shaping". In: *International Conference on Machine Learning*. PMLR, pp. 14398–14411.

Matignon, Laëtitia, Guillaume J Laurent, and Nadine Le Fort-Piat (2007). "Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams". In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 64–69.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Panait, Liviu and Sean Luke (2005). "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11, pp. 387–434.

Press, William H and Freeman J Dyson (2012). "Iterated Prisoner's Dilemma contains strategies that dominate any evolutionary opponent". In: *Proceedings of the National Academy of Sciences* 109.26, pp. 10409–10413.

Al-Shedivat, Maruan et al. (2017). "Continuous adaptation via meta-learning in nonstationary and competitive environments". In: *arXiv preprint arXiv:1710.03641*.

Silver, David et al. (2017). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway networks". In: *arXiv preprint arXiv:1505.00387*.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8, pp. 279–292.

Willi, Timon et al. (2022). "COLA: consistent learning with opponent-learning awareness". In: *International Conference on Machine Learning*. PMLR, pp. 23804–23831.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement Learning*, pp. 5–32.

Zhang, Chongjie and Victor Lesser (2010). "Multi-agent learning with policy prediction". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1, pp. 927–934.

Zhao, Stephen et al. (2022). "Proximal Learning With Opponent-Learning Awareness". In: *Advances in Neural Information Processing Systems* 35, pp. 26324–26336.

6.A. Normalized Bellman Equation

We introduce a slight tweak to the usual Bellman equation

$$V(s_t) = r_t + \gamma V(s_{t+1})$$

to address a scaling issue. Suppose the rewards fall in the range $(0, 1)$. Then V will take values in the range $\left(0, \frac{1}{1-\gamma}\right)$. We can multiply through by $1-\gamma$ to obtain a new equation that will force values $\tilde{V}$ into the same range as the rewards:

$$\begin{aligned}(1-\gamma)V(s_t) &= (1-\gamma)r_t + \gamma(1-\gamma)V(s_{t+1}) \\ \tilde{V}(s_t) &= (1-\gamma)r_t + \gamma\tilde{V}(s_{t+1}).\end{aligned}$$

This is useful in our case because otherwise changing γ would affect the scale of the gradient $\bar{\nabla}V$ and hence change the effective learning rate. We also found it helpful when using distributional RL with a fixed binning – the convex-combination form of the right-hand side guarantees it will fall within the same range as the left-hand side. Finally, we find the normalized values easier to interpret.

6.B. LOLA, HOLA, COLA

In §6.2 we described LOLA, HOLA and COLA on an abstract level, ignoring in particular that each of them involves extrapolating only the opponent. In this section we provide exact definitions for completeness.

For notational convenience in working with the concatenated vector x we define the slicing matrices

$$S_1 = \frac{dx}{dx_1}^\top = \begin{pmatrix} I_n & 0_n \end{pmatrix}, \quad S_2 = \frac{dx}{dx_2}^\top = \begin{pmatrix} 0_n & I_n \end{pmatrix},$$

where I_n is the $n \times n$ identity matrix and 0_n the $n \times n$ zero matrix, $n = \dim(x_1)$ being the dimensionality of a policy parameter vector. Now $x_1 = S_1 x, x_2 = S_2 x$ and $x = S_1^\top x_1 + S_2^\top x_2$.

LOLA (Foerster et al., 2018b) (sans Taylor approximation) uses the update

$$\bar{\nabla} f^{\text{LOLA}}(x) = \bar{\nabla} \begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f(x)) \end{pmatrix}.$$

This differs from what we presented in (6.2.2) in that each player only considers their opponent's update, and not their own.

HOLAn (Foerster et al., 2018c; Foerster et al., 2018b) (again sans Taylor approximation) applies the LOLA surrogate recursively to itself:

$$\bar{\nabla} f^{\text{HOLA}(n)}(x) = \bar{\nabla}\begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f^{\text{HOLA}(n-1)}(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f^{\text{HOLA}(n-1)}(x)) \end{pmatrix},$$

with base case $\bar{\nabla} f^{\text{HOLA}(0)}(x) = \bar{\nabla} f(x)$. Notice that HOLA1 recovers LOLA: $\bar{\nabla} f^{\text{HOLA}(1)}(x) = \bar{\nabla} f^{\text{LOLA}}(x)$.

COLA (Willi et al., 2022) considers

$$\bar{\nabla} f^{\text{COLA}}(x) = \bar{\nabla}\begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f^{\text{COLA}}(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f^{\text{COLA}}(x)) \end{pmatrix},$$

an implicit equation similar to (6.2.3). The equation is solved with a model $\hat{g}^{\text{COLA}}(x;\theta) \approx \bar{\nabla} f^{\text{COLA}}(x)$ of the gradient of the implicit surrogate. The model is trained to satisfy

$$\hat{g}^{\text{COLA}}(x;\theta) = \bar{\nabla}\begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \hat{g}^{\text{COLA}}(x;\theta)) \\ f_2(x + \alpha S_1^\top S_1 \hat{g}^{\text{COLA}}(x;\theta)) \end{pmatrix}$$

by minimizing the squared distance between the two sides of this equation.

6.C. Detailed Algorithm Description

Algorithm 4 describes the learning algorithm in detail, including the modifications discussed in §6.4.

After introducing quantile distributional RL, we have a model $\hat{U}(x;\gamma,\theta) \in \mathbb{R}^M$ that produces a vector of M quantiles. We denote by $\hat{U}_{\mathbb{E}}(x;\gamma,\theta)$ the expectation over the quantile distribution, which is just the uniform average of the quantile values

$$\hat{U}_{\mathbb{E}}(x;\gamma,\theta) = \frac{1}{M}\sum_{m=1}^{M} \hat{U}(x;\gamma,\theta)_m.$$

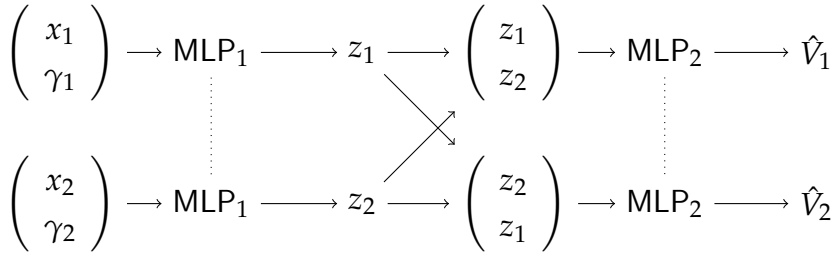
The divergence $D(\hat{y},y)$ is the quantile regression loss from (Dabney et al., 2018) between the predicted quantiles $\hat{y}$ and the target quantiles y.

6.D. Logistic Game Details

We use the $\hat{V}(x;\theta,\gamma)$ formulation. While conceptually, each agent maintains their own model $\hat{V}_1(x;\theta_1,\gamma)$, the implementation combines the computation of both models. The structure of the resulting single model can be seen in the following diagram:

Algorithm 4 Meta-Value Learning incorporating the techniques discussed in §6.4.

Require: Learning rates η,α, exploration trajectory length T, stride k, bootstrapping rate λ, target network inertia ρ.
Initialize meta-value functions θ_1,θ_2 and target networks $\bar{\theta} = \theta$.
while θ has not converged **do** ▷ outer loop
  Initialize policies $\tilde{x}^{(0)}$; draw $\tilde{\gamma}$ (§6.4.2); draw $\tilde{\theta}$ (§6.4.3).
  $t \leftarrow 0$
  while $t < T$ **do** ▷ inner loop
    for $\tau = t \dots t+k$ **do** ▷ take k exploration steps
      $\tilde{x}^{(\tau+1)} = \tilde{x}^{(\tau)} + \alpha\bar{\nabla}(1-\tilde{\gamma}) \odot \hat{f}(\tilde{x}^{(\tau)}) + \alpha\bar{\nabla}\tilde{\gamma} \odot \hat{U}_{\mathbb{E}}(\tilde{x}^{(\tau)};\tilde{\theta},\tilde{\gamma})$
    end for
    $t \leftarrow t+k$
    Let $x^{(0)} = \tilde{x}^{(t)}$; draw γ (§6.4.2). ▷ prepare to train $\hat{U}$ on $\tilde{x}^{(t)}$
    for $\tau = 0 \dots k-1$ **do** ▷ produce an on-policy trajectory of length k
      $x^{(\tau+1)} = x^{(\tau)} + \alpha\bar{\nabla}(1-\gamma) \odot \hat{f}(x^{(\tau)}) + \alpha\bar{\nabla}\gamma \odot \hat{U}_{\mathbb{E}}(x^{(\tau)};\theta,\gamma)$
    end for
    for $i \in 1,2$ **do**
      $Y_i^{(k)} = \hat{U}_i(x^{(k)};\bar{\theta}_i,\gamma) \in \mathbb{R}^M$
      for $\tau = k-1 \dots 0$ **do** ▷ compute λ-return distributions
        $Y_i^{(\tau)} = (1-\gamma_i)\hat{f}_i(x^{(\tau+1)}) + \gamma_i\Big((1-\lambda)\hat{U}_i(x^{(\tau+1)};\bar{\theta}_i,\gamma) + \lambda Y_i^{(\tau+1)}\Big)$
      end for
      $\theta_i \leftarrow \theta_i - \eta\nabla\mathcal{L}_i(\theta_i)$ where $\mathcal{L}_i(\theta_i) = \frac{1}{k}\sum_{\tau=0}^{k-1} D(\hat{U}_i(x^{(\tau)};\theta_i,\gamma), Y_i^{(\tau)})$.
    end for
    $\bar{\theta} \leftarrow \bar{\theta} + (1-\rho)(\theta - \bar{\theta})$ ▷ update target networks
  end while
end while



We first feed the x_i,γ_i pairs into a multi-layer perceptron (MLP) to obtain a representation z_i of each agent. Then for each agent we concatenate their own representation with that of their opponent. This is run through a second MLP which outputs the quantile estimates.

The dotted lines in the diagram indicate parameter sharing between the two players ($\theta_1 = \theta_2$), which exploits the symmetry of the games under consideration (specifically $f_1(x_1,x_2) = f_2(x_2,x_1)$) to improve sample efficiency of the learning process.

The MLPs consist of a residual block (Srivastava et al., 2015; He et al., 2016) sandwiched between two layer-normalized GELU (Hendrycks and Gimpel, 2016) layers. The residual block uses a layer-normalized GELU as nonlinearity, and uses learned unitwise gates to merge with the linear path. Each layer has 64 units.

We use the same model structure for COLA, albeit without quantile regression. In the case of COLA we pass the inner learning rate α in place of γ, and we trained a single model for values of $\alpha \sim U[0,10]$.

Policies $\tilde{x}_i^{(0)} \sim U(-8,+8)$ are initialized uniformly on an area around the origin.

We used hyperparameters $\alpha = 1, M = 16, T = k = 50$. In this experiment we do not use a target network or λ-returns (i.e. $\lambda = 0$), nor do we use exploration (i.e. $\tilde{\theta} = \theta$). The model is trained for 1000 outer loops using Adam (Kingma and Ba, 2014) with learning rate $\eta = 10^{-3}$ and batch size 128. Training takes about a minute on a single GPU.

6.E. Matrix Game Details

As matrix games are a step up in complexity, we use the $\hat{U}(x;\theta,\gamma)$ formulation. We use a similar shared-model structure as for the Logistic Game (see the diagram in the previous section), but introduce an elementwise scale and shift on z_1, z_2 that is *not* shared, in order to break the equivariance when the opponent is not a MeVa agent. This makes it straightforward to still model the opponent's own meta-value, which as an auxiliary task may help learn the primary task.

In the final nonlinear layer, the GELU is replaced by a hyperbolic tangent – a signed nonlinearity that enables our exploration scheme. During exploration, we flip signs on the units in this layer with Bernoulli probability $1/16$ (see §6.4.3).

Policy logits are initialized from a standard Normal distribution, i.e. $\tilde{x}^{(0)} \sim \mathcal{N}(0,I)$.

We used hyperparameters $\alpha = 2, M = 32, T = 100, k = 10, \rho = 0.99, \lambda = 0.9$. The model is trained for 2000 outer loops, using batch size 128 and AdamW (Loshchilov and Hutter, 2017) with learning rate 10^{-4} and 10^{-2} weight decay. Training the model on a matrix game takes about half an hour on a single GPU.

Tables 6.4, 6.5 and 6.6 report tournament results with standard errors. This is the standard deviation of the average of the seed performance. To estimate this, we first average the performances of the policy pairs under each seed. Then we compute the standard deviation of these results across seeds, and divide by the square root of the number of seeds to obtain the standard error. The results between naive and LOLA learners have no such errors, as they don't involve learning models from seeds.

	Naive	LOLA	M-FOS	MeVa
Naive	-1.99	-1.56	-2.02±0.06	-1.98±0.01
LOLA	-1.49	-1.09	-1.02±0.00	-1.08±0.00
M-FOS	**-0.56**±0.03	-1.02±0.00	**-1.01**±0.00	
MeVa	-0.57±0.02	**-0.98**±0.01		**-1.06**±0.01

Table 6.4. Head-to-head comparison of learning rules on Iterated Prisoner's Dilemma.

	Naive	LOLA	M-FOS	MeVa
Naive	0.00	0.02	-0.20±0.00	-0.23±0.01
LOLA	-0.02	-0.02	-0.19±0.00	-0.16±0.01
M-FOS	0.20±0.00	**0.19**±0.00	**0.00**±0.01	
MeVa	**0.23**±0.01	0.16±0.01		**0.00**±0.00

Table 6.5. Head-to-head comparison of learning rules on Iterated Matching Pennies.

	Naive	LOLA	M-FOS	MeVa
Naive	0.01	**-0.71**	-1.03±0.00	-0.93±0.03
LOLA	0.70	-5.32	**0.79**±0.12	**0.86**±0.03
M-FOS	**0.97**±0.00	-1.16±0.12	-0.01±0.13	
MeVa	0.92±0.03	-1.11±0.03		-0.05±0.01

Table 6.6. Head-to-head comparison of learning rules on the Chicken Game.

6.F. ZD-extortion on the IPD

Figure 6.4 shows how MeVa shapes a naive opponent. MeVa immediately takes on a ZD-extortion policy to induce cooperation in the naive agent, indicated by a > 1 slope on the blue front. Then, it appears to dynamically exploit the naive learner to reach and maintain a position of maximum extortion (nearly vertical front).

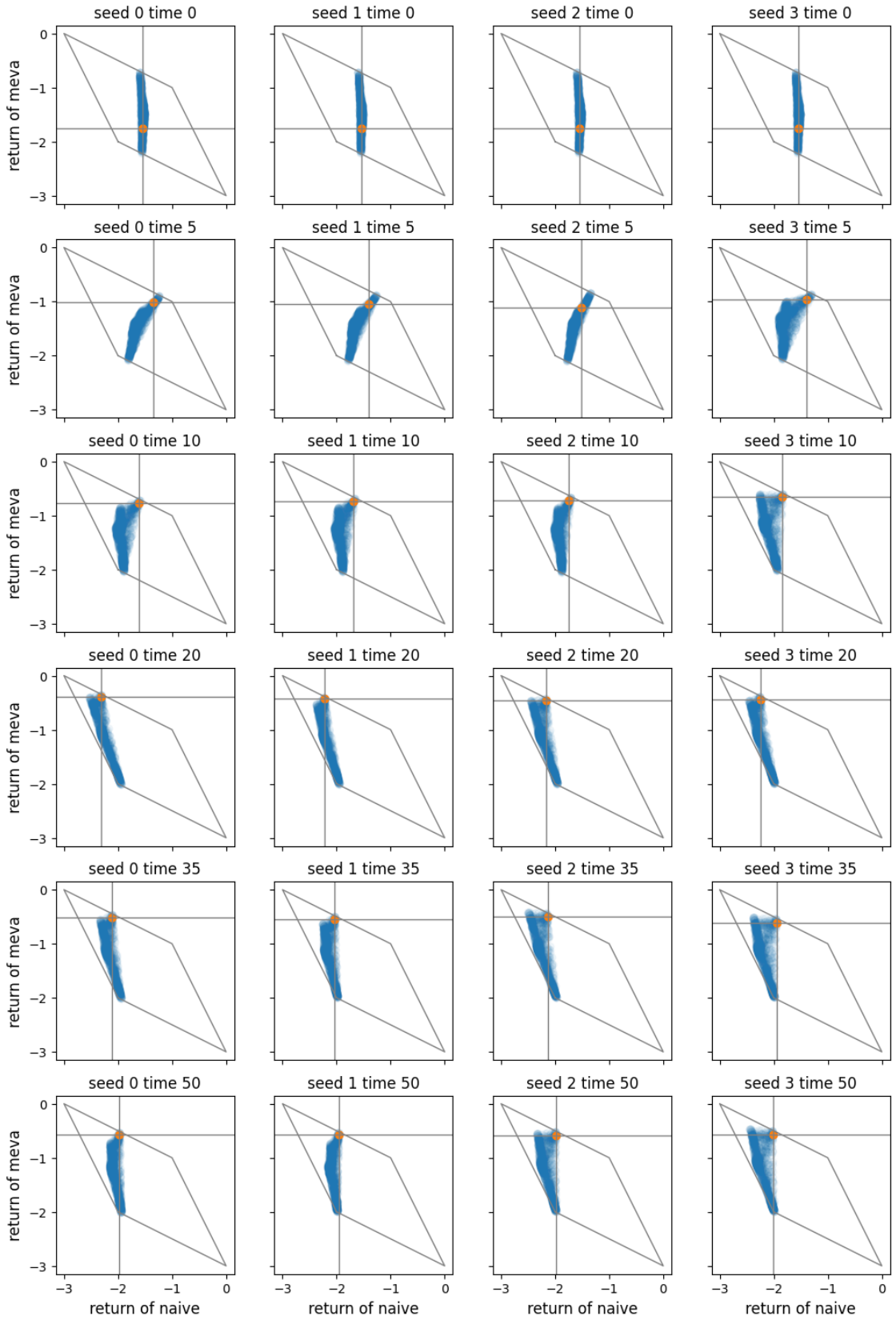


Fig. 6.4. Meta-value agents extorting naive agents on the Iterated Prisoner's Dilemma. We train $\hat{U}$ from four different initializations (columns). Then, we initialize a pair of policies and show their behavior as they learn (rows). Each panel shows the polytope of possible game returns (gray outline), the subset of possible game returns given the current MeVa policy x_1 (blue scatterplot of return pairs obtained by pitting random policies against x_1), and the actual return pair given x_1 and x_2 (orange, with lines to help tell the angle of the rightmost front of the blue region).

Conclusion

We made several contributions to obtain effective coordination among systems that have been disconnected. *Counterpoint by Convolution* (Chapter 3) considered coordination of information states of variables, transmuting independent distributions into joint ones. *On the Variance of UORO* (Chapter 4) discussed coordinated communication between graph nodes in, using broadcasts to send targeted messages. Finally, *Best Response Shaping* (Chapter 5) and *Meta-Value Learning* (Chapter 6) reconciled the dynamics of decentralized learning processes, allowing self-interested learners to reach and maintain cooperation. We close with a brief afterword on each of these projects.

In generative modeling of music (Chapter 3), we went beyond the usual unnatural approach of producing a composition in a single pass from front to back. We instead designed an algorithm, COCONET, that can generate in any order. Our method consists of a model that can fill in arbitrary musical blanks, and an iterative refinement that reconciles the model's conditionally independent predictions. Besides composing music from scratch, COCONET can be used to complete arbitrary musical fragments, which allows considerable control over the output. The versatility of the algorithm led Louie et al. (2020) to build composition software around it.

On March 21, 2019, COCONET was featured on Google's front page as the engine behind the Bach Doodle (Team, 2019; Huang et al., 2019a). The doodle would prompt users to compose a soprano melody, and our model would provide a harmonization in the alto, tenor and bass voices. Over the course of a day, we harmonized over 50 million melodies from users around the world (Huang et al., 2019b; Dinculescu, 2019).

COCONET produces a coordinated multivariate output by iteratively refining independent predictions. We were surprised at first to find that this works *better* than making properly conditioned autoregressive predictions. This finding has however been more than confirmed by the present success of diffusion models (see Croitoru et al. (2023) for a review), which essentially take a similar corrupt-and-reconstruct approach, albeit with a different mathematical interpretation. Diffusion models predict the corruption, rather than the reconstruction, and apply it in Langevin dynamics as the gradient of the potential.

Dieleman (2022) links diffusion models to denoising autoencoders and generative stochastic networks, which are in turn related to the orderless NADE model on which COCONET is based Yao et al. (2014).

Our work on credit assignment for RNNs (Chapter 4) has provided a deep theoretical understanding of the UORO algorithm, its sources of variance, and the ways in which they might be reduced. We proposed a practical variation, PreUORO, which destroys less information and as such has drastically lower variance, but increased computational expense. These findings are confirmed empirically. Finally, we made a theoretical connection between REINFORCE and UORO: as the noise vanishes, the two estimators become equal up to a mean-zero but high-variance term. The connection opens the door for cross-pollination of ideas.

It seems unlikely that the forward mode (RTRL-likes, including UORO) will ever displace the reverse mode (BPTT-likes) for practical training of RNNs. The gradient estimates are simply too noisy. However, recent work on Evolution Strategies (ES; Eigen, 1973) has found applications where exact gradient computation is not worth it in the first place, such as distributed reinforcement learning (Salimans et al., 2017) and meta-learning (Li et al., 2023). ES can be interpreted as REINFORCE with Gaussian perturbations on the parameters, and is hence related to UORO with projection in parameter space.

It is worth noting that RNNs have now fallen out of favor in part due to their sequential computational structure. Transformers (Vaswani et al., 2017) have an essentially feed-forward structure that allows parallel training. Additionally, they have practically random access to historical sequence elements, which enables much better long-term gradient flow. The downside is that Transformers have a finite horizon, whereas RNNs have infinite horizon. However this is only a theoretical point: the *effective* horizon of Transformers is much longer than that of RNNs due to difficulties in optimizing RNNs. As such, improving the speed or quality of gradient estimation may be less important than improving the optimization dynamics.

In multi-agent learning, we have proposed Best Response Shaping (Chapter 5), an agent that finds a good policy by training against the best response. The crucial ingredient is differentiation through the best response as a function of the agent's policy. This leads the agent to take on policies like tit-for-tat, which use *threats* to force a rational agent into cooperative behavior. While previous works perform an explicit optimization process to arrive at the best response (Zhang et al., 2020; Balaguer et al., 2022), we amortize this by training a conditional policy – a differentiable model that observes the policy and implements the best response. Conditioning on policies is complex when policies may be neural networks, but we demonstrated an approach that appears to scale well. We

found that Best Response Shaping beats POLA (the strongest baseline) on the Coin Game (a benchmark that requires scaling) along most dimensions of comparison.

We view our other work in this area, Meta-Value Learning (Chapter 6), as an extension of LOLA's idea of learning with learning awareness. Meta-Value Learning revolves around a function called the meta-value, which judges a meta-state (e.g. a policy pair) by its performance on the game, now and after continued optimization. We treat the meta-value function as a drop-in replacement for the original game, one on which naive learning is coordinated. Our Q-learning approach to approximating the meta-value avoids explicitly handling the continuous action space, and does not require any REINFORCE estimators. Meta-Value Learning competes with M-FOS (Lu et al., 2022) on repeated matrix games, matching its ability to extort naive learners and LOLA.

Meta-Value Learning was developed in the context of multi-agent learning and learning with learning awareness. However, it is a general meta-learning approach that uses the generalization power of neural networks to understand the optimization landscape and find better directions than the gradient. Scalability aside, meta-value learning should generally be helpful also in settings with a single objective, such as single-player games and supervised learning. In fact the thought of applying this to RNNs, which still suffer from pathological curvature and bad local optima, never left my mind. In the immediate future I aim to evaluate this possibility, first on optimization toy tasks, then on RNN toy tasks, which I believe are already within reach.