Master's thesis

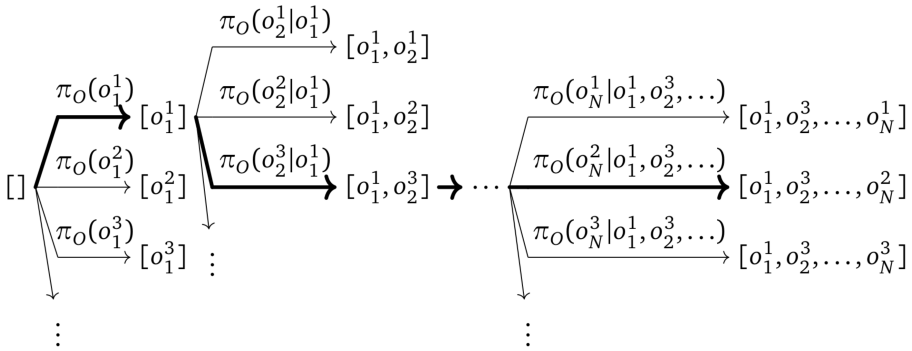
NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Thomas Løkkeborg

# Deep Reinforcement Learning for International Diplomacy

## Learning to Play Map Variants

Master's thesis in Informatics
Supervisor: Professor Keith L. Downing
June 2023

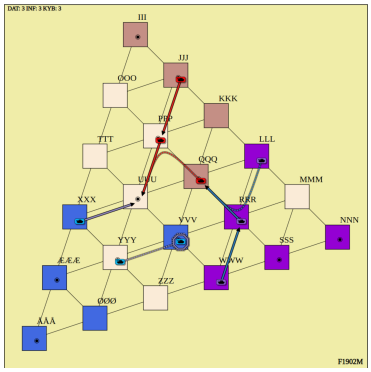



NTNU
Norwegian University of
Science and Technology

Thomas Løkkeborg

# Deep Reinforcement Learning for International Diplomacy

## Learning to Play Map Variants




Master's thesis in Informatics
Supervisor: Professor Keith L. Downing
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

# Abstract

The board game Diplomacy has received much attention in recent years as a benchmark problem for the field of artificial intelligence. Diplomacy features a massive combinatorial action space, a mix of cooperation and competition, negotiation in natural language, a deterministic ruleset, and simultaneous action selection. These traits pose a novel challenge to artificial intelligence research, and some are shared with real-life issues like negotiation, tactics, and coordination.

Recent research in artificial intelligence for Diplomacy has utilized deep reinforcement learning with (generalized) policy iteration, similar to AlphaGo Zero of Silver *et al.* [46]. State-of-the-art techniques have achieved success in the original formulation of the classic game. The success raises interest as to whether the techniques generalize to other problems.

The long lifespan and popularity of the game has spawned a culture of creating variants of the classic game. Game variants modify the ruleset, map topology, and player count to create new challenges for the player. As a step towards general applicability of state-of-the-art techniques, this thesis investigates their application in variants of the classic game.

Inspired by state-of-the-art, an agent is implemented that at each turn performs game-theoretic search over a subset of the joint action space, with payoff given by next-state end-game score prediction from a neural network. Action subsets are generated by a neural network that decomposes the action space as a sequential selection of sub-actions. Over time, the accuracy of end-game score prediction is improved via bootstrapped score estimates, and the quality of the generated action subset is improved by making actions valued by search more likely for inclusion in the future. The neural networks train from scratch with no human data, and an action exploration procedure helps discover reasonable actions.

Agents are trained through self-play on three non-communicative (No-Press) Diplomacy map variants, and evaluated through skill in tournaments with baseline opponent agents. The work shows that state-of-the-art deep reinforcement learning techniques that have seen success on the classic Diplomacy map can be applied successfully in alternative map topologies, which hints at the generality of the techniques and acts as a step toward their application in real-life issues.

# Sammendrag

Brettspillet Diplomacy har fått mye oppmerksomhet de siste årene som en problemstilling for kunstig intelligens. Diplomacy har et enormt kombinatorisk aksjonsrom, en miks av samarbeid og konkurranse, forhandlinger i naturlig språk, deterministiske regler, og simultane aksjonsvalg. Disse egenskapene ugjør en interessant utfordring for forskning innen kunstig intelligens. Noen av egenskapene er også delt med utfordringer i det virkelige liv, som forhandling, taktikk, og samarbeid.

Nylig forskning innen kunstig intelligens for Diplomacy har benyttet dyp forsterkningslæring med "(generalized) policy iteration", ikke ulikt AlphaGo Zero fra Silver *et al.* [46]. "State-of-the-art" teknikker har oppnådd suksess i den originale formuleringen av det klassiske spillet. Suksessen vekker en interesse rundt hvorvidt teknikkene er nyttige i andre problemstillinger.

Populariten og det lange leveløpet til spillet har gitt oppspring til varianter av det klassiske spillet. Spillvariantene endrer regler, kart-topologi, og antall spillere for å skape nye utfordringer for spilleren. Som et steg mot generell anvendelighet av "state-of-the-art" teknikker, utforsker denne masteroppgaven deres anvendelse på varianter av det klassiske spillet.

Inspirert av "state-of-the-art" skapes en agent som i hver runde gjør et spillteoretisk søk over et subset av felles aksjonsrom med payoff definert via poengestimat av spilltilstander i følgende runde gitt av et nevralnett. Subsettet av aksjoner genereres av et nevralnett som dekomponerer aksjonsrommet som et sekvensielt valg av sub-aksjoner. Over tid forbedres kvaliteten i poeng-estimatene via "bootstrapping", og kvaliteten på subsettet av aksjoner ved å gjøre aksjoner regnet som gode av spillteoretisk søk mer sannsynlige for inkludering i subsettet i fremtiden. Nevralnettene trenes fra grunnen av uten hjelp av menneskelig data, og en "action exploration"-prosedyre hjelper med å oppdage fornuftige aksjoner.

Agenter trenes via self-spill på tre ikke-kommunikative (No-Press) Diplomacy kartvarianter, og blir evaluert via ferdighet i turneringer med referanseagenter. Arbeidet viser at "state-of-the-art" teknikker innen dyp forsterkningslæring som har oppnådd suksess på det klassiske Diplomacy-kartet kan anvendes med gode resultater på alternative kart-topologier. Dette resultatet hinter til den generelle anvendeligheten av teknikkene og utgjør et steg mot deres anvendelse i utfordringer fra det virkelige liv.

# Preface

This master's thesis was written in the Spring of 2023 for the completion of the degree Master of Science in Informatics with a specialization in artificial intelligence at the Norwegian University of Science and Technology. The thesis is a continuation of an unpublished preparatory project conducted in the Fall of 2022 [31], which contains draft versions of chapters 1, 2, 3, and 4.

I would like to thank my supervisor Professor Keith. L. Downing for guiding me through this daunting task, and for giving me an opportunity to combine my master's degree with my passion for Diplomacy. I would also like to thank my family and friends for enduring more babbling about board games and programming than usual this past year.

Thomas Løkkeborg
Trondheim, 19.06.2023

# Contents

# Figures

# Tables

# Math Notation

$\mathbf{W} \in \mathbf{R}^{R \times C}$ Matrix of real-valued numbers with $R$ rows and $C$ columns.

$\mathbf{W} \in \{0,1\}^{R \times C}$ Matrix of 0 or 1 with $R$ rows and $C$ columns.

$O$ Set.

$O_i$ $i$-th set.

$|O|$ Number of elements in set.

$o \in O$ $o$ takes a value from set $O$.

$\vec{a}$ Vector.

$O = [O_0, O_1, \ldots]$ Vector of sets.

$A = O_0 \times O_1 \times \ldots]$ Set of vectors specified as all possible assignments of a value to the $i$-th element $o_i$ from set $O_i$. Result of cartesian product of the sets $O_i$.

$\vec{a} = [o_1, o_2, \ldots]$ Vector, whose $i$-th element $o_i$ is an element from set $O_i$. $\vec{a} \in A$.

$\pi(O_i|o_1, o_2, \ldots, o_{i-1})$ Conditional probability distribution over values in set $O_i$ conditioned on values $o_1, o_2, \ldots, o_{i-1}$.

# Chapter 1

# Introduction

This chapter introduces the thesis by presenting motivation and background, stating research questions, and outlining the report structure.

## 1.1 Motivation and Background

The field of artificial intelligence (AI) studies the creation of computationally intelligent agents able to act rationally in complex environments. Benchmark problems are required to empirically measure progress on this task. Games have a long standing as a benchmark of human intelligence, and serve as an appropriate benchmark for AI.

Each game presents a set of challenges: chess and go require long-term strategies, backgammon requires reasoning about a stochastic environment, and poker requires reasoning with imperfect information. Recent years have seen major milestones in AI for games, with computational agents playing at the expert level in chess [7], go [45], backgammon [48], Atari video games [34], StarCraft II [50], poker [35, 5], and Stratego [39].

A game that has received much attention in recent years as a benchmark for AI is the Diplomacy board game [37, 1, 3, 17, 23, 4, 11, 27]. Diplomacy features a combinatorial action space much larger than even that of go, a mix of cooperation and competition, negotiation in natural language, a deterministic ruleset, and simultaneous action selection. In addition to posing a novel challenge to AI researchers, some of these traits are shared with real-life issues like negotiation, tactics, and coordination. Techniques successful on the Diplomacy board game could therefore generalize onto issues with a real impact on the world.

Success in Diplomacy has been achieved using reinforcement learning (RL) with (generalized) policy iteration, similar to AlphaGo Zero of Silver *et al.* [46]. Diplomacy RL systems consist of several novel components: policy networks that model the action space as sequences of sub-actions, action exploration that enables learning from scratch with no human data, game-theoretic search over subsets of joint action space, and regularization of search with a human prior to achieve human-like play.

Due to the complexity of Diplomacy, game variants have been used to study aspects of the game in isolation. Bakhtin *et al.* [4] achieved an expert-level agent that plays the non-communicative No-Press game variant, and FAIR *et al.* [11] incorporated a language model to create the first agent to competently play the original variant of the game.

## 1.2 Goals and Research Questions

The thesis centers around the following goal and research questions.

**Goal** *The goal of the thesis is to explore the application of state-of-the-art reinforcement learning techniques in the game of Diplomacy.*

Diplomacy AI research has been successful within the context of the original game, with minor rule deviations[1]. However, the Diplomacy board game is only a benchmark problem for AI research, and the ultimate goal is to apply the developed techniques more generally. As a step towards general applicability of recent techniques, this thesis will investigate their application in variants of the classic game.

**Research Question 1** *Can state-of-the-art techniques successfully learn to play Diplomacy map variants?*

Major publications have tailored their techniques to fit the original map of the game. The first research question seeks to answer whether the techniques can be used to create learning agents more generally on map variants of the game.

**Research Question 2** *Do state-of-the-art generalized policy iteration techniques compare favorably to policy gradient techniques for Diplomacy on the Pure game variant?*

Early research on reinforcement learning for Diplomacy utilized the policy gradient paradigm, where agents are trained by acting in the environment and adjusting probabilities of taken actions directly based on received reward. State-of-the-art techniques instead uses the (generalized) policy iteration paradigm, where agents act by applying a search procedure, and train by approximating the result of search. Recent work utilizing policy iteration has been shown to outperform policy gradient techniques on the classic Diplomacy map [1]. The second research question seeks to reproduce this result on a variant of the game. The openly available Actor-Critic (A2C) policy gradient agent of Hatlø [20] will be used, which is trained on the Pure game variant.

[1] No-Press, Restricted-Press, FvA. See section 2.1.2.

## 1.3 Thesis structure

The thesis consists of the following chapters:

**1 Introduction** This chapter introduces the thesis by presenting motivation and background, stating research questions, and outlining the structure of the document.

**2 Background** This chapter introduces background theory for the domain, related work, and the methodology.

**3 Related Work** This chapter contextualizes the thesis by presenting related work.

**4 Methodology** This chapter details the methodology followed to achieve the goal and ultimately answer research questions.

**5 Results and Analysis** This chapter presents and analyses results gathered in carrying out the methodology of the previous chapter.

**6 Conclusion** This chapter concludes the thesis by addressing research questions in light of the analyzed results, listing thesis contributions, and outlining suggestions for future work. The chapter opens with a thesis review, and closes with an epilogue.

# Chapter 2

# Background

This chapter introduces background theory for the domain, related work, and the methodology. The chapter covers four main topics: The Diplomacy board game, neural networks, reinforcement learning, and game theory.

## 2.1 Diplomacy

Diplomacy [6, 10] is a strategic board game for 7 players first released in 1959 in which each player controls a national power in early 20th-century Europe. The game is renowned in the board game community for its elegant ruleset, the possibility of negotiating alliances with a real impact on the game, and for ruining friendships when those alliances inevitably are broken. From the perspective of using the game as a benchmark for AI, the key features are a massive combinatorial action space, a mix of cooperation and competition, negotiation in natural language, a deterministic ruleset, and simultaneous action selection.

The classic map is divided into regions[1], some of which contain "supply centers". The goal of the game is for one player to control over half the supply centers on the board. Regions of the board are either "land areas", "coastal areas" or "sea areas"[2]. Coastal areas are regions adjacent to sea areas that are not themselves sea areas. There are two types of units: armies move on land areas and coastal areas, and fleets move on sea areas and coastal areas.

The game is played in game-years, starting in 1901. Each year is divided into three seasons[3]: Spring, Fall and Winter. Each season consists of one or more phases. The phases are, along with a brief description:

- **Movement phase**: Movement orders are issued to every unit on the board.
- **Retreat phase**: Retreat orders are issued to "dislodged" units.

[1] Regions are more commonly called "provinces". "Regions" is chosen for this thesis to simplify notation; A variable $r$ unambiguously refers to a region, while a variable $p$ could be interpreted as a "power" or "player".

[2] As will be discussed in section 4.1, only land areas and army units are considered in this thesis.

[3] The official rules divide the game into two seasons. This thesis uses the convention of the MILA engine of Paquette *et al.* [37], as discussed in 4.2.

- **Adjustment phase**: Players "build" or "disband" units.

1901 consists of the following turns in the order shown. This pattern repeats for every game-year.

1. Spring 1901 Movement
2. Spring 1901 Retreat
3. Fall 1901 Movement
4. Fall 1901 Retreat
5. Winter 1901 Adjustment

At each turn of the game, all players secretly issue orders to their units on the board. Orders are written according to a syntax, loosely: "<unit type> <unit location> <unit command>". Orders are handed to an adjudicator who resolves them simultaneously according to the rules. Abbreviations are used in place of place names. Norway is denoted "NWY", for example.

### 2.1.1 Order Example

For the purpose of this thesis, a few examples of orders in the movement phase, their meaning, and how they interact, suffices to give insight into how the game is played. The examples also demonstrate the complexity of the game. The orders are discussed in the order an adjudicator would consider them while resolving their effects.

Consider the example orders in figure 2.1. Two units are trying to move into BUL (1, 2). Provided no support, no unit overpowers another, and there would be no movement. BUL would be left empty, which is called a "bounce" or a "standoff".

The move from BLA to BUL is, however, supported by the fleet in AEG (3). This results in BLA moving into BUL with double the strength of CON, leaving CON where it is and moving BLA to BUL. There is an army in BUL ordered to hold (4) - meaning to stay put. This unit is also overpowered by the strength of the supported move, and is "dislodged". Dislodged units are forced to retreat to an adjacent region in a following "retreat phase".

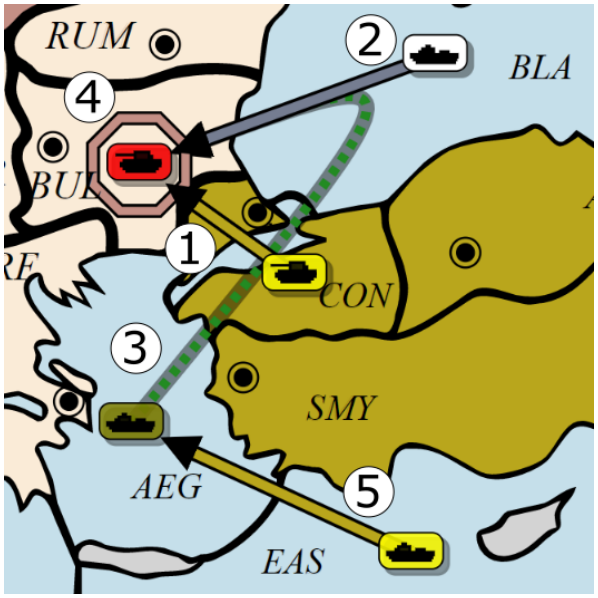
Finally, there is a fleet in EAS moving to AEG (5). This move is unsupported, and will not dislodge the fleet in AEG. It will, however, disturb AEG's order to support BLA into BUL (order 5 "cuts the support" of order 3). Left unsupported, BLA moves into BUL with the same strength as the holding army (and the army moving from CON), resulting in a bounce. BUL is therefore not disbanded, and the orders result in the units existing in the same configuration next turn.

Notice that this situation would have had the same result regardless of which power owned the units[4]. This demonstrates the fact that players are free to support each other's orders, and to disrupt their own orders - "self-bounces" are common

[4]There are two situations where power ownership matters: 1) A power cannot dislodge or support the dislodgement of one of its own units, and 2) A power cannot cut the support of one of its own units [10]. If green owned EAS in figure 2.1b, A BLA - BUL would have been succesfully supported by AEG due to 2).

| Nr. | Order | Explanation |
|---|---|---|
| 1 | A CON - BUL | The army in Constantinople moves to Bulgaria. |
| 2 | F BLA - BUL | The fleet in the Black Sea moves to Bulgaria. |
| 3 | F AEG S F BLA - BUL | The fleet in the Aegean Sea supports the fleet in the Black Sea moving to Bulgaria. |
| 4 | A BUL H | The army in Bulgaria holds. |
| 5 | F EAS - AEG | The fleet in the Eastern Mediterranean moves to the Aegean Sea. |

**(a)** Five example Diplomacy orders.




**(b)** Visualization of orders created with the MILA Diplomacy Engine (Section 4.2), annotated with order numbers.

**Figure 2.1:** Five example Diplomacy orders. The orders are presented and explained in table 2.1a and visualized in figure 2.1b.

to ensure a region is left unoccupied.

The situation above also shows why communication is an important aspect of the game. In the following turn, Russia might try to sway Austria to support BLA moving to CON, with the promise to leave BUL alone. This move will be successful provided Russia can trust Italy not to use their unit in AEG to support CON or cut support from BUL. Russia might propose that Italy move their fleet into SMY instead, anticipating that the Turkish fleet in EAS will repeat the same order. This would net Russia and Italy a supply center each at the expense of Turkey, and would avoid Austria losing one in a vulnerable position.

### 2.1.2 Diplomacy Game Variants

The long lifespan and popularity of the game has spawned many game variants, some officially licensed and others fan-made. The most prominent game variant is "No-Press" Diplomacy[5], which forbids conversation between players, limiting communication to what can be expressed through actions on the board[6]. The original game of Diplomacy is often labelled the "Full-Press" variant to distinguish it from "No-Press". Some Diplomacy AI research also makes use of a "Restricted-Press" game variant, where communication is limited to a computer-friendly protocol [27, 25]. Other relevant game variants are "France vs. Austria" (FvA), a 2-player variant using the classic map that includes only France and Austria, and "Pure", a 7-player game variant where the map consists of 7 fully connected provinces, one per power. The Pure game variant is further discussed in section 4.8.1.

## 2.2 Neural Networks

A **neural network**[7] is a parameterized function consisting of a composition of mathematical operations transforming an input space into an output space. A neural network can be utilized as a function approximator by adjusting its parameters until it performs the desired transformation. This adjustment is performed with the **back-propagation** algorithm [40], which utilizes parameter gradients calculated with respect to some **loss function** expressing a measure of error given network **parameters**, also called network **weights**.

Back-propagation can be utilized on any function that is differentiable with respect to its parameters, and thus also for any composition of such functions. The composition of functions in a neural network is called the **neural network architecture**. The advent of automatic differentiation tools like PyTorch [38] has empowered AI researchers to experiment with novel architectures in new domains.

[5] "No-Press" Diplomacy is also called "gunboat" Diplomacy by the Diplomacy player community.

[6] An article from 1995 published in a fan-made Diplomacy magazine discusses how communication can be achieved through actions in the "No-Press" game variant. `http://uk.diplom.org/pouch/Zine/W1995A/Szykman/NoPress.html`, accessed 08.09.2022.

[7] Also called artificial neural network (ANN) to distinguish from biological neural networks.

Therefore, a large space of possibilities exists when designing a neural network, and architectures can be tailor-made for specific problem domains.

The following sections present key architectures and concepts relevant to this thesis. Refer to Goodfellow *et al.* [16] for a detailed treatment of neural networks.

### 2.2.1 Computational Graphs

Neural networks are typically introduced with an analogy to biological neurons in the human brain, accompanied by visualizations showing how neurons of the network interconnect to produce output. For the novel network architectures necessary when modeling the Diplomacy board game, a more precise visual language is useful. **Computational graphs** [16, Chapter 6.5.1] are used, as they map more directly to the underlying mathematical expressions.

In this thesis, the nodes of computational graphs are variables and operations on variables. Variables are denoted through variable names, and operations are enclosed in rectangles. Edges denote the flow of information in the graph. An edge from a variable to an operation means that the variable is utilized by the operation, and an edge from an operation to a variable means the variable is the result of an operation. Edges can also exist between operations, denoting that the result of the operation is directly consumed by another, without the need for an intermediate variable. As an example, consider the expression $f(xv+yw+b) = o$ visualized with a computational graph in figure 2.2a.

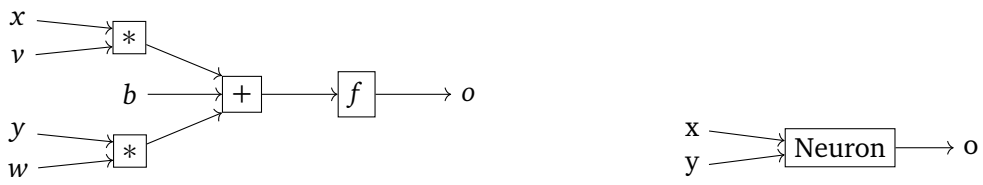
Computational graphs can be used to visualize neural networks at a desired level of abstraction. For example, figure 2.2b shows how the operation of a simple neuron in figure 2.2a can be encapsulated into an operation named "Neuron". Compositions of neurons could then be shown by reusing this higher-level abstraction.

The variables of a computational graph can be any mathematical object. Figure 2.2a takes $x$ and $y$ as scalars, but the graph could be rewritten to perform the same operation on vectors of arbitrary length. An example of this is given when introducing the feed-forward neural network in the following section. Variables could also be matrices, tensors, or sets.

Computational graphs inspire the notation for several figures in the methodology chapter of this thesis. The notation is not strictly followed, and neural network architecture is blended with other parts of the system for convenience. The meaning of symbols in figures should be clear from context, but the general idea of computational graphs lays the foundation.

### 2.2.2 Feed-Forward Neural Networks

The simplest neural network architecture is the **feed-forward neural network**. It takes some input and transforms it through a series of operations to produce an output. The name feed-forward stems from the fact that no cycles exist in the composition of functions, as opposed to recurrent neural networks discussed in the following section.




**(a)** $f(xv + yw + b) = o$ **(b)**

**Figure 2.2:** Simple computational graph. Figure 2.2a shows the computational graph for the expression $f(xv + yw + b) = o$. Figure 2.2b encapsulates this expression in a named operation, parameterized by $v$, $w$ and $b$. The operation corresponds to an individual neuron in a typical neural network with two inputs, assuming $f$ is a non-linear transformation. Neural networks can be visualized with higher-level graphs that compose encapsulated operations.

Typically, feed-forward neural networks consist of a series of layers, each layer consisting of a number of neurons. Each neuron performs a parameterized non-linear transformation of the layer's inputs. A simple neuron taking two inputs is shown in figure 2.2b. Instead of operating on individual neurons, it is common to view the layer as taking a vector of inputs and producing a vector of outputs as the output of each individual neuron. The more neurons and layers are added, the greater the network's capacity to approximate a desired function becomes.

As an example, consider a feed-forward neural network of one layer taking as input the vector $\vec{x} = [x_1, x_2, \ldots, x_N]$ and producing as output the vector $\hat{\vec{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M]$. The layer first performs a linear transformation of $\vec{x}$ by matrix multiplication with a weight matrix $\mathbf{W}^{M \times N}$, which parameterizes the layer. The layer then applies a non-linear transformation through the sigmoid activation function, denoted $\sigma$. Usually, the result of matrix multiplication would be shifted by "bias" weights. Bias is omitted in this example for notational convenience[8]. The network is called fully connected, since all inputs to the layer influence all output neurons. The activation function and neural network is denoted as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$\begin{aligned} f(\vec{x}|\mathbf{W}) &= \sigma(\mathbf{W}\vec{x}) = \hat{\vec{y}} \\ &= \sigma\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,N} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \ldots & w_{M,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}\right) = \hat{\vec{y}} \end{aligned} \tag{2.2}$$

This neural network can be visualized with a computational graph as shown in figure 2.3a, where "matmul" is the matrix multiplication operation. Notice that the variables are vectors and matrices, showing a higher level of abstraction than

[8]Bias can be included as an additional column in the weight matrix if a dummy input of 1 is assumed in the input vector.


**(a)** $\sigma(\mathbf{W}\vec{x}) = \hat{\vec{y}}$ **(b)**

**Figure 2.3:** Computational graph for a one-layer feed-forward neural network, without bias for notational convenience. Figure 2.3a visualizes the operation of equation 2.2. "matmul" is the matrix multiplication operation. Figure 2.3b encapsulates the operation, allowing for composition with other operations to create more sophisticated neural network architectures.

earlier example computational graphs. The network can also be encapsulated in a single operation as shown in 2.3b, allowing for composition with other operations to create more sophisticated neural network architectures.

Given samples of desired output $\vec{y}$ given input $\vec{x}$, and a loss function $L(\mathbf{W}|\vec{x}, \vec{y})$ measuring the error of $\mathbf{W}$, the weight matrix can be adjusted to bring network output closer to the desired output. The following equation shows a simple loss function defined as the sum of squared errors between each $\vec{x}_i, \vec{y}_i$ (input, output) pair[9].

$$L(\mathbf{W}) = \sum_i \left(f(\vec{x}_i|\mathbf{W}) - \vec{y}_i\right)^2 \tag{2.3}$$

The network is trained with back-propagation by iteratively adjusting the weight matrix $\mathbf{W}$ a small step in the direction that minimizes the loss function $\frac{\partial L(\mathbf{W})}{\partial \mathbf{W}}$, starting with some initial value for $\mathbf{W}$. This process is called **supervised learning** because the samples act as a guide supervising the training of the network toward the target function the network trying to approximate.

Simple fully connected feed-forward neural networks like that of equation 2.2 do not assume, nor take advantage of, structure in data. More complex network architectures have been proposed for this purpose. The following two sections present architectures that are particularly relevant to this thesis.

### 2.2.3 Graph Neural Networks (GNN) and Graph Convolutional Networks (GCN)

**Graph neural networks** are neural networks that leverage the structure and properties of graphs [41]. GNN take as input edges in the form of an adjacency matrix, and a number of features per node in the graph. They produce a (possibly different) number of output features per node. A general formula for GNN [18, Equation 5.4] is given in equation 2.4[10].

$$\vec{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\vec{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\vec{h}_v^{(k)}, \forall v \in \mathbf{N}(u)\})\right) \tag{2.4}$$

[9] $\vec{x}_i$ here refers to the $i$-th vector $\vec{x}$, not the $i$-th element of $\vec{x}$. Likewise for $\vec{y}_i$.

[10] The convention of Hamilton [18] to enclose superscripts in parenthesis is used when presenting GNN.

Each node $u$ has at every step $k$ of the GNN a "hidden message" $\vec{h}_u^{(k)}$, a vector. Denoting the neighborhood of $u$ as $\mathbf{N}(u)$, this message is updated at step $k+1$ by aggregating the messages of adjacent nodes $v \in \mathbf{N}(u)$ with some aggregation function AGGREGATE, and performing some UPDATE function given the previous message and the aggregated messages. $\vec{h}_u^{(0)}$ is set to the initial features of each node, and by performing iterations of equation 2.4 messages propagate through the graph following adjacencies.

Implementations of GNN have to define UPDATE and AGGREGATE. One relevant implementation of GNN is **graph convolutional networks** (GCN) [18, Equation 5.16][11], shown in equation 2.5 where $\sigma$ is the sigmoid activation function of equation 2.1.

$$\vec{h}_u^{(k+1)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v \in \mathbf{N}(u) \cup \{u\}} \frac{\vec{h}_v^{(k)}}{\sqrt{|\mathbf{N}(u)||\mathbf{N}(v)|}}\right) \tag{2.5}$$

$\mathbf{W}^{(k)}$ is a $n_{\mathbf{h}^{(k+1)}} \times n_{\mathbf{h}^{(k)}}$ weight matrix where $n_{\mathbf{h}^{(k)}}$ is the number of features of the hidden message at step $k$. The weight matrix is shared for all nodes, allowing the network to generalize over nodes of the graph.

A GCN aggregates messages of adjacent nodes with symmetric normalization [18, Chapter 5.2.1] - messages of neighboring nodes are normalized through division with the square root of the product of degrees for the node $u$ and neighbor $v$. This normalization helps keep node messages within the same order of magnitude despite variable node degrees, as opposed to simple summing. *Symmetric* normalization also has the advantage of assigning less importance to messages from very high-degree neighbors.

### 2.2.4 Recurrent Neural Networks (RNN) and the Long Short-Term Memory (LSTM) Model

**Recurrent neural networks** are neural networks that utilize parameter sharing to process **sequential data** and allow for variable length input [16]. Many classes of RNN exist, and the kind utilized in this thesis take as input a variable-length vector of values $\vec{x} = [\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N]$ and produces as output a vector $\vec{y} = [\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_N]$ of the same length.

Figure 2.4 shows an example of an RNN adapted from Goodfellow *et al.* [16, Figure 10.3]. The figure is similar to the one-layer feed-forward network of figure 2.3a, except for the introduction of a "hidden state" $\vec{h}_i$ that influences the next sequence step, and the addition of two more weight matrices with associated operations. Starting with an initial hidden state at $i = 0$, the following is calculated per sequence step $i$:

[11] The superscripts in Hamilton [18, Equation 5.16] are believed to be erroneous, and have been fixed for inclusion in this thesis.

$$\begin{aligned}\vec{h}_i &= \sigma(\mathbf{W}\vec{h}_{i-1} + \mathbf{U}\vec{x}_i) \\ \vec{y}_i &= \mathbf{V}\vec{h}_i\end{aligned} \tag{2.6}$$

When sequentially processing $\vec{x}_i$, the hidden state $\vec{h}_{i-1}$ is used by the network as a representation of $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{i-1}$, providing context for the output of $\vec{y}_i$. This allows the network to generalize over a sequence of data, similar to how graph convolutional networks generalize over a graph. It also allows the network to process a variable length sequence.

RNNs are trained by producing the full sequence of outputs, measuring loss through comparison with a ground-truth sequence, and then "unrolling" recurrent connections as shown in figure 2.4b. With the recurrent connections unrolled, back-propagation can be applied to propagate gradients back through the network, and the weights can be adjusted using the gradients to minimize loss. Back-propagation used in this way is called **back-propagation through time**.

RNNs are notoriously hard to train, as the propagation of gradients through time can make them explosively large or vanishingly small. **Gated RNN** employ paths through time with gradients that don't explode or vanish, conditioned on the current hidden state. These paths are gated with a parameterized function, allowing the network to learn to retain and forget information while processing a sequence.

The original gated RNN is the **Long Short-Term Memory** (LSTM) model. The LSTM has complex internals, but is commonly available as a black-box in machine learning frameworks. Figure 2.5 shows how a black-box implementation of LSTM can be used as part of a larger computational graph. For a detailed treatment of LSTM internals, refer to Goodfellow *et al.* [16, Chapter 10.10.1].

### 2.2.5 Teacher Forcing

Figure 2.4a shows an RNN with recurrent connections between hidden states, meaning the hidden state of one sequence step is influenced by that of the previous step. In principle, however, recurrent connections could be added anywhere, and some RNNs benefit from connections going from the output of the network back to the hidden state of the next step. This allows the network to produce the sequence conditioned on network output so far, not just on inputs. Figure 2.6 shows an example.

During the training of such a network, incorrectly predicting $\hat{\vec{y}}_i$ affects the prediction of $\hat{\vec{y}}_{i+1}$. This makes correctly producing a full sequence very difficult. **Teacher forcing** [16, Chapter 10.2.1] addresses this issue by supplying the ground-truth $\vec{y}_{i-1}$ when producing $\hat{\vec{y}}_i$ rather than the network output. Intuitively, this training scheme trains each sequence step as if the sequence was predicted correctly so far.

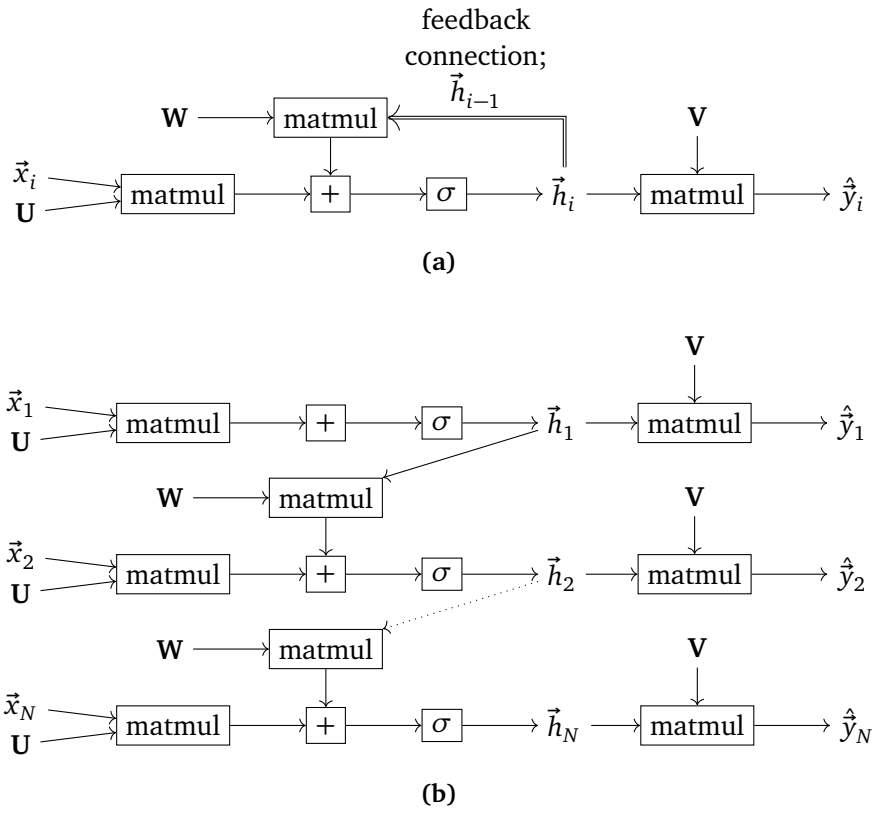



**Figure 2.4:** Computational graph for an RNN that maps a sequence of input vectors $\vec{x} = [\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N]$ to a corresponding sequence of output vectors $\vec{y} = [\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_N]$. Figure 2.4a shows the computational graph with a feedback connection to signify recurrence. Figure 2.4b shows the same graph unrolled for each sequence step. Notice that weight matrices $U$, $V$, and $W$ are shared between sequence steps. This is called parameter sharing. Biases are omitted for notational convenience. The RNN is adapted from Goodfellow *et al.* [16, Figure 10.3].

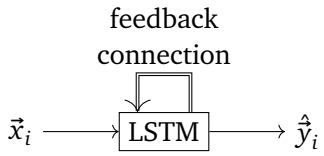



**Figure 2.5:** Usage of black-box LSTM implementation in computational graphs.

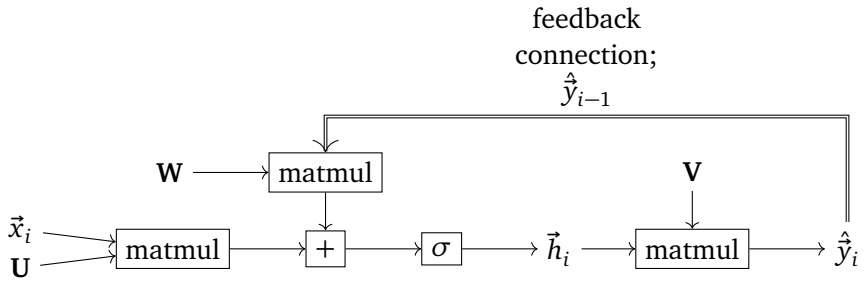



**Figure 2.6:** Modification of figure 2.4a with recurrent connection from output instead of the hidden state. This network can be trained with teacher forcing, which uses the ground-truth $\vec{y}_{i-1}$ when producing $\hat{\vec{y}}_i$ rather than the network output.

## 2.3 Autoencoders and the Encoder-Decoder Architecture

A crucial factor in the successful training of a neural network is the representation of the data being processed. **Representation learning** is an approach to neural network training with the objective of learning a useful representation of data. The **autoencoder** is the quintessential example of representation learning [16]. An autoencoder consists of an **encoder** that produces a compressed representation, or **encoding**, of its input, and a decoder whose job it is to reproduce the original input from the encoding. An autoencoder is trained by feeding it data and calculating loss based on the difference between the input and the output. A successfully trained autoencoder learns a useful representation of its input.

The **Encoder-Decoder** architecture is a generalization of the idea of producing an encoding with an encoder that is then used by a decoder for higher-level processing. The architecture is useful when a neural network has multiple outputs, or "heads", as the network can be modeled in terms of a single encoder and several decoders.

### 2.3.1 Embedding

An **embedding** transforms a categorical value into a real-valued vector representation [16, Figure 14.8]. The idea is that categorical values that are similar in meaning should be represented as points in a high-dimensional space that are close together, and conversely for values that are different in meaning. Embedding a categorical variable provides more context to downstream processing, and allows for generalization.

The embedding of a categorical variable $x$ into $M$-dimensional space is denoted $e(x) \in \mathbb{R}^M$. Embeddings are easily represented as matrices: Embeddings of $N$ categorical variables is denoted $\mathbb{R}^{N \times M}$. Each row corresponds to the embedding of a variable, and $e(x)$ is therefore just a lookup into this matrix.

Mathematically, the lookup of an embedding for a categorical value can be performed by multiplying the matrix by a simple "one-hot-encoding" (OHE) of

the value, provided each categorical value is assigned a unique index. Embeddings can be included as components of neural networks, as back-propagation can propagate back through the OHE multiplication to calculate loss with respect to the embedding matrix.

### 2.3.2 KL-Divergence as a Loss Function

**Kullback-Leibler Divergence** (KL-Divergence) is a measure of the difference between two probability distributions [16, Equation 3.50]. The KL-Divergence between two probability distributions $P$ and $Q$ is defined has:

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)] \tag{2.7}$$

KL-Divergence is non-negative, and for discrete probability distributions, $D_{\mathrm{KL}}(P||Q)$ is 0 if and only if $P$ and $Q$ are the same distribution. All probability distributions are discrete in the context of this thesis. KL-Divergence is asymmetric, $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$, and is thus not a true distance measure. Still, it serves as a useful loss function when the output and target of a neural network are probability distributions.

## 2.4 Reinforcement learning (RL)

According to Sutton and Barto [47]: "Reinforcement learning is learning what to do—how to map situations [states] to actions—so as to maximize a numerical reward signal." The two most important characteristics of RL [47] are trial-and-error search and delayed reward; The learner must discover optimal actions by interaction with the environment, and the reward for performing a given action is not only dependant on the following situation, but all subsequent situations.

The following sections provide a brief introduction RL and techniques relevant to this thesis. Refer to Sutton and Barto [47] for a detailed treatment.

### 2.4.1 Core RL concepts

**Reinforcement learning** uses the formalism of **states**, **actions** and **rewards**. The agent perceives its environment through a state $S_t$[12], chooses an action $A_t$, and receives a reward $R_{t+1}$ and a resulting new state $S_{t+1}$ from the environment. This is illustrated in figure 2.7.

A **policy function** $\pi$ maps states to actions. The learner seeks to find an optimal policy function that maps from states to actions that maximize expected future discounted total reward, also called **return** $G_t$, defined in (2.8). $\gamma$ is a discounting factor that introduces a preference for early reward over later reward.

[12]States and actions are capitalized in this introduction to RL following Sutton and Barto [47]. Throughout the rest of the thesis, lowercase variable names are used to avoid confusion with set notation.

$$S_0 \rightarrow A_0 \rightarrow R_1 \rightarrow S_1 \rightarrow A_1 \rightarrow \cdots \rightarrow R_N \rightarrow S_N$$

**Figure 2.7:** State, action, reward formalism of reinforcement learning

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.8}$$

A **value function** $v_\pi(s)$ maps states to expected return when following $\pi$ starting from state $s$:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \middle| S_t = s\right] \tag{2.9}$$

An optimal policy $\pi(s)$ is one whose value function $v_*(s)$ is maximal in all states:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S \tag{2.10}$$

We sometimes want to express the value of a state-action pair rather than the value of a state. In this case, we use $Q_\pi(s,a)$, defined as the expected return of taking action $a$ in state $s$, and following policy $\pi$ thereafter.

### 2.4.2 Learning in RL

The task of the learner is to find (an approximation of) optimal behavior in an environment. This is typically done by starting from some initial policy function, value function, or both, and iteratively improving toward optimal behavior. For environments of low complexity, tabular Dynamic Programming techniques can be used to find the functions. In practice, however, environments are typically complex and require the use of function approximators (eg. neural networks), algorithms that learn through interaction with the environment (eg. Sarsa, Q-learning), and bootstrapping the functions from some prior.

A learner interacts with the environment while learning by choosing actions from some policy $\pi$. If this policy is also the target for learning we call it **on-policy** learning. The learner can instead choose to use experience to train some separate target policy. We call this **off-policy** learning.

When an on-policy learner is simultaneously exploring the environment and optimizing its policy function, a crucial balance must be struck between **exploration** and **exploitation**: exploiting its experience prematurely will cause it to stop exploring its environment, while failing to exploit its knowledge will lead to sub-optimal behavior. An example of a purely exploitative learner is one whose policy function for all states greedily chooses the action leading to the successor state with maximal value according to its (possibly incorrect) current understanding of its environment. An example of a purely explorative learner is one whose policy

function for all states samples an action uniformly from the set of all actions. A common compromise is the $\epsilon$-greedy learner, which with probability $\epsilon$ samples a random action, and with probability $1-\epsilon$ greedily chooses the best action. In a stationary environment, $\epsilon$ can be discounted over time to gradually make the learner exploitative.

### 2.4.3 Policy Iteration (PI), Value Iteration (VI) and Generalized Policy Iteration (GPI)

The "policy improvement theorem" states that given any pair of deterministic policies $\pi$ and $\pi'$ where the following holds:

$$Q_\pi(s, \pi'(s)) \geq v_\pi(s) \text{ for all } s \in S \tag{2.11}$$

Then $\pi'$ must be as good as, or better than, $\pi$. [47] A technique that takes advantage of this fact is **policy iteration**, which alternatingly performs **policy evaluation** to find the value function of a policy, and **policy improvement** to construct a new policy by greedily choosing actions based on the value function. **value iteration** is an alternative technique that directly optimizes a value function by performing a single step of policy evaluation within each step of policy improvement. Strictly speaking, policy iteration and value iteration are both Dynamic Programming techniques, meaning we need access to a perfect model of the environment, and perform updates to all states simultaneously. However, the terms are used loosely in RL research. policy iteration is often used to refer to techniques that apply some form of policy improvement step to iteratively learn better policies, and value iteration is often used to refer to techniques that directly update a value function based next-state values with respect to a policy found by policy improvement.

**Generalized policy iteration** is a name given to the general concept of simultaneously making the value function consistent with the policy function (policy evaluation), and making the policy greedy with respect to the value function (policy improvement). PI and VI as presented above are instances of GPI. Generalized policy iteration allows for techniques that incorporate decision-time search over available actions [47, Chapter 8.8] as a powerful **policy improvement operator** [46], based on the policy and value functions. This is the foundation for several recent successes in AI for games [45, 5, 4]. The term generalized policy iteration is used in this thesis to mean any method that utilizes a policy improvement operator to improve either the policy function, the value function, or both.

### 2.4.4 Approximation and Deep Reinforcement Learning

Neural networks can be used as function approximators for policy and value functions. Reinforcement learning utilizing neural networks is called **deep reinforcement learning**, influenced by the term "deep learning" used to describe particularly deep neural networks. Networks used in this way are referred to as **policy**

**networks** and **value networks**. Using function approximators allows the agent to learn compressed representations that generalize across states and actions.

Neural networks can be iteratively trained through supervised learning to imitate the result of policy evaluation (value network) and policy improvement (policy network). A policy network can also be trained through policy gradient methods, as described in the following section.

### 2.4.5 Policy Gradient Methods and Actor-Critic (A2C)

Generalized policy iteration improves a policy by imitating an improved policy found through policy improvement. When the policy function is modeled as a neural network, **policy gradient** methods exist as an alternative that instead directly optimizes the policy with respect to some performance measure. A popular class of policy gradient methods is **Actor-Critic** [33], where an "actor" uses a policy network to choose actions given states, and a "critic" is responsible for "criticizing" the action choice of the actor using a value network.

### 2.4.6 Temporal-Difference Learning and Sarsa

**Temporal-Difference** (TD) learning is a category of RL methods that performs updates to a value function to minimize **TD error** $\delta_t$. Given a state $S_t$, successor state $S_{t+1}$ and reward $R_{t+1}$, TD error measures the difference between a previous estimation of the value of a state, and a new estimate calculated by combining the received reward and the estimated value of the successor state[13]:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.12}$$

A TD method can use this value to approximate the true value function corresponding to the followed policy, eg:

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t \tag{2.13}$$

**Sarsa** is an example on-policy TD method that simultaneously learns a state-value function and optimizes a policy function (derived from $Q$). At each step $t$ Sarsa chooses an action $A_t$ using $Q$. Higher-valued actions are preferred, but there must be a non-zero probability of choosing a lower-valued action for the method to converge to optimal behavior. Sarsa then receives a reward $R_{t+1}$ and a successor state $S_{t+1}$, and selects an action $A_{t+1}$ using the same approach. It now has access to $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$, and uses this to perform an update to $Q$:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\big[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\big] \tag{2.14}$$

[13] A formulation also exists for $Q$ functions.

### 2.4.7 Q-learning

**Q-learning** can be seen as an off-policy variant of Sarsa, where the update step is modified to directly optimize $Q$ independent of the policy being followed:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big] \tag{2.15}$$

The crucial difference to Sarsa is that TD error is calculated with respect to the *best* action from $t+1$, rather than the one chosen while stepping through the environment.

## 2.5 Game Theory

According to Shoham and Leyton-Brown [44]: "Game theory is the mathematical study of interaction among independent, self-interested agents." It provides a useful formalism when modeling games for the purpose of creating computational agents. This section introduces relevant terminology and techniques. To avoid confusion with reinforcement learning and simplify the reading of later chapters, some unconventional choices are made with respect to terminology. These choices are pointed out in footnotes, to allow the reader to relate the text to other work utilizing game theory. Refer to Shoham and Leyton-Brown [44] for a detailed treatment.

**Game theory** uses the formalism of **actions**, **strategies**, and **payoff**. The field consists of many **forms**, but only the **normal-form** game is directly relevant to this thesis. In a normal-form game with $N$ players, each player $p$[14] chooses an action $a_p \in A_p$ according to their strategy $\sigma_p$[15], where $A_p$ is the set of all available actions for player $p$, and $\sigma_p$ defines a probability distribution over available actions from which an action can be sampled[16]. Players then receive **payoff** $u_p(\vec{a})$ based on the **joint action**[17] $\vec{a} = [a_1, a_2, \ldots, a_N]$; A vector of actions for each player. The combination of all players' individual strategies is called a **joint strategy**[18] $\sigma$. We also define the **expected payoff** for a joint strategy $\mathbb{E}\big[u_p(\sigma)\big]$ as the expected payoff for player $p$ when the actions are chosen according to the joint strategy $\sigma$:

$$\mathbb{E}\big[u_p(\sigma)\big] = \sum_{\vec{a}} \sigma(\vec{a}) u_p(\vec{a}) \tag{2.16}$$

Given a joint strategy $\sigma$, the **best response** (BR) strategy $\sigma_p^*$ for player $p$ is any strategy that achieves maximal expected payoff [3]. Denoting the joint strategy defined for all players except $p$ as $\sigma_{-p}$, the best response strategy is defined as:

[14] Typically $i$.
[15] Typically $\pi$. $\sigma$ is usually reserved for equilibrium strategies.
[16] Strategies can also be deterministic, but they are always probabilistic (mixed) for the purpose of modeling Diplomacy.
[17] Typically "action profile".
[18] Typically "strategy profile".

$$\sigma_p^* = \underset{\sigma_p}{\operatorname{argmax}}\, \mathbb{E}\big[u_p(\sigma_p, \sigma_{-p})\big] \tag{2.17}$$

The **exploitability** of a joint strategy is measured as the maximal increase in expected payoff any one player can gain by playing their best response. A joint strategy $\sigma$ is a **Nash Equilibrium** if each players' strategy $\sigma_p$ is a best response to $\sigma_{-p}$, or put in other words: if the maximal exploitability is 0. A joint strategy is $\epsilon$**-NE** if the maximum gain any player can get by unilaterally deviating is $\epsilon$.

Normal-form games are constrained to modeling situations where players choose a single joint action and then immediately receive payoff. Other forms allow sequential selection of actions and inclusion of information gained throughout the game. The normal-form game can, however, be extended to a wider array of games with the concept of **stage games**. Within a more complex game, a stage game is a normal-form game defined by applying constraints to the larger game and introducing a substitute payoff function. Usefully defined stage games can help solve more complex games. As an example, consider a group of stock traders in a stock market. A stage game could be formed by defining a subset of stock options available for purchase for each trader (player), and defining payoff as the profit gained from a purchase after a few weeks. Repeatedly solving this stage game well achieves a profitable investment portfolio, and is simpler than applying a more expressive form of game theory. This idea applies in complex board games as well.

### 2.5.1 Regret Matching (RM)

**Regret Matching** [36, 17, 3] is an algorithm to approximate a Nash Equilibrium in two-player zero-sum (2p0s) normal-form games. A zero-sum game is a game where the sum of player payoffs is 0. RM is only guaranteed to converge to NE in 2p0s games, and converges to a "coarse correlated equilibrium" in general [19]. Recent successes in non-2p0s games, including Diplomacy, have demonstrated that it nevertheless produces competitive policies with low exploitability [3]. In line with related work, this thesis makes the working assumption that RM produces an approximation of NE in general, and leaves the study of the nuances of RM to other research.

The sampled form of RM proposed by Gray *et al.* [17] is described. It works by maintaining cumulative "regret" per action, intuitively understood as how much the agent has regretted not taking the action in the past. The cumulative regret of action $a_p$ at iteration $t$ for player $p$ is denoted $\text{Regret}_p^t(a_p)$. On each iteration $t$, the algorithm constructs a joint strategy $\sigma_p^t$ for player $p$ which prefers actions with higher positive regret. If some action has positive regret, $\sigma_p^t(a_p)$ is assigned a probability according to the action's regret relative to total regret. Otherwise, the probability is uniform.

$$\sigma_p^t(a_p) = \begin{cases} \frac{\max\{0,\text{Regret}_p^t(a_p)\}}{\sum_{a'_p \in A_p} \max\{0,\text{Regret}_p^t(a'_p)\}} & \text{if } \sum_{a'_p \in A_p} \max\{0, \text{Regret}_p^t(a'_p)\} > 0 \\ \frac{1}{|A_p|} & \text{otherwise} \end{cases} \tag{2.18}$$

Then, an action $a_p^*$ is sampled from $\sigma_p^t$ for each player $p$, and the regret of all actions $a_p$ is increased by the regret of "not having played" that action given that other players play according to $a_{-p}^*$: The difference between the payoff of playing that action and the expected payoff when playing by $\sigma_p^*$.

$$\text{Regret}_p^{t+1}(a_p) = \text{Regret}_p^t(a_p) + u_p(a_p, a_{-p}^*) - \sum_{a'_p \in A_p} \sigma_p^t(a'_p) u_p(a'_p, a_{-p}^*), \tag{2.19}$$

Over many iterations $T$ the average strategy $\overline{\sigma}_p(a_t) = \frac{\sum_t \sigma_p^t(a_p)}{T}$ is guaranteed to converge to a coarse correlated equilibrium in general [17, 19], and is assumed to converge to NE in this thesis.

Regret Matching is one implementation of **regret minimization**, the concept of minimizing regrets over time to arrive at an equilibrium. The Hedge algorithm [30, 15] is an example of another Regret Minimization implementation.

### 2.5.2 Double Oracle (DO)

**Double Oracle** [32, 3] is an algorithm for finding NE in normal-form games with large action spaces. An "oracle" in Double Oracle is an externally supplied best response operator. The name Double Oracle stems from the application of the algorithm to two-player normal-form games, where an oracle (BR operator) is assumed for both players (as opposed to one player - Single Oracle). The algorithm generalizes to $N$-player games, where an oracle is assumed for every player.

DO leverages the fact that iteratively computing a best response is computationally cheaper than computing an NE. It maintains a candidate action space $A_p^t \subseteq A_p$ as a subset of the full action space. On each iteration $t$, an NE $\sigma_p^t$ is computed for the matrix game restricted to the candidate actions $A_p^t$. Then, each player $p$ finds the best response to $\sigma_p^t$ among actions in the full action space $a_p^{t+1} \in A_p$. This best response is added to the candidate actions for the next iteration: $A_p^{t+1} = A_p^t \bigcup \{a_p^{t+1}\}$. The algorithm terminates once the maximal exploitation of the NE goes below a set threshold.

DO works well for games where a small subset of the action space has support in NE. In this case, DO can find an approximation of the NE quicker than a calculation on the full matrix game since it will only have to perform NE calculations on a subset of the full action space.

Implementations of DO must choose a method for calculating NE and BR. RM and Hedge are alternatives for calculating NE.

### 2.5.3 Nash Q-Learning

**Nash Q-learning** [22, 3] is an extension of the reinforcement learning algorithm Q-learning (section 2.15) adapted to work in multi-agent environments, based on the framework of stochastic games [43]. In Nash Q-learning, the learned $Q$ function is defined over joint actions, rather than over individual actions: $Q(s, a_1, a_2 \ldots, a_N) = Q(s, \vec{a})$, where $N$ is the number of players. $Q(s, \vec{a})$ approximates the expected return when all agents follow a Nash Equilibrium from $s_{t+1}$. Its update rule is similar to that of Q-learning, but replaces the max operation with the expected payoff when playing by a NE calculated for a stage game rooted in $s_{t+1}$ and payoffs given by $Q(s_{t+1}, \vec{a}_{t+1})$:

$$Q(s_t, \vec{a}_t) \leftarrow Q(s, \vec{a}_t) + \alpha\Big[R_t + \gamma \sum_{\vec{a}_{t+1}} \sigma(\vec{a}_{t+1}) Q(s_{t+1}, \vec{a}_{t+1})\Big] \tag{2.20}$$

Implementations of Nash Q-learning must choose a method for calculating NE in the stage game. Hu and Wellman [22] use the Lemke-Howson method [8], which only works for two-player games. With large action spaces, exact calculation of NE is intractable. An approximation of NE is useful in these scenarios, for example RM (section 2.5.1).

## 2.6 Summary of Background

This chapter has introduced relevant background theory. This section summarizes key concepts and highlights connections between the covered topics.

The Diplomacy board game was introduced. In Diplomacy, players control national powers in 20th-century Europe, aiming to gain control of a majority of the map through tactics and negotiation. Players take actions by simultaneously issuing orders to units on the board, and the orders allow for international cooperation. The original formulation of the game includes negotiation in natural language, but a culture of creating "game variants" exists, the most common of which is non-communicative Diplomacy, or No-Press Diplomacy. The original formulation is often labeled Full-Press to distinguish it from No-Press. Other notable variants are Restricted-Press, FvA, and Pure.

Neural networks were introduced as compositions of parameterized differentiable functions. They can be trained to perform a desired transformation from an input space to an output space by adjusting parameters (or "weights") with respect to a loss function using back-propagation. A loss function expresses a measure of distance to optimal parameters and is defined by comparison between network output given some input, and the ground-truth desired output for that input. KL-Divergence can be used as a loss function when the output of the network is a probability distribution.

The concept of neural network architecture as the composition of a neural network was introduced. The simplest architecture is the feed-forward neural network which consists of a series of layers, each layer performing a linear transform-

ation of its input with a weight matrix, and then applying a non-linear activation function to produce output. More complex architectures that take advantage of structure in data were also covered. A graph neural network takes advantage of the structure and properties of graphs. Graph convolutional networks were introduced as an instance of GNN that normalize node updates with respect to relative neighborhood sizes. A recurrent neural network generalizes over sequences of data and allow for variable-length input. RNN include recurrent connections so that the processing of a sequence step can influence future sequence steps. If recurrent connections exist between network output and the internals of the RNN (hidden state), successfully producing a correct sequence is difficult. Teacher forcing addresses this by supplying ground-truth output to future sequence steps, rather than network output. RNN are notoriously hard to train, and "gated" RNN exist that learn to retain and forget information while processing a sequence. LSTM is the original gated RNN. It is widely used, and available as a black-box component.

A visual language based on the concept of computational graphs was introduced to visualize and reason with neural network architecture.

Embeddings were introduced as a way to achieve a real-valued vector representation of categorical variables. Embeddings can be used as components of neural networks.

Reinforcement learning was introduced. In reinforcement learning, an agent (learner) perceives its environment through states, acts by choosing an action from a policy function, and receives a reward. The agent seeks to find an optimal policy function that maps from states to actions that maximize expected future discounted total reward, also called return. The value function of a policy function can be defined that maps from states to return when following the policy function from that state. The agent learns by starting from an initial policy function, value function, or both, and iteratively improving them towards optimal behavior.

Generalized policy iteration was introduced as the general concept of simultaneously making the value function consistent with the policy function (policy evaluation), and making the policy greedy with respect to the value function (policy improvement). Policy iteration and value iteration were introduced as instances of GPI. The term GPI is used in this thesis to refer to any method that utilizes a policy improvement operator to improve either the policy function, the value function, or both.

Function approximators are useful in RL to generalize across states and actions, and neural networks can be employed as policy networks and value networks. This is called deep reinforcement learning.

Policy gradient methods were introduced as an alternative to GPI when using a neural network as a policy function. Where GPI trains the policy network by imitating the result of a policy improvement operator, policy gradient methods optimize network parameters with respect to some performance measure. Actor-Critic (A2C) was introduced as a popular class of policy gradient methods.

Game theory was introduced as a useful formalism when modeling games for the purpose of creating computational agents. The anatomy of a normal-form

game was presented, introducing (joint) actions, (joint) strategies, and payoff. Players take joint actions following a joint strategy, and receive payoff. The expected payoff of a joint strategy is defined as the expected payoff when all players choose actions according to the joint strategy.

Best response was presented as the player strategy that achieves maximal expected payoff against some joint strategy, and the exploitability of a joint strategy was defined as the maximal increase one player can gain in payoff by playing their best response to that joint strategy. Nash Equilibrium was presented as a joint strategy that no player is able to exploit.

The concept of a stage game was introduced as a way of modeling complex games in normal-form by limiting the action space and choosing a substitute payoff function.

Some key game-theoretic techniques were introduced. Regret Matching is a regret minimization algorithm that is guaranteed to converge to a Nash Equilibrium in two-player zero-sum games. The algorithm has also been shown to produce competitive policies in the general case. Double Oracle is an algorithm for finding Nash Equilibrium in normal-form games with large action spaces. It iteratively expands a set of candidate actions by looking for actions that are best responses to the Nash Equilibrium over the candidate actions. Regret Matching can be used in DO to create NE.

Nash Q-learning was introduced as a game-theoretic reinforcement learning technique based on joint action return when all players play by Nash Equilibrium. It is an extension of Q-learning, which itself is a variation on Sarsa, which is an instance of Temporal-Difference learning. These antecedent techniques were presented to contextualize Nash Q-learning.

# Chapter 3

# Related Work

This chapter contextualizes the thesis by presenting related work. The chapter consists of an outline of the literature review process, and a presentation of related work in three parts: First, a brief summary of early work on Diplomacy AI is given. Then follows a brief summary of related work in RL for other board games. Finally, the chapter concludes with a detailed presentation of major papers on RL for Diplomacy leading up to state-of-the-art.

## 3.1 Literature Review Process

Literature review was conducted in the preparatory phase of the thesis. The overarching goals of the process were to:

1. Establish a good understanding of the current state-of-the-art in RL for Diplomacy.
2. Trail development back to the earliest work on RL for Diplomacy.
3. Obtain an overview of early work on AI for Diplomacy, prior to the application of RL.
4. Obtain an overview of the context within which the techniques used in RL for Diplomacy exist (other games, alternative approaches, etc.).

An adaptation of the "Snowballing" [26] method was used. "Snowballing" iteratively grows a base of related work by following citations backwards (papers cited in base of papers) and forwards (papers citing work in the base). The search engines Google, Google Scholar and Semantic Scholar were used during forwards snowballing. The initial base of related work consisted of the seed paper "Learning to Play No-Press Diplomacy with Best Response Policy Iteration" by Anthony *et al.* [1] (Section 3.4.2). When considering a discovered paper for inclusion into the base of related work, it was evaluated with respect to the literature review process goals.

## 3.2 Early Work on AI for Diplomacy

Research on Diplomacy prior to 2019 focused primarily on rule-based agents without learning. The earliest published work is Diplomat [28, 29] from the 1980s, which uses symbolic reasoning to evaluate and make proposals in a computer-friendly negotiation language. Other notable rule-based agents are DipBlue [12], D-Brane [24] and Albert [9]. Albert was the state-of-the-art in no-press Diplomacy until Paquette *et al.* [37] introduced DipNet in 2019.

Much of the early work on Diplomacy has been done in the context of the field of Automated Negotiation. The annual competition ANAC - Automated Negotiating Agents Competition - featured a Diplomacy League in 2017, 2018 and 2019. [2] The competition set in place strict requirements to ensure a winner truly performs meaningful negotiation, and does so better than its competitors. To date, no agent has been able to beat this challenge. The Diplomacy League was discontinued in 2019.

One early work that incorporates a form of RL is Shapiro *et al.* [42] from 2003. They utilize Temporal-Difference learning and self-play to fit the weights of a "Pattern-Weights" model to play the No-Press variant of the game.

## 3.3 Reinforcement Learning for Other Board Games

Three games stood out as clear inspirations behind reinforcement learning for Diplomacy while conducting the literature review. These are presented briefly to provide a broader context of work in RL for Diplomacy.

Silver *et al.* [46] propose AlphaGo Zero, which employs policy iteration with Monte Carlo tree search (MCTS) as the policy improvement operator in the board game of go. MCTS performs a heuristic search over available actions using a policy function to narrow down the search and a value function to evaluate non-terminal states. The policy and value functions are modeled as a neural network using the Encoder-Decoder[1] architecture. The policy head is trained to imitate the result of MCTS, and the value head is trained to predict end-game scores. AlphaGo Zero trains with self-play from scratch with no human data, and is the first agent to reach superhuman skill in go without human examples or guidance.

Vinyals *et al.* [50] propose AlphaStar, the first agent to achieve Grandmaster level in StarCraft II. The API of StarCraft II represents actions as functions taking arguments, and AlphaStar decomposes the action space into a sequential choice of a function name followed by a sequence of arguments.

Brown and Sandholm [5] propose Pluribus, the first agent to reach superhuman skill in six-player no-limit Texas hold'em poker. Pluribus trains with self-play from scratch with no human data, and utilizes game-theoretic search through Regret Matching[2].

[1] The term Encoder-Decoder is not used by them. The categorization was chosen to relate the work to this thesis.

[2] Counterfactual Regret Minimization is used, which according to Neller and Lanctot [36] in turn

## 3.4 Reinforcement Learning for Diplomacy

This section presents major Diplomacy RL papers in order of publication, with text on later papers often building upon the presentation of earlier papers. A summary is given to conclude the section, including an overview of the presented papers in table 3.1.

### 3.4.1 No Press Diplomacy: Modeling Multi-Agent Gameplay

Paquette *et al.* [37] propose *DipNet*, a neural network-based agent capable of beating state-of-the-art rule-based agents in the no-press variant. DipNet is first trained on human data with supervised learning to approximate human play, then further trained with reinforcement learning through self-play.

The model architecture consists of graph convolutional networks[3] (GCN) that "encode" the board state, and LSTM that "decode" the encoding into orders for each orderable unit. GCN allow the network to capture the complex board state of Diplomacy as a graph, with nodes corresponding to regions and edges corresponding to adjacencies. Use of a sequential decoding of orders reduces the output dimension of the network, while still allowing coordination between units. Rather than producing action probabilities over the whole action space, the network sequentially produces a distribution over all unit orders for each orderable unit. After illegal orders are masked out, this distribution is used to sample orders. Unit orders are sampled such that the sampled order $o_{i-1}$ is fed into the network when producing order $o_i$. As clarified by Bakhtin *et al.* [3], this sequential sampling results in action probabilities:

$$\pi(a) = \prod_{i=1}^{t} \pi(o_i | o_0, \ldots, o_{i-1}) \tag{3.1}$$

There is no natural sequential ordering of locations when decoding orders, and Paquette *et al.* [37] propose to use a "topological sorting" of unit locations, such that adjacent units are likely to be ordered in sequence.

The RL training of DipNet is done with A2C, rewarded both based on supply centers gained or lost from turn to turn, and the final score at the end of the game. Learning is bootstrapped from the supervised model. The performance between agents trained with SL and RL is not significant. They observe that the RL agent issues more support orders, but that less of these are successful, suggesting that RL agents are less effective at cooperation.

uses Regret Matching.

[3] Anthony *et al.* [1, Appendix C] points out that the GNN of DipNet is not truly convolutional.

### 3.4.2 Learning to Play No-Press Diplomacy with Best Response Policy Iteration

Anthony *et al.* [1] propose a Sampled Best Response (SBR) operator and a family of policy iteration algorithms tailored to using approximate BRs such as SBR, referred to collectively as Best Response Policy Iteration (BRPI) algorithms. They apply BRPI to no-press Diplomacy, and improve upon previous state-of-the-art.

SBR makes computational tradeoffs to become a tractable approximation of a true best response (BR). For a given state, they consider the stage game formed by each player taking one of a subset of actions sampled from a policy network, and receiving a payoff via the value of the resulting state according to a value network.

The proposed family of BRPI algorithms employ SBR as a policy improvement operator for policy iteration in two main ways: (1) Iterated Best Response (IBR) calculates a policy improvement with respect to the policy of the previous iteration. (2) Fictitious Play Policy Iteration (FPPI) calculates a policy improvement with respect to the empirical distribution over historical opponent strategies.

To train their model with a given BRPI variant, they first train a policy and value network to imitate human data like Paquette *et al.* [37]. They then iteratively generate a dataset with actions chosen by the improvement operator, and train the policy and value network to imitate this dataset. The policy is trained to imitate the result of the policy improvement operator and the value network is trained to predict end-game results, similar to the training of AlphaGo Zero Silver *et al.* [46].

Their neural architecture builds on the work of Paquette *et al.* [37], but makes several improvements. Most notably, they replace the LSTM decoder with a "Relational Order Decoder" - a stack of 4 GNNs with residual connections. Their supervised learning prediction accuracy improves upon state-of-the-art.

They cite the A2C-agent of Paquette *et al.* [37] as an unsuccessful application of RL to Diplomacy, since it did not improve upon their SL-agent when measured with the Trueskill rating system [21]. All of their BRPI algorithms outperform DipNet, the previous state-of-the-art. They consider this as the first successful application of RL to Diplomacy.

To conclude, they postulate that successful application of RL is a prerequisite for investigating the complex mixed motives and many-player aspects of Diplomacy. They list five questions for future work. Of special note is "(1) What is needed to achieve human-level No-Press Diplomacy AI?" and "(2) Can we build agents that reason about the incentives of others [...] by applying opponent shaping?" (1) has later been achieved by Gray *et al.* [17], and (2) was investigated by Hatlø [20], albeit without BRPI.

### 3.4.3 Human-Level Performance in No-Press Diplomacy via Equilibrium Search

Gray *et al.* [17] propose *SearchBot*, an agent that augments a policy trained with imitation learning on human data with Regret Matching at inference time. A sampled form of Regret Matching is used at each state to calculate an equilibrium in the stage game formed by each player taking one action and receiving a payoff based on a limited rollout from the resulting state using the current policy and value network. This one-ply lookahead search is reminiscent of the SBR operator proposed by Anthony *et al.* [1] with two key differences: (1) SearchBot performs rollouts, where SBR immediately queries its value network, and (2) SearchBot approximates an equilibrium, where SBR approximates a BR. A BR seeks to maximally exploit a fixed joint strategy, whereas an equilibrium seeks to find a joint strategy that minimizes exploitability. Gray *et al.* [17] empirically show that SearchBot achieves low exploitability compared with previous state-of-the-art, and that human experts were unable to exploit the agent during repeated play.

After calculating an approximate equilibrium with Regret Matching, SearchBot plays its part in the equilibrium. Notice that the equilibrium policy is not used for RL, as the search procedure is computationally expensive.

The architecture of SearchBot is built on that of Paquette *et al.* [37]. They leverage the improvements by Anthony *et al.* [1], except for the Relational Order Decoder, which was considered too computationally expensive relative to the improvement in performance. They propose a Featurized Order Decoder. Their supervised learning prediction accuracy improves upon state-of-the-art.

The approach is inspired by successes from applying Regret Matching in Poker [35, 5], another many-player game popular for AI research. SearchBot is the first successful application of RM to a complex game involving cooperation, and suggests that its use is not limited to purely adversarial games.

SearchBot performed better than the previous state-of-the-art of Diplomacy AI. Additionally, SearchBot was evaluated against human player on a popular online Diplomacy platform. It ranked in the top 2% of human players, thereby becoming the first Diplomacy agent to achieve human-level performance.

They propose three directions for future research on the topic of search: (1) n-ply search with Counterfactual Regret Minimization [51], (2) Leveraging RM in RL, and (3) Leveraging RM in Diplomacy variants with explicit communication. (2) was later achieved by Bakhtin *et al.* [3], and (3) is achieved by FAIR *et al.* [11].

### 3.4.4 No-Press Diplomacy from Scratch

Bakhtin *et al.* [3] propose an algorithm for action exploration and equilibrium approximation in games with combinatorial action spaces. Their focus is the creation of an agent which can learn Diplomacy from scratch without hand-crafted

reward signals.[4]

The basis for their approach is the proposed algorithm Deep Nash Value Iteration (DNVI), a variant of Nash Q-learning (section 2.5.3). DNVI transforms Nash Q-learning from using state-action values $Q(s, \vec{a})$ to using state values $V(s)$. This is an advantage since state-action networks are hard to express for games with large action spaces. They base their transformation on the assumption that the reward signal of going from state $s$ to state $s'$ through (joint) action $\vec{a}$ is independent of $s$ and $\vec{a}$. Since they make it a goal to avoid handcrafted reward signals, this assumption holds: The only state that matters in Diplomacy is the terminal one, and once you've won or lost, it doesn't matter how.

DNVI maintains a state value network and an action proposal network. On each game turn, the action proposal network suggests action candidates[5], a subset of the full action space. The algorithm then proceeds as Nash Q-learning, except only considering the candidate actions in a stage game defined similarly to Anthony *et al.* [1] and Gray *et al.* [17]. Once an NE has been calculated, it is used to train the networks. The action proposal network is trained to imitate the NE, and the value network is trained to imitate the expected payoff of the stage game when playing the NE. The agent then chooses an action by playing its part in the NE. For NE calculation they use RM, like Gray *et al.* [17].

To enable learning from scratch, DNVI is enhanced with an action exploration procedure inspired by Double Oracle (section 2.5.2). The candidate actions from the action proposal network are used as the initial action subset for DO, and DO extends the candidates with repeated BR and NE calculations. They alter DO to only consider a subset of all actions for BR calculation. This subset is generated by pertubing candidate actions, based on the intuition that pertubation of candidate actions is more likely to produce good actions than random sampling from all actions. They empirically show that this intuition holds in practice.

They include two stages of pre-training. The agent first trains with uniformly sampled action candidates and value targets for learning based on end-game scores instead of the expected payoff under NE. Then, a second pre-training stage trains the agent as normal, but without action exploration[6]. These pre-training stages prime the action proposal network and value network with reasonable values.

They propose DORA - Double Oracle Reinforcement Learning with Action exploration, an agent trained from scratch with DNVI and their DO-like action exploration procedure. The DO-like procedure is used both at training and at inference time. As their concern is the combinatorial action space of Diplomacy, they evaluate DORA using a variant called FvA - "France versus Austia". FvA is identical to classic Diplomacy, but includes only two players. DORA achieves superhuman performance against top players on a popular online Diplomacy platform on the

[4] A handcrafted reward signal in Diplomacy is any reward not directly associated with the win condition of the game.

[5] Action candidates are called "plausible actions" in the rest of the thesis.

[6] Only for 7-player Diplomacy.

FvA variant.

In addition to DORA, they propose HumanDNVI-NPU - "deep Nash value iteration, no proposal update". HumanDNVI-NPU also uses DNVI, but is bootstrapped from imitation learning on human data, does not train its action proposal network over time, and does not utilize their DO-procedure. HumanDNVI-NPU beats SOTA in 7-player no-press Diplomacy

DORA fails to beat state-of-the-art in 1v6 in classic Diplomacy, and Bakhtin *et al.* [3] attribute this to DORA playing by an NE incompatible with human play. This is used as evidence that self-play from scratch may be insufficient to achieve superhuman performance in Diplomacy. DORA also performs poorly against HumanDNVI-NPU in 1v6, and vice versa. Two separately trained DORA display the same behavior. This is used as evidence for the presence of a wide space of equilibria, and DORA is proposed as the first technique for exploring this wider space of equilibria without being constrained to human priors.

### 3.4.5 Learning to Play No-Press Diplomacy from Self-Play: Deep Reinforcement Learning Focusing on Collaboration Between Agents[7]

Hatlø [20] proposes in his master thesis an application of "Learning with Opponent-Learning Awareness" (LOLA) [13] to no-press Diplomacy, as suggested by Paquette *et al.* as one of the most exciting paths for future research. The thesis implements two variants: "France vs. Austria", and "Pure". LOLA was deemed as not suitable for the former variant, due to it being too computationally demanding. For the latter, results show that although a LOLA-agent was able to learn to play the variant, it performs worse than a reference A2C-agent. The LOLA-agent was also shown to be worse at cooperating than the reference A2C-agent, thereby providing evidence that LOLA is not appropriate as a method to increase cooperation in Diplomacy agents. The thesis implements an architecture inspired by Paquette *et al.* [37], Anthony *et al.* [1], Gray *et al.* [17] and Bakhtin *et al.* [3].

It should be noted that the RL system implemented by Hatlø [20] most resembles that of Paquette *et al.* [37], being policy gradient based. As pointed out in section 3.4.2, Anthony *et al.* [1] does not count the A2C agent of Paquette *et al.* [37] as a successful application of RL to Diplomacy, since it did not increase performance over its imitation learned bootstrapped policy. Later RL works [1, 3] opt for policy iteration techniques based on search instead of a policy gradient approach. The fact that no policy gradient approach can be said to have been successfully applied to Diplomacy can perhaps help explain why LOLA did not produce the desired results.

[7] Title translated from Norwegian to English. Original title: "Å lære No-Press Diplomacy fra Selvspill: Dyp Reinforcement Learning med Fokus på Samarbeid mellom Agenter"

### 3.4.6 Modeling Strong and Human-Like Gameplay with KL-Regularized Search

Jacob *et al.* [23] study the trade-off between human-like behavior and strong performance in multi-agent decision problems. They observe that agents trained with imitation learning on human data tend to fail to match human expert performance, while learning through self-play from scratch tends to create agents that diverge from human play. They study two kinds of game: Perfect-information games through chess and go, and imperfect-information, simultaneous action games through No-Press Diplomacy. For each of these kinds of game, a search algorithm penalized by divergence to some prior policy is proposed.

For imperfect-information, simultaneous action games they propose piKL-Hedge - Policy-Regularized Hedge, a variant of the Hedge algorithm regularized by KL-divergence to some prior policy. The influence of the regularization in piKL-Hedge is controlled through a parameter $\lambda$. When $\lambda = 0$, piKL-Hedge behaves like SearchBot, with no incentive to stay close to the prior policy. When $\lambda = \infty$, piKL-Hedge's behavior is dictated by the prior policy, and search does not enhance performance.

Using the approach of Gray *et al.* [17] of augmenting an agent trained with imitation learning on human data with search, they propose piKL-HedgeBot. The neural architecture of piKL-HedgeBot follows that of Bakhtin *et al.* [3], using a Transformer-based encoder and an LSTM-based decoder. Two notable enhancements to the model architecture are: (1) The input features describing each location to be encoded is extended with enough information that the representation becomes equivariant to permutation, allowing for steps in the training process that reduce overfitting, and (2) The value network approximates the result of rollout from a given state, rather than being used for evaluation of successor states after rollout, inspired by the success of Silver *et al.* [45].

The joint task of (1) Accurately predicting human actions, and (2) Achieving a high winrate against a purely imitation learned policy, is considered. A host of agents are compared, each employing varying degrees of regularization. The comparison shows that highly favorable combinations of the two goals can be achieved by varying the $\lambda$ term. Values for $\lambda$ exist that improve performance in both goals compared to an unregularized search agent and a pure IL agent, thereby advancing state-of-the-art in terms of IL agents enhanced by search at inference time.

Although their work is not focused on RL, they do note that DORA and HumanDNVI-NPU of Bakhtin *et al.* [3] perform poorly in predicting human moves.

With favorable empirical results for Chess, Go, and Diplomacy, Jacob *et al.* [23] demonstrate that regularizing search by divergence from human play can be beneficial both for accurately predicting human moves and achieving strong play.

### 3.4.7 Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning

Bakhtin *et al.* [4] combine the regularized search algorithm of Jacob *et al.* [23] with the self-play reinforcement learning of Bakhtin *et al.* [3] to create Diplodocus,

the first Diplomacy agent to reach expert-level performance in the no-press variant through self-play from scratch.

Based on piKL[8] of Jacob *et al.* [23] they propose Distributional Lambda piKL - DiL-piKL. The improvement of DiL-piKL over piKL is that the $\lambda$ parameter is sampled from a distribution $\beta$ at each iteration rather than staying fixed.

To utilize DiL-piKL for self-play reinforcement learning, they use DORA by Bakhtin *et al.* [3] as a basis, and replace the equilibrium-finding algorithm with DiL-piKL both during training and inference time. They call the resulting algorithm RL-DiL-piKL. While calculating an equilibrium they use a fixed distribution $\beta$, while when actually choosing an action given a found equilibrium they use a fixed low $\lambda$. This fixed low $\lambda$ makes the agent play closer to optimal given a found equilibrium.

Using RL-DiL-piKL they train Diplodocus. The policy and value networks are initialized from imitation learning on human data, and search is regularized against these same imitation learned networks. Interestingly, they do not utilize the DO action exploration procedure of DORA during training, as they found initialization from imitation learning made it unnecessary.

Two Diplodocus agents participating in a 200-player no-press Diplomacy tournament involving 62 human participants ranked first and third, and achieved the highest average score among participants who played more than two games. The participants spanned skill levels from beginner to expert.

### 3.4.8 Human-level Play in the Game of Diplomacy by Combining Language Models with Strategic Reasoning

FAIR *et al.* [11] propose Cicero, the first Diplomacy agent to achieve human-level play in the original Full-Press formulation of Diplomacy, including negotiation in natural language. This is achieved by combining a language model with Diplodocus of Bakhtin *et al.* [4], the state-of-the-art in non-communicative Diplomacy agents. The main innovation in this work is the way dialogue with other powers informs and is grounded in the agent's tactical understanding of game state. Since this thesis is focused on No-Press Diplomacy (Section 4.1), it has been given limited treatment.

### 3.4.9 Negotiation and Honesty in Artificial Intelligence Methods for the Board Game of Diplomacy

Kramár *et al.* [27] propose and study explicit negotiation algorithms for a "Restricted-Press" variant of Diplomacy that allows communication in a computer-friendly negotiation protocol. The negotiation algorithms build on the non-communicative agents of Anthony *et al.* [1]. Since this thesis is focused on No-Press Diplomacy (Section 4.1), the paper has been given limited treatment.

[8] The piKL-Hedge variant.

| Year | Section | Citation | Agent | Variant(s) |
|---|---|---|---|---|
| 2019 | 3.4.1 | Paquette *et al.* [37] | DipNet | No-Press |
| 2020 | 3.4.2 | Anthony *et al.* [1] | BRPI[9] | No-Press |
| 2021 | 3.4.3 | Gray *et al.* [17] | SearchBot | No-Press |
| 2021 | 3.4.4 | Bakhtin *et al.* [3] | DORA | No-Press, FvA |
| 2022 | 3.4.5 | Hatlø [20] | HatløA2C[10] | No-Press, FvA, Pure |
| 2022 | 3.4.6 | Jacob *et al.* [23] | piKL-HedgeBot | No-Press |
| 2022 | 3.4.7 | Bakhtin *et al.* [4] | Diplodocus | No-Press |
| 2022 | 3.4.8 | FAIR *et al.* [11] | CICERO | Full-Press |
| 2022 | 3.4.9 | Kramár *et al.* [27] | - | Restricted-Press |

**Table 3.1:** Summary of presented papers utilizing RL for Diplomacy, along with the agent proposed, and the Diplomacy game variant played.

## 3.5 Summary of Related Work

When Paquette *et al.* [37] showed how neural networks could be applied to Diplomacy in 2019, they opened the door to the application of modern deep reinforcement learning techniques to the game. Anthony *et al.* [1] demonstrated the value of introducing game-theoretic concepts to allow for RL, and Gray *et al.* [17] applied game-theoretic techniques with success in Poker [35, 5] to Diplomacy to achieve human-level play in the No-Press game variant. Bakhtin *et al.* [3] further developed game-theoretic techniques to allow an agent to learn the game from scratch, inspired by AlphaGo Zero of [46]. Jacob *et al.* [23] and Bakhtin *et al.* [4] regularized game-theoretic search by a human prior to create an agent that plays at the expert level with humans in the No-Press game variant. FAIR *et al.* [11] combine state-of-the-art in No-Press Diplomacy with language models to create the first human-level agent for the Full-Press variant of the game.

The trend in applying RL to Diplomacy is to define an Encoder-Decoder neural network architecture with separate heads for producing the policy and value functions, and then training this network through policy iteration (Anthony *et al.* [1] and Kramár *et al.* [27]) or value iteration (Bakhtin *et al.* [3], Jacob *et al.* [23], Bakhtin *et al.* [4] and FAIR *et al.* [11]) with game-theoretic search (BR or NE) acting as the policy improvement operator. Search is conducted in the stage game formed by each player taking one of a subset of actions sampled from a policy network acting as an "action proposer", and receiving payoff through next-state value estimation from a value network. The policy network decomposes the action space into sequences of unit orders. Work that instead opts for a policy gradient approach (A2C agents of Paquette *et al.* [37] and Hatlø [20]) does not seem to produce strong agents.

Diplomacy research that labels itself as "policy iteration" trains the value net-

[9] BRPI is a family of algorithms for creating Diplomacy agents.

[10] Agents are simply named A2C and LOLA by Hatlø [20]. The A2C agent is referenced as HatløA2C in this thesis. The LOLA agent is not considered in this thesis.

work through end-game scores when playing on-policy by the NE found in search at each turn. Diplomacy research that labels itself as "value iteration" instead trains the value network through bootstrapped next-state value estimation under the NE found through search. Both camps train the policy network to imitate the result of search. Both of these techniques fall under the term generalized policy iteration as defined in the background chapter, as they train their networks using a policy improvement operator. The term generalized policy iteration is used to allow for easy comparison with policy gradient techniques.

The majority of recent work considers the non-communicative No-Press variant of the game [42, 37, 1, 17, 3, 23, 20, 4]. Bakhtin *et al.* [3] and Hatlø [20] further deviates from the original game formulation by considering the 2-player "France vs. Austria" variant, and Hatlø [20] studies the 7-player "pure" variant featuring a tiny map. Little research exists that trains agents for maps other than the original. Earlier work has a larger focus on negotiation, and uses restricted-press variants that limit communication to a computer-friendly protocol. Kramár *et al.* [27] revisit this variant. Only FAIR *et al.* [11] have been successful on the classic variant of the game including negotiation in natural language.

# Chapter 4

# Methodology

Previous chapters have stated and motivated the goal of the thesis, introduced relevant background theory, and presented relevant work. This chapter details the methodology followed to achieve the goal and ultimately answer research questions.

A modular reinforcement learning system for Diplomacy based on state-of-the-art research is created. Section 4.1 describes the modified Diplomacy ruleset used. Section 4.2 describes the game engine. Section 4.3 describes the variant creation process. Section 4.4 describes the representation through which the agent interacts with the game environment. Section 4.5 describes the neural network architecture used by the agent to produce actions and state values. Section 4.6 describes how the agent leverages a neural network and game-theoretic search to take actions in the game. Section 4.7 describes how agents are trained. Section 4.8 describes how the system can be used to run experiments in service of addressing research questions.

## 4.1 Ruleset Modifications

Given the limited computational resources available to thesis experiments, a restricted version of the fundamental rules of the game is used.

### 4.1.1 Omission of Coastal Areas and Sea Areas

Coastal areas and sea areas are omitted. The game is played entirely on land. This effectively removes fleets, since they cannot exist on land, along with the "convoy" order, since it can only be issued by fleets.

### 4.1.2 No-Press

The non-communicative variant considered by most recent research [37, 1, 17, 3, 23, 4] is used. This restricts communication to what can be expressed through actions in the game (See footnote 6 on page 8).

### 4.1.3 Sum-of-Squares (SoS) Scoring and Turn Limiting

In its original formulation, a game of Diplomacy is supposed to last until one power reaches the victory condition of controlling a majority of the supply centers on the board. This is difficult to achieve, and the game is prone to getting stuck in stalemates. To mitigate this, a maximum number of in-game years is defined for each game variant. If a game ends without a clear victor, the Sum-of-Squares (SoS) scoring mechanism [14, 17] is used, scoring each power by their number of controlled supply centers squared, proportional to the sum of squared supply center counts. Denoting the set of all powers as $\mathcal{N}$ and the supply center count of power $i$ as $C_i$, the SoS score of power $i$ becomes:

$$\text{Score}(i) = \frac{C_i^2}{\sum_{j \in \mathcal{N}} C_j^2}$$

## 4.2 Game Engine

A game engine is necessary to perform computational experiments on Diplomacy. The Diplomacy ruleset is non-trivial to implement correctly: It is a known fact that logical paradoxes can arise which require fallback rules[1]. Therefore, an existing game engine is chosen for this thesis. The engine[2] developed by Paquette *et al.* [37] is chosen. Gray *et al.* [17] refers to it as the MILA[3] Diplomacy Engine[4]. This name is used throughout this thesis. The MILA Diplomacy Engine is chosen for the following reasons:

- Trivial API.
- Support for game state visualization.
- Trivial installation and debugging, as it is written in pure Python.
- Its usefulness is proven by the success of DipNet [37][5].
- Hatlø [20], operating with the same computational resources as this thesis has available, successfully used the engine for his thesis.
- Custom Diplomacy game variants are possible, as leveraged in section 4.3.

The dipcc[6] engine developed for research by Gray *et al.* [17], Bakhtin *et al.* [3], Jacob *et al.* [23], Bakhtin *et al.* [4] and FAIR *et al.* [11] was also considered. It is a faster alternative to the MILA Diplomacy Engine written in C++. The engine was

[1] See for example this article for some paradoxes and their workarounds: `http://www.dipwiki.com/index.php?title=Convoy_Paradoxes`, accessed 17.06.2023.

[2] `https://github.com/diplomacy/diplomacy`, accessed 06.06.2023.

[3] `https://mila.quebec/en/`, accessed 06.06.2023.

[4] `https://github.com/facebookresearch/diplomacy_searchbot/blob/main/dipcc/README.md`, accessed 06.06.2023.

[5] Although the RL in DipNet of Paquette *et al.* [37] was based on policy gradient, not generalized policy iteration as in this thesis. By nature, generalized policy iteration using search puts more pressure on the game engine.

[6] `https://github.com/facebookresearch/diplomacy_searchbot/tree/main/dipcc`

disregarded because it is written specifically for the original variant, and would require substantial rewrites before it could be used for arbitrary variants.

## 4.3 Variant Creation

A feature of the MILA Diplomacy Engine important for this thesis is the possibility of playing game variants beyond the classic formulation of Diplomacy. This is achieved by supplying a `.map` file, together with a corresponding `.svg` file with special annotations used for visualization of game states.

Listing 4.1 is the `.map` file behind the example game variant visualized in figure 4.2a, which pits the Informatics students against the Cybernetics students for control of three of Gløshaugen's cantinas. The visualization in the figure was created by the MILA Diplomacy Engine by combining the `.map` file with a `.svg` file with meticulously chosen content to match the variant.

```
BEGIN SPRING 1901 MOVEMENT

Informatikk (Informatiker) Element
A Element
A Hangaren

Kybernetikk (Kybber) Elektro
A Elektro

ELT = Element
HAN = Hangaren
ELO = Elektro

LAND Element ABUTS Hangaren Elektro
LAND Hangaren ABUTS Element Elektro
LAND Elektro ABUTS Hangaren Element

VICTORY 3
```

**Code listing 4.1:** Example `.map` file for the MILA Diplomacy Engine, specifying the game variant visualized in figure 4.2a. This file, along with the corresponding `.svg` file, is automatically generated.

Creating game variants for the MILA Diplomacy Engine by hand is tedious, as it requires writing a verbose `.map` file, and carefully creating a corresponding `.svg` file that nicely illustrates regions and region adjacencies. To enable more ergonomic experimentation with map variants, a system is created that can generate these files from a simple variant specification in the YAML file format. Importantly, this specification allows automatic generation of the `.svg` file via a combination of "ASCII art" and an adjacency matrix, enabling customizable visualization of variants. The adjacency matrix is also used as input to the neural network, and serves as a useful single source of truth for adjacencies in the variant.

## 4.4 Game Representation

The formalism of RL requires the definition of a state space and an action space. The agent senses the current state of the game through a state from the state space, and acts by supplying a chosen action from the action space.

The state space of Diplomacy can be described by representing features of the state of the board. The action space consists of combinations of unit orders. Due to the size of the action space, the agent is supplied both a board state representation and a representation of the legal action space subset, from which it chooses one legal action.

The following sections detail the board state representation and the action space representation including representation of the legal action subset. The chosen game representation is summarized in section 4.4.3.

### 4.4.1 Board State Representation

Board state is represented using the technique introduced by Paquette *et al.* [37] and adapted by later papers; As a matrix $\mathbf{B}$ with one row per region, and columns made up of concatenated one-hot-encoded features per region; $\mathbf{B} \in \{0,1\}^{R \times F}$, where $R$ is the number of regions and $F$ is the number of bits of the concatenated features. The features chosen mimic Gray *et al.* [17] figure 5, but are adapted to exclude features made irrelevant by the ruleset modifications described in section 4.1. The features are detailed in table 4.1. As an example, figure 4.1 shows how the board state visualized in figure 4.2a would be represented. Powers are sorted alphabetically with "none" first where applicable, and locations are sorted as they appear in the variant specification (listing 4.1).

The state representation does not include adjacency information. Instead, the adjacency matrix found in the variant specification is fed to the neural network during construction for use in GNN.

### 4.4.2 Action Space Representation

Out of the three kinds of phases in Diplomacy, the movement phase is the most interesting. It has by far the largest action space, and is critical to a power's success. Additionally, we can represent the retreat phase the exact same way as the movement phase, and the adjustment phase can make use of a representation that is similar. Therefore, only the action space of the movement phase is explained in detail in this section. Details on how the representation presented here can be utilized for the retreat and adjustment phases are found in appendix D.

In the movement phase, the action space in Diplomacy is combinatorial [3]; An action consists of choosing an order for each orderable unit. The action space representation chosen is detailed in the following sections. First, the space of orders per unit is defined. Then, the space of (joint) actions is defined using the order space. Finally, an elaborate example is given in figure 4.2.

| **Name** | **Value** | **Size** | **Description** |
|---|---|---|---|
| Unit Power | OHE | $P$+1 | Power controlling unit at region, or none. |
| Buildable | Boolean | 1 | Can region be used to build a unit? |
| Removable | Boolean | 1 | Does region unit we can remove? |
| Dislodged Unit | OHE | $P$+1 | Power controlling unit dislodged at region, or none. |
| Supply Center Owner | OHE | $P$+1 | Power controlling supply center at region, or none. |
| Season | OHE | 3 | Spring, Fall or Winter. |
| Build Numbers | Integer | $P$ | Number of builds per power. Can be negative. |

**Table 4.1:** Features for a single region in **B**. $P$ is used to mean the number of powers in the variant. OHE is used to mean one-hot-encoding.

| | Unit Power | | | Buildable | Removable | Dislodged Unit | | | Supply Center Owner | | | Season | | | Build Numbers | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ELT | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| HAN | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| ELO | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**Figure 4.1:** Board state representation **B** of game state in figure 4.2a There are $R = 3$ regions, and the features require $F = 16$ bits, resulting in $\mathbf{B} \in \{0,1\}^{3\times16}$. Rows are labelled with the region they denote, and columns are grouped by the feature they help describe. Table 4.1 details the meaning of each feature.

### Order Space

A list $\Omega$ of all orders possible by any unit in any region is defined for each variant. As an example, the indexed global list of all orders possible in the variant shown in figure 4.2a is included in appendix A. Orders are represented as integer indexes into $\Omega$. Throughout this thesis, the name "order" is often used interchangeably with "order index", since there is a one-to-one relationship between the two.

In the movement phase, an action consists of choosing an order for each orderable unit. Given $N$ orderable units denoted $i \in \{1, 2, \ldots, N\}$, the set of legal order indexes for unit $i$ is denoted $O_i$:

$$O_i = \{\, k \in \mathbb{N} \mid \Omega_k \text{ is a legal order for unit } i \,\}$$

Given unit $i$ with $M_i$ legal orders, the individual legal order indexes are denoted:

$$O_i = \{o_i^1, o_i^2, \ldots, o_i^{M_i}\}$$

The vector of legal orders per unit $i$ is denoted:

$$O = [O_1, O_2, \ldots, O_N]$$

The action space is highly variable: The number of legal orders $M_i$ in a game state varies per unit, and the number of orderable units $N$ varies across game states.

### Action Space

An action consists of choosing an order $o_i \in O_i$ for each orderable unit $i$. This makes the action space combinatorial [3]; The size of the action space is $|A| = \prod_{i=1}^{N} |O_i|$. Since $|O_i|$ can become large by itself, $|A|$ can grow explosively large. Rather than modeling the action space as a selection between $|A|$ actions, one can leverage the combinatorial nature of the game by modeling action selection as the assignment of an order to each orderable unit. [37]. This decomposes the process of choosing an action from choosing one of $\prod_{i=1}^{N} |O_i|$ actions to choosing one of $|O_i|$ orders $N$ times.

Order assignment is modeled as sequence; An action $\vec{a}$ is a sequence of order indexes, where the $i$-th element $o_i \in O_i$ is the order for unit $i$:

$$\vec{a} = [o_1 \in O_1, o_2, \in O_2, \ldots, o_N \in O_N]$$

The action space consists of all such sequences, and can be written as the cartesian product of $O_i$ for each unit:

$$A = O_1 \times O_2 \times \ldots \times O_N$$

Where $\vec{a} \in A$.

### About Joint Actions

This section has so far overlooked the fact that action selection in Diplomacy is simultaneous; Given $P$ powers denoted $p \in \{\text{p1}, \text{p2}, \ldots, \text{pP}\}$, each with a set of actions $A_p$ to choose from, each power $p$ secretly chooses an action $\vec{a_p} \in A_p$. The combination of action selections per power is called a joint action. A joint action is a vector of actions per power (a vector of vectors) and is denoted:

$$\vec{a_{\text{joint}}} = [\vec{a_{\text{p1}}}, \vec{a_{\text{p2}}}, \ldots, \vec{a_{\text{pP}}}]$$

The joint action space is the cartesian product of each power's action space, and is denoted:

$$A_{\text{joint}} = A_{\text{p1}} \times A_{\text{p2}} \times \ldots \times A_{\text{pP}}$$

When constructing actions for multiple powers, we need a vector of legal orders per unit per power. This is denoted:

$$O_{\text{joint}} = [O_{\text{p1}}, O_{\text{p2}}, \ldots, O_{\text{pP}}]$$

For notational convenience, the subscript $p$ was omitted earlier in this section. For some arbitrary power $p$, the set of actions $A_p$ was denoted $A$, and the vector of legal orders per unit $O_p$ was denoted $O$. This shortcut is used throughout the thesis, and the meaning should always be clear from context. To avoid confusion between joint actions and actions by a single power, the term "action" is always used to refer to an action by a single power.

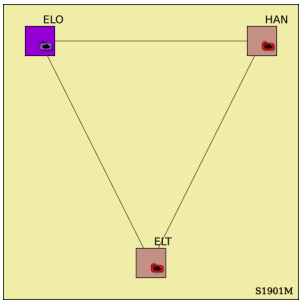
### Representation

The space of legal orders for a unit in any single game state is only a small subset of the space of all orders; i.e. $O_i$ is much smaller than $|\Omega|$. Conversely, the space of legal actions in any single game state becomes only a small subset of the space of all actions. Therefore, legal orders per orderable unit are supplied in the state representation alongside the board state representation of section 4.4.1. This allows for selection between legal actions only, reducing the complexity of action selection.

Because of simultaneous action selection, the Diplomacy agent implemented for this thesis has to reason about the actions of all powers. Therefore, the space of legal actions $O_{\text{joint}}$ is supplied to the agent along with the board state representation **B**.

### Example

An example of the action space representation for one power is given in figure 4.2.



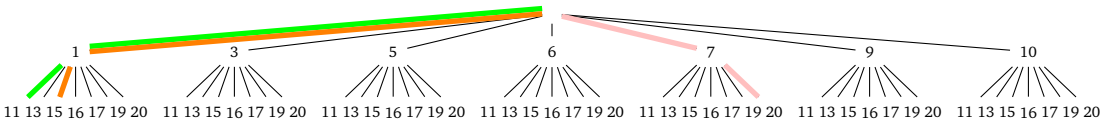

**(a)** Example game state.

| $k$ | $\Omega_k$ |
|---|---|
| 1 | A ELT H |
| 3 | A ELT - HAN |
| 5 | A ELT S A HAN |
| 6 | A ELT S A ELO - HAN |
| 7 | A ELT - ELO |
| 9 | A ELT S A ELO |
| 10 | A ELT S A HAN - ELO |

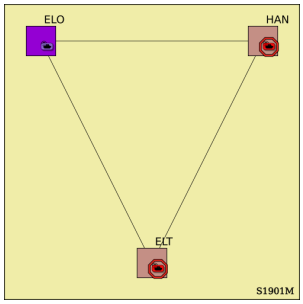
**(b)** Legal orders for unit at ELT.

| $k$ | $\Omega_k$ |
|---|---|
| 11 | A HAN H |
| 13 | A HAN - ELT |
| 15 | A HAN S A ELT |
| 16 | A HAN S A ELO - ELT |
| 17 | A HAN - ELO |
| 19 | A HAN S A ELO |
| 20 | A HAN S A ELT - ELO |

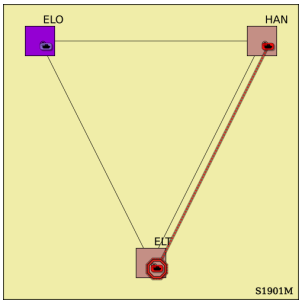
**(c)** Legal orders for unit at HAN.



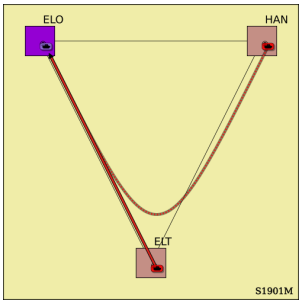

**(d)** Tree of order assignments.




**(e)** [A ELT H, A HAN H]




**(f)** [A ELT H, A HAN S A ELT]




**(g)** [A ELT - ELO, A HAN S A ELT - ELO]

**Figure 4.2:** Diplomacy action space representation visualized. In figure 4.2a, the red power has two orderable units in a movement phase, located at ELT and HAN. The legal orders for each are listed in tables 4.2b and 4.2c. Considering the unit at ELT as unit 1 and the unit at HAN as unit 2, $O$ becomes:

$$\begin{aligned} O &= [O_1, O_2] = [\{o_1^1, o_1^2, o_1^2, o_1^3, o_1^4, o_1^5, o_1^6\}, \{o_2^1, o_2^2, o_2^2, o_2^3, o_2^4, o_2^5, o_2^6\}] \\ &= [\{1,3,5,6,7,9,10\}, \{11,13,15,16,17,19,20\}] \end{aligned}$$

An action is a sequence of orders per orderable unit $[o_1, o_2]$ where $o_i \in O_i$. There are $|A| = |O_1| \cdot |O_2| = 7 \cdot 7 = 49$ legal actions. Figure 4.2d shows a tree of possible order assignments. Three actions are highlighted in color: $[1,11]$, $[1,15]$ and $[7,20]$. The highlighted actions are visualized in figures 4.2e, 4.2f and 4.2g; Given an action as a sequence of order indexes $[o_1, o_2]$, an action is constructed as $[\Omega_{o_1}, \Omega_{o_2}]$, referring to the global order list $\Omega$ of appendix A.

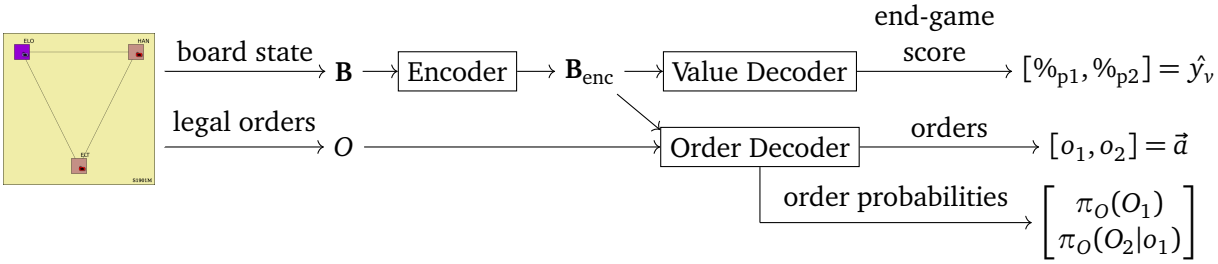



**Figure 4.3:** Overview of neural network architecture. The network takes the board state **B** and legal orders per unit $O$ as input (section 4.4.3), and produces a prediction $\hat{y}_v$ of end-game score in the form of a vector of SoS score per power (here denoted $\%_p$ for power $p$), an action $\vec{a}$ in the form of a sequence of orders $[o_1, o_2]$, and the probability distributions $[\pi_O(O_1), \pi_O(O_2|o_1)]$ from which $o_1$ and $o_2$ were sampled. The encoder creates an encoding of each region as $\mathbf{B}_{\text{enc}}$ based on **B**. The value decoder takes this encoding and returns $\hat{y}_v$. The order decoder uses the encoding in combination with $O$ to produce the sequence of orders $[o_1, o_2]$ and the probability distributions $[\pi_O(O_1), \pi_O(O_2|o_1)]$.

### 4.4.3 Summary of Game Representation

Sections 4.4.1 and 4.4.2 presented the chosen game representation. At every turn in the game, the agent is supplied:

- A representation of board state **B**.
- A representation of the legal orders per unit per power $O_{\text{joint}}$.

Using **B** and $O_{\text{joint}}$, an agent playing power $p$ can construct a legal action $\vec{a_p} \in A_p$ that is grounded in the state of the game. An agent is also free to construct actions for other powers, allowing for exploration of joint actions.

## 4.5 Neural Network Architecture

Deep Reinforcement Learning requires a differentiable model at the core of decision-making. This section describes how neural networks are utilized to construct a model that can produce actions and state values given a description of game state.

The neural network uses the Encoder-Decoder architecture with two decoding "heads": One for unit orders, the other for state value. Like Paquette *et al.* [37], Gray *et al.* [17] and Hatlø [20] and unlike Bakhtin *et al.* [3], Jacob *et al.* [23], Bakhtin *et al.* [4] and FAIR *et al.* [11], the two heads are attached to the same encoder.

Figure 4.3 gives an overview of the neural network architecture. An example forward pass of the neural network is found in appendix C.

### 4.5.1 Encoder

The task of the encoder part of the network is to produce a compressed representation (an "encoding") of each region on the board that reflects the state of neighboring regions. The encoding is denoted $\mathbf{B}_{\text{enc}} \in \{0,1\}^{R \times E}$, where $R$ is the number of regions on the board, and $E$ is the size of the encoding per region. In the decoder heads, $\mathbf{B}_{\text{enc}}$ is used to make decisions based on the state of the region and its neighboring environment.

Fully-connected feed-forward networks discussed in section 2.2.2 fail to take advantage of region adjacencies, and is not suitable. As a motivating example, the go agent AlphaGo Zero of Silver *et al.* [46] takes advantage of adjacencies by utilizing convolutional neural networks, made possible due to the game map resembling a grid. Diplomacy maps, however, do not conform to grids, and are best modelled as arbitrary graph structures. Graph convolutional networks are used (section 2.2.3), which apply a convolution-like operation over arbitrary graph structures.

The encoder takes as input the board state space representation $\mathbf{B}$ described in 4.4.1, and produces as its output $\mathbf{B}_{\text{enc}}$ an encoding for each region on the board. The encoder consists of enough GCN layers for information to propagate from any node to any other through adjacencies. The classic map requires 8 layers [37]. The variants used in this thesis require far less.

Note that $\mathbf{B}$ does not include information on region adjacency. Instead, the encoder is fed the adjacency matrix from the map description during initialization of the network.

The GCN implementation of Gray *et al.* [17] is used.

### 4.5.2 Order Decoder

The task of the order decoder is to produce an action $\vec{a} = [o_1, o_2, \ldots]$ as a sequence of orders given region encodings $\mathbf{B}_{\text{enc}}$ and the space of orders $O$. The order decoder always operates from the point of view of a single power, and has to be queried once per power[7] to obtain actions for all powers. Therefore, this section is written from the point of view of a single power taking an action given the legal orders available to it.

The order decoder is denoted $\pi$. When used to denote probabilities over actions, it is denoted $\pi_A$. When used to denote probabilities over orders for some individual unit, it is denoted $\pi_O$.

#### Principle

The high-level operation of the order decoder is to sequentially produce a probability distribution over legal orders for each unit, and sample an order from this distribution. To achieve unit coordination, the distributions are conditioned on

[7] The operation can be efficiently batched.

orders sampled earlier in the sequence. The probability distribution over legal orders $O_i$ for unit $i$ with sequence number $i$ is denoted as:

$$\pi_O(O_i|o_1, o_2 \ldots, o_{i-1}) \tag{4.1}$$

Sequentially sampling orders for $N$ units can be written as:

$$\begin{aligned} o_1 &\leftarrow \pi_O(O_1) \\ o_2 &\leftarrow \pi_O(O_2|o_1) \\ &\vdots \\ o_N &\leftarrow \pi_O(O_N|o_1, o_2, \ldots, o_{N-1}) \end{aligned} \tag{4.2}$$

Consider an action $\vec{a}$ consisting of orders $[o_1, o_2, \ldots, o_N]$ for $N$ units. The probability of this action being sampled from the order decoder becomes:

$$\pi_A(\vec{a}) = \prod_{i=1}^{N} \pi_O(o_i|o_1, o_2 \ldots, o_{i-1}) \tag{4.3}$$

In addition to producing the action $\vec{a}$, the order decoder also produces the probability distribution over legal orders $O_i$ for each unit $i$:

$$\begin{bmatrix} \pi_O(O_1), \\ \pi_O(O_2|o_1), \\ \vdots, \\ \pi_O(O_N|o_1, o_2, \ldots, o_{N-1}) \end{bmatrix} \tag{4.4}$$

This allows for learning, as loss can be defined with respect to the probability of producing the produced action.

The high-level sequential operation of the order decoder is visualized in figure 4.4.

### Model

A differentiable model of $\pi$ is required for deep reinforcement learning. Using a technique introduced by Paquette *et al.* [37], the order decoder is a custom model that employs a combination of LSTM, embeddings, and tensor operations. The order decoder implementation of Gray *et al.* [17] is used, with minor modifications[8]. The order decoder is visualized in figure 4.5 as a computational graph.

For unit $i$ to be ordered at sequence step $i$, $\pi_O(O_i|o_1, o_2, \ldots, o_{i-1})$ should be grounded in:

1. The environment around the unit.
2. The power that is ordering it.
3. The orders chosen by previously ordered units in the sequence.

[8]The "disband logic" was updated, with very minor effect to practical operation.

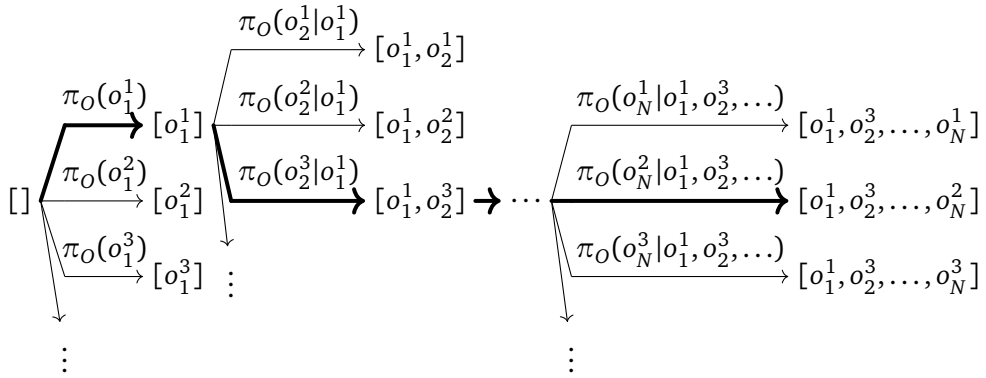



**Figure 4.4:** Principle of order decoder. Each node in the tree represents a (possibly partial) sequence of orders. Each level $i$ of the tree corresponds to the assignment of an order $o_i$ to unit $i$ from $O_i = \{o_i^1, o_i^2, \ldots, o_i^M\}$. Branches are labelled by the probability of choosing each order in $O_i$, conditioned on earlier orders in the sequence. Order assignment is sequential; Starting with an empty sequence at the root, orders assigned one by one by sampling branches according to the labelled probabilities. Leaves of the tree assign an order to each unit in the sequence, and correspond to actions. As per equation 4.3, the probability of producing an action $\vec{a} = [o_1^1, o_2^3, \ldots, o_N^2]$ is the probability of reaching that leaf node in the tree by following the highlighted path:

$$\begin{aligned}\pi_A(\vec{a}) &= \prod_{i=1}^{N} \pi_O(o_i|o_1, o_2, \ldots, o_{i-1}) \\ &= \pi_O(o_1) \cdot \pi_O(o_2|o_1) \cdot \ldots \cdot \pi_O(o_N|o_1, o_2, \ldots) \\ &= \pi_O(o_1^1) \cdot \pi_O(o_2^3|o_1^1) \cdot \ldots \cdot \pi_O(o_N^2|o_1^1, o_2^3, \ldots)\end{aligned}$$

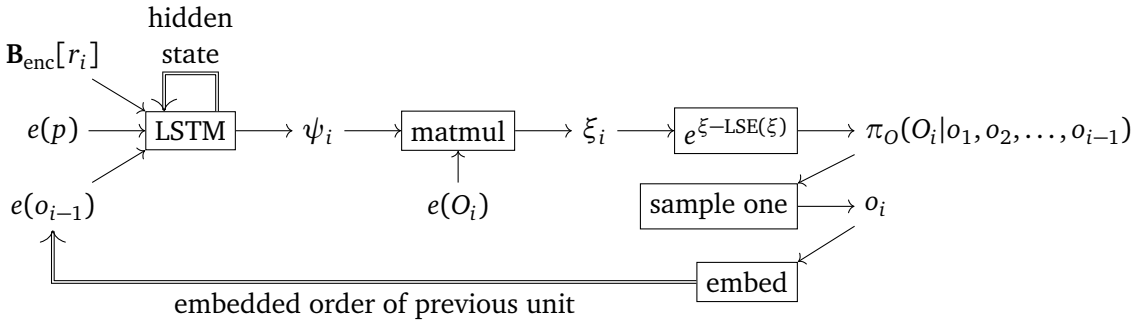



**Figure 4.5:** Overview of order decoder. The network sequentially produces an order for unit $i$ as $o_i$ along with the probability distribution from which it was sampled $\pi_O(O_i|o_1, o_2, \ldots o_{i-1})$, given board state encodings $\mathbf{B}_{\text{enc}}$, the region of unit $i$ denoted as $r_i$, embeddings denoted as $e(\cdot)$, the legal orders for unit $i$ denoted as $O_i$, and the power issuing the order denoted as $p$. Matrix multiplication between the output of LSTM and embeddings for legal orders produces unnormalized logits $\xi_i$. The operation $e^{\xi-\text{LSE}(\xi)}$ transforms the unnormalized logits in the logarithmic scale to normalized probabilities in the linear scale, producing a (conditional) probability distribution over legal orders $O_i$. $o_i$ is sampled from this distribution, and its embedding is used as input to the LSTM for the next unit in the sequence. Double arrows indicate recurrent connections: hidden state of the LSTM, and the embedding of the previously issued order. During training, this computational graph is unrolled as shown in figure 2.4.

This dictates the input to the LSTM: 1. is addressed by providing the region the unit to be ordered exists in: $\mathbf{B}_{\text{enc}}[r_i]$, denoting the region of unit $i$ as $r_i$. 2. is addressed by providing an embedding of the power $p$ ordering the unit, denoted as $e(p)$. 3. is addressed by providing an embedding of the order $o_{i-1}$ chosen for the previous unit in the sequence, denoted as $e(o_{i-1})$[9]. 3. is also addressed by the LSTM having the ability to remember earlier parts of the sequence through hidden state. Omitting hidden state, the output of the LSTM for unit $i$ is denoted $\psi_i$.

$$\psi_i = \text{LSTM}(\mathbf{B}_{\text{enc}}[r_i], e(p), e(o_{i-1})) \tag{4.5}$$

To achieve a probability distribution over legal orders, LSTM output $\psi_i$ is combined with embeddings of legal orders. With an embedding size of $X$, embeddings of legal orders are denoted $e(O_i) \in \mathbb{R}^{|O_i| \times X}$. [10] Making the size of LSTM output equal to the size of order embeddings, $\psi_i \in \mathbb{R}^X$, allows for matrix multiplication between the two, creating a vector $\xi \in \mathbb{R}^{|O_i|}$.

$$\xi_i = e(O_i) \cdot \psi_i \tag{4.6}$$

[9]For the first unit, there is no previous order. In this case, $e(o_{i-1})$ is substituted with a vector of all zeros with the same size.

[10]With $O_i$ is a set of elements $\{o_i^1, o_i^2, \ldots, o_i^{M_i}\}$ and an embedding size of $X$, $e(O_i)$ should be interpreted as constructing a matrix where the $j$-th row consists of $e(o_i^j)$.

$\xi_i$ is interpreted as the unnormalized "logits" of a probability distribution over $O_i$. $\xi_i$ is normalized in the logarithmic scale by subtracting each element by the "log-sum-exp" (LSE) of itself. Denoted as $\xi_i^{normalized}$, the normalized logits are defined as:

$$\text{LSE}(\xi) = \log(\sum_{j=1}^{|\xi|} e^{\xi[j]}) \tag{4.7}$$

$$\xi_i^{normalized} = \xi_i - LSE(\xi_i) \tag{4.8}$$

The normalized form of the distribution in the logarithmic scale is used throughout code, as many calculations are less computationally expensive in the logarithmic scale. For the purpose of explanation, however, let's pretend that probabilities in the linear scale are used. We can obtain probabilities in the linear scale by simply taking the exponent of each element of $\xi_i^{normalized}$. These probabilities form the conditional probability distribution over legal orders:

$$\pi_O(O_i|o_1, o_2 \ldots, o_{i-1}) = e^{\xi_i^{normalized}} \tag{4.9}$$

### 4.5.3 Value Decoder

The task of the value decoder is to produce a prediction of the end-game sum-of-squares score of every power given region encodings. A simple multi-layer fully-connected feed-forward network is used, taking flattened region encodings as input and producing a distribution summing to one at the final layer using a SoftMax layer.

## 4.6 Agent Implementation

Like many recent successes in Diplomacy AI [1, 17, 3, 23, 4, 11], this thesis chooses to train an agent using generalized policy iteration (GPI, section 2.4.3) with game-theoretic search as the improvement operator. Two classes of GPI exist for Diplomacy: Best Response Policy Iteration introduced by Anthony *et al.* [1] (section 3.4.2), and Deep Nash Value Iteration (DNVI) introduced by Bakhtin *et al.* [3] (Presented in section 3.4.4). DNVI has seen the most success, and inspires the GPI implementation for this thesis.

Bakhtin *et al.* [3] also proposes Double Oracle Reinforcement Learning for Action Exploration (DORA), presented in section 3.4.4. DORA extends DNVI with a Double Oracle-like (Section 2.5.2) procedure to enable learning from scratch without human data. Since little or no human data exists for Diplomacy game variants, learning from scratch must be assumed. Therefore, DORA is the primary inspiration for the GPI implementation in this thesis.

The agent selects an action by playing its part in an approximated Nash Equilibrium calculated with Regret Matching search. Because this search is intractable

for large action spaces, only a subset of the full action space for each power can be considered. The agent therefore calculates for each power a subset of "plausible" actions. The set of plausible actions is found by using the order decoder of section 4.5.2 as an "action proposal network". The order decoder is trained over time to propose actions that are "plausibly" good, hence the name. RM is performed in the stage game formed by each power only considering the subset of plausible actions, and payoff of joint actions being calculated as a predicted end-game score of the successor state. The end-game score prediction is obtained using the value decoder of section 4.5.3.

The following sections detail each step involved in action selection: selecting plausible actions, augmenting plausible actions with action exploration, and RM search over plausible actions. Finally, an example is given in figure 4.2.

### 4.6.1 Plausible Actions

To construct a set of $K$ plausible actions per power, the order decoder is queried a large number of times to produce action samples. Then, the sampled actions are sorted by number of occurrences. The top $K$ most occurring actions for each power form the plausible actions set for that power.

The plausible action set for power $p$ is denoted $A_p^* \subseteq A_p$ and has a maximum size of $K$: $|A_p^*| \leq K$. The joint action space formed by each power $p \in \{p1, p2, \ldots, pP\}$ only considering plausible actions $A_p^*$ is denoted:

$$A_{\text{joint}}^* = A_{p1}^* \times A_{p2}^* \times \ldots \times A_{pP}^*$$

### 4.6.2 Search

Regret Matching (Section 2.5.1) is conducted on the stage game formed by considering the subset of plausible actions per power and payoff defined through 1-step rollout of a joint action. Rollout of a joint action is performed by stepping the game engine to the resulting successor state. If the successor is a terminal state, the payoff of the joint action is the end-game score at the terminal state. Otherwise, payoff is the predicted end-game score queried from the value decoder.

RM produces a joint strategy $\sigma$ over the joint action space of plausible actions $A_{\text{joint}}^*$. The probability distribution over plausible actions $A_p^*$ for power $p$ under the joint strategy $\sigma$ is denoted $\sigma(A_p^*)$. For a plausible action $\vec{a_p} \in A_p^*$, the probability of playing that action under $\sigma$ is denoted $\sigma(\vec{a_p})$.

### 4.6.3 Action Exploration

When learning from scratch, the order decoder initially produces actions according to an arbitrary probability distribution. Sampling actions thus does not form a "plausible" set of actions, but instead an arbitrary subset. The order decoder can only be trained on actions that have been sampled, leading to bias toward

sampled actions. This can hinder learning over time, as exploratory actions not already sampled by the order decoder become unlikely.

To address this issue, a procedure inspired by Double Oracle (section 2.5.2) is utilized to augment the set of plausible actions produced by the order decoder:

1. A joint strategy $\sigma$ is found through search over $A^*_{\text{joint}}$ plausible actions with RM.
2. Then, for each power:
    a. A subset of candidate actions $A^c_p$ outside the plausible action subset is sampled randomly from the action space $A_p$, such that $A^c_p \cap A^*_p = \phi$ and $|A^c_p| < |A^*_p|$. [11]
    b. The expected payoff of unilaterally deviating from $\sigma$ with each candidate action $a^c_p \in A^c_p$ is calculated.
    c. The expected payoff of playing by $\sigma$ is calculated.
    d. If unilaterally deviating with a candidate action exploits $\sigma$ by yielding higher expected payoff than playing by $\sigma$, the candidate that maximally exploits $\sigma$ is added to the plausible action set for that power.

This process is repeated either until no candidate actions exploit $\sigma$, or a maximum number of iterations is reached.

The process is computationally expensive, and its usefulness decreases as the order decoder starts producing better actions. Therefore, action exploration is only used while training (except for the pre-training stage discussed in section 4.8.2). Action exploration is not used in tournament games.

### 4.6.4 Example

An example of how the agent chooses an action given the game state shown in figure 4.2 is given in table 4.2.

## 4.7 Training Loop

The agent learns by adapting its neural network to alter what actions are proposed by the order decoder, and what state value is predicted by the value decoder.

The training loop consists of alternating data generation and learning phases, where the data generation phase gathers data for learning through self-play, and the learning phase adapts the network based on generated data. Neural network checkpoints are used to synchronize weights between the two phases, and training data is stored in a "replay buffer". The training loop is illustrated in figure 4.6 and detailed in the following sections.

[11] Bakhtin *et al.* [3] generates candidates by mutating plausible actions. This thesis instead just randomly samples candidates, since this has proven to work well.

| **p** | **Actions** | **Sampled #** | $\in A_p^*$? | $\sigma(A_p^*)$ | $\pi_A(A_p^*)$ |
|---|---|---|---|---|---|
| p1 | $[1, 11]$ | 50 | y (top $K$) | 0% | 60% |
| | $[1, 15]$ | 25 | y (top $K$) | 0% | 25% |
| | $[3, 13]$ | 20 | n | | |
| | $[7, 16]$ | 5 | n | | |
| | $[7, 20]$ | | y (DO) | 100% | 1% |
| p2 | $[21]$ | 70 | y (top $K$) | 50% | 45% |
| | $[23]$ | 25 | y (top $K$) | 50% | 30% |
| | $[25]$ | 5 | n | | |

**Table 4.2:** Example of agent operation and training given game state in figure 4.2. The red power is denoted $p1$, the purple power is denoted $p2$. The agent is playing $p1$.

The agent starts by sampling 100 actions from the order decoder for each power. The number of occurrences of each sampled action is listed in the **Sampled #** column. The top $K = 2$ most occurring actions are included into the plausible action set $A_p^*$. The agent then performs one round of action exploration to augment the plausible actions, and discovers the action $[7, 20]$ for $p1$. This action is added to $A_p^*$. The $\in A_p^*$? column lists whether an action is included in the plausible action set or not, and if yes (y), the reason for its inclusion (top $K$ while sampling, or action exploration (DO)). Next, the agent performs RM search over the joint plausible action space by querying the value decoder for next-state values (not shown), and produces a joint strategy $\sigma$ approximating a Nash Equilibrium. The probability of each plausible action under $\sigma$ is shown in the $\sigma(A_p^*)$ column. The agent plays its part in the equilibrium, and samples $[7, 20]$ with 100% probability, since it conquers ELO (See figure 4.2g).

Notice that $[1, 11]$ and $[1, 15]$ are (correctly) assigned 0% probability by $\sigma$, since they do not conquer ELO (See figures 4.2e and 4.2f). Nothing $p2$ can do can stop $p1$ from conquering ELO, and it therefore has no preference between its two plausible actions.

In this example, one power has actions available to it that are obvious choices given the cramped environment. More complex game states are less clear cut, and this is where DORA shines.

The $\pi_A(A_p^*)$ column contains the teacher-forced action probabilities of each plausible action. For the order decoder, the goal of learning is to minimize distance between $\pi_A(A_p^*)$ and $\sigma(A_p^*)$. This is elaborated in section 4.7.3. Notice that the fit of the order decoder appears particularly bad here, as high probability is assigned to actions considered bad by $\sigma$ ($[1, 11]$ and $[1, 15]$), and little probability is assigned to the action considered good by $\sigma$ ($[7, 20]$). Notice also that the column $\pi_A(A_p^*)$ does not sum to 1.

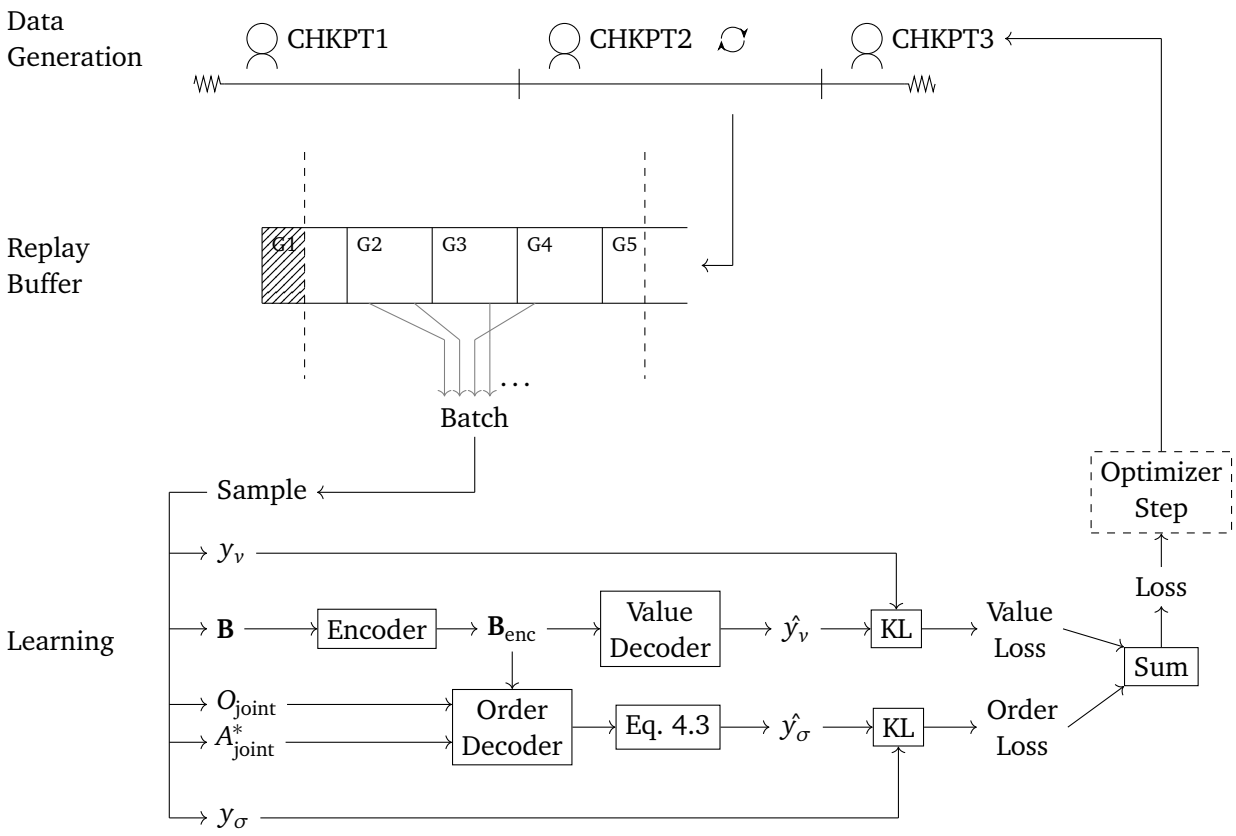



**Figure 4.6:** Overview of training loop. The neural network at checkpoint two (CHKPT2) is being used to generate the fifth game (G5) into the replay buffer through self-play, causing the earliest game states of game one (G1) to be discarded. The replay buffer consists of samples from five games (G1, G2, G3, G4, G5), three of which were created by an earlier neural network checkpoint CHKPT1 (G1, G2, G3), and two of which were generated by the current checkpoint CHKPT2 (G4, G5). In the learning phase, a batch of samples is sampled from the replay buffer, each consisting of $y_v, \mathbf{B}, O_{\text{joint}}, A^*_{\text{joint}}, y_\sigma$. The illustration shows how a sample is processed by the neural network to produce $\hat{y_v}$ (predicted end-game scores) and $\hat{y_\sigma}$ (action probabilities over plausible actions $A^*_{\text{joint}}$). $\hat{y_\sigma}$ is produced through the application of equation 4.3 to the conditional probability distributions produced by the order decoder (See figure 4.3), in combination with teacher forcing using the plausible action subset. The two outputs are compared with their target counterparts with KL-Divergence to calculate loss. The optimizer steps the network to minimize loss, eventually producing a new checkpoint CHKPT3 to take over for checkpoint CHKPT2. Data generation and learning are visualized as simultaneous processes, but in reality, they alternate.

### 4.7.1 Checkpoints and Replay Buffer

A checkpoint of neural network weights is stored after every learning phase. This enables the data generation phase to load the resulting network of the learning phase, and enables spinning up the agent from any iteration of training at a later point.

Training data is stored in and retrieved from a replay buffer of limited size. When the replay buffer is full, old datums are discarded. Data in the replay buffer spans multiple neural network checkpoints, enabling larger batch sizes for learning. It is also intended to make learning more resilient, as the training data is bound to have high variance.

### 4.7.2 Data Generation Phase

In the data generation phase, the agent plays games against itself using the most recent checkpoint of the neural network. The agent controls all powers in the game. Each time an agent selects a (joint) action, the following datums are stored together to form a sample for learning:

- **B**: Representation of board state.
- $O_{\text{joint}}$: Representation of legal orders for each power.
- $A^*_{\text{joint}}$: Plausible actions for each power.
- $y_\sigma$: The joint strategy over plausible actions produced by search.
- $y_v$: An updated estimate of the value of the current state.

Section 4.4.1 details **B** and $O_{\text{joint}}$. Section 4.6 details how the agent acts by producing plausible actions $A^*_{\text{joint}}$ and conducting game-theoretic search to find $\sigma$. $y_\sigma$ is $\sigma$, renamed in this context to make clear that it is used as a target for learning. $y_v$ is not a direct result of regular agent operation, and can be computed in one of two ways: As the expected state value when playing by $\sigma$, or as the discounted end-game score. The former is more proper, and is used for the majority of training. The latter is beneficial during early stages of training to quickly achieve a reasonable value network, and is used for the pre-training stage discussed in section 4.8.2.

The agent chooses an action to play $\epsilon$-greedily with respect to $\sigma$. This means the agent is most likely to gain experience in valuable states while also having some probability of visiting less valuable states, thereby balancing between exploration and exploitation. Learning is off-policy since learning targets use $\sigma$ greedily. During pre-training, state value targets are calculated with respect to the discounted end-game score when following the behavioral policy, leading to values based on the $\epsilon$-greedy policy. This is not believed to be an issue.

### 4.7.3 Learning Phase

In the learning phase, the neural network is adapted using the datums in the replay buffer. First, a batch of samples is fetched randomly from the replay buffer.

Then, a forward pass of the neural network is conducted for all samples in the batch. Loss is calculated for the order decoder and value decoder by comparison with $y_\sigma$ and $y_v$. Finally, the neural network weights are adjusted to minimize the sum of order decoder loss and value decoder loss, and a checkpoint is stored.

The following sections detail loss calculation for the order decoder and value decoder.

### Order Decoder Loss Calculation

For a power $p$, the target probability distribution over plausible actions $y_\sigma$ contains the probability for each plausible action $\vec{a_p} \in A_p^*$ as $y_\sigma(\vec{a_p})$. The goal of learning is that the order decoder should more probably produce actions that are probable under $y_\sigma(A_p^*)$. This way, the plausible action subset will contain actions that have been shown to be good through search in the past. Over time, the quality of the plausible action subset should increase.

In addition to producing an action $\vec{a}$ as a sequence of orders $[o_1, o_2, \ldots]$, the order decoder also produces the conditional probability distributions from which the orders were sequentially sampled: $[\pi_O(O_1), \pi_O(O_2|o_1), \ldots]$ (section 4.5.2). As per equation 4.3, the action probability $\pi_A(\vec{a})$ can be computed using the conditional probability distributions and the sequence of sampled orders.

Normal feed-forward operation of the order decoder samples an order at each sequence step, producing a random action. To obtain the probability of some specific action $\vec{a'} = [o'_1, o'_2, \ldots]$, teacher-forcing is used: For unit $i$ at sequence step $i$, $o'_{i-1}$ is injected as the previously sampled order. This way, the returned probability distribution at each sequence step $i$ is conditioned on $[o'_1, o'_2, \ldots, o'_{i-1}]$, and $\pi_A(\vec{a'})$ can be calculated.

By using teacher forcing, the probability of each action in the plausible action set can be calculated, producing $\hat{y_\sigma}$[12]. Loss can then be defined by comparing $y_\sigma$ and $\hat{y_\sigma}$. For this thesis, the KL-Divergence metric is used, which gives a measure of the distance between probability distributions.

Figure 4.2 includes a column illustrating an example comparison between $y_\sigma = \sigma(A_p^*)$ and $\hat{y_\sigma} = \pi_A(A_p^*)$.

### Value Decoder Loss Calculation

A state value prediction $\hat{y_v}$ is produced with the value decoder. The value of a state is a prediction of end-game Sum-of-Squares score, and thus a distribution summing to 1. Similar to the order decoder, loss is calculated as the KL-Divergence between the value target $y_v$ and value prediction $\hat{y_v}$.

[12] If the plausible action set is smaller than the full action space, $\hat{y_\sigma}$ will not sum to 1, and is thus not a proper probability distribution. This is problematic, but has worked well in practice for this thesis.

## 4.8 Experimental Setup

To answer the research questions of section 1.2, several experiments are run. An experiment consists of training an agent for some game variant and running tournaments for comparison with other agents. The following sections detail the three game variants chosen, how agents are trained, and how agents are evaluated.

### 4.8.1 Game Variants

Research question 1 poses a question about the generality of state-of-the-art techniques in map variants. In order to say something about generality, several map variants need to be utilized. Three variants will be considered in this thesis. As will be discussed in section 4.8.3, research question 2 dictates that one of the variants considered be the Pure variant. Since Pure features 7 powers, it is computationally heavy for search-based agents (the joint action space grows large). Therefore, the other two variants are chosen to be 3-power variants. Since Pure is topologically trivial, the two other variants focus on topological challenges. The two other variants are the Lattice variant, featuring a relatively large regular topology, and the Hub variant, featuring a central highly-connected "hub" region. The three variants are visualized in figure 4.7.

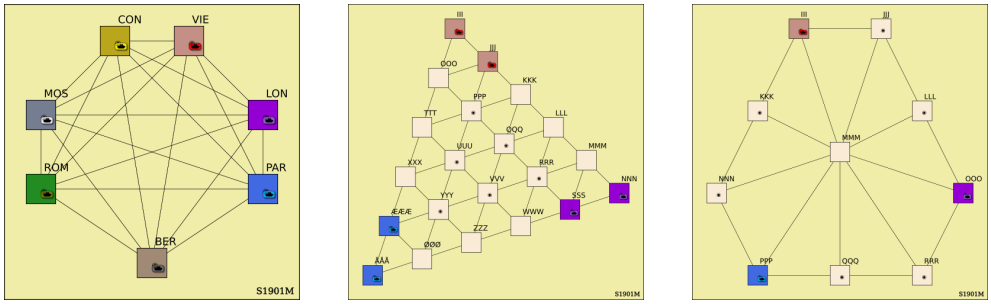
All three chosen game variants happen to be symmetric both in terms of map topology and the starting positions for powers. Typical Diplomacy game variants (including the original game) are asymmetric, mirroring real-world geography. The choice of symmetric maps is deliberate, as it simplifies visual inspection of game state, and can be argued to reduce the number of tournament games needed to evaluate agents: For asymmetric maps, a large number of games is needed to "even out" score differences due to power assignment. For example, Paquette *et al.* [37] reports that Turkey wins twice as many games as Italy in a large dataset of human games[13].

#### Pure Game Variant

Pure [49] is an unofficial, but established, map variant for Diplomacy. The variant is playable on `vdiplomacy.com`, one of the major online Diplomacy platforms, and has existed since at least 1990[14]. The Pure variant features 7 powers and 7 fully-connected regions. Powers have one region each as their home supply center. The game is won by controlling four regions. The maximum number of in-game years is chosen to be 10. The initial board state of the Pure variant is visualized in figure 4.7a.

[13] Table 1 of Paquette *et al.* [37].

[14] A webpage detailing the ruleset of Pure was "last revised" August 9, 1990: `http://uk.diplom.org/pouch/Email/judge/info.pure.html`, accessed 12.05.2023.




**(a)** Pure game variant **(b)** Lattice game variant **(c)** Hub game variant

**Figure 4.7:** Initial board state of the three game variants considered in this thesis.

### Lattice Game Variant

The Lattice map variant features 3 powers and 21 regions laid out in a "triangular lattice" structure, where region neighborhoods follow a regular pattern. Like the classic map, and unlike the Pure variant, not every region on the board features a supply center. The variant features 12 supply centers, and 7 are required to win the game. Each power has two home supply centers, and therefore starts off with two units. The maximum number of in-game years is chosen to be 10. This variant requires an agent to maneuver a large number of units to win the game. The initial board state of the Lattice variant is visualized in figure 4.7b. It was created for this thesis, and is not played by the board game community.

### Hub Game Variant

The Hub map variant features 3 powers, 10 regions and 9 supply centers. 9 regions feature supply centers, and form a circle. The home centers are found on this circle, spaced two supply centers apart. A 10th region without a supply center (the "hub") is found in the middle of the circle, neighboring all other regions. The maximum number of in-game years is chosen to be 10. The initial board state of the Hub variant is visualized in figure 4.7c. It was created for this thesis, and is not played by the board game community.

## 4.8.2 Agent Training

An agent is trained for each of the three game variants. The training of each agent is conducted following a YAML specification file that includes agent hyperparameters, parameters for the training loop, and the game variant specification as per section 4.3. Training is split in two stages, starting with a pre-training stage that aims to achieve reasonable operation quickly, before making full use of the agent implementation of section 4.6.

The following sections detail how agent hyperparameters were selected and the specifics of the pre-training stage.

### Agent Hyperparameter Selection

The agent consists of many moving parts, leading to many tweakable hyperparameters. Since the research questions do not strictly require creating an optimal agent, hyperparameter selection is not considered an important part of this thesis. Hyperparameters are selected based on intuition and related work, as well as brief manual experimentation before starting the training sessions that produce the results of chapter 5. Some key hyperparamters are shown in table 5.1. The full agent hyperparameters for each of the three game variants can be found in the uploaded thesis codebase.

### Pre-Training Stage

The value decoder sits at the core of the training loop, as it informs both value decoder targets given successor states, and order decoder targets through the result of search (which again is based on the value decoder). Since the value decoder normally is trained through the expected bootstrapped next-stated value, only the last non-terminal state is updated with valuable targets in early training. Many iterations are required for that information to propagate throughout the state space. Meanwhile, training data is littered with arbitrary targets that do nothing but confuse learning. To mitigate this, training is split up into two stages, starting with a "pre-training" stage. The pre-training stage is intended to prime the value decoder with meaningful state values, before enabling the full capacity of agent implementation of section 4.6. The pre-training stage is inspired by Bakhtin *et al.* [3], and makes the following modifications to regular operation:

- Value decoder targets are computed using backed-up discounted end-game scores.
- The order decoder is unused, and not trained. Plausible actions are sampled uniformly.[15]
- No action exploration is performed.

Search over (uniformly chosen) plausible actions is performed as usual, using the value decoder for state values.

The pre-training stage is run until the value decoder can be seen to output "reasonable" values near the end-game through manual inspection of visualizations produced throughout training. As an example, in a game state where one power has a clear advantage over the others in the end-game, the value decoder is considered "reasonable" if it predicts this power's end-game score as higher than the others.

[15] Bakhtin *et al.* [3] does train the order decoder during pre-training, based on the result of search over the uniformly chosen plausible action subset. This was omitted from this thesis to simplify interpretation of the loss function while pre-training, and because pre-training seems to yield good results regardless.

### 4.8.3 Agent Evaluation

Agents are evaluated through inspection of skill throughout training. Skill is measured through tournaments with comparison agents. The simplest comparison agent is the "Uniform" agent, that at each turn chooses a random action. Any competent agent should easily beat the Uniform agent, yet it still acts as a cheap and useful method of evaluating skill over time. Agents are also compared against different (sometimes earlier) versions of themselves. For the Pure variant, a comparison with the A2C agent of Hatlø [20] is made, here called HatløA2C. The following two sections detail how tournaments are held to compare agents, and the HatløA2C agent.

#### Tournaments

Since the game of Diplomacy features more than two players, comparing agents against each other is not as simple as playing one-on-one games and recording win probabilities. On the classic 7-player Diplomacy map utilized for all major Diplomacy research, the most common technique for comparing two agents is to play one instance of one agent against six instances of the other, and vice versa. This kind of "one-versus-all" tournament tests whether a population of the six-instance agent can be invaded by the one-instance agent, and hence whether it plays a Evolutionarily Stable Strategy. [1] These settings are referred to as 1v6 and 6v1. For 3-player game variants in this thesis (Lattice and Hub), tournament settings are referred to as 1v2 and 2v1.

In balanced 2-player games like chess, an agent facing a copy of itself should be expected to win 50% of the time. Similarly, in an $N$-player Diplomacy game variant, an agent facing copies of itself achieves an expected Sum of Squares score of $1/N$.[16] This expected score is important when inspecting tournament results for Diplomacy, as scores above $1/N$ indicate a single agent is able to successfully invade the opponent, and scores below indicate the opponent is able to successfully protect against invasion from the single agent.

Importantly, the presence of a wide space of equilibria in Diplomacy [3] means that some pairs of agents will be unable to invade each other, while performing well against less competent agents.

Action exploration is a computationally heavy, and its value diminishes at later iterations of training [3]. Therefore, it is disabled during tournament play. This means search-based agents are restricted to the plausible actions sampled from the order decoder.

[16] A Sum of Squares score of exactly $1/N$ might not be possible in a single game due to the layout of supply centers. However, when playing a large number of games, the average score will converge towards $1/N$. Also, this average score assumes powers are assigned at random, for game variants with asymmetric powers.

### Computational Restrictions During Tournaments (Pure)

Playing games with search-based agents is computationally heavy, and it is hard to play a large number of games. This can become a problem when evaluating an agent throughout training, as running a statistically significant number of tournament games for comparison with a baseline agent could take more time than training itself. The Pure game variant features 7 players, and is badly affected by this problem. Therefore, a compromise is struck for this variant. When evaluating the progression of skill throughout training, a computationally restricted version of the agent (limited search iterations, limited plausible action space) is used. The skill of the restricted agent should be correlated with the skill of the unrestricted agent, and the intuition is that if the restricted agent is improving over time, the unrestricted agent should be as well. This way of evaluating an agent gives useful insight over time, but does not indicate the true power of the agent. Therefore, additional tournaments are held with the final iteration of the agent without restrictions.

To avoid confusion when presenting results in chapter 5, the unrestricted version of the agent is called GPI, and the restricted version is called GPI-R.

### Comparison with HatløA2C (Pure)

Research question 2 requires a policy gradient agent as a benchmark. Several such agents exist, with the most important being the RL version of DipNet of Paquette *et al.* [37]. In later research utilizing GPI, this agent is used as a benchmark [1, 17, 3]. Later research also develop their own Actor-Critic agents building on improvements in neural architecture arising from the development of their GPI agents. Since DipNet RL is the common Actor-Critic benchmark agent in recent research, comparison to it would be optimal for this thesis. However, DipNet RL is written specifically for the classic game map, which is much to large for consideration in this thesis, in addition to falling outside the ruleset modifications discussed in section 4.1. Luckily, Hatlø [20] proposes an Actor-Critic agent based partly on DipNet RL trained on the Pure game variant; one of the game variants chosen for consideration in this thesis. The agent of Hatlø [20] is discussed in Related Work section 3.4.5. Throughout the rest of this thesis, it is referenced as HatløA2C, to distinguish it from its LOLA sibling.

Code and neural network weights for the agent are available online.[17] A thin compatibility layer is implemented to allow the HatløA2C agent to interface with this thesis. No agent code is modified, but one major quirk exists: HatløA2C can output illegal unit orders. The MILA game engine, also utilized by Hatlø [20], is known to gracefully ignore illegal orders, treating them as if no order was submitted. Failing to submit an order triggers a default order. The game engine for this thesis (section 4.2) does not allow illegal actions. Therefore, illegal unit orders by HatløA2C are converted to their equivalent default order under the MILA engine

[17] At `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3024711`, accessed 12.05.2023

(with one exception):

- An illegal order in the movement phase is converted into a Hold order
- The default order when building in the adjustment phase is to opt out of building. This order is not supported in this thesis, on the grounds that building a unit when able is always beneficial. Therefore, an illegal order in the build phase is converted into a random build order. This should only improve the performance of the agent, especially on the small Pure variant.
- An illegal order when removing units in the adjustment phase is turned into a random legal disband order.
- An illegal order in the Retreat phase is turned into a disband order.

An additional quirk is that the agent is trained for a maximum of 10 order phases. The number of in-game years involved in these 10 order phases can vary, but will always be less than the maximum number of in-game years considered for the Pure variant in this thesis. HatløA2C is therefore strictly speaking not trained for the game length considered in this thesis.

## 4.9 Summary of Methodology

In this chapter, a modular reinforcement learning system for Diplomacy based on state-of-the-art research was described.

The system operates on a modified version of the original Diplomacy rules: Coastal and sea areas are omitted, the non-communicative No-Press game variant is assumed, and Sum-of-Squares (SoS) scoring is used to score games reaching a turn limit.

The MILA game engine is utilized, as it is simple to utilize and supports game variants. The game variant creation process is automated through a specification in the YAML file format, enabling game variants to be fully described alongside agent hyperparameters, including configuration of game visualization.

A game representation is defined which enables the agent to interact with the game environment through the formalism of states and actions. The agent perceives the game through a representation of board state. The action space is modeled as sequences of unit orders, and a representation of the space of legal actions is supplied to the agent alongside the board state representation, allowing the agent to choose among legal actions only.

A neural network serves as a differentiable model at the core of decision-making for the agent. The neural network takes as input the board state and space of legal actions, and produces as output a prediction of end-game SoS score, an action as a sequence of orders, and the conditional probability distributions from which the orders were sampled. The neural network uses the Encoder-Decoder architecture with two decoding "heads". A GCN-based encoder produces an encoding of board state. Then, this encoding is processed by the value decoder to produce a prediction of end-game SoS score, and by the LSTM-based order decoder (alongside the space of legal actions) to produce an action and the condi-

tional probability distributions from which it was sampled.

Primarily inspired by Double Oracle Reinforcement Learning for Action Exploration (DORA) of Bakhtin *et al.* [3] (section 3.4.4), an agent is implemented that samples a subset of the action space from a order decoder, optionally augments this subset with action exploration to add reasonable actions, and then performs game-theoretic Regret Matching (RM) search with next-state values from the value decoder to produce a joint strategy over the action subset approximating a Nash Equilibrium. The agent acts by playing its part in the joint strategy.

The agent is trained through self-play on three Diplomacy game variants: Pure, Lattice and Hub. For each turn of the game, a sample for learning is stored consisting of the input to the neural network, an improved estimate of the current state value, and the resulting joint strategy produced by RM. The value estimate is either the expected payoff when playing by the joint strategy from RM and using bootstrapped next-state values from the value decoder, similar to value iteration, or the discounted end-game score. Then, the neural network is trained to imitate the learning targets, improving the value decoder estimate, aand making the order decoder more likely to predict actions considered good by search in the future. Training begins with a pre-training stage that aims to achieve a reasonable fit of the value decoder Agents are evaluated through inspection of skill throughout training, measured through "one-versus-all" tournaments. All agents are compared against the Uniform agent. Additionally, the agent trained for the Pure game variant is compared with the A2C agent of Hatlø [20], dubbed HatløA2C in this thesis.

# Chapter 5

# Results and Analysis

This chapter presents and analyses results gathered in carrying out the methodology of the previous chapter. The chapter opens by introducing the figures and tables used in detailing results. Then, the results for each game variant are presented and analysed one by one. Analysis for each subsequent game variant builds upon earlier analysis. The chapter ends with a summary.

## 5.1 Anatomy of Results

Three agents are trained, one for each game variant. The agents are evaluated through skill in tournament games against other agents. Two kinds of tournament results are presented for each game variant: A skill-progression plot and a comparison table. Both are made in the style of Gray *et al.* [17]. The trained agents are summarized in table 5.1.

In the results, GPI refers to a fully trained agent, GPI-R refers to a computationally restricted version of the fully trained agent (limited search iterations, limited number of plausible actions), and GPI-Pre-iN refers to the agent at pre-training iteration N. The Uniform agent chooses a legal move at random each turn, and HatløA2C is the agent presented in section 4.8.3.

| Variant | Iterations (Pre) | Search Iterations | Plausible Actions |
|---|---|---|---|
| Pure | 1822 (522) | 1024 (GPI-R: 128) | 10 (GPI-R: 3) |
| Lattice | 1200 (800) | 1024 | 5 |
| Hub | 900 (100) | 1024 | 5 |

**Table 5.1:** Summary of trained agents. The Iterations column shows the number pre-training iterations in parenthesis. For Pure, the parameters of the restricted agent are shown in parenthesis.

## 5.2 Presentation and Analysis of Results for the Pure game variant

For the Pure game variant, 1822 iterations of training are performed. The first 522 iterations form the pre-training stage as discussed in section 4.8.2. Training was conducted over a period of 26 days, and tournament games took around a week to complete. 1024 iterations of search are performed at each turn, searching over 10 plausible actions per power.

As outlined in 4.8.3, the large joint action space of the Pure game variant makes evaluating skill through time infeasible. Therefore, a computationally restricted version of the agent is introduced (GPI-R). GPI-R performs only 128 iterations of search, and only searches over 3 plausible actions per power.

The bug discussed in appendix B.1 affects results for the Pure variant (and only the Pure variant), leading retreat orders to become ineffective in many game states. The bug only affects game states where the search-based agent is retreating, meaning games between HatløA2C and Uniform are unaffected. The bug is primarily detrimental to the search-based agent, as most or all of its retreat orders are rendered ineffective. In game states where an opponent agent is retreating simultaneously with a search-based agent, the bug affects the opponent agent as well.

### 5.2.1 Progression of Skill

Figure 5.1 shows the (restricted) agent's progression of skill throughout training, measured through tournaments with HatløA2C and the Uniform agent. The figure shows that the agent improves against both opponents over time, and ends up significantly outperforming them. The agent is able to both invade the opponent when playing as a single agent, and protect against invasion when playing the majority of powers. The agent performs better against the Uniform agent than against HatløA2C, which is to be expected, considering HatløA2C is trained to be competent at the game.

Interestingly, the agent starts off outperforming the Uniform agent already at iteration 0 (with arbitrary network weights), while performance against HatløA2C is expectedly poor. This is likely due to the fact that terminal successor states are evaluated by their true end-game score, rather than predictions from the value decoder. This means that the agent has an advantage in the end-game over the Uniform agent already before any training is performed, since it is making a decision informed by true end-game scores. This advantage is not massive, as several random events have to coincide in order for the agent to capitalize. For example, actions that likely lead to good end-game states have to exist in the plausible action set (the selection is uniform during pre-training, and action exploration is not used), the evaluation of non-terminal successor states have to be (randomly) worse, and the agent has to have randomly ended up in a game state where a good end-game score is possible. The agent has the best advantage when games

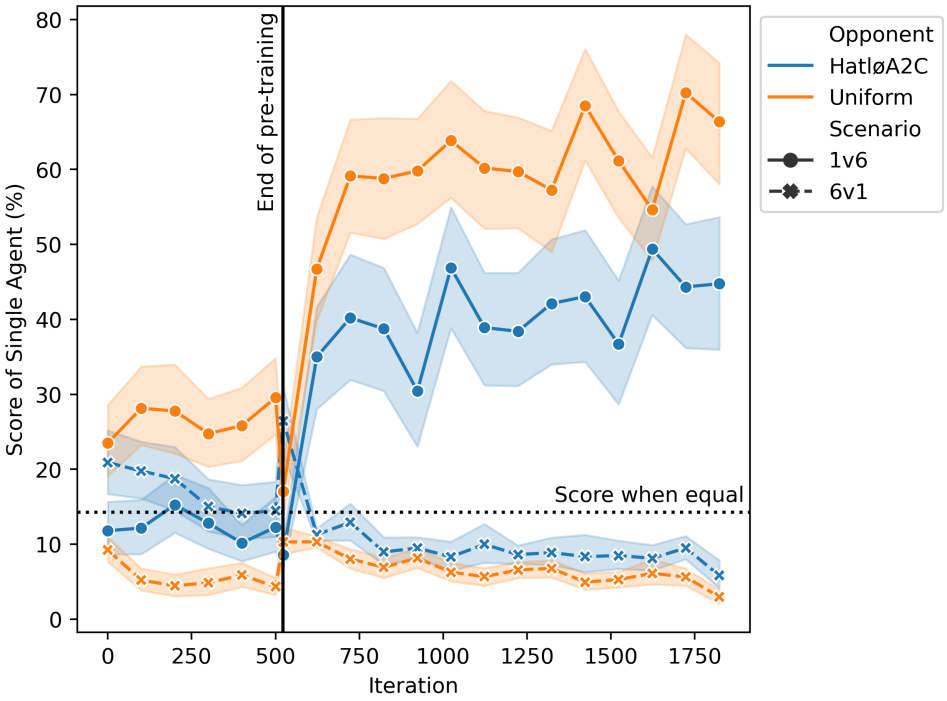



**Figure 5.1:** Skill of (restricted) agent on Pure game variant across training iterations. Each individual graph shows, as a function of time, the average SoS score of a single agent playing 100 games against 6 copies of another, along with the 95% confidence interval. The color of a line indicates the opponent agent being compared against (HatløA2C or Uniform). Solid lines show games with 1 copy of the agent and 6 copies of the opponent. Dashed lines show games with 6 copies of the agent and 1 copy of the opponent. Scores are always from the perspective of the single agent. The solid vertical line indicates the end of the pre-training stage at iteration 522. The dashed horizontal line shows a score of $1/7 \approx 14.3\%$, which is the expected score of a single agent playing against 6 copies of itself.

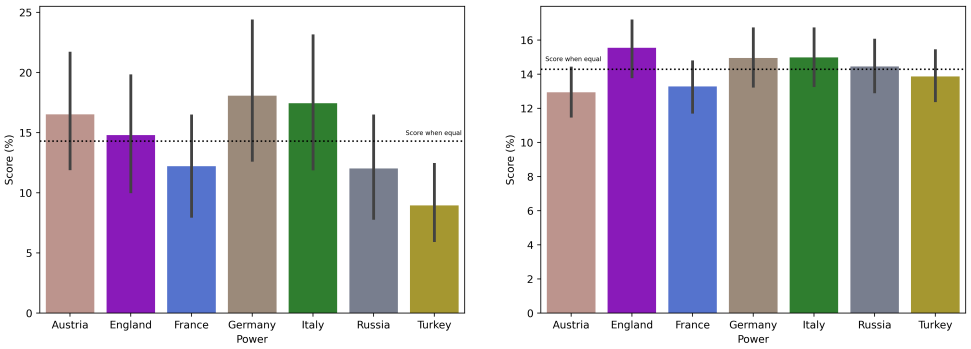
end due to the turn limit, since all successor states when making the last action are terminal, leading the agent to effectively search based on true end-game scores only.

The agent's poor performance against HatløA2C during early iterations is expected, since the agent is only playing slightly better than the Uniform agent at this stage (in the end-game). As training progresses, the invasion score of HatløA2C diminishes to almost exactly $1/7 \approx 14.3\%$, meaning it is not able to invade. The invasion score against HatløA2C stays poor throughout pre-training, and this is likely due to the agents poor understanding of early turns of the game during pre-training.

A large dip in performance can be observed at iteration 523, the first iteration after the pre-training stage. This iteration simply copies neural network weights from the iteration 522, and removes the pre-training stage modifications of section 4.8.2; The order decoder is enabled, action exploration is enabled (for training), and the value decoder starts learning with bootstrapped next-state values. Since no further training has occurred yet, and action exploration is disabled for tournament play, the only agent modification that differentiates the iteration from the last iteration of pre-training is the enablement of the (so far untrained) order decoder. The order decoder must therefore be to blame for the performance dip. The performance dip can be attributed to the fact that the untrained order decoder produces arbitrarily biased subsets of the action space as plausible actions.

Comparing iteration 0 and 523 one can make the interesting observation that an untrained order decoder in combination with a "reasonable" (pre-trained) value decoder seems to perform worse than uniformly choosing plausible actions in combination with an untrained value decoder. The difference is particularly noticeable in the 6v1 scenario against HatløA2C, and the 1v6 scenario against the Uniform agent. The reason for this is unclear, but could be due to the bias in the untrained order decoder being so adverse that the agent consistently fails to consider good actions, and therefore fails to take advantage of both the trained value decoder and terminal scores in the end-game.

After pre-training, performance against both agents quickly rises, with an upwards trend indicating performance could improve even more if training was continued. Large dips in performance are observed throughout training, which can be attributed to several factors. First, since the value decoder and order decoder share a common encoder, learning weights is complex. A good fit for one of the decoders can lead to a worse fit for the other. Second, it is the nature of RL algorithms that the neural network is being fit to a moving target. An update to the value decoder could outdate the fit of the order decoder, and an updated order decoder requires further training of the value decocer. An example scenario where this could lead to worse performance is if an updated value decoder leads search (and action exploration) to recommend a (joint) action that so far has been considered of little value, and therefore leads to a part of the state space that the neural network is undertrained on. Third, although the game variants are symmetric (section 4.8.1), the skill of an agent with the different powers can




**(a)** First 100 games. **(b)** 1000 games.

**Figure 5.2:** Average SoS score per power in tournaments with 1x GPI playing 6x Uniform on the Pure game variant, plotted with the 95% confidence interval. 5.2a shows the result after playing 100 games (the games behind the entry in table 5.2). 5.2b shows the result after playing 1000 games. Notice that the scores deviate more after 100 games than after 1000, and that the score $1/7 \approx 14.3\%$ (horizontal line) is within confidence interval for all powers after 1000 games.

vary due to the nature of RL. This can decrease accuracy in the results as the number of games played can become insufficient for particularly skewed agents. As an example, 100 tournament games with random power assignments is more appropriate for measuring the skill of an agent that is equally skilled with all 7 powers, than for a skewed agent that is very skilled with one power, and poor with the others. For roughly $100 * 1/7 \approx 14$ games, the skewed agent will be playing competently. For the rest of the games, it will be playing poorly. In comparison, a non-skewed agent will be equally competent regardless of the power it happens to play, and should yield more consistent tournament scores. As shown in figure 5.2b and expanded upon in the next section, the fully trained GPI agent does not seem to be significantly skewed. It is therefore not believed to be a major contributing factor to the dips in performance.

### 5.2.2 Skill of Final Iteration

Table 5.2 shows the result of tournaments between the agent at its final iteration in both unrestricted and restricted form (GPI and GPI-R), the Uniform agent, and HatløA2C.

Consider first tournaments held between HatløA2C and the Uniform agent. Hatlø [20] reports[1] an average score of 22.6% (stddev 18.6%) when a single HatløA2C agent plays six Uniform agents. Conversely, he reports an average score of 6.7% (stddev 8.1%) for a single Uniform agent playing six copies of HatløA2C. Cross-referencing these scores with the results in table 5.2, one can observe that

[1]The scores are extracted from table 5.1, showing the average end-game "reward". The reward signal is Sum-of-Squares scoring, matching the scoring mechanism used in this thesis.

| 1x ↓ vs 6x → | **Uniform** | **HatløA2C** | **GPI-R** | **GPI** |
|---|---|---|---|---|
| **Uniform** | - | 9.0%±1.1% | 3.0%±0.5% | 6.6%±0.8% |
| **HatløA2C** | 33.9%±2.5% | - | 5.8%±1.0% | 7.1%±0.8% |
| **GPI-R** | 66.4%±4.1% | 44.8%±4.5% | - | 9.6%±1.3% |
| **GPI** | 57.0%±4.0% | 33.4%±3.9% | 10.5%±1.6% | - |

**Table 5.2:** Average SoS scores of agents playing 100 games against 6 copies of another agent on the Pure game variant, along with standard error (after ±). An agent playing against 6 copies of itself scores an average of $1/7 \approx 14.3\%$, and such comparisons are therefore omitted ("-"). The comparisons of GPI-R with HatløA2C and the Uniform agent correspond to the final iteration of figure 5.1.

HatløA2C scores significantly better against 6x Uniform agents, and that the Uniform agent also scores slightly better against 6x HatløA2C. This disparity can be attributed to the fact that agents are forced to select legal actions in this thesis' game environment, which should improve the performance of both agents: The Uniform agent is more likely to choose an action with impact, and less likely to choose an action that reduces to a default (no-op) order, and illegal orders of HatløA2C in the build phase are converted to legal build orders in situations where it otherwise would have chosen not to build (a "waive" order, see section 4.8.3), which is better a vast majority of the time. It can be concluded from this comparison with the scores of Hatlø [20] that HatløA2C performs better here than in its original context.

Both GPI and GPI-R clearly outperform the Uniform agent and HatløA2c in all respects: When playing as the 1x agent, they obtain scores far higher than $1/7 \approx 14.3\%$, and when playing the 6x agent, their opponent scores far lower.

A surprising result is that GPI-R scores better than GPI against both HatløA2C and the Uniform agent in all respects. Given more computational headroom, one would imagine that GPI would be the more competent of the two. This result could be due to the restricted plausible action space of GPI-R turning out to be a benefit when playing less competent agents: When playing against the Uniform agent, the game-theoretic assumption that opponents play competently by the same equilibrium does not hold. This is true to a lesser extent for HatløA2C as well. A smaller plausible action space reduces the considerations the search operator has to take with respect to opponents actions, leading to more greedy plays, which might be beneficial when faced with an opponent that takes its actions following a completely different model.

Neither GPI nor GPI-R scores above $1/7 \approx 14.3\%$ when playing six copies of the other. GPI does obtain a higher invasion score against GPI-R than vice versa, however. Because both GPI and GPI-R can be said to be competent agents based on comparison with the Uniform agent and HatløA2C, the fact that neither is able to invade the other can be attributed to the finding of Bakhtin *et al.* [3] that multiple equilibria exist for the game of Diplomacy. To significantly outperform GPI and GPI-R as implemented, one would therefore have to make use of regularized

search as introduced by Jacob *et al.* [23].

Figure 5.2 visualizes the average score of each power in tournaments of a single GPI agent playing 6 copies of the Uniform agent. Measuring the relative skill of an agent with each power intuitively takes a larger number of games than measuring the skill of the agent playing an arbitrary power. Therefore, the visualization is made first using the 100 tournament games behind the entry in table 5.2 in figure 5.2a, then with 1000 tournament games in figure 5.2b. Figure 5.2b indicates that the agent is somewhat skewed. Since the Pure game variant (like Lattice and Hub) is symmetric, this can only be due to the operation of the trained agent. Therefore, the figure is evidence that RL can produce an agent more skilled with some powers than others.

## 5.3 Presentation and Analysis of Results for the Lattice Game Variant

For the Lattice game variant, 1200 iterations of training are performed. The first 800 iterations form the pre-training stage as discussed in section 4.8.2. Training was conducted over a period of 2 days, and tournament games took around a day to complete. 1024 iterations of search are performed at each turn, searching over 5 plausible actions per power.

In addition to the Uniform agent, comparisons are made with the agent at the final iteration of pre-training (iteration 800), referred to as GPI-Pre-i800.

### 5.3.1 Progression of Skill

Figure 5.3 shows the agent's progression of skill throughout training, measured through tournaments with the Uniform agent and GPI-Pre-i800.

Several observations made in analysing results for the Pure game variant in the previous section apply here as well. The agent outperforms the Uniform agent already at iteration 0, and there is a dip in performance at the first iteration after pre-training, after which the agent quickly outperforms the pre-training stage.

In this variant, the agent reaches close to a 100% score against the Uniform agent after only about 900 iterations. In comparison, the agent achieves its maximum measured average score of about 70% against the Uniform agent in the Pure variant after about 1700 iterations (see figure 5.1). This comparison must be seen in context: The Pure game variant features 7 players, whereas the Lattice variant only features 3. The high player count of the Pure variant means the action of a single agent has a lesser influence on the game. A high player count also puts harder requirements on the agent to properly model its opponent, and allows for complex relationships to form. The map topologies should also be considered. The Pure variant features a small number of fully-connected regions, leading to a cramped map. The Lattice variant features a smaller number of regions connected in a more complex structure. Intuitively, it makes sense that a search-based agent

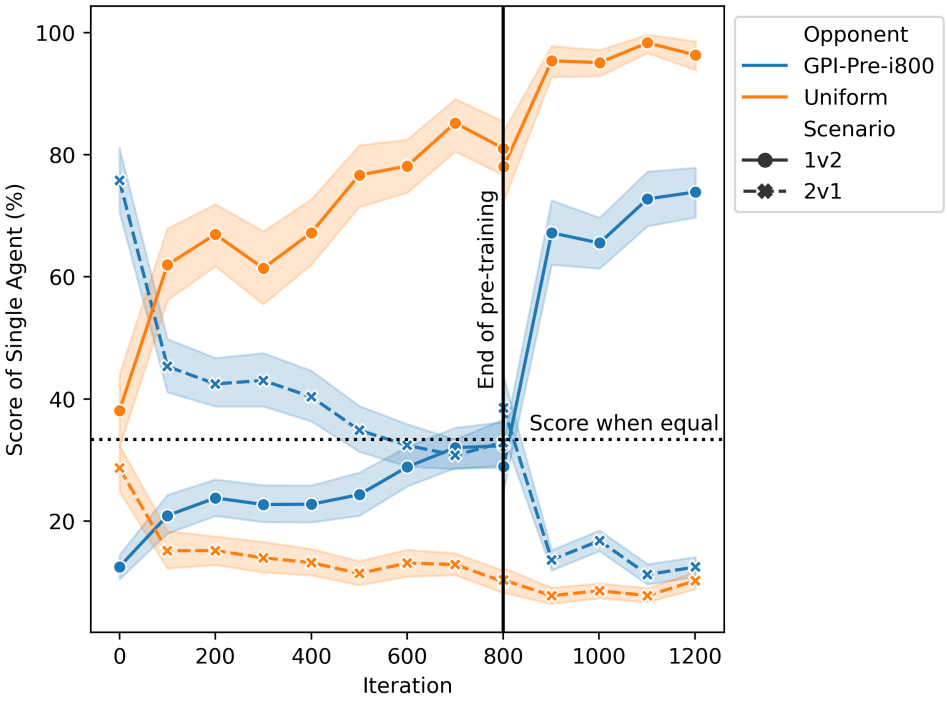



**Figure 5.3:** Skill of agent on Lattice game variant across training iterations. Each individual graph shows, as a function of time, the average SoS score of a single agent playing 100 games against 2 copies of another, along with the 95% confidence interval. The color of a line indicates the opponent agent being compared against (GPI-Pre-i800 or Uniform). Solid lines show games with 1 copy of the agent and 2 copies of the opponent. Dashed lines show games with 2 copies of the agent and 1 copy of the opponent. Scores are always from the perspective of the single agent. The solid vertical line indicates the end of the pre-training stage at iteration 800. The dashed horizontal line shows a score of $1/3 \approx 33.3\%$, which is the expected score of a single agent playing against 2 copies of itself.

| 1x ↓ vs 2x → | **Uniform** | **GPI-Pre-i800** | **GPI** |
|---|---|---|---|
| **Uniform** | - | 10.1%±0.9% | 10.2%±0.7% |
| **GPI-Pre-i800** | 81.0%±2.2% | - | 12.5%±0.8% |
| **GPI** | 96.3%±1.2% | 73.9%±2.3% | - |

**Table 5.3:** Average SoS scores of agents playing 100 games against 2 copies of another agent on the Lattice game variant, along with standard error (after ±). An agent playing against 2 copies of itself scores an average of $1/3 \approx 33.3\%$, and such comparisons are therefore omitted ("-"). The comparisons of GPI with GPI-Pre-i800 and the Uniform agent correspond to the final iteration of figure 5.3, and the comparisons of GPI-Pre-i800 with the Uniform agent corresponds to the orange points at iteration 800.

has a greater advantage against the Uniform agent in a complex environment where each player has a larger proportion of influence.

Additional tournaments are held against GPI-Pre-i800. This version of the agent is more powerful than the Uniform agent, but should by design be outclassed by further training. As expected, GPI-Pre-i800 soundly beats the agent at the first iteration of training, and performance tends towards $1/3 \approx 33.3\%$ as pre-training progresses. At the final iteration of pre-training, the agent is effectively playing tournament games against itself. Therefore, an average score of 33.3% is expected in both the 1v2 and 2v1 scenarios against GPI-Pre-i800 at iteration 800. This seems approximately correct. As training progresses, GPI-Pre-i800 is soundly outperformed by the agent.

### 5.3.2 Skill of Final Iteration

Table 5.3 shows the result of tournaments between the agent at its final iteration (GPI), the agent at pre-training iteration 800 (GPI-Pre-i800), and the Uniform agent.

Both GPI and GPI-Pre-i800 clearly outperform the Uniform agent in all respects: When playing as the 1x agent, they obtain scores far higher than $1/3 \approx 33.3\%$, and when playing the 2x agent, the Uniform agent scores far lower. By the same metric, GPI clearly outperforms GPI-Pre-i800.

## 5.4 Presentation and Analysis of Results for the Hub Game Variant

For the Hub game variant, 900 iterations of training are performed. The first 100 iterations form the pre-training stage as discussed in section 4.8.2. Training was conducted over a period of 2 days, and tournament games took around a day to complete. 1024 iterations of search are performed at each turn, searching over 5 plausible actions.

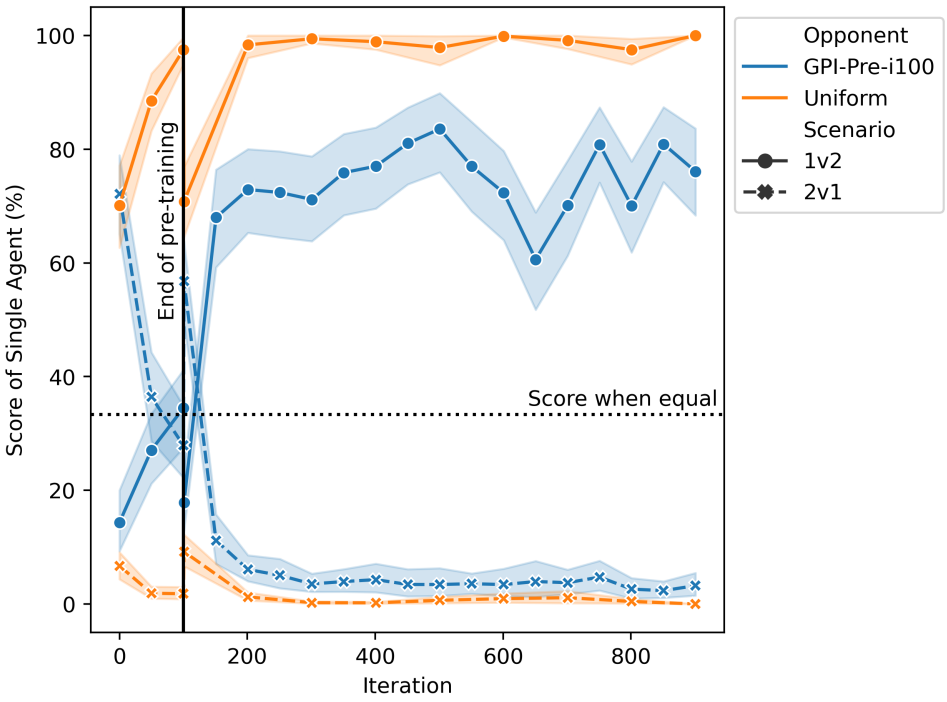



**Figure 5.4:** Skill of agent on Hub game variant across training iterations. Each individual graph shows, as a function of time, the average SoS score of a single agent playing 100 games against 2 copies of another, along with the 95% confidence interval. The color of a line indicates the opponent agent being compared against (GPI-Pre-i100 or Uniform). Solid lines show games with 1 copy of the agent and 2 copies of the opponent. Dashed lines show games with 2 copies of the agent and 1 copy of the opponent. Scores are always from the perspective of the single agent. The solid vertical line indicates the end of the pre-training stage at iteration 100. The dashed horizontal line shows a score of $1/3 \approx 33.3\%$, which is the expected score of a single agent playing against 2 copies of itself.

In addition to the Uniform agent, comparisons are made with two alternative versions of the agent: GPI-Pre-i100 the agent at the final iteration of pre-training (iteration 100), and GPI-Pre-i900, which results from 800 additional iterations of pre-training initialized with GPI-Pre-i100.

### 5.4.1 Progression of Skill

Figure 5.4 shows the agent's progression of skill throughout training, measured through tournaments with the Uniform agent and GPI-Pre-i100.

Consistent with the results for the Pure and Lattice game variants, the agent outperforms the Uniform agent already at iteration 0, and its performance dips at the first iteration after pre-training. Unlike those variants, however, the agent is able to soundly beat the Uniform agent already at the final iteration of the

| 1x ↓ vs 2x → | **Uniform** | **GPI-Pre-i100** | **GPI-Pre-i900** | **GPI** |
|---|---|---|---|---|
| **Uniform** | - | 0.1%±0.1% | 0.4%±0.2% | 0.0%±0.0% |
| **GPI-Pre-i100** | 99.4%±0.6% | - | 17.8%±2.9% | 3.2%±1.0% |
| **GPI-Pre-i900** | 100.0%±0.0% | 36.6%±3.8% | - | 6.1%±1.9% |
| **GPI** | 100.0%±0.0% | 76.1%±3.9% | 69.1%±4.2% | - |

**Table 5.4:** Average SoS scores of agents playing 100 games against 2 copies of another agent on the Hub game variant, along with standard error (after ±). An agent playing against 2 copies of itself scores an average of $1/3 \approx 33.3\%$, and such comparisons are therefore omitted ("-"). GPI-Pre-i900 is an agent produced by continuing pre-training until iteration 900. The comparisons of GPI with GPI-Pre-i100 and the Uniform agent correspond to the final iteration of figure 5.4, and the comparisons of GPI-Pre-i100 with the Uniform agent corresponds to the orange points at iteration 100.

(relatively short) pre-training stage. Therefore, the Uniform agent is not a good benchmark of skill after the pre-training stage.

Comparisons to the final iteration of pre-training (GPI-Pre-i100) are mostly consistent with those for the Lattice game variant: The agent is soundly beaten by GPI-Pre-i100 in early iterations, then the score tends towards $1/3 \approx 33.3\%$, and finally the agent soundly outperforms GPI-Pre-i100 as training progresses. At iteration 100, the performance of the agent in the 2v1 scenario deviates noticably, however. The expected score is within the confidence interval, but the mean deviates. Had more than 100 games been played per scenario, the means should converge to 33.3%. This result is instructive to the plot at large, as the relatively low amount of games (100 per comparison) can lead to misleading results: Statistically we know the true average score is 33.3% against GPI-Pre-i100 at iteration 100 since the agent under training is equal to GPI-Pre-i100 at this iteration, but the plot seems to indicate a different score.

The 1v2 invasion score against GPI-Pre-i100 fluctuates significantly starting at iteration 500. The reason for this is likely the same as for the performance dips analysed for the Pure game variant.

### 5.4.2 Skill of Final Iteration

Table 5.4 shows the result of tournaments between the agent at its final iteration (GPI), the agent at pre-training iteration 100 (GPI-Pre-i100), an agent resulting from 800 additional pre-training iterations (GPI-Pre-i900), and the Uniform agent.

GPI-Pre-i900 scores better than GPI-Pre-i100 in most respects: It better protects against invasion from GPI, it is more able to invade GPI, it is more able to invade GPI-Pre-i100 than vice versa, and it is more able to invade the Uniform agent. The only tournament setting where GPI-Pre-i100 wins over GPI-Pre-i900 is in invading 6x Uniform agents, and the difference is very small.

All of GPI, GPI-Pre-i100 and GPI-Pre-i900 clearly outperform the Uniform

agent in all respects: When playing as the 1x agent, they obtain scores far higher than $1/3 \approx 33.3\%$ (close to 100%), and when playing the 2x agent, the Uniform agent scores far lower (close to 0%). By the same metric, GPI clearly outperforms both GPI-Pre-i100 and GPI-Pre-i900.

From this, one can conclude that pre-training produced a reasonably good agent (GPI-Pre-i100), and that while continuing the pre-training stage 800 additional iterations produces a better agent (GPI-Pre-i900), the same amount of iterations spent without the limitations of the pre-training stage produce an even better agent (GPI).

## 5.5 Summary of Results and Analysis

An agent was trained for each of the three game variants Pure, Lattice and Hub. Each agent was evaluated through skill in tournaments, the results of which was presented and analysed in this chapter. Comparisons between game variants were pointed out as part of the analysis of the preceding sections. This section summarizes the chapter by first focusing on similarities and differences between game variants, and then on results and analysis unique to the Pure game variant.

### 5.5.1 Summary of Similarities and Differences

Trained agents for all game variants outperform the Uniform agent. In the Hub game variant, the pre-training stage is enough to beat the Uniform agent convincingly, whereas the Pure and Lattice variants benefit from further training. Trained agents for all game variants have an advantage against the Uniform agent already at iteration 0, hinting at the value of search in the end-game even without a properly trained value network.

For the Lattice and Hub game variants, comparisons are made against the final pre-training agent. Further training outperforms the pre-training agent in both cases. For the Hub variant, the pre-training stage was extended to match the total iterations behind the search-based agent. This extended pre-training outperforms the younger pre-training agent, but is outperformed by the search-based agent.

The results for the Hub variant demonstrate weakness in the measurement process, and highlight the importance of showing confidence intervals: For two of the tournaments in its skill-progression plot, the true expected score is known. The mean score deviates noticably, but the true score is within the confidence interval.

For all game variants a dip in performance exists at the first iteration after the pre-training stage. The dip is especially significant for the Pure game variant, where it dips below the performance at the first iteration. This dip is attributed to the use of an untrained order decoder, and shows the importance of having a good plausible action set. Since the untrained order decoder is demonstratively bad, it also shows the importance of action exploration.

For all game variants, performance roughly increases against all opponent agents throughout training. Variation exist, however, and can be attributed to the nature of the RL algorithm and the fact that skewed agents can require a larger number of measurement games.

### 5.5.2 Summary of Results and Analysis for the Pure Game Variant

For the Pure game variant, additional tournament games are played to measure the fully trained agent's relative skill with each power. Results indicate that the agent is somewhat more skilled with some powers than others, but only by a small amount.

Three factors differentiate the tournament results of the Pure game variant from the other two: The bug documented in appendix B.1, usage of a computationally restricted version of the agent for the skill-progression plot, and the presence of an existing competent baseline agent, i.e., HatløA2C.

The computationally restricted agent turns out to perform better than the original unrestricted version in comparison to other agents. This is surprising, and could be either because a smaller plausible action space is beneficial, or because "proper" game-theoretic play is not an advantage when playing agents that are not also approaching the game with game theory. In tournaments between the restricted and unrestricted version, neither is able to successfully invade the other. The unrestricted version achieves the highest invasion score, however. The result likely relates to the presence of a wide space of equilibria in Diplomacy [3].

HatløA2C is shown to be competent when ported to this thesis' game environment by comparisons with the Uniform agent, cross-referenced with the results reported in its original environment. The search-based agent soundly beats HatløA2C.

# Chapter 6

# Conclusion

This chapter concludes the thesis by discussing research questions in light of the analyzed results, listing thesis contributions, and outlining suggestions for future work. The chapter opens with a thesis review, and closes with an epilogue.

## 6.1 Thesis Review

This thesis has explored the application of state-of-the-art deep reinforcement learning techniques in the game of Diplomacy. Diplomacy is an unconventional domain for AI where the best results have been achieved using (generalized) policy iteration with game-theoretic search as the improvement operator. Recent research has achieved success on the classic formulation of the game, which raises an interest as to whether the proposed techniques generalize to other problems. The game-theoretic situations that arise in Diplomacy can be related to real-life issues like negotiation, tactics, and cooperation. The game also features a large action space that poses special challenges for RL and require a novel representation technique. The thesis has focused on the generality of state-of-the-art techniques by training agents on variants of the classic game.

Chapter 1 motivated the thesis by introducing the idea of board games as benchmarks for AI, Diplomacy as a particularly interesting benchmark, and by giving an overview of the state of AI for Diplomacy. The chapter then stated the thesis goal and the two research questions, both focusing on the application of state-of-the-art techniques in Diplomacy game variants.

Chapter 2 presented the Diplomacy board game, as well as relevant background theory: neural networks, reinforcement learning, and game theory. Importantly for the following discussion of research questions, it established "generalized policy iteration" (GPI) as a term used to refer to any reinforcement learning method that utilizes a policy improvement operator during training, and introduced policy gradient methods as an alternative to this approach.

Chapter 3 contextualized the thesis by giving a brief summary of early work on AI for Diplomacy and RL for board games in general, and then giving a detailed summary of relevant research in RL for Diplomacy.

Chapter 4 detailed the methodology used to address research questions. Primarily inspired by Double Oracle Reinforcement Learning for Action Exploration (DORA) of Bakhtin *et al.* [3] (section 3.4.4), an agent is implemented that samples a subset of the action space from a neural network "order decoder", optionally augments this subset with action exploration to add reasonable actions, and then performs game-theoretic Regret Matching search with next-state values from a neural network "value decoder" to produce a joint strategy over the action subset approximating a Nash Equilibrium. The agent plays its part in this joint strategy. The neural network decoders share a common "encoder", and use the Encoder-Decoder architecture. Agents are trained through self-play on three Diplomacy game variants: Pure, Lattice and Hub.

Chapter 5 presented and analysed results from training an agent for each of the three game variants Pure, Lattice and Hub. As is common in recent RL research for Diplomacy, the agents were evaluated based on "one-versus-all" tournaments against chosen opponent agents. All agents were compared against the Uniform agent. Additionally, the agent trained for the Pure game variant was compared with the A2C agent of Hatlø [20] (here called HatløA2C), and the Lattice and Hub agents were compared with their pre-training version. The insight gained from chapter 5 will be used to address the two research questions in the following section.

## 6.2 Discussion

With the thesis reviewed, the research questions of section 1.2 can be discussed in light of the analyzed results:

**Research Question 1** *Can state-of-the-art techniques successfully learn to play Diplomacy map variants?*

In answering the question it is necessary to first establish the developed system as an implementation of state-of-the-art techniques, to elaborate on what it means to successfully learn to play map variants, and to establish success criteria. Then, the results can be discussed in relation to the established success criteria.

The RL system presented in chapter 4 is based on the latest directly relevant work in RL for Diplomacy: DORA of Bakhtin *et al.* [3]. Later work builds on Bakhtin *et al.* [3] by regularizing search by a human prior (Sections 3.4.6, 3.4.7 and 3.4.8). These later works assume the presence of human data, which is not generally available for Diplomacy game variants. When learning to play Diplomacy game variants, the agent is expected to learn from scratch without human data. Learning from scratch is exactly the focus of Bakhtin *et al.* [3], and the implemented RL system for this thesis can therefore be considered an implementation of state-of-the-art techniques[1].

[1] State-of-the-art in the context of learning from scratch with no human data.

The developed system does deviate from state-of-the-art in several areas, however. First, the KL-Divergence loss function is used for both the order decoder and value decoder. This loss function is novel for Diplomacy RL, and was chosen because it is intuitive given that learning targets for both decoders are probability distributions[2]. Loss functions more commonly used in relevant work would enhance the relevancy of the results, but the choice of loss function is not believed to be a major factor in answering the research questions. Second, Bakhtin *et al.* [3] train their order decoder in the pre-training stage, whereas this thesis has chosen not to. This choice is likely to blame for the performance dips observed for the first iteration after pre-training in all game variants, and training the order decoder from the start could have led to more efficient training. On the other hand, the simplified pre-training stage has reduced complexity in training, and has been shown to work well in practice. Third, although early work on Diplomacy RL shared the encoder between the value decoder and order decoder, later work (including Bakhtin *et al.* [3]) has opted to split the network in two, with separate encoders for each decoder. The reasons for splitting the network are sound, but this thesis decided to stick with the shared architecture because it reduces the number of weights and allows the board state encoding to be reused across the decoders. Most GPI works use a split architecture, but Anthony *et al.* [1] is an example of a GPI implementation with a shared architecture. The choice of a shared architecture therefore does not disqualify the system as representing state-of-the-art[3]. Fourth, similar to the previous point, Bakhtin *et al.* [3] swaps out the GCN-based encoder of earlier work with a transformer-based encoder. This thesis again decided to stick with the "original" architecture of using a GCN. As Gray *et al.* [17] exists as an example of a GPI work that uses a GCN-based encoder, the choice is not considered to disqualify the system. Fifth, the board state representation in related work is typically more complex than that chosen for this thesis: They include a representation of the board state and orders chosen last turn. The simpler board state representation simplifies the learning task, but robs the agent of the opportunity to react to opponent actions in the previous turn. For the simple variants considered in this thesis, this trade-off is considered acceptable.

When evaluating whether agents can successfully learn to play a map variant, three aspects are considered. First, learning is a process that happens over time, and improvement over time is therefore expected in a well-functioning RL agent. Second, improvement over time is only impressive if the agent eventually becomes somewhat competent at the game. The Uniform agent is used as a trivial baseline. Third, state-of-the-art RL for Diplomacy consists of many moving parts, and all of these parts are expected to contribute to the success of the agent; The RL system is expected to be cohesive. As an example, consider a system that reaches some level of competence through simple pre-training, but fails to apply its more intricate components in a useful way. This system would do just as well playing by the

[2]The output of the order decoder is most often a probability distribution over only a subset of the action space (It doesn't necessarily sum to 1), and is thus not a true probability distribution.

[3]In the context of GPI as a state-of-the-art technique, as opposed to earlier techniques.

simple pre-training procedure, and the rest of the system becomes superfluous. In this scenario, the system as a whole can be considered to fail the learning task although part of the system did its job well.

The research question inquires about map variants in general. Therefore, experimentation on several different map variants is needed to address the question. The three game variants chosen for this thesis differ in map topology and player count. Also, both intuition and results show that they pose varying degrees of difficulty to the RL system. Hub turns out to be a simple variant to learn, as pre-training is enough to dominate the Uniform agent. Pure is the most difficult variant due to its high player count. Lattice has a low player count, but features a relatively complex map topology that takes the agent many iterations to learn in pre-training, before eventually dominating the Uniform agent. The chosen game variants are diverse, and are therefore considered appropriate candidates for addressing the research question.

With the question contextualized, and the methodology argued to be appropriate for answering it, the results and analysis of chapter 5 can be evaluated on the three defined success criteria: improvement over time, competence, and cohesion. The skill of all trained agents against all chosen baseline opponent agents is shown to roughly improve over time. The improvement is not monotonic, but the observed variation in skill over time can be attributed to the nature of RL algorithms. All trained agents are shown to be competent against the chosen baseline opponents. The Uniform agent is significantly outperformed in the complex Pure variant, and completely outclassed in the Lattice and Hub variants. Comparisons are also made with more competent baseline agents: The agents at their pre-training stage (Lattice and Hub) and HatløA2C (Pure). These agents are shown to be competent through tournaments with the Uniform agent, and the fully trained agents are shown to outperform them. The system can be argued to be cohesive based on the successful results for the fully trained agents against their pre-training counterpart for Lattice and Hub. This shows that the pre-training stage alone is not sufficient for reaching maximal skill. Analysis also indicates that a well-trained order decoder and action exploration is crucial to the success of the agent.

These results give a strong indication that, indeed, state-of-the-art techniques can successfully learn to play Diplomacy map variants.

**Research Question 2** *Do state-of-the-art generalized policy iteration techniques compare favorably to policy gradient techniques for Diplomacy on the Pure game variant?*

The arguments for the developed RL system being an appropriate implementation of state-of-the-art techniques apply to this research question also. As elaborated when presenting the research question in section 1.2, the openly available Actor-Critic (A2C) agent of Hatlø [20] (HatløA2C) is used as a competent reference policy gradient agent. As the game environment of this thesis differs slightly from the environment HatløA2C was originally trained for, modifications to its op-

eration were necessary. To prove the competence of HatløA2C, tournaments from Hatlø [20] were reproduced. Results show that HatløA2C is at least as competent as in its original game environment, and that it serves as a good reference agent for comparison. An agent was trained on the Pure game variant, the game variant that HatløA2C plays. Results show that the trained agent soundly beats HatløA2C, showing that state-of-the-art generalized policy iteration techniques can outperform a competent policy Gradient agent.

## 6.3 Contributions

This thesis makes the following contributions:

- A detailed and holistic explanation of state-of-the-art in reinforcement learning for Diplomacy through chapter 2, 3, and 4. Special focus is put on the order decoder, as this complex component receives little attention in the way of explanation in prior research.
- A thorough exploration of the two research questions through the discussion in the preceding section.
- A successful application of state-of-the-art techniques in three variants of the classic Diplomacy game.
- An agent that outperforms Hatlø [20], establishing a new state-of-the-art for the Pure game variant.
- An application of GPI techniques using the MILA game engine which can train agents for arbitrary game variants[4]. Prior research has utilized faster game engine implementations written in C++, hardcoded for the classic game variant.
- A possible bug in the MILA game engine is uncovered, which affects search-based agents seeking to minimize copying by mutating state. See appendix B.1.

## 6.4 Future Work

Reinforcement Learning for Diplomacy is a complex domain, with many interesting topics to explore. The following sections outline suggestions for future work building upon this thesis.

### 6.4.1 Exploration of Search Regularized by a Computational Agent

A key component in RL for Diplomacy when playing against humans is regularized search, as presented in related work sections 3.4.6 and 3.4.7, and briefly discussed in the analysis of results for Pure when comparing against HatløA2C. Further improvement against HatløA2C (or the agent developed in this thesis)

[4]Within the ruleset modifications of section 4.1.

likely requires an application of regularized search, to enable the agent to imitate the opponent. Developing an agent that regularizes against a computational agent is a novel avenue for further research, and would enable experimentation with regularized search on game variants lacking human data.

### 6.4.2 Investigation of Computational Restrictions

For the Pure game variant, a computationally restricted version of the agent performed best. This is surprising, and was not explored further in this thesis due to not being directly relevant to the research questions at hand. Reproducing these results and explaining them can enhance understanding of the DORA algorithm.

### 6.4.3 Application in Other Games

This thesis has shown that state-of-the-art techniques can generalize to variants of the classic Diplomacy game. Exploring the application of the techniques to other games with similar traits is a natural next step.

### 6.4.4 Fast Engine Supporting Game Variants

The MILA engine is flexible, but slow. The development of a fast OpenSource Diplomacy engine supporting game variants could benefit the Diplomacy research community.

### 6.4.5 Exploration of Symmetry in Diplomacy Maps

All game variants considered in this thesis are symmetric (section 4.8.1). It is possible to take advantage of this to achieve more compact state spaces. This was not explored as it does not generalize to game variants in general. Exploration of asymmetry in Diplomacy through game variants could also be an interesting topic.

## 6.5 Epilogue

In general, this work shows that state-of-the-art reinforcement learning techniques that have seen success on the classic Diplomacy map can be applied successfully in alternative map topologies. This result hints at the generality of the techniques and is a step toward their application in real-life issues.

# Bibliography


[1] T. Anthony, T. Eccles, A. Tacchetti, J. Kramár, I. Gemp, T. Hudson, N. Porcel, M. Lanctot, J. Perolat, R. Everett, S. Singh, T. Graepel and Y. Bachrach, 'Learning to Play No-Press Diplomacy with Best Response Policy Iteration,' in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 17 987–18 003. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/d1419302db9c022ab1d48681b13d5f8b-Abstract.html (visited on 26/08/2022).

[2] R. Aydoğan, T. Baarslag, K. Fujita, J. Mell, J. Gratch, D. de Jonge, Y. Mohammad, S. Nakadai, S. Morinaga, H. Osawa, C. Aranha and C. M. Jonker, 'Challenges and Main Results of the Automated Negotiating Agents Competition (ANAC) 2019,' en, in *Multi-Agent Systems and Agreement Technologies*, N. Bassiliades, G. Chalkiadakis and D. de Jonge, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 366–381, ISBN: 978-3-030-66412-1. DOI: 10.1007/978-3-030-66412-1_23.

[3] A. Bakhtin, D. Wu, A. Lerer and N. Brown, *No-Press Diplomacy from Scratch*, arXiv:2110.02924 [cs], Oct. 2021. DOI: 10.48550/arXiv.2110.02924. [Online]. Available: http://arxiv.org/abs/2110.02924 (visited on 29/08/2022).

[4] A. Bakhtin, D. J. Wu, A. Lerer, J. Gray, A. P. Jacob, G. Farina, A. H. Miller and N. Brown, *Mastering the Game of No-Press Diplomacy via Human-Regularized Reinforcement Learning and Planning*, arXiv:2210.05492 [cs], Oct. 2022. DOI: 10.48550/arXiv.2210.05492. [Online]. Available: http://arxiv.org/abs/2210.05492 (visited on 23/11/2022).

[5] N. Brown and T. Sandholm, 'Superhuman AI for multiplayer poker,' *Science*, vol. 365, no. 6456, pp. 885–890, Aug. 2019, Publisher: American Association for the Advancement of Science. DOI: 10.1126/science.aay2400. [Online]. Available: https://www.science.org/doi/full/10.1126/science.aay2400 (visited on 12/10/2022).

[6] A. B. Calhamer, *Diplomacy*, 1959.

[7] M. Campbell, A. J. Hoane and F.-h. Hsu, 'Deep Blue,' en, *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, Jan. 2002, ISSN: 0004-3702. DOI: 10.1016/S0004-3702(01)00129-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291 (visited on 12/12/2022).

[8] R. Cottle, *The linear complementarity problem* (Classics in applied mathematics ;), eng, SIAM ed., [Classics ed.]. Philadelphia, Pa.: Society for Industrial and Applied Mathematics SIAM Market Street, Floor 6, Philadelphia, PA 19104, 2009, vol. 60, ISBN: 978-0-89871-900-0.

[9] *Diplomacy AI - Albert*, no. [Online]. Available: https://sites.google.com/site/diplomacyai/ (visited on 11/12/2022).

[10] *Diplomacy_rules.pdf*. [Online]. Available: https://media.wizards.com/2015/downloads/ah/diplomacy_rules.pdf (visited on 03/11/2022).

[11] FAIR, A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, A. P. Jacob, M. Komeili, K. Konath, M. Kwon, A. Lerer, M. Lewis, A. H. Miller, S. Mitts, A. Renduchintala, S. Roller, D. Rowe, W. Shi, J. Spisak, A. Wei, D. Wu, H. Zhang and M. Zijlstra, 'Human-level play in the game of Diplomacy by combining language models with strategic reasoning,' *Science*, vol. 0, no. 0, eade9097, Nov. 2022, Publisher: American Association for the Advancement of Science. DOI: 10.1126/science.ade9097. [Online]. Available: https://www.science.org/doi/10.1126/science.ade9097 (visited on 24/11/2022).

[12] A. Ferreira, H. L. Cardoso and L. P. Reis, 'DipBlue: A Diplomacy Agent with Strategic and Trust Reasoning,' in *Proceedings of the International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,*, Backup Publisher: INSTICC, SciTePress, 2015, pp. 54–65, ISBN: 978-989-758-073-4. DOI: 10.5220/0005205400540065.

[13] J. Foerster, I. A. Assael, N. de Freitas and S. Whiteson, 'Learning to Communicate with Deep Multi-Agent Reinforcement Learning,' in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf.

[14] B. Fogel, 'To Whom Tribute Is Due: The Next Step in Scoring Systems,' en, Mar. 2020.

[15] Y. Freund and R. E. Schapire, 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,' en, *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997, ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002200009791504X (visited on 14/11/2022).

[16] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[17] J. Gray, A. Lerer, A. Bakhtin and N. Brown, *Human-Level Performance in No-Press Diplomacy via Equilibrium Search*, arXiv:2010.02923 [cs], May 2021. DOI: 10.48550/arXiv.2010.02923. [Online]. Available: http://arxiv.org/abs/2010.02923 (visited on 29/08/2022).

[18] W. L. Hamilton, 'Graph Representation Learning,' en, *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, {:publisher: Morgan and Claypool}.

[19] J. Hannan, '4. APPROXIMATION TO RAYES RISK IN REPEATED PLAY,' en, in *4. APPROXIMATION TO RAYES RISK IN REPEATED PLAY*, Princeton University Press, Mar. 2016, pp. 97–140, ISBN: 978-1-4008-8215-1. DOI: 10.1515/9781400882151-006. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/9781400882151-006/html (visited on 16/06/2023).

[20] A. M. Hatlø, 'Å lære No-Press Diplomacy fra Selvspill: Dyp Reinforcement Learning med Fokus på Samarbeid mellom Agenter,' Norwegian, M.S. thesis, NTNU, Trondheim, May 2022.

[21] R. Herbrich, T. Minka and T. Graepel, 'TrueSkill™ : A Bayesian Skill Rating System,' in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt and T. Hoffman, Eds., vol. 19, MIT Press, 2006. [Online]. Available: https://proceedings.neurips.cc/paper/2006/file/f44ee263952e65b3610b8ba51229d1f9-Paper.pdf.

[22] J. Hu and M. P. Wellman, 'Nash q-learning for general-sum stochastic games,' *The Journal of Machine Learning Research*, vol. 4, no. null, pp. 1039–1069, 2003, ISSN: 1532-4435.

[23] A. P. Jacob, D. J. Wu, G. Farina, A. Lerer, H. Hu, A. Bakhtin, J. Andreas and N. Brown, 'Modeling Strong and Human-Like Gameplay with KL-Regularized Search,' in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, Jul. 2022, pp. 9695–9728. [Online]. Available: https://proceedings.mlr.press/v162/jacob22a.html.

[24] D. de Jonge and C. Sierra, 'D-Brane: A diplomacy playing agent for automated negotiations research,' en, *Applied Intelligence*, vol. 47, no. 1, pp. 158–177, Jul. 2017, ISSN: 1573-7497. DOI: 10.1007/s10489-017-0919-y. [Online]. Available: https://doi.org/10.1007/s10489-017-0919-y (visited on 07/09/2022).

[25] D. d. Jonge, T. Baarslag, R. Aydoğan, C. Jonker, K. Fujita and T. Ito, 'The Challenge of Negotiation in the Game of Diplomacy,' in *Agreement Technologies, 6th International Conference, AT 2018, Bergen, Norway, December 6-7, 2018, Revised Selected Papers*, M. Lujak, Ed., ser. Lecture Notes in Computer Science, vol. 11327, Cham: Springer International Publishing, 2019, pp. 100–114, ISBN: 978-3-030-17294-7. DOI: 10.1007/978-3-030-17294-7_8.

[26] A. Kofod-Petersen, *Finding literature — and how to read it*, en, Sep. 2021. [Online]. Available: https://research.idi.ntnu.no/aimasters/files/2021_09_28_SLR.pdf (visited on 25/05/2023).

[27] J. Kramár, T. Eccles, I. Gemp, A. Tacchetti, K. R. McKee, M. Malinowski, T. Graepel and Y. Bachrach, 'Negotiation and honesty in artificial intelligence methods for the board game of Diplomacy,' en, *Nature Communications*, vol. 13, no. 1, p. 7214, Dec. 2022, Number: 1 Publisher: Nature Publishing Group, ISSN: 2041-1723. DOI: 10.1038/s41467-022-34473-5. [Online]. Available: https://www.nature.com/articles/s41467-022-34473-5 (visited on 09/12/2022).

[28] S. Kraus and D. Lehmann, 'Diplomat, an agent in a multi agent environment: An overview,' in *Seventh Annual International Phoenix Conference on Computers an Communications. 1988 Conference Proceedings*, Mar. 1988, pp. 434–438. DOI: 10.1109/PCCC.1988.10117.

[29] S. Kraus, D. Lehmann and E. Ephrati, 'An Automated Diplomacy Player,' in *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympiad*, 1989, pp. 134–153.

[30] N. Littlestone and M. K. Warmuth, 'The Weighted Majority Algorithm,' en, *Information and Computation*, vol. 108, no. 2, pp. 212–261, Feb. 1994, ISSN: 0890-5401. DOI: 10.1006/inco.1994.1009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540184710091 (visited on 14/11/2022).

[31] T. Løkkeborg, 'Deep Reinforcement Learning (DRL) for International Diplomacy, IT3915 - Master in Informatics, Preparatory Project,' en, NTNU, Tech. Rep., Dec. 2022.

[32] H. B. McMahan, G. J. Gordon and A. Blum, 'Planning in the presence of cost functions controlled by an adversary,' in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03, Washington, DC, USA: AAAI Press, Aug. 2003, pp. 536–543, ISBN: 978-1-57735-189-4. (visited on 05/10/2022).

[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver and K. Kavukcuoglu, 'Asynchronous methods for deep reinforcement learning,' in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, Jun. 2016, pp. 1928–1937. (visited on 16/06/2023).

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, 'Human-level control through deep reinforcement learning,' en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, Number: 7540 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/

nature14236. [Online]. Available: https://www.nature.com/articles/nature14236 (visited on 12/12/2022).

[35] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson and M. Bowling, 'DeepStack: Expert-level artificial intelligence in heads-up no-limit poker,' en, *Science*, vol. 356, no. 6337, pp. 508–513, May 2017, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aam6960. [Online]. Available: https://www.science.org/doi/10.1126/science.aam6960 (visited on 17/11/2022).

[36] T. W. Neller and M. Lanctot, 'An Introduction to Counterfactual Regret Minimization,' en, p. 38, 2013.

[37] P. Paquette, Y. Lu, S. Bocco, M. O. Smith, S. Ortiz-Gagne, J. K. Kummerfeld, S. Singh, J. Pineau and A. Courville, *No Press Diplomacy: Modeling Multi-Agent Gameplay*, arXiv:1909.02128 [cs], Nov. 2019. [Online]. Available: http://arxiv.org/abs/1909.02128 (visited on 26/08/2022).

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, arXiv:1912.01703 [cs, stat], Dec. 2019. DOI: 10.48550/arXiv.1912.01703. [Online]. Available: http://arxiv.org/abs/1912.01703 (visited on 14/06/2023).

[39] J. Perolat, B. De Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J. T. Connor, N. Burch, T. Anthony, S. McAleer, R. Elie, S. H. Cen, Z. Wang, A. Gruslys, A. Malysheva, M. Khan, S. Ozair, F. Timbers, T. Pohlen, T. Eccles, M. Rowland, M. Lanctot, J.-B. Lespiau, B. Piot, S. Omidshafiei, E. Lockhart, L. Sifre, N. Beauguerlange, R. Munos, D. Silver, S. Singh, D. Hassabis and K. Tuyls, 'Mastering the game of Stratego with model-free multiagent reinforcement learning,' *Science*, vol. 378, no. 6623, pp. 990–996, Dec. 2022, Publisher: American Association for the Advancement of Science. DOI: 10.1126/science.add4679. [Online]. Available: https://www.science.org/doi/10.1126/science.add4679 (visited on 09/12/2022).

[40] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning representations by back-propagating errors,' en, *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, Number: 6088 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://www.nature.com/articles/323533a0 (visited on 14/06/2023).

[41] B. Sanchez-Lengeling, E. Reif, A. Pearce and A. B. Wiltschko, 'A Gentle Introduction to Graph Neural Networks,' en, *Distill*, vol. 6, no. 9, e33, Sep. 2021, ISSN: 2476-0757. DOI: 10.23915/distill.00033. [Online]. Available: https://distill.pub/2021/gnn-intro (visited on 05/09/2022).

[42] A. Shapiro, G. Fuchs and R. Levinson, 'Learning a Game Strategy Using Pattern-Weights and Self-play,' in *Computers and Games*, J. Schaeffer, M. Müller and Y. Björnsson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 42–60, ISBN: 978-3-540-40031-8.

[43] L. S. Shapley, 'Stochastic Games,' eng, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 10, pp. 1095–1100, Oct. 1953, ISSN: 0027-8424. DOI: 10.1073/pnas.39.10.1095.

[44] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, 1st ed. Cambridge University Press, Dec. 2008, ISBN: 978-0-521-89943-7 978-0-511-81165-4. DOI: 10.1017/CBO9780511811654. [Online]. Available: https://www.cambridge.org/core/product/identifier/9780511811654/type/book (visited on 05/09/2022).

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, 'Mastering the game of Go with deep neural networks and tree search,' en, *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, Number: 7587 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/nature16961. [Online]. Available: https://www.nature.com/articles/nature16961 (visited on 14/11/2022).

[46] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, 'Mastering the game of Go without human knowledge,' en, *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, Number: 7676 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/nature24270. [Online]. Available: https://www.nature.com/articles/nature24270 (visited on 22/11/2022).

[47] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive computation and machine learning series), Second edition. Cambridge, Massachusetts: The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[48] G. Tesauro, 'Temporal difference learning and TD-Gammon,' *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995, ISSN: 0001-0782. DOI: 10.1145/203330.203343. [Online]. Available: https://doi.org/10.1145/203330.203343 (visited on 12/12/2022).

[49] *Variants - vDiplomacy*. [Online]. Available: https://www.vdiplomacy.com/variants.php?variantID=11 (visited on 12/05/2023).

[50] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama,

D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps and D. Silver, 'Grandmaster level in StarCraft II using multi-agent reinforcement learning,' en, *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, Number: 7782 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. [Online]. Available: https://www.nature.com/articles/s41586-019-1724-z (visited on 19/01/2023).

[51] M. Zinkevich, M. Johanson, M. Bowling and C. Piccione, 'Regret Minimization in Games with Incomplete Information,' in *Advances in Neural Information Processing Systems*, vol. 20, Curran Associates, Inc., 2007. [Online]. Available: https://papers.nips.cc/paper_files/paper/2007/hash/08d98638c6fcd194a4b1e6992063e944-Abstract.html (visited on 16/06/2023).

# Appendix A

# Orders for Game Variant in Figure 4.2a

Denoted as $\Omega$ in section 4.4.2. $o_i$ is an integer index into $\Omega$, and $O_i$ is a set of these indexes.

1. A ELT H
2. A ELT D
3. A ELT - HAN
4. A ELT R HAN
5. A ELT S A HAN
6. A ELT S A ELO - HAN
7. A ELT - ELO
8. A ELT R ELO
9. A ELT S A ELO
10. A ELT S A HAN - ELO
11. A HAN H
12. A HAN D
13. A HAN - ELT
14. A HAN R ELT
15. A HAN S A ELT
16. A HAN S A ELO - ELT
17. A HAN - ELO
18. A HAN R ELO
19. A HAN S A ELO
20. A HAN S A ELT - ELO
21. A ELO H
22. A ELO D
23. A ELO - ELT
24. A ELO R ELT
25. A ELO S A ELT
26. A ELO S A HAN - ELT

27. A ELO - HAN
28. A ELO R HAN
29. A ELO S A HAN
30. A ELO S A ELT - HAN
31. A ELT B
32. A ELO B

# Appendix B

# Known Bugs

This appendix documents all known bugs. The first bug is significant and has been fixed, but affects results for the Pure variant since results were gathered prior to the fix. The other two are minor, and persist at the time of delivery.

## B.1 Retreat Bug

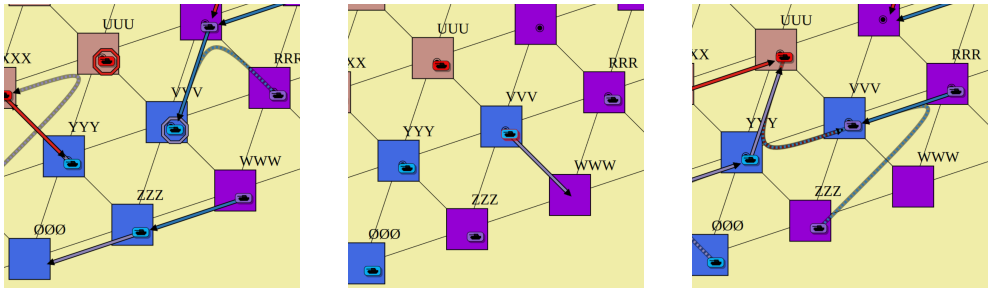
Search as an improvement operator requires that rollout games are played with the game engine. The game engine is stateful. One could make deepcopies of the game engine to perform rollout games, but this leads to an unacceptable proportion of runtime spent making copies. Instead, this thesis chooses to make use of functionality in the MILA game engine that allows state to be reset to a previous point. For each rollout game, the following steps are performed:

1. "phase = game.get_current_phase()" is used to obtain the current phase.
2. The game object is used to statefully step the game to a successor state given a joint action.
3. Estimated or actual end-game score is produced for the successor.
4. "game.set_state(game.state_history[phase])" is used to reset state back to the state before stepping the game to a successor, allowing for more rollout games from the same state, and eventually for using the game object for the rest of the game.

To the author's knowledge, no previous research has utilized the MILA game engine in this way, as no previous GPI research for Diplomacy has used the MILA game engine. The A2C agents of Paquette *et al.* [37] and Hatlø [20] do not need to perform rollout games in this way. It is believed that the "set_state" functionality might exist primarily for visualization purposes.

After training an agent for the Pure variant and running tournament games, it was discovered that retreat orders sometimes do not take effect when rollout games are used. An example is given in figure B.1, where a legal retreat to another region is ignored, and the unit is disbanded instead.

After digging through MILA code, it was discovered that a variable named




**(a)** VVV is dislodged.

**(b)** VVV orders a legal retreat to WWW.

**(c)** WWW is empty. VVV was disbanded.

**Figure B.1:** Example of retreat bug.

"popped" listing disbanded units is not cleared at the start of each retreat phase, only at the start of each movement phase. When performing rollout games in the retreat phase, an agent is very likely to have a disband order as one of it's legal orders try rolling out. After rolling out a disband order the first time, all following retreat orders are ignored, since the game engine considers the unit to be disbanded even after resetting state as presented above.

Based on this understanding of the problem, a fix was made before training the Lattice and Hub map variants that manually resets the "popped" variable before each retreat phase, as is done internally before each movement phase. Inspection of code indicates that resetting this variable should not have an adverse effect on the regular operation of the MILA game engine.

Refer to section 5.2 for a discussion of how this bug affects results for the Pure map variant.

## B.2 "Multiple orders" bug

While training, the MILA engine gives intermittent warnings like this:

```
-- MULTIPLE ORDERS FOR UNIT: A UUU
```

An agent has never been observed to choose multiple orders for a unit in visualized games, and the bug is therefore believed to appear while performing rollout games, which are not visualized. The bug has not been observed during tournament games. The bug is believed to have a similar cause as the retreat bug of the previous section. The bug appears especially often when training for the Lattice map variant.

As manual inspection of games has failed to discover situations where an agent has been negatively affected by the bug, and trained agents seem to operate reasonably, time has not been invested into fixing it.

## B.3 Rare Crash Bug

Training crashes very rarely with this stack trace:

```
Traceback (most recent call last):
  File "train.py", line 309, in <module>
    train(cfg)
  File "train.py", line 263, in train
    dataloader.generate_into_buffer(os.path.join(cfg['viz_directory'], f'datagen_i{iteration}'), cfg['trainer']['dataloader']['generate_per_iteration'])
  File "/work/thomahl/master_thesis/data_loader.py", line 226, in generate_into_buffer
    for i, played_game in yield_game_and_viz(self.model, self.cfg, update_step, viz_file_prefix + f'_local'):
  File "/work/thomahl/master_thesis/data_loader.py", line 54, in yield_game_and_viz
    for i, played_game in enumerate(yield_played_game(model, cfg, update_step)):
  File "/work/thomahl/master_thesis/game_generator.py", line 572, in yield_played_game
    step_value_targets[power_i] = batched_compute_joint_strategy_exact_ev(
  File "/work/thomahl/master_thesis/action_exploration.py", line 309, in batched_compute_joint_strategy_exact_ev
    assert u.shape[0] == all_combination_orders.shape[0],\
AssertionError: all_combination_orders should consist of unique combinations 1000 1100
```

Note the "should" part of the assertion message. This assertion was written in relation to an optimization that was made to avoid duplicate work while processing. The assertion is there to provide peace of mind that the optimization is working. If the assertion was not there and its condition was met, the program would operate as normal, but spend slightly longer than it needs to during action exploration. The bug most often happens for the Lattice map variant.

Since training is checkpointed, a crashed training session can simply be continued with no adverse effects. The bug has only occurred a handful of times across the thousands of training iterations performed during development and gathering of results. Therefore, a fix has not been prioritized.

# Appendix C

# Example Forward Pass of Neural Network

This section gives an example of how the neural network of section 4.5 produces an action and state value given the game state of figure 4.2. Neural network building blocks covered in the background chapter like GCN, LSTM and SoftMax are treated as black boxes. The example exists primarily to accompany the explanation of the order decoder in section 4.5.2, which is the most novel component of the neural network.

### Encoder

The board state representation of 4.2a becomes:

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

For the sake of example, assume the size of the encoding per region $E$ is 3, that there is only single GCN layer, and that its output given $\mathbf{B}$ and the adjacency matrix of the map is:

$$\mathbf{B}_{\text{enc}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

The row $\mathbf{B}_{\text{enc}}[r]$ holds the encoding of region $r$.

### Order Decoder

Assume the embedding of the red power is $e(p) = [.3, .4, .5]$, and that the size of order embeddings is 2. As per 4.2b and 4.2c, legal orders for units 1 and 2 are:

$$\begin{aligned} O_1 &= \{o_1^1, o_1^2, o_1^3, o_1^4, o_1^5, o_1^6, o_1^7\} \\ &= \{1, 3, 5, 6, 7, 9, 10\} \\ O_2 &= \{o_2^1, o_2^2, o_2^3, o_2^4, o_2^5, o_2^6, o_2^7\} \\ &= \{11, 13, 15, 16, 17, 19, 20\} \end{aligned}$$

For unit 1, the input to the LSTM is the concatenation of $\mathbf{B}_{\text{enc}}[1]$, $e(p)$, and a dummy previous order embedding of dummy $= [0, 0]$, since there is no previous order.

$$\begin{aligned} \text{lstm\_input}_1 &= \text{concat}(\mathbf{B}_{\text{enc}}[1], e(p), \text{dummy}) \\ &= \text{concat}([[1, 2, 3], [.3, .4, .5], [0, 0]]) \\ &= [1, 2, 3, .3, .4, .5, 0, 0] \end{aligned}$$

Assume that the application of LSTM returns:

$$\begin{aligned} \text{LSTM}(\text{lstm\_input}_1) &= \psi_1 \\ &= [1, 2] \end{aligned}$$

Note that the number of elements in vector $\psi_1$ must be 2, since that is the size of the order embedding. Assume further that the legal order embeddings form the matrix:

$$e(O_1) = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \\ 8 & 9 \\ 10 & 11 \\ 12 & 13 \end{bmatrix}$$

Where the $j$-th row of $e(O_1)$ corresponds to the embedding of the $j$-th legal order $o_1^j$.

The unnormalized logits $\xi_1$ is calculated as the matrix multiplication of $e(O_1) \in \mathbb{R}^{7\times 2}$ and $\psi_1 \in \mathbb{R}^2$ (treated as a matrix of size $2 \times 1$) is:

$$\begin{aligned} \xi_1 &= e(O_1) \cdot \psi_1 \\ &= \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \\ 8 & 9 \\ 10 & 11 \\ 12 & 13 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 8 & 14 & 20 & 26 & 32 & 38 \end{bmatrix} \end{aligned}$$

$\pi_O(O_1)$ can now be calculated:

$$
\begin{aligned}
\text{LSE}(\xi_1) &\approx \log(\sum_{j=1}^{|} \xi_1 | e^{\xi_1[j]}) \approx 38.002 \\
\xi_1^{normalized} &= \xi_1 - LSE(\xi_1) \\
&\approx \xi_1 - 38.002 \\
&\approx [-36.002, -30.002, -24.002, -18.002, -12.002, -6.002, -0.002] \\
\pi_O(O_1) &= e^{\xi_1^{normalized}} \\
&\approx e^{[-36.002, -30.002, -24.002, -18.002, -12.002, -6.002, -0.002]} \\
&\approx [0.000, 0.000, 0.000, 0.000, 0.000, 0.002, 0.998]
\end{aligned}
$$

$\pi_O(O_1)$ becomes a probability distribution over $O_1$ that assigns 0.2% probability to the order $o_1^6 = 9$, and 99.8% probability to the order $o_1^7 = 10$. The order decoder samples an $o_1$ from this distribution, in this case let's say $o_1^7$ since it is very likely. For unit 1, the order decoder returns $\pi_O(O_1)$ and $o_1$.

The order decoder then progresses to unit 2, using the embedding of the previous order $e(o_1)$ as one of the inputs to the LSTM. $e(o_1) = [1314]$, as per $e(O_1)$ shown earlier.

$$
\begin{aligned}
\text{lstm\_input}_2 &= \text{concat}(\mathbf{B}_{\text{enc}}[2], e(p), e(o_1)) \\
&= \text{concat}([[4,5,6],[.3,.4,.5],[13,14]]) \\
&= [4,5,6,.3,.4,.5,13,14]
\end{aligned}
$$

Assume that the LSTM [1] returns $\text{lstm\_input}_2$, and that the legal order embeddings form the matrix $e(O_2)$:

$$
\begin{aligned}
\text{LSTM}(\text{lstm\_input}_2) &= \psi_2 \\
&= [3,4] \\
e(O_2) &= \begin{bmatrix} 15 & 16 \\ 17 & 18 \\ 19 & 20 \\ 21 & 22 \\ 23 & 24 \\ 25 & 26 \\ 27 & 28 \end{bmatrix}
\end{aligned}
$$

The unnormalized logits $\xi_2$ is calculated as the matrix multiplication of $e(O_2) \in \mathbb{R}^{7\times 2}$ and $\psi_2 \in \mathbb{R}^2$ (treated as a matrix of size $2 \times 1$) is:

$$\xi_2 = e(O_2) \cdot \psi_2$$

[1] Hidden state from the previous sequence step is implicitly passed.

$\pi_O(O_2|o_1)$ can now be calculated:

$$
\begin{aligned}
\text{LSE}(\xi_2) &\approx \log(\sum_{j=1}^{|\xi_2|} e^{\xi_2[j]}) \approx -23.449 \\
\xi_2^{normalized} &= \xi_2 - LSE(\xi_2) \\
&\approx \xi_2 - (-23.449) \\
&\approx [-0.051, -3.051, -6.051, -9.051, -12.051, -15.051, -18.051] \\
\pi_O(O_2) &= e^{\xi_2^{normalized}} \\
&\approx e^{[-36.002, -30.002, -24.002, -18.002, -12.002, -6.002, -0.002]} \\
&\approx [0.950, 0.047, 0.002, 0.000, 0.000, 0.000, 0.000]
\end{aligned}
$$

$\pi_O(O_2|o_1)$ becomes a probability distribution over $O_2$ that assigns 95% probability to the order $o_2^1 = 11$, 4.7% probability to the order $o_2^2 = 13$, and 0.2% probability to the order $o_2^3 = 15$. The order decoder samples an $o_2$ from this distribution, in this case let's say $o_2^1$ since it is very likely. For unit 2, the order decoder returns $\pi_O(O_2|o_1)$ and $o_2$.

The full output of the order decoder in this example is:

$$
\begin{aligned}
\pi_O(O_1) &= [0.000, 0.000, 0.000, 0.000, 0.000, 0.002, 0.998] \\
o_1 &= 10 \\
\pi_O(O_2|o_1) &= [0.950, 0.047, 0.002, 0.000, 0.000, 0.000, 0.000] \\
o_2 &= 11
\end{aligned}
$$

The sequence of orders $[o_1, o_2] = [10, 11]$ corresponds to the Diplomacy action "A ELT S A HAN - ELO, A HAN H".

### Value Decoder

The value decoder flattens its input $\mathbf{B}_{\text{enc}}$, transforms it down to a vector with one element per power, then applies SoftMax to achieve a distribution summing to 1. Assume a single linear transformation followed by the application of the sigmoid activation function (Equation 2.1). Assume weight matrix $\mathbf{W} \in \mathbb{R}^{R \cdot E \times P}$, where $R$ is the number of regions, $E$ is the region encoding size, and $P$ is the number of powers. For the sake of explanation, assume $\mathbf{W}$ is all ones. The output of the value decoder becomes:

$$\hat{y_v} = \text{SoftMax}(\sigma(\text{flatten}(\mathbf{B}_{\text{enc}}) \cdot \mathbf{W})) = [0.5, 0.5]$$

$\hat{y_v}$ predicts that the game will end in a tie between the two players.

### Summary of example

The example showed how the neural network operates given input **B** and $O$. The order decoder in effect produces the action $\vec{a} = [o_1, o_2] = [10, 11]$, which corresponds to the Diplomacy action "A ELT S A HAN - ELO, A HAN H". The order decoder also outputs the probability distributions from which the orders were sampled, to allow for learning. The value decoder predicts the game will end in a tie between the two players.

# Appendix D

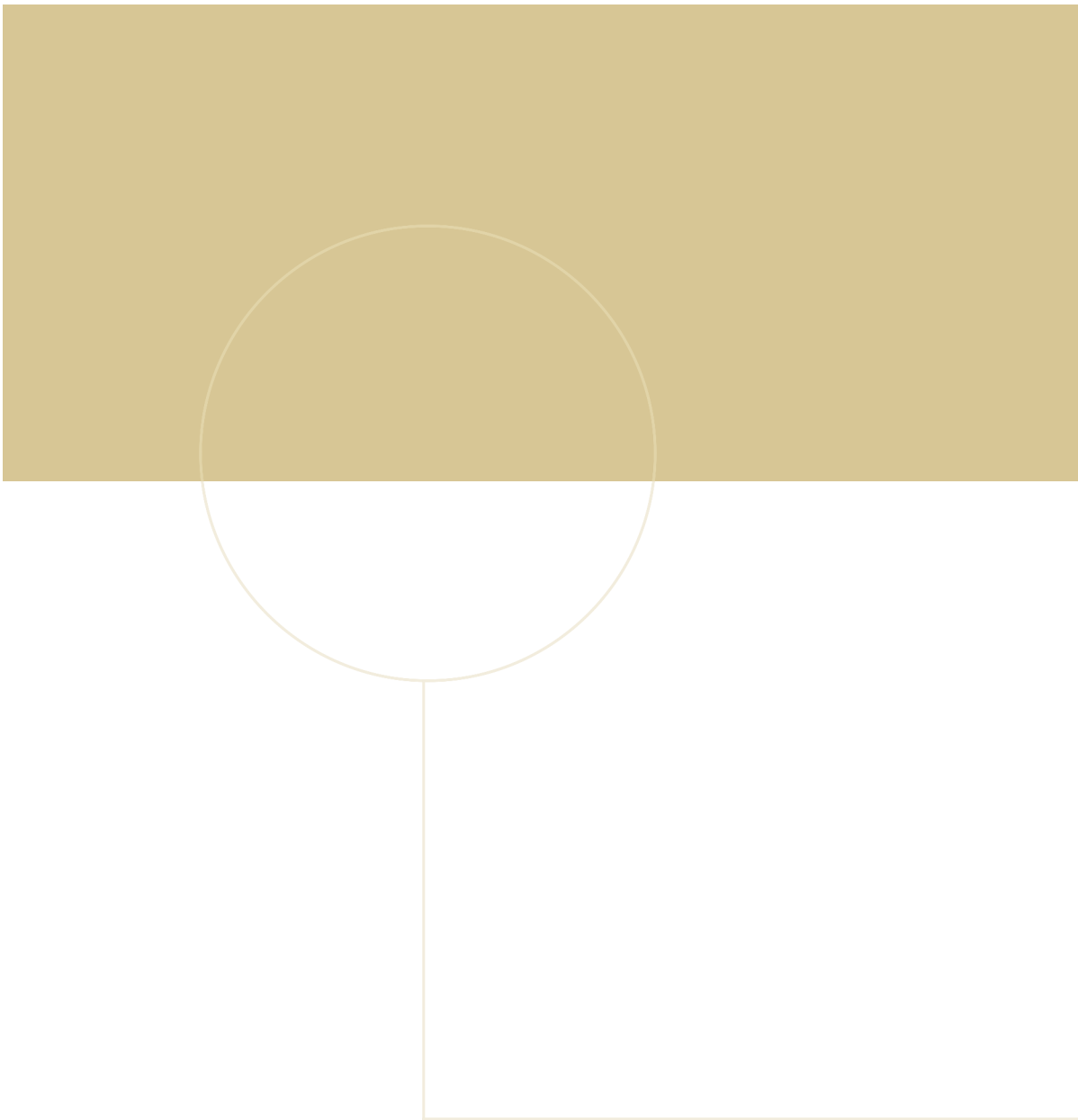
# Representation of Actions in the Retreat and Adjustment Phase

The methodology chapter detailed the action representation in the movement phase, and made the argument that the other phases could be represented in a similar way (Section 4.4.2). This appendix presents the action representation of the retreat and adjustment phases based on the notation used for the movement phase.

The retreat phase can be seen as identical to the movement phase, except that not all units are orderable, and that the orders available are retreat and disband orders. This difference has no effect on the operation of the agent, as it simply chooses a sequence of legal orders, regardless of which units it is requested to issue units for and the specifics of the legal orders.

When building units in the adjustment phase, the approach of Gray *et al.* [17, Appendix A] is followed. The space of possible combinations of build orders is relatively small, and each combination is therefore represented as a "combined build" order. The agent is asked to choose a legal order for a single unit, where the legal orders are all legal combined build orders. Instead of supplying the agent with an embedding of the region of a unit (Equation 4.5), the agent is supplied an aggregation of the embeddings of all buildable regions. The game engine receives a combined build order from the agent, and "deserializes" it into individual build orders.

When removing $N$ units in the adjustment phase, the action space is modeled as the selection of an order to $N$ units, where the legal orders available to each "unit" is all possible unit disbands, minus the disband orders chosen by previous "units". This approach introduces masking to the order decoder's operation, and differs from Gray *et al.* [17] that instead re-samples order sequences when encountering duplicate disbands.



NTNU

Norwegian University of
Science and Technology