[Re] No Press Diplomacy: Modeling Multi-Agent Gameplay

Daniel Ritter
Department of Computer Science
Brown University
Providence, RI 02912
`daniel_ritter@brown.edu`

Dylan Sam
Department of Computer Science
Brown University
Providence, RI 02912
`dylan_sam@brown.edu`

Kevin Du
Department of Computer Science
Brown University
Providence, RI 02912
`kevin_du@brown.edu`

Shamay G Samuel
Department of Computer Science
Brown University
Providence, RI 02912
`shamay_samuel@brown.edu`

Cody West
Department of Computer Science
Brown University
Providence, RI 02912
`cody_west@brown.edu`

Aaron Zhang
Department of Computer Science
Brown University
Providence, RI 02912
`aaron_zhang2@brown.edu`

1 Introduction

Diplomacy is a strategic board game where different powers battle over control of supply centers in Europe. The original authors [1] developed supervised learning and reinforcement learning models to learn to play the No Press version of Diplomacy, beating the existing state of the art rule-based bots.

The original paper utilizes various different machine and reinforcement learning techniques, including attention, encoder and decoder blocks, graph convolutional networks (GCN), LSTM, and FiLM [2].

Their implementation and code built off of extensive existing software frameworks like DAIDE [3], developed by the Diplomacy research community for interfacing with other bots. Furthermore, the authors have also developed a game engine that provides a simple interface for playing Diplomacy games.

Because the authors of the paper released all their code for their models, the paper is not not entirely comprehensive with their implementation details. Without being able to refer to their code, these ambiguities proved to make replication fairly difficult. We relied on communication with the paper authors in order to resolve a variety of ambiguities.

2 Background

2.1 Diplomacy

Diplomacy is a strategic board game where two to seven powers battle for ownership of various supply centers in Europe. The board consists of 75 provinces, which can be land, coast, or water. 37 of these 75 provinces are the supply centers. Each power has a number of units that are either fleets or armies that can be ordered to move, support, hold, or convoy. Diplomacy progresses as a sequence of phases, where each phase is either a movement, retreat, or adjustment phase; the allowed orders depend on the type of phase. During each phase, every player submits an order for each of their units, and these orders are executed simultaneously. Additionally, given the set of submitted orders on a given phase, the results of these orders is deterministic. While a submitted order may have a different outcome based on an order submitted by another player, there is no randomness in the game itself.

A crucial part of Diplomacy is the negotiation and disagreements between human players. However, there are variants of the game that are “No-Press", where no communication is allowed. Therefore, alliances and agreements are created solely through actions on the board. The authors further “propose Diplomacy as a new multi-agent benchmark for dynamic cooperation emergence in a rich environment".

2.2 SL DipNet

The Supervised DipNet architecture is essentially a modified encoder-decoder architecture, with multiple other features such as GCN and FiLM.

The encoder is meant to output a summary representation of the input board state and the orders from the previous phase. The encoder architecture consists of blocks, which are a GCN layer, batch normalization, a FiLM layer, and ReLU. The GCN layer multiplies the inputs by an adjacency matrix that encodes which provinces are adjacent to one another on the board and feeds the resulting graph through a linear layer. Each encoder block then applies batch normalization.

The output of the GCN and batch normalization is passed to the FiLM layer, which creates parameters to linearly shift the outputs, conditioned on the current season of the game and which power the agent plays as. The encoder is N of these blocks in sequence (N=16 in the paper and in our code). There are also residual connections between the encoder blocks, except for the case where one encoder block might have a different output shape from another(e.g. differently sized dense layers), in which case the residual connection is left out.

The decoder consists of an LSTM layer that sequentially receives a concatentation of the previous taken order and the province embeddings that can receive orders from the agent’s power. The outputs of the LSTM are passed through a linear layer and masked softmax to compute a probability distribution over the possible valid orders. The paper implements teacher forcing and beam search to improve the results of the decoder model.

In testing, against 6 RandomPlayer opponents, the SL DipNet model won 100% of the time (out of 1000 games). Against 6j the state-of-the-art **Albert Level 0** bots, SL DipNet won 28.9% of the total 208 games.

2.3 RL DipNet

RL DipNet is based on an A2C architecture. Specifically, in this paper the A2C architecture uses 15-step returns for approximately 20,000 updates (approx. 1 million steps). The “actor" follows the same model as SL DipNet (and uses SL DipNet for pretraining). The “critic" predicts the value of each state and is briefly mentioned to be “pre-trained on human games by predicting the final rewards". Paquette et al. construct a reward function that is “the average of (1) a local reward function (+1/-1 when a supply center is gained or lost (updated every phase and not just in Winter)), and (2) a terminal reward function (for a solo victory, the winner gets 34 points; for a draw, the 34 points are divided proportionally to the number of supply centers)."

RL DipNet was only tested againt SL DipNet. In testing, against 6 SL DipNet opponents, the RL DipNet model won 14% of the time (out of 1000 games).

3 Implementation Details

This section describes our replication attempt of the SL DipNet and RL DipNet models. Our code can be found here.

3.1 Data Processing

The authors provide their training dataset of 150,000 Diplomacy games as raw JSON files. We extract the features of unit type, unit power, buildable, removable, dislodged unit type, dislodged unit power, area type, and supply center owner and convert them into board states to pass into the first set of blocks of our encoder architecture. We also convert the unit type, issuing power, order type, source power, and destination power into our previous order into the second set of blocks of the encoder architecture.

3.2 SL DipNet reproduced architecture

The replication of SL DipNet includes all the components of the SL encoder architecture described original paper. However, the decoder replication did not include teacher forcing or beam search to improve the results. We were unsure how to incorporate teacher forcing and only found out that the authors used beam search through email correspondence; it was not originally mentioned in their paper.

3.3 RL DipNet reproduced architecture

Our replication of RL DipNet has all of the components of the RL architecture described in the paper, such as the value function trained on the supervised dataset and the encoder-decoder architecture matching the SL model for the actor. However, due to memory overflow issues and time constraints, we could not get the RL agent to successfully train.

3.4 Ambiguities

This section details methods that were ambiguous or unclear from the paper about implementing the SL DipNet and RL DipNet models.

3.4.1 Decoder Architecture

In the paper's experiments section, attention based on the location of current order increases performance of the supervised model. However, the paper did not include details of implementation. We originally thought that attention was a separate learnable layer, but we struggled to incorporate this into the LSTMs of the decoder. We had to contact the authors, and they told us to compute attention by indexing into the output of the encoder model by province.

The original authors use a special “GO" token as the original state of the decoder's LSTM layer, which is not mentioned in the paper.

The additional techniques of teacher forcing and beam search are not addressed in the paper. Teacher forcing is mentioned, but there are no details for implementation. Furthermore, beam search is not mentioned in the paper but is used in their implementation. In our correspondence with the original authors, they directed us to a portion of their decoder which used Beam search. Since we are not using any of the authors' original code and only looking when necessary, we were unable to implement Beam search as the authors originally have.

3.4.2 Dimensions

Because of ambiguities regarding how attention was implemented in the decoder, it was unclear what sizes inputs to the LSTM Cells were supposed to be. Since the number of orderable locations changes based on the power, the outputs of the LSTM don't have the same size. Through correspondence with the authors, it was clarified how the dimensions should be structured as including embedding of valid orders, which would then be passed into the LSTM with the attention.

Furthermore, in several equations presented by the paper, subscripts are omitted, either throughout the equation or in only one part of the equation. This creates confusion as to whether the subscript-less constants presented are the same as the subscripted constants.

3.4.3 Value function

For the value function used as a critic in RL DipNet, the paper mentions that they “used a value function pre-trained on human games by predicting the final rewards." Without more detail, we were unsure as to the training data and labels for this network and had to clarify with the authors the exact architecture of this value network.

3.4.4 Reward function

Section “6.2 Reinforcement Learning and Self-play” of the original paper describes a reward function for training *DipNet*. A small ambiguity we ran into here was that the paper describes the reward function as being “the average of (1) a local reward function (+1/-1 when a supply center is gained or lost (updated every phase and not just in Winter)), and (2) a terminal reward function (for a solo victory, the winner gets 34 points; for a draw, the 34 points are divided proportionally to the number of supply centers)”. This was confusing since local rewards are received throughout the game, while terminal reward is only received at the end of a game. It was not clear how to go about averaging the two rewards because they were not received at the same time. Follow-up with the authors clarified that the rewards are computed only after the game is completed, and that the the terminal reward is considered to be 0 for all states other than the last.

In order to simplify implementation, we decided to omit the fact that the local reward should be updated every phase and not just in Winter. In the game engine, powers receive new supply centers in the Winter. Simply assigning reward based on this made the implementation simpler, which was important given our time constraint. Given more time, we would divide up the reward over the other seasons based on the results of the game, or as a simpler alternative, interpolate the reward gained in the winter over the other seasons so that the reward in all the seasons are less sparse.

3.5 Implementation Difficulties

This section details difficulties faced when implementing and training the SL DipNet and RL DipNet models.

3.5.1 Diplomacy Game Engine

Since there is no existing implementation of the Diplomacy game, the authors create their own Diplomacy Game Engine. This is separate from the repository containing the implementations of their SL DipNet and RL DipNet models. Since many aspects of the model architecture require knowledge of the Diplomacy board game and rules, we incorporate parts of the Diplomacy Game Engine into our replication.

Section 4.3 “Decoder” in the original paper describes a “top-left to bottom-right ordering based on topological sorting, aiming to prevent jumping across the map during decoding”. This topological ordering is not listed in the paper and no instructions for creating this ordering are present. For consistency with the original implementation, we use a hardcoded ordering that is included in the Diplomacy Game Engine.

Similarly, the decoder model uses a masked softmax based on the possible orders for a given board state. The paper does not describe how to compute possible orders and no previous implementation of the Diplomacy rules exist, so we use the authors’ Game objects to compute the masks.

3.5.2 Other Players

Section 6.2 “Reinforcement Learning and Self-play" of the paper mentions the different opponents used to test the agents. These agents are briefly listed in the paper. We used the implementations of other agent types provided in the same repository as the SL DipNet and the RL DipNet models. We did not use the bot that incorporated these models and instead used the bots that were based on rulesets like random/greedy/dumb play.

3.5.3 Training time

The authors do not mention the training times and required computing resources for the SL DipNet and RL DipNet models in their paper. When contacted through email, they state their implementation of SL DipNet requires 24 hours on 1 GPU to train. Their RL DipNet model requires roughly 30 days on 4 GPUs; therefore, their RL DipNet model is very difficult to reproduce without access to a company or a university's computing resources.

Our implementation of SL DipNet struggles with efficiency and long training times because it interacts with the game engine to create the masks for the decoder model. This forces our model to train with eager execution, which is the bottleneck of our training process. We were unable to switch to graph execution, which significantly slowed our training time. We originally trained the replicated SL DipNet model 36 hours, which equates to roughly 8000 Diplomacy games; however, we encountered an error in a training that required us to discard those weights. Therefore, a combination of training errors and efficiency issues causes us to training our model on roughly 1500 games, which is 1% of the total data.

3.5.4 RL DipNet memory overflows and training time

Two main issues prevented us from training the RL DipNet: memory overflows and infeasibly long training times.

1. Memory overflows: even on a machine with 52 gigabytes of memory, the machine would run out of memory while self-playing games. We believe that the amount of memory used increases drastically with the length of the game, and experienced memory overflow after 100-200 phases (time-steps). As we are using Google Cloud Platform for compute 1, we upgraded to using a machine with 160gb of memory. Under our GCP constraints, however, this meant we could not use a machine with a GPU.
2. Training time: since one phase of self-play requires calling the actor 7 times (one for each player), self-playing even one game is estimated to take between 5-30 minutes, depending on the length of the game. As the authors trained for approximately 1 million time steps, this would take an unreasonably long time to train.

4 Results

The following table demonstrates our model's performance against 6 Random bots. Due to our time constraint, the sample size is 15 games.

4.1 Figures

Agent A (1x)	Agent B (6x)	% Win	% Most SC	% Survived	% Defeated	# Games
SL DipNet	Random	6.7%	6.7%	60.0%	26.7%	15

Table 1: Evaluating Reproduced SL DipNet against RandomBot

5 Discussion/Conclusion

Overall, the paper makes good use of diagrams for explaining the overall architecture of the models. Though the paper doesn't describe all the minute details of the implementation, it does lay out the general architecture while providing intuition for the design decisions.

The majority of the ambiguities we faced in reproducing the models were due to underspecification in the paper. Details like the type of attention used in the decoder and the dimension of the decoder's output were left unspecified, making it difficult to determine what choices to make in replication.

However, this is understandable given that all of the code is published. As such, it is likely intended that readers be able to refer to the code in order to fill in gaps in the paper itself. Being part of a complex software project, it cannot be expected that the paper describe all the intricacies of the functionality. As part of the replication challenge, we were not able to view the code, though it is clear that the paper was written with the availability of the code in mind. As such, it may have been a poor choice of paper for a replication challenge.

Additionally, while the game engine/environment is developed very well, due to the very expensive training times and resources required (e.g. 30 days on 4 GPUs), Diplomacy may be difficult as a research benchmark.

6 Future Work

Future work would follow in a number of directions:

1. We only trained our SL DipNet model for 1500 games, which is a very small portion of the training dataset. We would like to improve the efficiency of our model and the integration with the original authors' game engine so that we can train on the full dataset of roughly 150000 games to achieve better results. We would also like to train our RL DipNet model for 30 days, as the original authors' have, to better evaluate the RL results.
2. Since we were unable to incorporate teacher forcing and beam search into our decoder model, we would like to see how these added techniques affect the SL DipNet results.
3. Although the original paper performed ablation studies, we would have also liked to perform our own ablation studies (e.g. the number of blocks in the encoder).
4. With fully trained SL DipNet and RL DipNet, we would like to also try reproducing the coalition analysis described in section 6.3 of the original paper. While we wrote code to compute the $X\text{-}support\text{-}ratio$, we did not include the results here as the main purpose is to use it to support the claim that "the supervised agent was able to learn to coordinate support orders while this behaviour appears to deteriorate during self-play training." In the future, with a trained RL DipNet, it would be interesting to see how the results hold up.
5. As deep reinforcement learning is known to be very unstable, with more time, we would try training with different hyperparameters (e.g. discount rate gamma, the n-step return for the A2C agent, etc.).

Acknowledgments

The authors would like to thank Philip Paquette and the WebDiplomacy team for clarifying implementation details of the DipNet models and providing the training dataset. The authors would also like to thank Michael Littman and the CSCI 2951F staff for their mentorship and guidance.

References

[1] Philip Paquette, Yuchen Lu, Steven Bocco, Max O. Smith, Satya Ortiz-Gagné, Jonathan K. Kummerfeld, Satinder Singh, Joelle Pineau, Aaron Courville. No Press Diplomacy: Modeling Multi-Agent Gameplay. In *NeuRIPS 2019*

[2] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] David Norman. Daide - Diplomacy artificial intelligence development environment. http:www.daide.org.uk/, 2013. Accessed: 2019-11-28.