Coevolutionary Neural Population Models

Nick Moran[1] and Jordan Pollack[1]

[1]Brandeis University, 415 South St, Waltham, MA 02453
nemtiax@brandeis.edu
pollack@brandeis.edu

Abstract

We present a method for using neural networks to model evolutionary population dynamics, and draw parallels to recent deep learning advancements in which adversarially-trained neural networks engage in coevolutionary interactions. We conduct experiments which demonstrate that models from evolutionary game theory are capable of describing the behavior of these neural population systems.

Introduction

Recent works in the field of deep learning have developed algorithms which train neural networks through interaction, rather than through optimization of a static loss function. These algorithms display dynamics which are similar to those observed in evolutionary simulations and those described by evolutionary game theory. One such algorithm, Generative Adversarial Networks (GANs) by (Goodfellow et al., 2014), seeks to train a network to generate realistic images based on a training set. It uses a pair of networks trained against one another to simultaneously develop a network capable of generating images and a discriminator network capable of evaluating images for realism. Of particular interest to this work is the design of the generator network, which takes as input vectors sampled from a random distribution, and transforms them into output images. Through the lens of evolutionary simulation, the output distribution induced by the generator can be viewed as a population of candidate images which are evaluated for fitness by the discriminator. The members of this population compete with each other to occupy niches of realistic image types, and cooperate to collectively match the distribution in the training set.

We will investigate the following questions: Does embedding evolutionary population dynamics in the substrate of a neural network trained by stochastic gradient descent fundamentally alter the behavior of the system? Can existing analysis tools from the fields of evolutionary computation and artificial life predict the behavior of such systems, or explain divergence from the standard predictions? To this end, we first present a neural model to simulate a coevolutionary population over time. We then conduct experiments using this model in which we simulate well-understood matrix games, and compare the resulting behavior to that obtained from traditional simulation techniques. We thereby lay the groundwork for future investigations into the application of evolutionary game theory to more complex adversarially-trained neural networks (such as GAN models), as well as the use of such models as a testbed for artificial life simulations.

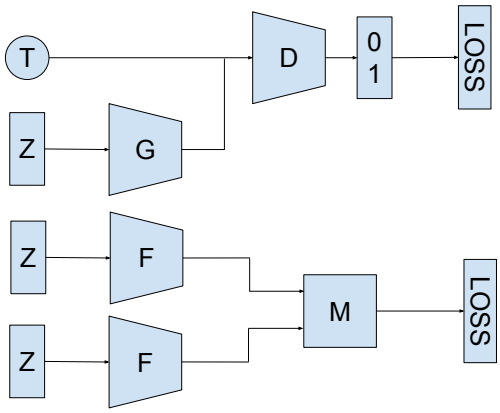

Figure 1: The layout of a standard GAN (top) and our model (bottom). In the GAN, G and D are the generator and discriminator networks, T is the set of training examples, Z is the latent input of G. In our model, Z is the latent input, F is our neural network, and M is a game matrix.

Background

There have been many intersections between the fields of evolutionary computation and neural networks. Neuroevolution algorithms such as GNARL (Angeline et al., 1994), NEAT (Stanley and Miikkulainen, 2002) and HyperNEAT (Stanley et al., 2009) employ evolutionary algorithms to evolve the structure and weights of neural networks. (Miikkulainen et al., 2017) proposes a hybrid approach in which network structure is evolved while weights are trained via

back-propagation.

Neural networks have also been used as controllers for artificial agents in evolutionary simulations, as proposed in (Harvey et al., 1992) and (Sims, 1994). Recurrent Neural Networks are evolved to control physically embodied robots in (Lipson and Pollack, 2000).

Deep reinforcement learning has applied neural models to multi-agent systems. (Bloembergen et al., 2015) gives an overview of the applicability of evolutionary models to reinforcement learning in general. In (Such et al., 2017), genetic algorithms are proposed as a method to evolve deep neural networks in reinforcement learning settings.

Generative Adversarial Networks

We present here a brief overview of the design of GAN systems as described in (Goodfellow et al., 2014). Figure 1 shows the basic layout of a GAN model (top) and our proposed model (bottom). Two neural networks, G (the generator) and D (the discriminator) are trained adversarially, with the goal of training G to produce novel realistic images. G takes as input a random vector drawn from a random distribution, Z. It transforms that vector into an output image, which is then passed to D. D takes as input images either drawn from a training set of real images, T, or the output of G, and attempts to classify them as either real (0) or fake (1). D is trained by minimizing the error of its classifications through gradient descent. G, on the other hand, is trained by maximizing the error of D on G's outputs.

We are particularly concerned with the architecture of G, as it will form the basis for our neural population model. G transforms a simple distribution, Z into a complex distribution (an approximation of the manifold of realistic images from which the samples in T are drawn). In order for G to successfully fool D, it must not only produce realistic samples, it must produce them with the proper frequencies.

Matrix Games and Evolutionary Game Theory

Matrix games have been used as a testbed for analyzing evolutionary dynamics in computer models as presented in (Maynard Smith and Price, 1973). This analysis is often focused on the discovery of evolutionarily stable strategies (ESS) (Maynard Smith, 1972), and the dynamics leading towards them. (Fogel et al., 1998) and (Ficici and Pollack, 2000) examine the ability of evolutionary models to discover and maintain an ESS using the Hawk-Dove game.

Neural Population Models

Our model is concerned with the class of matrix games defined by a fixed set of strategies S and a payoff matrix M of dimension $|S| \times |S|$ which defines the payoff received by each strategy when played against each other strategy. We will use the notation $M(s_1, s_2)$ to indicate the payoff received by strategy s_1 against s_2, where s_1 and s_2 may be pure or mixed strategies. If they are mixed strategies, $M(s_1, s_2)$ is the weighted average of payoffs received.

The goal of our model is to represent the distribution of the set of strategies in a population, which we represent as P. Instead of explicitly representing that distribution, we will construct a function F_θ (in the form of a neural network with weights θ) which transforms a simple input distribution Z to P. In this work, we let Z be a uniform distribution over $[0, 1]^n$ for small n.

Given a sample z drawn from Z, we can compute $F_\theta(z)$ to determine the strategy associated with z. For example, in a game with three possible strategies, A, B, and C, we might have $F_\theta(z) = [0.3, 0.5, 0.2]$, indicating a mixed strategy composed of $30\%A$, $50\%B$ and $20\%C$. To determine the overall composition of the population, we repeatedly sample from Z and apply F_θ to retrieve a representative sample of P.

Figure 1 shows an overview of our system, in which two sets of samples are drawn from Z, each set is passed through F, and the resulting populations interact according to payoff matrix M, resulting in a loss (or fitness) value.

Model Architecture

We represent $F_\theta(z)$ as a feed-forward neural network in which the input is passed through a fully-connected layer containing ten hidden units with sigmoid activations, followed by a second fully-connected layer with $|S|$ output units and a softmax activation. The softmax operation in the output layer has the effect of normalizing the output values to sum to one, so as to represent a valid mixed strategy in the game. The number of hidden units as well as the depth of the network are effectively hyperparameters of the model. More complex games with many strategies may require more complex networks to successfully model.

Training

To model the evolutionary trajectory of the population, we use stochastic gradient descent (SGD) to optimize the weights θ to maximize the payoffs individual samples from the population receive when matched against opponents also drawn from the population. At each step of training, we draw two sets of random samples from Z, called z_1 and z_2, which will serve as mini-batches for the SGD algorithm. z_1 and z_2 will have dimension $b \times n$, where b is the mini-batch size and n is the dimension of Z. We then compute $p_1 = F_\theta(z_1)$ and $p_2 = F_\theta(z_2)$, which have dimension $b \times s$ where s is the number of strategies in the game. We then compute fitness values $M(p_1, p_2)$ for each row of p_1 by competing it against the corresponding entry of p_2 using the game matrix.

We now have $M(F_\theta(z_1), F_\theta(z_2))$, which is a differentiable function parameterized by θ. We can therefore compute gradients for θ, and apply gradient ascent to maximize

the payoff received. Critically, we do not treat the computation of $F_\theta(z_2)$ as a differentiable component of the system. Instead, the values of $F_\theta(z_2)$ are treated as constants. This is done to prevent the system from improving the payoff of $F_\theta(z_1)$ by moving $F_\theta(z_2)$ towards a worse strategy.

Algorithm 1 Training Procedure

Require: b, the batch size, n, the dimension of Z
  for number of training iterations **do**
    Draw samples z_1 and $z_2 \in [0,1]^{b \times n}$ from Z
    Compute $F_\theta(z_1)$ and $F_\theta(z_2)$
    Compute $M(F_\theta(z_1), F_\theta(z_2))$
    Compute $\nabla_\theta \frac{1}{b} \sum M(F_\theta(z_1), F_\theta(z_2))$, treating $F_\theta(z_2)$ as a constant
    Update θ by ascending ∇_θ
  end for

Initialization

In most neural network applications, the parameters of the network are initialized to small random values. This means that the initial outputs of the network are unpredictable. However, in many evolutionary applications it is desirable to be able to initialize the model at a variety of starting configurations to observe the different trajectories that result. For example, in the Hawk-Dove-Retaliator game, Maynard Smith found two attractors, such that the fate of the population depends on the initial proportions of the three strategies.

We propose an optional period of non-adversarial training to initialize the network to output a desired starting configuration. Note that this process is only necessary if we wish to run experiments with a particular initial population in mind; it is also possible to use a random initialization, in which case no initial training is needed. Given a target distribution D, which is an n-element vector representing the desired frequencies of each of the n strategies in a game, and whose elements sum to 1, we compute the Jensen-Shannon (JS) divergence, between D and the distribution resulting from $F_\theta(z)$:

$$JS(D, F_\theta(z)) = \frac{1}{2} KLD(D||M) + \frac{1}{2} KLD(M||F_\theta(z))$$

where $M = \frac{1}{2}(D + F_\theta(z))$ is the average of the two distributions, and KLD is the discrete KullbackLeibler divergence calculated as:

$$KLD(P, Q) = \sum_{s \in strategies} P(s) log \frac{P(s)}{Q(s)}$$

The important property of the JS divergence for our purposes is that it will be minimized when $D = F_\theta(z)$. Furthermore, $JSD(D, F_\theta(z))$ is a differentiable function parameterized by θ, so we can compute gradients and use SGD to find values of θ which minimize it. We construct minibatches for SGD by samplying from z, and apply the training procedure until approximate convergence. The result is a network which outputs a distribution very close to D.

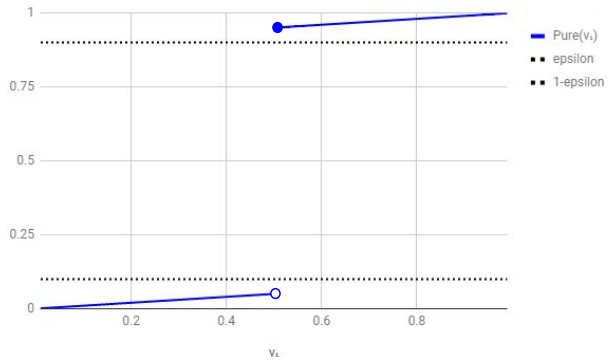

Figure 2: $Pure_\epsilon(v_1)$ with $\epsilon = 0.1$ and $max_v = 0.5$. Note that regardless of the value of max_v, $Pure_\epsilon(v_1)$ is within ϵ of 0 or 1.

Algorithm 2 Initialization Procedure

Require: b, the batch size, n, the dimension of Z and D, the desired output distribution
  while network has not converged **do**
    Draw samples $z \in [0,1]^{b \times n}$ from Z
    Compute $F_\theta(z)$
    Compute $JSD(D, F_\theta(z))$
    Compute $\nabla_\theta JSD(D, F_\theta(z))$
    Update θ by descending ∇_θ
  end while

Simulating Quasi-Pure Strategies

To this point, the model as described has been free to map an individual sample z to an arbitrary mixed strategy. However, it is sometimes desirable to examine models in which an individual must adopt a single pure strategy. This presents a potential difficulty, as our training and initialization algorithms require that network outputs be differentiable and present meaningful gradients. Simply clamping the results to 0 or 1 to enforce pure strategies would violate these constraints. The standard softmax operator is also insufficient, as it does not restrict the outputs from falling anywhere along the [0,1] range. A sigmoid or softmax operation with a high exponent can force the outputs towards 0 or 1, but will present near-zero gradients as they approximate a step function.

To address this, we introduce a soft clamp operator, which we term $Pure_\epsilon(\cdot)$. This operator takes as input a vector representing a mixed strategy, and maps the largest strategy to a

value close to one, and the other strategies to values close to zero. It does so in a way which preserves the property that the strategy values sum to one, and which presents consistent, meaningful gradients for training. The operator is parameterized by ϵ, a small value which determines how close to 0 to 1 the outputs must be. Given a mixed strategy vector v, with elements v_0 through v_n, this operator is defined as:

$$Pure_\epsilon(v_i) = \begin{cases} (1-\epsilon) + \epsilon * v_i & \text{if } \arg\max_v = i \\ \epsilon * v_i & \text{otherwise} \end{cases}$$

This operator maps the largest strategy to the range $[1-\epsilon, 1]$, and all other strategies to the range $[0, \epsilon]$, creating an output which is very close to a pure strategy. Further, the derivative of this function is a constant ϵ, allowing the training procedure to direct the network towards improvement. Figure 2 shows the behavior of the $Pure_\epsilon(\cdot)$ operator.

Experiments

We present two experiments to demonstrate the ability of a neural population model to capture the dynamics predicted by standard methods such as the replicator equation. First, we will analyze its behavior on the Hawk-Dove game (Maynard Smith, 1988). This game has been used in the past as a benchmark to analyze the ability of a proposed simulation model to discover and maintain an evolutionary stable state (ESS) (Ficici and Pollack, 2000). The Hawk-Dove game has a single ESS, and the dynamics leading towards it are straightforward. We should expect to see smooth convergence. The second experiment will focus on the ability of the neural model to capture non-convergent dynamics. We will use the noisy iterated prisoner's dilemma game, as described in (Lindgren, 1992). We will restrict ourselves to the dynamics of the four strategies of history length one: All-C, Tit-for-Tat, Anti-Tit-for-Tat, and All-D. The replicator dynamics of this game result in a continuously changing population distribution, which does not fall into a stable ESS or a simple cycle.

We will compare the behavior of these systems to the behavior shown under the time-discrete replicator equation:

$$x_i(t) = x_i(t-1) + \alpha[f_i(x(t)) - \phi(x(t))]$$

Where $x_i(t)$ is the frequency of strategy i at time t, $f_i(\cdot)$ is the fitness of strategy i given a population, and $\phi(\cdot)$ is the average fitness of a population and α is a (small) step size. We elect to use the time-discrete equation to match the discrete nature of SGD training for the neural model.

Model Hyperparameters

Neural network models are often sensitive to the settings of various hyperparameters. The details of these parameters and their values are somewhat tangential to the motivation of this work, so we will give only a brief overview of our settings. The hyperparameters of our model are: the dimension of the latent space, Z, the optimizer, batch size, and learning rate used for training, and the value of ϵ in the $Pure_\epsilon(\cdot)$ layer. For our experiments, we use the following values. Z is of dimension 10, networks are trained using the Adam optimizer (Kingma and Ba, 2014)[1] with a learning rate of 0.0002 and a batch size of 2048, and $\epsilon = 0.1$. In general, these parameters have been chosen through experimentation.

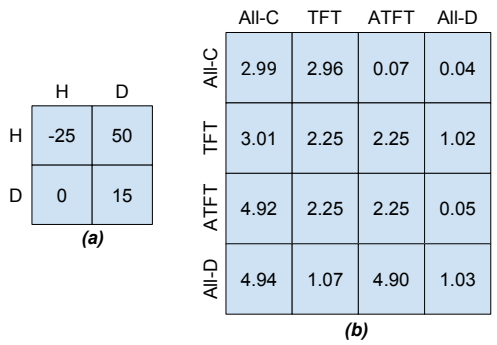

Figure 3: Payoff matrices for the hawk-dove game (a) and noisy iterated prisoners' dilemma (b).

Hawks and Doves

There are a variety of formulations of the Hawk Dove game in the literature which vary the relative payoffs received by the strategies. The fundamental dynamic is that when a hawk faces a dove, the hawk receives the highest payoff while the dove receives a small payoff (because the hawk forces the dove away), when two hawks face each other, both receive the lowest payoff (because they fight over the reward, resulting in injury), and when two doves face each other they receive a moderate payoff (because they split the reward without fighting). We will use the payoffs defined in (Ficici and Pollack, 2000)[2], shown in Figure 3. This game has a single ESS in which the population is $\frac{7}{12}$ Hawks. Any initial mix of Hawks and Doves will converge to this ESS.

We will test the neural model after initialization to a range of starting population ratios. We will present results for the 10-dimensional Z space, as well as results with a 2-dimensional Z space to allow visualization of how the network maps inputs to strategies. In order to calculate population frequencies in the neural model, we will draw 10,000 samples from Z.

Figure 4 shows the trajectory for the game under the replicator equation and the neural population model with both quasi-pure strategies and mixed strategies. Note that the

[1]The Adam optimizer is chosen because it is the standard optimizer used in GAN training (Chintala et al., 2016).

[2]Ficici et al. scale payoffs upward by 26 to make them all positive

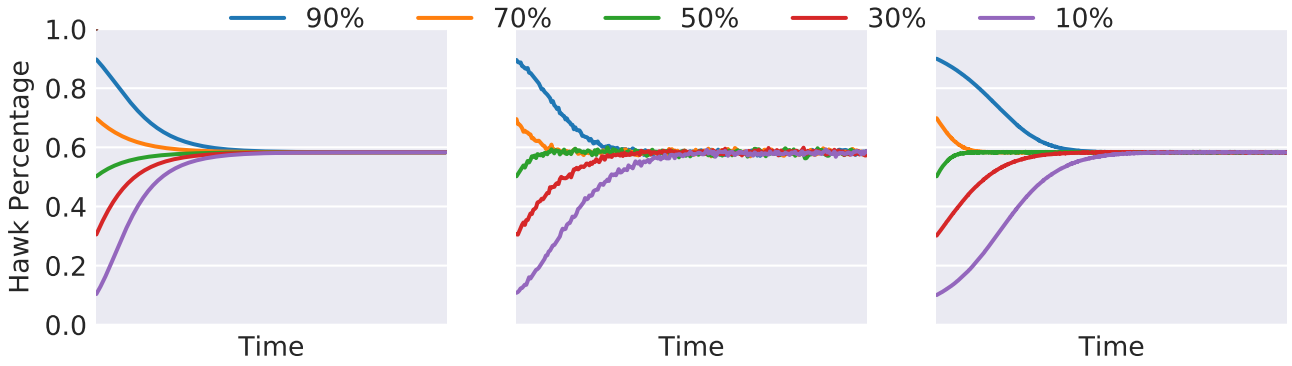

Figure 4: Trajectories of the Hawk-Dove game under the replicator equation (left) and neural population model with quasi-pure strategies (center) and mixed strategies (right). Each graph shows the portion of hawks in the population over time for five different initial population mixes.

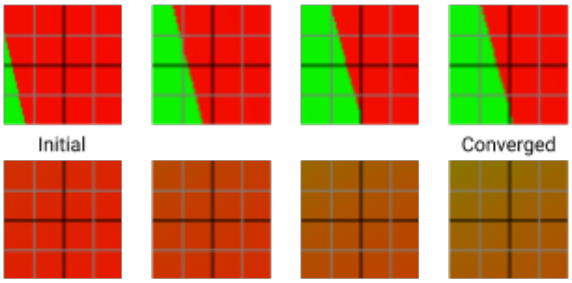

Figure 5: Mapping of a 2-dimensional Z to strategies for the Hawk-Dove game. The top row shows a simulation with quasi-pure strategies, while the bottom row shows a simulation with mixed strategies. Red represents hawks, and green represents doves. Intermediate colors represent mixed strategies.

time axes have been scaled to show similar slopes because the choice of step size is somewhat arbitrary for both models. The dynamics of the neural model are noisier, due to randomness in batch sampling and the indirect nature of the relationship between fitness values and model updates. Despite this, the model is able to converge to the ESS, and remains near it with only minor fluctuations. The choice of pure or mixed strategies does not affect the overall dynamics of the model.

Figure 5 presents church-window plots (Warde-Farley and Goodfellow, 2016) which show the way the network transforms input vectors into strategies. To enable easy visualization, we restrict the network to a 2-dimensional Z space. Because the Hawk and Dove game presents simple dynamics, this has little effect on the behavior of the model. In these plots, the x and y coordinates correspond to the values of a sampled z vector, and the color indicates which strategy the network transforms that vector to, with the red channel representing Hawks, and the green channel representing Doves. We can see that the mixed strategy network tends to produce a relatively homogeneous population, with only minor spatial variation in strategy (at convergence, the upper left, $z = (0.0, 1.0)$, gives about 53% Hawks, while the lower right, $z = (1.0, 0.0)$ gives about 68% Hawk). On the other hand, the quasi-pure network creates a strongly differentiated population with a simple boundary.

Iterated Prisoner's Dilemma

We will use the payoffs for the noisy iterated prisoner's dilemma as defined by (Lindgren, 1992), as shown in Figure 3(c). The matrix reflects the average payoffs for a game between four strategies, all-C, which always cooperates, TFT (tit-for-tat), which copies the opponent's last move, ATFT (anti-tit-for-tat), which plays the opposite of the opponent's last move, and all-D, which always defects. We imagine the players playing an infinitely iterated game using the standard prisoner's dilemma payoffs, as in (Axelrod et al., 1987), with the addition of stochastic noise, which will randomly alter a player's move in a small fraction (0.01) of rounds. The derivation of the payoff values is given in (Lindgren, 1992).

Figure 6 shows the dynamics of this game as predicted by the time-discrete replicator equation given an initial population which is an equal mix of all strategies. We see that, unlike the hawk-dove game, the system is non-convergent[3], with strategies rising and falling in frequency in a non-cyclical manner.

As shown in Figure 7, we find that the neural model with quasi-pure strategies is able to approximate these dynamics, in that the strategies rise and fall in the same order, but the magnitudes and durations of the peaks are often amplified. As we see in the replicator dynamics, a population consist-

[3]Although the frequencies appear to level off near the end of the simulation, this is actually a transient period of slow change, not a final stable state.

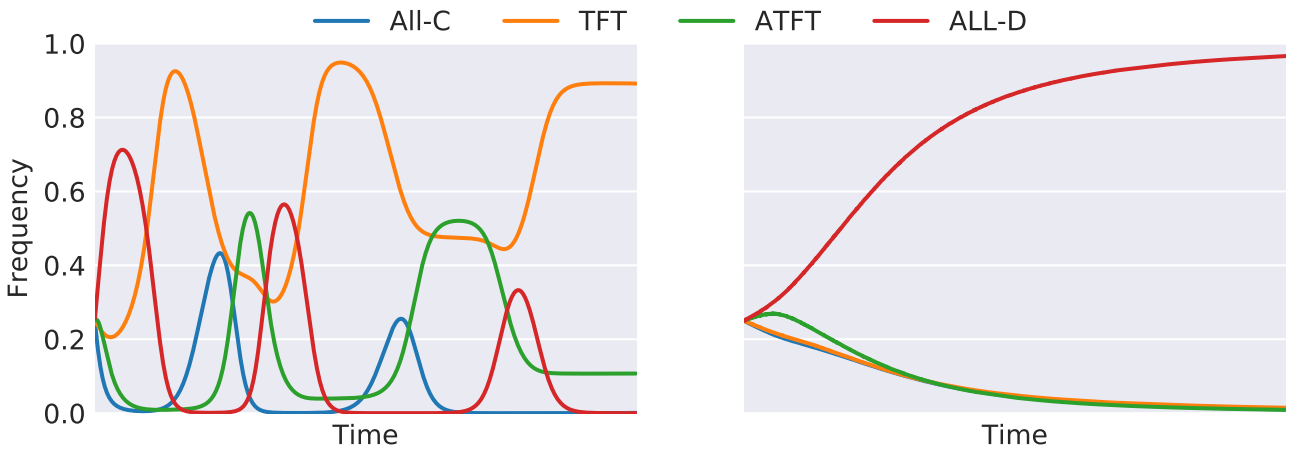

Figure 6: Dynamics of the noisy iterated prisoner's dilemma game under the time-discrete replicator equation (left) and the neural model with mixed strategies (right).

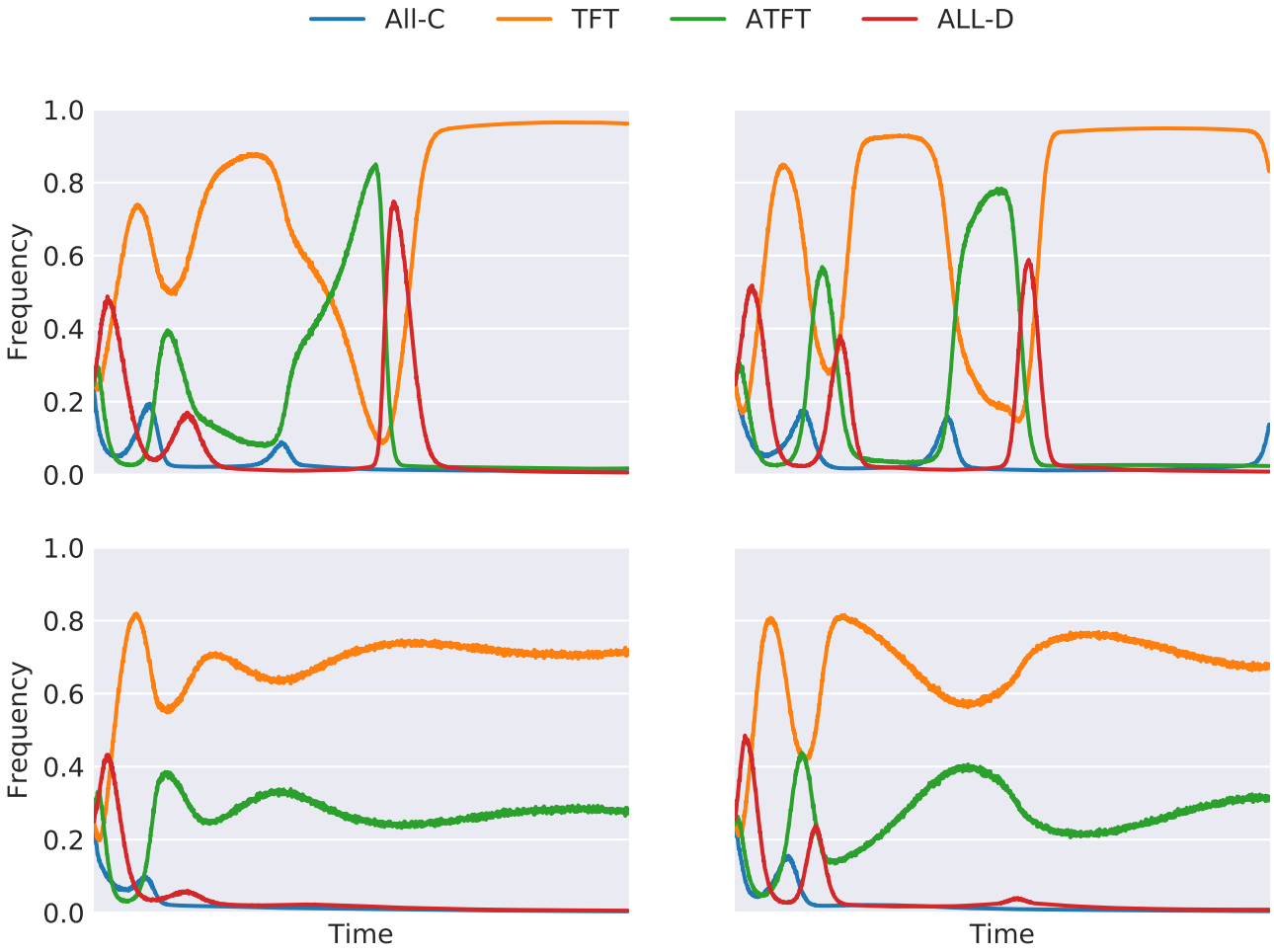

Figure 7: Dynamics of the noisy iterated prisoner's dilemma game under the neural population model with quasi-pure strategies. Four runs are shown, selected to highlight varying degrees of success.

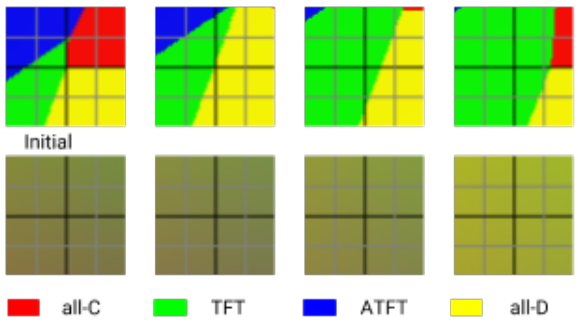

Figure 8: Mapping of a 2-dimensional Z to strategies for the noisy iterated prisoner's dilemma . The top row shows a simulation with quasi-pure strategies, while the bottom row shows a simulation with mixed strategies.

ing almost entirely of TFT and ATFT is semi-stable, and the magnified fluctuations in the neural model sometimes allow it to enter such a state earlier in the simulation, leading to long periods of minimal change.

Unlike in the Hawk-Dove game, we observe a wide variety of outcomes in the noisy iterated prisoner's dilemma, even from identical initial strategy frequencies[4]. In some sense, this behavior is expected - the game does not have a simple attractor, and so deviations from the predicted replicator dynamics can be compounded, and steer the system towards a different outcome.

On the other hand, the mixed strategy model is unable to capture these dynamics, as seen on the right side of Figure 7. The population converges steadily towards all-D, with only a slight rise in ATFT in the early phases.

Figure 8 shows the mapping of a 2-dimensional Z space to strategies for both the quasi-pure and mixed models. As in the Hawk-Dove game, the mixed model produces a relatively uniform population which smoothly converges towards its final outcome. By contrast, the quasi-pure model divides the Z space into discrete regions for each strategy, which grow and shrink and as the population distribution changes.

Discussion

Our motivation is not to present these neural models as a practical way to investigate games as simple as Hawks and Doves, but rather to demonstrate the applicability of lessons from evolutionary game theory and coevolution to understanding the dynamics of neural network systems in which loss is calculated through interaction. Ultimately, the goal of systems like GANs is to discover a Nash equilibrium using optimization tools designed for finding static minima. Our neural population models allow us to examine the suitability of these tools to this new application in a setting where we have a firm theoretical understanding of the target dynamics.

For example, even in the more successful runs, the neural model struggles to maintain the dynamics of the noisy iterated prisoner's dilemma game. As time progresses, the model swings more and more wildly towards extremes, until it enters long periods of stability in which the activations of the neural network are saturated (which in turn causes vanishingly small gradients, contributing to the stability). As we see in some of the less successful runs, this dynamic threatens to derail the simulation by prematurely removing diversity from the population (in particular by reducing the population to contain only TFT and ATFT). These problems seem to mirror problems observed in GAN training, called "mode collapse" - the generator often prematurely collapses to produce only a few types of outputs, failing to capture the diversity of the target distribution.

The very fact that there are successful and unsuccessful runs is itself a cause for concern, the only difference between these runs is the random initialization of the network and the randomly sampled z values used during training. GAN training is similarly sensitive to initial conditions (Lucic et al., 2017) (as is back-propagation in general (Kolen and Pollack, 1991)), but the problem space of image generation is far too high dimensional to allow careful analysis of the effect of parameter initialization. We propose that the low-dimensionality of neural population models makes them more amenable to detailed understanding of the effects of different initialization procedures and hyperparameter settings.

The failure of the mixed strategy model to capture complex dynamics is somewhat surprising - why is the $Pure_\epsilon(\cdot)$ operator so critical to the system? Our analysis indicates that this occurs because the network in the mixed strategy model has a tendency to couple the frequencies of different values by using the same internal weights to control multiple outputs. In particular, in the replicator dynamics and the quasi-pure neural dynamics, we see a strong differentiation between the initial trajectories of All-C and TFT. All-C drops precipitously, while TFT has only a brief initial decline. By contrast, in the mixed neural dynamics, we see All-C and TFT decline at the same rate, never diverging from each other by more than a fraction of a percentage point.

The initial separation of All-C and TFT is critical to the long-term dynamics of the system. TFT is able to outperform All-D only when the All-C strategies which All-D preys upon have been eliminated, but sufficient TFT strategies remain (because TFT performs well against itself compared to the performance of All-D against TFT). If by the time All-C is nearly eliminated, TFT is also nearly eliminated, the TFT players cannot gain enough fitness to outpace

[4]Note that identical initial frequencies does not mean an identical initialization. A homogeneous population of mixed strategies may have the same total frequencies as a heterogeneous population of pure strategies, and even two homogeneous populations may have different network weights.

the All-D players, and All-D remains the dominant strategy.

To verify that this is indeed what's happening, we examined the gradients of our loss function with respect to the four strategy outputs. As expected, we observed that the initial gradients for All-C and TFT are different - they suggest that All-C should fall much faster than TFT. However, when these gradients are propagated back to the internal weights of the network, the actual impact of the updates is to make both fall at a similar rate.

We conjecture that maintaining a diverse population, as is forced by the $Pure_\epsilon(\cdot)$ operator, is critical to preventing the degenerate behavior observed in the mixed neural model. The importance of maintaining population diversity in evolutionary algorithms is well-studied (Ursem, 2002), and Goodfellow et al. report the occasional occurrence of catastrophic loss of diversity in GAN training.

Conclusion & Future Work

We have presented a model which combines elements from the fields of deep learning and artificial life to demonstrate the potential for intellectual cross-pollination between these disciplines. Our model demonstrates the ability of neural networks to simulate population dynamics, and the applicability of evolutionary game theory results to the behavior of these networks.

Our future work will focus on extending this model with the goal of providing a unifying bridge between the two fields, in the manner of (Farmer, 1990). We will seek to analyze obstacles facing GAN training through the lens of evolutionary game theory, and seek to demonstrate the power of an individual neural network to compactly model an entire population of complex agents, as a step towards open-ended evolution.

References

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65.

Axelrod, R. et al. (1987). The evolution of strategies in the iterated prisoners dilemma. *The dynamics of norms*, pages 1–16.

Bloembergen, D., Tuyls, K., Hennes, D., and Kaisers, M. (2015). Evolutionary dynamics of multi-agent learning: A survey. *J. Artif. Intell. Res.(JAIR)*, 53:659–697.

Chintala, S., Denton, E., Arjovsky, M., and Mathieu, M. (2016). How to train a gan? tips and tricks to make gans work.

Farmer, J. D. (1990). A rosetta stone for connectionism. *Physica D: Nonlinear Phenomena*, 42(1-3):153–187.

Ficici, S. G. and Pollack, J. B. (2000). Effects of finite populations on evolutionary stable strategies. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 927–934. Morgan Kaufmann Publishers Inc.

Fogel, G. B., Andrews, P. C., and Fogel, D. B. (1998). On the instability of evolutionary stable strategies in small populations. *Ecological Modelling*, 109(3):283–294.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Harvey, I., Husbands, P., Cliff, D., et al. (1992). *Issues in evolutionary robotics*. School of Cognitive and Computing Sciences, University of Sussex.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kolen, J. F. and Pollack, J. B. (1991). Back propagation is sensitive to initial conditions. In *Advances in neural information processing systems*, pages 860–867.

Lindgren, K. (1992). Evolutionary phenomena in simple dynamics. In *Artificial life II*, pages 295–312.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2017). Are gans created equal? a large-scale study. *arXiv preprint arXiv:1711.10337*.

Maynard Smith, J. (1972). Game theory and the evolution of fighting. *On evolution*, pages 8–28.

Maynard Smith, J. (1988). Evolution and the theory of games. In *Did Darwin Get It Right?*, pages 202–215. Springer.

Maynard Smith, J. and Price, G. R. (1973). The logic of animal conflict. *Nature*, 246(5427):15.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., and Hodjat, B. (2017). Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*.

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.

Ursem, R. K. (2002). Diversity-guided evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 462–471. Springer.

Warde-Farley, D. and Goodfellow, I. (2016). 11 adversarial perturbations of deep neural networks. *Perturbations, Optimization, and Statistics*, page 311.