# Opponent Modeling in Deep Reinforcement Learning

**He He** HHE@UMIACS.UMD.EDU
University of Maryland, College Park, MD 20740 USA

**Jordan Boyd-Graber** JORDAN.BOYD.GRABER@COLORADO.EDU
University of Colorado, Boulder, CO 80309 USA

**Kevin Kwok** KKWOK@MIT.EDU
Massachusetts Institute of Technology, Cambridge, MA 02139 USA

**Hal Daumé III** HAL@UMIACS.UMD.EDU
University of Maryland, College Park, MD 20740 USA

## Abstract

Opponent modeling is necessary in multi-agent settings where secondary agents with competing goals also adapt their strategies, yet it remains challenging because strategies interact with each other and change. Most previous work focuses on developing probabilistic models or parameterized strategies for specific applications. Inspired by the recent success of deep reinforcement learning, we present neural-based models that jointly learn a policy and the behavior of opponents. Instead of explicitly predicting the opponent's action, we encode observation of the opponents into a deep Q-Network (DQN); however, we retain explicit modeling (if desired) using multitasking. By using a Mixture-of-Experts architecture, our model automatically discovers different strategy patterns of opponents without extra supervision. We evaluate our models on a simulated soccer game and a popular trivia game, showing superior performance over DQN and its variants.

## 1. Introduction

An intelligent agent working in strategic settings (e.g., collaborative or competitive tasks) must predict the action of other agents and reason about their intentions. This is important because all active agents affect the state of the world. For example, a multi-player game AI can exploit suboptimal players if it can predict their bad moves; a negotiating agent can reach an agreement faster if it knows the other party's bottom line; a self-driving car must avoid accidents by predicting where cars and pedestrians are going. Two critical questions in opponent modeling are what variable(s) to model and how to use the predicted information. However, the answers depend much on the specific application, and most previous work (Billings et al., 1998a; Southey et al., 2005; Ganzfried & Sandholm, 2011) focuses exclusively on poker games which require substantial domain knowledge.

We aim to build a general opponent modeling framework in the reinforcement learning setting, which enables the agent to exploit idiosyncrasies of various opponents. First, to account for changing behavior, we model uncertainty in the opponent's strategy instead of classifying it into a set of stereotypes. Second, domain knowledge is often required when prediction of the opponents are separated from learning the dynamics of the world. Therefore, we jointly learn a policy and model the opponent probabilistically.

We develop a new model, DRON (Deep Reinforcement Opponent Network), based on the recent deep Q-Network of Mnih et al. (2015, DQN) in Section 3. DRON has a policy learning module that predicts Q-values and an opponent learning module that infers opponent strategy.[1] Instead of explicitly predicting opponent properties, DRON learns hidden representation of the opponents based on past observations and uses it (in addition to the state information) to compute an adaptive response. More specifically, we propose two architectures, one using simple concatenation to combine the two modules and one based on the Mixture-of-Experts network. While we model opponents implicitly, additional supervision (e.g., the action or strategy taken)

*Proceedings of the $33^{rd}$ International Conference on Machine Learning*, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

[1] Code and data: https://github.com/hhexiy/opponent

can be added through multitasking. Compared to previous models that are specialized in particular applications, DRON is designed with a general purpose and does not require knowledge of possible (parameterized) game strategies.

A second contribution is DQN agents that learn in multi-agent settings. Deep reinforcement learning has shown competitive performance in various tasks: arcade games (Mnih et al., 2015), object recognition (Mnih et al., 2014), and robot navigation (Zhang et al., 2015). However, it has been mostly applied to the single-agent decision-theoretic settings with stationary environments. One exception is Tampuu et al. (2015), where two agents controlled by independent DQNs interact under collaborative and competitive rewards. While their focus is the collective behavior of a multi-agent system with known controllers, we study from the view point of a single agent that must learn a reactive policy in a stochastic environment filled with *unknown* opponents.

We evaluate our method on two tasks in Section 4: a simulated two-player soccer game in a grid world, and a real question-answering game, quiz bowl, against users playing online. Both games have opponents with a mixture of strategies that require different counter-strategies. Our model consistently achieves better results than the DQN baseline. In addition, we show our method is more robust to non-stationary strategies; it successfully identifies the opponent's strategy and responds correspondingly.

## 2. Deep Q-Learning

Reinforcement learning is commonly used for solving Markov-decision processes (MDP), where an agent interacts with the world and collects rewards. Formally, the agent takes an action $a$ in state $s$, goes to the next state $s'$ according to the transition probability $\mathcal{T}(s,a,s') = Pr(s'|s,a)$ and receives reward $r$. States and actions are defined by the state space $\mathcal{S}$ and the action space $\mathcal{A}$. Rewards $r$ are assigned by a real-valued reward function $\mathcal{R}(s,a,s')$. The agent's behavior is defined by a policy $\pi$ such that $\pi(a|s)$ is the probability of taking action $a$ in state $s$. The goal of reinforcement learning is to find an optimal policy $\pi^*$ that maximizes the expected discounted cumulative reward $R = \mathbb{E}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$, where $\gamma \in [0,1]$ is the discount factor and $T$ is the time step when the episode ends.

One approach to solve MDPs is to compute its $Q$-function: the expected reward starting from state $s$, taking action $a$ and following policy $\pi$: $Q^\pi(s,a) \equiv \mathbb{E}\left[\sum_t \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$. Q-values of an optimal policy solve the Bellman Equation (Sutton & Barto, 1998):

$$Q^*(s,a) = \sum_{s'} \mathcal{T}(s,a,s')\left[r + \gamma \max_{a'} Q^*(s',a')\right].$$

Optimal policies always select the action with the highest Q-value for a given state. Q-learning (Watkins & Dayan, 1992; Sutton & Barto, 1998) finds the optimal Q-values without knowledge of $\mathcal{T}$. Given observed transitions $(s,a,s',r)$, Q-values are updated recursively:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right].$$

For complex problems with continuous states, the $Q$-function cannot be expressed as a lookup table, requiring a continuous approximation. Deep reinforcement learning such as DQN (Mnih et al., 2015)—a deep Q-learning method with experience replay—approximates the $Q$-function using a neural network. It draws samples $(s,a,s',r)$ from a replay memory $M$, and the neural network predicts $Q^*$ by minimizing squared loss at iteration $i$:

$$L^i(\theta^i) = \mathbb{E}_{(s,a,s',r)\sim U(M)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta^{i-1}) - Q(s,a;\theta^i)\right)^2\right],$$

where $U(M)$ is a uniform distribution over replay memory.

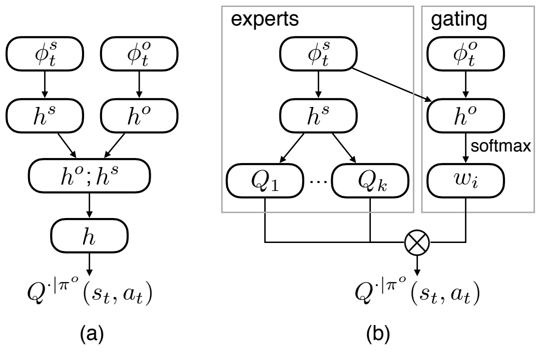


*Figure 1.* Diagram of the DRON architecture. (a) DRON-concat: opponent representation is concatenated with the state representation. (b) DRON-MoE: Q-values predicted by $K$ experts are combined linearly by weights from the gating network.

## 3. Deep Reinforcement Opponent Network

In a multi-agent setting, the environment is affected by the *joint* action of all agents. From the perspective of one agent, the outcome of an action in a given state is no longer stable, but is dependent on actions of other agents. In this section, we first analyze the effect of multiple agents on the Q-learning framework; then we present DRON and its multitasking variation.

### 3.1. Q-Learning with Opponents

In MDP terms, the joint action space is defined by $\mathcal{A}^M = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n$ where $n$ is the total number of

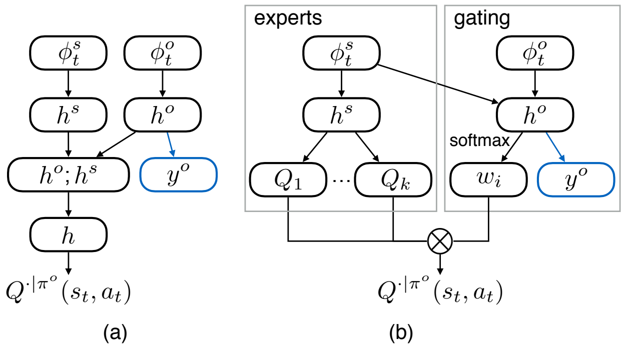


*Figure 2.* Diagram of the DRON with multitasking. The blue part shows that the supervision signal from the opponent affects the Q-learning network by changing the opponent features.

agents. We use $a$ to denote the action of the agent we control (the primary agent) and $o$ to denote the joint action of all other agents (secondary agents), such that $(a, o) \in \mathcal{A}^M$. Similarly, the transition probability becomes $\mathcal{T}^M(s, a, o, s') = Pr(s'|s, a, o)$, and the new reward function is $\mathcal{R}^M(s, a, o, s')$. Our goal is to learn an optimal policy for the primary agent given interactions with the joint policy $\pi^o$ of the secondary agents.[2]

If $\pi^o$ is stationary, then the multi-agent MDP reduces to a single-agent MDP: the opponents can be considered part of the world. Thus, they redefine the transitions and reward:

$$\begin{aligned}\mathcal{T}(s, a, s') &= \sum_o \pi^o(o|s)\mathcal{T}^M(s, a, o, s'),\\ \mathcal{R}(s, a, s') &= \sum_o \pi^o(o|s)\mathcal{R}^M(s, a, o, s').\end{aligned}$$

Therefore, an agent can ignore other agents, and standard Q-learning suffices.

Nevertheless, it is often unrealistic to assume opponents use fixed policies. Other agents may also be learning or adapting to maximize rewards. For example, in strategy games, players may disguise their true strategies at the beginning to fool the opponents; winning players protect their lead by playing defensively; and losing players play more aggressively. In these situations, we face opponents with an unknown policy $\pi_t^o$ that changes over time.

Considering the effects of other agents, the definition of an optimal policy in Section 2 no longer applies—the effectiveness policies now depends on policies of secondary agents. We therefore define the optimal $Q$-function relative to the joint policy of opponents: $Q^{*|\pi^o} = \max_\pi Q^{\pi|\pi^o}(s, a)\ \forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$. The recurrent relation between Q-values holds:

$$Q^{\pi|\pi^o}(s_t, a_t) = \sum_{o_t} \pi_t^o(o_t|s_t) \sum_{s_{t+1}} \mathcal{T}(s_t, a_t, o_t, s_{t+1}) \Big[\mathcal{R}(s_t, a_t, o_t, s_{t+1}) + \gamma \mathbb{E}_{a_{t+1}} \Big[Q^{\pi|\pi^o}(s_{t+1}, a_{t+1})\Big]\Big]. \quad (1)$$

[2]While a joint policy defines the distribution of joint actions, the opponents may be controlled by independent policies.

## 3.2. DQN with Opponent Modeling

Given Equation 1, we can continue applying Q-learning and estimate both the transition function and the opponents' policy by stochastic updates. However, treating opponents as part of the world can slow responses to adaptive opponents (Uther & Veloso, 2003), because the change in behavior is masked by the dynamics of the world.

To encode opponent behavior explicitly, we propose the Deep Reinforcement Opponent Network (DRON) that models $Q^{\cdot|\pi^o}$ and $\pi^o$ jointly. DRON is a Q-Network ($N_Q$) that evaluates actions for a state and an opponent network ($N_o$) that learns representation of $\pi^o$. The remaining questions are how to combine the two networks and what supervision signal to use. To answer the first question, we investigate two network architectures: DRON-concat that concatenates $N_Q$ and $N_o$, and DRON-MOE that applies a Mixture-of-Experts model. To answer the second question, we consider two settings: (a) predicting Q-values only, as our goal is the best reward instead of accurately simulating opponents; and (b) also predicting extra information about the opponent when it is available, e.g., the type of their strategy.

**DRON-concat** We extract features from the state ($\phi^s$) and the opponent ($\phi^o$ ) and then use linear layers with rectification or convolutional neural networks—$N_Q$ and $N_o$—to embed them in separate hidden spaces ($h^s$ and $h^o$). To incorporate knowledge of $\pi^o$ into the Q-Network, we concatenate representations of the state and the opponent (Figure 1a). The concatenation then jointly predicts the Q-value. Therefore, the last layer(s) of the neural network is responsible for understanding the interaction between opponents and Q-values. Since there is only one Q-Network, the model requires a more discriminative representation of the opponents to learn an adaptive policy. To alleviate this, our second model encodes a stronger prior of the relation between opponents' actions and Q-values based on Equation 1.

**DRON-MOE** The right part of Equation 1 can be written as $\sum_{o_t} \pi_t^o(o_t|s_t)Q^\pi(s_t, a_t, o_t)$, an expectation over different opponent behavior. We use a Mixture-of-Experts network (Jacobs et al., 1991) to explicitly model the opponent action as a hidden variable and marginalize over it (Figure 1b). The expected Q-value is obtained by combining predictions from multiple *expert networks*:

$$\begin{aligned} Q(s_t, a_t; \theta) &= \sum_{i=1}^{K} w_i Q_i(h^s, a_t) \\ Q_i(h^s, \cdot) &= f(W_i^s h^s + b_i^s). \end{aligned}$$

Each expert network predicts a possible reward in the current state. A *gating network* based on the opponent representation computes combination weights (distribution over experts):

$$w = \text{softmax}\left(f(W^o h^o + b^o)\right).$$

Here $f(\cdot)$ is a nonlinear activation function (ReLU for all experiments), $W$ represents the linear transformation matrix, and $b$ is the bias term.

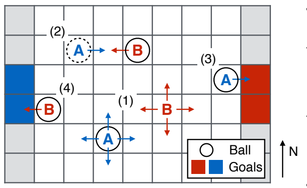
Unlike DRON-concat, which ignores the interaction between the world and opponent behavior, DRON-MOE knows that Q-values have different distributions depending on $\phi^o$; each expert network captures one type of opponent strategy.

**Multitasking with DRON** The previous two models predict Q-values only, thus the opponent representation is learned indirectly through feedback from the Q-value. Extra information about the opponent can provide direct supervision for $N_o$. Many games reveal additional information besides the final reward at the end of a game. At the very least the agent has observed actions taken by the opponents in past states; sometimes their private information such as the hidden cards in poker. More high-level information includes abstracted plans or strategies. Such information reflects characteristics of opponents and can aid policy learning.

Unlike previous work that learns a separate model to predict these information about the opponent (Davidson, 1999; Ganzfried & Sandholm, 2011; Schadd et al., 2007), we apply multitask learning and use the observation as extra supervision to learn a *shared* opponent representation $h^o$. Figure 2 shows the architecture of multitask DRON, where supervision is $y^o$. The advantage of multitasking over explicit opponent modeling is that it uses high-level knowledge of the game and the opponent, while remaining robust to insufficient opponent data and modeling error from Q-values. In Section 4, we evaluate multitasking DRON with two types of supervision signals: future action and overall strategy of the opponent.

## 4. Experiments

In this section, we evaluate our models on two tasks, the soccer game and quiz bowl. Both tasks have two players against each other and the opponent presents varying behavior. We compare DRON models with DQN and analyze their response against different types of opponents.



| | Defensive | Offensive |
|---|---|---|
| w/ ball | Avoid opponent | Advance to goal |
| w/o ball | Defend goal | Intercept the ball |

*Figure 3. Left:* Illustration of the soccer game. *Right:* Strategies of the hand-crafted rule-based agent.

All systems are trained under the same Q-learning framework. Unless stated otherwise, the experiments have the following configuration: discount factor $\gamma$ is 0.9, parameters are optimized by AdaGrad (Duchi et al., 2011) with a learning rate of 0.0005, and the mini-batch size is 64. We use $\epsilon$-greedy exploration during training, starting with an exploration rate of 0.3 that linearly decays to 0.1 within 500,000 steps. We train all models for fifty epochs. Cross Entropy is used as the loss in multitasking learning.

### 4.1. Soccer

Our first testbed is a soccer variant following previous work on multi-player games (Littman, 1994; Collins, 2007; Uther & Veloso, 2003). The game is played on a $6 \times 9$ grid (Figure 3) by two players, A and B.[3] The game starts with A and B in a randomly squares in the left and right half (except the goals), and the ball goes to one of them. Players choose from five actions: `move N, S, W, E` or `stand` still (Figure 3(1)). An action is invalid if it takes the player to a shaded square or outside of the border. If two players move to the same square, the player who possesses the ball before the move loses it to the opponent (Figure 3(2)), and the move does not take place. A player scores one point if they take the ball to the opponent's goal (Figure 3(3), (4)) and the game ends. If neither player gets a goal within one hundred steps, the game ends with a zero–zero tie.

**Implementation** We design a two-mode rule-based agent as the opponent Figure 3(right). In the offensive mode, the agent always prioritize attacking over defending. In 5000 games against a random agent, it wins 99.86% of the time and the average episode length is 10.46. In defensive mode, the agent only focuses on defending its own goal. As a result, it wins 31.80% of the games and ties 58.40% of them; the average episode length is 81.70. It is easy to find a strategy to defeat the opponent in either mode, however, the strategy does not work well for both modes, as we will show in Table 2. Therefore, the agent randomly chooses between the two modes in each game to create a varying strategy.

[3] Although the game is played in a grid world, we do not represent the $Q$-function in tabular form as in previous work. Therefore it can be generalized to more complex pixel-based settings.

| Model | Basic | Multitask +action | Multitask +type |
|---|---|---|---|
| | Max $R$ | | |
| DRON-concat | 0.682 | 0.695* | 0.690* |
| DRON-MOE | **0.699*** | 0.697* | 0.686* |
| DQN-world | 0.664 | - | - |
| | Mean $R$ | | |
| DRON-concat | 0.660 | 0.672 | 0.669 |
| DRON-MOE | 0.675 | 0.664 | 0.672 |
| DQN-world | 0.616 | - | - |

*Table 1.* Rewards of DQN and DRON models on Soccer. We report the maximum test reward ever achieved (Max $R$) and the average reward of the last 10 epochs (Mean $R$). Statistically significant ($p < 0.05$ in two-tailed pairwise $t$-tests) improvement for DQN (*) and all other models in **bold**. DRON models achieve higher rewards in both measures.

The input state is a $1 \times 15$ vector representing coordinates of the agent, the opponent, the axis limits of the field, positions of the goal areas and ball possession. We define a player's move by five cases: approaching the agent, avoiding the agent, approaching the agent's goal, approaching self goal and standing still. Opponent features include frequencies of observed opponent moves, its most recent move and action, and the frequency of losing the ball to the opponent.

The baseline DQN has two hidden layers, both with 50 hidden units. We call this model DQN-world: opponents are modeled as part of the world. The hidden layer of the opponent network in DRON also has 50 hidden units. For multitasking, we experiment with two supervision signals, opponent action in the current state (+action) and the opponent mode (+type).

**Results** In Table 1, we compare rewards of DRON models, their multitasking variations, and DQN-world. After each epoch, we evaluate the policy with 5000 randomly generated games (the test set) and compute the average reward. We report the mean test reward after the model stabilizes and the maximum test reward ever achieved. The DRON models outperform the DQN baseline. Our model also has much smaller variance (Figure 4).

Adding additional supervision signals improves DRON-concat but not DRON-MOE (multitask column). DRON-concat does not explicitly learn different strategies for different types of opponents, therefore more discriminative opponent representation helps model the relation between opponent behavior and Q-values. However, for DRON-MOE, while better opponent representation is still desirable, the supervision signal may not be aligned with "classification" of the opponents learned from the Q-values.

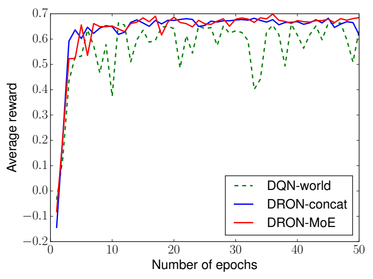


*Figure 4.* Learning curves on Soccer over fifty epochs. DRON models are more stable than DQN.

| | DQN O only | DQN D only | DQN -world | DRON -concat | DRON -MOE |
|---|---|---|---|---|---|
| O | **0.897** | -0.272 | 0.811 | 0.875 | 0.870 |
| D | 0.480 | **0.504** | 0.498 | 0.493 | 0.486 |

*Table 2.* Average rewards of DQN and DRON models when playing against different types of opponents. Offensive and defensive agents are represented by O and D. "O only" and "D only" means training against O and D agents only. Upper bounds of rewards are in bold. DRON achieves rewards close to the upper bounds against both types of opponents.

To investigate how the learned policies adapt to different opponents, we test the agents against a defensive opponent and an offensive opponent separately. Furthermore, we train two DQN agents targeting at each type of opponent respectively. Their performance is best an agent can do when facing a single type of opponent (in our setting), as the strategies are learned to defeat this particular opponent. Table 2 shows the average rewards of each model and the DQN upper bounds (in bold). DQN-world is confused by the defensive behavior and significantly sacrifices its performance against the offensive opponent; DRON achieves a much better trade-off, retaining rewards close to both upper bounds against the varying opponent.

Finally, we examine how the number of experts in DRON-MOE affects the result. From Figure 5, we see no significant difference in varying the number of experts, and DRON-MOE consistently performs better than DQN across all $K$. Multitasking does not help here.

## 4.2. Quiz Bowl

Quiz bowl is a trivia game widely played in English-speaking countries between schools, with tournaments held most weekends. It is usually played between two teams. The questions are read to players and they score points by "buzzing in" first (often before the question is finished) and answering the question correctly. One example question with buzzes is shown in Figure 7. A successful quiz bowl

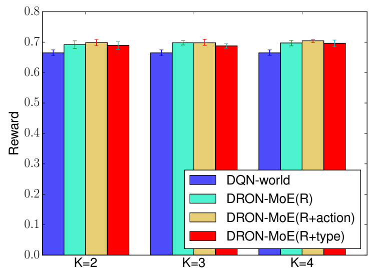


*Figure 5.* Effect of varying the number experts (2–4) and multitasking on Soccer. The error bars show the 90% confidence interval. DRON-MOE consistently improves over DQN regardless of the number of mixture components. Adding extra supervision does not obviously improve the results.

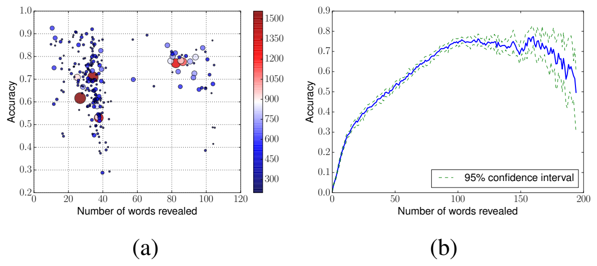


*Figure 6.* Accuracy vs. the number of words revealed. (a) Real-time user performance. Each dot represents one user; dot size and color correspond to the number of questions the user answered. (b) Content model performance. Accuracy is measured based on predictions at each word. Accuracy improves as more words are revealed.

player needs two things: a content model to predict answers given (partial) questions and a buzzing model to decide when to buzz.

**Content Model** We model the question answering part as an incremental text-classification problem. Our content model is a recurrent neural network with gated recurrent units (GRU). It reads in the question sequentially and outputs a distribution over answers at each word given past information encoded in the hidden states.

**Buzzing Model** To test depth of knowledge, questions start with obscure information and reveals more and more obvious clues towards the end (e.g., Figure 7). Therefore, the buzzing model faces a speed-accuracy tradeoff: while buzzing later increases one's chance of answering correctly, it also increases the risk of losing the chance to answer. A safe strategy is to always buzz as soon as the content model is confident enough. A smarter strategy, however, is to adapt to different opponents: if the opponent often buzzes late, wait for more clues; otherwise, buzz more aggressively.

To model interaction with other players, we take a reinforcement learning approach to learn a buzzing policy. The state includes words revealed and predictions from the content model, and the actions are `buzz` and `wait`. Upon buzzing, the content model outputs the most likely answer at the current position. An episode terminates when one player buzzes and answers the question correctly. Correct answers are worth 10 points and wrong answers are $-5$.

**Implementation** We collect question/answer pairs and log user buzzes from Protobowl, an online multi-player quizbowl application.[4] Additionally, we include data from Boyd-Graber et al. (2012). Most buzzes are from strong tournament players. After removing answers with fewer than five questions and users who played fewer than twenty questions, we end up with 1045 answers, 37.7k questions and 3610 users. We divide all questions into two non-overlapping sets: one for training the content model and one for training the buzzing policy. The two sets are further divided into train/dev and train/dev/test sets randomly. There are clearly two clusters of players (Figure 6(a)): aggressive players who buzz early with varying accuracies and cautious players who buzz late but maintain higher accuracy. Our GRU content model (Figure 6(b)) is more accurate with more input words—a behavior similar to human players.

Our input state must represent information from the content model and the opponents. Information from the content model takes the form of a *belief vector*: a vector $(1 \times 1045)$ with the current estimate (as a log probability) of each possible guess being the correct answer given our input so far. We concatinate the belief vector from the previous time step to capture sudden shifts in certainty, which are often good opportunities to buzz. In addition, we include the number of words seen and whether a wrong buzz has happened.

The opponent features include the number of questions the opponent has answered, the average buzz position, and the error rate. The basic DQN has two hidden layers, both with 128 hidden units. The hidden layer for the opponent has ten hidden units. Similar to soccer, we experiment with two settings for multitasking: (a) predicting how opponent buzzes; (b) predicting the opponent type. We approximate the ground truth for (a) by $\min(1, t/\text{buzz position})$ and use the mean square error as the loss function. The ground truth for (b) is based on dividing players into four groups according to their buzz positions—the percentage of question revealed.

**Results** In addition to DQN-world, we also compare with DQN-self, a baseline without interaction with opponents at all. DQN-self is ignorant of the opponents and plays the safe strategy: answer as soon as the content model is confident.

[4] `http://protobowl.com`

| Model | Basic | Multitask +action | Multitask +type | Basic vs. opponents buzzing at different positions (%revealed (#episodes)) 0 − 25% (4.8k) | | | 25 − 50% (18k) | | | 50 − 75% (0.7k) | | | 75 − 100% (1.3k) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R\uparrow$ | | | $R\uparrow$ | rush$\downarrow$ | miss$\downarrow$ | $R\uparrow$ | rush$\downarrow$ | miss$\downarrow$ | $R\uparrow$ | rush$\downarrow$ | miss$\downarrow$ | $R\uparrow$ | rush$\downarrow$ | miss$\downarrow$ |
| DRON-concat | 1.04 | **1.34*** | **1.25** | -0.86 | 0.06 | 0.15 | 1.65 | 0.10 | 0.11 | -1.35 | 0.13 | 0.18 | 0.81 | 0.19 | 0.12 |
| DRON-MOE | **1.29*** | 1.00 | **1.29*** | -0.46 | 0.06 | 0.15 | 1.92 | 0.10 | 0.11 | -1.44 | 0.18 | 0.16 | 0.56 | 0.22 | 0.10 |
| DQN-world | 0.95 | - | - | -0.72 | 0.04 | 0.16 | 1.67 | 0.09 | 0.12 | -2.33 | 0.23 | 0.15 | -1.01 | 0.30 | 0.09 |
| DQN-self | 0.80 | - | - | -0.46 | 0.09 | 0.12 | 1.48 | 0.14 | 0.10 | -2.76 | 0.30 | 0.12 | -1.97 | 0.38 | 0.07 |

*Table 3.* Comparison between DRON and DQN models. The left column shows the average reward of each model on the test set. The right column shows performance of the basic models against different types of players, including the average reward ($R$), the rate of buzzing incorrectly (rush) and the rate of missing the chance to buzz correctly (miss). ↑ means higher is better and ↓ means lower is better. In the left column, we indicate statistically significant results ($p < 0.05$ in two-tailed pairwise $t$-tests) with boldface for vertical comparison and $^*$ for horizontal comparison.

The antibiotic erythromycin works by disrupting this organelle , which contains E , **P** , and A sites on its large subunit . **The** parts ***of***▲ this■ organelle✤★ **are** assembled★ at nucleoli✤ , **and** when bound to■✤★ **a** membrane , these create the rough ER . Codons✤ are translated at this organelle where the tRNA and mRNA meet . For 10 points , name this organelle that is the site of protein synthesis .

▲: DQN-self ■: DQN-world ✤: DRON-MOE ★: DRON-concat

*Figure 7.* Buzz positions of human players and agents on one science question whose answer is "ribosome". Words where a player buzzes is displayed in a color unique to the player; a wrong buzz is shown in *italic*. Words where an agent buzzes is subscripted by a symbol unique to the agent; color of the symbol corresponds to the player it is playing against. A gray symbol means that buzz position of the agent does not depend on its opponent. DRON agents adjust their buzz positions according to the opponent's buzz position and correctness. Best viewed in color.

During training, when the answer prediction is correct, it receives reward 10 for `buzz` and -10 for `wait`. When the answer prediction is incorrect, it receives reward -15 for `buzz` and 15 for `wait`. Since all rewards are immediate, we set $\gamma$ to 0 for DQN-self.[5] With data of the opponents' responses, DRON and DQN-world use the game payoff (from the perspective of the computer) as the reward.

First we compare the average rewards on test set of our models—DRON-concat and DRON-MOE (with 3 experts)—and the baseline models: DQN-self and DQN-world. From the first column in Table 3, our models achieve statistically significant improvements over the DQN baselines and DRON-MOE outperforms DRON-concat. In addition, the DRON models have much less variance compared to DQN-world as the learning curves show in Figure 9.

To investigate strategies learned by these models, we show their performance against different types of players (as defined at the end of "Implementation") in Table 3, right column. We compare three measures of performance, the average reward ($R$), percentage of early and incorrect buzzes (rush), and percentage of missing the chance to buzz correctly before the opponent (miss). All models beat Type 2 players, mainly because they are the majority in our dataset. As expected, DQN-self learns a safe strategy that tends to buzz early. It performs the best against Type 1 players who answer early. However, it has very high rush rate against cautious players, resulting in much lower rewards against Type 3 and Type 4 players. Without opponent modeling, DQN-world is biased towards the majority player, thus having the same problem as DQN-self when playing against players who buzz late. Both DRON models exploit cautious players while holding their own against aggressive players. Furthermore, DRON-MOE matches DQN-self against Type 1 players, thus it discovers different buzzing strategies.

Figure 7 shows an example question with buzz positions labeled. The DRON agents demonstrate dynamic behavior against different players; DRON-MOE almost always buzzes right before the opponent in this example. In addition, when the player buzzes wrong and the game continues, DRON-MOE learns to wait longer since the opponent is gone, while the other agents are still in a rush.

As with the Soccer task, adding extra supervision does not yield better results over DRON-MOE (Table 3) but significantly improves DRON-concat. Figure 8 varies the number of experts in DRON-MOE ($K$) from two to four. Using a mixture model for the opponents consistently improves over the DQN baseline, and using three experts gives better performance on this task. For multitasking, adding the action supervision does not help at all. However, the more high-level type supervision yields competent results, especially with four experts, mostly because the number of experts matches the number of types.

[5]This is equivalent to cost-sensitive classification.

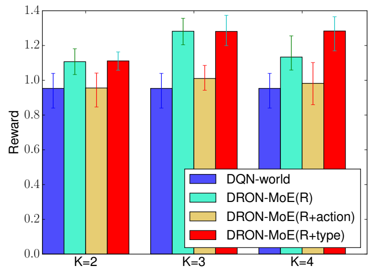


*Figure 8.* Effect of varying the number experts (2–4) and multitasking on quiz bowl. The error bars show the 90% confidence interval. DRON-MOE consistently improves over DQN regardless of the number of mixture components. Supervision of the opponent type is more helpful than the specific action taken.

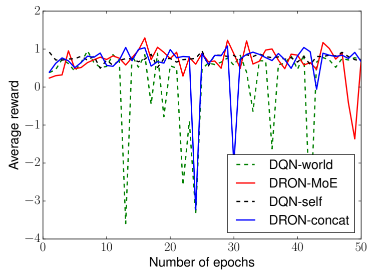


*Figure 9.* Learning curves on Quizbowl over fifty epochs. DRON models are more stable than DQN.

## 5. Related Work and Discussion

**Implicit vs. Explicit opponent modeling** Opponent modeling has been studied extensively in games. Most existing approaches fall into the category of explicit modeling, where a model (e.g., decision trees, neural networks, Bayesian models) is built to directly predict parameters of the opponent, e.g., actions (Uther & Veloso, 2003; Ganzfried & Sandholm, 2011), private information (Billings et al., 1998b; Richards & Amir, 2007), or domain-specific strategies (Schadd et al., 2007; Southey et al., 2005). One difficulty is that the model may need a prohibitive number of examples before producing anything useful. Another is that as the opponent behavior is modeled separately from the world, it is not always clear how to incorporate these predictions robustly into policy learning. The results on multitasking DRON also suggest that improvement from explicit modeling is limited. However, it is better suited to games of incomplete information, where it is clear what information needs to be predicted to achieve higher reward.

Our work is closely related to implicit opponent modeling. Since the agent aims to maximize its own expected reward without having to identify the opponent's strategy, this approach does not have the difficulty of incorporating prediction of the opponent's parameters. Rubin & Watson (2011) and Bard et al. (2013) construct a a portfolio of strategies offline based on domain knowledge or past experience for heads-up limit Texas hold'em; they then select strategies online using multi-arm bandit algorithms. Our approach does not have a clear online/offline distinction. We learn strategies and their selector in a joint, probabilistic way. However, the offline construction can be mimicked in our models by initializing expert networks with DQN pre-trained against different opponents.

**Neural network opponent models** Davidson (1999) applies neural networks to opponent modeling, where a simple multi-layer perceptron is trained as a classifier to predict opponent actions given game logs. Lockett et al. (2007) propose an architecture similar to DRON-concat that aims to identify the type of an opponent. However, instead of learning a hidden representation, they learn a mixture weights over a pre-specified set of cardinal opponents; and they use the neural network as a standalone solver without the reinforcement learning setting, which may not be suitable for more complex problems. Foerster et al. (2016) use modern neural networks to learn a group of parameter-sharing agents that solve a coordination task, where each agent is controlled by a deep recurrent Q-Network (Hausknecht & Stone, 2015). Our setting is different in that we control only one agent and the policy space of other agents is unknown. Opponent modeling with neural networks remains understudied with ample room for improvement.

## 6. Conclusion and Future Work

Our general opponent modeling approach in the reinforcement learning setting incorporates (implicit) prediction of opponents' behavior into policy learning without domain knowledge. We use recent deep Q-learning advances to learn a representation of opponents that better maximizes available rewards. The proposed network architectures are novel models that capture the interaction between opponent behavior and Q-values. Our model is also flexible enough to include supervision for parameters of the opponents, much as in explicit modeling.

These gains can further benefit from advances in deep learning. For example, Eigen et al. (2014) extends the Mixture-of-Experts network to a stacked model—deep Mixture-of-Experts—which can be combined with hierarchical reinforcement learning to learn a hierarchy of opponent strategies in large, complex domains such as online strategy games. In addition, instead of hand-crafting opponent features, we can feed in raw opponent actions and use a recurrent neural network to learn the opponent representation. Another important direction is to design online algorithms that can adapt to fast changing behavior and balance exploitation and exploration of opponents.

## Acknowledgements

We thank Hua He, Xiujun Li and Mohit Iyyer for helpful discussions about deep Q-learning and our model. We also thank the anonymous reviewers for their insightful comments. This work was supported by NSF grant IIS-1320538. Boyd-Graber is also partially supported by NSF grants CCF-1409287 and NCSE-1422492. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the view of the sponsor.

## References

Bard, Nolan, Johanson, Michael, Burch, Neil, and Bowling, Michael. Online implicit agent modelling. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, 2013.

Billings, Darse, Papp, Denis, Schaeffer, Jonathan, and Szafron, Duane. Opponent modeling in poker. In *Association for the Advancement of Artificial Intelligence*, 1998a.

Billings, Darse, Papp, Denis, Schaeffer, Jonathan, and Szafron, Duane. Opponent modeling in poker. In *Association for the Advancement of Artificial Intelligence*, 1998b.

Boyd-Graber, Jordan, Satinoff, Brianna, He, He, and Daumé III, Hal. Besting the quiz master: Crowdsourcing incremental classification games. In *Empirical Methods in Natural Language Processing*, 2012.

Collins, Brian. Combining opponent modeling and model-based reinforcement learning in a two-player competitive game. Master's thesis, School of Informatics, University of Edinburgh, 2007.

Davidson, Aaron. Using artifical neural networks to model opponents in texas hold'em. CMPUT 499 - Research Project Review, 1999. URL http://www.spaz.ca/aaron/poker/nnpoker.pdf.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

Eigen, David, Ranzato, Marc'Aurelio, and Sutskever, Ilya. Learning factored representations in a deep mixture of experts. In *ICLR Workshop*, 2014.

Foerster, Jakob N., Assael, Yannis M., de Freitas, Nando, and Whiteson, Shimon. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *Arxiv:1602.02672*, 2016.

Ganzfried, Sam and Sandholm, Tuomas. Game theory-based opponent modeling in large imperfect-information games. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, 2011.

Hausknecht, Matthew and Stone, Peter. Deep recurrent q-learning for partially observable MDPs. *Arxiv:1507.06527*, 2015.

Jacobs, Robert A., Jordan, Michael I., Nowlan, Steven J., and Hinton, Geoffrey E. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Littman, Michael L. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the International Conference of Machine Learning*, 1994.

Lockett, Alan J., Chen, Charles L., and Miikkulainen, Risto. Evolving explicit opponent models in game playing. In *Proceeedings of the Genetic and Evolutionary Computation Conference*, 2007.

Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, and Kavukcuoglu, Koray. Recurrent models of visual attention. In *Proceedings of Advances in Neural Information Processing Systems*, 2014.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

Richards, Mark and Amir, Eyal. Opponent modeling in Scrabble. In *International Joint Conference on Artificial Intelligence*, 2007.

Rubin, Jonathan and Watson, Ian. On combining decisions from multiple expert imitators for performance. In *International Joint Conference on Artificial Intelligence*, 2011.

Schadd, Frederik, Bakkes, Er, and Spronck, Pieter. Opponent modeling in real-time strategy games. In *Proceedings of the GAME-ON 2007*, pp. 61–68, 2007.

Southey, Finnegan, Bowling, Michael, Larson, Bryce, Piccione, Carmelo, Burch, Neil, Billings, Darse, and Rayner, Chris. Bayes' bluff: Opponent modelling in poker. In *Proceedings of Uncertainty in Artificial Intelligence*, 2005.

Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*, volume 1. MIT Press Cambridge, 1998.

Tampuu, Ardi, Matiisen, Tambet, Kodelja, Dorian, Kuzovkin, Ilya, Korjus, Kristjan, Aru, Juhan, Aru, Jaan, and Vicente, Raul. Multiagent cooperation and competition with deep reinforcement learning. *ArXiv:1511.08779*, 2015.

Uther, William and Veloso, Manuela. Adversarial reinforcement learning. Technical Report CMU-CS-03-107, School of Computer Science, Carnegie Mellon University, 2003.

Watkins, Christopher J. C. H. and Dayan, Peter. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

Zhang, Marvin, McCarthy, Zoe, Finn, Chelsea, Levine, Sergey, and Abbeel, Pieter. Learning deep neural network policies with continuous memory states. *ArXiv:1507.01273*, 2015.