NEURON AS AN AGENT

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose *Neuron as an Agent* (NaaA) as a novel framework for reinforcement learning (RL), and explain its optimization method. NaaA incorporates all neural network units as agents and optimizes the reward distribution as a multi-agent RL problem. First, showing optimization of NaaA, this report describes the negative result that the performance decreases if we naively consider the units as agents. To resolve that difficulty, we introduce a mechanism from game theory. As a theoretical result, we demonstrate that the agent obeys the system to maximize its *counterfactual return* as the Nash equilibrium of the mechanism. Subsequently, we show that learning counterfactual returns leads the model to learning optimal topology among units. We propose *adaptive dropconnect*, a natural extension of dropconnect. Finally, we confirm that optimization with the framework of NaaA leads to better performance of RL, with numerical experiments. Specifically, we use a single-agent environment from Open AI gym, and a multi-agent environment from ViZDoom.

1 INTRODUCTION

Deep reinforcement learning (DRL) succeeds in many areas. Deep Q-Network (DQN) (Mnih et al., 2015; Silver et al., 2016) finds the optimal action from a screen sequence from Atari, and selects the move closest to win from a face of a board of Go. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) realizes the multiple-join control considering conditions such as friction and gravity factors in a physical space. The applicability of DRL is becoming wider year by year. Reasonable performance is reported for 3D games such as Doom (Dosovitskiy & Koltun, 2016).

A neural network is workable for DRL because a neural network abstracts the implicit state in an environment and obtains an informative state representation. From a micro perspective, the abstraction capability of each unit contributes to the return of the entire system. Therefore, we address the following question.

Will reinforcement learning work even if we consider each unit as an autonomous agent?

The contribution of this paper is that we propose *Neuron as an Agent* (NaaA) as a novel framework for RL, and explain its optimization method. NaaA incorporates all neural network units as agents and optimizes the reward distribution as a multi-agent RL problem. In the of NaaA reward design, a unit distributes its received reward to other input units, passing its activation to the unit as cost. Consequently, the actual reward is profit, defined as the difference between inflow (received reward) and outflow (paid cost). In the setting, the economic metaphor can be introduced: profit is the balance of revenue and cost. Therefore, a unit should address tradeoffs between optimization of cumulative revenue maximization and cumulative cost minimization.

This paper is organized as presented below. First, showing the optimization of NaaA, this report describes the negative result that the performance decreases if we naively consider units as agents. As a solution to this difficulty, we introduce a mechanism of auction which applies game theory. As a theoretical result, we demonstrate that the agent obeys to maximize its *counterfactual return* as the Nash equilibrium. The counterfactual return is that by which we extend counterfactual reward, the criterion proposed for multi-agent reward distribution problem (Agogino & Tumer, 2006), along a long time axis.

Subsequently, we present that learning counterfactual return leads the model to learning optimal topology between the units. In addition, we propose *adaptive dropconnect*, a natural extension of dropconnect (Wan et al., 2013). Adaptive dropconnect combines dropconnect, which pure-randomly masks the topology, with adaptive algorithm, which prunes the connection with less counterfactual return with higher probability. It uses ε-greedy as a policy, and is equivalent to dropconnect in the case of $\varepsilon = 1$. It is equivalent to counterfactual return maximization, which constructs the topology deterministically in the case of $\varepsilon = 0$.

Finally, we confirm that optimization with the framework of NaaA leads to better performance of RL, with numerical experiments. Specifically, we use a single-agent environment from Open AI gym, and a multi-agent environment from ViZDoom.

Although considering all the units as agents might be simplistic at first glance, it has a wider applicable area. From the perspective of optimization for a single neural network, it can be applied to pruning by optimizing the topology. Furthermore, introducing the concept of reward distribution divides the single neural network to numerous autonomous parts. It enables us not only to address sensor placing problem in IoT for partially observed Markov decision process (POMDP): arbitrary incentivized participants can join the framework.

2 RELATED WORK

NaaA belongs to a class of partially observable stochastic game (POSG) (Hansen et al., 2004) because it processes multiple units as agents. POSG, a class of reinforcement learning with multiple agents in a POMDP environment, presents several research issues, one of which is communication. CommNet (Sukhbaatar et al., 2016), which exploits the characteristics of a unit that is agnostic to the topology of other units, employs backpropagation to train multi-agent communication. Another one is credit assignment. Instead of reward $R(a_t)$ of an agent i for actions at t a_t, QUICR-learning (Agogino & Tumer, 2006) maximizes counterfactual reward $R(a_t) - R(a_t - a_{it})$, the difference in the case of the agent i takes an action a_{it} (a_t) and not ($a_t - a_{it}$). COMA (Foerster et al., 2017) also maximizes counterfactual rewards in an actor–critic setting. In the setting, all actors have common critics, which improves both actors and critics with time difference (TD)-error of a counterfactual reward. This paper unifies both issues: communication and credit assignment. The main proposal is a framework to manage the agents to maximize the *counterfactual return*, the extended counterfactual reward along the time axis.

Training a neural network with a multi-agent game is an emerging methodology. Generative adversarial nets (GAN) (Goodfellow et al., 2014) have the goal of obtaining true generative distribution as a Nash equilibrium of a competitive game that includes two agents with contradictory rewards: a generator and a discriminator. In game theory, the outcome maximizing overall reward is named Pareto optimality. Nash equilibrium is not guaranteed to converge to Pareto optimality. The difference between them is designated as a dilemma. Because the existence of a dilemma depends on the reward design, methods to resolve dilemmas with good reward design are being investigated: mechanism design (Myerson, 1983) is also known as inverse game theory. Mechanism design is applied to auctions (Vickrey, 1961) and matching (Gale & Shapley, 1962). GAN and our proposal, NaaA, are outcomes from mechanism design. NaaA applies a digital goods auction (Guruswami et al., 2005) to reinforcement learning with a multi-agent neural network, to obtain a maximized return by units as a Nash equilibrium.

Adaptive DropConnect (ADC), which we propose in a later part of this paper, extends DropConnect (Wan et al., 2013), a regularization technique. The idea of ADC (instead of dropping each connection between units in constant probability, using skew probability correlated to the absolute value of weights) is eventually closer to Adaptive DropOut (Ba & Frey, 2013), although the derivation differs. The adjective "adaptive" is added with respect to the method. Optimizing the neural network with RL was investigated by Andrychowicz et al. (2016). In contrast to their methods, which use recurrent neural network (RNN) and which therefore have difficult implementation, our method is RNN-free and forms as a layer. For those reasons, its implementation is simple and fast. Moreover, it has a wide area of applicability.

3 BACKGROUND

First, we consider a POMDP environment in which a single agent acts. The POMDP environment is a seven-tuple $(\mathcal{S}_{\mathrm{H}}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_{\mathrm{O}}, \mathcal{O}, \gamma)$, where $\mathcal{S}_{\mathrm{H}}$ represents a set of states, $\mathcal{A}$ stands for a set of actions, $\mathcal{T}$ denotes a transitive probability, $\mathcal{S}_{\mathrm{O}}$ represents a possible set of observations, $\mathcal{O}$ denotes a set of observation probability, and γ is the discount rate. An agent partially predicts state $h \in \mathcal{S}_{\mathrm{H}}$ through an observation $s \in \mathcal{S}_{\mathrm{O}}$. Generally, s has higher dimensions than h, and is complex. For example, although Atari 2600 has a read only memory (RAM) as the true state, which contains 128 bytes, the generated image from that s has more than 10,000 dimensions. Therefore, DQN and DRQN abstract s, and create original state representation to predict good action efficiently. (Although the original paper of DQN assumes MDP, the paper of DRQN pointed out that the environment is POMDP). Although DQN does not address the state transition directly because it is model-free method, some interpretations hold that the hidden state representation is learned in the previous layer of the output layer (Zahavy et al., 2016) Using the method below, we assume that the agent chooses an action through a neural network. The POSG environment is a natural extension of POMDP to multi-agent environment defined by a tuple $(\mathcal{S}_{\mathrm{H}}, \mathcal{A}^i, \mathcal{T}, \mathcal{R}^i, \mathcal{S}_{\mathrm{O}}^i, \mathcal{O}^i, \gamma^i)_{i \in \mathcal{I}}$, where $\mathcal{I}$ is a finite set of agents indexed 1.

We employ several concepts from game theory. Although RL and game theory are typically investigated in parallel, several concepts in game theory can be written in the domain of RL. The (Bayesian) Nash equilibrium $\hat{\pi}_i$ is a policy by which all agents maximize their expected reward. That is,

$$\hat{\pi}(s_{it}) = \operatorname*{argmax}_{a_i \in \mathcal{A}^i} \mathbb{E}_{\mathbf{a}_{-i} \in A^{-i}, h \sim \mathcal{O}^i} \left[\mathcal{R}^i(h, \mathbf{a}) | s_{it} \right] \; \forall i \in \mathcal{I}, \tag{1}$$

where $\mathcal{A}^{-i}$ is a set of actions except for i. Intuitively, the equation took an expected value of reward to integrate out other agents' unobserved actions. Because the Nash equilibrium is sufficient to state only the best action in the most cases, we use the notation with action $\hat{\mathbf{a}}$ in the following.

The design of NaaA is inspired by neuroscience. A neuron in a neurocircuit consumes adenosine triphosphate (ATP) supplied from connected astrocytes. The astrocyte is a glia cell, which forms the structure of a brain. It supplies fuel from the vessel. Because the amount of ATP is constrained, the discarded neuron will become extinct with execution of apoptosis. Also, because apoptosis of a neuron is restrained by neurotrophins (NTFs) such as nerve growth factor (NGF) and brain-derived neurotrophic factor (BDNF), neurons which can obtain much NTF will live. The perspective of interpreting a neuron as an independent living object is known as neural Darwinism (Edelman, 1987).

4 NEURON AS AN AGENT

A typical artificial neural network is a directed graph $\mathfrak{G} = (\mathcal{V}, \mathcal{E})$ among the units. $\mathcal{V} = \{v_1, \ldots, v_N\}$ is a set of the units. $\mathcal{E} \subset \mathcal{V}^2$ is a set of edges representing connections between two units. If $(v_i, v_j) \in \mathcal{E}$, then connection $v_i \to v_j$ holds, indicating that v_j observes activation of v_i. We denote activation of the unit v_i at time t as $x_{it} \in \mathbb{R}$. Additionally, we designate a set of units which unit i connects to as $N_i^{\mathrm{out}} = \{j | (v_i, v_j) \in \mathcal{E}\}$ and a set of units which unit i is connected from as $N_i^{\mathrm{in}} = \{j | (v_j, v_i) \in \mathcal{E}\}$. We denote $N_i = N_i^{\mathrm{in}} \cup N_i^{\mathrm{out}}$.

NaaA interprets v_i as an agent. Therefore, $\mathfrak{G}$ is a multi-agent system. An environment for v_i comprises an environment that the multi-agent system itself touches and a set of the unit to which v_i directly connects: $\{v_i \in V | i \in N_i\}$. We distinguish both environments by naming the former as an external environment, and by naming the latter as an internal environment. v_i will receive rewards from both environments. We add the following assumption for characteristics of the v_i.

- N1: (Selfishness) Instead of minimizing the global training error, at each timing t, v_i acts to maximize toward maximizing its own return (cumulative discounted reward) $G_{it} = \sum_{k=0}^{T} \gamma^k R_{i,t+k}$, where $\gamma \in [0, 1]$ is the discount rate and T is the terminal time.
- N2: (Conversation) The summation of a reward by which $\mathcal{V}$ will receive both an internal and external environment R_{it} over all the units are equivalent to reward R_t^{ex}, which the entire multi-agent system receives from the external environment.

N3: (Trade) The v_i receives internal reward ρ_{jit} from $v_j \in \mathcal{V}$ in exchange of activation signal x_i before transferring the signal to the unit. At the same time, ρ_{jit} is subtracted from the reward of v_j.

N4: (NOOP) v_i has NOOP (no operation), for which the return is $\delta > 0$ as an action. With NOOP, the unit inputs nothing and outputs nothing.

In terms of neuroscience, N1 states that the unit acts as a cell. N2 and N3 state the distribution of NTF. N4 corresponds to apoptosis. NOOP is selected when the expected returns of the other actions are non-positive. In the following, we construct the framework of NaaA from the assumptions.

4.1 Cumulative Discounted Profit Maximization Framework

We denote the external reward by which unit v_i receives at time step t as R_{it}^{ex}, where $\sum_{i=1}^{n} R_{it}^{\text{ex}} = R_t^{\text{ex}}$ holds. From N3, reward R_{it}, which v_i receives at t can be written as the following.

$$R_{it} = R_{it}^{\text{ex}} + \sum_{j \in N_i^{\text{out}}} \rho_{jit} - \sum_{j \in N_i^{\text{in}}} \rho_{ijt}. \tag{2}$$

The equation is divided into positive terms and a negative term, we name the former as revenue, and the latter as cost, and denote them respectively as $r_{it} = R_{it}^{\text{ex}} + \sum_{j \in N_i^{\text{out}}} \rho_{jit}$, $c_{it} = \sum_{j \in N_i^{\text{in}}} \rho_{ijt}$. We name R_{it} as profit.

In this case, v_i maximizes the cumulative discounted profit G_{it} represented as

$$G_{it} = \sum_{k=0}^{T} \gamma^k R_{i,t+k} = \sum_{k=0}^{T} \gamma^k (r_{i,t+k} - c_{i,t+k}) = r_t - c_t + \gamma G_{i,t+1}. \tag{3}$$

G_{it} is unobserved unless the time is reached at the end of the episodes. Because prediction based on the current value is needed to select the optimal actions, we approximate G_{it} with value function $V_i^{\pi_i}(s_{it}) = \mathbb{E}_{\pi_i}\left[G_{it} \mid s_{it}\right]$ where $s_{it} \in \mathcal{S}_{\text{O}}$. In this case, the following equation holds.

$$V_i^{\pi_i}(s_{it}) = r_{it} - c_{it} + \gamma V_i^{\pi_i}(s_{i,t+1}), \tag{4}$$

Therefore, we need only consider maximization of revenue, the value function, and cost minimization. $R_{it} > 0$, i.e., $r_{it} > c_{it}$ indicates that the unit gives the additional value to the obtained data. The unit acts NOOP because $V_i^{\pi_i}(s_{it}) \le 0 < \delta$ if $R_{it} \le 0$ for all t.

5 Optimization

To maximize the cumulative discounted profit in a framework of NaaA, it is important to balance the two contradicting criteria: revenue r_{it} and cost c_{it}.

To achieve that, we use the mechanism design. We introduce mechanism design because, unlike several existing studies (Sukhbaatar et al., 2016), NaaA assumes that all agents are not cooperative but selfish. If we naively optimize the optimization problem of NaaA, then we obtain the trivial solution that the internal rewards will converge to 0, and that all the units become NOOP. Therefore, the multi-agent system should select the action with no information. It is equivalent to taking an action randomly. For that reason, the external reward R_t^{ex} shrinks markedly.

5.1 Envy-free Auction

To achieve Pareto optimality, we borrow the idea from the digital goods auction. The auction theory belongs to mechanism design. It is intended to unveil the true price of goods. Digital goods auction is one mechanism from auction theory. It is target to copyable goods without cost, such as digital books and music.

Although several variations of digital goods auctions exist, we use an envy-free auction (Guruswami et al., 2005) because it requires a simple assumption: the same goods have one price simultaneously. In NaaA, it can be represented as the following assumption:

N5: (Law of one price) If $\rho_{j_1,i,t}, \rho_{j_2,i,t} > 0$, then $\rho_{j_1,i,t} = \rho_{j_2,i,t}$.

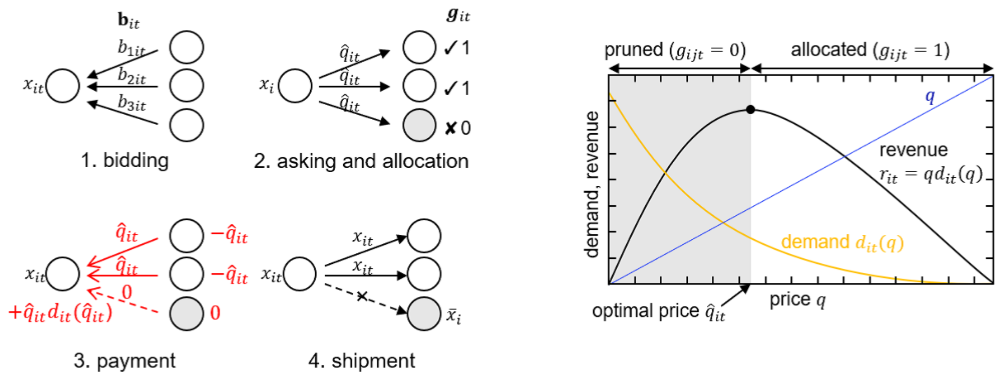

Figure 1: **Left**: The process of trade in an envy-free auction. **Right**: A price determination curve for a unit. Revenue of a unit is a product of monotonically decreasing demand and price. The price maximizing the revenue is the optimal price.

Therefore, v_i has an intrinsic price at the same timing t. We denote the price as q_{it}.

TODO: Describe for allocation

We present the envy-free auction process at the left of Figure 1. It shows the negotiation process between one unit in sending activation and a group of units that buy the activation. The negotiation performed per time step in RL. We name the unit in sending activation as a seller, and units in buying activation as buyers. First, the buyer bids the unit in bidding price b_{jit} (**1**). Next, the seller decides the optimal price $\hat{q}_{it}$, and performs allocation (**2**). After allocation, the buyers perform payment as $\rho_{jit} = g_{jit}\hat{q}_{it}$ (**3**). The seller only sends activation x_i to the allocated buyers (**4**). A buyer which cannot receive the activation approximates x_i with $\mathbb{E}_\pi[x_i]$.

In the following, we discuss revenue, cost, and value functions based on Eq:(4).

Revenue: The revenue of a unit is given as

$$
\begin{aligned}
r_{it} &= \sum_{j \in N_i^{\text{out}}} g(b_{jit}, q_{it}) q_{it} + R_i^{\text{ex}} = q_{it} \sum_{j \in N_i^{\text{out}}} g(b_{jit}, q_{it}) + R_i^{\text{ex}} \\
&= q_i d_{it}(q_t) + R_i^{\text{ex}}, \qquad (5)
\end{aligned}
$$

where $g(\cdot, \cdot)$ is allocation. It is defined using a step function $H(\cdot)$ as $g(b, q) = H(b - q)$. $d_{it}(q_{it})$ is a count of units for which the bidding price for q_{it} is greater than or equal to q_{it}, designated as demand. q_{it} maximizing the equation is designated as the optimal price. It is denoted as $\hat{q}_{it}$. Because the second term in the equation is independent of q_t, the optimal price $\hat{q}_{it}$ is given as

$$\hat{q}_{it} = \operatorname*{argmax}_{q \in [0, \infty)} q d_{it}(q). \qquad (6)$$

We present the curve of q_{it} on the right side of Figure 1.

Cost: The cost is an internal reward that the unit should pay to other units. It is represented as shown below.

$$c_{it} = \sum_{j \in N^{\text{in}}} g(b_{ijt}, q_j) q_j \qquad (7)$$

Although c_{it} itself is minimized when $b_{ijt} = 0$, this represents a tradeoff with the following value function.

Value Function: Activation x_i depends on input from the units in N_i^{in}. It affects the bidding price from units in N_i^{out}. If we minimize b_{ijt} and let $b_{ijt} = 0$, then the purchase of activation fails, and the reward the unit can obtain from the units to which the unit connects becomes lower in the future.

Then, we designate the allocation as $\mathbf{g}_{it} = (g_{i1t}, \ldots, g_{iNt})^{\mathrm{T}}$, and consider effects for value functions in the cases when a unit succeeds in purchasing v_j or not. The value function can be written as the equation using a state-value function $Q(s_{i,t+1}, \mathbf{g}_{i,t+1})$.

$$\begin{aligned} V_i^{\pi_i}(s_{it}) &= Q_i^{\pi_i}(s_{it}, \mathbf{g}_{it}) \\ &= \sum_{j \in N_i^{\text{in}}} g_{ijt}(Q_i^{\pi_i}(s_{it}, \mathbf{e}_j) - Q_i^{\pi_i}(s_{it}, \mathbf{0})) + Q_i^{\pi_i}(s_{it}, \mathbf{0}) \\ &= \sum_{j \in N_i^{\text{in}}} g_{ijt} o_{ijt} + Q_i^{\pi_i}(s_{it}, \mathbf{0}) \\ &= \mathbf{g}_{it}^{\mathrm{T}} \mathbf{o}_{it} + Q_i^{\pi_i}(s_{it}, \mathbf{0}) \end{aligned} \tag{8}$$

We designate $o_{ijt} = Q_i^{\pi_i}(s_{it}, \mathbf{e}_j) - Q_i^{\pi_i}(s_{it}, \mathbf{0})$ as the *counterfactual return*, which is equivalent to the cumulative discount value of counterfactual reward (Agogino & Tumer, 2006). That is, the cost the unit will pay is $\hat{q}_{it}$ in success of purchasing data, and o_{it} otherwise.

Therefore, the optimization problem is presented below.

$$\max_{\mathbf{b},q} \mathbb{E}_{\hat{\mathbf{q}}_t}\left[V_i^{\pi_i}(s_{it})\right] = \max_q q d_{it}(q) - \min_{\mathbf{b}} \mathbb{E}_{\hat{\mathbf{q}}_t}\left[\mathbf{g}_{it}(\mathbf{b})^{\mathrm{T}}(\hat{\mathbf{q}}_t - \gamma \mathbf{o}_{i,t+1})\right] + \text{const.}. \tag{9}$$

We take the expectation $\mathbb{E}_{\hat{\mathbf{q}}_t}[\cdot]$ because the asked price $\hat{\mathbf{q}}_t$ is unknown for v_i, except for $\hat{q}_{it}$, and $g_{iit} = 0$.

Then, what is bidding price b_{it} to maximize return? The following theorem holds.

Theorem 5.1. *(Truthfulness) the optimal bidding price for maximizing return is* $\hat{\mathbf{b}}_{it} = \mathbf{o}_{it}$.

See the Appendix for the proof.

That is, the unit should only consider its counterfactual return (!). Consequently, in the mechanism of NaaA, the unit obeys as if performing valuation to the other units, and declares the value truthfully.

Then, the following corollary holds:

Corollary 5.1. *The Nash equilibrium of an envy-free auction* $(\mathbf{b}_{it}, q_{it})$ *is* $(\mathbf{o}_{it}, \operatorname{argmax}_q q d_{it}(q))$.

The remaining problem is how to predict $\mathbf{o}_t$. Although several method can be applied to this problem, we use Q-learning to predict $\mathbf{o}_t$. As $\mathbf{o}_{it}$ is difference of two Qs, we approximate each of Q. Other RL such as SARSA and A3C can be employed. We parametrize the state with a vector $\mathbf{s}_t$ which contains input and weight. ϵ-greedy policy with Q-learning typically suppose that discrete actions So, as an action, we employ allocation g_{ijt} instead of $\mathbf{b}_{it}$ and q_{it}. The overall algorithm is shown in Algorithm 1.

5.2 Adaptive DropConnect

Actually, NaaA is useful not only for multi-agent RL, but also for training of the network. Typical training algorithms of a neural network such as those of RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014) are based on stochastic gradient descent (SGD). The optimization is performed sequentially. Therefore, the problem can be interpreted as a problem to update the state (i.e., weight) to the goal, which is minimization of the expected likelihood.

The learning can be accelerated by application of NaaA to the optimizer. We designate the application of NaaA to SGD as *Adaptive DropConnect* (ADC), which is eventually a combination of DropConnect (Wan et al., 2013) and Adaptive DropOut (Ba & Frey, 2013). We introduce ADC herein as one application of NaaA.

ADC uses NaaA for supervised optimization problem with several revisions. First, an environment has an input state such as an image. The agent is expected to update its parameters to maximize its reward obtained from the criterion calculator. The criterion calculator gives batch-likelihood as the reward to the agent. The agent is a classifier which updates its weights to maximize the reward from the criterion calculator. The weights are recorded as an internal state. As a counterfactual return o_{ijt}, we used a heuristic that uses the absolute value of weight $|w_{ijt}|$, which is the same technique as that used by Adaptive DropOut. We use the absolute value of weights because it is the update amount for which the magnitude of error of the output of units is proportional to $|w_{ijt}|$.

Algorithm 1 Envy-free auction for NaaA

for $t = 1$ **to** T **do**
    Compute a bidding price for every edge: **for** $(v_j, v_i) \in \mathcal{E}$ **do** $b_{ijt} \leftarrow Q^{\pi_i}(\mathbf{s}_{it}, \mathbf{e}_j) - Q^{\pi_i}(\mathbf{s}_{it}, \mathbf{0})$
    Compute an asking price for every node: **for** $v_i \in \mathcal{V}$ **do** $\hat{q}_{it} \leftarrow \operatorname*{argmax}_{q \in [0,\infty)} q d_{it}(q)$.
    for $(v_i, v_j) \in \mathcal{E}$ **do**
        Compute allocation: $g_{jit} \leftarrow H(b_{jit} - \hat{q}_{it})$
        Compute the price the agent should pay: $\rho_{jit} \leftarrow g_{jit}\hat{q}_{it}$
    end for
    Make a payment: **for** $v_i \in \mathcal{V}$ **do** $R_{it} \leftarrow \sum_{j \in N_i^{\text{out}}} \rho_{jit} - \sum_{j \in N_i^{\text{in}}} \rho_{ijt}$,
    Make a shipment: **for** $v_i \in \mathcal{V}$ **do** $\tilde{x}_{ijt} = g_{ijt} x_{ijt} + (1 - g_{ijt})\bar{x}_{ijt}$
    for $v_i \in \mathcal{V}$ **do**
        Observe external state $\mathbf{s}_{it}^{\text{ex}}$
        $\mathbf{s}_{it} \leftarrow (\mathbf{s}_{it}^{\text{ex}}, \tilde{\mathbf{x}}_{it}, \boldsymbol{\theta}_i)$
        Sample action $a_{it}^{\text{ex}} \sim \pi_i^{\text{ex}}(\mathbf{s}_{it})$
        Receive external reward $R_{it} \leftarrow R_{it} + R_{it}^{\text{ex}}(a_{it}^{\text{ex}})$
        Update Q^{π_i} under the manner of Q-learning by calculating the time difference (TD)-error
    end for
end for

Algorithm 2 Adaptive DropConnect

for $t = 1$ **to** T **do**
    Compute a bidding price for every edge: **for** $(v_j, v_i) \in \mathcal{E}$ **do** $b_{ijt} \leftarrow |w_{ijt}|$
    Compute an asking price for every node: **for** $v_i \in \mathcal{V}$ **do** $\hat{q}_{it} \leftarrow \operatorname*{argmax}_{q \in [0,\infty)} q d_{it}(q)$.
    for $(v_i, v_j) \in \mathcal{E}$ **do**
        Compute allocation: $g_{jit} \leftarrow H(b_{jit} - \hat{q}_{it})$
    end for
    Sample a switching matrix U_t from a Bernoulli distribution: $U_t \sim \text{Bernoulli}(\varepsilon)$
    Sample the random mask M_t from a Bernoulli distribution: $M_t \sim \text{Bernoulli}(1/2)$
    Generate the adaptive mask: $M_t' \leftarrow U_t \circ M_t + (1 - U_t) \circ G_{ijt}$
    Compute $\mathbf{h}_t$ for making a shipment: $\mathbf{h}_t \leftarrow (M_t' \circ W_t)\mathbf{x}_t + \mathbf{b}_t$
    Update W_t and $\mathbf{b}_t$ by backpropagation.
end for

The algorithm is presented as Algorithm 2. Because the algorithm is quite simple, its implementation can be performed easily. For that reason, it can be widely applied for most general deep learning problems such as image recognition, sound recognition, and even for deep reinforcement learning.

6 EXPERIMENT

To confirm that NaaA works widely with machine learning tasks, we confirm our method of supervised learning tasks as well as reinforcement learning tasks. As supervised learning tasks, we use typical machine learning tasks such as image classification using MNIST, CIFAR-10, and SVHN.

As reinforcement tasks, we confirm single- and multi-agent environment. The single-agent environment is from OpenAI gym. We confirm the result using a simple reinforcement task: CartPole. In multi-agent, we use ViZDoom, a 3D environment for reinforcement learning.

6.1 CLASSIFICATION

In the classification task, we experiment with our method using several standard datasets: MNIST, CIFAR-10, and SVHN. As a method of comparison, we compare baseline (vanilla feed-forward neural network) and DropConnect. In a hyperparameter setting, we set $\epsilon = 0.2$, which means that the agent randomly masks the weights in 0.2 of the chance rate, and playing the auction game in 0.8 of chance rate. Table **??** presents our framework of NaaA complements disadvantage of DropConnect.

It randomly drops the weight completely, and does not consider implicit counterfactual return of units.

6.2 Single-agent RL

Next, we set the single-agent reinforcement learning task. We used the CartPole task from OpenAI gym with visual input. In this setting, the agent must balance a pole while moving a cart. There is much non-useful information related to the image. For that reason, pruning the pixels is important. The result demonstrates that our method improves the standard RL.

6.3 Multi-agent RL

The additional feature of NaaA is credit assignment for reward distribution, meaning that if the neural network is divided into multiple agents, it works by playing the auction game. We confirmed that additional agents complement the main player using ViZDoom, an environment for Doom. A player in Doom environment should seek the enemy in the map, and then defeat the enemy. Because ViZDoom provides several maps, we used ViZDoom.

6.3.1 Setup

We used a scenario based on Defend the Center (DtC), provided by ViZDoom platform. In DtC, players are placed in the center of a field of circle. They attack enemies that come from the wall. The game has two players: a main player and a cameraman. Alhough the main player can attack the enemy with bullets, the cameraman has no way to attack, and only scouts for the enemy. The action space for the main player is the combination of { attack, turn left, turn right }. Therefore, the total number of actions is $2^3 = 8$. The cameraman has two possible actions: { turn left, turn right }. Although the players can only change direction, they cannot move on the field. The enemy will die if have the attack (bullet) from the main player once, then player receives +1. As a default on an episode, the ammunition amount is 26. The main player will die if under attack from the enemy to the extent that health becomes 0, then the player receives -1. The cameraman will not die if attacked by the enemy. The episode will terminate when the maim player dies, or after 525 steps have elapsed.

6.3.2 Model

We compared three models: the proposed method and two comparison targets.

Baseline DQN without communication. The main player learns standard DQN with the perspective that the player is viewing. Because the cameraman does not learn, the player continues to move randomly.

Comm DQN with communication. The main player learns DQN with two perspectives: the player's own and the cameraman's. The communication vector is learned with a feed-forward neural network. The method is inspired by Commnet.

NaaA The proposed method. The main player learns DQN with two perspectives: the player's own and the cameraman's. The transmission of reward and communication are performed using the proposed method.

6.3.3 Results

Training is performed in 10 million steps. Figure 2 Left presents that our model NaaA outperforms two methods. Improvement is achieved by Adaptive DropConnect. We confirmed that the cameraman sees the enemy through an episode. This can be interpreted as the cameraman reporting the enemy position. In addition to seeing the enemy, the cameraman sees the area behind of main player several times. This action enables the cameraman to observe attacks from the enemy while seizing a better relative position.

For further interpretation of the result, we present visualization of the revenue that the agent earned in Figure 2 Right as a heatmap. The background picture is a screen in Doom taken at the moment when the filter in CNN is mostly activated. Figure 3 shows an example of learnt sequence of actions by our method.

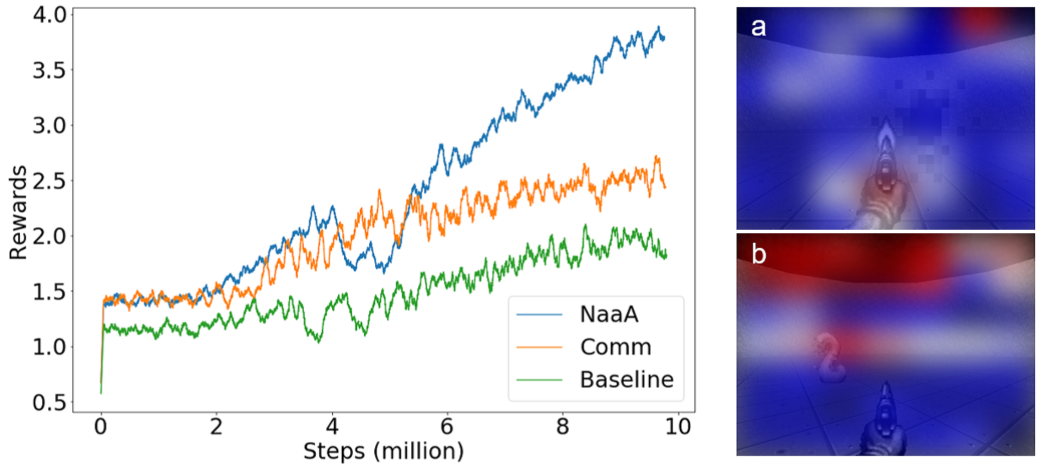

Figure 2: **Left:** Learning curve for the multi-agent task of VizDoom. Our method based on NaaA outperforms the other two methods: baseline and Comm DQN. **Right:** Reward visualization shows us what the cameraman sees: (a) The cameraman sees the pistol. (b) The cameraman sees the point which enemy appear and come closer.

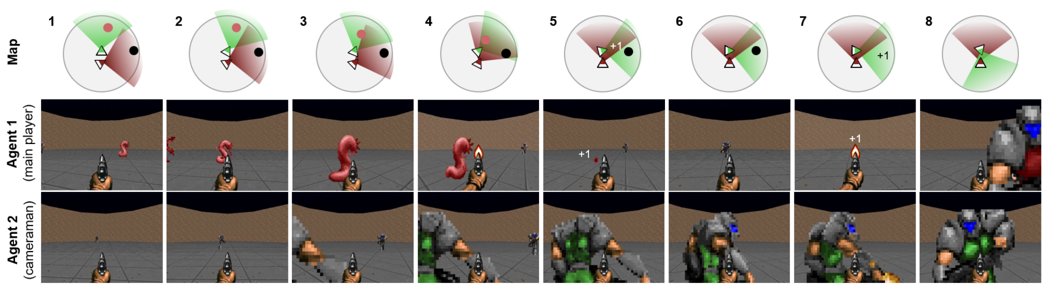

Figure 3: NaaA leads the agents to obtain cooperative relationship. First, the two agents are facing in different directions, and the cameraman sells its information to the main player (**1**). The main player who bought the information starts to turn right to find the enemy. The cameraman who sold the information starts to turn left to seek new information by finding the blind area of the main player (**2** and **3**). With turning, the main player attacks the first enemy which he already saw (**4** and **5**). After the main player finds out the enemy, he attacks the enemy, and obtain the reward (**6** and **7**). Until the next enemy appears, the agents watch their dead area each other (**8**).

7 Discussion

Regarding the optimization method, although envy-free auction guarantees truthfulness if the buyer prices are sealed, in cases where buyers can mutually communicate and share price information, the buyer can fake the price with lower demand in a process of collusion. To address the issue, several solutions such as random sample auction Goldberg et al. (2006) are proposed.

NaaA is applicable to learning distributed environments on a computer network such as a peer-to-peer network, and controlling the sub-modules of robots such as multiple cameras. Specifically, it is applicable to various methods as described below.

- Hyperparameter tuning. Several algorithms have been proposed such as neuroevolution using genetic algorithms. In the case, profit or counterfactual return is useful for a fitness function.
- Pruning. Computing costs can be reduced by downsizing a neural network.
- Attention control. Research of attention is using reinforcement learning to control attention.

- Ensemble. Our method is applicable to mixed multiple models.

These applications illustrate the direction of our research.

8 Conclusion and Future Works

This paper proposed NaaA, a reinforcement learning framework that treats each unit on a neural network as an agent. First, we pointed out there are dilemma problems if we naively optimize NaaA. We proposed an optimization method with auction. Consequently, an action by which units evaluate the counterfactual return of other units is obtained as a Nash equilibrium. Furthermore, we proposed Q-learning based algorithm, adaptive dropconnect, to optimize the neural network topology dynamically with evaluation of counterfactual return. For the evaluation, we performed experiments based on single-agent and multi-agent platforms, demonstrating that our experimentally obtained results improve existing methods.

As a direction of future research, we use on-policy methods to perform adaptive dropconnect, and consider applications combining genetic algorithms.

References

A. K. Agogino and K. Tumer. QUICR-learning for multi-agent coordination. AAAI'06, 2006.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 3084–3092, 2013.

A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *ICLR'17*, 2016.

G. M Edelman. *Neural Darwinism: The theory of neuronal group selection.* Basic books, 1987.

J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv:1705.08926*, 2017.

David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

Andrew V Goldberg, Jason D Hartline, Anna R Karlin, Michael Saks, and Andrew Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, 2006.

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

Venkatesan Guruswami, Jason D Hartline, Anna R Karlin, David Kempe, Claire Kenyon, and Frank McSherry. On profit-maximizing envy-free pricing. In *ACM-SIAM symposium on Discrete algorithms*, 2005.

Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pp. 709–715, 2004.

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

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ICLR'16*, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Roger B Myerson. Mechanism design by an informed principal. *Econometrica: Journal of the Econometric Society*, pp. 1767–1797, 1983.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

S. Sukhbaatar, R. Fergus, et al. Learning multiagent communication with backpropagation. In *NIPS'16*, 2016.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1058–1066, 2013.

T. Zahavy, N. Ben-Zrihem, and S. Mannor. Graying the black box: Understanding DQNs. In *ICML'16*, 2016.

APPENDIX

A.1 PROOF OF THEOREM 5.1

As for a buyer, the asking price q for a seller is unknown, we address q which has support $[0, \infty)$, and consideration to maximize $\mathbb{E}_q\left[G(b, q)\right]$, In this case, the following equation holds.

$$
\begin{aligned}
\frac{\partial}{\partial b}\mathbb{E}_q\left[G(b,q)\right] &= \frac{\partial}{\partial b}\int_0^\infty (H(b-q)\cdot(v-q) + G_0)p(q)dq \\
&= \frac{\partial}{\partial b}\left[\int_0^b (v-q)p(q)dq + G_0\int_0^\infty p(q)dq\right] \\
&= \frac{\partial}{\partial b}\int_0^b (v-q)p(q)dq \\
&= (v-b)p(q=b),
\end{aligned}
$$

Therefore, the condition to maximize $\mathbb{E}_q\left[G(b, q)\right]$ is $b = v$.