# A Boosting-based Deep Neural Networks Algorithm for Reinforcement Learning

Yu Wang† and Hongxia Jin†

*Abstract*— **In this paper, a new boosting-based deep neural networks algorithm is designed for improving the performance of model-free reinforcement learning structures. Based on theoretical proof and performance analysis, it is going to demonstrate that the new approach gives a faster convergence speed and a better return compared to existing deep neural network based RL approaches, like deep $Q$ network (DQN)[30], [13] and etc. A complete and detailed exploration of the new algorithm will be given in the paper as well. Also, simulation studies are conducted and compared with several other RL algorithms on the RL benchmark experiment tasks as given in [22]. The results demonstrate a great performance improvement on $Q$ learning by using our new boosting-based deep neural networks algorithm.**

## I. Introduction

The attractiveness of reinforcement learning is that it imitates human's learning behavior to some extend, and obtain an optimal result based on one's collected rewards through the entire learning procedure. This concept of reinforcement learning has a long history. Some of the earliest results can be traced back to 1960s, when Bellman gave the definition of Dynamic Programming, and also a corresponding solution using Bellman equation in his book ([1]). Other research groups, during the same period of time, started invegating the topic as a stochastic optimal control problem, and solved it from the view as control scientists ([2]). In 1990s, the reinforcement learning problem becomes popular in the computer science community. A well accepted definition of "reinforcment learning" is given by involving the discounted expected rewards in a Stochastic Markov Decision Process (MDP) ([4]). Some of the popular conventional reinforcement learning schemes includes Temporal differences (TD), $Q$-learning, SARSA, Approximate dynamic programming (ADP)etc. ([5]-[9])

Though the popularity of various reinforcement learning schemes, the difficulty of implentation in real environment and the slow training speed, especially due to the large number of states and actions, has constraint the development of applications using reinforcement learning. Recently, a group of researcher proposed a new reinforcement methodology, named "deep $Q$ network/learning (DQN)" ([13]), by applying a neural network to identify the unknown $Q$ value function. It has been a popular research topics due to its feasiblity for large states space [15] or even continuous state space ([16]).The approach can reduce the computational complexities to find the optimal $Q$ values at each state by estimating the non-linear $Q$ functions through a neural network ([17]). Researchers also try to use space-sampling approach to improve the speed of learning ([8], [27]). Despite the success of the algorithms, some of the theoretical analysis of using NN to identify $Q$ function hasn't been discussed, and also the training speed of neural network identifier itself are still a barrier to further expand the idea into more complicated problems in reinforcement learning field.

In this paper, a new boosting-based deep neural networks (BDNN) algorithm is proposed for improving the performance of model-free reinforcement learning. The idea is using the boosting information from multiple neural networks to identify the unknown $Q$ value function. Instead of only adjusting neural networks through back-propagation, in our new algorithm, a gradient based adjustment of convex boosting parameters for multiple neural networks is designed. Without loss of generality, the model-free based RL algorithm on which our new algorithm is implemented is chosen as deep $Q$ network in this paper.

The paper structure is organized as following: Section 2 gives a general background overview of reinforcement learning, $Q$-learning, neural network for system identification and deep $Q$ networks. A detailed boosting-based deep neural networks (BDNN) algorithm for deep $Q$-learning is demonstrated in Section 3. Section 4 is the experiment section by comparing our new designed boosting-based deep neural networks algorithm with several other deep neural network based approaches.

## II. Background

### A. Reinforcement Learning and Q-learning

*1) Reinforcement Learning:* Most of the general reinforcement learning problems are formed under a Markov Decision Process (MDP) environment. The MDP process is developed in a stochastic manner, i.e. the next state can be reached by an action at a specific state is supposed to be random in some manner (or with some probabilities). Figure(1) demonstrates a general structure of reinforcement learning problem.

*2) Q-Learning:* $Q$-learning is one of the most popular model-free reinforcement learning scheme, it use the $Q$ values to store the expected reward information to be optimized in reinforcement learning and the optimal $Q$-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma max_{a'} Q^*(s',a') \mid s,a] \qquad (1)$$

†: Both Yu Wang and Hongxia Jin are with Samsung Research America, Mountain View, CA

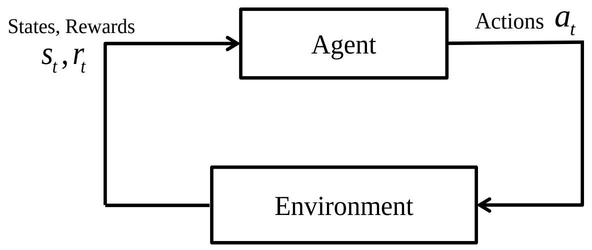

**Fig. 1: General Reinforcement Learning Structure**

where $s \in S$ is the state space and $a \in A$ is the action space, and $Q(s,a)$ is the action-value function and $Q^*(s,a)$ is the desired value function such that $Q \to Q^*$ when $i \to \infty$.

Most of the $Q$ value based schemes can be updated by using an iterative approach solving Bellman equation. In general, they can be represented as:

$$Q_t(s,a) = \mathbb{E}_{s'}[r + \gamma max_{a'} Q_{t+1}(s',a') \mid s,a] \quad (2)$$

As a very mature reinforcement learning scheme, there are a lot of literatures to explain the algorithm in detail ([18])-([20]), hence only the most general updating rule is given as above.

It should be noticed that it is almost impractical to implement algorithm (2) in practice, as the learning is totally on-line, and people who want to obtain the estimated $Q$ value should pretraining the algorithm in real applications (like robotics, games etc.), where high costs will be generated. For example, if someone wants to implement the $Q$-learning in controlling an unmanned vehicle, it is supposed to update the $Q$ value while the vehicle is running on highway (which is the real environment). Otherwise, the $Q$ value generated from any simulated environment will be deviated from its true value, as $Q$ itself does not have any physical meaning in the environment besides it is am action value function.

Some of the solutions to overcome this issue in engineer field, is to build a parametrized estimator function $Q(s,a,\theta)$ to identify $Q^*(s,a)$ and try to make $Q(s,a,\theta)$ to be as close as possible to the $Q^*(s,a)$ by adjusting value function's parameter $\theta$.

*3) Deep Q Network/Learning (DQN):* As mentioned earlier, one of the major issue in $Q$-learning is that the algorithm itself, does not works well if the state space is large or even continuous. It is requires a lot of memory space to store all the $Q$ values for every state, and further affect the convergence speed and returns of the system.

One of the solutions for the above issue is given as deep $Q$ learning. In the deep $Q$ learning algorithm introduced in ([13]), a convolutional neural network (CNN) is implemented to identify the unknown $Q(s,a)$. In general, any feed-forward neural network structure can be used to implement the DQN algorithm depending on the requirement of application. The fundamental principle of the algorithm is an optimization problem by minimizing the loss/error function given by

$$\mathcal{L}_t = \mathbb{E}_{s,a}[\sum_{i=1}^{k}(y_t(i) - Q_t(i))^2] \quad (3)$$

where $y_t(i)$ is the $i^{th}$ output generated from the neural network at time step $t$, and $k$ is the total number of $y$ labels. The deep $Q$ network in [13] is a special neural network based $Q$ learning algorithm developed using experience replay. A general detailed neural network based identification structure for $Q$-learning is shown as in Figure (2):

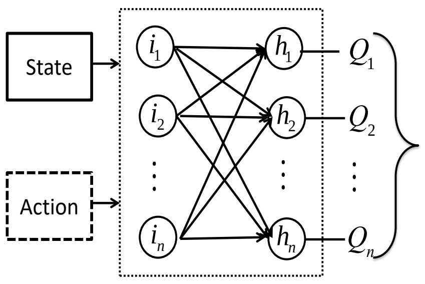

**Fig. 2: Deep $Q$-learning Structure**

*Comment:* The action in dash box is optional for the structure, which depends on the input of the model. In the Atari example given in ([13]), only the image frames state information are used as the input of the network.

Though the simulation result given by the team is quite satisfactory, the authors also mention that the algorithm is mostly empirical and there is also no guarantee of convergence using DQN due to its unsupervised training property. In this paper, our new boosting-based deep neural networks (BDNN) algorithm is trying to improve the convergence by training the boosting coefficients of multiple neural networks in a synchronous manner.

In the following sections, some of the most important theoretical results and challenges in the deep $Q$-learning will be firstly addressed, then a new concept of using boosting-based deep neural networks (BDNN) algorithm for reinforcement learning will be proposed.

## *B. Challenges in DQN*

In this subsection, two major challenges in Deep $Q$-learning will be discussed. One is the training speed in deep $Q$ network and the other is the convergence issue in DQN. We will first discuss the convergence of NN-based Q-learning before giving the issues of convergence in DQN..

*1) Speed of training DQN:* During the replication of the examples given based on the deep $Q$-learning (DQN) algorithm in ([13]), the training speed is extremely slow even for games like Atari. As described by the DQN algorithm, the neural network is trained based on a group of tuples of states and actions, i.e., $e_t$=$(s_t, a_t, r_t, s_{t+1})$ in the reinforcement learning loop as shown in the Figure (1). The data set $e_t$ is generated by replaying the game and pooled out many tuples into a replay memory. The slow speed of training an

DQN is mainly because of the slow training speed of neural network itself.

In order to improve this result, a boosting-based deep neural networks (BDNN) algorithm is designed and trying to avoid the issue mentioned above by separately training a new set of convex boosting coefficients for multiple neural networks.

*2) Convergence of NN-based reinforcement learning:* One of the basic questions in NN-based reinforcement learning approach is that why a neural network can be used to substitute the $Q$ function in a conventional $Q$- learning approach. Using neural network as an identification model is a topic has been discussed widely under many control schemes ([3],[26]), including the feasibility of using recurrent neural network structures like LSTM ([21], [31]). The proof can be divided into two steps. The first step is show that the neural network approach can identify the nonlinear $Q$ values generated by the general $Q$-learning iteration in (2), by using the Universal Approximation Algorithm, as long as appropriate initial condition is given and the neural network is complicated enough. The second step of the proof is given directly by the general $Q$-learning approach in equation (2) can converge to the optimal $Q$ value, i.e. $Q^*$, in Bellman Equation. the The first step of proof will be contributed by the Stone-Weierstrass Theorem and the Universal Approximation Theorem, the second step of the proof will be given by the fact that the $Q$ functions obtaining from the existing $Q$-learning approaches satisfying equation (2) is a non-linear solution of Bellman equation.

Following is the well-known The Stone-Weierstrass Theorem

*Proposition 1:* The Stone-Weierstrass Theorem:

Suppose $X$ is a compact Hausdorff space and $A$ is a subalgebra of $C(X,\mathbb{R})$ which contains a non-zero constant function. Then A is dense in $C(X,\mathbb{R})$ if and only if it separates points.

By following the Stone-Weierstrass Theorem, the Universal Approximation Theorem for neural network can be further given as below. A general well-accepted form is as following:

*Proposition 2:* The Universal-Approximation Theorem

Let $\varphi(t)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let $I_m$ denotes the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on $I_m$ is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exists an integer $N$, real constants $v_i$,$b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right) \tag{4}$$

as an approximate realization of the function $f$ where $f$ is independent of $\varphi$ ; that is,

$$|F(x) - f(x)| < \varepsilon \tag{5}$$

or all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

The Universal Approximation Theorem gives a simple fact that: if the neural network itself is large enough and the initial conditions are properly chosen, it can approximate any non-linear continuous function. A strict proof is given by Theorem 1.

**Theorem 1** *Convergence of Neural Network identification for Q functions satisfying (2).*

*Let $\varphi(t)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let $I_m$ denotes the m-dimensional unit hypercube $[0,1]^m$. Given any values $Q(s_t)$ satisfying (2) and $Q(s_t) \in D(I_m)$, i.e. the states information at time $t$ and the element in $s_t$ is normalized within $[0,1]$, $D(I_m)$ is a discrete space of $I_m$ and $\epsilon > 0$, there exists integers $M$ and L, real constants $v_j$,$b_j \in \mathbb{R}$ and real vectors $w_j \in \mathbb{R}^m$, where $j = 1, \cdots, L$, such that we may define:*

$$N(s_t) = \sum_{j=1}^{L} v_j \varphi\left(w_j^T s_T + b_j\right) \tag{6}$$

*as an approximate realization of the function $Q(s_t)$. And for all $i > M$, such that:*

$$|N(s_t) - Q(s_t)| < \varepsilon \tag{7}$$

*for all $s_t \in I_m$.*

*Proof:* The theorem itself is a simple derivation of the Universal Approximation theorem for time-varying function, the extra proof for the difference is that $Q(e)$ will converge to $Q^*$ a discrete function in $D(I_m)$, which is a subset of the continuous function space $C(I_m)$, hence the result from the Universal Approximation Theorem can be carried out to the proof of the convergence of Neural Network identification for reinforcement learning.

By the convergence of the general $Q$-learning algorithm given by (2), $\exists\ M_1 \in \mathbb{Z}^+$ such that for all $i > M_1$ and $\epsilon > 0$, the following inequality

$$|Q^*(s_t) - Q(s_t)| < \frac{\epsilon}{2} \tag{8}$$

is satisfied.

Also, by Universal Approximation Theorem, it can be shown that $\exists\ M_2 \in \mathbb{Z}^+$ for all $i > M_2$ and $\forall\, \epsilon > 0$, such that

$$|N(s_t) - Q^*(s_t)| < \frac{\epsilon}{2} \tag{9}$$

By combining the inequality (8) and (II-B.3), it can be shown that for $\forall\ \epsilon > 0$, $\exists\, M = sup\{M_1, M_2\}$, such that for all $i > M$

$$\begin{aligned} |N(s_t) - Q(s_t)| &< |N(s_t) - Q^*(s_t)| + |Q^*(s_t) - Q(s_t)| \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} \\ &= \epsilon \end{aligned} \tag{10}$$

Hence the above theorem is proved. ■

*3) Convergence issue in DQN:* In reality, as discussed earlier, it is not possible to guarantee the convergence of a

DQN. The main reason is because that the value $Q(s)$ is also an estimated value in order to make the algorithm work in large state space, hence is not based on the known $Q$ tables.

$Q(s_t)$, the value function in iteration $t$, is estimated as the sum of current state reward plus future states' expected rewards:

$$Q(s_t) = r_t + \gamma \max_{a_{t+1}} N(s_{t+1}, a_{t+1}) \tag{11}$$

where $\gamma \in [0,1]$ is the discount factor, and $s_{t+1}$ is a neighbor of $s_t$.

This generates the convergence issue in DQN as $N(s_{t+1}, a_{t+1})$ is an estimated value from the neural network. The proof given in () becomes the difference between two network estimated values. It is an unsupervised learning and the convergence cannot be guaranteed.

## III. Boosting-based deep neural networks (BDNN) for $Q$-Learning

Last section gives a proof of the convergence of neural network to the $Q$ value functions in DQN, which shows the reason that why DQN can converge as the conventional $Q$ learning approach in theory but not practically.

The other challenge, which is the slowing training speed of neural network, however, is also a hard issue to be tackled. Many algorithms has been tried to improve the training speed of neural networks in the last several decades, and the back-propagation still the main reason limits the speed of a network's convergence. In this section, a new boosting-based deep neural networks (BDNN) is attempted for the issue. Better experimental results are observed for a faster convergence and better reward returns on several benchmark experiments, which give the possibility that it can be an improving approach from conventional DQN. The detail algorithm is illustrated in section 3.1 and 3.2. A explanation on why BDNN may give a faster convergence speed and better return will be given in section 3.3. Experiment results are given in section 3.4.

### A. Structure of BDNN

Following is a graphical explanation of Boosting-based deep neural networks for $Q$-learning.(Figure (3)). A convex boosting combination of $N$ neural networks' output is used as the output of the entire network for identifying and learning the $Q$ value function, where the neighbors or a reachable set of state $s_t$ is represented as $B(s_t)$.

In the structure shown in Figure (3), $n$ Neural Networks $N^i (i = \{1 \cdots n\})$ with the same network structure (number of layers and hidden nodes)are used and connected by $n$ convex boosting coefficients $\alpha^i$, which have values within the range [0,1]. The boosting-based deep neural networks satisfy the following three properties:
1. $\sum^i \alpha^i$=1. $(i = \{1 \cdots n\})$
2. $\sum^i \alpha^i(0) N^i(0)$=$N(0)$
3. $\sum^i \alpha^i(\infty) N^i(\infty)$=$N(\infty)$

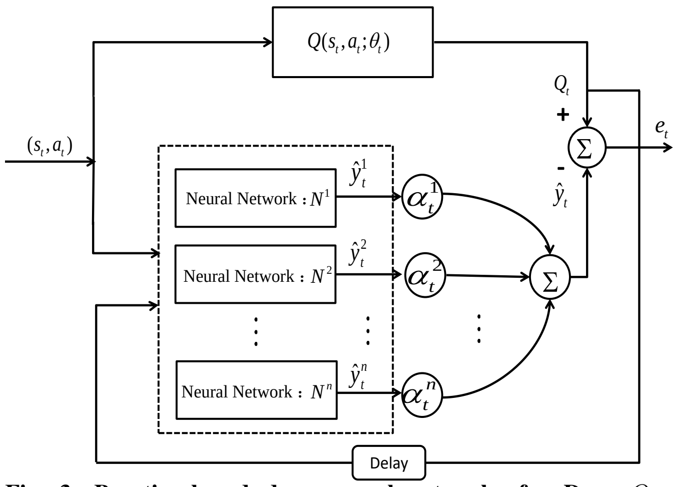

**Fig. 3: Boosting-based deep neural networks for Deep $Q$-learning (BDNN) Structure**

where $N(\cdot)$ is a virtual model satisfying the properties 2 and 3, and sharing the same structure as each single model $N^i$. The first property is the convex criteria need to be satisfied for $\alpha^i$. The second property claims that the convex sum of the initial values for each model should be equal to that of the virtual model, and the third one indicate that the convergence of each single neural network in the convex-based boosting structure should be equal to that of the virtual model.

*Comment:* The major reason to introduce a virtual model is to simplify the representation of $n$ convex boosting-based NN models. Also it will be used as a comparison model to conventional neural network for identification purpose.

### B. Mathematical Representation

According to the system structure given in Figure (3), a detailed mathematical representation of BDNN can be formulated as:

$$\begin{aligned} \lim_{t \to \infty} Q(s_t, a_t, \theta(t)) &= \lim_{t \to \infty} N(s_t, a_t, w_t) \\ &= \lim_{t \to \infty} \sum_{i=1}^{n} \alpha^i N^i(s_t, a_t, w_t) \end{aligned} \tag{12}$$

where $N^i$ are $n$ neural networks $N^i (i = \{1 \cdots n\})$ with the same network structure (number of layers and hidden nodes) but different initial conditions $N_0^i$. Also both the convex boosting coefficient $\alpha^i$ and the networks $N^i$ will satisfy the three properties demonstrated in section 3.1.

The proof of the convergence of the BDNN follows by the single neural network for identification of $Q$ value function. According to the third property given in section 3.1, i.e.

$$\sum_i \alpha^i(\infty) N^i(\infty) = N(\infty) \tag{13}$$

The outputs at convergence of BDNN for $Q$-learning structure will be the same as the result of a single NN structure. A simple mathematical explanation following Theorem 1 will

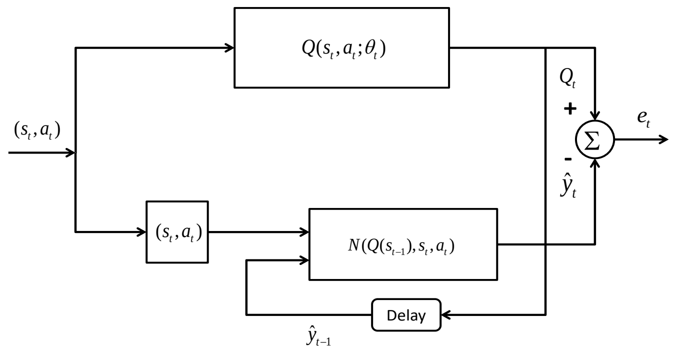

**Fig. 4: Series-Parallel Learning Model**

be like:

$$| \sum_i \alpha^i(\infty) N^i(\infty) - Q(\infty) | = | N(\infty) - Q(\infty) | < \epsilon \quad (14)$$

where the definition of $\epsilon$ and $Q$ will follow those in Theorem 1.

### C. Learning Algorithm of BDNN

The learning algorithm in BDNN for deep $Q$-learning is the most important part to be addressed in this paper. The learning procedure can be divided into two parts mainly: one is training multiple neural networks to identify the $Q$ value function, which is very similar to the conventional approach, the other is the training of convex boosting parameters using gradient based approach, which is very important for improving the performance of deep $Q$-learning.

*1) Neural Network identification using BDNN:* The learning process includes building (an) appropriate identification model(s) to estimate the real $Q$ action-value function. The basic target is to minimize the identification/learning error between the constructed neural network model and real $Q$ value function. Based on the prove Theorem 1 in section 2, by properly choosing the size and parameters of neural network, the nonlinear $Q$ value function given by (2) can be identified or learnt by NN under relatively weak initial conditions. The learning of $Q$ value function for each NN model $N_i$ in BDNN is similar to that of the single neural network case in DQN. One of the most popular identification approach using NN is the "series-parallel" model ([3]). The series-parallel model takes advantage of both the output signal $Q(s_t)$ from the real $Q$ value function and $\hat{y}_t^i$ from the estimator. The model has the form:

$$\hat{y}_t^i = N^i[Q(s_{t-1}), s_t, a_t] \quad (15)$$

where $Q(s_{t-1})$ is the real $Q$ value obtained through the online learning process at time instant $k-1$. Noticing that on the right hand of the equation, $Q(s_{t-1})$ is used to substitute $\hat{y}_{t-1}^i$ to ensure the convergence. The learning process, on the other hand, requires the accessibility of past learning system output, which is true for most of the time. Following is a graphical illustration for the series-parallel model (Figure (4)):

*Comment:* Noticing that in the for the model described in Figure (4), only one step of previous output information (i.e. k-1) is used, there are also cases that multiple previous steps of output can be fed back into the system. The discussion of the these will be included in future papers.

*2) Convex Boosting Coefficient Learning in BDNN:* An important part of the algorithm is the convex boosting coefficient learning using neural networks. The idea is generated from the second level adpative control using convex parameters in ([6], [12], [14]). Similar idea of adapting the convex weights of each model using gradient based approach is applied here to identify $Q$ functions through a boosting of multiple neural networks.

In previous section, a convex boosting-based deep neural networks is introduced. The corresponding gradient based learning approach for obtaining the convex boosting coefficients $\alpha^i$ will be discussed in this section. An explanation of why this "add-on" coefficient learning may generate a faster learning response will be discussed in next section.

Figure (3) shows a detail structure of a boosting-based neural networks. The output of the identification model $\hat{y}_t$ is a convex boosting of $n$ neural network models' outputs $\hat{y}_t^1, \cdots, \hat{y}_t^n$, as follows:

$$\hat{y}_t = \alpha_t^1 \hat{y}_t^1 + \cdots + \alpha_t^n \hat{y}_t^n \quad (16)$$

$Q_t$ is a simplified representation of the value of $Q$ function at iteration t. Also defining the system output errors as:

$$e_t = Q_t - \hat{y}_t \quad (17)$$

Substituting (17) into (16), and combining with the convex property that $\alpha^n$=1-$\sum_{i=1}^{n-1} \alpha^i$, a rearranged form of $e_t$ can be obtained

$$\begin{aligned} e_t &= Q_t - (\alpha_t^1 \hat{y}_t^1 + \cdots + \alpha_t^n \hat{y}_t^n) \\ &= \sum_{i=1}^{n} \alpha_t^i Q_t - (\alpha_t^1 \hat{y}_t^1 + \cdots + \alpha_t^n \hat{y}_t^n) \\ &= \sum_{i=1}^{n-1} \alpha_t^i e_t^i + \alpha_t^n e_t^n \\ &= \sum_{i=1}^{n-1} \alpha_t^i e_t^i + (1 - \sum_{i=1}^{n-1} \alpha_t^i) e_t^n \\ &= \sum_{i=1}^{n-1} \alpha_t^i \tilde{e}_t^i + e_t^n \end{aligned} \quad (18)$$

where $e_t^i$ is defined as the output error between the $i^{th}$ neural network model and $Q$ value function in $t^{th}$ iteration, i.e. $Q_t - \hat{y}_t^i$, and $\tilde{e}_t^i$ is the difference between $e_t^i$ and $e_t^i$, i.e. $\tilde{e}_t^i \triangleq e_t^i - e_t^n$, which is the error differences between the $i^{th}$ and $n^{th}$ model.

The error equation obtained from (18) can be further simplified when $k \to \infty$ and $\lim_{k \to \infty} e_t = 0$:

$$-e_t^n = \mathbf{\tilde{E}_t^T} \mathbf{\tilde{\alpha}_t} \quad (19)$$

where $-e_t^n$ is the error between the $n^{th}$ model and $Q$ value function in $k^{th}$ iteration. $\mathbf{\tilde{E}} \in \mathbb{R}^{1 \times (n-1)}$ is a vector defined as $[\tilde{e}^1, \cdots, \tilde{e}^{n-1}]$, as well as $\tilde{\alpha} = [\alpha^{(1)}, \cdots, \alpha^{(n-1)}] \in$

$\mathbb{R}^{1\times(n-1)}$.

The update rule for $\tilde{\alpha}$ can also be derived by multiplying $\tilde{\mathbf{E}}$ on both sides of the equation, and move the left hand side of the equation to the right:

$$\begin{aligned}\tilde{\alpha}_{\mathbf{t}} - \tilde{\alpha}_{\mathbf{t-1}} &= -\tilde{\mathbf{E}}\tilde{\mathbf{E}}^{\mathbf{T}}\tilde{\alpha}_{\mathbf{t-1}} - \tilde{\mathbf{E}}\mathbf{e}_{\mathbf{t}}^{\mathbf{n}} \\ \tilde{\alpha}_{\mathbf{t}} &= \tilde{\alpha}_{\mathbf{t-1}} - \tilde{\mathbf{E}}\tilde{\mathbf{E}}^{\mathbf{T}}\tilde{\alpha}_{\mathbf{t-1}} - \tilde{\mathbf{E}}\mathbf{e}_{\mathbf{t}}^{\mathbf{n}}\end{aligned} \tag{20}$$

Hence, equation (20) has become the new back propagation law for updating the convex parameter $\tilde{\alpha}_t$, which will give us the first $n-1$ elements in the convex coefficient vectors $\alpha = [\alpha^1, \cdots, \alpha^n]$. By the convex property of $\alpha$, the last element $\alpha^n$ can be obtained by $1 - \sum_{i=1}^{n-1} \alpha^i$ once $\tilde{\alpha}$ is obtained.

Also, for each single neural network model, its weights will be updated by the standard back-propagation law using the model error $e^i$, which has been illustrated in detail in the first part of section 3.3. It should be noticed that the update procedure of $\tilde{\alpha}$ and each network's weights are both on-line and in a simultaneous manner.

## D. Performance analysis of BDNN

In the boosting-based multiple neural networks structure, a new set of convex boosting coefficient vector $\alpha$ is introduced. The question involved is that how this new parameter can change the performance of neural networks. Two properties will be claimed here as below:
1) The error between $\alpha$ and its true value $\alpha^*$ is decreasing exponentially with respect to the iteration round number $t$.
2) The BDNN learning system will converge when $\alpha$ converges, i.e. $\sum_1^n \alpha^i N_t^i = y_t$, once $\alpha^i = \alpha^{i*}$ for all $i \in \{1 \cdots n\}$.

The first property can be easily obtained from equation (20). As it is shown, the change difference of $\alpha$ is proportional to its last iteration's value. Hence, an exponential property will be given by solving the difference equation. The interesting property is the second one, which indicates that the convergence speed of $\alpha$ dominates the identification speed , as that of conventional back-propagation part for training the weights of each neural networks is far inferior than the exponential convergence. The error $e$ is defined as the convex boosting of all models' errors:

$$\begin{aligned}e_t &= \sum_{i=1}^{n} \alpha_t^i N^i - Q_t \\ &= \sum_{i=1}^{n} \alpha_t^i (N^i - Q_t) \\ &= \sum_{i=1}^{n} \alpha_t^i e_t^i\end{aligned} \tag{21}$$

where the convex boosting coefficient property of $\sum_{i=1}^n \alpha^i = 1$ is applied. Equation (21) indicates that only the convex sum of error $e_i$, i.e. $\sum_{i=1}^n \alpha^i e^i$, needs to be zero for identification, instead of single model error $e_i \to 0$. It gives the potential reason that why the boosting-based approach gives a better learning response than the conventional DQN.

## E. Choice of n - number of attention coefficients/neural networks

The convex property of $\sum_{i=1}^n \alpha^i = 1$ gives us a possibility to design the gradient based law as shown in equation (20), by ensuring the robustness of the system and speed of convergence at the same time. Also, noticing that $\alpha_i$ is within a range $[0, 1]$, which gives us a relatively short range for parameter to update, which is another potential reason that why it may converge much faster than the network itself using back-propagation. The choice of $n$, which is the number of neural networks or convex coefficients, depends on the system complexity, will be briefly discussed here.

Assume the output of the $Q$ value function has a dimensionality of $M$, which is formed by the product of number of states $S$ and number of actions $A$, i.e. $M = S \times A$.

By the definition of convex, in order that the convex boosting coefficients given by equation (16) has at least one solution, the number of neural networks or convex boosting coefficients $n$ should be equal or larger than $M + 1$, which gives us a lower bound of $n$.

On the other side, the upper bound of $n$ is not limited, however, the speed of the training process will be greatly affected as the number of neural networks increases. This should be another factor how an appropriate $n$ to be selected.

## IV. Experiment

### A. Setup

In our experiment, we are testing our new BDNN based $Q$-leaning algorithm with several other popular reinforcement algorithm on three benchmark experiments (Walker, Swimmer and Cheetah) as given in ([22]). We first compared our algorithm with several policy-based algorithms, including DDPG ([23]),TNPG ([24], [28], [29]) and TRPO ([25]). And furthermore, we added DQN ([13]) into our comparison list as our algorithm is based on it as well. The experiment setup using DDPG ,TNPG and TRPO follows the one given in [22]. For DQN and our BDNN model, we use a feedforward neural network with 3 hidden layers, consisting of 100 hidden units with tanh activation function at the first two hidden layers.

### B. Results

We evaluate the model's performance based on the number of iterations for the convergence of each task, and the average returns as defined in [22].

**TABLE I: Performance of different models in terms of returns on three RL tasks**

| Model | Walker | Swimmer | Cheetah |
|---|---|---|---|
| TNPG [24] | 1382.6 | 96.0 | 1729.5 |
| TRPO [25] | 1353.8 | 96.0 | 1914.0 |
| DDPG [23] | 318.4 | 85.8 | 2148.6 |
| DQN [30] | 121.2 | 32.3 | 765.3 |
| BDNN | **1389.6** | **98.1** | **2245.3** |

Table 1 gives a comparison of the average returns by performing three tasks. It is shown that BDNN outperforms all the other approaches in terms of the average returns. It worth notice that due to the value-based property of DQN, it gives the worst result in these tasks as they are continuous control problem and the action space is also continuous. Our BDNN model based on deep $Q$ network overcomes this disadvantages and gives far better results.

**TABLE II: Performance for different models in terms of convergence iterations on three RL tasks**

| Model | Walker | Swimmer | Cheetah |
|---|---|---|---|
| TNPG [24] | 524 | 26 | 152 |
| TRPO [25] | 636 | 23 | 183 |
| DDPG [23] | 423 | 18 | 98 |
| DQN [30] | 925 | 35 | 368 |
| BDNN | **82** | **5** | **32** |

Table 2 shows the number of iterations required for the convergence of each model (no future improvement in 20 consecutive iterations). It is also shown that our new BDNN model demonstrate at least 3 times faster convergence speed compared to the other four approaches.

## V. Conclusion

In this paper, a new concept of using boosting-based neural networks for reinforcement learning is proposed. By firstly proving the convergence of neural network identification for general $Q$-learning algorithm (DQN), the boosting-based neural networks algorithm is discussed with a focus on the convex boosting coefficients learning. Detailed model explanations and performance analysis are also given mathematically. Simulations based on benchmark RL experiments are designed at the end to compare the performance of our BDNN model with other popular reinforcement learning schemes. At a result, BDNN outperforms all other approaches by comparing their average returns and convergence speed. Though more experiments should be conducted on real applications in further papers, it can conclude the BDNN approach may be a potential choice to speed up the reinforcement learning procedure in a general context.

## References

[1] Bellman, Richard. "Dynamic programming and Lagrange multipliers." Proceedings of the National Academy of Sciences 42.10 (1956): 767-769.

[2] Bryson, Arthur E., and Ho Yu Chi. "Applied optimal control." (1969).

[3] Narendra, Kumpati S., and Kannan Parthasarathy. "Identification and control of dynamical systems using neural networks." IEEE Transactions on neural networks 1.1 (1990): 4-27.

[4] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[5] Sutton, Richard S. "Learning to predict by the methods of temporal differences." Machine learning 3.1 (1988): 9-44.

[6] Han, Zhuo, and Narendra, Kumpati S. "New concepts in adaptive control using multiple models." IEEE Transactions on Automatic Control 57.1 (2012): 78-89.

[7] Rummery, Gavin A., and Mahesan Niranjan. On-line $Q$-learning using connectionist systems. University of Cambridge, Department of Engineering, 1994.

[8] Narendra, Kumpati S., Yu Wang, and Snehasis Mukhopadyhay. "Fast Reinforcement Learning using Multiple Models.", 2016 Control and Decision Conference, Las Vegas

[9] Powell, Warren B. Approximate Dynamic Programming: Solving the curses of dimensionality. Vol. 703. John Wiley & Sons, 2007.

[10] Werbos, Paul J. "A menu of designs for reinforcement learning over time." Neural networks for control (1990): 67-95.

[11] Glscher, Jan, et al. "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning." Neuron 66.4 (2010): 585-595.

[12] Narendra, Kumpati S., Yu Wang, and Wei Chen. "The Rationale for Second Level Adaptation." arXiv preprint arXiv:1510.04989 (2015).

[13] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

[14] Narendra, Kumpati S., Yu Wang, and Wei Chen. "Stability, robustness, and performance issues in second level adaptation." 2014 American Control Conference. IEEE, 2014.

[15] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[16] Gu, Shixiang, et al. "Continuous deep q-learning with model-based acceleration." arXiv preprint arXiv:1603.00748 (2016).

[17] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double $Q$-Learning." AAAI. 2016.

[18] Watkins, Christopher JCH, and Peter Dayan. "$Q$-learning." Machine learning 8.3-4 (1992): 279-292.

[19] Watkins, Christopher John Cornish Hellaby. Learning from delayed rewards. Diss. University of Cambridge, 1989.

[20] Rummery, Gavin A., and Mahesan Niranjan. On-line $Q$-learning using connectionist systems. University of Cambridge, Department of Engineering, 1994.

[21] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[22] Duan Y, Chen X, Houthooft R, et al. Benchmarking deep reinforcement learning for continuous control[C]//International Conference on Machine Learning. 2016: 1329-1338.

[23] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[24] Kakade, Sham M. "A natural policy gradient." Advances in neural information processing systems. 2002.

[25] Schulman, John, et al. "Trust region policy optimization." International Conference on Machine Learning. 2015.

[26] Yu Wang, and Xiaoxi Zhu. "A Supervised Adaptive Learning-based Fuzzy Controller for a non-linear vehicle system using Neural Network Identification." American Control Conference (ACC), 2016. IEEE, 2016.

[27] Narendra, Kumpati S., Snehasis Mukhopadyhay, and Yu Wang. "Improving the Speed of Response of Learning Algorithms Using Multiple Models: An Introduction.", the 17th Yale Workshop on Adaptive and Learning Systems

[28] Peters, Jan, Sethu Vijaykumar, and Stefan Schaal. "Policy gradient methods for robot control." Rapport nCS-03-787, University of Southern California (2003).

[29] Bagnell, J. Andrew, and Jeff Schneider. "Covariant policy search." IJCAI, 2003.

[30] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[31] Wang, Yu. "A new concept using LSTM Neural Networks for dynamic system identification." American Control Conference (ACC), 2017. IEEE, 2017.