DeepMCTS: Deep Reinforcement Learning Assisted Monte Carlo Tree Search for MIMO Detection

Tz-Wei Mo, Ronald Y. Chang, and Te-Yi Kan
Research Center for Information Technology Innovation, Academia Sinica, Taiwan
Email: tzwmo1998@gapp.nthu.edu.tw, rchang@citi.sinica.edu.tw, dexter.ty.kan@gmail.com

***Abstract*—This paper proposes a multiple-input multiple-output (MIMO) symbol detector that incorporates a deep reinforcement learning (DRL) agent into the Monte Carlo tree search (MCTS) detection algorithm. A self-designed deep reinforcement learning agent, consisting of a policy value network and a state value network, is trained to detect MIMO symbols. The outputs of the trained networks are adopted into a modified MCTS detection algorithm to provide useful node statistics and facilitate enhanced tree search process. The resulted scheme, termed the DeepMCTS detector, demonstrates significant performance and complexity advantages over the original MCTS detection algorithm under varying channel conditions.**

***Index Terms*—MIMO detection, neural networks, deep reinforcement learning, Monte Carlo tree search.**

I. Introduction

The computational complexity of an optimal maximum likelihood (ML) detector for multiple-input multiple-output (MIMO) systems grows exponentially with the number of transmit antennas. Many previous works have pursued suboptimal detectors with reduced complexity at the cost of various degrees of performance degradation [1]. Linear suboptimum detectors such as zero forcing (ZF) and minimum mean square error (MMSE) generally have suffered performance, but have also been shown to reach ML-approaching performance as the number of antennas increases, due to a "channel-hardening" effect [2]. However, the complexity of high-order matrix inversion in linear methods also becomes prohibitive in large systems, and, as such, matrix inversion approximation methods have been proposed to accelerate the inversion process [3].

Another set of detectors use nonlinear methods. Interference cancellation [4], local search [5], and approximate message passing (AMP) [6] are all well-known nonlinear detection methods. Tree search algorithms are also used in detectors [7], [8], where the MIMO detection problem is formulated as a decision tree and a symbol is recovered at each layer of the tree. In [9], a statistical approach with the Monte Carlo tree search (MCTS) algorithm was proposed with hardware acceleration. Nonlinear methods typically achieve better performance than linear methods, but most nonlinear methods require some parameter tuning, which can be difficult to determine.

In recent years, deep neural networks (DNN) attract great attention and also find applications in MIMO detection. In [10], MIMO detection was modeled as a classification problem and then solved using DNN and convolutional neural network (CNN) approaches. DetNet [11] uses DNNs to unfold a projected gradient descent algorithm. WeSNet [12] improves DetNet by introducing a weight scaling framework which further reduces the network size and allows the network to self-adjust to the detection complexity. In [13], DNNs were used to unfold two belief propagation (BP) algorithms with two separate networks. In [14], DNN was incorporated into the soft successive interference cancellation (SIC) algorithm. In [15], layers of DNNs were used to implement algorithmic iterations in an orthogonal AMP (OAMP) algorithm.

The DNN approaches mentioned above all show improvements over traditional methods. However, since these neural network methods are trained using supervised learning, they face a labeling dilemma over whether to use the ML solutions or the actual transmitted symbols as labels during training [16]. Using the actual transmitted sequence may cause the network to learn undesirable boundaries. The ML solution required for training, however, is difficult to obtain for large MIMO systems. Thus, it was suggested [16] that the neural networks be trained under a fixed signal-to-noise ratio (SNR) corresponding to a 10^{-2} error probability in the actual transmitted sequence. This suggestion however only has empirical basis, and, to our knowledge, there still lacks definite answers.

In this work, we propose to incorporate a deep reinforcement learning (DRL) architecture into the traditional MCTS algorithm for MIMO detection. Unlike the aforementioned DNN approaches, we do not unfold existing algorithms; instead, our DNNs are used to improve the MCTS detection algorithm by providing useful statistics given the channel information and the received symbol vector. The proposed scheme proves significantly more effective than either the DRL or MCTS alone. Besides, by using DRL and a reward system to calculate losses instead of using labels from the transmitted symbol vectors, we avoid the previously mentioned labeling dilemma. Only the received symbol vector, channel information, and an estimate of the transmitted symbol vector are required in the reward system.

Notations: $(\cdot)^*$, $(\cdot)^\top$, and $(\cdot)^\dagger$ denote complex conjugate, transpose, and conjugate transpose, respectively. $\|\cdot\|$ denotes the l_2-norm of a vector. $\Re(\cdot)$ and $\Im(\cdot)$ denote the real and imaginary parts of a complex number, respectively. $|\cdot|$ denotes

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grants MOST 106-2628-E-001-001-MY3 and MOST 109-2221-E-001-013-MY3.

the cardinality of a set.

II. System Model and Monte Carlo Tree Search (MCTS) Algorithm

A. System Model

We consider an uncoded $N_T \times N_R$ MIMO system with N_T transmit antennas and N_R receive antennas. The complex baseband received signal can be expressed as

$$\mathbf{y}_c = \mathbf{H}_c \widetilde{\mathbf{x}}_c + \mathbf{w}_c. \tag{1}$$

Here, $\widetilde{\mathbf{x}}_c$ is the $N_T \times 1$ transmit signal vector containing uncorrelated entries selected equiprobably from a square quadrature amplitude modulation (QAM) alphabet $\mathcal{S} = \{a+ib \,|\, a,b \in \mathcal{Q}\}$, where $\mathcal{Q}$ is the pulse amplitude modulation (PAM) alphabet. $\widetilde{\mathbf{x}}_c$ has zero mean and covariance matrix $\sigma_x^2 \mathbf{I}_{N_T}$. $\mathbf{H}_c$ is the $N_R \times N_T$ channel matrix containing independent and identically distributed (i.i.d.) complex Gaussian elements with zero mean and unit variance, and is assumed perfectly known at the receiver but not at the transmitter. $\mathbf{w}_c$ is the additive white Gaussian noise (AWGN) with i.i.d. complex elements and has zero mean and covariance matrix $\sigma_w^2 \mathbf{I}_{N_R}$. The complex signal model (1) can be transformed to an equivalent real signal model by defining $\mathbf{y}' = [\Re(\mathbf{y}_c)\, \Im(\mathbf{y}_c)]^\top$, $\widetilde{\mathbf{x}} = [\Re(\widetilde{\mathbf{x}}_c)\, \Im(\widetilde{\mathbf{x}}_c)]^\top$, $\mathbf{w} = [\Re(\mathbf{w}_c)\, \Im(\mathbf{w}_c)]^\top$, and $\mathbf{H} = \begin{bmatrix} \Re(\mathbf{H}_c) & -\Im(\mathbf{H}_c) \\ \Im(\mathbf{H}_c) & \Re(\mathbf{H}_c) \end{bmatrix}$. The resulting real-valued model is given by

$$\mathbf{y}' = \mathbf{H}\widetilde{\mathbf{x}} + \mathbf{w}, \tag{2}$$

where $\mathbf{y}' \in \mathbb{R}^n$, $\mathbf{H} \in \mathbb{R}^{n \times m}$, $\widetilde{\mathbf{x}} \in \mathcal{Q}^m$, and $\mathbf{w} \in \mathbb{R}^n$, with $n = 2N_R$ and $m = 2N_T$.

It is well-known that the optimal maximum likelihood (ML) detector to recover the transmitted symbol vector $\widetilde{\mathbf{x}}$ from $\mathbf{y}'$ is to find $\widetilde{\mathbf{x}}_{\text{ML}} = \arg\min_{\mathbf{x} \in \mathcal{Q}^m} \|\mathbf{y}' - \mathbf{H}\mathbf{x}\|^2$. By a QR decomposition of the channel matrix $\mathbf{H}$ such that $\mathbf{H} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{R}^{n \times m}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{m \times m}$ is an upper-triangular matrix, the ML detection criterion can equivalently be expressed as $\widetilde{\mathbf{x}}_{\text{ML}} = \arg\min_{\mathbf{x} \in \mathcal{Q}^m} \|\mathbf{y} - \mathbf{R}\mathbf{x}\|^2$, where $\mathbf{y} = \mathbf{Q}^\top \mathbf{y}'$. Due to the upper-triangular structure of $\mathbf{R}$, $\|\mathbf{y} - \mathbf{R}\mathbf{x}\|^2$ can be expanded as the summation of m terms, where the $(m-k+1)$th term (for $k = 1, 2, \ldots, m$) depends only on the partial symbol vector $\mathbf{x}_k^m \triangleq (x_k, \ldots, x_m)^\top \in \mathcal{Q}^{m-k+1}$:

$$(y_m - r_{m,m}x_m)^2 + \Big(y_{m-1} - \sum_{i=m-1}^{m} r_{m-1,i}x_i\Big)^2 + \cdots + \Big(y_1 - \sum_{i=1}^{m} r_{1,i}x_i\Big)^2, \tag{3}$$

where y_i is the ith element of $\mathbf{y}$, x_i is the ith element of $\mathbf{x}$, and $r_{i,j}$ is the (i,j)-entry of $\mathbf{R}$. We denote the $(m-k+1)$th term in (3) by $b(\mathbf{x}_k^m)$ and the summation of the first $m-k+1$ terms by $d(\mathbf{x}_k^m)$. Then, ML detection becomes

$$\widetilde{\mathbf{x}}_{\text{ML}} = \underset{\mathbf{x}_1^m \in \mathcal{Q}^m}{\arg\min}\; d(\mathbf{x}_1^m). \tag{4}$$

Note that (3) creates a rooted tree structure that allows sequentially detecting the transmitted signal vector $\mathbf{x}$ from $\mathbf{x}_m^m$ to $\mathbf{x}_1^m$, one element at a time. Specifically, we can first recover x_m and calculate $b(\mathbf{x}_m^m)$, and then recover x_{m-1} with known x_m to obtain $\mathbf{x}_{m-1}^m$ and calculate $b(\mathbf{x}_{m-1}^m)$, etc.

B. Monte Carlo Tree Search (MCTS) Algorithm

The MCTS algorithm outputs a best action in a discrete action space based on the current state. In MIMO detection, an action is defined as recovering an element of the transmitted symbol $\mathbf{x}$, and the action space is the PAM alphabet $\mathcal{Q}$ with $|\mathcal{Q}|$ discrete actions. A terminal state is reached when a complete symbol vector $\mathbf{x}_1^m = \mathbf{x}$ is recovered. A total of m actions/tree searches are required to reach a terminal state. Initially, the Monte Carlo tree consists of only the root node, and as the search progresses, the tree expands by adding new leaf nodes to the existing tree. The root node of the search tree is the current state at some step l ($l = 0, 1, 2, \ldots, m-1$) and represents the partial symbol vector $\mathbf{x}_{m-l+1}^m$. The algorithm aims to determine the best x_{m-l} at step l such that (3) is minimized at the end of detection. Each node i in the tree is associated with two node statistics: i) the expected $-d(\mathbf{x}_1^m)$ that this node will achieve by going downward through layers of tree from this node, which is denoted as $\overline{D}^{(i)}$, and ii) the number of times this node has been selected during the MCTS process, which is denoted as $v^{(i)}$. Note that a negative sign is added in front of $d(\mathbf{x}_1^m)$ to convert a metric minimization problem in (4) into a reward maximization problem in the framework of MCTS. $\overline{D}^{(i)}$ and $v^{(i)}$ are initially set to zero.

The MCTS algorithm comprises four phases:

1) *Selection:* Starting from the root node, successive child nodes that maximize the upper confidence bound for trees (UCT) [17], defined as

$$\overline{D}^{(i)} + c_{\text{uct}}\sqrt{\frac{\ln V^{(i)}}{v^{(i)}}}, \tag{5}$$

are selected until a leaf node is reached. Here, $V^{(i)}$ is the number of times the parent node of a node i has been selected, and c_{uct} is a positive-valued exploration parameter. The first term of UCT enforces a reward-based selection, and the second term of UCT encourages exploration.
2) *Expansion:* If the selected leaf node is not a terminal state, $|\mathcal{Q}|$ child nodes are added to the tree. Otherwise, no new leaf node will be added, and Simulation will be skipped and Backpropagation will be performed.
3) *Simulation:* Randomly select one of the child nodes that has just been added to the tree in Expansion, and simulate detection by expanding the partial symbol vector with random elements $x \in \mathcal{Q}$ until $\mathbf{x}_1^m$ is reached.
4) *Backpropagation:* The mock detection result $\mathbf{x}_1^m$ from either Simulation or Expansion is evaluated by calculating $-d(\mathbf{x}_1^m)$. The value is fed back upward toward the root node of the tree to update node statistics of the leaf node and all the nodes selected during the MCTS process. Specifically, $\overline{D}^{(i)}$ is updated by averaging all $-d(\mathbf{x}_1^m)$ values backpropagated from the leaf node below node i,

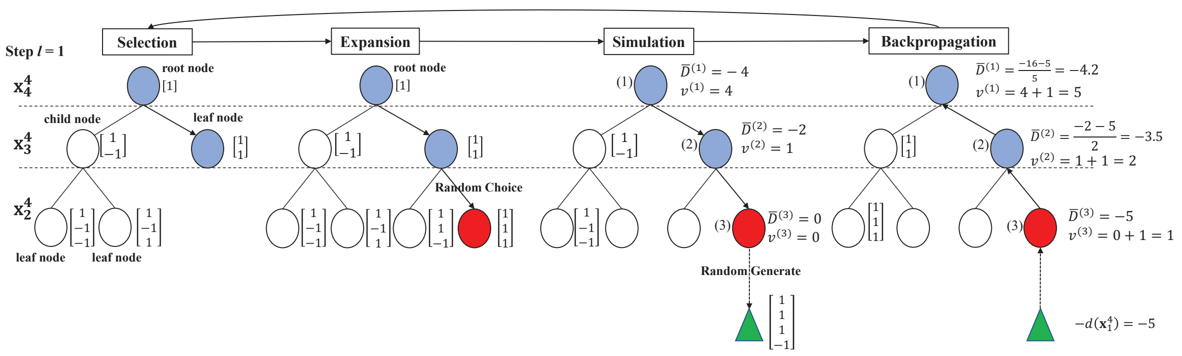

Fig. 1. MCTS for MIMO detection, for the example of a 2×2 QPSK system ($m = 4$ and $|\mathcal{Q}| = 2$). MCTS algorithm at step $l = 1$ is shown. Each node in the tree represents a unique partial symbol vector. In Selection, a leaf node is reached using UCT. During Expansion, all possible child nodes are added to the tree and become the new leaf nodes. In Simulation and Backpropagation, where the node index i is shown to the left side of each node in parentheses and the node statistics $\overline{D}^{(i)}$ and $v^{(i)}$ are shown to the right side of each node, node statistics are updated by backpropagating the simulated $-d(\mathbf{x}_1^4)$ result.

and $v^{(i)}$ is updated by adding by one if node i is on the backpropagation path.

A completion of these four phases is termed one *playout*. After completing all playouts from the current state, the action represented by the best child node (corresponding to the largest $\overline{D}^{(i)}$) among all child nodes of the root node is selected as the best action x_{m-l} at step l. Fig. 1 illustrates the MCTS algorithm in search of the best action x_{m-l} at each step l (here, $l = 1$ is shown).

III. The Proposed Deep Reinforcement Learning Assisted Monte Carlo Tree Search (DeepMCTS) Algorithm

It is observed that the MCTS algorithm relies on *random* sampling in the Simulation phase. Besides, while UCT allows sampling paths efficiently by growing asymmetrical trees and focusing more on promising subtrees, good solutions could still be overlooked if the sample size is not large enough. We propose to enhance MCTS with a learned and guided Simulation process enabled by DRL. The DRL architecture consists of a policy value network and a state value network, both taking some defined state $\mathbf{s}_l$ at step l of the detection process as input, and outputting the *policy value* $\widehat{\mathbf{p}}_l$ and the *state value* u_l, respectively. Here, $\widehat{\mathbf{p}}_l$ has $|\mathcal{Q}|$ elements, representing the probability distribution over $|\mathcal{Q}|$ discrete possible moves, and u_l represents the final expected $-d(\mathbf{x}_1^m)$ of that state s_l. The two outputs are integrated into the MCTS algorithm, where the policy value is adopted into the UCT for enhanced exploration, and the state value is used to replace the Simulation phase as the expected final outcome.

A. State, Action, and Reward

We first define the state, action, and reward for DRL for the MIMO detection problem:

1) State: The state vector $\mathbf{s}_l$ at step l ($l = 0, \ldots, m-1$) consists of some compressed sufficient representations of the channel matrix $\mathbf{H}$, the received signal $\mathbf{y}'$, the partial symbol vector $\mathbf{x}_{m-l+1}^m$ and its metrics, i.e., $\mathbf{s}_l = \left[\mathbf{y}, \mathbf{y}', \mathbf{H}^\top\mathbf{y}', \boldsymbol{\chi}_{m-l+1}^m, \boldsymbol{\chi}_{m-l+2}^m, b(\mathbf{x}_{m-l+1}^m), d(\mathbf{x}_{m-l+1}^m)\right]^\top$, where $\boldsymbol{\chi}_{m-l+1}^m = [\mathbf{0}_{1\times(m-l)}, (\mathbf{x}_{m-l+1}^m)^\top]^\top$ and $\boldsymbol{\chi}_{m-l+2}^m$ (similarly defined) are the zero-padded partial symbol vectors at step l and step $l-1$, respectively, both having a fixed dimension $m \times 1$. Note that at step $l = 0$, the last four entries of $\mathbf{s}_l$ degenerate to zero vectors/scalars; at step $l = 1$, $\boldsymbol{\chi}_{m-l+2}^m = \boldsymbol{\chi}_{m+1}^m = \mathbf{0}_{m\times 1}$.

2) Action: The action a_l at step l recovers x_{m-l} with $|\mathcal{Q}|$ possible actions.

3) Reward: The reward value r_l for an action a_l at step l is $-d(\mathbf{x}_{m-l}^m)$. Note that the reward for DRL slightly differs from the Simulation phase in the MCTS algorithm, where only $-d(\mathbf{x}_1^m)$ of the full recovered symbol vector is used to update the node statistics.

The training data for the policy value network and state value network are collected by sending N transmit symbol vectors through a fixed channel and recording the detection process at the receiver with the tuples $(s_l^j, a_l^j, r_l^j, s_{l+1}^j)$, where the superscript denotes the jth transmit symbol vector ($j = 1, 2, \ldots, N$).

B. Policy Value Network

The policy value network outputs a vector $\widehat{\mathbf{p}}_l$ representing the probability distribution over $|\mathcal{Q}|$ actions in the action space. Our policy value network uses an actor-critic architecture consisting of two fully-connected neural networks. The actor network outputs $\widehat{\mathbf{p}}_l$ and the critic network outputs a scalar value q_l representing the expected discounted cumulative reward, or Q-value, from the current input state until the terminal state. The loss function of the critic network incorporates the squared temporal difference (TD) error, i.e.,

$$L(\boldsymbol{\theta}_c) = \frac{1}{N}\sum_{j=1}^{N}\sum_{l=0}^{m-1}(\mathrm{TD})^2 + c_1\|\boldsymbol{\theta}_c\|^2, \tag{6}$$

where $\mathrm{TD} \triangleq r_l^j + \gamma^{m-l}q_{l+1}^j - q_l^j$ is the TD error, with $\gamma \in [0.9, 1]$ being a discount factor, and q_l^j and q_{l+1}^j being the critic network outputs for input states $\mathbf{s}_l$ and $\mathbf{s}_{l+1}$, respectively, for the jth transmit symbol vector; $\boldsymbol{\theta}_c$ is the critic network parameters; and c_1 is a scaling constant. The last term of (6) is the $l2$ regularization of the network parameters. Note that $q_m^j \triangleq 0$,

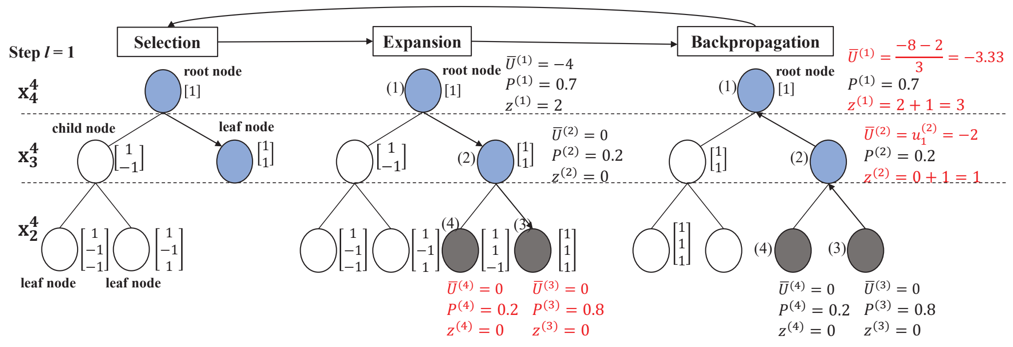

Fig. 2. (Cf. Fig. 1, for the same example of a 2×2 QPSK system) DeepMCTS algorithm at step $l = 1$ is shown. In Selection, a leaf node is reached. During Expansion, all possible child nodes are added to the tree and become the new leaf nodes. In Expansion and Backpropagation, where the node index i is shown to the left side of each node in parentheses and the node value $\overline{U}^{(i)}$, prior node probability $P^{(i)}$, and selection count $z^{(i)}$ are shown to the right side of each node, child nodes are added to the tree (Expansion) and node statistics are updated and backpropagated (Backpropagation).

since after step $l = m - 1$ the detection terminates and there is no expected reward.

The loss function of the actor network is defined as

$$L(\boldsymbol{\theta}_a) = \frac{1}{N}\sum_{j=1}^{N}\sum_{l=0}^{m-1}\Big(\text{TD} \times \mathcal{L}_{\text{CE}}(\mathbf{p}_l^j, \widehat{\mathbf{p}}_l^j) - c_2 \times \mathcal{L}_{\text{CE}}(\widehat{\mathbf{p}}_l^j, \widehat{\mathbf{p}}_l^j)\Big) + c_3\|\boldsymbol{\theta}_a\|^2, \quad (7)$$

where $\mathcal{L}_{\text{CE}}(\cdot,\cdot)$ is the cross-entropy function

$$\mathcal{L}_{\text{CE}}(\mathbf{p}_l^j, \widehat{\mathbf{p}}_l^j) = -\sum_{k=1}^{|\mathcal{Q}|}\mathbf{p}_{l,k}^j \log \widehat{\mathbf{p}}_{l,k}^j, \quad (8)$$

and $\boldsymbol{\theta}_a$ is the actor network parameters. $\mathbf{p}_l^j$ represents the true probability of action taken in step l for the jth transmit symbol vector. In fact, $\mathbf{p}_l^j$ is the one-hot encoding of the element x_{m-l} recovered at step l. $\widehat{\mathbf{p}}_l^j$ is the predicted probability distribution from the actor network. The subscript k in (8) denotes the kth element in $\mathbf{p}_l^j$ and $\widehat{\mathbf{p}}_l^j$. c_2 and c_3 are scaling constants. This actor-critic network architecture allows for training a strong actor to output good policy values to increase the efficiency and accuracy of the MCTS algorithm for MIMO detection.

C. State Value Network

The state value network outputs a scalar u_l as the predicted $-d(\mathbf{x}_1^m)$ to replace Simulation in the original MCTS algorithm. The input to the state value network at step l is $\mathbf{s}_l$ concatenated with the policy value network output $\widehat{\mathbf{p}}_l$, i.e., $\mathbf{s}_l' = [\mathbf{s}_l^\top, \widehat{\mathbf{p}}_l^\top]^\top$. The loss function of the state value network is defined as

$$L(\boldsymbol{\theta}_s) = \frac{1}{N}\sum_{j=1}^{N}\sum_{l=0}^{m-1}(u_l^j - D^j)^2 + c_4\|\boldsymbol{\theta}_s\|^2, \quad (9)$$

where $\boldsymbol{\theta}_s$ is the network parameters, u_l^j is the state value network output at step l for the jth transmit symbol vector, D^j is the final $-d(\mathbf{x}_1^m)$ for the jth transmit symbol vector and is obtained after step $l-1$, and c_4 is a scaling constant.

D. The DeepMCTS Algorithm

By a slight abuse of notation, we use $\widehat{\mathbf{p}}_l^{(i)}$ and $u_l^{(i)}$ to denote the outputs of the policy value network and state value network, respectively, corresponding to the transmit symbol vector represented by node i in the tree at step l. Each node i in the tree is associated with three node statistics: i) the expected $-d(\mathbf{x}_1^m)$, or *node value*, that this node i will achieve as evaluated by the state value network output, which is denoted as $\overline{U}^{(i)}$; ii) a *prior node probability*, $P^{(i)}$, which is taken directly from one of the elements in the policy value network output of the parent node of node i; and iii) the number of times this node i has been selected, which is denoted as $z^{(i)}$. $\overline{U}^{(i)}$ and $z^{(i)}$ are initialized to 0, while $P^{(i)}$ is initialized directly from the policy value network output of the parent node of node i.

The proposed DeepMCTS algorithm comprises three phases:

1) *Selection:* The process is the same as the Selection phase in the MCTS algorithm, except that a variant of UCT, defined as

$$\overline{U}^{(i)} + c_{\text{puct}}P^{(i)}\frac{\sqrt{Z^{(i)}}}{1 + z^{(i)}}, \quad (10)$$

is used for node selection. Here, $Z^{(i)}$ is the number of times the parent node of node i has been selected.

2) *Expansion:* If the selected leaf node i is not a terminal state, $\widehat{\mathbf{p}}_l^{(i)}$ and $u_l^{(i)}$ are evaluated, where the $|\mathcal{Q}|$ values in $\widehat{\mathbf{p}}_l^{(i)}$ are used to initialize the prior node probabilities of $|\mathcal{Q}|$ child nodes as the child nodes are added to the tree. If the selected leaf node is a terminal state, no new leaf node will be added, and Backpropagation will be performed.

3) *Backpropagation:* If the leaf node i is not a terminal state, $u_l^{(i)}$ is backpropagated, where the node value of each node in the selection path is updated by averaging all recorded node values of that node, and $z^{(i)}$ is updated by adding by one if node i is on the backpropagation path. If the leaf node i is a terminal state, directly compute $-d(\mathbf{x}_1^m)$ and the value is backpropagated toward the root node to update the statistics of all the nodes selected during the Selection phase.

After completing a designated number of playouts from the current state, the action represented by the best explored child node (corresponding to the largest $z^{(i)}$) among all child nodes of the root node is selected as the best action x_{m-l} at step l. Fig. 2 illustrates the DeepMCTS algorithm.

IV. Simulation Results

A. Settings

All schemes are tested on the same hardware (Intel i7-8700 CPU processor) for a fair comparison. The networks of DeepMCTS are optimized using the RMSProp optimizer. The learning rates for the actor, critic, and state value network are all initially set to 0.0001. The discount value γ in (6) is set to 0.95, and the $l2$ regularization coefficients c_1, c_3, and c_4 are all set to 0.0001. The scaling constant c_2 for the entropy term in (7), c_{uct}, and c_{puct} will be provided later for each configuration.

We train our detector under a varying channel model similar to [18], where an $N_R \times N_T$ complex channel matrix $\mathbf{H}_c$ is first randomly generated with i.i.d. complex Gaussian elements from $\mathcal{CN}(0, 1)$ and then the jth randomly generated symbol vector $\widetilde{\mathbf{x}}_c^j$ is transmitted through a varying channel:

$$\mathbf{H}_c^j = \sqrt{1-\epsilon^2}\mathbf{H}_c + \epsilon \mathbf{W}_c^j, \tag{11}$$

where $\mathbf{H}_c^j$ is the channel matrix corresponding to the jth transmitted symbol vector, $\epsilon \in [0, 1]$ is a constant, and $\mathbf{W}_c^j$ is an $N_R \times N_T$ complex noise matrix sampled from $\mathcal{CN}(0, 1)$ corresponding to the jth transmitted symbol vector. Testing data used for simulation results are generated in the same fashion but separately from the training data set. The SNR is defined as $\mathbb{E}\|\mathbf{H}_c\widetilde{\mathbf{x}}_c\|^2/\mathbb{E}\|\mathbf{w}_c\|^2 = N_T\sigma_x^2/\sigma_w^2$, and the symbol error rate (SER) is calculated to evaluate the performance of the detectors.

We compare the proposed DeepMCTS with the following schemes: 1) MMSE, 2) MCTS, as described in Sec. II-B, 3) DRL, which detects MIMO symbols by choosing the action that corresponds to the highest probability from the policy value network output of the DRL agent, and 4) ML, the optimal detector as defined in (4).

B. Results and Discussion

Fig. 3(a) presents the SER performance of MCTS and DeepMCTS with different playout numbers over a varying 8×8 MIMO channel with QPSK and $\epsilon = 0.08$. We set $c_2 = 1$, $c_{\text{uct}} = 350$, and $c_{\text{puct}} = 22$. As can be seen, the performance of both detectors improves as the playout number increases. DeepMCTS outperforms MCTS by a large margin with significantly reduced playout numbers. Since playout number is one of the dominating factors of computational complexity, this suggests that DeepMCTS demonstrates both performance and complexity advantages as compared to MCTS. The performance comparison of various detectors for the same setting is shown in Fig. 3(b). It can be seen that the proposed DeepMCTS algorithm significantly outperforms either DRL or MCTS alone. Comparing the selection criteria (5) and (10) for MCTS and DeepMCTS, respectively, (10) incorporates the state value network output in evaluating the first term and the policy value network output in the second term to enhance tree exploration. DRL achieves suffered performance because it directly chooses the action that corresponds to the highest probability from the policy value network output. This, when placed in the context of the DeepMCTS algorithm, is equivalent to replacing the selection criterion (10) by $P^{(i)}$, without utilizing the output of the state value network ($\overline{U}^{(i)}$) and with only one playout ($Z^{(i)} = 1$ and $z^{(i)} = 0$). The omission of the state value network output and the lack of exploration due to one playout lead to the degraded performance of DRL.

Fig. 3(c) shows the SER result over a varying 8×16 MIMO channel with QPSK and $\epsilon = 0.08$. We set $c_2 = 1$, $c_{\text{uct}} = 300$, and $c_{\text{puct}} = 25$. The comparative trends of DeepMCTS, DRL, and MCTS are similar to those in the previous scenario. In asymmetric channels here with more receive antennas than transmit antennas, the channel matrix $\mathbf{H}$ tends to be more well-conditioned [1]. This creates a favorable condition for traditional detectors such as the sphere decoder (producing extensive tree pruning and therefore higher detection speed and lower complexity) and MMSE detector (producing generally satisfactory solutions), and is similarly so for the tree search-based DeepMCTS algorithm. Thus, MMSE achieves satisfactory performance here, and DeepMCTS requires only a small number of playouts to achieve good performance.

We evaluate the complexity of all schemes by the average detection runtime per symbol vector. The runtime results for 8×8 QPSK are presented in Table I (top). Comparing DeepMCTS and MCTS, DeepMCTS not only achieves significantly improved performance as discussed earlier, but also reduced complexity. This is due to the smaller number of playouts required for DeepMCTS to reach satisfactory performance. Comparing DeepMCTS and DRL, DRL has a lower complexity without the tree search process, but much worse performance. The linear MMSE detector has a low complexity as expected, at the cost of performance. The complexity of ML quickly grows prohibitively large with the order of modulation and/or the number of antennas, and thus DeepMCTS presents a good performance-complexity tradeoff among all schemes.

The runtime results for 8×16 QPSK are shown in Table I (bottom). Comparing the two configurations in Table I, the complexity of DeepMCTS significantly reduces while that of DRL and MCTS increases in this larger antenna system. The increased complexity for DRL and MCTS results from calculating the reward value of larger matrices for each move [9] and the function operations of higher-dimensional matrices. The reduced complexity for DeepMCTS is due to the reduced number of playouts required.

V. Conclusion

We have proposed the DeepMCTS algorithm for MIMO symbol detection, which incorporates a DRL architecture into the traditional MCTS algorithm. By using DRL, we avoid the labeling dilemma encountered in other supervised learning-based detectors. The proposed scheme innovatively combines DRL and MCTS to enhance MCTS with a learned and guided

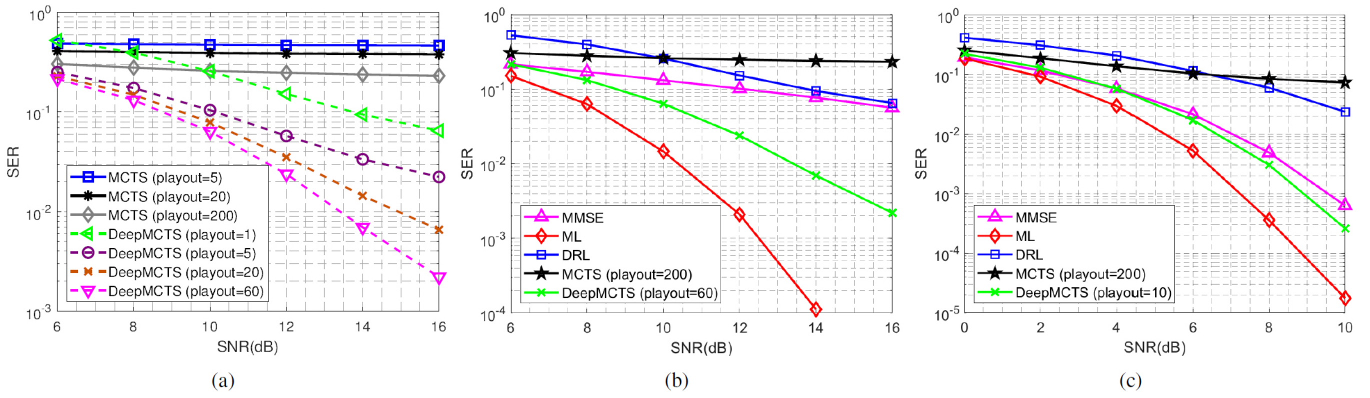

Fig. 3. SER performance comparison for (a)(b) 8×8 MIMO with QPSK and (c) 8×16 MIMO with QPSK.

TABLE I
RUNTIME COMPARISON FOR 8×8 MIMO WITH QPSK (TOP) AND 8×16 MIMO WITH QPSK (BOTTOM)

Runtime (seconds)		**SNR**					
		6	8	10	12	14	16
Algorithm	ML	0.4091	0.4873	0.4981	0.5048	0.4950	0.5045
	MMSE	5.2E−05	5.2E−05	5.2E−05	5.2E−05	5.2E−05	5.2E−05
	DRL	0.0072	0.0067	0.0066	0.0066	0.0066	0.0066
	MCTS (playout = 200)	0.3064	0.3045	0.3037	0.3042	0.3019	0.3018
	DeepMCTS (playout = 60)	0.2640	0.2546	0.2455	0.2410	0.2419	0.2401

Runtime (seconds)		**SNR**					
		0	2	4	6	8	10
Algorithm	ML	0.4734	0.4735	0.4754	0.4779	0.4755	0.4741
	MMSE	5.3E−05	5.2E−05	5.2E−05	5.2E−05	5.3E−05	5.2E−05
	DRL	0.0075	0.0070	0.0069	0.0070	0.0068	0.0067
	MCTS (playout = 200)	0.3807	0.4039	0.3798	0.3913	0.3919	0.3869
	DeepMCTS (playout = 10)	0.0735	0.0631	0.0668	0.0594	0.0581	0.0580

tree search process enabled by DRL. Simulation results demonstrated that DeepMCTS significantly increases the detection accuracy and search efficiency of MCTS. Insightful discussions on the performance and complexity of various schemes were presented.

REFERENCES

[1] S. Yang and L. Hanzo, “Fifty years of MIMO detection: The road to large-scale MIMOs,” *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 1941–1988, Fourth Quarter 2015.
[2] V. A. Marčenko and L. A. Pastur, “Distribution of eigenvalues for some sets of random matrices,” *Math USSR Shornik*, vol. 1, no. 4, pp. 457–483, Nov. 1967.
[3] D. Zhu, B. Li, and P. Liang, “On the matrix inversion approximation based on Neumann series in massive MIMO systems,” *in 2015 IEEE International Conference on Communications (ICC)*, pp. 1763–1769, Jun. 2015.
[4] S. Bittner, E. Zimmermann, and G. Fettweis, “Low complexity soft interference cancellation for MIMO-systems,” *in 2006 IEEE 63rd Vehicular Technology Conference*, pp. 1993–1997, May 2006.
[5] P. Li and R. D. Murch, “Multiple output selection-LAS algorithm in large MIMO systems,” *IEEE Commun. Lett.*, vol. 14, no. 5, pp. 399–401, May 2010.
[6] C. Jeon, R. Ghods, A. Maleki, and C. Studer, “Optimality of large MIMO detection via approximate message passing,” in *2015 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2015, pp. 1227–1231.
[7] Y. de Jong and T. Willink, “Iterative tree search detection for MIMO wireless systems,” *IEEE Trans. Commun.*, vol. 53, no. 6, pp. 930–935, Jun. 2005.
[8] R. Y. Chang and W.-H. Chung, “Best-first tree search with probabilistic node ordering for MIMO detection: Generalization and performance-complexity tradeoff,” *IEEE Trans. Wireless Commun.*, vol. 11, no. 2, pp. 780–789, Feb. 2012.
[9] J. Chen, C. Fei, H. Lu, G. E. Sobelman, and J. Hu, “Hardware efficient massive MIMO detector based on the Monte Carlo tree search method,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 7, no. 4, pp. 523–533, Aug. 2017.
[10] Q. Chen, S. Zhang, S. Xu, and S. Cao, “Efficient MIMO detection with imperfect channel knowledge - a deep learning approach,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2019, pp. 1–6.
[11] N. Samuel, T. Diskin, and A. Wiesel, “Learning to detect,” *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2554–2564, May 2019.
[12] A. Mohammad, C. Masouros, and Y. Andreopoulos, “Complexity-scalable neural-network-based MIMO detection with learnable weight scaling,” *IEEE Trans. Commun*, vol. 68, no. 10, pp. 6101–6113, 2020.
[13] X. Tan, W. Xu, K. Sun, Y. Xu, Y. Be’ery, X. You, and C. Zhang, “Improving massive MIMO message passing detectors with deep neural network,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1267–1280, Dec. 2020.
[14] N. Shlezinger, R. Fu, and Y. C. Eldar, “DeepSIC: Deep soft interference cancellation for multiuser MIMO detection,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1349–1362, Feb. 2021.
[15] H. He, C. Wen, S. Jin, and G. Y. Li, “A model-driven deep learning network for MIMO detection,” in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 584–588.
[16] V. Corlay, J. J. Boutros, P. Ciblat, and L. Brunel, “Multilevel MIMO detection with deep learning,” in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Oct. 2018, pp. 1805–1809.
[17] C. B. Browne, E. Powle, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of Monte Carlo tree search methods,” *IEEE Trans. Comput. Intell. AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[18] Y.-S. Jeon, N. Lee, and H. V. Poor, “Robust data detection for MIMO systems with one-bit ADCs: A reinforcement learning approach,” *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1663–1676, Mar. 2020.