Resource Management for Multifunction Multichannel Cognitive Radars

Mahdi Shaghaghi
Complementary AI Systems Corp.
Toronto, Canada
Email: m.shaghaghi@complement.ai

Raviraj S. Adve
Department of Electrical and Computer Engineering
Toronto, Canada
Email: rsadve@ece.utoronto.ca

Zhen Ding
Defence Research and Development Canada (DRDC)
Ottawa, Canada
Email: Zhen.Ding@drdc-rddc.gc.ca

***Abstract*—A modern radar may be designed to perform multiple functions, such as surveillance, tracking, and communications. A radar resource management (RRM) module makes decisions on parameter selection, prioritization, and scheduling of such tasks. RRM becomes especially challenging in overload situations, where some of the tasks may need to be delayed or even dropped. In general, task scheduling is an NP-hard problem. Finding the optimal solution has high computational complexity, and heuristic methods, while having low complexity, provide relatively poor performance. In this work, we use machine learning-based techniques to address this issue. We develop a cognitive scheduler which is trained through the interactions of the radar with the environment. Specifically, we propose an approximate algorithm based on the Monte Carlo tree search method. A policy network is also used to help to reduce the width of the search. Such a network can be trained using reinforcement learning techniques. Furthermore, we consider the implementation of the above methods with constraints on channels and tasks, e.g., nonhomogeneous channels, blocked channels, periodic tasks, etc. We have modified the MCTS method to make sure that the constraints are met. The efficiency of the proposed methods are shown using simulation results.**

I. Introduction

A cognitive radar acquires knowledge and understanding of its operating environment through estimation, reasoning and learning, and exploits this knowledge and understanding to enhance information extraction, data processing and radar management [1], [2]. A multifunction radar (MFR) is designed to perform various tasks including surveillance, tracking, fire control, and communications. This raises the problem of assigning radar resources, such as the time, frequency and power, to different tasks. Specifically, a radar resource management (RRM) module makes decisions on parameter selection, prioritization, and scheduling of such tasks [3]. In this work, we consider the RRM problem for cognitive radars, where the RRM module is designed to gain knowledge through the operation of the radar over time.

In a multichannel radar, it is possible to execute multiple tasks simultaneously [4]. While this greatly enhances task execution, it also brings up a new complicated problem of scheduling tasks on multiple timelines. RRM, already an NP-hard problem, becomes especially challenging in overload situations, where some of the tasks may need to be delayed or even dropped [3]. In this work, we consider a system that improves and learns from its previous operations.

We have considered RRM for multichannel radars in our previous works, and we have developed heuristic methods as well as the optimal branch-and-bound (B&B) technique [5]. It is shown that heuristic methods in the literature (such as "earliest start time first (EST)") have poor performance, and the B&B algorithm can have high computational complexity. We proposed that the complexity of the B&B algorithm can be reduced by using machine learning (ML) techniques [6]. ML methods eliminate nodes from the search tree without compromising much on the performance. However, the complexity still remains rather high.

In [7], we proposed to further reduce the complexity using the Monte Carlo tree search (MCTS) method [8], [9]. Along with using bound and dominance rules to eliminate nodes from the search tree, we used a policy network to help to reduce the width of the search. The neural network was trained using labeled data obtained by running the B&B method offline on problems with feasible complexity. We showed that the proposed method has near-optimal performance, while its computational complexity is orders of magnitude smaller than the B&B algorithm.

The current work has two aspects: 1) we consider implementation of the above methods with new constraints, and 2) we also introduce new machine learning techniques for training the neural networks. In our previous work, we had considered a number of simplifying assumptions. In practice, however, there can be requirements and constraints which need to be met by the RRM module. For example, we had assumed that all the radar channels are identical and always available. In practice, channels may have different properties, and may not always be available (e.g., consider a spectrum sharing scenario where other users are using the same channels, or the case when one of the channels is out of service for some reason). We had also assumed that the radar tasks are independent from each other and can be executed on any channel. However, there can be constraints on the tasks. For example, a task may need to occupy more than one channel, or a task may be required to be executed before another task. Moreover, the parameters of the tasks can depend on the channel on which they are scheduled. For example, different channels may

978-1-7281-4300-2/19/$31.00 ©2019 IEEE

provide different accuracies, and this needs to be taken into account by the RRM module.

We have developed new efficient methods to account for the constraints on the tasks and channels. Specifically, we have derived new formulations for the decision making stage of the MCTS method, which make sure that the requirements and constraints on the tasks and channels are met. In the new methods, the RRM module investigates variations on scheduling tasks on different channels and also eliminates unfeasible schedules. At every node of the MCTS search tree, we consider possible decisions which meet the constraints and requirements of the problem. For example, we only take into account the channels which are available and not blocked or occupied by other users. In order to deal with nonhomogeneous channels, we define the branches of the search tree in a way that encompasses the selection of the channel as well as the task to be executed on that channel. In this way, we can investigate different possibilities of scheduling tasks on various timelines.

The above design will result in a search tree which has large numbers of branches at its nodes. To deal with this complexity, a policy network is used to guide the search towards more promising branches. Such a network is trained through the data collected form the operation of the radar with a reinforcement learning approach. This results in a cognitive system which learns from its experiences with the environment.

II. Problem Formulation

Consider a multifunction radar with K channels. Each channel is associated with a timeline on which tasks can be scheduled. There are N tasks, indexed as $1, \ldots, N$, which need to be executed to accomplish the radar missions. Tasks can run concurrently on different channels, but cannot overlap on a given timeline. Furthermore, we consider a non-preemptive scheduling scenario, i.e., once a task starts execution, it cannot be stopped until completion. Unlike the assumption made in [7] on the identicalness of the channels, in this work, different timelines may have different characteristics. In other words, tasks may be executed differently on the channels. Furthermore, some of the timelines may not be available for some of the tasks, and if available, they may provide different levels of accuracy for performing the tasks. However, independent of the channels, each task has a *start time* $r_n, n = 1, \ldots, N$ after which the task is ready to be executed. This parameter is the same for all the timelines. Moreover, each task has a *completion time deadline* $x_n, n = 1, \ldots, N$ by which the execution of the task needs to be completed. If it is not possible to complete the task by this deadline on any channel, the task is dropped with an associated *dropping cost* D_n.

As an illustrative example, consider a tracking task. The start time of the task is determined by the required tracking accuracy and the time when the last measurement was made. The completion time deadline depends on the beamwidth of the radar and the estimated trajectory of the target. We do not perform the task after the target is assumed to have moved out of the radar beam, and the task is therefore dropped with its associated dropping cost. Consequently, further actions may be required to compensate for the dropping. The dropping cost also acts as a proxy for task priority.

Depending on the selected timeline, the dwell time of the task may be different. We denote the length of the n-th task on the k-th channel with $\ell_{nk}, n = 1, \ldots, N$ and $k = 1, \ldots, K$. The last moment for the n-th task to begin execution on the k-th channel is denoted by the *end time* d_{nk}.

A scheduled, but delayed, task suffers a tardiness cost which is, here, modeled as linearly proportional to the delay; let c_n be the time when task n completes execution; the tardiness cost is given by $\max\{0, w_n(c_n - s_n)\}$, where w_n is the weight which scales the delay, and s_n is the *desired completion time*. The completion time depends on the channel on which the task is scheduled and is equal to the *execution time* e_n plus the task length on that timeline, ℓ_{nk}, i.e., $c_n = e_n + \ell_{nk}$.

Let the binary variable x_{nk} indicate if task n is scheduled ($x_{nk} = 1$) on the k-th timeline or not ($x_{nk} = 0$). Task n is dropped if $x_{nk} = 1\ \forall k$. Then, the cost associated with the n-th task is given by $\sum_{k=1}^{K} x_{nk}\ \max\{0, w_n(e_n + \ell_{nk} - s_n)\} + (1 - \sum_{k=1}^{K} x_{nk}) D_n$. Our joint task selection and scheduling problem is a minimization of the total cost C, given by

$$C = \sum_{n=1}^{N} \sum_{k=1}^{K} x_{nk} \max\{0, w_n(e_n + \ell_{nk} - s_n)\} + \left(\tfrac{1}{K} - x_{nk}\right) D_n. \tag{1}$$

Here, the optimization variables are x_{nk} and e_n. If a task is scheduled, i.e., $x_{nk} = 1$ for some k, the execution time e_n needs to be determined as well. Furthermore, let B_n be the set of channels which are blocked for the n-th task. In other words, task n cannot be scheduled on the channels included in B_n, i.e., $x_{nk} = 0$ if $k \in B_n$. Therefore, our optimization problem is

$$\begin{aligned} \{x_{nk}^*, e_n^*\} = \underset{x_{nk}, e_n}{\arg\min} & \sum_{n=1}^{N} \sum_{k=1}^{K} x_{nk} \max\{0, \\ & w_n(e_n + \ell_{nk} - s_n)\} + \left(\frac{1}{K} - x_{nk}\right) D_n \\ \text{s.t. } & x_{nk} \in \{0, 1\} \\ & x_{nk} = 0 \ \text{ if } \ k \in B_n \\ & \sum_{k=1}^{K} x_{nk} \leq 1 \\ & r_n \leq e_n \leq d_{nk} \\ & n = 1, \ldots, N \\ & k = 1, \ldots, K \\ & \text{and no tasks overlap in time.} \end{aligned} \tag{2}$$

The first constraint indicates that the optimization variable x_{nk} is binary. The second constraint means task n cannot be scheduled on the blocked channels. The third constraint, $\sum_{k=1}^{K} x_{nk} \leq 1$, ensures that, if scheduled, the task is only assigned to one timeline.

Due to the binary variables, the scheduling problem without deadlines (and therefore without dropped tasks) is NP-hard [10]. It can be shown that the joint task selection and scheduling as given in (2) is also NP-hard.

III. Monte Carlo Tree Search with Policy Networks

Our proposed method is based on the Monte Carlo tree search (MCTS) technique, which provides near-optimal solutions with reduced complexity. The MCTS method focuses the search on the (probabilistically) more promising nodes of a search tree. Each node is associated with a subset of the solutions and represents a state of the system. Each branch is considered as an action (decision) which causes a transition to a new state.

The root node of the tree represents the whole solution space while the remaining nodes are associated with a subset of the solution space. The branching operation splits the space of the parent node into the subspaces of the resulting children nodes. Each subspace represents a partial solution. In the context of the problem in (2), a partial solution is a schedule which includes a subset of the tasks. In this method, each node represents sequences of subsets of tasks which are scheduled on the timelines of the radar.

In our proposed method, an action a is defined to be the scheduling of a given task n on a given timeline k, i.e., $a = (n, k)$. Therefore, each action is composed of both selecting a task and also the channel on which it is going to be scheduled. Using this approach, the actions (branches) which are not feasible according to the constraints in (2) are removed from the search tree. Furthermore, variations on scheduling tasks on different channels are investigated.

At a given node s, a *policy* is defined as the probability distribution over taking possible actions from s. The optimal policy at node s is the distribution that, with probability one, chooses the branch which leads to the best solution. In practice, an approximate policy is used to determine the probability of taking action a at node s, $P(s, a)$. We use a neural network to obtain these probabilities.

The MCTS method has four steps: selection, expansion, backup, and decision making. Starting from a base node, a number of simulations are performed to analyze possible branches. This includes the first three steps of selection, expansion, and backup. These simulations create information and insight about taking different branches. After the analysis is complete, a decision is made which results in the transition to another state. The new node becomes the base for the next round and the rest of the search tree is discarded. This procedure is continued until reaching a complete solution (terminal node).

For a given edge (s, a) representing branch a from node s, the following statistics are held: a prior policy $P(s, a)$ which is the initial probability of taking action a, a visit count $N(s, a)$ which is the number of times branch a has been selected during the previous simulations, and the action value $Q(s, a)$ which is an *estimate of the expected utility* obtained by taking action a. In our proposed method, $Q(s, a)$ is initialized to zero, and is updated based on the solutions which we have obtained so far by taking action a at state s. Particularly, we set $Q(s, a)$ to be the value of the best solution found so far by taking edge (s, a). The value of a solution is obtained from the total cost of the scheduling, C, as $1-C/C_{EST}$, where C_{EST} is the cost of the solution obtained using the EST method.

In the selection step, the action which maximizes $Q(s, a) + U(s, a)$, where

$$U(s, a) = P(s, a)\frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{3}$$

is taken. The first term in this selection rule, i.e., $Q(s, a)$, encourages exploitation by favoring branches with higher action values while the second term, i.e., $U(s, a)$, encourages exploration by favoring branches with lower visit counts. The algorithm keeps selecting actions until it reaches a terminal node.

The backup step is done by updating the statistics of the branches which have been visited during the current simulation. The visit count of these edges is incremented by one. Furthermore, the action value of each edge is updated based on the value of the complete solution at the terminal node. After the backup step, the next simulation is performed. When all the simulations are done, a decision is made, which causes the transition to the next base node. We choose the branch which has the highest action value. Then, the next round of simulations is performed at the new base node. This procedure is repeated until reaching a complete solution, i.e., a terminal node.

In the expansion step of the MCTS method, the prior policy needs to be initialized. At a given node s, we use a *policy network* to obtain $P(s, a)$. Particularly, a *neural network* $\boldsymbol{p} = f_\theta(s)$ is used such that its input is the state s, and its output is the estimate of the policy for node s. The parameters θ of such a network are trained using reinforcement learning [9].

We begin the training by initializing the network parameters θ randomly. For an instance of the problem, the MCTS algorithm is performed with the current policy network to find a solution. The statistics which have been acquired during the search are now used to train the network parameters. For a node s on the path to the final solution, define the vector $\boldsymbol{\pi}$ such that its elements are proportional to the visit counts of the branches of s, i.e., $N(s, a)$. Vector $\boldsymbol{\pi}$ is used as the target policy for node s. Then, the parameters θ are adjusted by minimizing the following cross-entropy loss

$$\ell(\theta) = -\boldsymbol{\pi}^T \log \boldsymbol{p} \tag{4}$$

where $\boldsymbol{p} = f_\theta(s)$. For the next sample problem, the network with the new parameters is used. Then, the acquired data is employed to update θ. This procedure is repeated to improve the performance of the network.

A. Policy Network Architecture

As mentioned in the previous section, in the expansion step of the MCTS algorithm, we use the policy network to provide

the prior probability distribution over the possible actions in the given node, s. The input to the network is the state of the node, and the output is the prior policy, i.e., $\boldsymbol{p} = f_\theta(s)$, where θ denotes the weights of the network.

The input to the network is treated as an image with height 1, width N_p, and N_c channels (features). Each column of the input image is associated with a task, and for each task, we have the N_c features as given in Table I. Here, $N_c = 5 + 4K$ where K is the number of timelines (See Table I for a list of the features for $K = 4$.). At a given node, the active input tasks are those which are yet to be scheduled. The features, g_i $(1 \leq k \leq K)$ are indicators of the time after which the k-th channel is available at the given state s.

the overall number of active tasks may be smaller, equal, or greater than N_p. However, the number of input tasks to the network is fixed to N_p. To deal with this issue, the first feature is used to indicate whether a given input task is active or not. The rest of the features are derived from the parameters of the tasks.

The output of the network is a vector of length N_pK which represents a probability distribution for the input actions to the network.

We implement the policy network with 7 layers. The first four layers are convolutional and the last three layers are fully connected. Each convolutional layer is composed of the following operations. First, the input is convolved with a filter. Next, we perform batch normalization [11]. Then, the result goes through a non-linear rectifier function (ReLU). In the training stage, we also perform the dropout [12] operation (with a 50 percent keep probability) before passing the result to the next layer.

At all the convolutional layers the kernel size is 1×7, i.e., the filter has a height of 1 and width of 7. Therefore, we are looking at the features of 7 consecutive tasks at each stride. The first filter has an input depth of N_c (which is equal to 21 in our implementation) and an output depth of 96. The rest of the filters have the same input and output depth of 96. We use valid padding for the convolution operations, i.e., the filters do not go past the edges of the input, as there is no zero padding. Furthermore, the convolutions are performed with a stride of 1.

The output of the fourth layer is vectorized and then passed to a fully connected layer with 2048 hidden units. The second fully connected layer has 1024 hidden units, and the output of the last layer has a length of N_pK. The first two fully connected layers have the following components. First, the input is multiplied by a weight matrix (note that there is no bias term at this stage). Next, we perform batch normalization. Then, the result goes through the ReLU activation function. In the training phase, we also perform the dropout operation before passing the result to the next layer. The last fully connected layer simply is a linear transformation with a weight matrix and a bias term.

Finally, the output of the last layer goes through the softmax function to produce the output of the network.

IV. Performance Evaluation

We consider a multichannel radar with $K = 4$ channels. There are different numbers of tasks distributed over a timeline window of 100 ms. The objective is to schedule the tasks such that the overall cost of dropping and delaying the tasks is minimized (see (2)).

The start time of the tasks is uniformly distributed on the time window, i.e., $r_n \sim \mathcal{U}(0, 100)$, where $x \sim \mathcal{U}(a, b)$ represents a random variable uniformly distributed between a and b. For each task, the interval between the start time and the deadline, i.e., $d_n - r_n$, is sampled from $\mathcal{U}(2, 12)$. The task length, ℓ_n, is distributed according to $\mathcal{U}(2, 11)$. Furthermore, the dropping costs, D_n, and the tardiness weights, w_n, have distributions $\mathcal{U}(100, 500)$ and $\mathcal{U}(1, 5)$, respectively. We have added the following constraints to the problems. First, regarding timeline availability, every forth task cannot be scheduled on timeline 1. Second, regarding timeline accuracy, timeline 1 has higher accuracy in a way that executing a task on this timeline takes 20 percent less time.

We perform the simulations for 1000 sample problems to obtain the average performance of the considered methods. For each simulation scenario, the same set of parameters is used for all the algorithms.

We investigate the performance of the proposed method for different number of tasks, N. The number of Monte Carlo rollouts is set to either $MC = 10$ or $MC = 100$. The average cost of various methods versus the number of tasks is illustrated in Fig. 1. Heuristic policy refers to the MCTS method when the prior probabilities are set according to a heuristic probability distribution. In our simulations, action probabilities are set proportional to $1/n^2$ on the possible timeline which is available first. For the rest of timelines, action probabilities are set proportional to $1/Kn^2$.

As can be seen in Fig. 1, the proposed MCTS method has a better performance than the EST and the MCTS method with the heuristic policy.

Next, we compare the above methods with respect to their ability of scheduling all the tasks without dropping any of them. The probability that no task is dropped versus the number of tasks is depicted in Fig. 2. As it can be seen, the proposed method achieves the highest probability of not dropping any of the tasks.

V. Conclusion

In this paper, we consider the problem of radar resource management for a multichannel multifunction radar. We introduce how machine learning techniques can be used to acquire knowledge during the operation of the cognitive radar in a way that such knowledge can be used in future decision making. We propose a method based on the Monte Carlo tree search which uses a policy network to focus the search on more promising branches.

For the future work, one can consider resource management for a multisensor system. We have developed methods for a multichannel radar. These solutions may be extended to multisensor systems, where each sensor is modeled as a

Table I: Features of the tasks as used for the input of the policy network

Feature	Description
1	status (0: inactive input, 1: active task)
2	start time, r_n
3	tardiness weight, w_n
4	dropping cost, D_n
5	desired completion time, s_n
$6, 7, 8, 9$	end time for each timeline, $d_{n1}, d_{n2}, d_{n3}, d_{n4}$
$10, 11, 12, 13$	task length per timeline, $\ell_{n1}, \ell_{n2}, \ell_{n3}, \ell_{n4}$
$14, 15, 16, 17$	possible timelines to execute the task (0: unavailable, 1: available)
$18, 19, 20, 21$	earliest availability time for each timeline, g_1, g_2, g_3, g_4

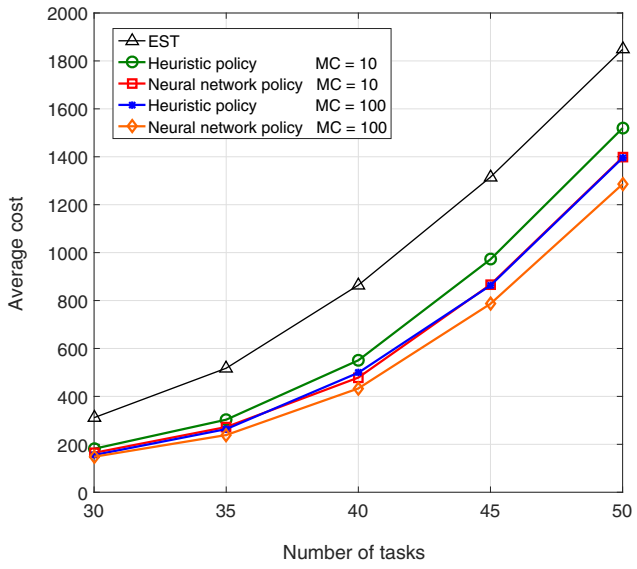

Fig. 1: Average cost versus the number of tasks, N.

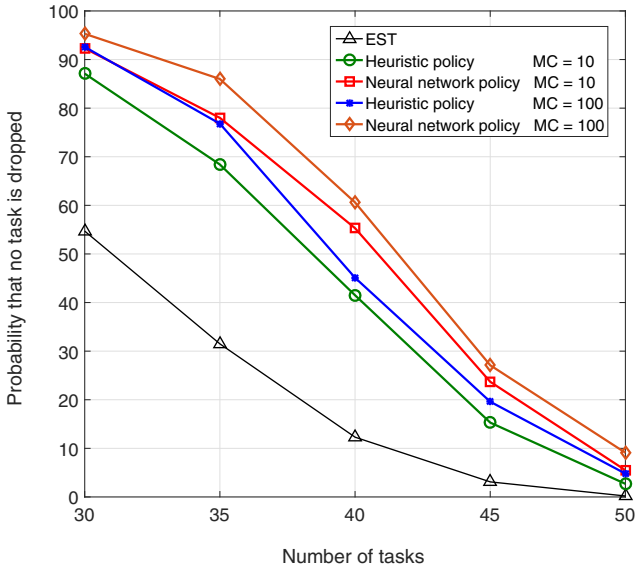

Fig. 2: Probability that all the tasks are scheduled without any dropping versus the number of tasks, N.

channel. It will, however, need a central control unit which has information of the states of the sensors.

ACKNOWLEDGMENT

This work is supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Defence Research and Development Canada (DRDC).

REFERENCES

[1] A. Charlish and F. Hoffmann, "Cognitive Radar Management," in *Novel Radar Techniques and Applications: Waveform Diversity and Cognitive Radar, and Target Tracking and Data Fusion*. IET, 2017, pp. 157–194.

[2] S. Haykin, "Cognitive radar: a way of the future," *IEEE Sig. Process. Mag.*, vol. 23, no. 1, pp. 30–40, Jan. 2006.

[3] P. W. Moo and Z. Ding, *Adaptive Radar Resource Management*. Academic Press, 2015.

[4] J. Yan, H. Liu, B. Jiu, B. Chen, Z. Liu, and Z. Bao, "Simultaneous multibeam resource allocation scheme for multiple target tracking," *IEEE Trans. on Sig. Proc.*, vol. 63, no. 12, pp. 3110–3122, June 2015.

[5] M. Shaghaghi and R. S. Adve, "Task selection and scheduling in multifunction multichannel radars," in *2017 IEEE Radar Conference (RadarConf)*, May 2017, pp. 969–974.

[6] ——, "Machine learning based cognitive radar resource management," in *2018 IEEE Radar Conference (RadarConf)*, April 2018.

[7] M. Shaghaghi, R. S. Adve, and Z. Ding, "Multifunction cognitive radar task scheduling using Monte Carlo tree search and policy networks," *IET Radar, Sonar & Navigation*, vol. 12, no. 12, pp. 1437–1447, Sept. 2018.

[8] D. Silver, A. Huang, C. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[9] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354 – 359, Oct. 2017.

[10] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, vol. 1, pp. 343–362, 1977.

[11] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37. Lille, France: PMLR, Jul. 2015, pp. 448–456.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.