Reinforcement Learning-Based Bit-Flipping Decoding Strategy for Channel Coding

EECE 571E - Error Correction Codes - Project Report

Idan Roth [redacted] Flynn Dowey [redacted] Cee Rikhh [redacted]

***Abstract*—This project report delves into the innovative application of Reinforcement Learning (RL) techniques for decoding error-correcting codes, with a focus on binary linear codes and a bit-flipping (BF) decoding strategy. Utilizing the SARSA and Q-learning algorithms within a Markov Decision Process (MDP) framework, the study extends the conventional approach to decoding by iteratively making BF decisions based on individual bits, thereby transforming the decoding process into a series of RL problems. The exploration includes a comprehensive background on channel coding, detailing the transition of received signals over an additive white Gaussian noise channel to a binary symmetric channel, and explicates the BF decoding algorithm as well as the foundational elements of Markov Decision Processes. A paramount challenge addressed in this work is the prohibitive memory complexity associated with tabular RL algorithms. To mitigate this, a solution involving the integration of a low-complexity, parameterized approximator for the Q-function is discussed, utilizing a Neural Network (NN). This approach not only offers a scalable alternative to tabular methods but also leverages the advantages of machine learning to enhance decoding performance. Empirical results from simulations illustrate the efficacy of the tabular SARSA decoder, which surpasses both traditional and BF decoders in terms of Bit Error Rate (BER), with higher computational efficiency. While the parameterized SARSA decoder does not outperform its tabular counterpart, it nonetheless presents a viable option for decoding large codes, where tabular methods are impractical. The comparison between SARSA and Q-learning highlights the situational advantages of each, suggesting avenues for future research to optimize decoder performance across different channel models and coding schemes. This investigation underscores the potential of RL techniques in the realm of error correction coding, particularly in optimizing decoding strategies for binary linear codes.**

I. Introduction

The process of decoding error-correcting codes can be conceptualized as a classification problem and addressed through the application of supervised machine learning techniques. Fundamentally, this approach entails considering the decoder as a parameterized function (e.g., a neural network), and employing data-driven optimization methods to ascertain optimal parameter configurations [1]–[7]. In the absence of additional constraints on the code, this method demonstrates efficacy primarily for short codes, and becomes ineffective for unstructured codes comprising more than a few hundred codewords. In the context of linear codes, the complexity of the problem is substantially reduced, as it necessitates learning only a singular decision region rather than individual regions for each codeword. The code linearity can be exploited via message-passing techniques [4] or by using syndromes [1], [7]. Nevertheless, the decoding challenge persists, as good codes often exhibit complicated decision regions, a consequence of the extensive number of neighboring codewords. Despite these challenges, near-optimal performance of decoders, trained through learning methodologies in practical settings, has been evidenced in various types of codes. Moreover, machine learning-based decoders may overcome the computational complex implementation of conventional decoders.

In this project we aim to study and reproduce the results of [8], which proposed a reinforcement learning (RL) approach for the aforementioned problem. The focus is on the decoding of binary linear codes from a machine-learning perspective. Yet, rather then learning the direct mapping from observations (corrupted codewords) to estimated codewords in a supervised fashion, the decoding is performed in an iterative way based on individual bit-flipping (BF) decisions. This approach facilitates the formulation of the problem within the framework of a Markov Decision Process (MDP), enabling the application of RL techniques to find effective decision-making strategies. The approach used is a syndrome-based approach where the state space of the MDP includes all possible binary syndromes.

II. Background

A. Channel Coding

Let $\mathcal{C}$ be a (n, k) binary linear code defined by a parity-check (PC) matrix $H \in \mathbb{F}_2^{m \times n}$, where n is the code length, k is the code dimension, and $m \geq n - k$. The code is used to encode messages $x \in \mathbb{F}_2^k$ into codewords $c = (c_1, c_2, ..., c_n)^T \in \mathbb{F}_2^n$, which are then transmitted over the additive white Gaussian noise (AWGN) channel according to:

$$y_i = (-1)^{c_i} + w_i \quad \forall i \in [n] \tag{1}$$

where y_i is the i-th element in the received vector (corrupted codeword) $y = (y_1, y_2, ..., y_n)^T$, $w_i \sim \mathcal{N}\left(0, (2RE_b/N_0)^{-1}\right)$, the code rate is $R = k/n$, and E_b/N_0 denotes the signal-to-noise-ratio (SNR). The received vector is then mapped by a hard-decision rule to obtain $z = (z_1, z_2, ..., z_n)^T$. I.e., z_i is obtained by mapping the sign of y_i according to $+1 \to 0$, $-1 \to 1$. A decoding that is based on the hard-decisions z is equivalent to the transmission over the binary symmetric channel (BSC) [9], where every bit is flipped with

probability p. The crossover probability p of a BSC_p in that case is proportional to the AWGN variance, since larger noise variance correspond to a larger bit flipping probability.

BF decoding is an iterative decoding algorithm, that has been studied extensively in literature, which involves a decision-making process. The fundamental concept underlying BF decoding involves the development of an appropriate metric that enables the decoder to systematically evaluate and rank the bits in terms of their reliability, taking into consideration the constraints imposed by the code. In its most basic version, BF employs the hard-decision output, denoted as z and proceeds iteratively to identify the bit which, upon being flipped, would result in the maximum reduction of the currently violated PC equations. BF is a syndrome decoding-based algorithm which takes advantage of the property of the parity check matrix that for every codeword $c \in \mathcal{C} : Hc = 0$ [9], and we denote $s = Hz \in \mathbb{F}_2^m$ as the observed syndrome. Pseudocode for a generic BF decoding is provided in Alg. 1, where $e_j \in \mathbb{F}_2^n$ is a basis vector whose j-th element is 1 and the rest are 0.

Algorithm 1 Bit-Flipping Decoding

Input: hard decision z, parity-check matrix H
Output: estimated codeword $\hat{c}$

$\hat{c} \leftarrow z$
while $H\hat{c} \neq 0$ **and** max. iterations not exceeded **do**
  $V \leftarrow \sum_{i=1}^{m} s_i$ where $s = H\hat{c}$
  for $j = 1, 2, ..., n$ **do**
    $Q_j \leftarrow V - \sum_{i=1}^{m} s_i$, where $s = H\left(\hat{c} + e_j\right)$
  end for
  update $\hat{c} \leftarrow \hat{c} + e_j$, where $j = \arg\max_{j \in [n]} Q_j$
end while

B. Markov Decision Processes

A time-invariant MDP is characterized by a Markovian random process $S_0, S_1, ...$ in which the probability of transitioning between states $P(s'|s, a) \triangleq P(S_{t+1} = s'|S_t = s, A_t = a)$ is influenced only by the action A_t taken by an agent currently being in state S_t at time step t. Here, $s, s' \in \mathcal{S}, a \in \mathcal{A}$ where $\mathcal{S}, \mathcal{A}$ are finite sets containing all possible states and actions. Moreover, the agent receives a reward $R_t = R(S_t, A_t, S_{t+1})$ which depends only on the previous state, updated state, and action taken for this transition. The agent's decision making process is described by a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps states into actions in either a probabilistic fashion or deterministic one. The goal is to find the optimal policy π^* that returns the best action for each possible state in terms of the total expected discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$, where $0 < \gamma < 1$. Should the probabilities associated with transitions and rewards are known, dynamic programming techniques are applicable for the derivation of optimal policies. Conversely, in scenarios where such probabilities remain undefined, optimal policies can yet be identified via repeated interactions with the environment, on the condition that the states and rewards are observable. This approach is referred to as RL.

Two of the simplest forms of RL are known as Q-learning [10] and state-action-reward-state-action (SARSA) [11], wherein the optimal policy is defined through the Q-function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ according to

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a), \tag{2}$$

where the optimal Q-function $Q^*(s, a)$ is defined by the Bellman Optimality equation as

$$Q^*(s, a) = R(s, a, s') + \gamma \sum_{s'} P(s'|s, a) \max_{a' \in \mathcal{A}} Q^*(s', a') \tag{3}$$

and the Q-function can be recursively expressed by the Bellman equation as

$$Q_\pi(s, a) = R(s, a, s') + \gamma \sum_{s'} P(s'|s, a) \mathop{\mathbb{E}}_{a' \sim \pi(\cdot|s')} Q_\pi(s', a') \tag{4}$$

Both Q-learning and SARSA are a part of a family of RL algorithms called TD methods. The goal is to iteratively learn the optimal Q-function $Q^*(s, a)$ which is based on the optimal policiy π^*. In Q-learning this is done by

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{5}$$

SARSA is closely related by updating the Q function

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma Q(s', a') - Q(s, a) \right] \tag{6}$$

Remark: SARSA is closely related to the Bellman equation and Q-learning is based off of the Bellman optimality equation.

Pseudocode for Q-learning and SARSA are given in Alg. 2 and Alg. 3, where a popular choice for updating the policy is:

$$\pi(a|s) = \begin{cases} \text{unif. random over} \mathcal{A} & \text{w.p. } \epsilon \\ \arg\max_a Q(s, a) & \text{w.p. } 1 - \epsilon \end{cases} \tag{7}$$

which is known as ϵ-greedy exploration.

Algorithm 2 Q-learning

Input:learning rate α, discount factor γ
Output: estimated Q function

initialize $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
for i=1,2,..., max_iters **do**
  initialize starting state s
  while s is not terminal **do**
    choose action a ▷ using ϵ-greedy (7)
    execute a, observe next state s' and reward r
    $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) +$
      $+ \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'))$
    $s \leftarrow s'$
  end while
end for

Algorithm 3 SARSA

Input: learning rate α, discount factor γ
Output: estimated Q function

initialize $Q(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
for i=1,2,..., max_iters **do**
  initialize starting state s
  choose action a ▷ using ϵ-greedy (7)
  while s is not terminal **do**
    execute a, observe next state s' and reward r
    choose action a' ▷ using ϵ-greedy (7)
    $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma Q(s',a')\right)$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for

III. Markov Decision Process Modeling

In this section we present the mapping of bit-flipping decoding into a an MDP so that Q-learning or SARSA could be conducted [8].

1) Choosing Action and State Spaces: We will define the action A_t as the flipping of a single bit in the received corrupted codeword at time t. The codeword block length is n, therefore, there are n possible bits to flip and the action space can be represented as $\mathcal{A} = \{1, 2, ..., n\} = [n]$. Since bit-flipping decoding is a syndrome-based algorithm, the state space $\mathcal{S}$ would be formed by all possible binary syndromes of length m. Assuming the initial state S_0 is the syndrome Hz, the next state can be derived by simply adding the A_t-th column of H to the current state $S_{t+1} = H(z + e_{A_t})$. The all-zero syndrome corresponds to the terminal state, i.e., a valid codeword has been decoded. This construction make the MDP deterministic, where the transition probabilities $P(s'|s,a)$ take the values in $0, 1$. A limitation on the number of bit-flips per codeword is also enforced such that at most T bit-flips could performed before the current iteration is ended and a new received codeword will be decoded.
2) Choosing the Reward Strategy: A straightforward reward function for the decoding purpose is when a return value of 1 is assigned for accurately decoded codewords, and 0 for inaccuracies. This criterion suggests that an optimal policy minimizes the codeword error rate. Nonetheless, it is mandated that the reward function's dependencies are confined to the current state, next state and the executed action. Based on the maximum-likelihood (ML) decoding optimization problem for a binary linear code over general discrete memoryless channels discussed in [8], the following reward function was established for the BSC:

$$R(s,a,s') = \begin{cases} -\frac{1}{T} + 1 & \text{if } s' = 0 \quad \text{(terminal state)} \\ -\frac{1}{T} & \text{otherwise.} \end{cases} \tag{8}$$

This reward function enables us to view optimal BF decoding as a "maze-playing game" within the syndrome domain, with the objective of discovering the shortest path to reach the all-zero syndrome. The incorporation of a small negative penalty for each move is a conventional method aimed at promoting the discovery of shorter paths.
3) Choosing the Exploration Strategy: As previously stated, we will use the ϵ-greedy exploration from Eq. (7).

IV. Parameterized Q-function Approximators

Standard SARSA and Q-learning require the storing of all Q-function real values for each state and action (refer to step 1. in Algo. 3 and 2). Accordingly, a table of size $|\mathcal{S}| \times |\mathcal{A}|$ must be stored in memory, which would become infeasible for code with large block length. For example, standard tabular Q-learning would be feasible for binary $RM(r = 32, m = 16)$ code which has $|\mathcal{S}| = n - k = 2^m - \sum_{j=0}^{r} \binom{m}{j} = 2^{16}$ and $|\mathcal{A}| = 32$, the table has $|\mathcal{S}||\mathcal{A}| \approx 2 \cdot 10^{16}$ entries.

To overcome this issue, a fitted Q-learning algorithm is introduced in [8] where the idea is to learn a low complexity approximation of $Q(s,a)$ which would be represented as a parameterized function $Q_\theta(s,a)$ where θ are the learnable parameters. In this approach, we would alternate between simulating the MDP and updating/training the current parameters to obtain the estimate of the Q-function. In particular for SARSA, given a tuple (s,a,r,s',a') we would update the parameters θ based on minimizing the loss

$$\mathcal{L}(\theta) = r + \gamma Q_\theta(s',a') - Q_\theta(s,a). \tag{9}$$

using gradient descent. Pseudocode for fitted SARSA is provided in Alg. 4.

Algorithm 4 Fitted SARSA

Input: learning rate α, discount factor γ
Output: estimated Q function

initialize parameters θ
for i=1,2,..., max_iters **do**
  initialize starting state s
  choose action a ▷ using ϵ-greedy (7)
  while s is not terminal **do**
    execute a, observe next state s' and reward r
    choose action a' ▷ using ϵ-greedy (7)
    $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for

A common choice of $Q_\theta(s,a)$ is as a neural network (NN) where θ represents the weights. Note that the loss function is based on how we update the Q-function in Alg. 3, where the minimization will approximate $Q_\theta(s,a)$. We use a fully connected NN with one hidden layer to represent $Q_\theta(s,a)$. The NN f_θ maps a syndrome s into a length-n vectors $f_\theta(s,a) \in \mathbb{R}^n$, which represent the Q-function value

approximation at state s for every possible action $\hat{Q}(s,a)$ where $a \in \mathcal{A}$. The exact value would be chosen according to the randomly generated action a. f_θ structure was chosen as $(n-k)\times k \times n$ which has a decoder architecture. Moreover, ReLU was used as the activation function of the hidden layer and the last layer is linear.

V. Simulation Results

In this section, numerical results are presented for learned BF decoder alongside the conventional decoders which were used as benchmarks. We considered the following binary codes:

- $RM(r=3, m=6)$
- $BCH(n=63, d=45)$

The code that was used in the project is available at https://github.com/FlynnDowey/RL_decoding/tree/main.

A. Experimental Setup

We set the maximum number of decoding iterations to $T = 10$, which correspond to allowing a maximum of 10 bit flips. The number of episodes used in the tabular setting was 9×10^5. Furthermore, the discount factor was set to $\gamma = 0.95$ and epsilon greedy was initialized at $\epsilon = 0.9$ where ϵ followed an exponential decay during learning. Tabular SARSA and Q-learning were trained with a learning rate between $\alpha \in [0.01, 1]$, however performance was consistent amongst all values. The learning rate in parameterized SARSA was set to $\alpha = 0.1$, and followed an exponential decay. Weights were updated according to stochastic gradient descent (SGD). The parameterized setting used the same ϵ-greedy values as the tabular case. For each episode, the environment would randomly select a message to encoder and subsequently transmit over the channel. This is a more general approach, as [8] only considered a constant message throughout inference and training. The agents were trained on a constant SNR value of 4dB and inference was conducted with a range of SNR values between 1dB to 7dB.

B. Performance Evaluation

In Fig. 1 and 2 the decoding performance of $RM(3,6)$ and $BCH(63,45)$ code is presented. The figures illustrate the bit error rate (BER) performance metric over different values of SNR for four different decoders: traditional decoder, BF decoder, tabular SARSA decoder, and parameterized SARSA decoder. For BCH code, the traditional decoder followed the Berlekamp-Massey decoding algorithm, while RM code was decoded using Reed's majority-logic algorithm. Tabular SARSA outperform both the traditional decoder and the BF decoder. Moreover, it does it in a more efficient way, which is crucial for real time applications. Although the parameterized SARSA does not perform as well as the tabular version, it showcases the ability to generalize on codes with long block length, where the tabular version would become computationally infeasible. Furthermore, it outperform the traditional decoder, and the BF decoder in most of the SNR values, and more importantly, in the low SNR regime.

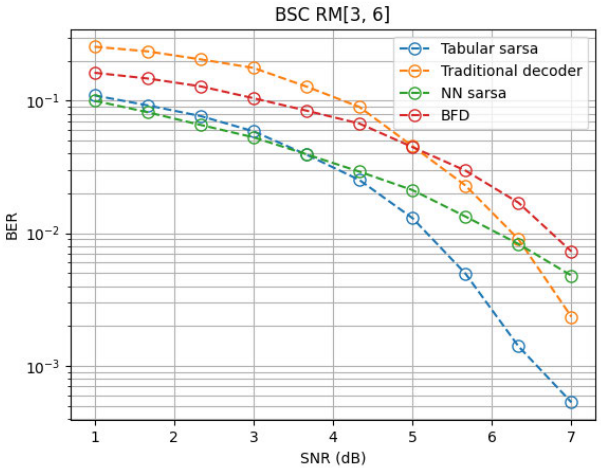

Fig. 1. Simulation results for learned BF decoding for $RM(6,3)$ code. BER over several SNR values.

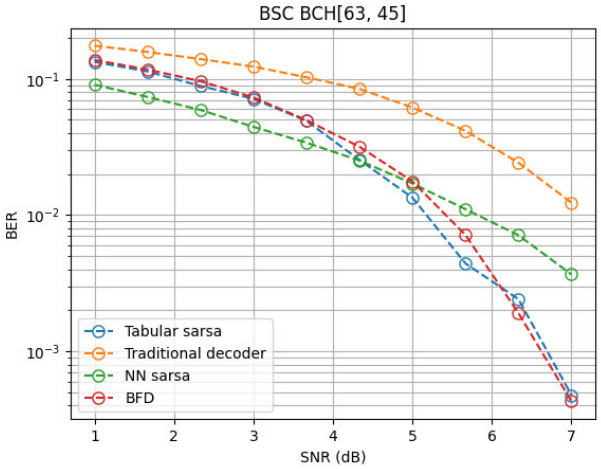

Fig. 2. Simulation results for learned BF decoding for $BCH(63,45)$ code. BER over several SNR values.

Next, we present a performance comparison between SARSA and Q-learning. Fig 3 and 4 depict the BER performance of tabular SARSA, tabular Q-learning, traditional decoding, and BF decoding for the two codes we considered. We can see that tabular SARSA outperforms tabular Q-learning for both codes under the BSC for the presented MDP formulation setting. If Q-learning was trained on more episodes, then its performance would asymptotically converge with SARSA. The performance difference between Q-learning and SARSA is clearly explained in [11].

Remark: we explored the performance of Q-learning on channels that are not BSC, where the environment uses other types of reward functions. We noticed that it can outperform SARSA in various settings. We suggest that the understanding of when Q-learning can outperform SARSA, or under which setting to use either algorithm would be left for future research.

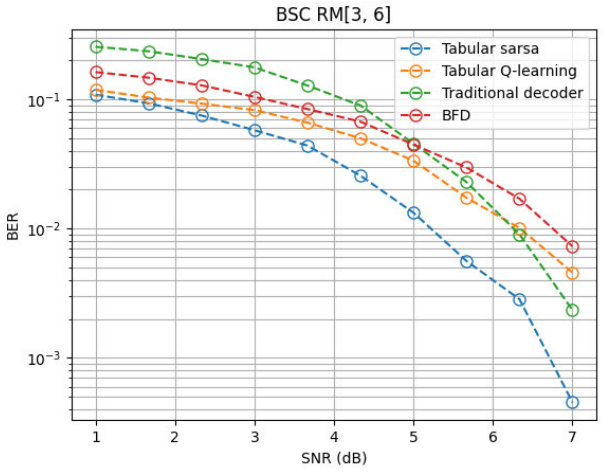

Fig. 3. Comparison results between learned BF decoding based on SARSA or Q-learning for $RM(6,3)$ code. BER over several SNR values.

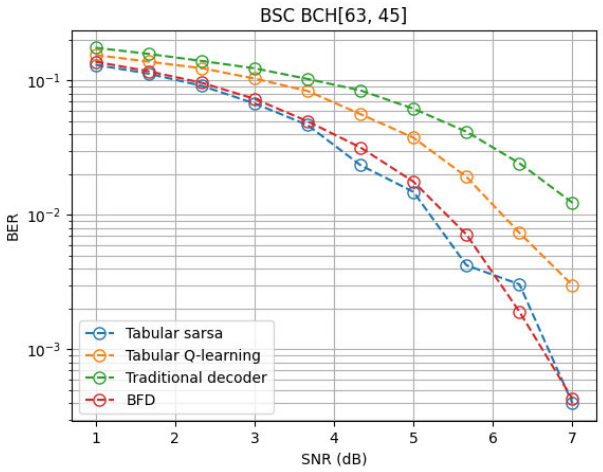

Fig. 4. Comparison results between learned BF decoding based on SARSA or Q-learning for $BCH(63,45)$ code. BER over several SNR values.

VI. Conclusion

In this report, an exploration of a RL decoder, based on bit-flipping decoding, utilizing both SARSA and Q-learning methodologies, was undertaken. The foundational concepts of coding theory alongside RL were presented, followed by the formulation of the RL-centric approach. Addressing the challenge posed by the memory complexity inherent in tabular RL algorithms, we discussed a solution that involves the incorporation of a parameterized, low-complexity approximator for the Q-function values. This approximator is realized through the implementation of a fully-connected NN. Preliminary results demonstrated that the tabular SARSA algorithm surpassed both conventional and BF decoders in terms of BER, and is computational efficient. Moreover, while the parameterized SARSA variant exhibited inferior performance compared to its tabular analog, it still achieved commendable results, underscoring the feasibility of applying RL techniques to large codes. Lastly, a comparison between SARSA and Q-learning has been introduced. Future research directions include extending this RL approach to non-binary codes, various channel models, and further refining the balance between computational efficiency and decoding accuracy, broadening the applicability of machine learning methodologies in communication systems.

References

[1] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes-a syndrome-based approach," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1595–1599, IEEE, 2018.

[2] T. Gruber, S. Cammerer, J. Hoydis, and S. Ten Brink, "On deep learning-based channel decoding," in *2017 51st annual conference on information sciences and systems (CISS)*, pp. 1–6, IEEE, 2017.

[3] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," *arXiv preprint arXiv:1805.09317*, 2018.

[4] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, IEEE, 2016.

[5] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[6] R. Raheli, H. D. Pfister, C. Häger, M. Martalò, and F. Carpi, "Exploring machine learning algorithms for decoding linear block codes," 2018.

[7] L. Tallini and P. Cull, "Neural nets for decoding error-correcting codes," in *IEEE Technical applications conference and workshops. Northcon/95. Conference record*, p. 89, IEEE, 1995.

[8] F. Carpi, C. Häger, M. Martalò, R. Raheli, and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 922–929, IEEE, 2019.

[9] V. Guruswami, A. Rudra, and M. Sudan, "Essential coding theory," *Draft available at http://www. cse. buffalo. edu/atri/courses/coding-theory/book*, vol. 2, no. 1, 2012.

[10] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.