2019 57th Annual Allerton Conference on Communication, Control, and Computing
(Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, September 24-27, 2019

# Reinforcement Learning for Channel Coding: Learned Bit-Flipping Decoding

Fabrizio Carpi[1], Christian Häger[2], Marco Martalò[3], Riccardo Raheli[3], and Henry D. Pfister[4]

***Abstract*— In this paper, we use reinforcement learning to find effective decoding strategies for binary linear codes. We start by reviewing several iterative decoding algorithms that involve a decision-making process at each step, including bit-flipping (BF) decoding, residual belief propagation, and anchor decoding. We then illustrate how such algorithms can be mapped to Markov decision processes allowing for data-driven learning of optimal decision strategies, rather than basing decisions on heuristics or intuition. As a case study, we consider BF decoding for both the binary symmetric and additive white Gaussian noise channel. Our results show that learned BF decoders can offer a range of performance–complexity trade-offs for the considered Reed–Muller and BCH codes, and achieve near-optimal performance in some cases. We also demonstrate learning convergence speed-ups when biasing the learning process towards correct decoding decisions, as opposed to relying only on random explorations and past knowledge.**

## I. Introduction

The decoding of error-correcting codes can be cast as a classification problem and solved using supervised machine learning. The general idea is to regard the decoder as a parameterized function (e.g., a neural network) and learn good parameter configurations with data-driven optimization [2]–[7]. Without further restrictions on the code, this only works well for short codes and typically becomes ineffective for unstructured codes with more than a few hundred codewords. For linear codes, the problem simplifies considerably because one has to learn only a single decision region instead of one region per codeword. One can take advantage of linearity by using message-passing [4] or syndromes [5], [6]. Still, the problem remains challenging because good codes typically have complicated decision regions due to the large number of neighboring codewords. Near-optimal performance of learned decoders in practical regimes has been demonstrated, e.g., for convolutional codes [7], which possess even more structure.

This work was done while F. Carpi was a student at University of Parma and was visiting Duke University. Preliminary results appeared in the thesis [1]. The work of C. Häger was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant No. 749798. The work of H. D. Pfister was supported in part by the National Science Foundation (NSF) under Grant No. 1718494. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors. Please send correspondence to henry.pfister@duke.edu.

[1]Department of Electrical and Computer Engineering, New York University, Brooklyn, New York, USA

[2]Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

[3]Department of Engineering and Architecture, University of Parma, Parma, Italy

[4]Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA

In this paper, we study the decoding of binary linear block codes from a machine-learning perspective. Rather than learning a direct mapping from observations to estimated codewords (or bits) in a supervised fashion, the decoding is done in steps based on individual bit-flipping (BF) decisions. This allows us to map the problem to a Markov decision process (MDP) and apply reinforcement learning (RL) to find good decision strategies. Following [5], [6], our approach is syndrome-based and the state space of the MDP is formed by all possible binary syndromes, where bit-wise reliability information can be included for general memoryless channels. This effectively decouples the decoding problem from the transmitted codeword.

BF decoding has been studied extensively in the literature and is covered in many textbooks on modern coding theory, see, e.g., [8]–[13], [14, Ch. 10.7]. Despite its ubiquitous use, and to the best of our knowledge, the learning approach to BF decoding presented in this paper is novel. In fact, with the exception of the recent work in [15], we were unable to find references that discuss RL for channel coding. Thus, we briefly review some other iterative decoding algorithms, based on sequential decision-making steps, for which RL is applicable. For a comprehensive survey of RL in the general context of communications, see [16].

## II. Channel Coding Background

Let $\mathcal{C}$ be an $(N, K)$ binary linear code defined by an $M \times N$ parity-check (PC) matrix $\mathbf{H}$, where $N$ is the code length, $K$ is the code dimension, and $M \geq N - K$. The code is used to encode messages into codewords $\boldsymbol{c} = (c_1, ..., c_N)^\mathsf{T}$, which are then transmitted over the additive white Gaussian noise (AWGN) channel according to $y_n = (-1)^{c_n} + w_n$, where $y_n$ is the $n$-th component in the received vector $\boldsymbol{y} = (y_1, ..., y_N)^\mathsf{T}$, $w_n \sim \mathcal{N}(0, (2RE_\mathrm{b}/N_0)^{-1})$, $R \triangleq K/N$ is the code rate, and we refer to $E_\mathrm{b}/N_0$ as the signal-to-noise ratio (SNR). The vector of hard-decisions is denoted by $\boldsymbol{z} = (z_1, ..., z_N)^\mathsf{T}$, i.e., $z_n$ is obtained by mapping the sign of $y_n$ according to $+1 \to 0$, $-1 \to 1$. If the decoding is based only on the hard-decisions $\boldsymbol{z}$, this scenario is equivalent to transmission over the binary symmetric channel (BSC).

### A. Decision Making in Iterative Decoding Algorithms

In the following, we briefly review several iterative decoding algorithms that involve a decision-making process at each step.

*1) Bit-Flipping Decoding:* The general idea behind BF decoding is to construct a suitable metric that allows the decoder to rank the bits based on their reliability given the

978-1-7281-3151-1/19/$31.00 ©2019 IEEE

code constraints [14, Ch. 10.7]. In its simplest form, BF uses the hard-decision output $\boldsymbol{z}$ and iteratively looks for the bit that, after flipping it, would maximally reduce the number of currently violated PC equations. Pseudocode for standard BF decoding is provided in Alg. 1, where $\boldsymbol{e}_n \in \mathbb{F}_2^N$ is a standard basis vector whose $n$-th component is 1 and all other components are 0, $\mathbb{F}_2 \triangleq \{0,1\}$ and $[N] \triangleq \{1,2,\dots,N\}$. BF can be extended to general memoryless channels by including weights and thresholds to decide which bits to flip at each step. This is referred to as weighted BF (WBF) decoding, see, e.g., [8]–[13], [14, Ch. 10.8] and references therein.

*2) Residual Belief Propagation:* Belief propagation (BP) is an iterative algorithm where messages are passed along the edges of the Tanner graph representation of the code. In general, it is known that sequential message-passing schedules can lead to faster convergence than standard flooding schedules where multiple messages are updated in parallel. Residual BP (RBP) [17] is a particular instance of a sequential updating approach without a predetermined schedule. Instead, the message order is decided dynamically, where the decisions are based on the residual—defined as the norm of the difference between the current message and the message in the previous iteration. The residual is a measure of importance or "expected progress" associated with sending the message. In the context of decoding, various extensions of this idea have been investigated under the name of informed dynamic scheduling [18].

*3) Anchor Decoding:* Consider the iterative decoding of product codes[1] over the BSC, where the component codes are iteratively decoded in some fixed order. For this algorithm, undetected errors in the component codes, so-called miscorrections, significantly affect the performance by introducing additional errors into the iterative decoding process. To address this problem, anchor decoding (AD) was recently proposed in [19]. The AD algorithm exploits conflicts due to miscorrections where two component codes disagree on the value of a bit. After each component decoding, a decision is made based on the number of conflicts whether the decoding outcome is indeed reliable. This can lead to backtracking previous component decoding outcomes and to the designation of reliable component codes as anchors.

### B. Decision Making Through Data-Driven Learning

While the above decoding algorithms appear in seemingly different contexts, the sequential decision-making strategies in the underlying iterative processes are quite similar. Decisions are typically made in a greedy fashion based on some heuristic metric that assesses the quality of each possible action. As concrete examples for this metric, we have

- the decrease in the number of violated PC equations in BF decoding, measuring the reliability of bits;
- the residual in RBP, measuring expected progress and the importance of sending messages;
- the number of conflicts in AD, measuring the likelihood of being miscorrected.

[1]Given a linear code $\mathcal{C}$ of length $n$, the product code of $\mathcal{C}$ is the set of all $n \times n$ arrays such that each row and column is a codeword in $\mathcal{C}$.

**Algorithm 1:** Bit-Flipping Decoding

**Input:** hard decisions $\boldsymbol{z}$, parity-check matrix $\mathbf{H}$
**Output:** estimated codeword $\hat{\boldsymbol{c}}$

1 $\hat{\boldsymbol{c}} \leftarrow \boldsymbol{z}$
2 **while** $\mathbf{H}\hat{\boldsymbol{c}} \neq \mathbf{0}$ **and** max. iterations not exceeded **do**
3 $\quad V \leftarrow \sum_{m=1}^{M} s_m$, where $\boldsymbol{s} = \mathbf{H}\hat{\boldsymbol{c}}$ // no. unsat checks
4 $\quad$ **for** $n = 1, 2, \dots, N$ **do**
5 $\quad\quad Q_n \leftarrow V - \sum_{m=1}^{M} s_m$, where $\boldsymbol{s} = \mathbf{H}(\hat{\boldsymbol{c}} + \boldsymbol{e}_n)$
6 $\quad$ update $\hat{\boldsymbol{c}} \leftarrow \hat{\boldsymbol{c}} + \boldsymbol{e}_n$, where $n = \arg\max_{n \in [N]} Q_n$

In the next section, we review MDPs which provide a mathematical framework for modeling decision-making in deterministic or random environments. MDPs can be used to obtain optimal decision-making strategies, effectively replacing heuristics with data-driven learning of optimal metrics.

## III. Markov Decision Processes

A time-invariant MDP is a Markov random process $S_0, S_1, \dots$ whose state transition probability $P(s'|s,a) \triangleq \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$ is affected by the action $A_t$ taken by an agent based only on knowledge of past events. Here, $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets containing all possible states and actions. The agent also receives a reward $R_t = R(S_t, A_t, S_{t+1})$ which depends only on the states $S_t$, $S_{t+1}$ and the action $A_t$. The agent's decision-making process is formally described by a policy $\pi : \mathcal{S} \to \mathcal{A}$, mapping observed states to actions. The goal is to find an optimal policy $\pi^*$ that returns the best action for each possible state in terms of the total expected discounted reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$, where $0 < \gamma < 1$ is the discount factor for future rewards.

If the transition and reward probabilities are known, dynamic programming can be used to compute optimal policies. If this is not the case, optimal policies can still be discovered through repeated interactions with the environment, assuming that the states and rewards are observable. This is known as RL. In the following, we describe two RL algorithms which will be used in the next sections.

### A. Q-learning

The most straightforward instance of RL is called Q-learning [20], where the optimal policy is defined in terms of the Q-function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ according to

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s,a). \tag{1}$$

The Q-function measures the quality of actions and is formally defined as the expected discounted future reward when being in state $s$, taking action $a$, and then acting optimally. The key advantage of the Q-function is that it can be iteratively estimated from observations of any "sufficiently-random" agent. Pseudocode for Q-learning is given in Alg. 2, where a popular choice for generating the actions in line 5 is

$$a = \begin{cases} \text{unif. random over } \mathcal{A} & \text{w.p. } \varepsilon \\ \arg\max_a Q(s,a) & \text{w.p. } 1-\varepsilon. \end{cases} \tag{2}$$

**Algorithm 2:** Q-learning

**Input:** learning rate $\alpha$, discount factor $\gamma$
**Output:** estimated Q-function
1 initialize $Q(s,a) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
2 **for** $i = 1, 2, \ldots$ **do**
3  initialize starting state $s$ // restart the MDP
4  **while** $s$ is not terminal **do**
5  choose action $a$ // $\varepsilon$-greedy (2) or $(\varepsilon, \varepsilon_g)$-goal (14)
6  execute $a$, observe reward $r$ and next state $s'$
7  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a'))$
8  $s \leftarrow s'$

This is referred to as $\varepsilon$-greedy exploration. For any $0 < \varepsilon < 1$, this strategy is sufficient to allow Q-learning to eventually explore the entire state/action space. In the next section, we also describe an alternative exploration strategy for our application that can converge faster than $\varepsilon$-greedy exploration.

To motivate the update equation in line 7 of Alg. 2, we note that the Q-function can be recursively expressed as

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

This expression forms the theoretical basis for Q-learning which converges to the true Q-function under certain conditions[2]. For a more details, we refer the reader to [20], [21].

### B. Fitted Q-learning with Function Approximators

For standard Q-learning, one must store a table of $|\mathcal{S}| \times |\mathcal{A}|$ real values. This will be infeasible if either set is prohibitively large. The idea of fitted Q-learning is to learn a low-complexity approximation of $Q(s,a)$ [21]. Let $Q_\theta(s,a)$ be an approximation of the Q-function, parameterized by $\theta$. Fitted Q-learning alternates between simulating the MDP and updating the current parameters to obtain a better estimate of the Q-function. In particular, assume that we have simulated and stored $B$ transition tuples $(s,a,r,s')$ in a set $\mathcal{D}$. Then, updating the parameters $\theta$ is based on reducing the empirical loss

$$\mathcal{L}_\mathcal{D}(\theta) = \sum_{(s,a,r,s') \in \mathcal{D}} \left( r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s',a') - Q_\theta(s,a) \right)^2. \tag{4}$$

Pseudocode for fitted Q-learning is provided in Alg. 3, where gradient descent is used to update the parameters $\theta$ based on the loss (4). It is now common to choose $Q_\theta(s,a)$ to be a (deep) neural network (NN), in which case $\theta$ are the network weights and fitted Q-learning is called deep Q-learning.

## IV. Case Study: Bit-Flipping Decoding

In this section, we describe how BF decoding can be mapped to an MDP. In general, this mapping involves multiple design choices that affect the results. We therefore also comment on alternative choices and highlight some potential pitfalls that we encountered during this process.

[2]For example, if $R(s,a,s')$ depends non-trivially on $s'$, then $\alpha$ must decay to zero at sufficiently slow rate.

**Algorithm 3:** Fitted Q-learning

**Input:** learning rate $\alpha$, batch size $B$
**Output:** parameterized estimate of the Q-function
1 initialize parameters $\theta$ and $\mathcal{D} \leftarrow \emptyset$
2 **for** $i = 1, 2, \ldots$ **do**
3  initialize starting state $s$ // restart the MDP
4  **while** $s$ is not terminal **do**
5  choose action $a$ // $\varepsilon$-greedy (2) or $(\varepsilon, \varepsilon_g)$-goal (14)
6  execute $a$, observe reward $r$ and next state $s'$
7  store transition $(s,a,r,s')$ in $\mathcal{D}$
8  $s \leftarrow s'$
9  **if** $|\mathcal{D}| = B$ **then**
10  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\mathcal{D}(\theta)$ // see (4) for def. of $\mathcal{L}_\mathcal{D}$
11  empty $\mathcal{D}$

### A. Theoretical Background

We start by reviewing the standard maximum-likelihood (ML) decoding problem for a binary linear code $\mathcal{C} \subseteq \mathbb{F}_2^N$ over general discrete memoryless channels. The resulting optimization problem forms the basis for the reward function that is used in the MDP. To that end, consider a collection of $N$ discrete memoryless channels described by conditional probability density functions $\{P_{Y_n|C_n}(y_n|c_n)\}_{n \in [N]}$, where $c_n \in \mathbb{F}_2$ is the $n$-th code bit and $y_n$ is the $n$-th channel observation. The ML decoding problem can be written as

$$\arg\max_{\boldsymbol{c} \in \mathcal{C}} \prod_{n=1}^N P_{Y_n|C_n}(y_n|c_n) = \arg\max_{\boldsymbol{c} \in \mathcal{C}} \sum_{n=1}^N (-1)^{c_n} \lambda_n, \tag{5}$$

where

$$\lambda_n \triangleq \ln \frac{P_{Y_n|C_n}(y_n|0)}{P_{Y_n|C_n}(y_n|1)} \tag{6}$$

is the channel log-likelihood ratio (LLR). Equivalently, one can rewrite the maximization over all possible codewords in terms of error patterns as

$$\arg\max_{\boldsymbol{e}\,:\,\boldsymbol{z}+\boldsymbol{e} \in \mathcal{C}} \sum_{n=1}^N (-1)^{z_n} (-1)^{e_n} \lambda_n \tag{7}$$

$$= \arg\max_{\boldsymbol{e}\,:\,\boldsymbol{z}+\boldsymbol{e} \in \mathcal{C}} \sum_{n=1}^N (-1)^{e_n} |\lambda_n| \tag{8}$$

$$= \arg\max_{\boldsymbol{e}\,:\,\mathbf{H}\boldsymbol{e}=\boldsymbol{s}} \sum_{n=1}^N (-1)^{e_n} |\lambda_n| \tag{9}$$

$$= \arg\max_{\boldsymbol{e}\,:\,\mathbf{H}\boldsymbol{e}=\boldsymbol{s}} \sum_{n=1}^N -e_n |\lambda_n| \tag{10}$$

where $\boldsymbol{s} = \mathbf{H}\boldsymbol{z}$ is the observed syndrome.

Now, consider a multi-stage process where bit $a_t$ is flipped during the $t$-th stage until the syndrome of the bit-flip pattern matches $\boldsymbol{s}$. In this case, the optimization becomes

$$\arg\max_{\tau, a_1, \ldots, a_\tau \,:\, \sum_{t=1}^\tau \boldsymbol{h}_{a_t} = \boldsymbol{s}} \sum_{t=1}^\tau -|\lambda_{a_t}|, \tag{11}$$

where $\boldsymbol{h}_n$ is the $n$-th column of the parity-check matrix $\mathbf{H}$. By interpreting $-|\lambda_{a_t}|$ as a reward, one can see that the objective function in (11) has the same form as the cumulative reward (without discount) in an MDP. The following points are worth mentioning:

- For the BSC, all LLRs have the same magnitude and (11) returns the shortest flip pattern that matches the observed syndrome.
- For general channels, (11) returns the shortest *weighted* flip pattern that matches the syndrome, where the weighting is done according to the channel LLRs. In other words, the incurred penality for flipping bit $a_t$ is directly proportional to the reliability of the corresponding received bit.
- If a bit is flipped multiple times, then there must be a shorter bit-flip sequence with lower cost and the same syndrome. Therefore, it is sufficient to only consider flip patterns that contain distinct bits.

### B. Modeling the Markov Decision Process

*1) Choosing Action and State Spaces:* We assume that the action $A_t$ encodes which bit is flipped in the received word at time $t$. Since there are $N$ possible choices, we simply use $\mathcal{A} = \{1, 2, \ldots, N\} \triangleq [N]$. The state space $\mathcal{S}$ is formed by all possible binary syndromes of length $M$. The initial state $S_0$ is the syndrome $\mathbf{H}\boldsymbol{z}$ and the next state is formed by adding the $A_t$-th column of $\mathbf{H}$ to the current state. The transition probabilities $P(s'|s, a)$ therefore take values in $\{0, 1\}$, i.e., the MDP is deterministic. The all-zero syndrome corresponds to a terminal state. We also enforce a limit of at most $T$ bit-flips per codeword. After this, we exit the current iteration and a new codeword will be decoded.[3]

*Remark* 1. For the BSC, we also tried (unsuccessfully) to learn BF decoding with fitted Q-learning directly from the channel observations using the state space $\mathbb{F}_2^N$.

*Remark* 2. For the AWGN channel, the state space can be extended by including the reliability vector $\boldsymbol{r} = |\boldsymbol{y}|$, similar to the setup in [6]. In this case, each state would correspond to a tuple $(\boldsymbol{s}, \boldsymbol{r})$, where $\boldsymbol{s} \in \mathbb{F}_2^M$ and $\boldsymbol{r}$ remains constant during decoding. In this paper, we follow a different strategy for BF decoding over the AWGN channel which relies on permuting the bit positions based on their reliability and subsequently discarding the channel LLRs prior to decoding. This approach is described in Sec. V and does not require any modifications to the state space.

*2) Choosing the Reward Strategy:* A natural reward function for decoding is to return 1 if the codeword is decoded correctly and 0 otherwise. This would imply that an optimal policy minimizes the codeword error rate. However, the reward is only allowed to depend on the current/next state and the action, whereas the transmitted codeword and its estimate are defined outside the context of the MDP. Based on (11) and the discussion in the previous subsection, we instead use the reward function

$$R(s, a, s') = \begin{cases} -c|\lambda_a| + 1 & \text{if } s' = \mathbf{0} \\ -c|\lambda_a| & \text{otherwise}, \end{cases} \tag{12}$$

where $c > 0$ is a scaling factor. The additional reward for matching the syndrome is required to prevent the decoder from just flipping the bits where $|\lambda_a|$ is minimal. For example, it could happen that a single error in position $a$ with large $|\lambda_a|$ matches the syndrome, but instead one chooses to flip $T$ bits with small absolute LLRs. The scaling factor $c$ is chosen such that the syndrome-matching reward $+1$ always dominates the expected cummulative term $-\sum_{t=1}^{T} c|\lambda_{a_t}|$. As an example, for the BSC, $c$ is chosen such that the reward function becomes

$$R(s, a, s') = \begin{cases} -\frac{1}{T} + 1 & \text{if } s' = \mathbf{0} \\ -\frac{1}{T} & \text{otherwise.} \end{cases} \tag{13}$$

This reward function allows us to interpret optimal BF decoding as a "maze-playing game" in the syndrome domain where the goal is to find the shortest path to the all-zero syndrome. Applying a small negative penalty for each step is a standard technique to encourage short paths. Another alternative in this case is to choose a small discount factor $\gamma < 1$.

*3) Choosing the Exploration Strategy:* Compared to (2), we propose another exploration strategy as follows. Let $\boldsymbol{e}$ be the current error pattern, i.e., the channel error pattern plus any bit-flips that have been applied so far. Then, with probability $\varepsilon_{\mathrm{g}}$, we choose the action randomly from $\mathrm{supp}(\boldsymbol{e}) \triangleq \{i \in [N] \,|\, e_i = 1\}$, i.e., we flip one of the incorrect bits. When combined with $\varepsilon$-greedy exploration, we refer to this as $(\varepsilon, \varepsilon_{\mathrm{g}})$-goal exploration, where $\varepsilon, \varepsilon_{\mathrm{g}} > 0$ and $0 < \varepsilon + \varepsilon_{\mathrm{g}} < 1$:

$$a = \begin{cases} \text{unif. random over } \mathcal{A} & \text{w.p. } \varepsilon \\ \text{unif. random over } \mathrm{supp}(\boldsymbol{e}) & \text{w.p. } \varepsilon_{\mathrm{g}} \\ \arg\max_a Q(s, a) & \text{w.p. } 1 - \varepsilon - \varepsilon_{\mathrm{g}}. \end{cases} \tag{14}$$

*Remark* 3. It may seem that biasing actions towards flipping erroneous bits leads to a form of supervised learning where the learned decisions merely imitate ground-truth decisions. To see that this is not exactly true, consider transmission over the BSC where the error pattern has weight $d_{\min} - 1$ (where $d_{\min}$ is the minimum distance of the code) and the observation is at distance 1 from a codeword $\tilde{\boldsymbol{c}}$. Then, the optimal decision is to flip the bit that leads to $\tilde{\boldsymbol{c}}$, whereas flipping an erroneous bit is suboptimal in terms of expected future reward, even though it moves us closer to the transmitted codeword $\boldsymbol{c} \neq \tilde{\boldsymbol{c}}$.

*4) Choosing the Function Approximator:* We use fully-connected NNs with one hidden layer to represent $Q_\theta(s, a)$ in fitted Q-learning. In particular, the NN $\boldsymbol{f}_\theta$ maps syndromes to length-$N$ vectors $\boldsymbol{f}_\theta(\boldsymbol{s}) \in \mathbb{R}^N$ and the Q-function is given by $Q_\theta(s, a) = [\boldsymbol{f}_\theta(\boldsymbol{s})]_a$, where $[\cdot]_n$ returns the $n$-th component of a vector and $\boldsymbol{s}$ is the syndrome for state $s$. The NN parameters are summarized in Tab. I. In future work, we

[3] Strictly speaking, the resulting process is not an MDP unless the time $t$ is included in the state space.

TABLE I: Neural network parameters

| layer | input | hidden | output |
|---|---|---|---|
| number of neurons | $M$ | 500 / 1500 | $N$ |
| activation function | - | ReLU | linear |

plan to explore other network architectures, e.g., multi-layer NNs or graph NNs based on the code's Tanner graph.

## V. Learned Bit-Flipping with Code Automorphisms

Let $\mathcal{S}_N$ be the symmetric group on $N$ elements so that $\pi \in \mathcal{S}_N$ is a bijective mapping (or permutation) from $[N]$ to itself.[4] The permutation automorphism group of a code $\mathcal{C}$ is defined as $\mathrm{PAut}(\mathcal{C}) \triangleq \{\pi \in \mathcal{S}_N \,|\, \boldsymbol{x}^\pi \in \mathcal{C}, \forall \boldsymbol{x} \in \mathcal{C}\}$, where $\boldsymbol{x}^\pi$ denotes a permuted vector, i.e., $x_i^\pi = x_{\pi(i)}$. The permutation automorphism group can be exploited in various ways to improve the performance of practical decoding algorithms, see, e.g., [22], [23]. In the context of learned decoders, the authors in [6] propose to permute the bit positions prior to decoding (and unpermute after) such that the channel reliabilities are approximately sorted. If the applied permutations are from $\mathrm{PAut}(\mathcal{C})$, the decoder simply decodes a permuted codeword, rather than the transmitted one. The advantage is that certain bit positions are now more reliable than others due to the (approximate) sorting. This can be advantageous in terms of optimizing parameterized decoders because of the additional structure that the decoder can rely on [6].

### A. A Permutation Strategy for Reed–Muller Codes

In [6], the permutation preprocessing approach is applied for Bose–Chaudhuri–Hocquenghem (BCH) codes and permutations are selected from $\mathrm{PAut}(\mathcal{C})$ such that the total reliabilities of the first $K$ permuted bit positions are maximized, see [6, App. II] for details. In the following, we propose a variation of this idea for RM codes. In particular, our goal is to find a permutation that sends as many as possible of the *least reliable* bits to positions $\{0, 1, 2, 4, \ldots, 2^{m-1}\} \triangleq \mathcal{B}$. Recall that the automorphism group of $\mathrm{RM}(r, m)$ is the general affine group of order $m$ over the binary field, denoted by $\mathrm{AGL}(m, 2)$ [24, Th. 24]. The group $\mathrm{AGL}(m, 2)$ is the set of all operators of the form

$$T(\boldsymbol{v}) = \mathbf{A}\boldsymbol{v} + \boldsymbol{b}, \tag{15}$$

where $\mathbf{A} \in \mathbb{F}_2^{m \times m}$ is an invertible binary matrix and $\boldsymbol{b}, \boldsymbol{v} \in \mathbb{F}_2^m$. By interpreting the vector $\boldsymbol{v}$ as the binary representation of a bit position index, (15) defines a permutation on the index set $\{0, 1, \ldots, N-1\}$ and thus on $[N]$.

A set of vectors $\{\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ is called *affinely independent* if and only if the set $\{\boldsymbol{v}_1 - \boldsymbol{v}_0, \ldots, \boldsymbol{v}_m - \boldsymbol{v}_0\}$ is linearly independent. The binary representations of the indices in $\mathcal{B}$ correspond to the all-zero vector and all unit vectors of length $m$. One can verify that they are affinely independent. The proposed strategy relies on the fact that, for any given set of $m+1$ affinely independent bit positions (in the sense that their binary representation vectors are affinely independent), there always exists a permutation in $\mathrm{AGL}(m, 2)$ such that the bit positions are mapped to $\mathcal{B}$ in any desired order. In particular, we perform the following steps to select the permutation prior to decoding:

1) Let $\pi$ be the permutation that sorts the reliability vector $\boldsymbol{r} = |\boldsymbol{y}|$, i.e., $\boldsymbol{r}^\pi$ satisfies $r_i^\pi < r_j^\pi \iff i < j$.
2) Find the first $m+1$ affinely independent indices for $\pi$ (e.g., using Gaussian elimination) and denote their binary representations by $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$.
3) The permutation is then defined by (15), where $\boldsymbol{b} = \boldsymbol{v}_0$ and the columns of $\mathbf{A}$ are $\boldsymbol{v}_1 - \boldsymbol{v}_0, \ldots, \boldsymbol{v}_m - \boldsymbol{v}_0$.

[4] For a group $(G, \circ)$, we also informally refer to the set $G$ as the group. In our context, the group operation $\circ$ represents function composition defined by $(\pi \circ \sigma)(i) = \pi(\sigma(i))$.

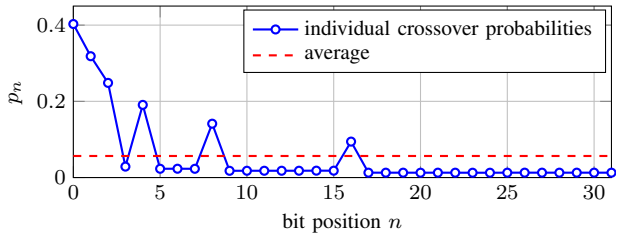



Fig. 1: BSC crossover probabilities after the proposed permutation strategy for $\mathrm{RM}(32, 16)$ at $E_\mathrm{b}/N_0 = 4\,\mathrm{dB}$.

### B. (Approximate) Sort and Discard

For the learned BF decoders over the AWGN channel, our approach is to first apply the permutation strategy described in the previous section and subsequently discard the channel LLRs. From the perspective of the decoder, this scenario can be modeled as $N$ parallel BSCs, where the crossover probabilities for the bit positions in $\mathcal{B}$ satisfy $p_0 > p_1 > p_2 > p_4 > \cdots > p_{2^{m-1}}$. This is related to approaches where channel reliabilities are used to mark highly reliable and/or unreliable bit positions, while the actual decoding is performed without knowledge of the reliability values using hard-decision decoding, see, e.g., [25].

The absolute values of the channel LLRs for the parallel BSCs used in the reward function (12) are given by

$$|\lambda_n| = \log \frac{1 - p_n}{p_n}, \tag{16}$$

where $p_n$ is the crossover probability of the $n$-th BSC. The individual crossover probabilities can be determined via Monte Carlo estimation before the RL starts. For example, Fig. 1 show the expected crossover probabilities after applying the proposed permutation strategy for $\mathrm{RM}(32, 16)$ assuming transmission at $E_\mathrm{b}/N_0 = 4\,\mathrm{dB}$.

*Remark* 4. One can estimate the capacity of strategies that permute the received bits using the reliabilities and then discard them. Fig. 2 shows the estimated information rates for the proposed strategy obtained via Monte Carlo averaging. Our results show that a significant fraction of the achievable information rate is preserved, especially for high-rate codes. For permutations restricted to $\mathrm{AGL}(m, 2)$, this is less effective as the blocklength increases because the fraction of sorted channels satisfies $(m+1)/N = (\log_2(N) + 1)/N$.

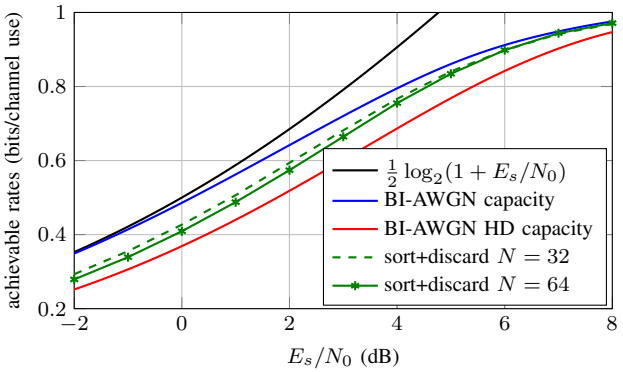



Fig. 2: Estimation of achievable information rates when applying the proposed permutation strategy for RM codes and subsequently discarding the reliability values. (BI-AWGN: binary-input AWGN, HD: hard decision)

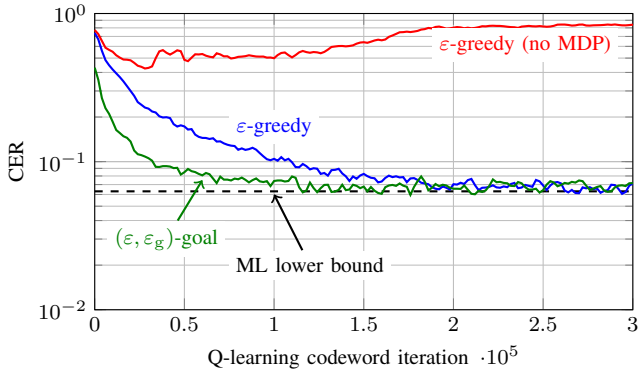



Fig. 3: Q-learning convergence for RM$(32,16)$ on the BSC (crossover prob. 0.0565 corresponding to $E_b/N_0 = 4$ dB) assuming $T = 10$, $\alpha = 0.1$, $\gamma = 1.0$, and $\varepsilon = 0.9$ for $\varepsilon$-greedy and $\varepsilon = 0.6$, $\varepsilon_g = 0.3$ for $(\varepsilon,\varepsilon_g)$-goal.

## VI. Results

In this section, numerical results are presented for learned BF (LBF) decoders[5] for the following RM and BCH codes:

- RM$(32,16)$ with the standard $16\times 32$ PC matrix $\mathbf{H}_{\text{std}}$ and overcomplete $620\times 32$ PC matrix $\mathbf{H}_{\text{oc}}$ whose rows are all minimum-weight dual codewords, see [8], [26]
- RM$(64,42)$ with the standard $22\times 64$ PC matrix $\mathbf{H}_{\text{std}}$ and overcomplete $2604\times 64$ PC matrix $\mathbf{H}_{\text{oc}}$
- BCH$(63,45)$ with the standard $18\times 63$ circulant PC matrix $\mathbf{H}_{\text{std}}$ and overcomplete $189\times 63$ PC matrix $\mathbf{H}_{\text{oc}}$
- RM$(128,99)$ with the standard $29\times 128$ PC matrix $\mathbf{H}_{\text{std}}$ and overcomplete $10668\times 128$ PC matrix $\mathbf{H}_{\text{oc}}$

For some of the considered codes, standard table Q-learning is feasible. For example, RM$(32,16)$ has $|\mathcal{S}| = 2^{16} = 65536$ and $|\mathcal{A}| = 32$ so the Q-table has $|\mathcal{S}||\mathcal{A}| \approx 2\cdot 10^6$ entries.

### A. Training Hyperparameters

In the following, we set the maximum number of decoding iterations to $T = 10$ and the discount factor to $\gamma = 0.99$. For standard table Q-learning, the $(\varepsilon,\varepsilon_g)$-goal exploration strategy is adopted with fixed $\varepsilon = 0.6$, $\varepsilon_g = 0.3$, and learning rate $\alpha = 0.1$. For fitted Q-learning based on NNs, we use $\varepsilon$-greedy exploration where $\varepsilon$ is linearly decreased from $0.9$ to $0$ over the course of $0.9K$ learning episodes (i.e., number of decoded codewords), where the total number of episodes $K$ depends on the scenario. For the gradient optimization, the Adam optimizer is used with a batch size of $B = 100$ and learning rate $\alpha = 3\cdot 10^{-5}$. The training SNR for both standard Q-learning and fitted Q-learning is fixed at $E_b/N_0 = 5$ dB for RM$(128,99)$ and $E_b/N_0 = 4$ dB for all other codes. In general, better performance may be obtained by re-optimizing parameters for each SNR or by adopting parameter adapter networks that dynamically adapt the network parameters to the SNR [28].

### B. Learning Convergence in Q-Learning

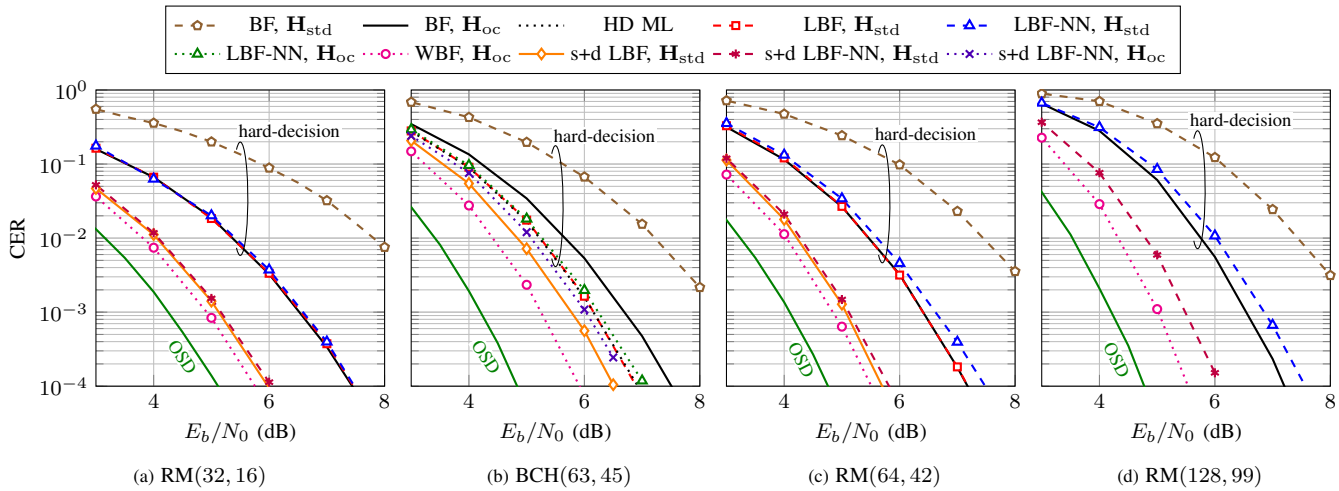
We start by comparing the learning convergence of the proposed exploration strategy (14) to the $\varepsilon$-greedy exploration for standard Q-learning assuming RM$(32,16)$ over the BSC. In Fig. 3, the obtained performance in terms of codeword error rate (CER) is shown as a function of the Q-learning iteration. The shown learning curves are generated as follows. During Q-learning, we always decode first the new channel observations (line 3 of Alg. 2) with the current Q-function without exploration and save the binary outcome (success/failure). Then, we plot a moving average (window size 5000) of the outcomes to approximate the CER. It can be seen that the proposed strategy converges significantly faster than $\varepsilon$-greedy exploration. We also show a learning curve for training when a reward of 1 is given only for finding the transmitted codeword; in this case, however, the process is not an MDP (see Sec. IV) and the performance can become worse during training.

### C. Binary Symmetric Channel

Fig. 4 shows the CER performance for all considered scenarios as a function of $E_b/N_0$. We start by focusing on the "hard-decision" decoding cases, which are equivalent to assuming transmission over the BSC. Supplementary bit error rate (BER) results for the same scenarios are shown in Fig. 5.

*1) Baseline Algorithms:* As a baseline for the LBF decoders over the BSC, we use BF decoding according to Alg. 1 (see also [8, Alg. II] and [14, Alg. 10.2]) applied to both the standard and overcomplete PC matrices $\mathbf{H}_{\text{std}}$ and $\mathbf{H}_{\text{oc}}$, respectively. We also implemented optimal syndrome decoding for RM$(32,16)$ and BCH$(63,45)$. In general, BF decoding shows relatively poor performance when applied to $\mathbf{H}_{\text{std}}$, whereas the performance increases drastically for $\mathbf{H}_{\text{oc}}$ (see also [8], [26]). In fact, for RM$(32,16)$, standard BF for $\mathbf{H}_{\text{oc}}$ gives virtually the same performance as optimal decoding and the latter performance curves are omitted from the figure. This performance increase comes at a significant increase in complexity, e.g., for RM$(32,16)$, the overcomplete PC matrix has 620 rows compared to the standard PC matrix with only 16 rows. For the BCH code, there still exists a visible performance gap between optimal decoding and BF decoding based on $\mathbf{H}_{\text{oc}}$.

[5] $\mathbf{H}$-matrices and source code for the simulations are available online at `https://github.com/fabriziocarpi/RLdecoding`. We first used our own Tensorflow RL implementation and later switched to `RLlib` [27] in order to use multi-core parallelism for training rollouts.




(a) RM$(32, 16)$ (b) BCH$(63, 45)$ (c) RM$(64, 42)$ (d) RM$(128, 99)$

Fig. 4: Simulation results for learned BF decoding. In (a), results for standard BF (Alg. 1) applied to $\mathbf{H}_{\text{oc}}$ overlap with hard-decision ML and are omitted. (BF: bit-flipping, WBF: weighted BF, LBF: learned BF (table Q-learning), LBF-NN: LBF with neural networks (fitted/deep Q-learning), s+d: sort and discard the channel reliabilities, HD ML: hard-decision maximum-likelihood, OSD: ordered statistics decoding)

*2) Q-learning:* From Figs. 4(a) and (b), it can be seen that the LBF decoders based on table Q-learning for RM$(32, 16)$ and BCH$(63, 45)$ converge essentially to the optimal performance. For RM$(64, 42)$ in Fig. 4(c), the performance of LBF decoding is virtually the same as for standard BF decoding using $\mathbf{H}_{\text{oc}}$, which leads us to believe that both schemes are optimal in this case. These results show that the proposed RL approach is able to learn close-to-optimal flipping patterns given the received syndromes. Note that for RM$(128, 99)$, Q-learning would require a table with $|\mathcal{S}||\mathcal{A}| \approx 7 \cdot 10^{10}$ entries which is not feasible to implement on our system.

*3) Fitted Q-learning:* The main disadvantage of the standard Q-learning approach is the large storage requirements of the Q-table. Indeed, the requirements are comparable to optimal syndrome decoding and this approach is therefore only feasible for short or very-high-rate codes. Therefore, we also investigate to what extend the Q-tables can be approximated with NNs and fitted Q-learning. The number of neurons in the hidden layer of the NNs is chosen to be $1500$ for RM$(128, 99)$ and $500$ for all other cases. The achieved performance is shown in Fig. 4, labeled as "LBF-NN". For the RM codes, it was found that good performance can be obtained using fitted Q-learning using the standard PC matrix $\mathbf{H}_{\text{std}}$. The performance loss compared to table Q-learning is almost negligible for RM$(32, 16)$ and increases slightly for the longer RM codes. For the BCH code, we found that fitted Q-learning works better using $\mathbf{H}_{\text{oc}}$ compared to $\mathbf{H}_{\text{std}}$. For this case, the gap compared to optimal decoding is less than $0.1$ dB at a CER of $10^{-3}$.

### D. AWGN Channel

Next, we consider the AWGN channel assuming that the reliability information is exploited for decoding.

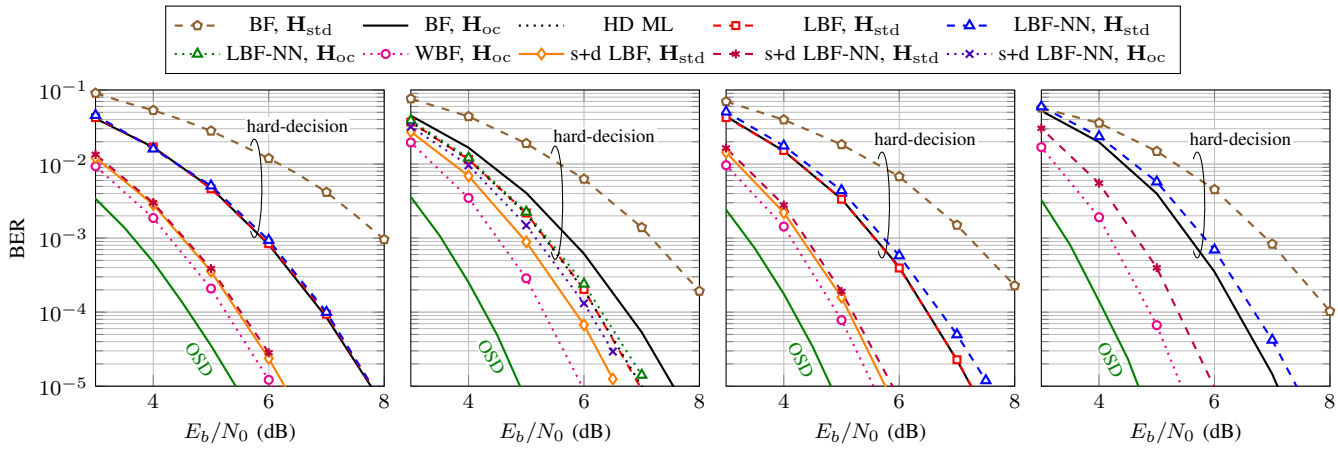
*1) Baseline Algorithms:* Ordered statistics decoding (OSD) is used as a benchmark, whose performance is close to ML [29]. In this paper, we use order-$\ell$ processing where $\ell = 3$ in all cases. Furthermore, we employ WBF decoding according to [14, Alg. 10.3] using $\mathbf{H}_{\text{oc}}$. Similar to BF decoding over the BSC, the performance of WBF is significantly better for overcomplete PC matrices compared to the standard ones (results for WBF on $\mathbf{H}_{\text{std}}$ are omitted). From Fig. 4, WBF decoding is within $0.6$–$1.1$ dB of OSD for the considered codes. We remark that there also exist a number of improved WBF algorithms which may reduce this gap at the expense of additional decoding complexity and the necessity to tune various weight and threshold parameters, see [8]–[13]. For RM codes of moderate length, ML performance can also be approached using other techniques [30].

*2) Q-Learning:* As explained in Sec. V, our approach to LBF decoding over the AWGN channel in this paper consists of permuting the bit positions based on $\boldsymbol{r}$ and subsequently discarding the reliability values. For the RM codes, the particular permutation strategy is described in Sec. V. The performance results for standard Q-learning shown in Figs. 4(a) and (c) (denoted as "s+d LBF") demonstrate that this strategy performs quite close to WBF decoding and closes a significant fraction of the gap to OSD, even though reliability information is only used to select the permutation and not for the actual decoding. For the BCH code, we use the same permutation strategy as described in [6]. In this case, however, the performance improvements due to applying the permutations are relatively limited.

*3) Fitted Q-Learning:* For the NN-based approximations of the Q-tables for the sort-and-discard approach, we use the NN sizes from the previous section for the BSC. In this case, fitted Q-learning obtains performance close to the standard Q-learning approach for RM codes. Similar to the BSC, the performance gap is almost negligible for RM$(32, 16)$ and increases for the longer RM codes. For RM$(128, 99)$, sort-and-discard LBF decoding with NNs closes roughly half the gap between soft-decision ML (approximated via OSD) and hard-decision ML (approximated via BF on $\mathbf{H}_{\text{oc}}$).

## VII. Conclusion

In this paper, we have proposed a novel RL framework for BF decoding of binary linear codes. It was shown how BF




(a) RM$(32, 16)$ (b) BCH$(63, 45)$ (c) RM$(64, 42)$ (d) RM$(128, 99)$

Fig. 5: Bit error rate (BER) results for the same scenarios as considered in Fig. 4.

decoding can be mapped to a Markov decision process by properly choosing the state and action spaces, whereas the reward function can be based on a reformulation of the ML decoding problem. In principle, this allows for data-driven learning of optimal BF decision strategies. Both standard (table-based) and fitted Q-learning with NN function approximators were then used to learn good decision strategies from data. Our results show that the learned BF decoders can offer a range of performance–complexity trade-offs.

## REFERENCES


[1] F. Carpi, "Exploring machine learning algorithms for decoding linear block codes," Master Thesis, University of Parma, Italy, Oct. 2018.

[2] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, "On deep learning-based channel decoding," in *Proc. Annual Conf. Information Sciences and Systems (CISS)*, Baltimore, MD, 2017.

[3] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[4] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *Proc. Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, 2016.

[5] L. G. Tallini and P. Cull, "Neural nets for decoding error-correcting codes," in *Proc. IEEE Technical Applications Conf. and Workshops*, Portland, USA, 1995.

[6] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes - a syndrome-based approach," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, 2018.

[7] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," in *Proc. Int. Conf. Learning Representations (ICLR)*, Vancouver, Canada, 2018.

[8] M. Bossert and F. Hergert, "Hard- and soft-decision decoding beyond the half minimum distance—an algorithm for linear codes (corresp.)," *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 709–714, Sept. 1986.

[9] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[10] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 165–167, March 2004.

[11] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Commun. Lett.*, vol. 9, no. 9, pp. 814–816, Sept. 2005.

[12] Z. Liu and D. A. Pados, "A decoding algorithm for finite-geometry LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 3, pp. 415–421, March 2005.

[13] M. Shan, C. Zhao, and M. Jiang, "Improved weighted bit-flipping algorithm for decoding LDPC codes," *IEE Proc. Commun.*, vol. 152, no. 6, p. 919, Dec. 2005.

[14] W. Ryan and S. Lin, *Channel Codes Classical and Modern*. Cambridge University Press, 2009.

[15] X. Wang, H. Zhang, R. Li, L. Huang, S. Dai, Y. Huangfu, and J. Wang, "Learning to flip successive cancellation decoding of polar codes with LSTM networks," *arXiv:1902.08394*, Feb. 2019.

[16] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *arxiv:1810.07862*, 2018.

[17] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *Proc. Conf. Uncertainty in AI (UAI)*, Boston, MA, 2006.

[18] A. I. Vila Casado, M. Griot, and R. D. Wesel, "LDPC decoders with informed dynamic scheduling," *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3470–3479, Dec. 2010.

[19] C. Häger and H. D. Pfister, "Approaching miscorrection-free performance of product codes with anchor decoding," *IEEE Trans. Commun.*, vol. 66, no. 7, pp. 2797–2808, July 2018.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, 1989.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.

[22] J. Jiang and K. R. Narayanan, "Iterative soft decoding of Reed-Solomon codes," *IEEE Commun. Lett.*, vol. 8, no. 4, pp. 244–246, April 2004.

[23] T. R. Halford and K. M. Chugg, "Random redundant soft-in soft-out decoding of linear block codes," *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2006.

[24] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Elsevier, 1977.

[25] Y. Lei, A. Alvarado, B. Chen, X. Deng, Z. Cao, J. Li, and K. Xu, "Decoding staircase codes with marked bits," in *Proc. Int. Symp. Turbo Codes and Iterative Information Processing (ISTC)*, Hong Kong, 2018.

[26] E. Santi, C. Häger, and H. D. Pfister, "Decoding Reed-Muller codes using minimum-weight parity checks," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Vail, CO, 2018.

[27] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, I. Stoica, "RLlib: Abstractions for Distributed Reinforcement Learning," in *Proc. Int. Conf. on Machine Learning (ICML)*, Stockholm, Sweden, 2018.

[28] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, "Learned belief-propagation decoding with simple scaling and SNR adaptation," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Paris, France, 2019.

[29] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1379–1396, Sept. 1995.

[30] I. Dumer and K. Shabunov, "Soft-decision decoding of Reed-Muller codes: Recursive lists," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, March 2006.