Reinforcement Learning Enhancements on Layered LDPC Decoding

Tirtankar Mittra[1]

***Abstract*—Low Density Parity Check (LDPC) codes, invented by Gallager [1], have revolutionized communications by being capacity approaching codes while possessing a relatively low decoding complexity. With the advent of 5G communications, vRAN and O-RAN technologies, the usual LDPC decoder does not perform well with respect to low latency requirements. Layered LDPC decoder [2], one such variation which addresses the low latency requirements. In this paper, we discuss how machine learning algorithms like Reinforcement Learning(RL) can be used in conjunction with Layered LDPC decoder to further reduce the latency.**

***Index Terms*—Layered LDPC codes, Reinforcement Learning.**

I. Introduction

LDPC decoding involves belief propagation which is an iterative process.There are many variations of LDPC decoder, the most popular of them is Layered LDPC decoder because the soft LLR values converge faster with Layered LDPC decoder. Figure 1 shows a parity check matrix or H matrix

$$\mathbf{H} = \left[\begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

Fig. 1. Parity Check Matrix divided into Layers.

divided in two layers. In normal operation of Layered LDPC decoder, soft received(Rx) LLRs are first decoded using layer 1 as a result some of the bits are corrected then layer 2 is used which may further correct some other bits in the codeword i.e. in traditional Layered LDPC decoder layers are taken sequential for decoding the data. My proposal is to use some different permutation of layers which would result in faster convergence of the codeword. I am at finding this using Reinforcement learning algorithm. Figure 2 shows the block

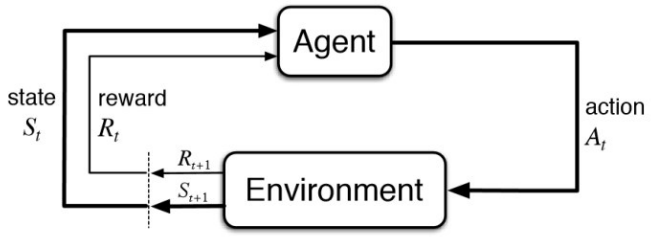

Fig. 2. Generic Block diagram of Reinforcement Learning.

diagram of general Reinforcement learning scheme. Based on the current state (S_t) an action (A_t) is selected by the agent according to some probability distribution which maps states into actions this is called the initial policy π_0. Based on the action the environment transitions to state (S_{t+1}) and generates an immediate reward of r this dynamics is dictated by the conditional probability distribution $p(S_{t+1}, r|S_t, A_t)$. Our aim is to use iterative policy evaluation using bellman optimality equation which is given by following Equation [1] to calculate the state value function under the policy π. Since the dynamics of the system is deterministic which means given state(S) and action(a) the system will only transition to state($S^{'}$), we can simplify Equation [1] and Equation [3] to Equation [2] and Equation [4] respectively.

$$v_{\pi k+1}(S_t) = \sum_{A_t} \pi(A_t|S_t) \sum_{S_{t+1},r} p(S_{t+1}, r|S_t, A_t)[r + \gamma v_{\pi k}(S_{t+1})] \tag{1}$$

$$= \sum_{A_t} \pi(A_t|S_t)[r + \gamma v_{\pi k}(S_{t+1})] \tag{2}$$

In Equation [1], k is the k-th iteration, $v_\pi(S)$ is the value of the state which gives us an approximate expected long term reward we get from state S when policy π is followed, γ is the discount factor and $S^{'}$ is the next state when action a is taken under the state S.
Then we use Equation [3] to find the next better policy. Ideally policy evaluation and policy improvement is carried on greedily until an optimal policy is found[$\pi = \pi_*$].

$$\pi^{'}(S_t) = argmax_{A_t} \sum_{r,S_{t+1}} p(S_{t+1}, r|S_t, A_t)[r + \gamma v_\pi(S_{t+1})] \tag{3}$$

$$= argmax_{A_t}[r + \gamma v_\pi(S_{t+1})] \tag{4}$$

II. Proposed Scheme

In this section, we describe the algorithm used in detail. Figure 3 shows the state transition diagram of our Reinforcement learning model. S_0 is the initial state $(S_1, S_2....S_N)$ are special states, each permutation of layers is a special state. The initial policy is to select actions which transitions to all special states uniformly randomly from the initial state. All policies also selects actions which transitions the special state to itself until R_x codeword converges. In every special state when the codeword doesn't converge an immediate reward of -1 is added, this encourages the codeword to converge faster. Instead of always greedily choosing next state $(S^{'})$ which give maximum $v(S^{'})$ we also explore other actions(A_t) which lead to sub-optimal next state, using the upper-confidence bound action selection method. Equation [5].

$$A_t = argmax_a \left[Q_t(S,a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{5}$$

Where, $c > 0$ controls the degree of exploration, $N_t(a)$ is the number of times action a was selected, t is incremented every time an action is selected and $Q_t(S,a)$ is the action value pair which gives the long term expected reward when action a is selected in state S. Corner cases were handled for $t = 0$ and $N_t(a) = 0$ in our code.

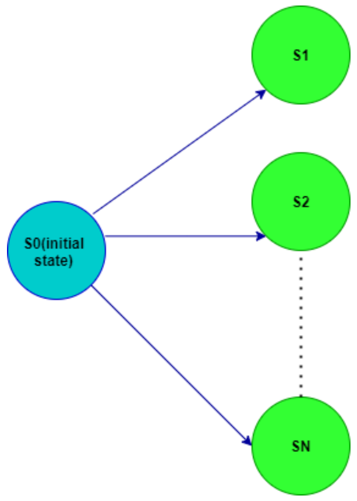

Fig. 3. State Transition Diagram of our Reinforcement Learning Model.

Algorithm 1 Layered LDPC Decoding using Reinforcement Learning.

```
1:  Assign (number of layers L),(codeword bit-size N),(message bit-size K) such that (N-K)%L=0.
2:  Design a H matrix and G matrix.
3:  Generate M number of K sized message bits.
4:  Divide M number into B batches i.e. $[b_0, b_1, b_2 \ldots b_{B-1}]$.
5:  Encode then, Modulate(BPSK) then, Add AWGN noise to the batch of K sized message bits $[b_0, b_1, b_2 \ldots b_{B-1}]$ $[b_0^{'}, b_1^{'}, ^{'} b_2 \ldots b_{B-1}^{'}]$ (modified batch).
6:  Divide H matrix into layers $[h_0, h_1, h_2 \ldots h_{L-1}]$.
7:  Create state transition diagram & RL model.
8:  Every time the code doesn't converge an immediate reward of -1 is generated.
9:  For each batch $b_i^{'}$ in $[b_0^{'}, b_1^{'}, ^{'} b_2 \ldots b_{B-1}^{'}]$
10:     For each codeword $R_x$ in $b_i^{'}$
11:         Run policy evaluation on $R_x$; use Equation [2].
12:         Find pattern $\forall A_t$ which maximise $v_\pi(S^{'})$ for $R_x$.
13:     end-for.
14:     Update policy; use Equation [4].
15: end-for
16: Print Terminal States with maximum state value.
```

III. RESULTS

I ran the code for a codeword of size(n=15) and message bits of size(k=6) with number of layers(L=9), for different noise levels. BPSK modulation used and an AWGN noise channel was assumed. For different noise-levels the best case v/s worst case number of tanner graph iteration are plotted in Figure 4.

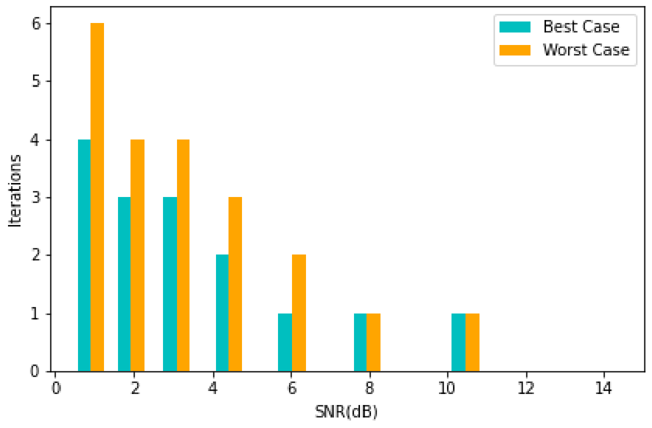

Fig. 4. Worst and Best case iterations for different noise levels (n=15/k=6)

Also while manually studying policies which yielded faster convergence for a particular codeword; i.e. step[12] of Algorithm 1 following observations were noticed.

- Observation 1- While designing the H-matrix it is preferable that we design layers such that columns that have 1s only have one 1. If certain columns have more than one 1 we can remove the rows contributing that extra 1 to the next layer.
- Observation 2- While designing the H-matrix it is preferable that we ensure every column in one layer has at least one 1.
- Observation 3- While designing the H-matrix it is preferable that we keep layers which have maximum number of rows satisfying the parity check equations before the other layers.

IV. CONCLUSION

We found out that for codewords which are decoded using certain ordering of layers provide faster convergence to the solution i.e. 1-2 iterations faster. We also used Reinforcement learning to determine which ordering of layers provide faster convergence for both average and specific cases.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[2] X. H. K. Zhang and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic ldpc codes," *IEEE Journal on Selected Areas in Communications, vol. 27, no. 6, pp. 985-994, August 2009, doi: 10.1109/JSAC.2009.090816.*