



TBSI 清华-伯克利深圳学院
Tsinghua-Berkeley Shenzhen Institute



Cyclically Equivariant Neural Decoders for Cyclic Codes

Xiangyu Chen Min Ye

Tsinghua-Berkeley Shenzhen Institute

Tsinghua Shenzhen International Graduate School

Tanner Graph

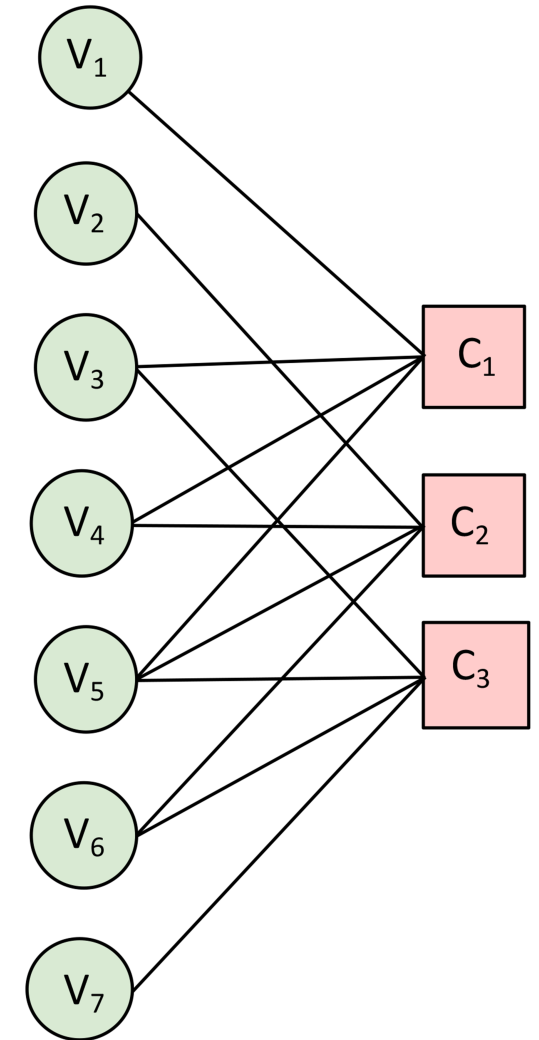
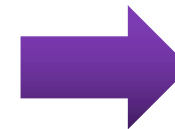
A linear code with code length n and code dimension k

Binary parity check matrix H of size $(n-k) \times n$

A parity check matrix of the $(n=7, k=4)$ Hamming code

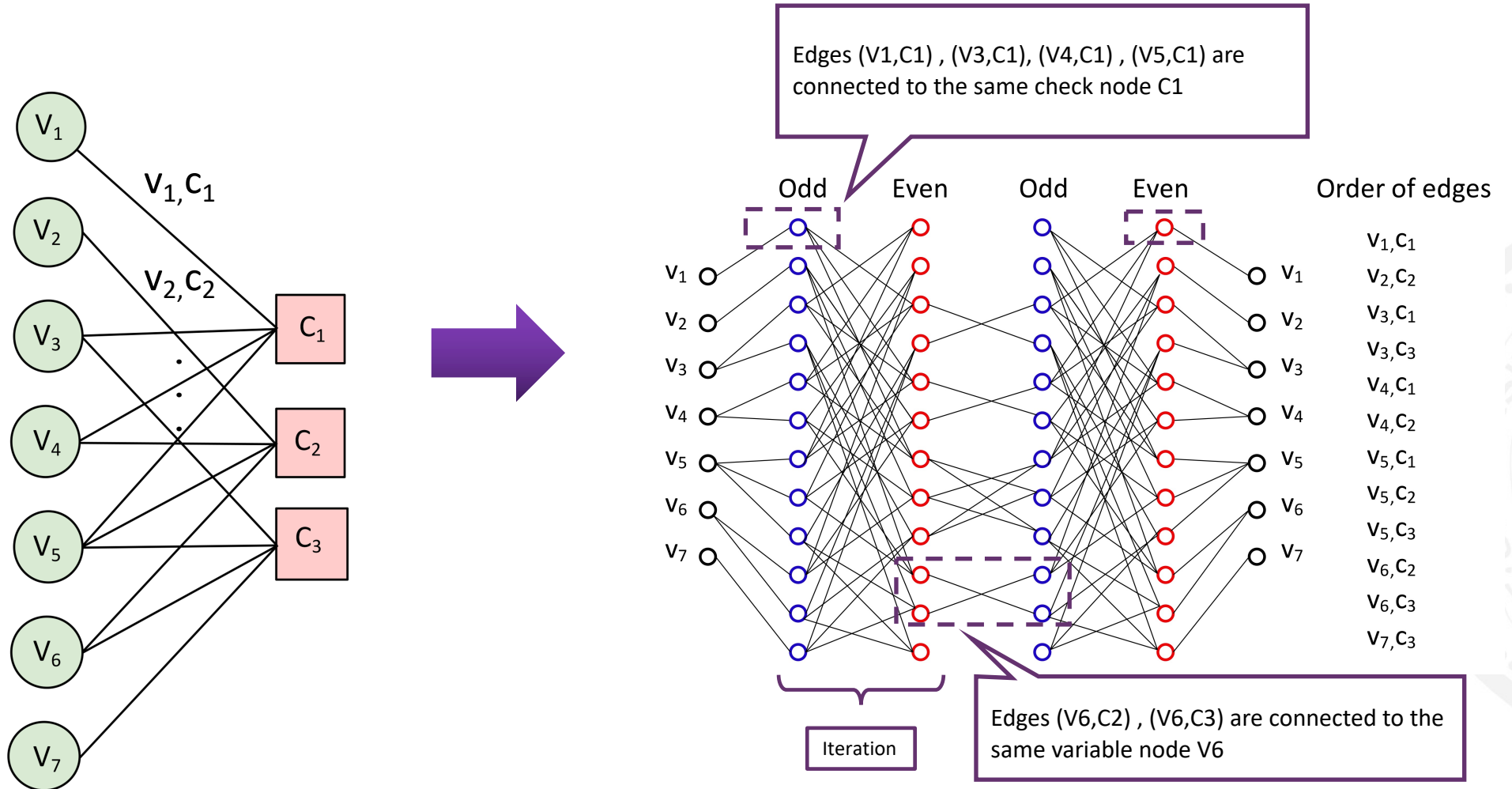
	v_1	v_2	v_3	v_4	v_5	v_6	v_7
c_1	1	0	1	1	1	0	0
c_2	0	1	0	1	1	1	0
c_3	0	0	1	0	1	1	1

Tanner graph: n variable nodes and $(n - k)$ check nodes



Trellis Graph

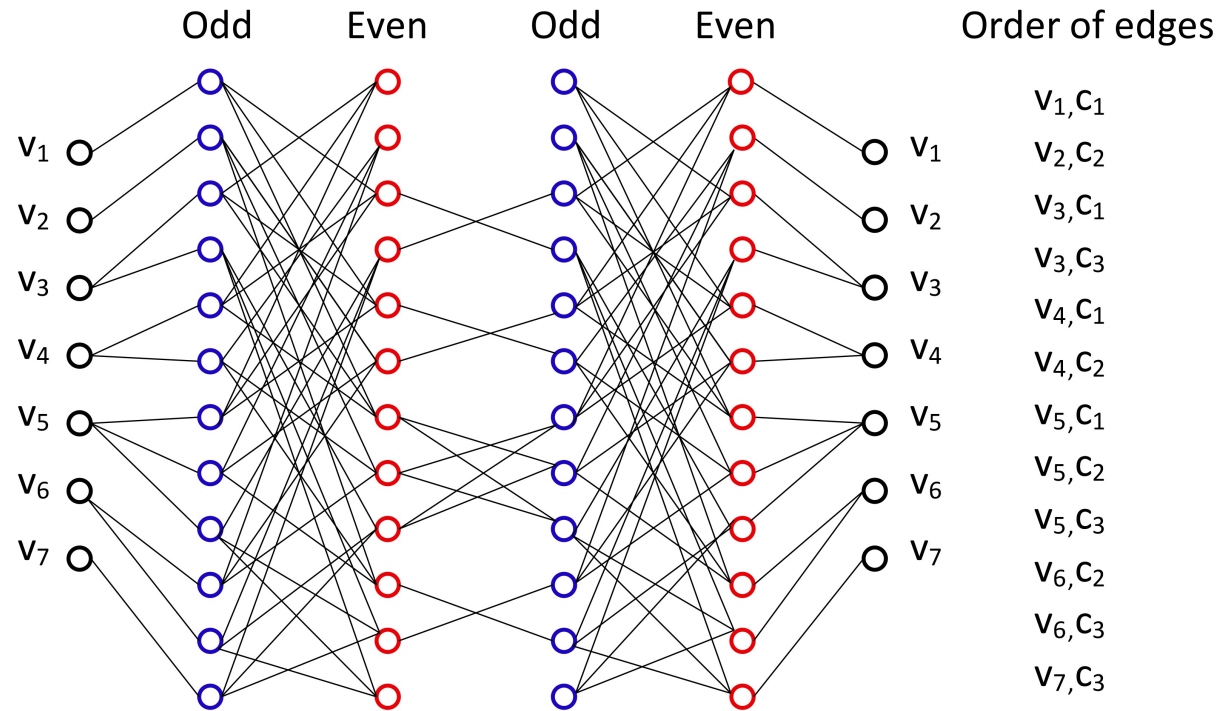
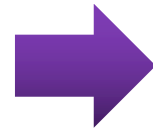
The trellis graph is obtained from the unrolling Tanner graph



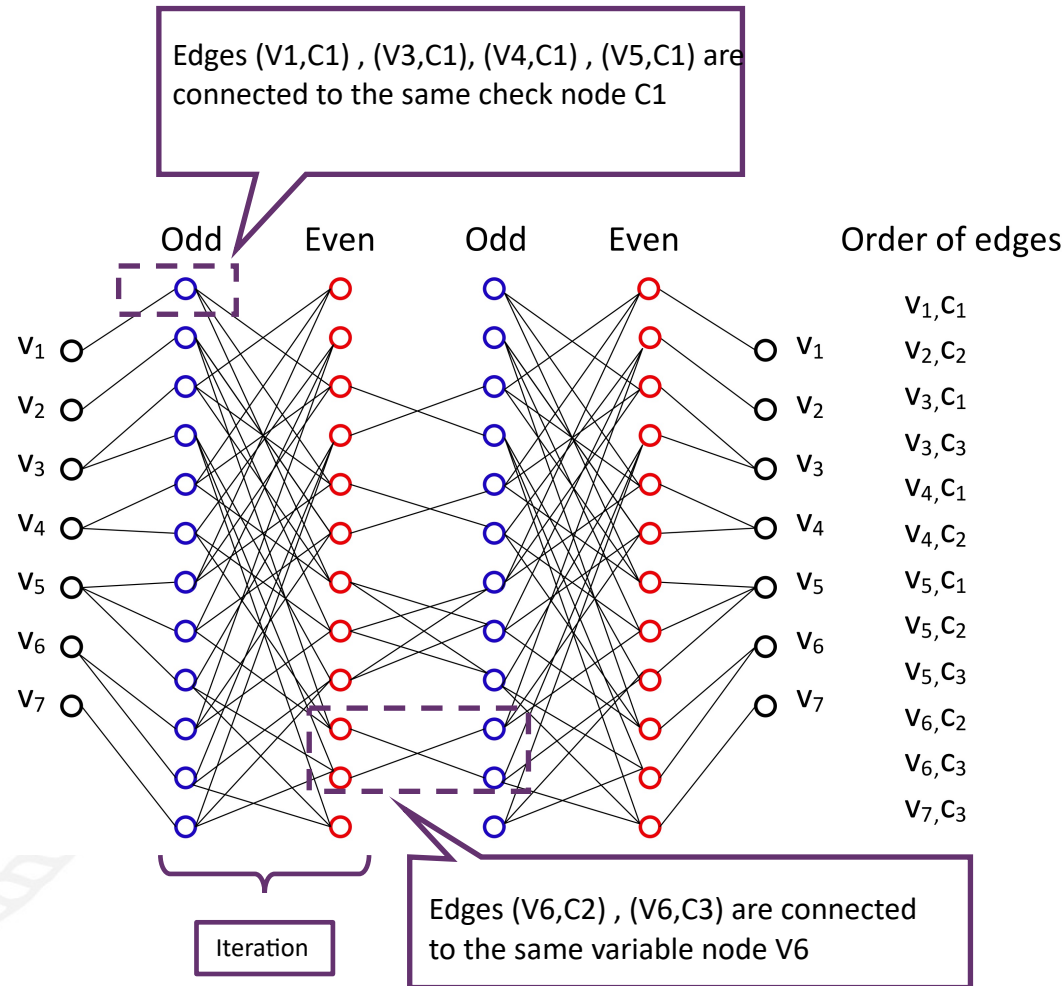
Belief Propagation Decoding

The n variable nodes in the input layer hold the log likelihood ratios (LLR) of input bits

$$L_j = \log \frac{\mathbb{P}(y_j | C_j = 0)}{\mathbb{P}(y_j | C_j = 1)}$$



Belief Propagation Decoding



The classic Belief Propagation (BP) decoding algorithms

For odd s :

$$x^{[s]}(e) = x^{[s]}((c_i, v_j)) \\ = \tanh \left(\frac{1}{2} \left(L_j + \sum_{e' \in N(v_j) \setminus \{e\}} x^{[s-1]}(e') \right) \right)$$

For even s :

$$x^{[s]}(e) = 2 \tanh^{-1} \left(\prod_{e' \in N(c_i) \setminus \{e\}} x^{[s-1]}(e') \right)$$

The output of the classic BP:

$$o_j = L_j + \sum_{e \in N(v_j)} x^{[2t]}(e) \quad \text{for } j \in [n]$$

Neural Belief Propagation Decoding

The classic Belief Propagation (BP) decoding algorithms

For odd s :

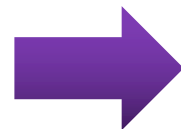
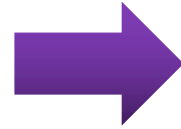
$$\begin{aligned} x^{[s]}(e) &= x^{[s]}((c_i, v_j)) \\ &= \tanh\left(\frac{1}{2}\left(L_j + \sum_{e' \in N(v_j) \setminus \{e\}} x^{[s-1]}(e')\right)\right) \end{aligned}$$

For even s :

$$x^{[s]}(e) = 2 \tanh^{-1}\left(\prod_{e' \in N(c_i) \setminus \{e\}} x^{[s-1]}(e')\right)$$

The output of the classic BP:

$$o_j = L_j + \sum_{e \in N(v_j)} x^{[2t]}(e) \quad \text{for } j \in [n]$$



Neural belief propagation define in (Nachmani et al., 2016; 2018)

For odd s :

$$\begin{aligned} x^{[s]}(e) &= x^{[s]}((c_i, v_j)) = \tanh\left(\frac{1}{2}\left(w^{[s]}(j, e)L_j \right. \right. \\ &\quad \left. \left. + \sum_{e' \in N(v_j) \setminus \{e\}} w^{[s]}(e', e) x^{[s-1]}(e')\right)\right) \end{aligned}$$

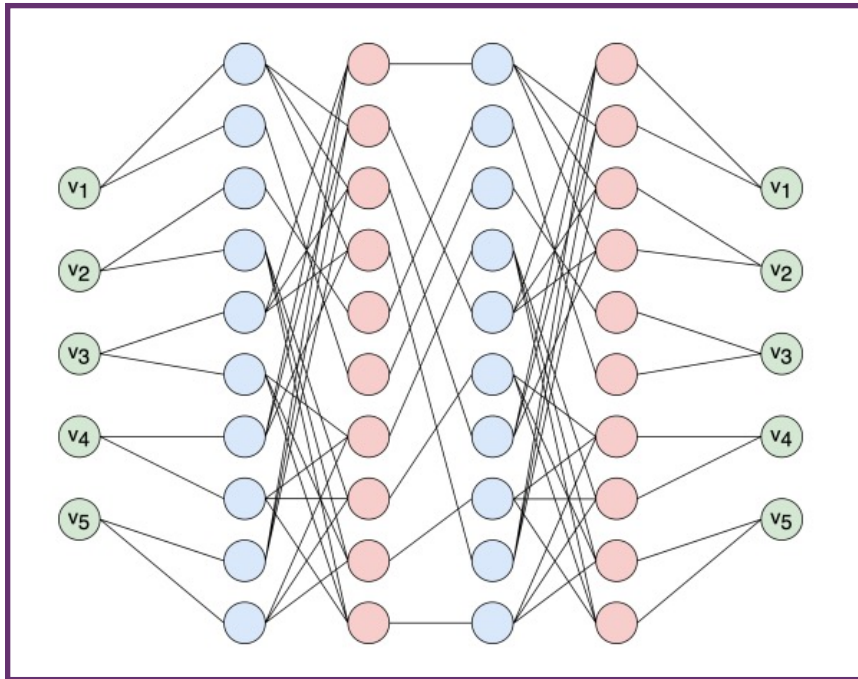
For even s :

$$x^{[s]}(e) = 2 \tanh^{-1}\left(\prod_{e' \in N(c_i) \setminus \{e\}} x^{[s-1]}(e')\right)$$

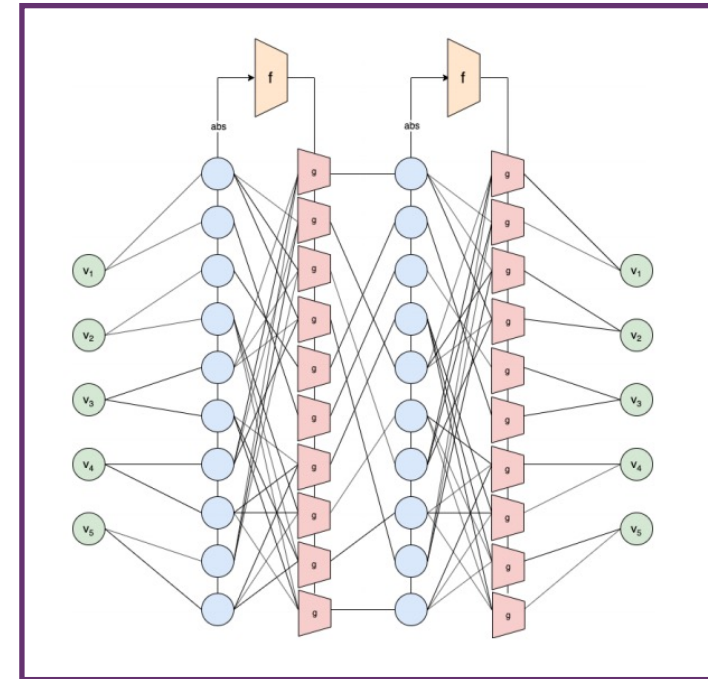
The output of the neural BP:

$$o_j = L_j + \sum_{e \in N(v_j)} w^{\text{out}}(e, j) x^{[2t]}(e)$$

Previous works



(Nachmani et al., 2018)^[1]



(Nachmani et al., 2019)^[2]

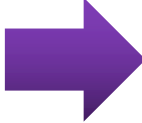
[1] Nachmani E, Marciano E, Lugosch L, Gross WJ, Burshtein D, Beery Y. Deep Learning Methods for Improved Decoding of Linear Codes[J]. IEEE Journal of Selected Topics in Signal Processing, 2018.

[2] Nachmani, E., & Wolf, L. (2019). Hyper-Graph-Network Decoders for Block Codes. Advances in Neural Information Processing Systems, 32, 2329-2339.

Cyclic code

Cyclic codes are invariant under cyclic shifts

Example: (n=7, k=4) Hamming code

1	0	1	1	1	0	0	cyclic shift	0	1	0	1	1	1	0
0	1	0	1	1	1	0		0	0	1	0	1	1	1
0	0	1	0	1	1	1		1	0	0	1	0	1	1

These two matrices are cyclic shifts of each other, and they have the same row space

ML decoder of a cyclic code is equivariant to all cyclic shifts

Cyclic Neural Decoder

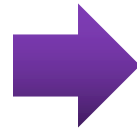
Shares the equivariant property of the ML decoder

Use parity check matrix of size $n \times n$ instead of $(n-k) \times n$

e.g. ($n = 7, k = 4$) Hamming code

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
c_1	1	0	1	1	1	0	0
c_2	0	1	0	1	1	1	0
c_3	0	0	1	0	1	1	1

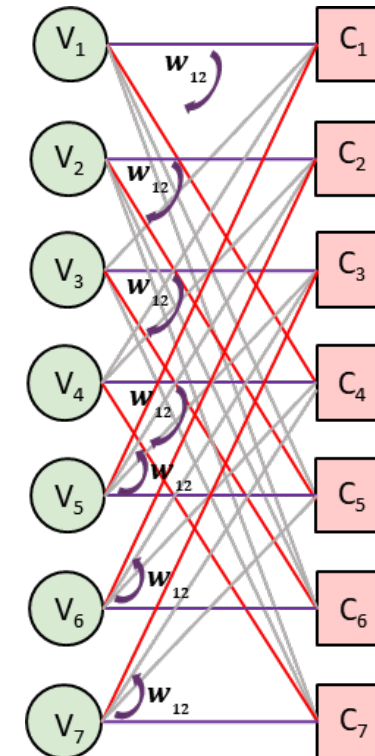
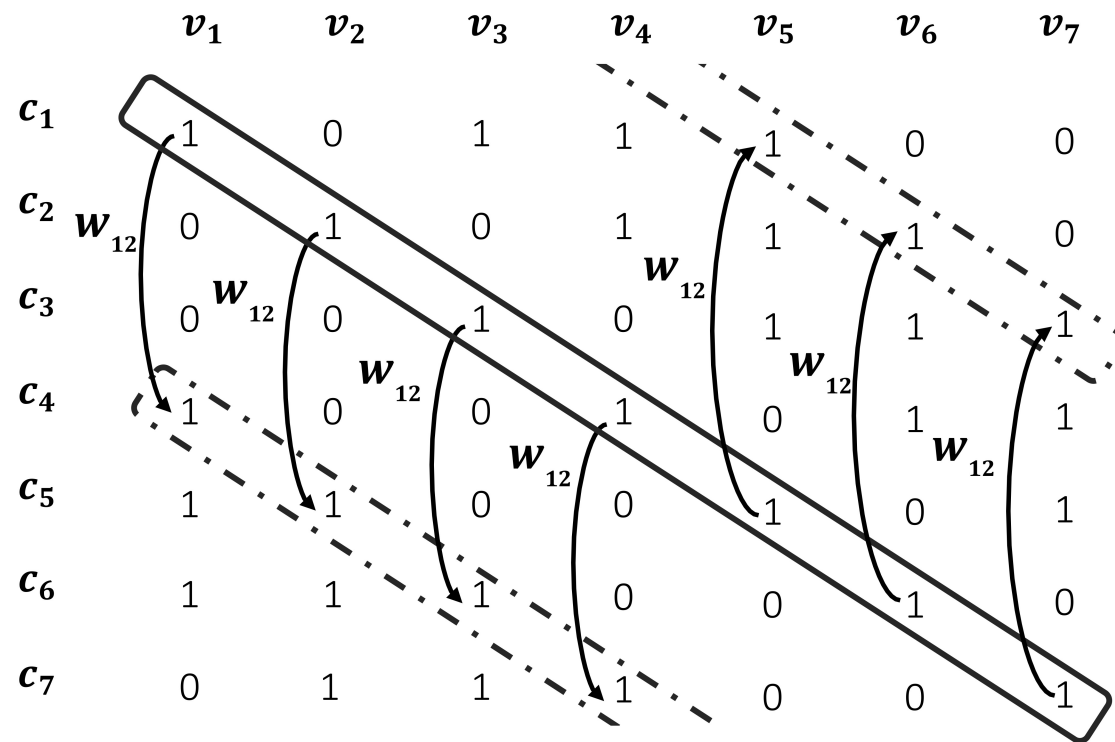
cyclic shift



	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
c_0	1	0	1	1	1	0	0	0
c_1	0	1	0	1	1	1	0	0
c_2	0	0	1	0	1	1	1	0
c_3	0	0	0	1	0	1	1	1
c_4	1	0	0	0	1	0	1	1
c_5	1	1	0	0	0	1	0	1
c_6	1	1	1	0	0	0	1	0
c_7	0	1	1	1	0	0	0	1

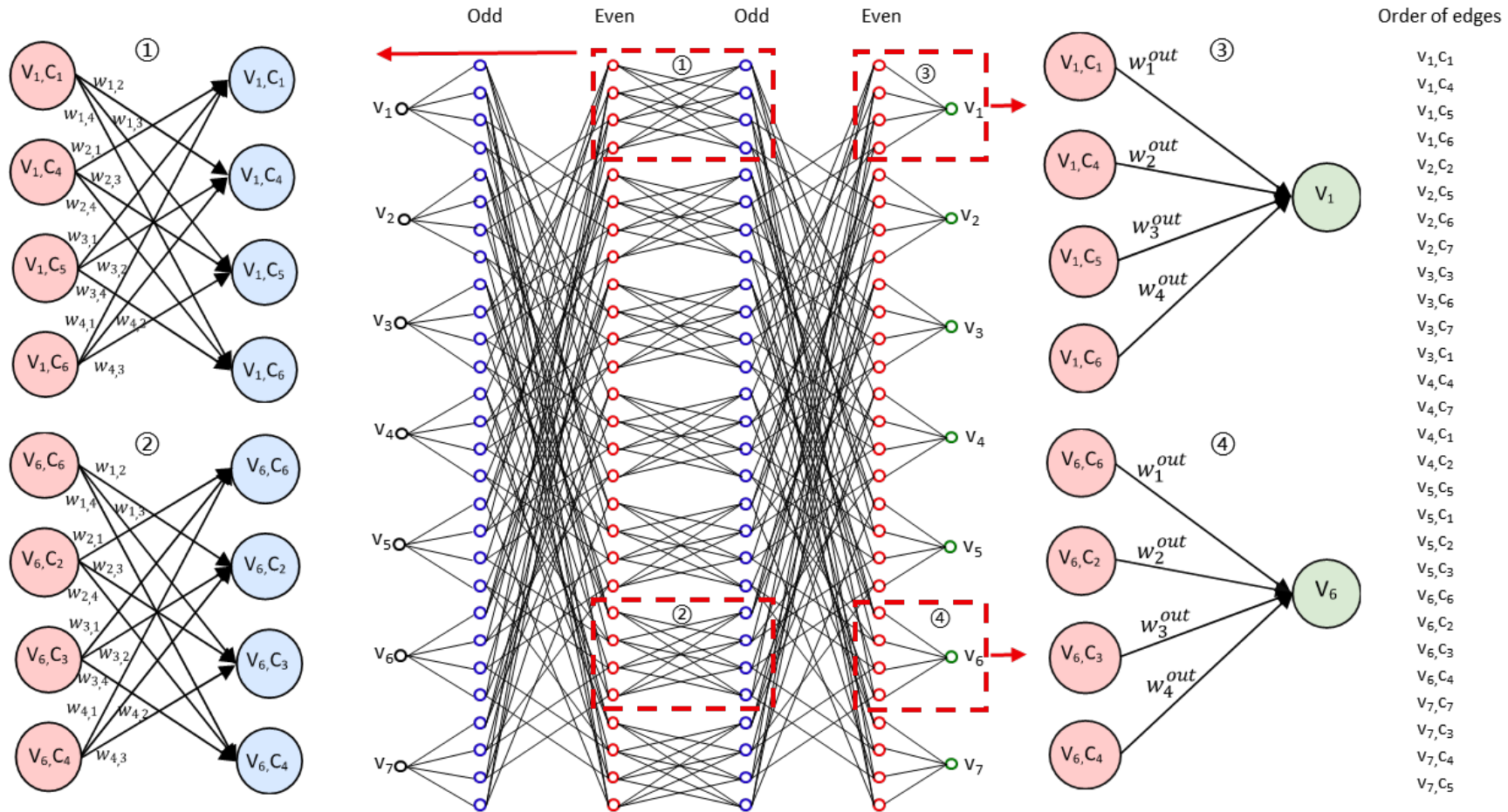
Cyclic Neural Decoder

Impose a shift-invariant structure on the weights of our new decoder



Cyclic Neural Decoder

The Trellis graph corresponding to the matrix ($n=7, k=7$)



Cyclic Neural Decoder

Neural belief propagation define in
(Nachmani et al., 2016; 2018)

For odd s :

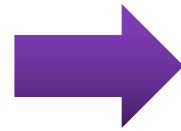
$$x^{[s]}(e) = x^{[s]}((c_i, v_j)) = \tanh \left(\frac{1}{2} \left(w^{[s]}(j, e) L_j + \sum_{e' \in N(v_j) \setminus \{e\}} w^{[s]}(e', e) x^{[s-1]}(e') \right) \right)$$

For even s :

$$x^{[s]}(e) = 2 \tanh^{-1} \left(\prod_{e' \in N(c_i) \setminus \{e\}} x^{[s-1]}(e') \right)$$

The output of the neural BP:

$$o_j = L_j + \sum_{e \in N(v_j)} w^{\text{out}}(e, j) x^{[2t]}(e)$$



Cyclic Neural Decoder

For odd s :

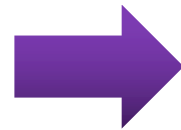
$$x^{[s]}(e) = x^{[s]}((c_{\pi_j(i_b)}, v_j)) = \tanh \left(\frac{1}{2} \left(w_b^{[s]} L_j + \sum_{b' \in [u] \setminus \{b\}} w_{b', b}^{[s]} x^{[s-1]}((c_{\pi_j(i_{b'})}, v_j)) \right) \right)$$

For even s :

$$x^{[s]}(e) = 2 \tanh^{-1} \left(\prod_{e' \in N(c_i) \setminus \{e\}} x^{[s-1]}(e') \right)$$

The output of the neural BP:

$$o_j = L_j + \sum_{b \in [u]} w_b^{\text{out}} x^{[2t]}((c_{\pi_j(i_b)}, v_j))$$



List Decoding Procedure

In the list decoding procedure, we use the permutations from a subset S of the affine group

$$|\mathcal{S}| = n + 1$$

(n+1,n+1) Permutations Matrix

0	1	2	3	4	5	6	7
1	0	4	7	2	6	5	3
2	4	0	5	1	3	7	6
4	2	1	6	0	7	3	5
3	7	5	0	6	2	4	1
7	3	6	1	5	4	2	0
5	6	3	2	7	0	1	4
6	5	7	4	3	1	0	2

Step 1: Prepend a dummy symbol $L_0 = 0$ to the LLR vector

L_1	L_2	L_3	L_4	L_5	L_6	L_7
-------	-------	-------	-------	-------	-------	-------

L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
-------	-------	-------	-------	-------	-------	-------	-------

Step 2: Apply ℓ permutations to (L_0, L_1, \dots, L_n)

$\ell = 1$

L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7
-------	-------	-------	-------	-------	-------	-------	-------

$\ell = 2$

L_1	L_0	L_4	L_7	L_2	L_6	L_5	L_3
-------	-------	-------	-------	-------	-------	-------	-------

...
-----	-----	-----	-----	-----	-----	-----	-----

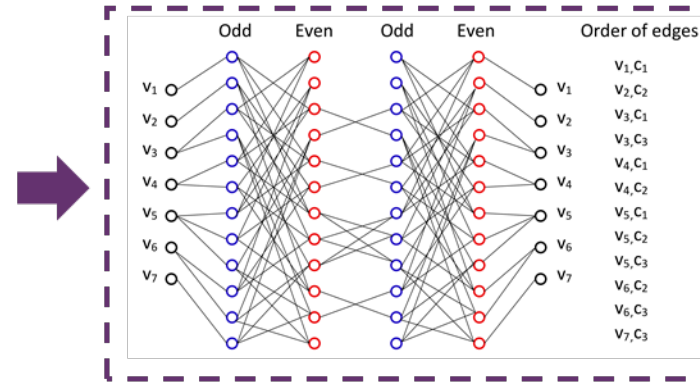
$\ell = 7$

L_6	L_5	L_7	L_4	L_3	L_1	L_0	L_2
-------	-------	-------	-------	-------	-------	-------	-------

List Decoding Procedure

Step 3: Decode $(L_1^{(i)}, L_2^{(i)}, \dots, L_n^{(i)})$ using our neural decoder

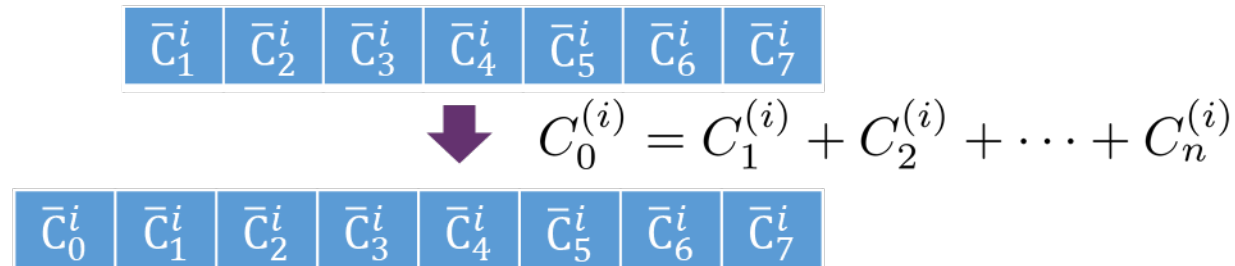
$\ell = 1$	L0	L1	L2	L3	L4	L5	L6	L7
$\ell = 2$	L1	L0	L4	L7	L2	L6	L5	L3
...
$\ell = 7$	L6	L5	L7	L4	L3	L1	L0	L2



\bar{C}_1^1	\bar{C}_2^1	\bar{C}_3^1	\bar{C}_4^1	\bar{C}_5^1	\bar{C}_6^1	\bar{C}_7^1
\bar{C}_1^2	\bar{C}_2^2	\bar{C}_3^2	\bar{C}_4^2	\bar{C}_5^2	\bar{C}_6^2	\bar{C}_7^2
...
\bar{C}_1^7	\bar{C}_2^7	\bar{C}_3^7	\bar{C}_4^7	\bar{C}_5^7	\bar{C}_6^7	\bar{C}_7^7

Step 4: Check whether $(\bar{C}_1^{(i)}, \bar{C}_2^{(i)}, \dots, \bar{C}_n^{(i)})$ is a codeword or not

Step 5: Prepend an overall parity to $(\bar{C}_1^{(i)}, \dots, \bar{C}_n^{(i)})$



List Decoding Procedure

Step 6: Apply the inverse permutation σ_i^{-1} to $(\bar{C}_0^{(i)}, \bar{C}_1^{(i)}, \dots, \bar{C}_n^{(i)})$

σ_i^{-1} Inverse permutations

\bar{C}_0^1	\bar{C}_1^1	\bar{C}_2^1	\bar{C}_3^1	\bar{C}_4^1	\bar{C}_5^1	\bar{C}_6^1	\bar{C}_7^1
\bar{C}_0^2	\bar{C}_1^2	\bar{C}_2^2	\bar{C}_3^2	\bar{C}_4^2	\bar{C}_5^2	\bar{C}_6^2	\bar{C}_7^2
...
\bar{C}_0^7	\bar{C}_1^7	\bar{C}_2^7	\bar{C}_3^7	\bar{C}_4^7	\bar{C}_5^7	\bar{C}_6^7	\bar{C}_7^7



0	1	2	3	4	5	6	7
1	0	4	7	2	6	5	3
2	4	0	5	1	3	7	6
4	2	1	6	0	7	3	5
3	7	5	0	6	2	4	1
7	3	6	1	5	4	2	0
5	6	3	2	7	0	1	4
6	5	7	4	3	1	0	2



\bar{C}_0^1	\bar{C}_1^1	\bar{C}_2^1	\bar{C}_3^1	\bar{C}_4^1	\bar{C}_5^1	\bar{C}_6^1	\bar{C}_7^1
\bar{C}_1^2	\bar{C}_0^2	\bar{C}_4^2	\bar{C}_7^2	\bar{C}_2^2	\bar{C}_6^2	\bar{C}_5^2	\bar{C}_3^2

\bar{C}_6^7	\bar{C}_5^7	\bar{C}_7^7	\bar{C}_4^7	\bar{C}_3^7	\bar{C}_1^7	\bar{C}_0^7	\bar{C}_2^7

Step 7: ML decoding among the ℓ candidates $(\hat{C}_0^{(i)}, \hat{C}_1^{(i)}, \dots, \hat{C}_n^{(i)})$, $i \in [\ell]$

\hat{C}_0^i	\hat{C}_1^i	\hat{C}_2^i	\hat{C}_3^i	\hat{C}_4^i	\hat{C}_5^i	\hat{C}_6^i	\hat{C}_n^i
---------------	---------------	---------------	---------------	---------------	---------------	---------------	---------------

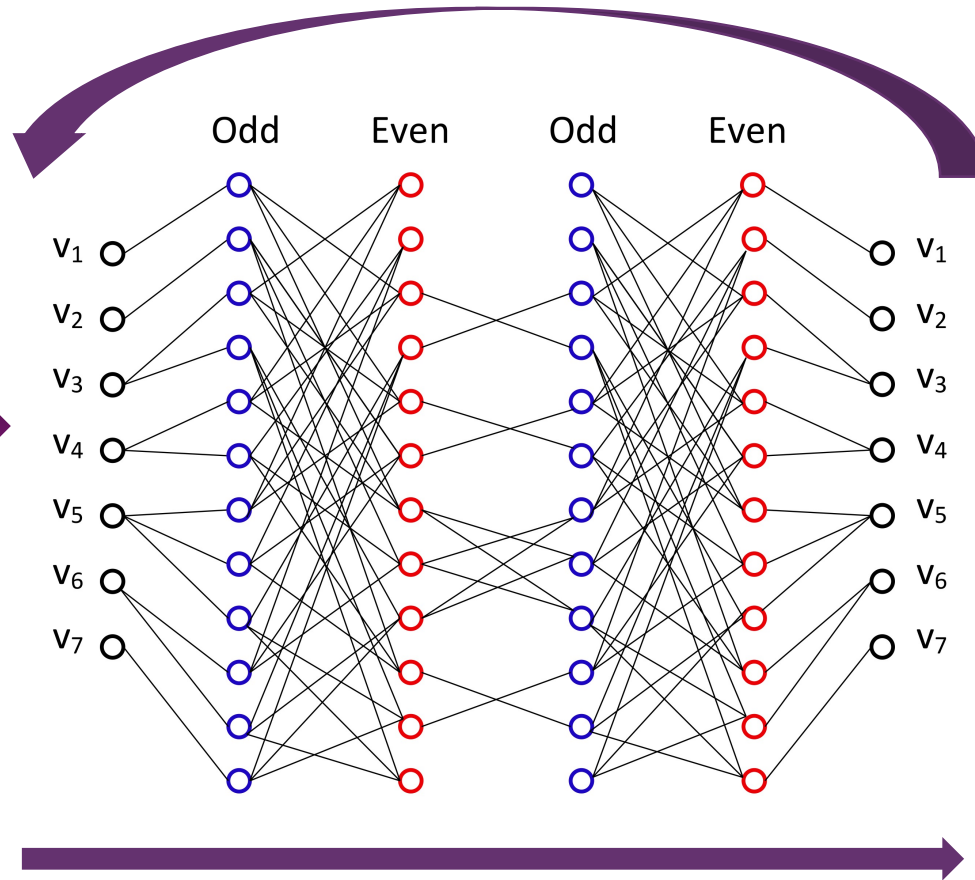
Step 8: Discard the first bit \hat{C}_0



\hat{C}_1^i	\hat{C}_2^i	\hat{C}_3^i	\hat{C}_4^i	\hat{C}_5^i	\hat{C}_6^i	\hat{C}_n^i
---------------	---------------	---------------	---------------	---------------	---------------	---------------

Boosting Method

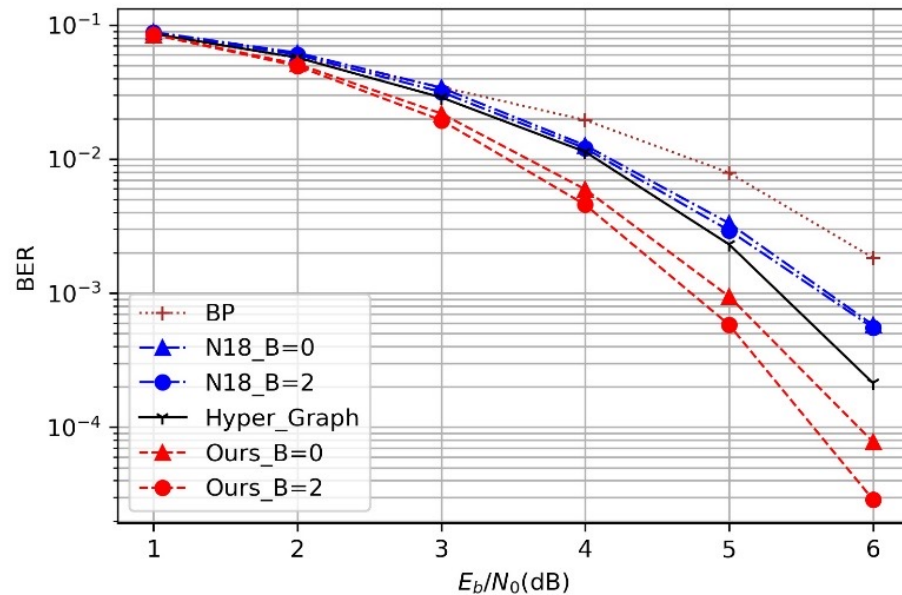
$$L_j = \log \frac{\mathbb{P}(y_j | C_j = 0)}{\mathbb{P}(y_j | C_j = 1)}$$



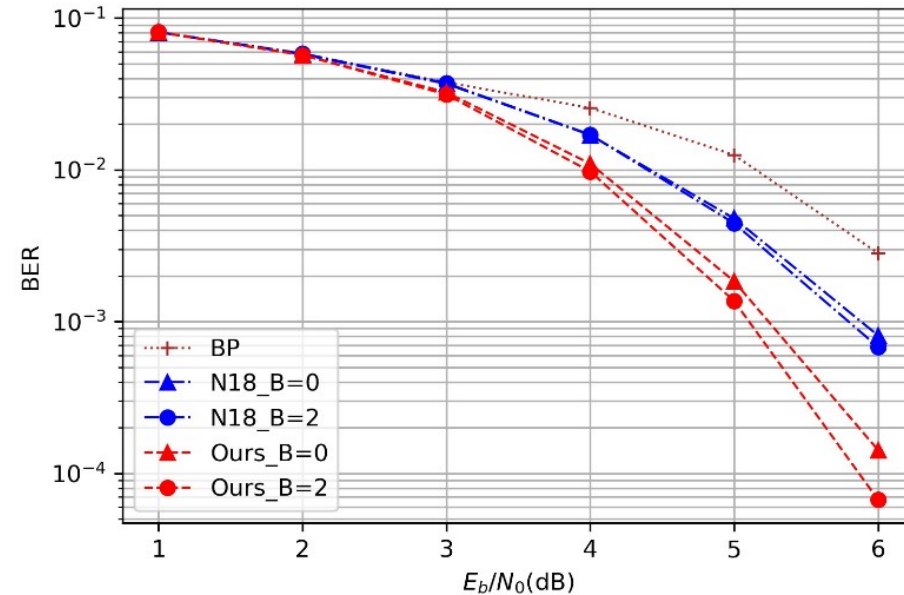
$$L_j = \log \frac{\mathbb{P}(y_j | C_j = 0)}{\mathbb{P}(y_j | C_j = 1)}$$

Cyclic decoder Results

BCH (63,45)



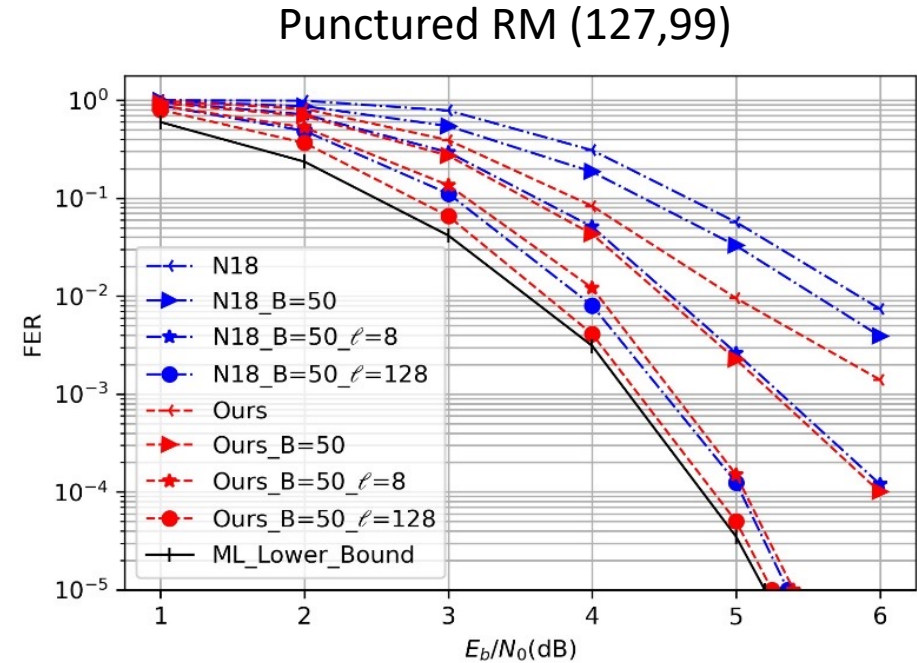
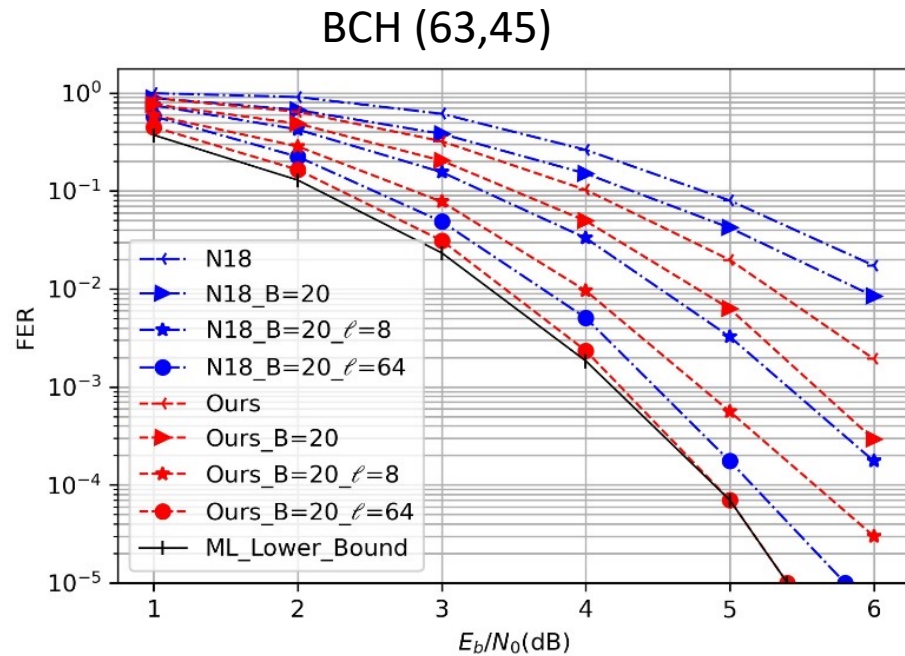
BCH (127,99)



Improve upon N18 (Nachmani et al., 2018) by **0.7dB**

Improve upon the hyper-graph-network decoder (Nachmani et al., 2019) by **0.3dB**

List Decoding Results



List size $n+1$ gives **3dB gain**, almost the same performance as the **ML decoder**
Small list size $l=8$ also gives **0.7~0.9dB gain**

Complexity Analysis

Decoding time of N18 (Nachmani18) and our decoder

Code	N18	Ours	N18/Ours
BCH(63,36)	4.09ms	4.68ms	0.87
BCH(63,45)	3.85ms	4.94ms	0.78
BCH(127,64)	8.60ms	8.63ms	0.99
BCH(127,99)	4.05ms	16.7ms	0.24
PRM(127,99)	3.97ms	7.45ms	0.53

The memory requirements of our decoder is much smaller than N18 due to the cyclically invariant structure of weights

Ablation analysis

Ablation analysis. The numbers are negative natural logarithm of BER.

Code	BCH(63,24)			BCH(63,36)			BCH(63,45)			PRM(63,22)		
	4	5	6	4	5	6	4	5	6	4	5	6
N18, $(n - k) \times n$ parity matrix	3.24	4.36	5.80	3.97	5.27	7.05	4.37	5.71	7.45	2.85	3.76	4.98
N18, randomly extended $n \times n$ matrix	3.27	4.38	5.78	3.97	5.28	7.07	4.49	5.81	7.47	2.90	3.86	5.17
N18, $n \times n$ cyclic matrix	3.67	5.08	6.78	4.42	5.93	7.84	4.87	6.19	7.76	3.15	4.33	5.86
Ours, $n \times n$ cyclic matrix	3.78	5.11	7.07	4.63	6.48	8.86	5.12	6.97	9.46	3.32	4.52	6.23

When both using the $n \times n$ cyclic matrix, our decoder still demonstrates a **0.25dB** improvement over N18

Conclusions

A novel neural decoder that is provably equivariant to cyclic shifts

- Improve upon (Nachmani et al., 2018) by **0.7dB**
- Improve upon the hyper-graph-network decoder (Nachmani & Wolf, 2019) by **0.3dB** with **300 times smaller training time.**

A list decoding procedure that provides up to **3dB** gain.

- For certain high-rate codes, our list decoder has **almost the same performance as the ML decoder**