Hyper-Graph-Network Decoders for Block Codes

Eliya Nachmani, Lior Wolf

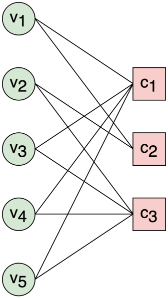
Facebook AI Research, Tel Aviv University

October 24, 2019

Overview

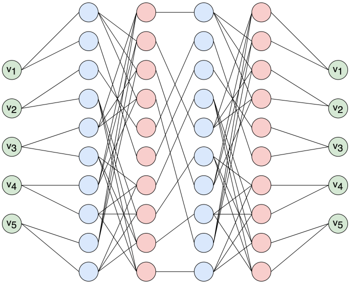
Tanner Graph

- Block Code with a block size of n bits, and k information bits
- Binary parity check matrix H
- The parity check matrix entails a Tanner graph, which has n variable nodes and $(n-k)$ check nodes:



Trellis Graph

- The Tanner graph is unrolled into a Trellis graph:
  - Starts with n variable nodes
  - Composed of two types of columns, variable columns (blue) and check columns (red)
  - Ends with an output layer of n variable nodes



Belief Propagation Decoding

- Message passing algorithms operate on the Trellis graph
- The messages propagate from variable columns to check columns and from check columns to variable columns, in an iterative manner
- The leftmost layer corresponds to a vector of log likelihood ratios (LLR) $l \in \mathbb{R}^n$ of the input bits:

$$l_v = \log \frac{\Pr(c_v = 1 | y_v)}{\Pr(c_v = 0 | y_v)},$$

- Let x^j be the vector of messages that a column in the Trellis graph propagates to the next column

Neural Belief Propagation Decoding

- Neural belief propagation define in Nachmani et al. [1]
- For odd j:

$$x_e^j = x_{(c,v)}^j = \tanh\left(\frac{1}{2}\left(l_v + \sum_{e' \in N(v) \setminus \{(c,v)\}} w_{e'} x_{e'}^{j-1}\right)\right)$$

- For even j:

$$x_e^j = x_{(c,v)}^j = 2arctanh\left(\prod_{e' \in N(c) \setminus \{(c,v)\}} x_{e'}^{j-1}\right)$$

- The vth bit output at layer $2L+1$:

$$o_v = \sigma\left(l_v + \sum_{e' \in N(v)} \bar{w}_{e'} x_{e'}^{2L}\right)$$

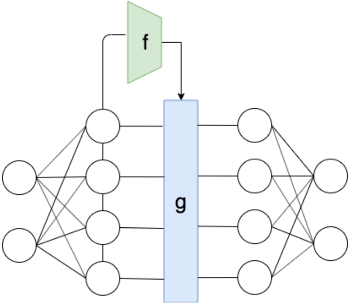
Neural Belief Propagation Decoding

Where:

- w_e and $\bar{w}_{e'}$ are sets of learned weights
- The w_e learned weights are shared across all iterations j
- The network has L layers of variable and check nodes

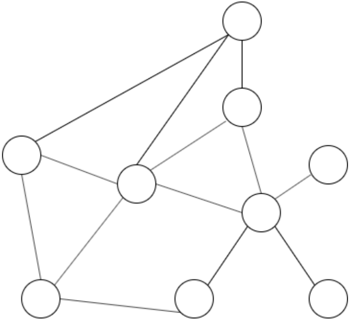
Hypernetwork

- A neural architecture that has adaptive capabilities
- A network f is trained to predict the weights θ_g of another network g:



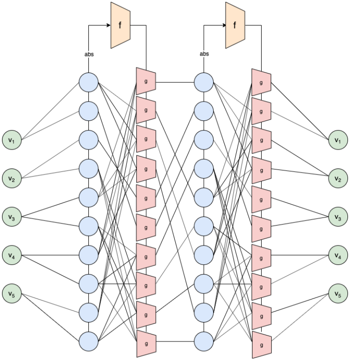
Graph Neural Network

- A deep learning architecture that operate on graphs structure
- Every node in the graph is a neural network
- The connection between nodes obtained from the graph adjacency matrix:



Hyper-Graph-Network Decoder

- We suggest adding learned components:
  - Graph neural network - replace each variable neuron with neural network
  - Hypernetwork - adding network f to predict the weights of the variables nodes network



Hyper-Graph-Network Decoder

- Replace odd j equation with the following equations:

$$x_e^j = x_{(c,v)}^j = g(l_v, x_{N(v,c)}^{j-1}, \theta_g^j)$$

$$\theta_g^j = f(|x^{j-1}|, \theta_f)$$

- θ_g^j is the weights of network g at iteration j. θ_f are the learned weights of network f
- The absolute value of the message can be seen as measure for the correctness and the sign corresponding bit value
- In order to focus on the correctness of the message and not the information bits, the input to f is in absolute value

Hyper-Graph-Network Decoder

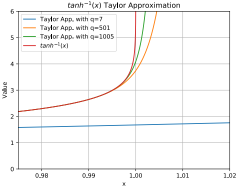
- The architecture of both f and g does not contain bias terms and employs the *tanh* activations
- The network g has p layers, i.e., $\theta_g = (W_1, ..., W_p)$, for some weight matrices W_i
- The network f ends with p linear projections, each corresponding to one of the layers of network g

Hyper-Graph-Network Decoder

- In order to regularize training, we replace the *arctanh* in the updating equation of even j with Taylor approximation:

$$x_e^j = x_{(c,v)}^j = 2\sum_{m=0}^{q} \frac{1}{2m+1} \left(\prod_{e' \in N(c) \setminus \{(c,v)\}} x_{e'}^{j-1} \right)^{2m+1}$$

- Where q is the Taylor approximation of degree q
- The *arctanh* activation, has asymptotes in $x = 1, -1$, and training with it often explodes. Its Taylor approximation is a well-behaved polynomial:



Hyper-Graph-Network Decoder - Training

- We consider the following marginalization for each iteration where j is odd:

$$o_v^j = \sigma\left(l_v + \sum_{e' \in N(v)} \bar{w}_{e'} x_{e'}^j\right)$$

- We employ the cross entropy loss function, which considers the error after every check node iteration out of the L iterations:

$$\mathcal{L} = -\frac{1}{n}\sum_{h=0}^{L}\sum_{v=1}^{n} c_v \log(o_v^{2h+1}) + (1 - c_v)\log(1 - o_v^{2h+1})$$

- Where c_v is the ground truth bit

Hyper-Graph-Network Decoder - Symmetry Conditions

- For block codes that maintain certain symmetry conditions, the decoding error is independent of the transmitted codeword
- A direct implication is that we can train our network to decode only the zero codeword
- There are two symmetry conditions:
  - $\Phi\left(b^\top x^{j-1}_{N(v,c)}\right) = \left(\prod_1^K b_k\right) \Phi\left(x^{j-1}_{N(v,c)}\right)$
  - $\Psi\left(-l_v, -x^{j-1}_{N(v,c)}\right) = -\Psi\left(l_v, x^{j-1}_{N(v,c)}\right)$
  - where Φ the check node function and Ψ is the variable node function
- Our method, by design, maintains the symmetry condition on both the variable and the check nodes. See Lemma 1 and 2 in our paper.

Experiments

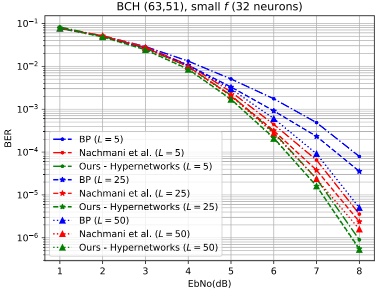
- We train the proposed architecture with three classes of linear block codes: Low Density Parity Check (LDPC) codes, Polar codes and Bose–Chaudhuri–Hocquenghem (BCH) codes
- Training examples are generated as a zero codeword transmitted over an additive white Gaussian noise
- The learning rate was $1e-4$ for all type of codes
- The decoding network has ten layers which simulates $L = 5$ iterations of a modified BP algorithm

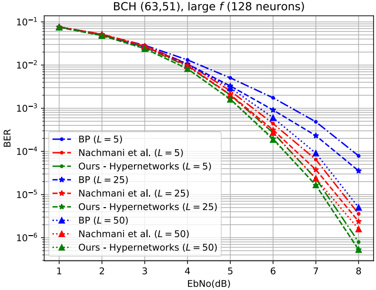
Experiments

- Each training batch contains examples with different Signal-To-Noise (SNR) values
- The order of the Taylor series of *arctanh* is set to $q = 1005$
- The network f has four layers with 32 neurons at each layer. The network g has two layer with 16 neurons at each layer
- For BCH codes, we also tested a deeper configuration in which the network f has four layers with 128 neurons at each layer

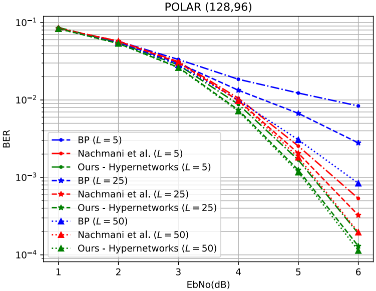
Results

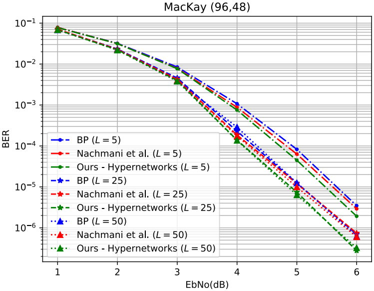
- We present the BER for BCH(63,51) with small and large f. As can be seen, we achieve improvements of 0.45dB, 0.43dB respectively:





- We present the BER for Polar(128,96) and LDPC MacKay(96,48). As can be seen, we achieve improvements of 0.48dB, 0.15dB respectively:





Ablation Analysis

- To evaluate the contribution of the various components of our method, we ran an ablation analysis:
    1. our complete method
    2. a method in which the parameters of g are fixed and g receives and additional input of $|x^{j-1}|$
    3. a similar method where the number of hidden units in g was increased to have the same amount of parameters of f and g combined
    4. a method in which f receives the x^{j-1} instead of the absolute value of it
    5. a variant of our method in which *arctanh* replaces its Taylor approximation
    6. a similar method to the previous one, in which gradient clipping is used to prevent explosion

Ablation Analysis

The negative natural logarithm of BER results of our complete method are compared with alternative methods. Higher is better.

Code	BCH (31,16)		BCH (63,51)	
Variant/SNR	4	6	4	6
(1) Complete method	4.96	8.80	4.67	8.22
(2) No hypernetwork	2.94	3.85	3.83	5.18
(3) No hypernetwork, higher capacity	2.94	3.85	3.83	5.18
(4) No abs in f input	2.86	3.99	3.84	5.20
(5) Not truncating *arctanh*	0.69	0.69	0.69	0.69
(6) Gradient clipping	0.69	0.69	0.69	0.69
[1]	4.74	8.00	4.54	7.73
[1] with truncated *arctanh*	4.78	8.24	4.53	7.84

Ablation Analysis

- The results, demonstrate the advantage of our complete method
- We can observe that without hypernetwork and without the absolute value in f input, the results degrade below those of [1]
- For (2), (3) and (4) the method reaches the same low quality performance
- For (5) and (6), the training process explodes and the performance is equal to a random guess

Conclusions

- We presents graph networks decoder in which the weights are a function of the node's input
- We present a method to avoid gradient explosion
- By carefully designing our networks, important symmetry conditions are met and we can train efficiently
- Our method introduce a new learnable component to neural decoders
- Our results go far beyond the current literature on learning block codes and we present results for a large number of codes from multiple code families

References

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)*. IEEE, 2016, pp. 341–346.