「pruning neural belief propagation decoders」

↳ tailor an overcomplete parity-check matrix to BP decoding using machine learning.

①
weight = indication of importance of the connected check nodes (CNs) to decoding.

② use them to prune unimportant CNs.

↳ 각 iteration 마다 서로 다른 parity-check matrix 사용.

---

ML (maximum likelihood decoding)

⇒ achieving near ML performance

= computationally complex.

↓

sol: BP decoding.

LDPC + sparse parity check matrix.

⇒ BP decoding ≃ near optimal performance.

WBP「weighted belief propagation」=

① In each iteration, the VNs and CNs are now referred to as VN layers and CN layers, respectively.

② weights can be introduced at the edges which then are optimized using SGD.

③ each edge has a different weight

* while WBP decoding improves upon conventional BP decoding, its performance is still limited by the underlying parity-check matrix.

↳ we introduce a pruning-based approach to selecting the best parity-check equations for each iteration of the BP decoder for short linear block codes.

① starts with a large overcomplete parity-check matrix under WBP decoding.

② consider the weights in the Tanner graph, the magnitude of the weights gives an indication of the importance of the edge in the decoding process.

↳ A magnitude close to zero indicate that edge has low importance.

• By tying the weights for each CN, 「enforcing that the weights of all incoming edges to a single CN are equal」, the weights can be interpreted as an indication of importance of the CN in the decoding process.

CNs with connected low-weight edges do not play an important role in the decoding process and can be removed.

↓

by allowing pruning of different CNs in each iteration, the optimization results in a different parity-check equation for each iteration.

「각 iteration에서 다른 CN에 pruning을 적용함으로써, 각 iteration 마다 다른 parity check equation이 최적화된 결과로 나온다」

* The weights in the corresponding Tanner graph can be tied, leading to the largest complexity.

---

$$\lambda_{c\to v}^{(l)} = 2\cdot w_c^{(l)} \cdot \tanh^{-1}\left(\prod_{\tilde{v}\in N(c)\setminus v} \tanh\left(\frac{1}{2}\lambda_{\tilde{v}\to c}^{(l)}\right)\right)$$

↓

modified wBP: weights are tied at the CNs 「same weight $w_c^{(l)}$」

- $w_c^{(l)}$의 크기 = how much the CN contributes to decoding.
- A large magnitude indicates high importance whereas a magnitude of zero indicates that the CN is irrelvant to the decoding process.

1. Initialize 「$l$'iteration의 PCM」

$H = \{H_1, H_2, \cdots, H_L\}$, $w = \{1, 1, \cdots, 1\}$ ← weight

2. Run one batch of gradient descent to optimize $w$.

3. converged or max. number of batches? — No ↑

↓ Yes

4. Find the min. check node weight and remove the check node

↓

5. Target complexity or performance reached? → No : (2)

↓

Yes.

---

decoder $D_1 = H_{opt}, w_{opt}$

$D_2 = H_{opt}$, 'all weight = 1'

$D_3 = H_{opt}$, untied optimized weights over all iterations.

---

The weights in $w$ are then optimized using the Adam optimizer within the Tensorflow programing framework.

After the optimization has converged, we find the index & iteration of lowest CN weight $w_c^{(l)}$ and set it to zero.

As this may change the optimal value for the remaining weights, we rerun the training.

neural decoding with optimization of node activations.

↳ Abstract= two novel loss terms on the node's activations. (노드 활성화에 대한 새로운 loss terms이 제시되었다.

① first loss term = sparse constraint on the node's activations).

② second loss term tried to mimic the node's activations from a teacher decoder( which has better performance),

I. introduction

The neural belief propagation(NBP) was the first deep neural decoder that 「NBP > Vanilla BP decoder」 provide an improvement over the vanilla belief propagation decoder.

이전 연구 ① = to find better neural ☆architecture to improve the decoder's performance.

② tried to change the loss function to improve the performance.

⇒ Learning from the syndrome = penalizes the neural decoder for producing outputs that do not correspond to valid codewords. 「the loss function at the output layer」. 「추가 복잡도 없이 BER 향상」.

III. Method

A) the Architecture

B) The knowledge distillation loss term

C) The sparse node activation Loss Term.

IV. results

「Pruning neural belief propagation decoders」

↳ remove irrelevant check nodes.

③ finding better sparse parity check matrix.

↓

① list-based decimation stage.
② learned decimation stage.

weight are tied over iterations.

$\lceil w^{(l)}_{ch,V_i} = w_{ch,V_i},\ w^{(l)}_{V_i \to C_j} = w_{V_i \to C_j} \rfloor$.

NBP-D($l_{max}$, $n_D$, $n_{LD}$)
① ②

☆ iterate between decimation process and conventional NBP decoder.

① list-based decimation.

identify VN 'v' with the lowest absolute a posteriori LLR $\mu_v^{(l_{max})}$

∴ least reliable LLR ($2^{n_D}$)

$\pm\infty$ : decimate.
$= \mu_{ch,v}$

build decoding tree.
sign 값 알수X

$\begin{bmatrix} +\infty \\ -\infty \end{bmatrix}$ two new graph.

② learned decimation stage.
↳ ① = 성능 ↑, 복잡도 ↑↑

→ use an NN to decide to which value each VN should be decimated.

a posteriori LLR in $l$ th iteration

$$\mu_{V_i}^{(l)} = \mu_{ch,V_i} + \sum_{c \in N(V_i)} \mu_{C \to V_i}^{(l)}$$

$$\mu_{ch,v} = \mu_{ch,v} + \mathrm{sign}(\mu_v^{(l_{max})})$$

$$\cdot\ \left| f_{NN}(\mu_{ch,v}, \{\mu_{c \to v}^{(l_{max})} \mid c \in N(v)\}, \theta) \right|$$

all trainable parameters of the NN.
모든 VN에
↳ 같은 NN이 적용되고 weight는 shared.

---

~~two-stage~~ decimation

① we build a list by iterating between a conventional NBP decoder
「가장 신뢰성 낮은 bit를 추출하고, 기존의 NBP decoder 사이에서 list를 받으며」
and guessing the least reliable bit.

② learned decimation = 어떤 값을 decimation할지 결정.
↳ The second stage iterates between a conventional NBP decoder and learned decimation, where we use a neural network to decide the decimation value for each bit.

☆ a posteriori LLR.