Training Iterative Decoders using Deep Learning Approaches

Eran Sharon, Ariel Navon
March 2018
LDPC codes and Iterative Decoders

Sparse bipartite graph

Variable nodes

Check nodes

$d_v$ – variable node degree = the number of parity checks in which the bit participates

$d_c$ – check node degree = the number of bits which participate in the parity check

Belief Propagation (BP) Decoding:

Initial bit estimate from Flash

Bit estimates from parity check-nodes

Combining independent estimates of bit $v$

Decoder final estimate of bit $v$

"Soft" XOR operation

$Q = P_v + \sum_{c_j \in N(v, G)} R_{j,v}$
Background – Sub optimality of iterative decoders

• Long LDPC codes can approach the Shannon limit
• Long code → Latency and Complexity penalties 😞
• Short codes are required for certain applications, such as SCM
• Conventional iterative coding schemes perform poorly for short codes:
  • BP computation rules assume statistical independence of messages
    → Assume cycle free computation tree (holds asymptotically...)
  • Cannot avoid many short cycles in the graph of short codes
  • Impair the statistical independence assumption → sub-optimal decoding
Training Iterative Decoders using Deep Learning

- Improved iterative decoder for short codes is proposed

- Modifying the standard message-passing computation rules, according to the specific statistical dependencies (induced by cycles in the code’s graph) of certain code

- Use Deep Learning methods for training an iterative decoder on a specific code - similar ideas suggested in [1,2]

- Iterative decoding fits very well to the Deep Learning framework:
  - **Net Structure**: Iterative decoder computations along several iterations can be described in terms of a Deep-Neural-Net (DNN)
  - **“Big Data”**: Unlimited labeled data generated via an endless Monte-Carlo simulation
    - Noisy codewords serve as network inputs
    - Clean codewords serve as labeled network outputs

Deep Learning Network for Improved Belief Propagation

- Representation BP process as Neural Net
- Optimizing net weights is equivalent to learning the coefficients of improved BP rules
Deep Learning Network for Improved Belief Propagation

- Representation BP process as Neural Net
- Optimizing net weights is equivalent to learning the coefficients of improved BP rules

Standard BP computation rules:

\[ Q_{vc}^j = P_v + \sum_{c' \in N(v) \setminus c} R_{cv}^j \]

\[ R_{cv}^{j+1} = 2 \cdot \tanh^{-1} \left( \prod_{v' \in N(c) \setminus v} \tanh \left( \frac{Q_{v'c}^j}{2} \right) \right) \]

Modified BP computation rules:

\[ Q_{vc}^j = P_v + \sum_{c' \in N(v) \setminus c} \alpha_{c',v,c}^j \cdot R_{c'v}^j \]

\[ R_{cv}^{j+1} = 2 \cdot \tanh^{-1} \left( \prod_{v' \in N(c) \setminus v} \left( \tanh \left( \frac{Q_{v'c}^j}{2} \right) \right)^{\beta_{v',v,c}^j} \right) \]
Training Procedure for Improved Belief Propagation

Generate a batch of noisy codewords – for each one perform:

**Forward pass –**
- **Message passing decoding:** converts a-priori LLR inputs \( P \) to a-posteriori LLR outputs \( Q \)
- Clean codeword serves as “ground-truth” \( b \)
- **Loss-Function:** Mutual Information (MI) between \( Q \) and \( b \): \( I(b, Q) = \sum_v 1 - \log(1 + 2^{-Q_v \cdot b_v}) \)

**Backward pass -**
- **Backward message passing** of partial derivatives
- Simultaneously computes gradients for all the network parameters via back propagation
- Gradients indicate how changing each of the parameters affects the MI loss function

**Update Network Parameters:**
- Adjust \( \alpha, \beta \) parameters of the computation rules according to averaged gradients over the batch to maximize \( I(b, Q) \)
  (Stochastic Gradient Descent)
Training Procedure for Improved Belief Propagation

Standard BP computation rule:
\[ Q_v^i = P_v^i + \sum_{c \in \text{check nodes}} R_v^c \]
Modified BP computation rule:
\[ Q_v^i = P_v^i + \sum_{c \in \text{check nodes}} R_v^c \cdot R_v^i \]

\( b \in \{-1, 1\} \)
Add noise and normalization
A-priori LLRs

Check nodes

\( \frac{dQ_v^i}{d\alpha_{v,c}^i} = R_v^i \)
\( \frac{dQ_v^i}{d\alpha_{v,c}^i} = R_v^i \)
\( \frac{dQ_v^i}{d\beta_{v,c}^i} = \frac{dR_v^i}{d\alpha_{v,c}^i} \cdot \frac{dQ_v^i}{dR_v^i} \cdot \frac{dR_v^i}{dQ_v^i} \)

\( \frac{dR_v^i}{dQ_v^i} = \frac{1}{\text{h}(Q_v^i / 2)} \cdot \left( \frac{\tanh(Q_v^i / 2)}{\tanh(Q_v^i / 2)} \right)^2 \)

Output of the network:
\( I(b, Q) = \sum 1 - \log_2(1 + 2^{-Q \cdot b}) \)
("Labeled" output as it is a function of \( b \))
Training Procedure for Improved Belief Propagation

Non standard DL problem – Hard to use existing DL packages...

→ Developed tailored DL package for iterative decoders from scratch (Math + Coding)
Parametric Computation Rule Models

• There may be various options for parametric computation rules.
• E.g. for the variable-to-check computation rule, we may use: $Q_{vc}^j = P_v + \sum_{c \in N(v) \setminus c} \alpha_{crv,v,c}^j \cdot R_{crv}^j$

• Alternatively, we may use a more involved parametric computation rule:

$Q_{vc} = P_v + \sum_{c \in N(v) \setminus c} \left( \alpha_{crv,v,c,1} + \alpha_{crv,v,c,2} \cdot e^{-\alpha_{crv,v,c,3} \cdot |\delta(R_{cv'},c)|} \right) \cdot R_{crv}$, \hspace{1cm} \text{where} \hspace{1cm} R_{cv} = \{R_{crv}, c' \in N(v) \setminus c\}$

and $\delta(R_{cv'}, c') = R_{crv} - \frac{1}{|N(v) \setminus c|} \sum_{c'' \in N(v) \setminus c} R_{crv''}$

In this case, higher “damping” factor is applied to a message if it resembles its accompanying messages

\textbf{Intuition}: message resemblance indicate stronger statistical dependencies between the messages

• What is a good parametric model?
  • A model that will allow the modified BP to approach MAP decoding
• How can we come up with such a model?
  • A parametric model may be determined by analyzing a “Toy” graph
    • Simple enough for computing the expression for optimal MAP decoding
    • Model derived as the one which allows for the best fit between the Modified-BP and MAP decoding
Toy Example – Case Study

- Choose a toy graph for which computing the MAP decoder is feasible
- Compare the MAP decoder to the Modified BP decoder in order to determine what parametric model can be used to approximate MAP

\[ H = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \]

\[ C^\perp = \begin{cases} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{cases} \]

(codebook of the dual code used for computing the MAP expression based on [1])

\[ T_j = \tanh \left( \frac{P_j}{2} \right) \]

\[ Q_{0,BP} = P_0 + R_{1,1} + R_{2,1} = P_0 + \log \frac{1 + T_1 \cdot T_3}{1 - T_1 \cdot T_3} + \log \frac{1 + T_2 \cdot T_3}{1 - T_2 \cdot T_3} = P_0 + \log \frac{1 + T_1 \cdot T_3 + T_2 \cdot T_3 + T_1 \cdot T_2 \cdot T_3^2}{1 - T_1 \cdot T_3 - T_2 \cdot T_3 + T_1 \cdot T_2 \cdot T_3^2} \]

\[ Q_{0,MAP} = P_0 + \log \frac{1 + T_1 \cdot T_3 + T_2 \cdot T_3 + T_1 \cdot T_2}{1 - T_1 \cdot T_3 - T_2 \cdot T_3 + T_1 \cdot T_2} \]

(computed based on dual codebook [1])

Assume: \[ Q_{0,MAP} = Q_{0,ModifiedBP} = P_0 + R_{1,1} + \alpha \cdot R_{2,1} \]

\[ \Rightarrow \alpha = \frac{\log \frac{1 + T_1 \cdot T_3}{1 - T_1 \cdot T_3}}{ \log \frac{1 + T_2 \cdot T_3}{1 - T_2 \cdot T_3}} \]

Toy Example – Case Study

The following figure shows $\alpha$ as a function of $|\delta| = |R_{2,1} - (R_{2,1} + R_{1,1})/2|$ for random selections of $P_0, P_1, P_2$, and $P_3$.

This indicates that a parametric model such as $\alpha(|\delta|) = \alpha_1 + \alpha_2 \cdot e^{-\alpha_3 \cdot |\delta|}$ may allow the modified BP to better approximate MAP decoding, by deriving the parameters of the model through the proposed deep learning approach.
Training results

Training a length 512 code over the **same batch** over and over again – leads to “over fitting” - near perfect correction of the specific “noise” samples of the batch (but poor correction of other “noise” samples)
Training results

Training a length 512 code over different batches:
Simulation results

![Graph showing BER vs. Eb/No with Conventional BP and Modified BP through Deep Learning. The Modified BP shows a performance improvement of approximately 0.5 dB compared to the Conventional BP.](image)
Conventional iterative decoders perform poorly for short codes.

**Deep Learning** approach proposed for *training an iterative decoder*
- learning complex statistical dependencies in the code’s graph
  - Iterative decoder computations along several iterations is described as DNN
  - Unlimited labeled data generated via endless Monte-Carlo simulation – “Big Data”
- Improved correction capability demonstrated for short codes

**Summary - Training Iterative Decoders using DL**

- Future work includes decoder **On-Line training** – decoder improves performance over memory lifetime by adapting to the observed host traffic and/or NAND noises – complements the memory characteristic degradation during memory lifetime
Thank you!

Questions?

Contact:
eran.sharon@wdc.com