SPCOM 2020:
Deep Learning for Communication Algorithms

Sreeram Kannan

Hyeji Kim

Presenting Team

Sreeram Kannan

Hyeji Kim

Thanks:

Yihan Jiang

Pramod Viswanath

Sewoong Oh

Himanshu Asnani

Success of Deep Learning

Speech

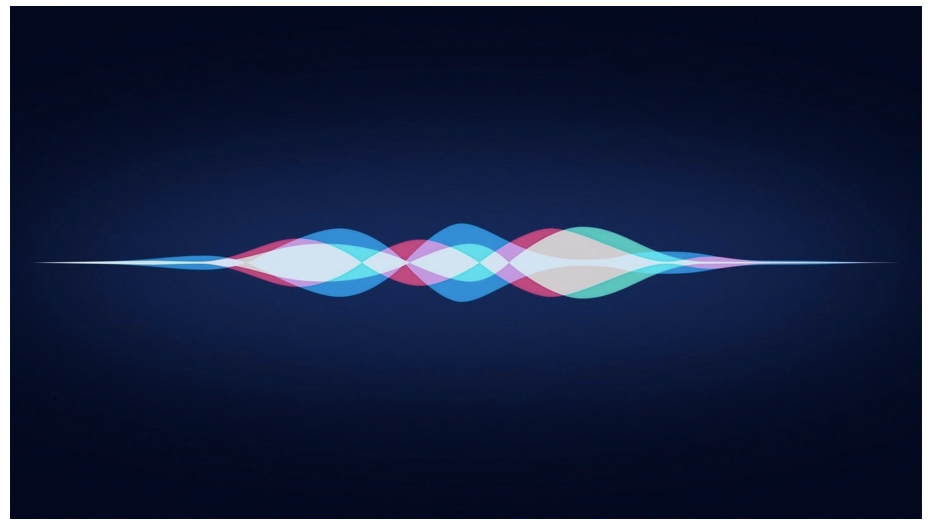
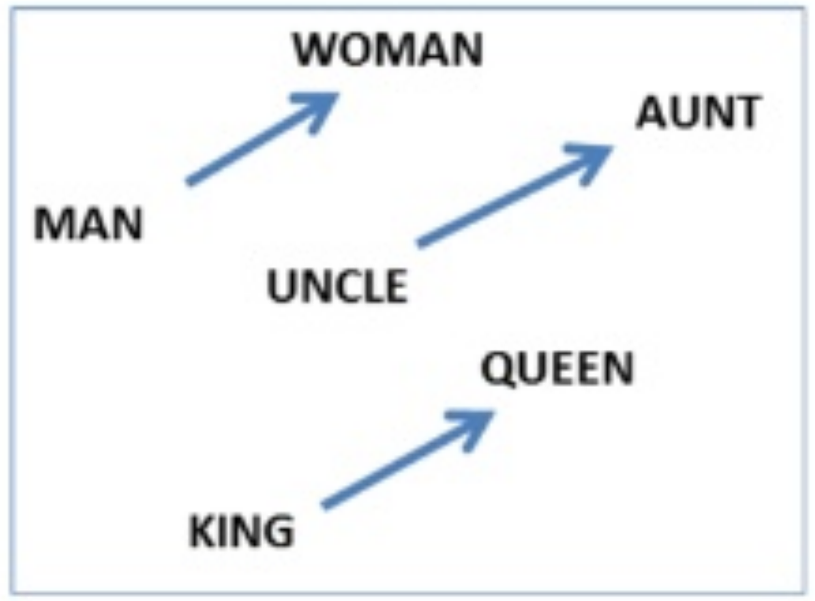
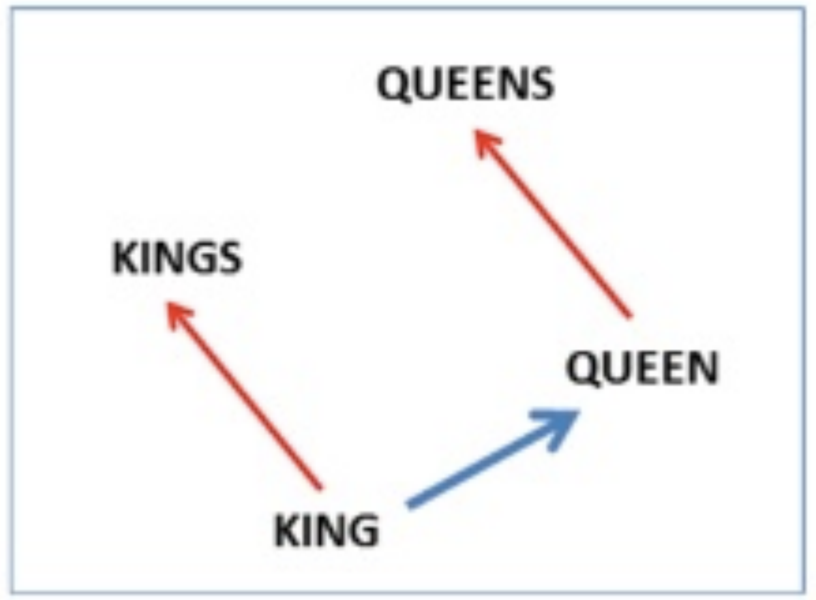


Image recognition

"construction worker in orange safety vest is working on road."

NLP



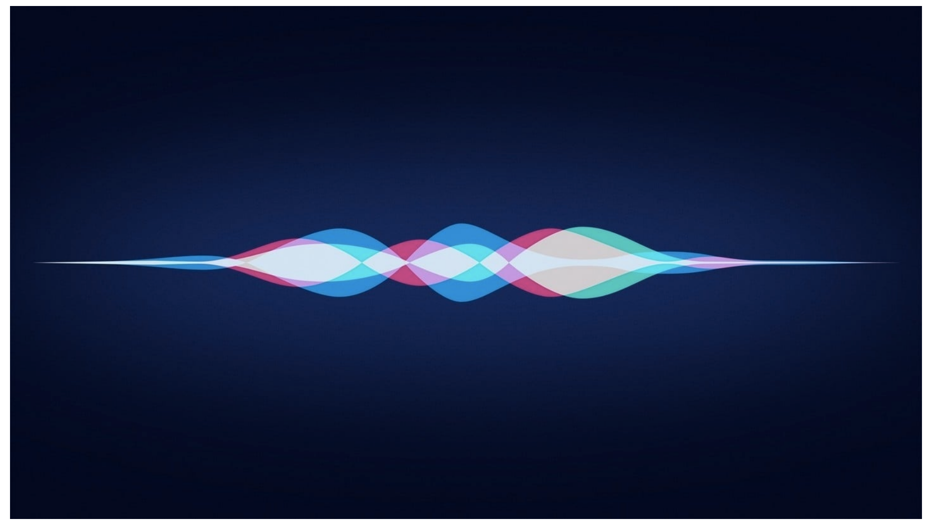


Video

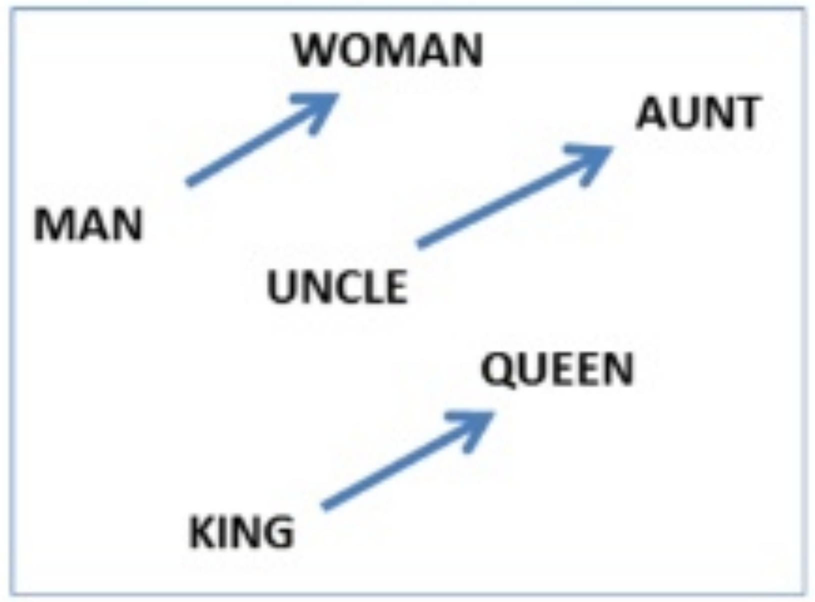
https://www.youtube.com/watch?v=9Yq67CjDqvw

Success of Deep Learning

Speech



NLP



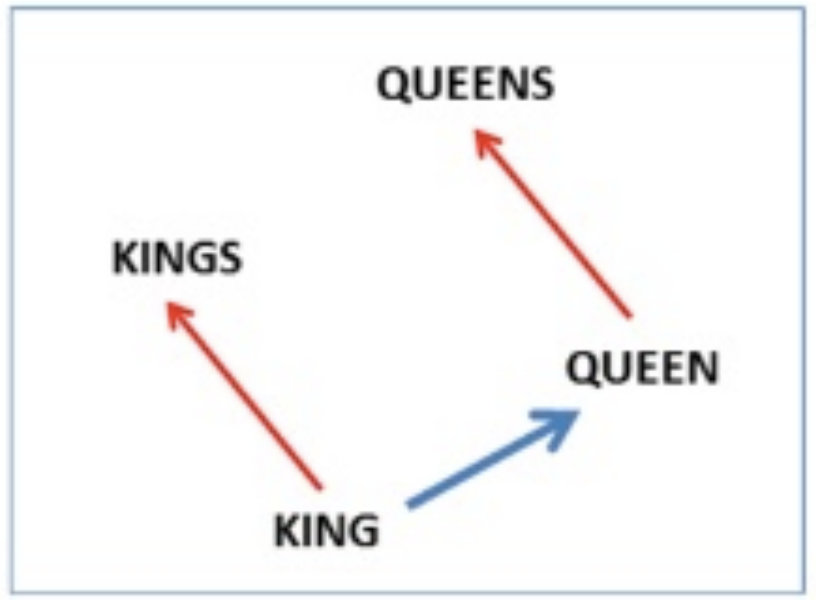


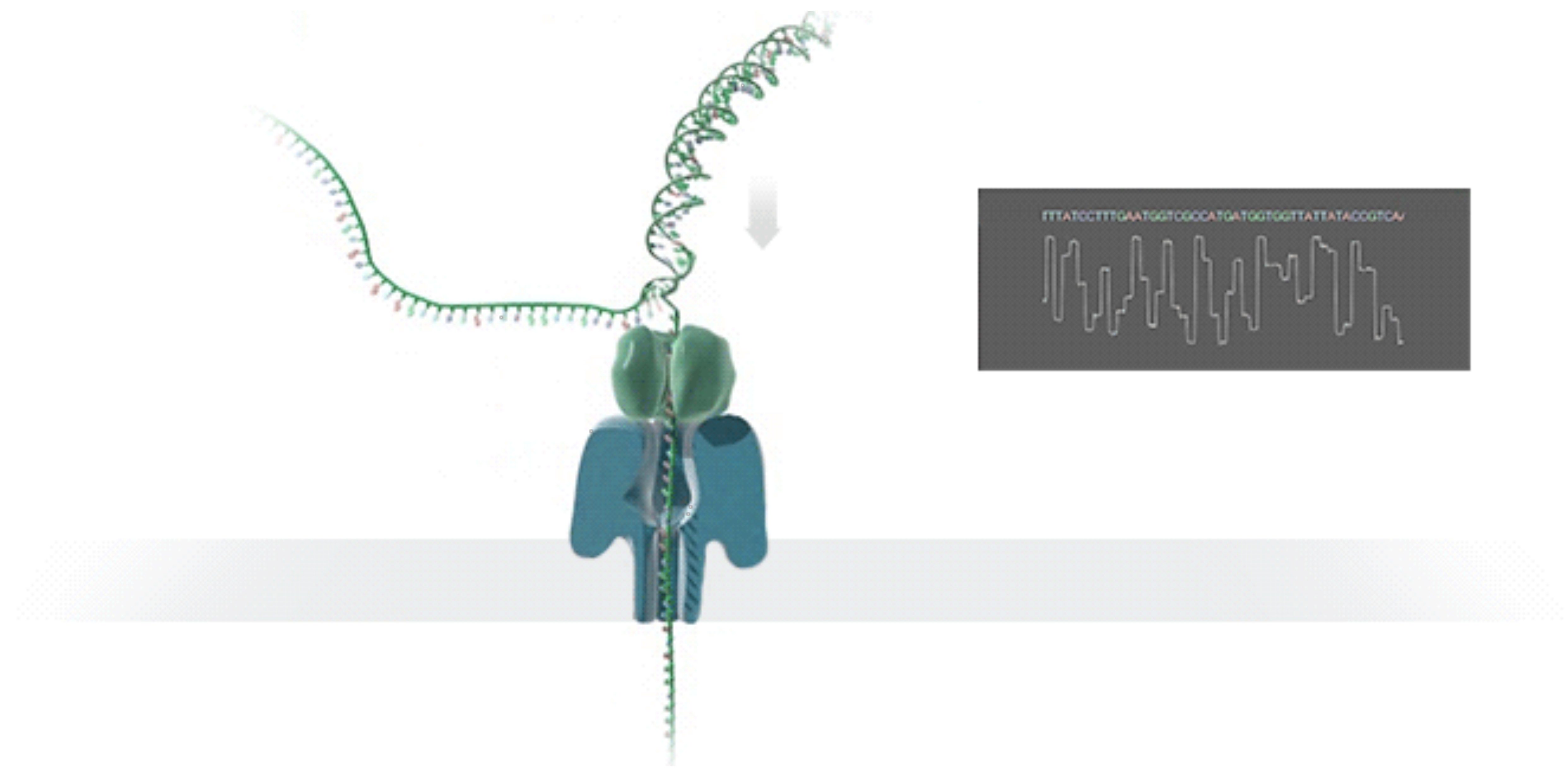
Image recognition

"construction worker in orange safety vest is working on road."

Video

https://www.youtube.com/watch?v=9Yq67CjDqvw

Nanopore Sequencing



Nearly a markov model

- Yet deep learning does “better”. Why?

Why Deep Learning Works

Model deficit

- Hard to model image, speech, language, video..

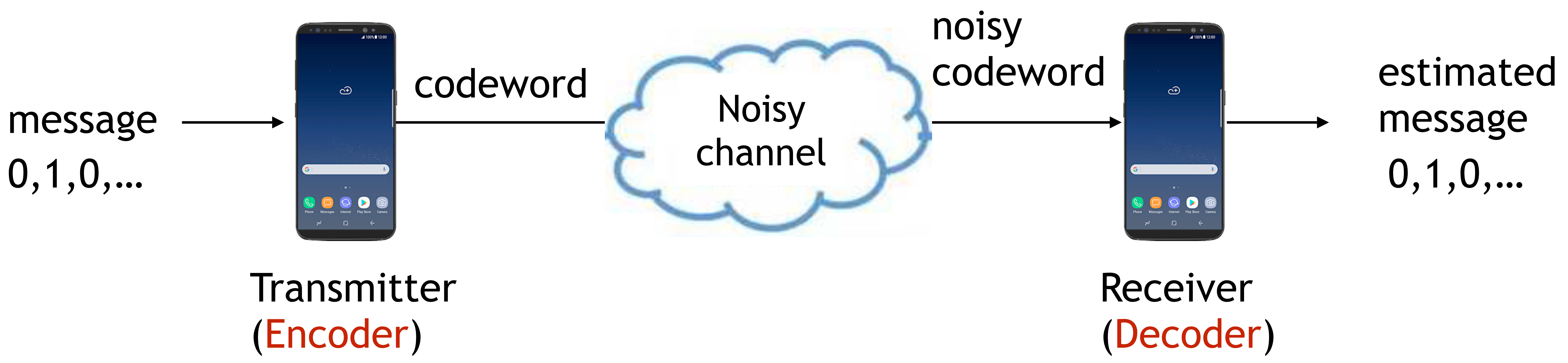
alphaGo => No model deficit

Algorithm deficit

- Hard to find optimal algorithms for known model..

Communication

- Models are well-defined
- Designing a robust code (encoder/decoder) is critical
- Challenge: space of encoder/decoder mappings very large



Design of codes

- Huge practical impact

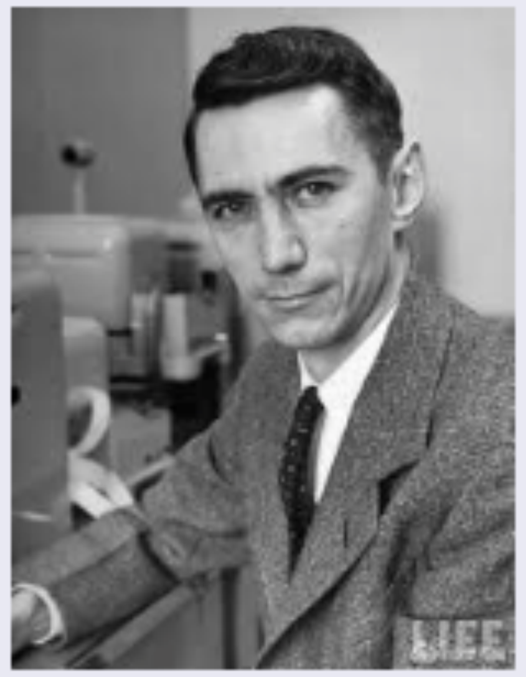
Design of codes

- Technical communities

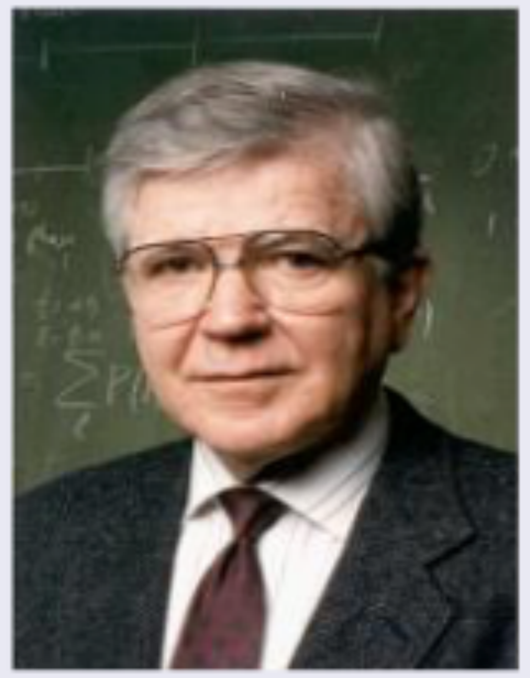
Design of codes

- Technical communities

- Sporadic progress

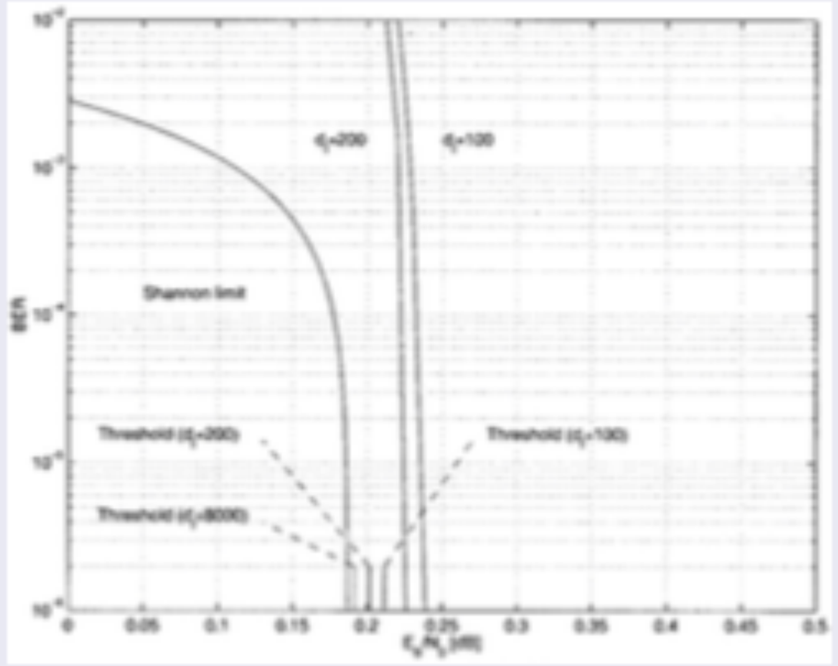


C.E. Shannon
Definition
1948



R.G. Gallager
LDPC Codes
1960

C. Berrou
Turbo Codes
1993
0.7dB



S.Y. Chung
LDPC Codes
2001
0.0045dB

E. Arıkan
Polar Codes
2009
0dB

Overview

- Introduction to Neural Networks
- Inventing neural decoders
- Inventing neural codes
- Other applications of deep learning to information theory

Overview

- Introduction to Neural Networks
- Inventing neural decoders
- Inventing neural codes
- Other applications of deep learning to information theory

Introduction to Neural Networks

Slides made by Sewoong Oh (University of Washington)

Classification

- Problem statement

Given labelled examples $\{(X_i, Y_i)\}_{i=1}^n$, find a classifier f that minimizes the loss $\mathcal{L}$ of our choice

$$\min_f \; \mathbb{E}_{X,Y} \left[\, \mathcal{L}(f(X), Y) \, \right]$$

Classification

- Problem statement

Given labelled examples $\{(X_i, Y_i)\}_{i=1}^n$, find a classifier f that minimizes the loss $\mathcal{L}$ of our choice

$$\min_f \; \mathbb{E}_{X,Y} \left[\, \mathcal{L}(f(X), Y) \, \right]$$

- As we access the joint distribution $P_{X,Y}$ through samples, we minimize the sample mean instead,

$$\min_f \; \frac{1}{n} \sum_{i=1}^{n} \; \mathcal{L}(f(X_i), Y_i)$$

Classification

- Problem statement

Given labelled examples $\{(X_i, Y_i)\}_{i=1}^n$, find a classifier f that minimizes the loss $\mathcal{L}$ of our choice

$$\min_f \ \mathbb{E}_{X,Y} \left[\, \mathcal{L}(f(X), Y) \,\right]$$

- As we access the joint distribution $P_{X,Y}$ through samples, we minimize the sample mean instead,

$$\min_f \ \frac{1}{n} \sum_{i=1}^{n} \ \mathcal{L}(f(X_i), Y_i)$$

- To avoid overfitting to the training samples, we search over a restricted class of functions

$$\min_{f \in \mathcal{F}} \ \frac{1}{n} \sum_{i=1}^{n} \ \mathcal{L}(f(X_i), Y_i)$$

Classification

- Problem statement

Given labelled examples $\{(X_i, Y_i)\}_{i=1}^{n}$, find a classifier f that minimizes the loss $\mathcal{L}$ of our choice

$$\min_{f} \ \mathbb{E}_{X,Y} \left[\, \mathcal{L}(f(X), Y) \, \right]$$

- As we access the joint distribution $P_{X,Y}$ through samples, we minimize the sample mean instead,

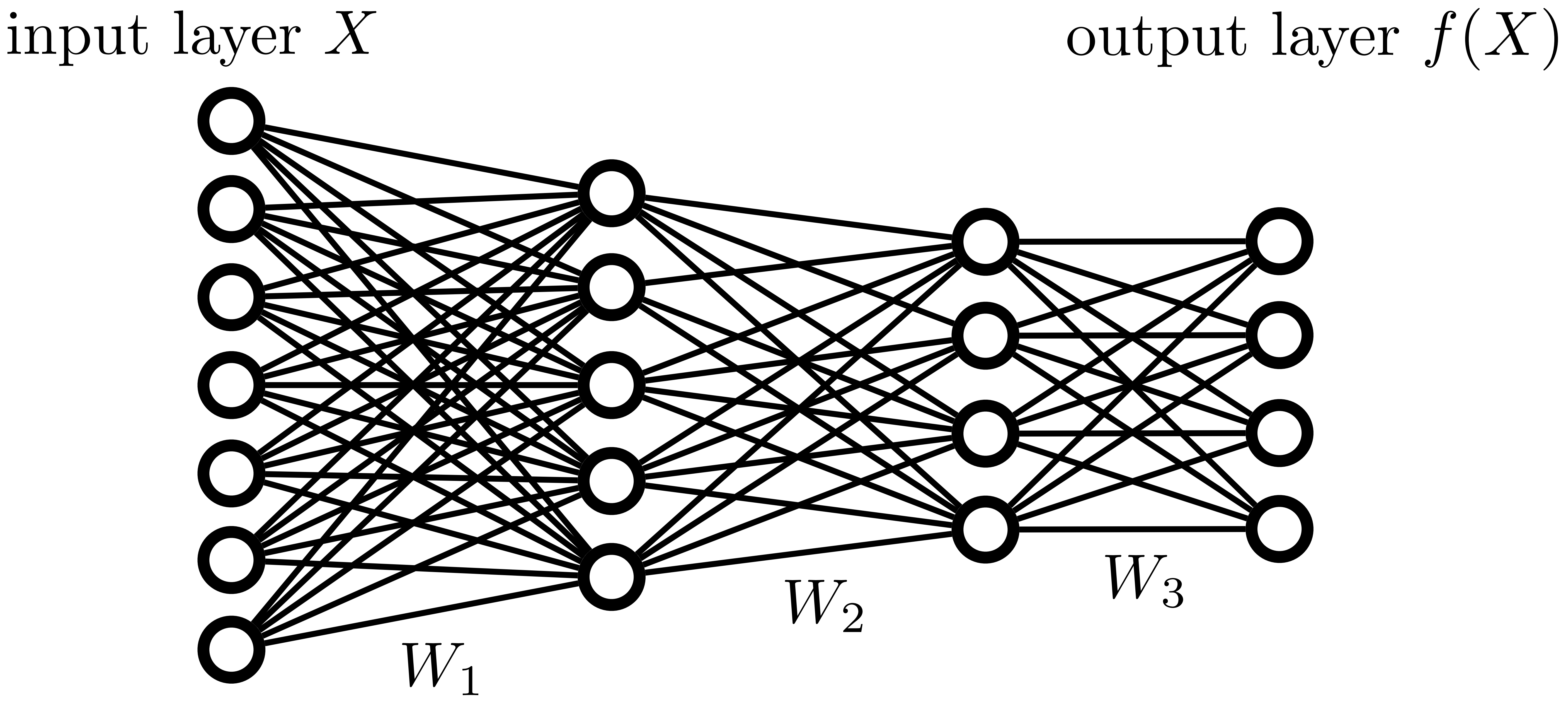
$$\min_{f} \ \frac{1}{n} \sum_{i=1}^{n} \ \mathcal{L}(f(X_i), Y_i)$$

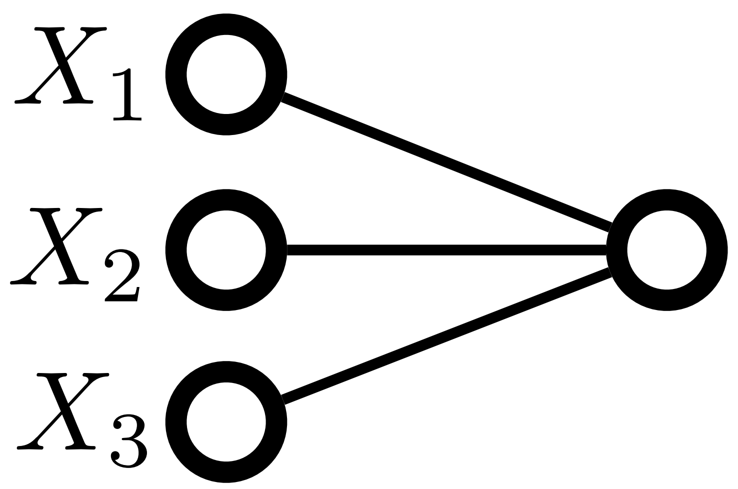
- To avoid overfitting to the training samples, we search over a restricted class of functions

$$\min_{f \in \mathcal{F}} \ \frac{1}{n} \sum_{i=1}^{n} \ \mathcal{L}(f(X_i), Y_i)$$

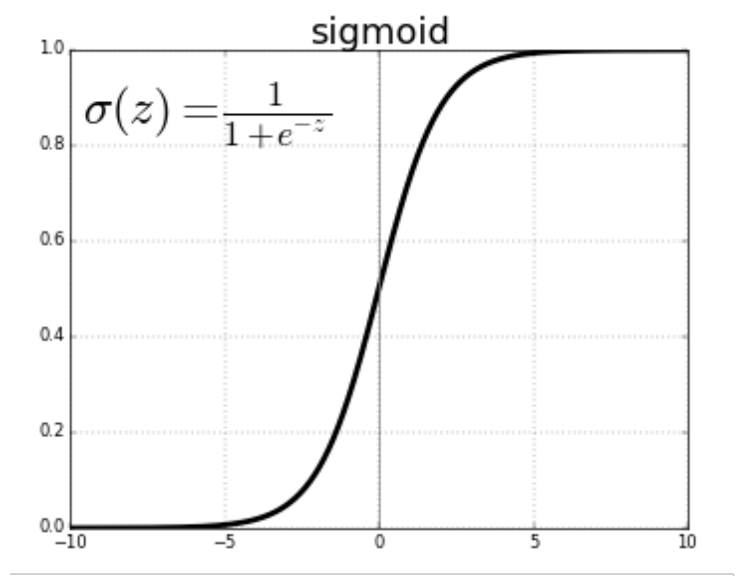
- Neural networks: a parametric family with a graceful tradeoff between representation and generalization

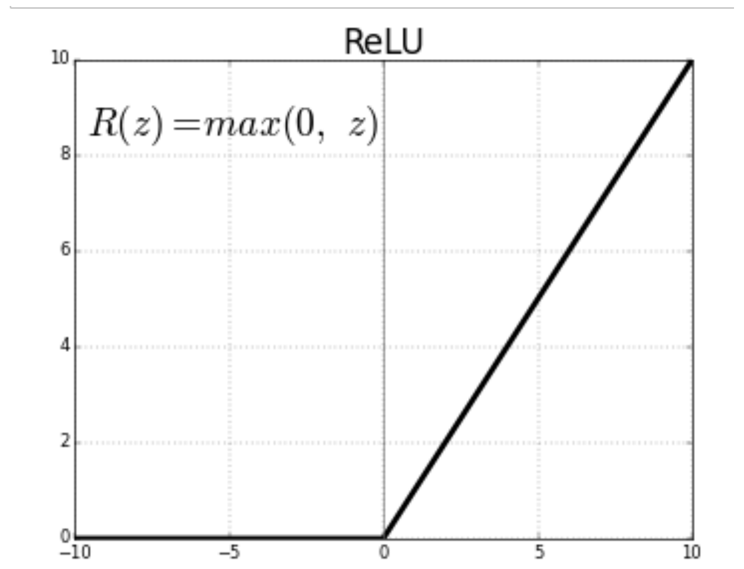
Neural Network of depth d and weights $(W_1, \ldots, W_d)$





$$\sigma(W_{11}X_1 + W_{13}X_3 + W_{13}X_3)$$





$$f(X) = \sigma\Big(W_d \cdots \sigma \big(W_2 \, \sigma(W_1 X) \big) \cdots \Big)$$

Gradient computation is simple

- Choose the loss function (e.g. for binary classification)
  - L_2 loss

$$\min_{W_1,\ldots,W_d} \frac{1}{n}\sum_{i=1}^{n}(Y_i - f(X_i))^2$$

Gradient computation is simple

- Choose the loss function (e.g. for binary classification)
  - L_2 loss

$$\min_{W_1,\ldots,W_d} \frac{1}{n}\sum_{i=1}^{n}(Y_i - f(X_i))^2$$

  - Cross entropy loss

$$\min_{W_1,\ldots,W_d} \frac{1}{n}\sum_{i=1}^{n} -\{Y_i \log(f(X_i)) + (1 - Y_i)\log(1 - f(X_i))\}$$

Gradient computation is simple

- Choose the loss function (e.g. for binary classification)
  - L_2 loss

$$\min_{W_1,\ldots,W_d} \frac{1}{n}\sum_{i=1}^{n}(Y_i - f(X_i))^2$$

  - Cross entropy loss

$$\min_{W_1,\ldots,W_d} \frac{1}{n}\sum_{i=1}^{n} -\{Y_i \log(f(X_i)) + (1-Y_i)\log(1-f(X_i))\}$$

- (variants of) gradient descent are used
  - Efficient gradient computation via backpropagation

$$f(X) = \sigma\Big(W_d \cdots \sigma\left(W_2\, \sigma(W_1 X) \right) \cdots \Big)$$

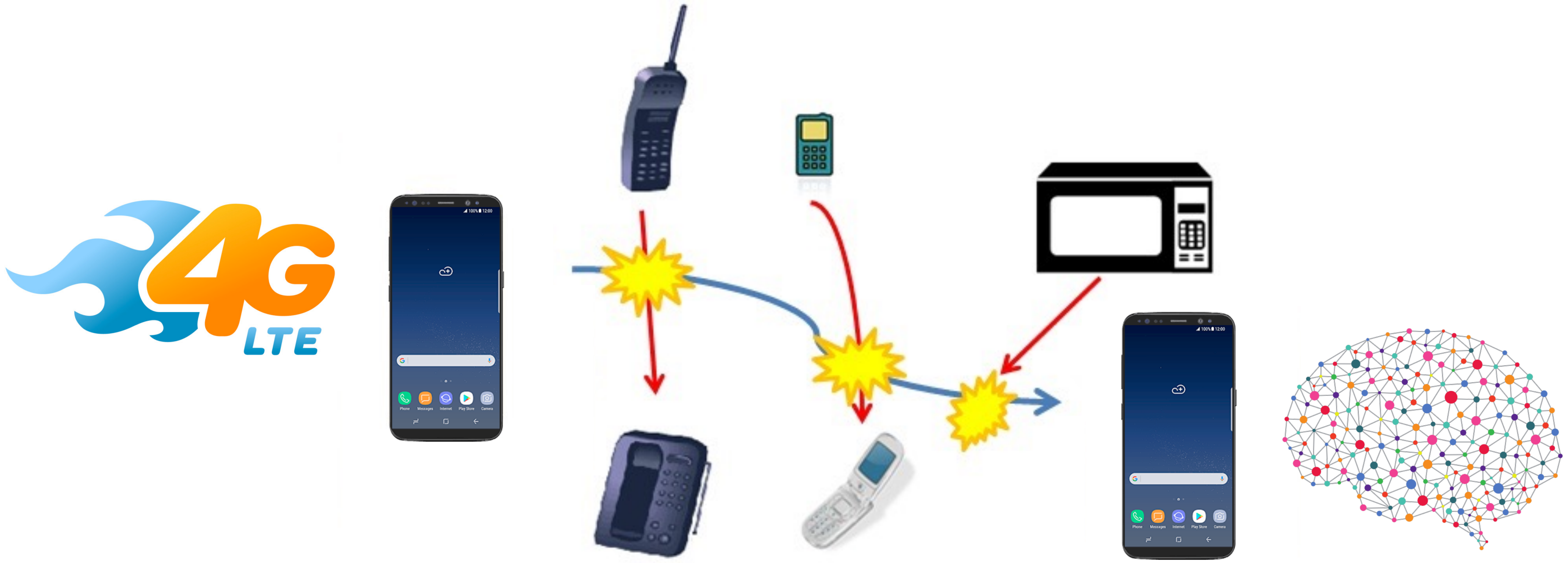
Overview

- Introduction to Neural Networks
- Inventing neural decoders
- Inventing neural codes
- Other applications of deep learning to information theory

Overview

- Inventing neural decoders
  - Example
    - Learning state-of-the-art decoders for AWGN channels
    - Improving the state-of-the-art decoders for non-AWGN channels
  - Literature
  - Open problems

Deep learning based decoder for practical channels



“*Communication Algorithms via Deep Learning,*” Kim-Jiang-Rana-Kannan-Oh-Viswanath ICLR ’18

Sequential codes

- Convolutional codes, turbo codes

Sequential codes

- Convolutional codes, turbo codes
- Practical
  - 3G/4G mobile communications (e.g., in UMTS and LTE)
  - (Deep space) satellite communications

Sequential codes

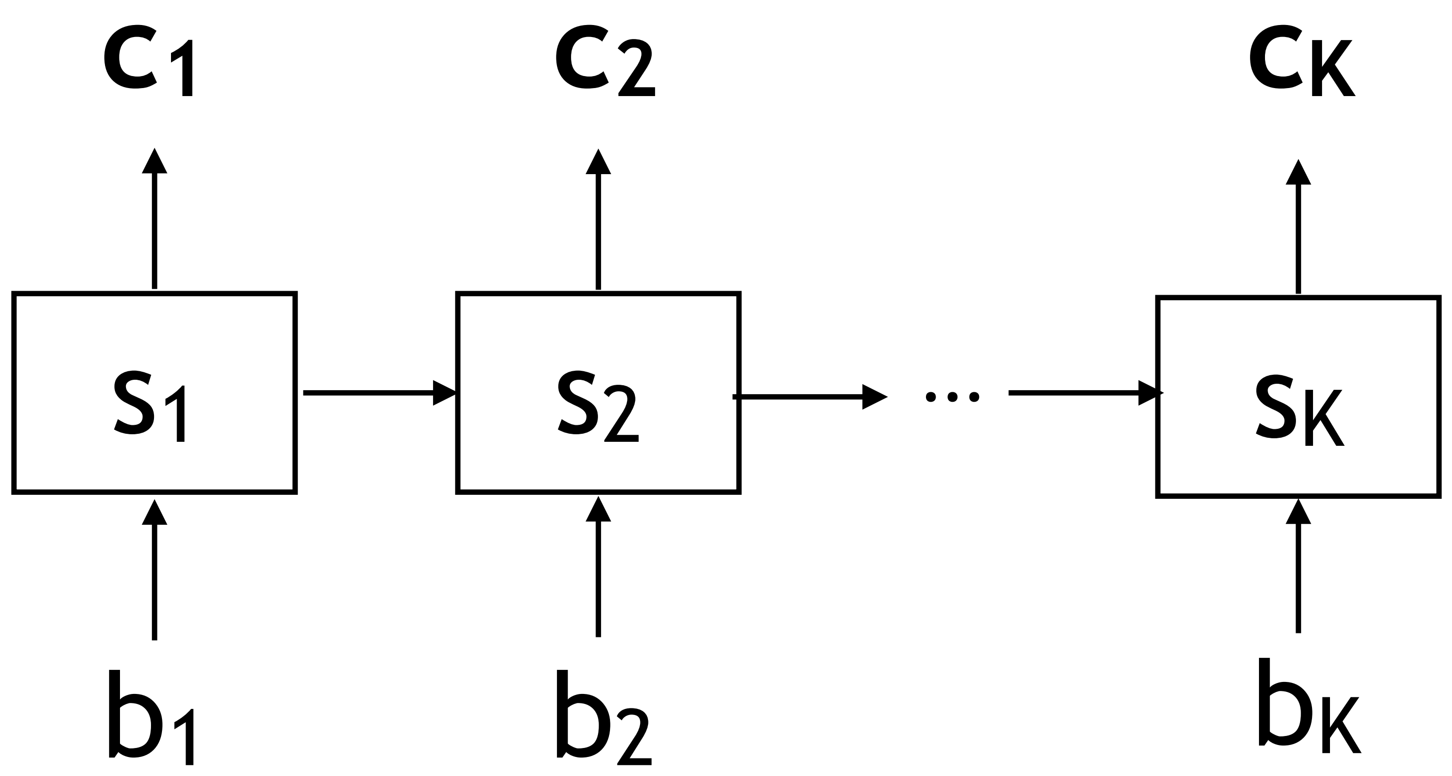
- Convolutional codes, turbo codes
- Practical
  - 3G/4G mobile communications (e.g., in UMTS and LTE)
  - (Deep space) satellite communications
- Achieve performance close to fundamental limit

Sequential codes

- Convolutional codes, turbo codes
- Practical
  - 3G/4G mobile communications (e.g., in UMTS and LTE)
  - (Deep space) satellite communications
- Achieve performance close to fundamental limit
- Recurrent structure aligns well w. Recurrent Neural Networks

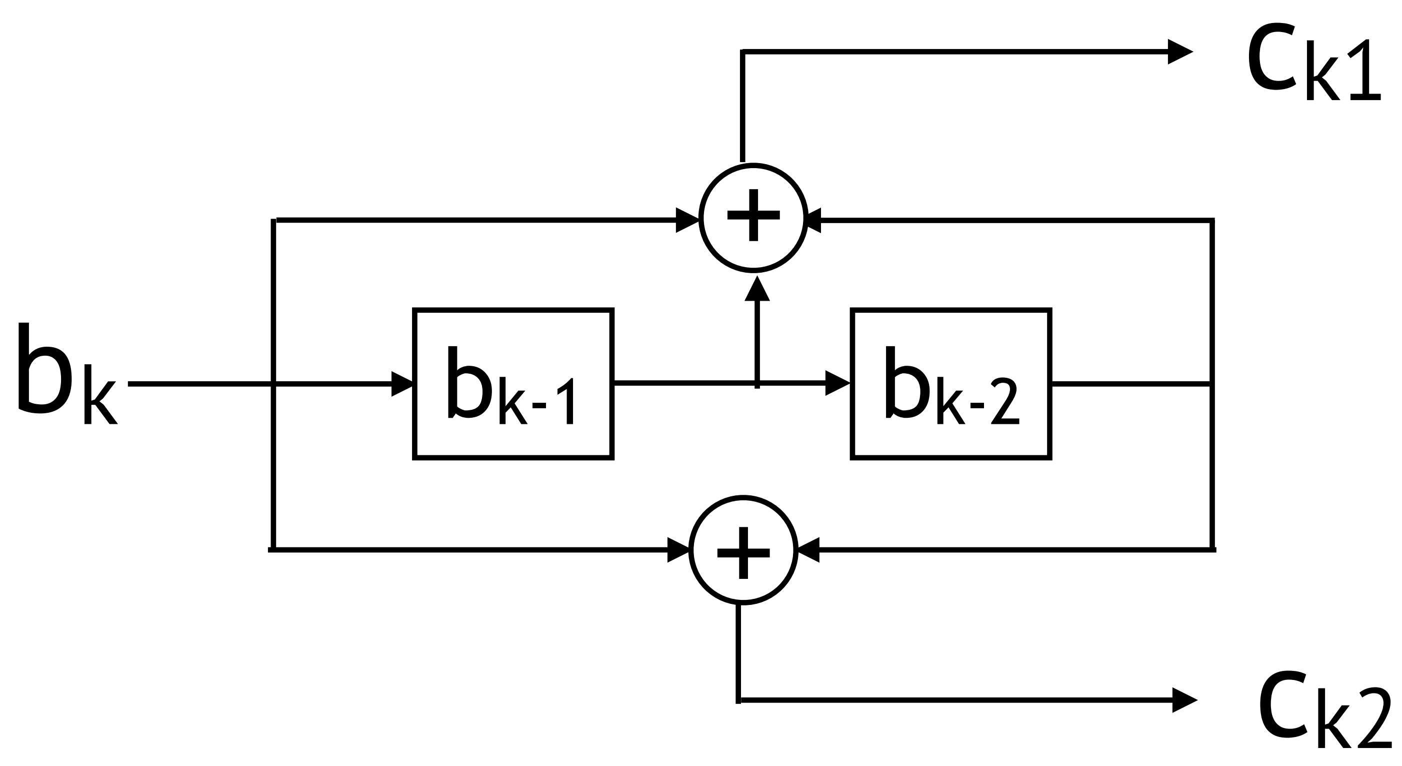
Sequential codes

- Mapping a message bit sequence **b** to a codeword seq. **c**



Sequential codes

- Convolutional codes

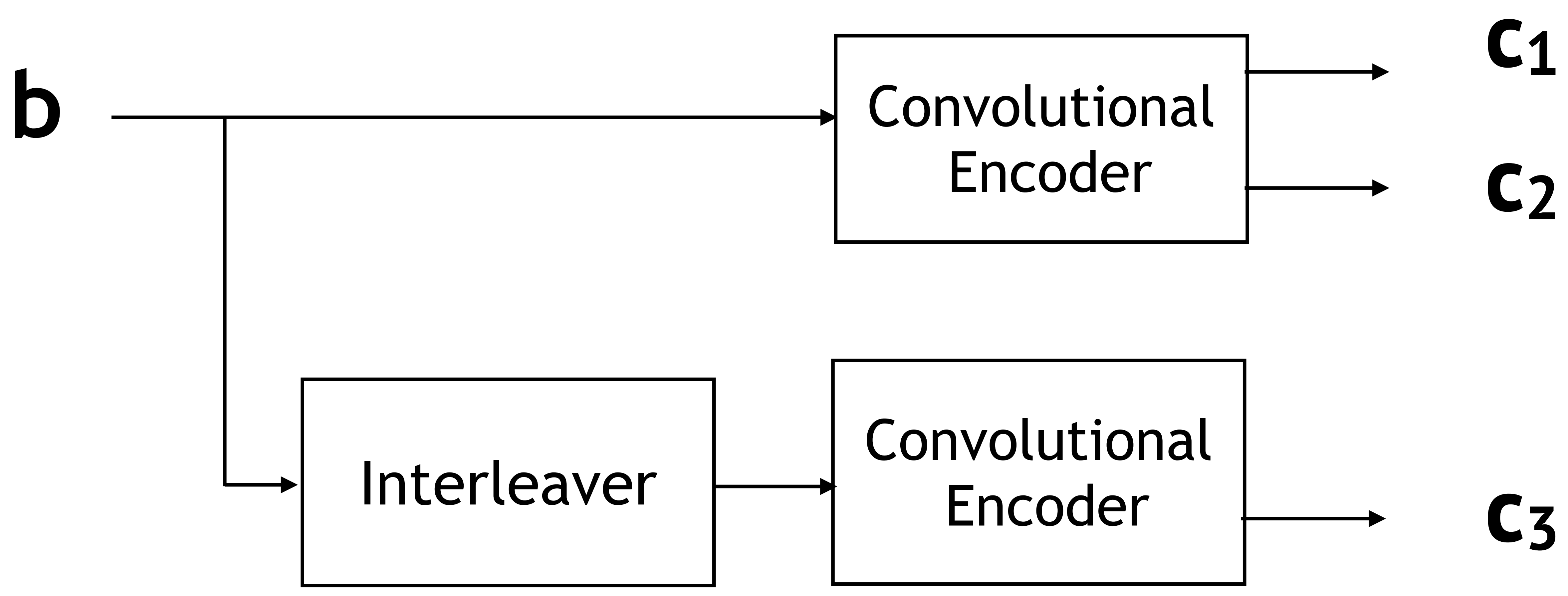


$s_k = (b_k, b_{k-1}, b_{k-2})$

Example of a rate 1/2 convolutional code

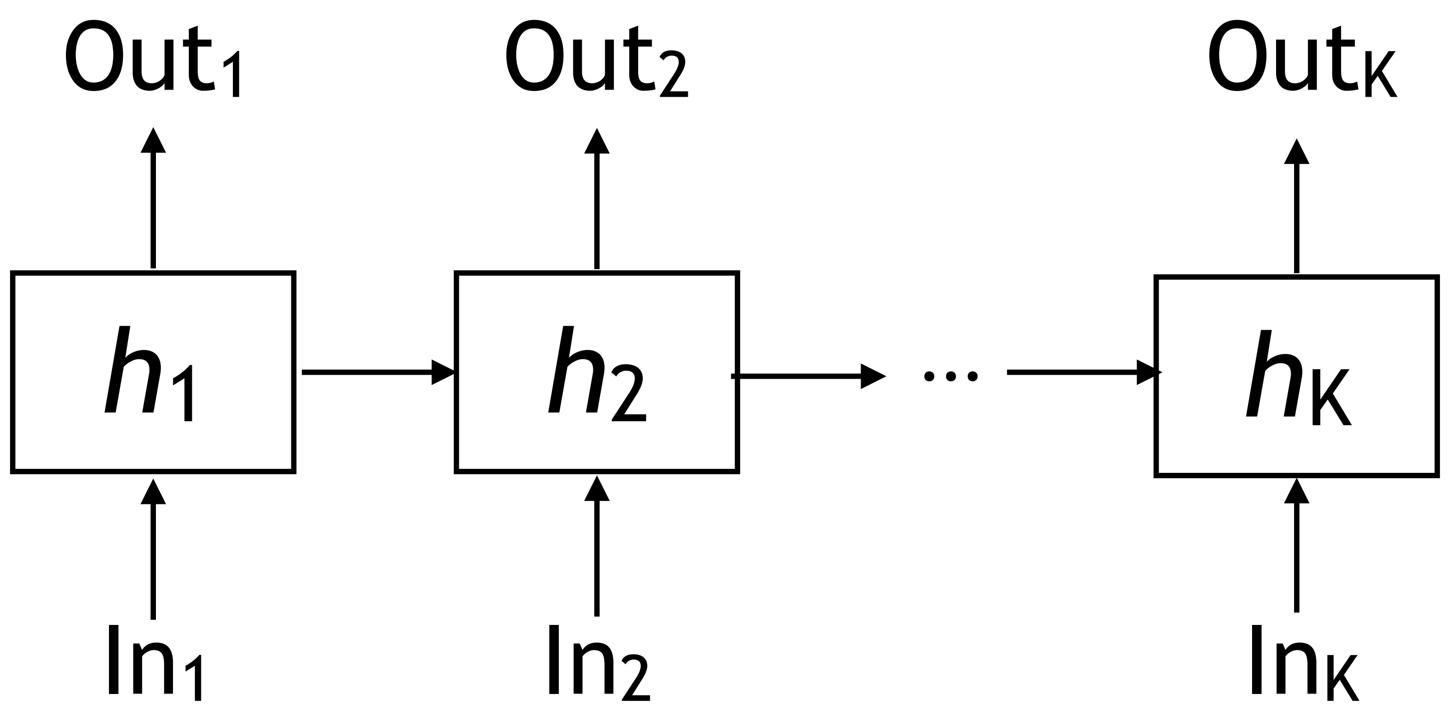
Sequential codes

- Turbo codes
  - Concatenate codewords from two convolutional encoders



Recurrent Neural Network (RNN)

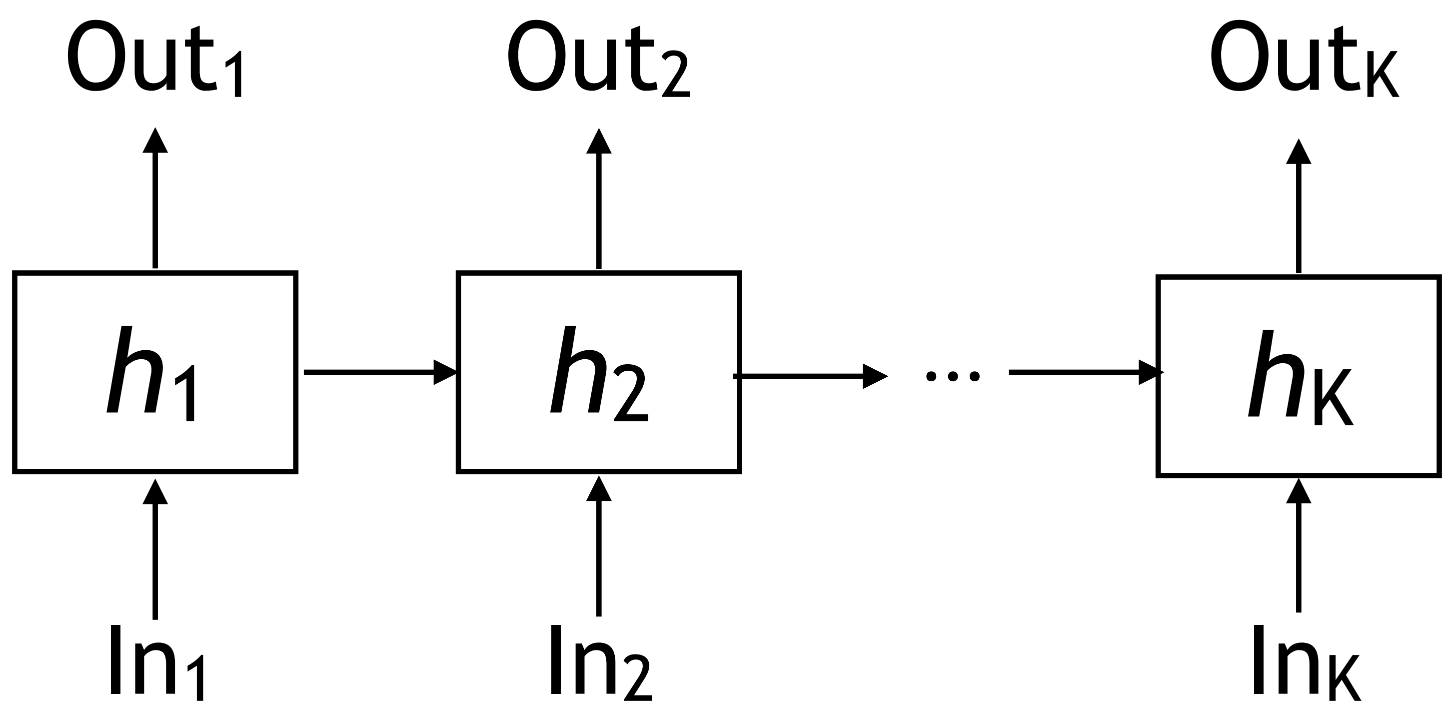
- Sequential mappings with memory



$$h_k = f(h_{k-1}, \mathrm{In}_k)$$
$$\mathrm{Out}_k = g(h_k)$$

Recurrent Neural Network (RNN)

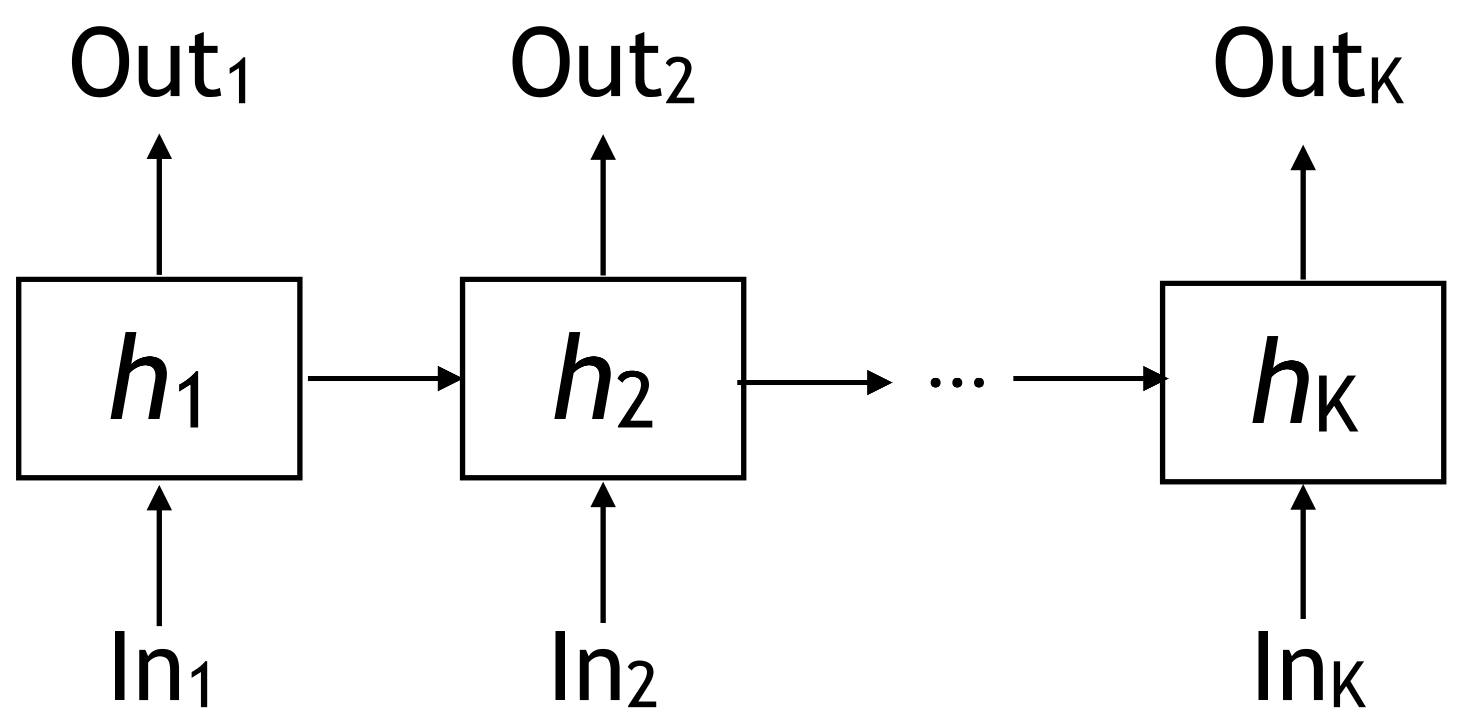
- Sequential mappings with memory



$$h_k = f(h_{k-1}, \mathrm{In}_k)$$
$$\mathrm{Out}_k = g(h_k)$$

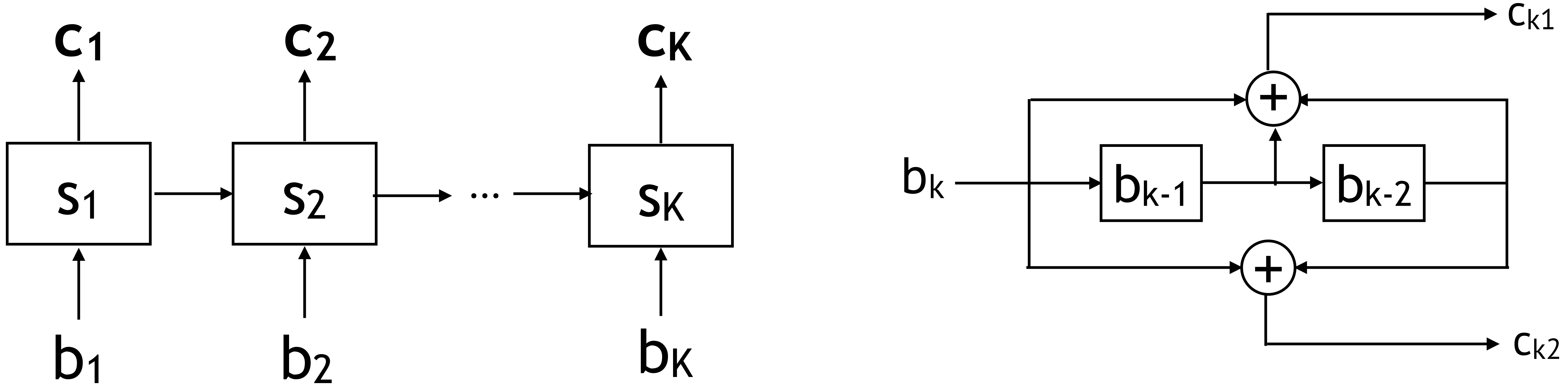
$$h_k = \tanh(W\mathrm{In}_k + Uh_{k-1})$$
$$\mathrm{Out}_k = Vh_k$$

Recurrent neural network and sequential codes



$$h_k = f(h_{k-1}, \mathrm{In}_k)$$
$$\mathrm{Out}_k = g(h_k)$$

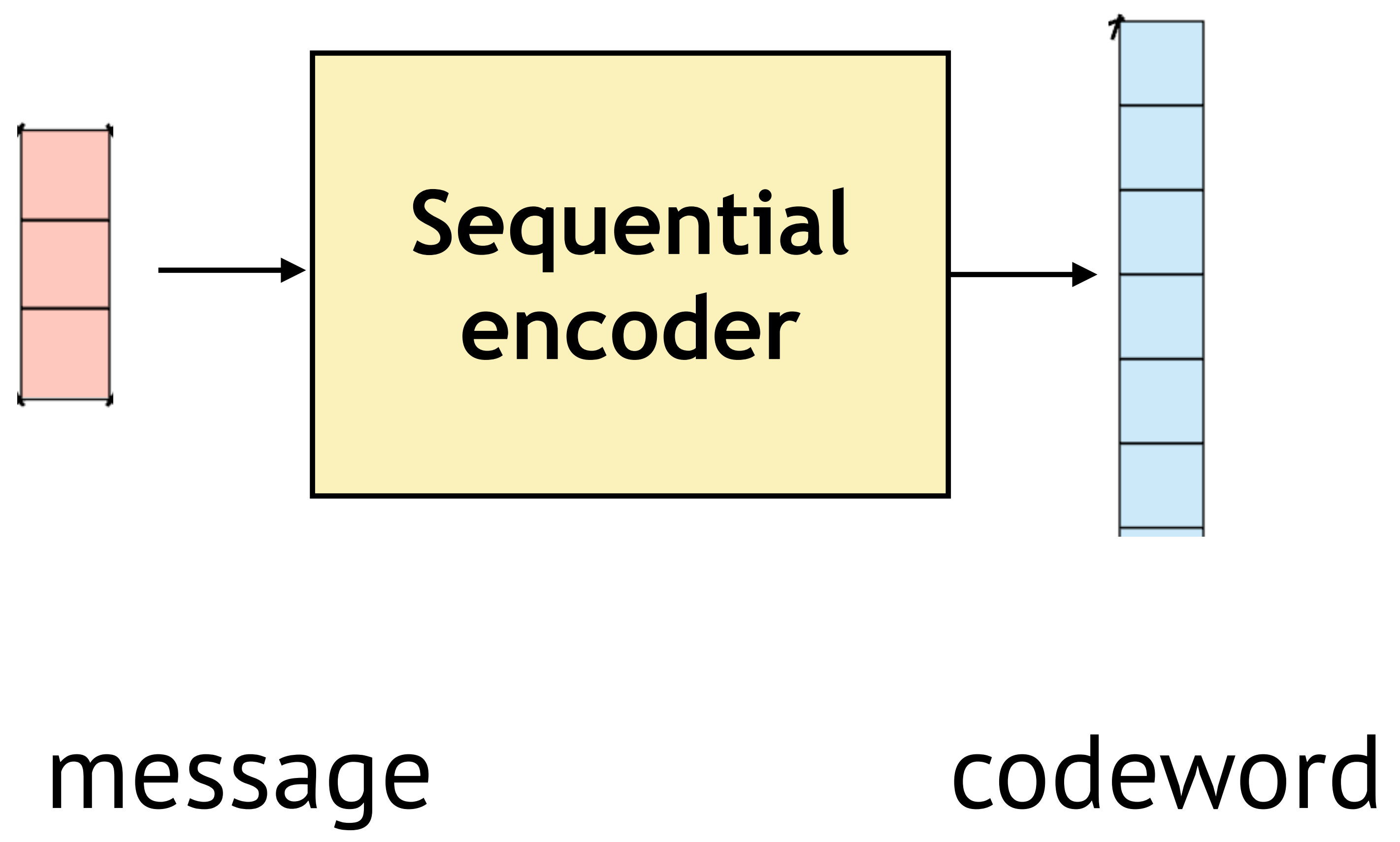
Recurrent Neural Network



$s_k = (b_k, b_{k-1}, b_{k-2})$

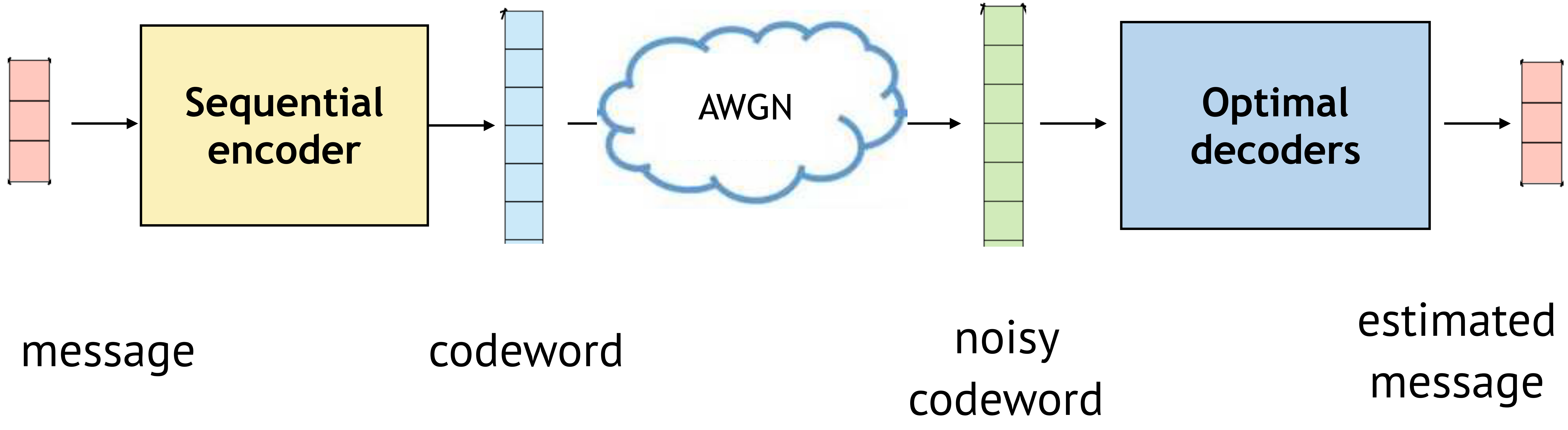
Convolutional codes

Sequential codes



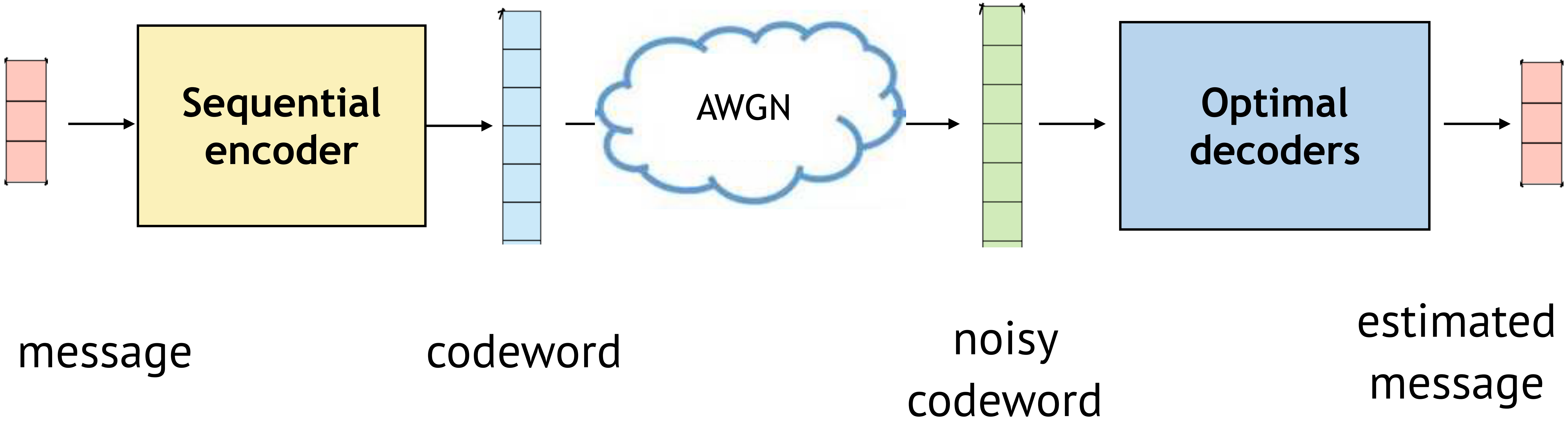
Sequential codes under AWGN

- Optimal decoders known for convolutional codes
  - Viterbi (Viterbi '67) – dynamic programming
  - BCJR (Bahl-Cocke-Jelinek-Raviv '74) – forward-backward alg.



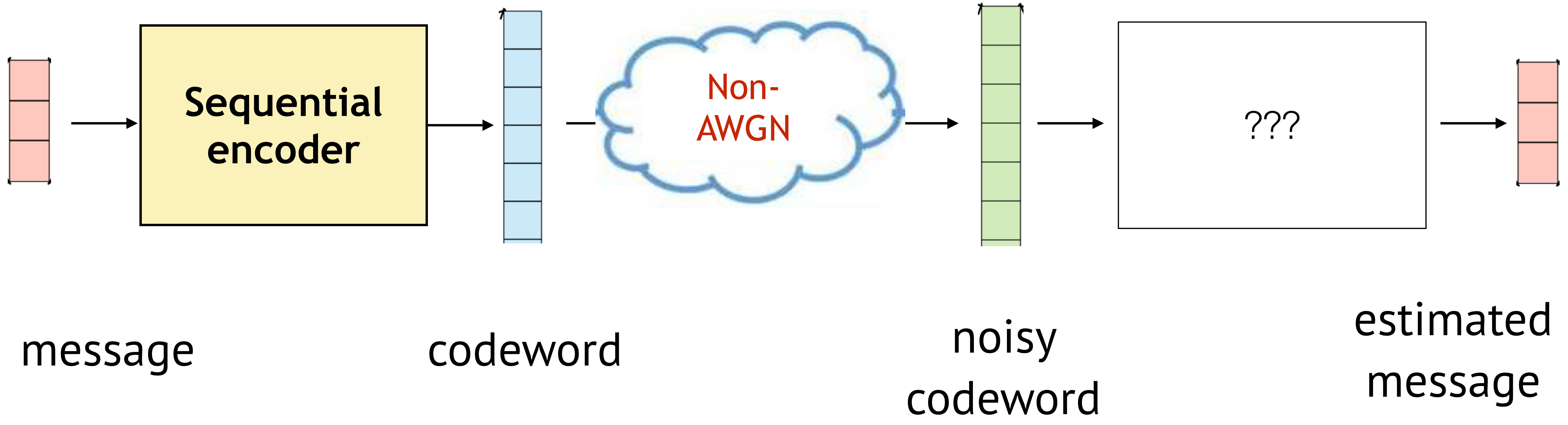
Sequential codes under AWGN

- Optimal decoders known for convolutional codes
  - Viterbi (Viterbi ’67) – dynamic programming
  - BCJR (Bahl-Cocke-Jelinek-Raviv ’74) – forward-backward alg.
- Efficient iterative decoders for turbo codes



Non-AWGN channel

- Decoding becomes challenging



Bursty noise

- High-power noise is added occasionally

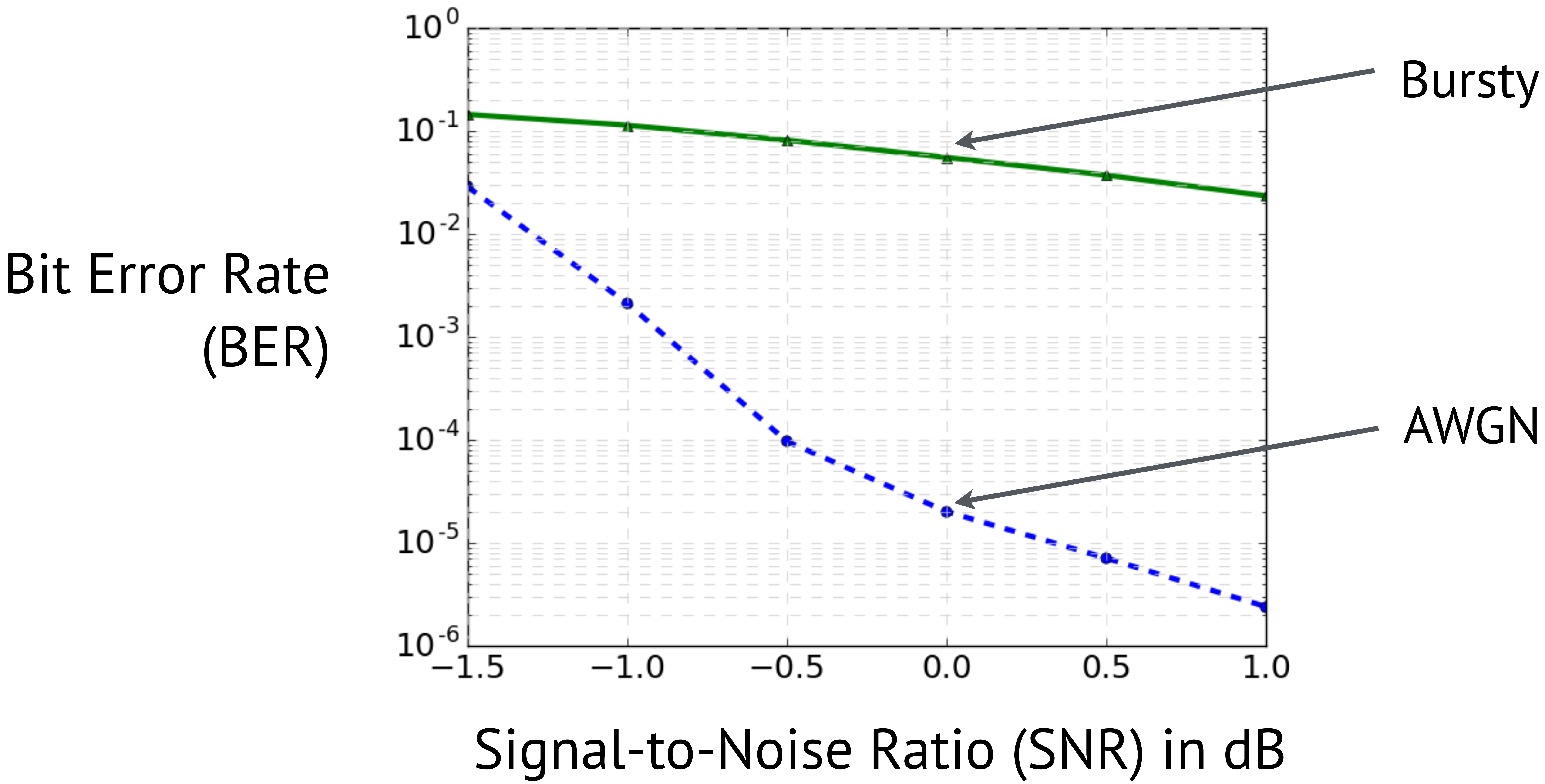
Bursty noise

- High-power noise is added occasionally

- Decoders designed for AWGN channels fail significantly

Bursty noise

- Decoding turbo codes under bursty vs. AWGN channels



Rate 1/3 turbo code, block length 1000

Bursty noise

- Decoders designed for AWGN channels fail significantly
  - Challenge 1. decoders that are robust to channel statistics?

Bursty noise

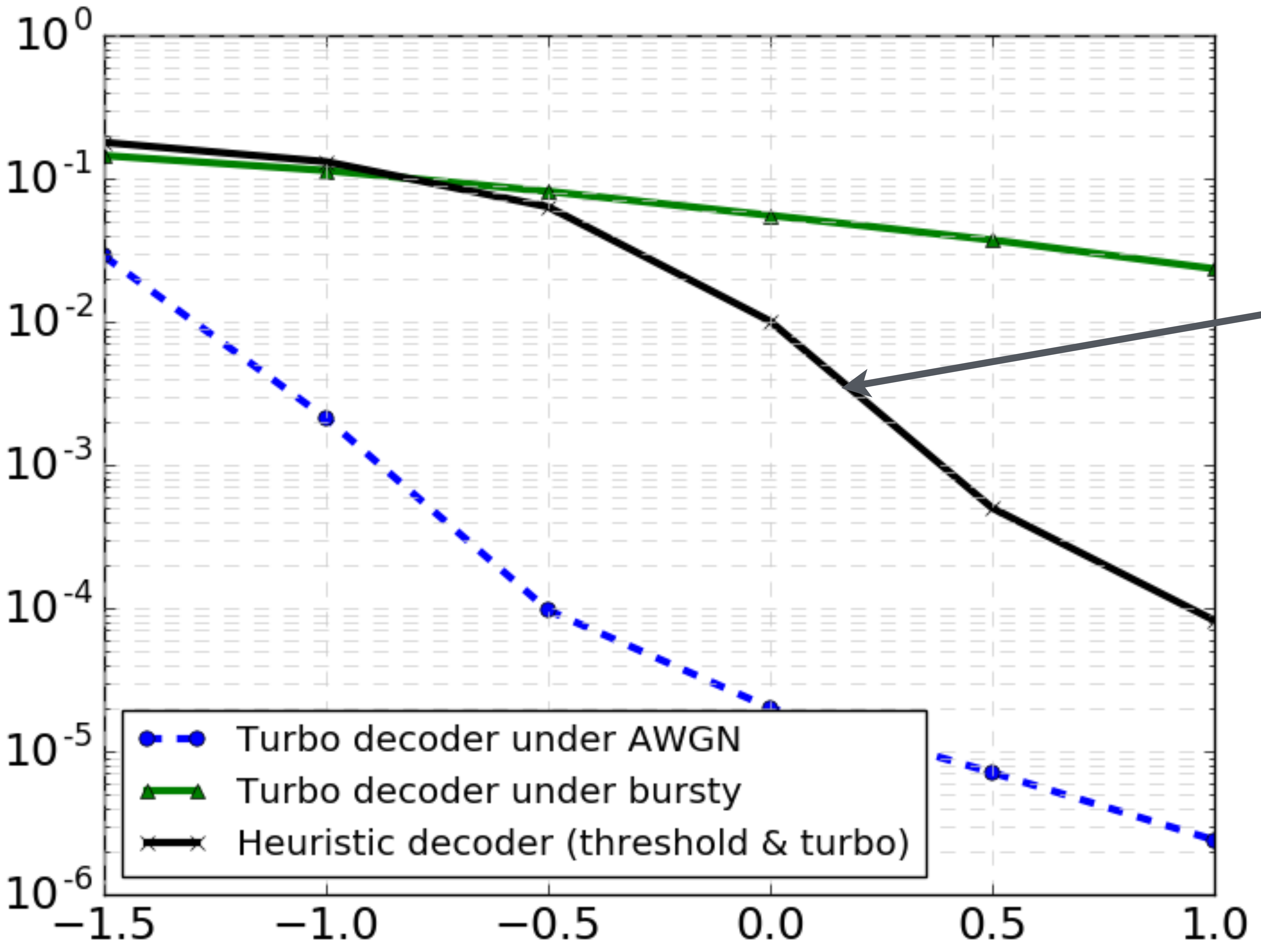
- Decoders designed for AWGN channels fail significantly
  - Challenge 1. decoders that are robust to channel statistics?

- Heuristic decoders are used

Bursty noise

- Heuristic decoders (thresholding) are used

Bit Error Rate (BER)



Heuristic under bursty

Signal-to-Noise Ratio (SNR) in dB

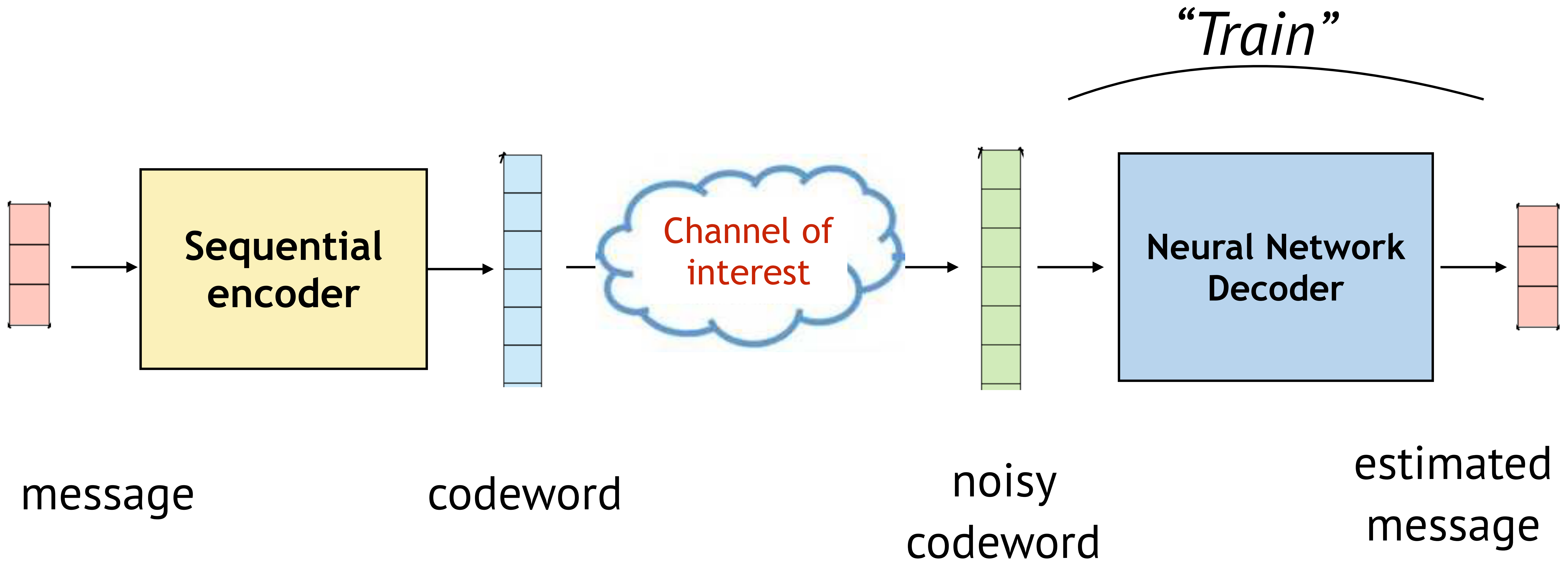
Rate 1/3 turbo code, block length 1000

Bursty noise

- Decoders designed for AWGN channels fail significantly
  - Challenge 1. decoders that are robust to channel statistics?
- Heuristic decoders are used
  - Challenge 2. decoders that adapt better?

Our approach

- Model decoder as a neural network and learn

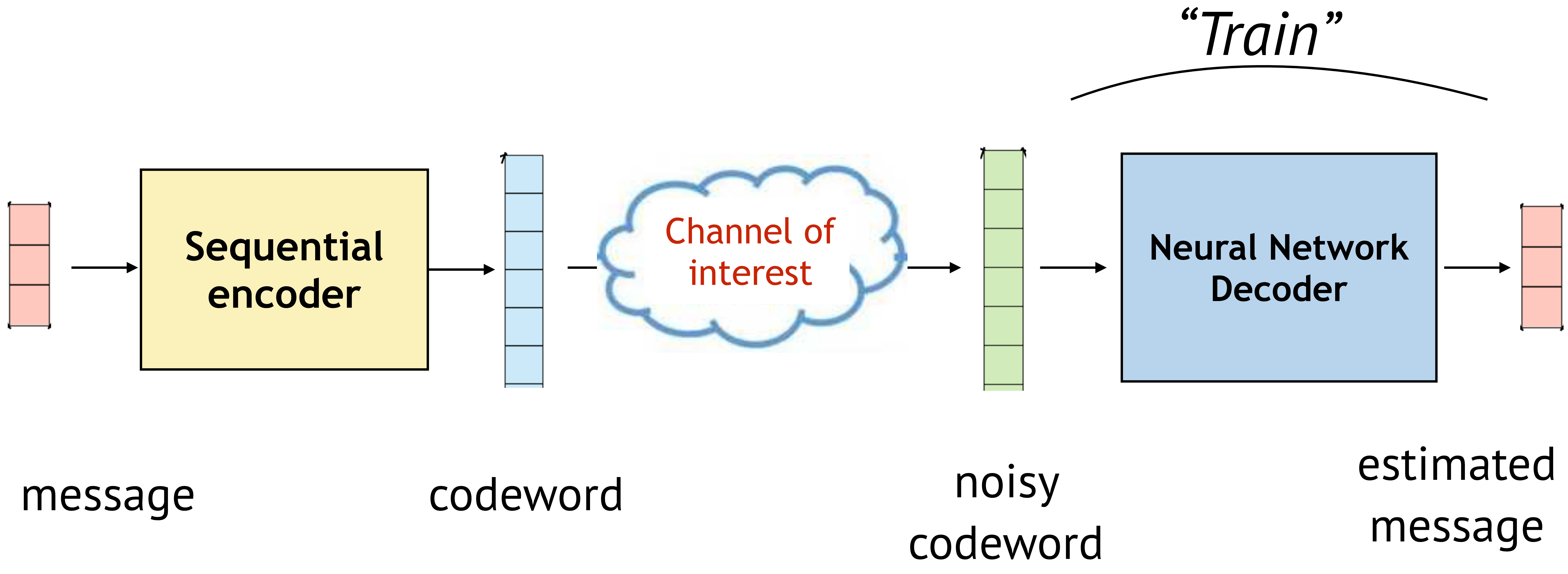


Our approach

- Model decoder as a neural network and learn

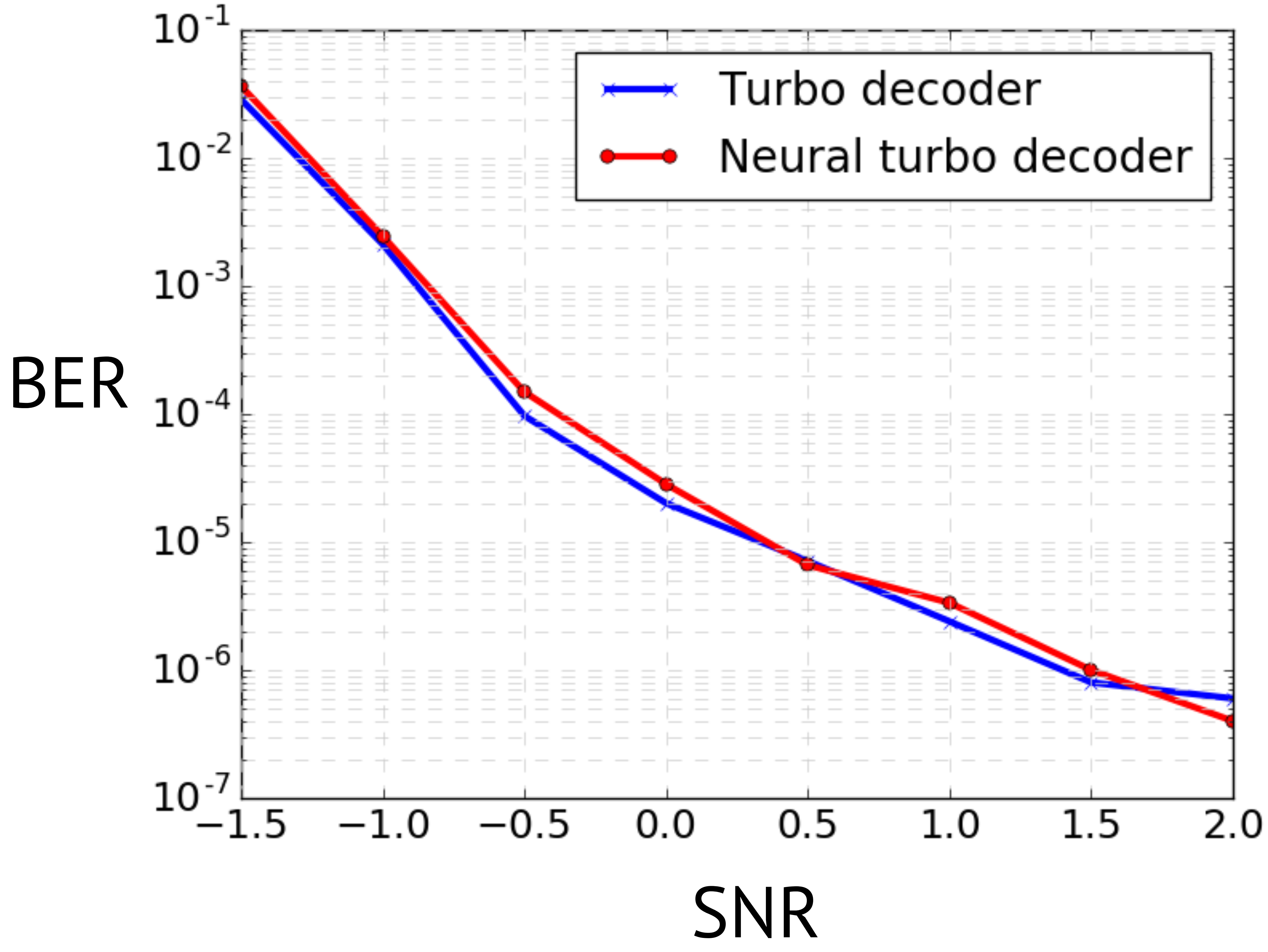
- Neural network based decoder is robust and adaptive

  (Kim-Jiang-Rana-Kannan-Oh-Viswanath '18)



Main results: Robustness

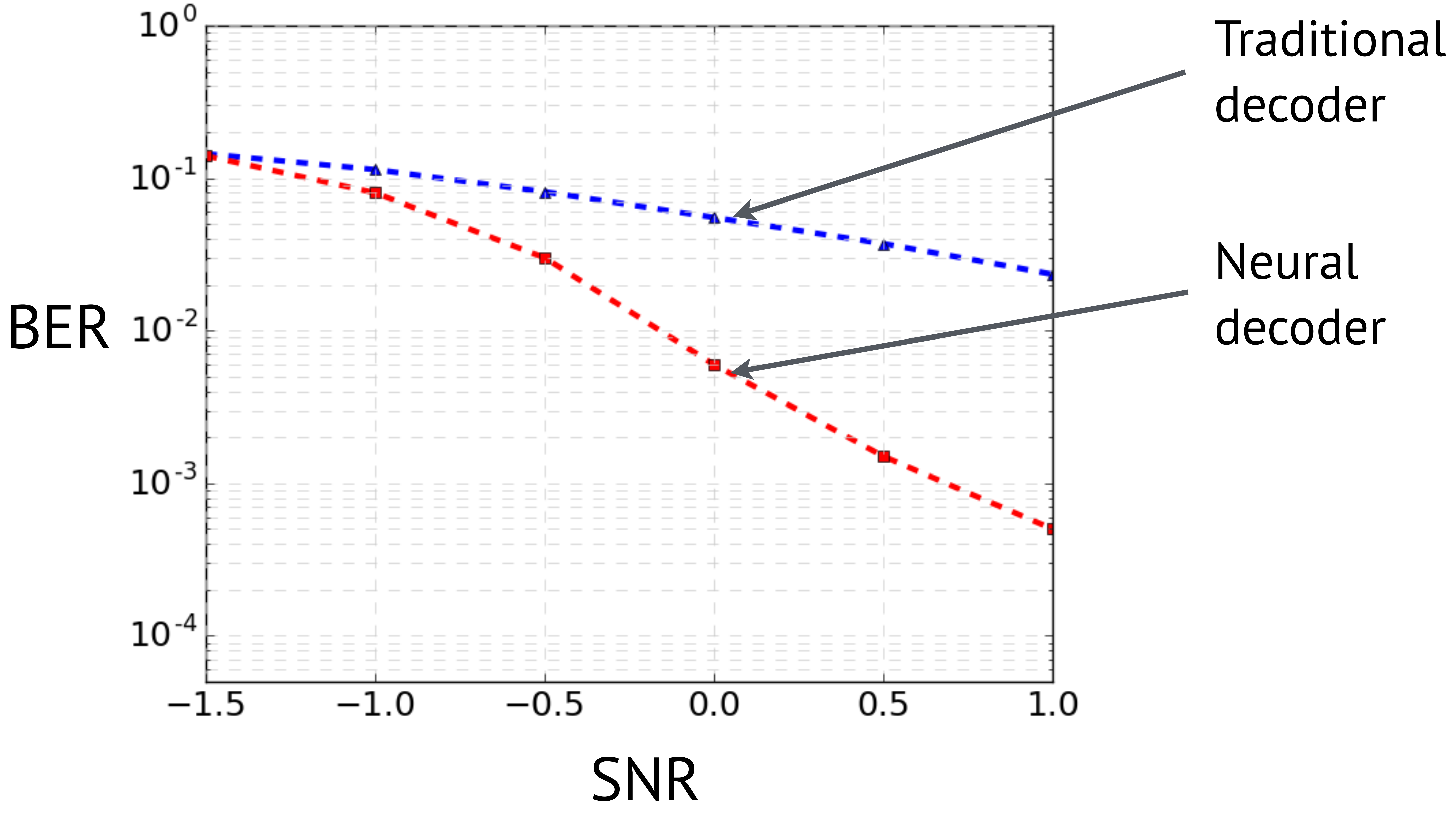
- Neural decoder as reliable as traditional dec. under AWGN



Rate 1/3 turbo code, block length 1000

Main results: Robustness

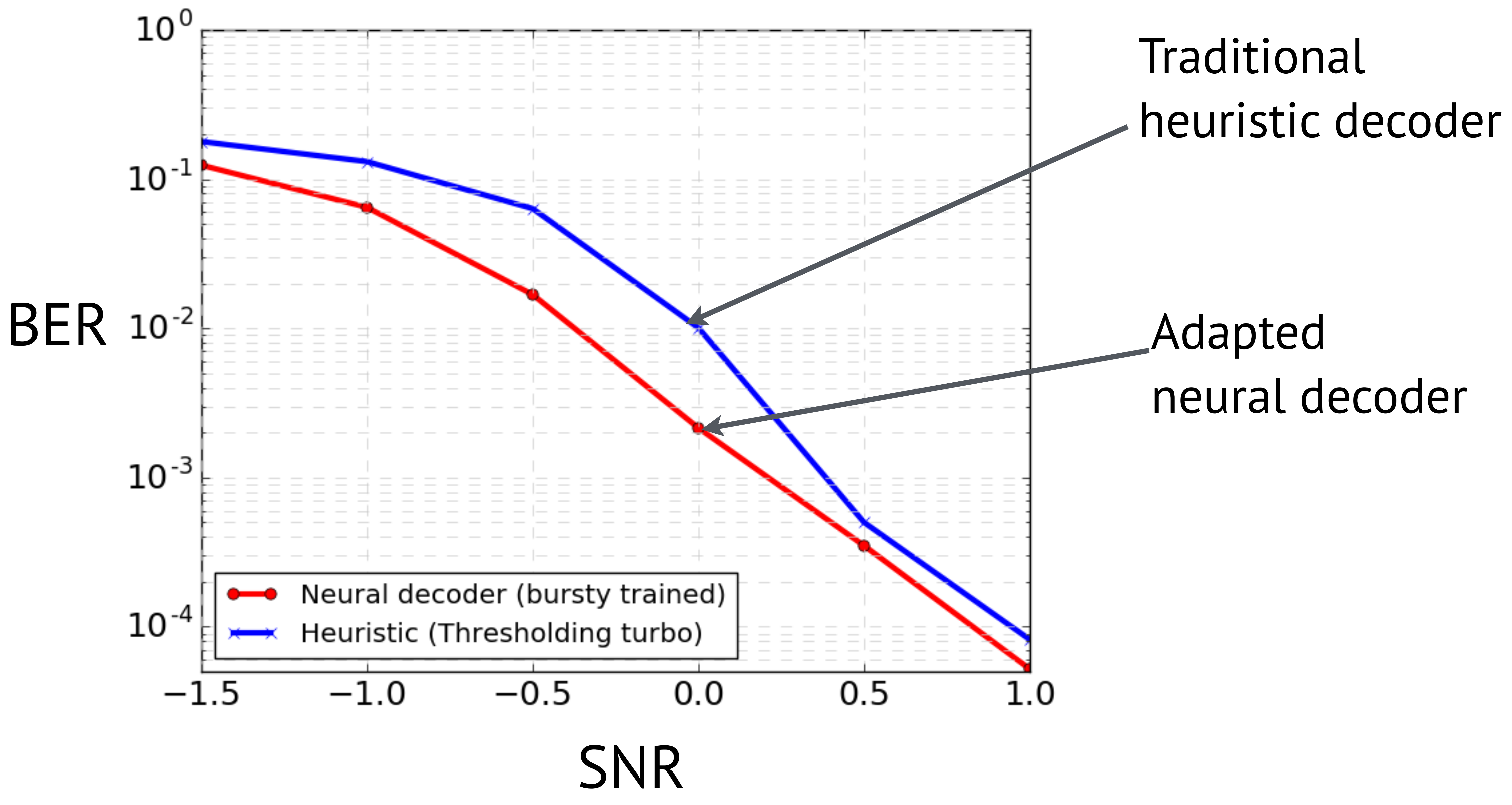
- Neural decoder is more reliable under bursty channels



Rate 1/3 turbo code, block length 1000

Main results: Adaptivity

- Adapted neural decoder is more reliable than heuristics



Rate 1/3 turbo code, block length 1000

Outline - learning a decoder

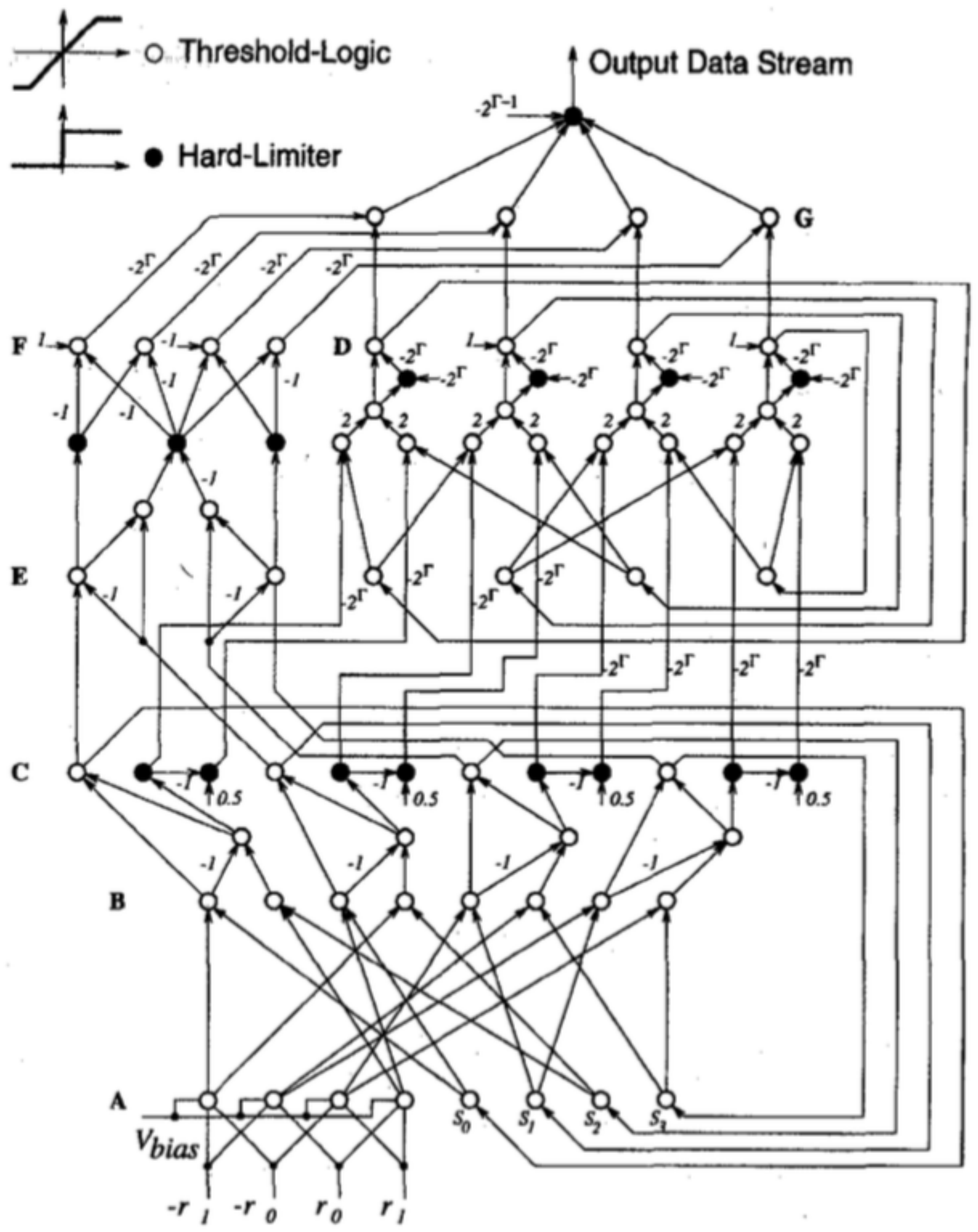
1. Convolutional codes under AWGN

2. Turbo codes under AWGN

3. Turbo codes under bursty

Convolutional codes under AWGN

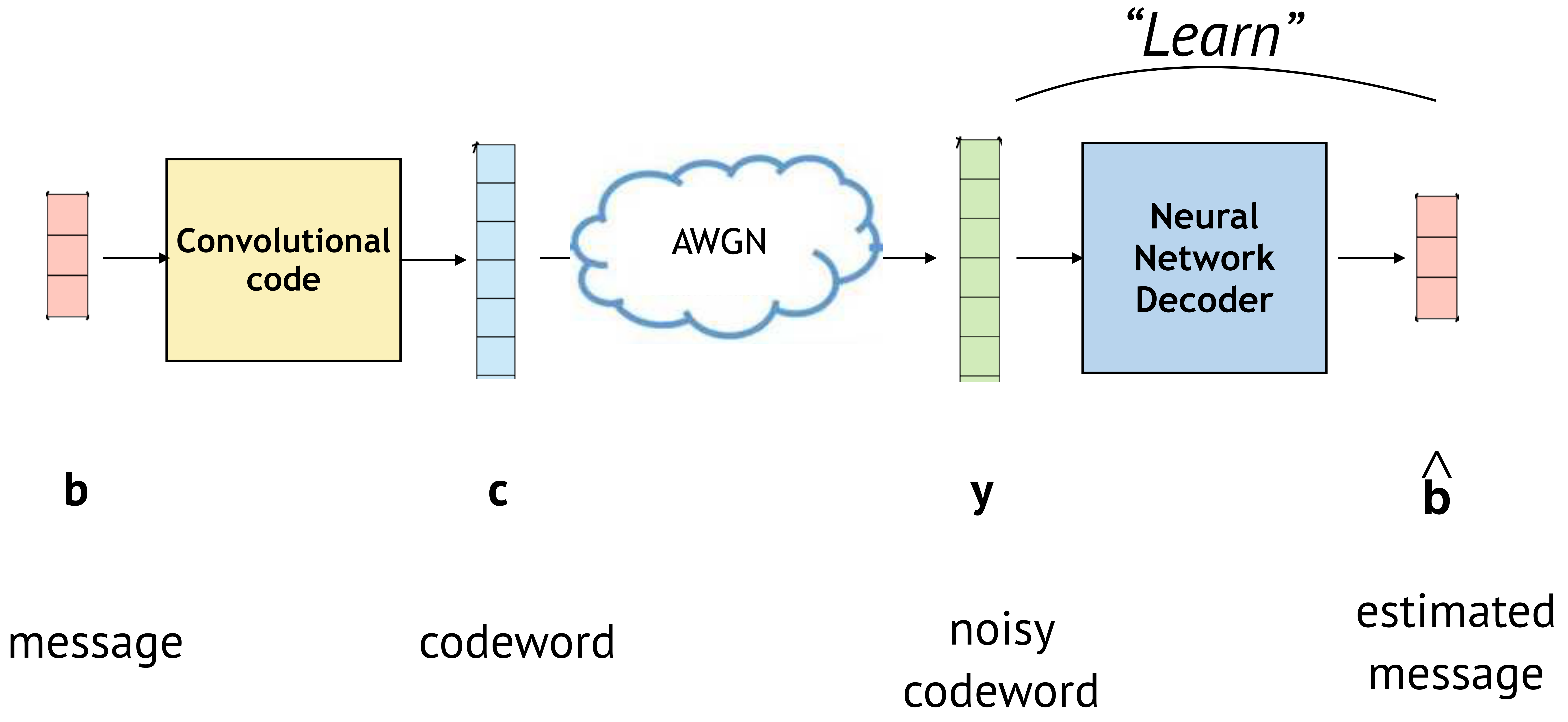
- Neural networks can emulate optimal decoders
  - Viterbi (Wang-Wicker '96)
  - BCJR (Sazli-Icsik '07)



ANN Viterbi decoder (Wang-Wicker'96)

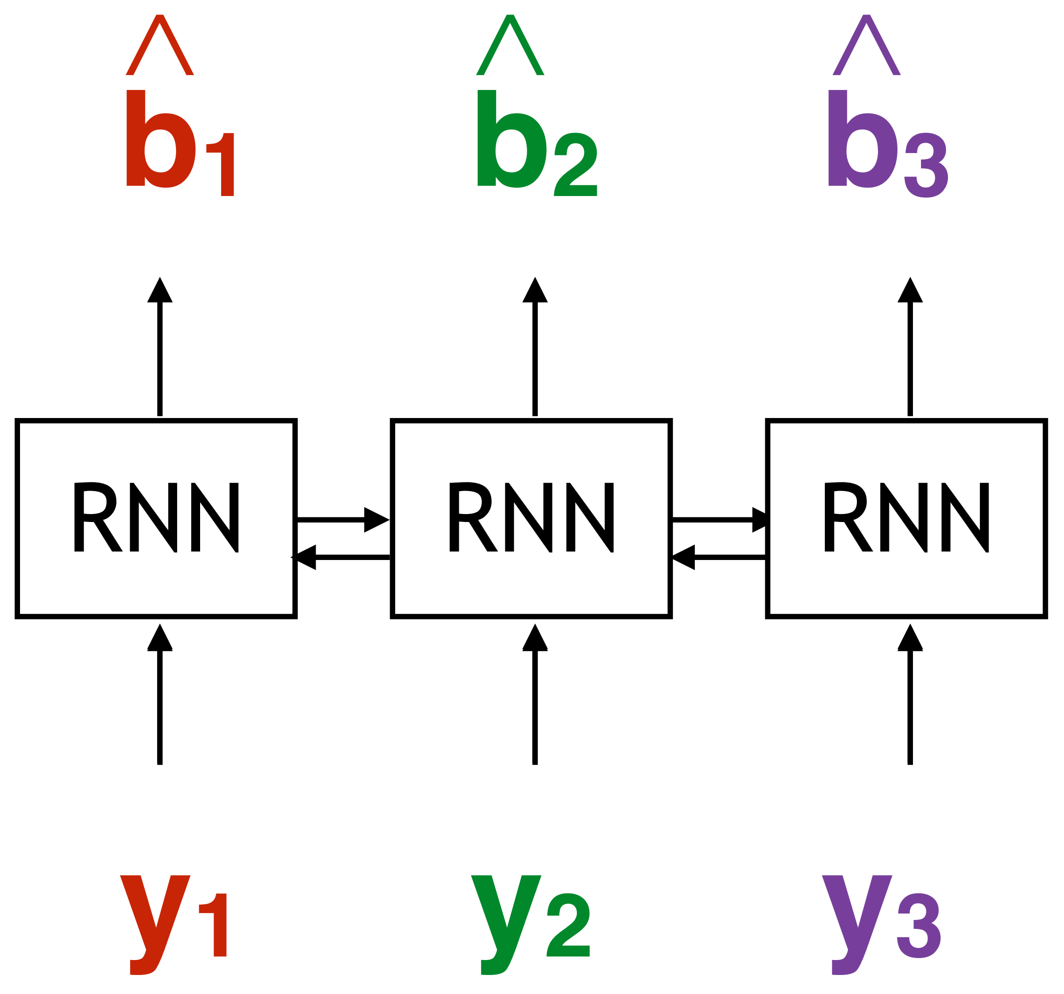
Convolutional codes under AWGN

- Can optimal decoders be learned from data alone?



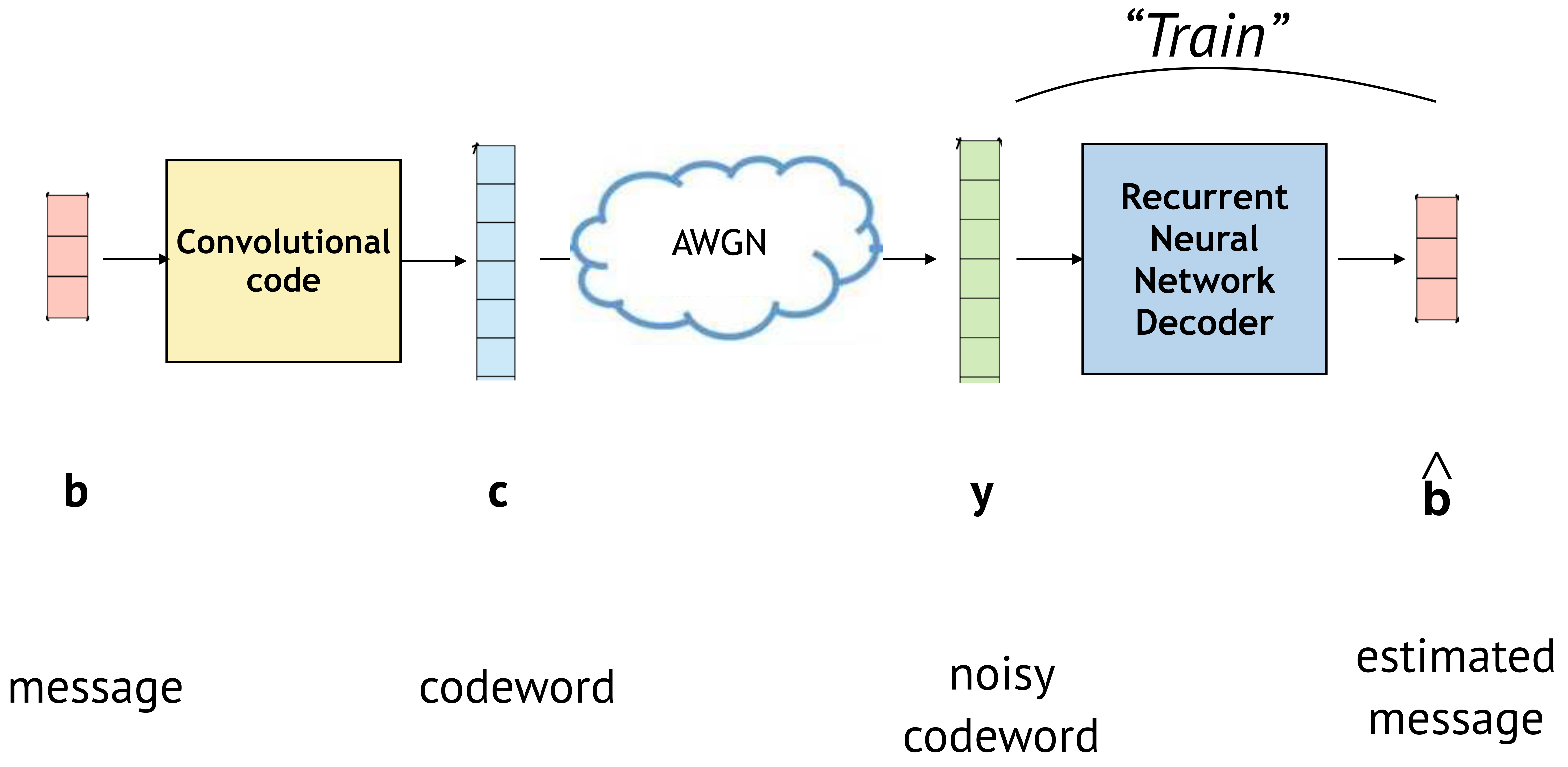
Decoder as a Recurrent Neural Network

- Maps $(y_1, y_2, y_3) \longrightarrow (\hat{b}_1, \hat{b}_2, \hat{b}_3)$ via bi-directional RNN



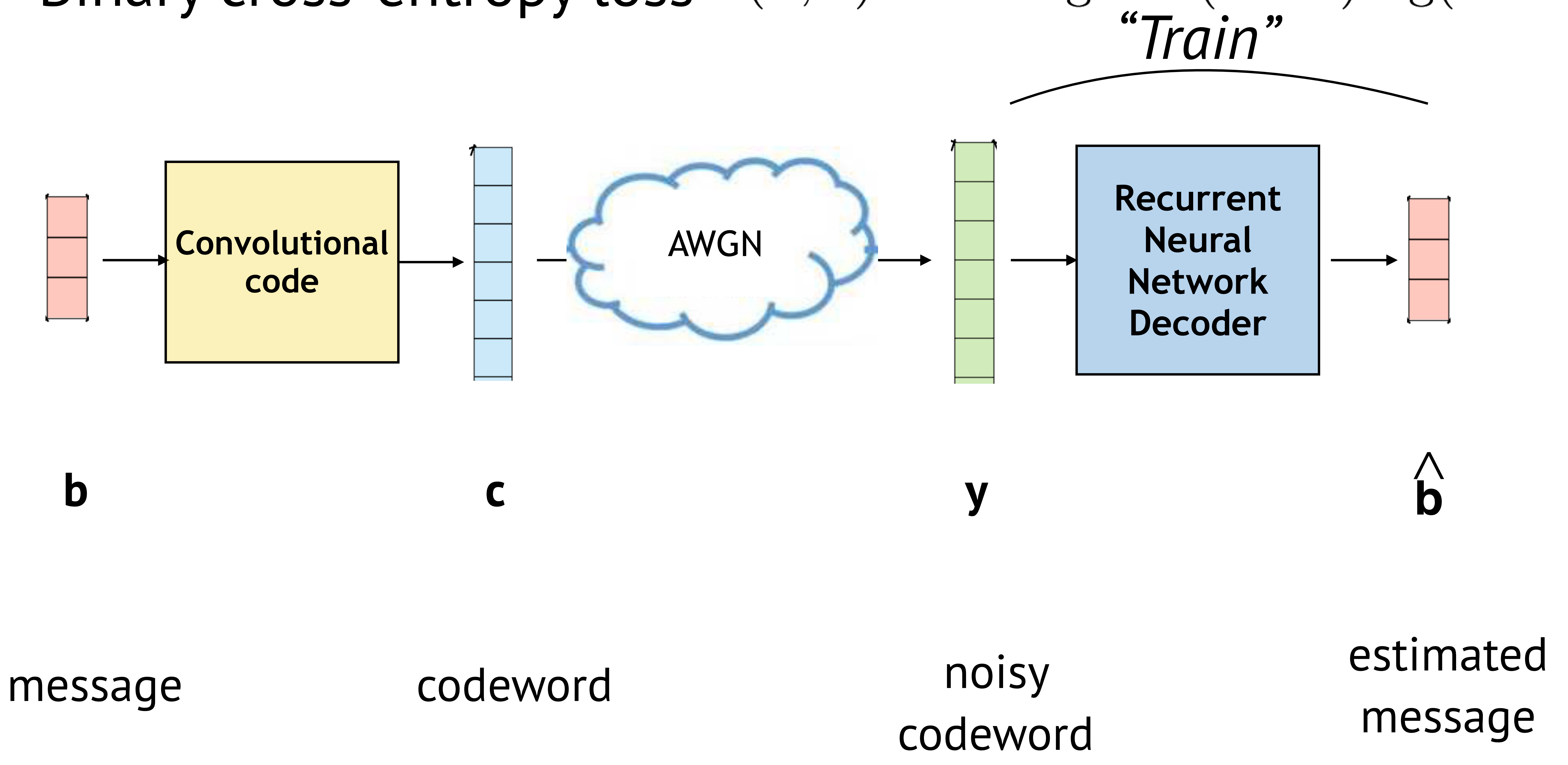
Training

- Supervised training with (noisy codeword **y**, message **b**)



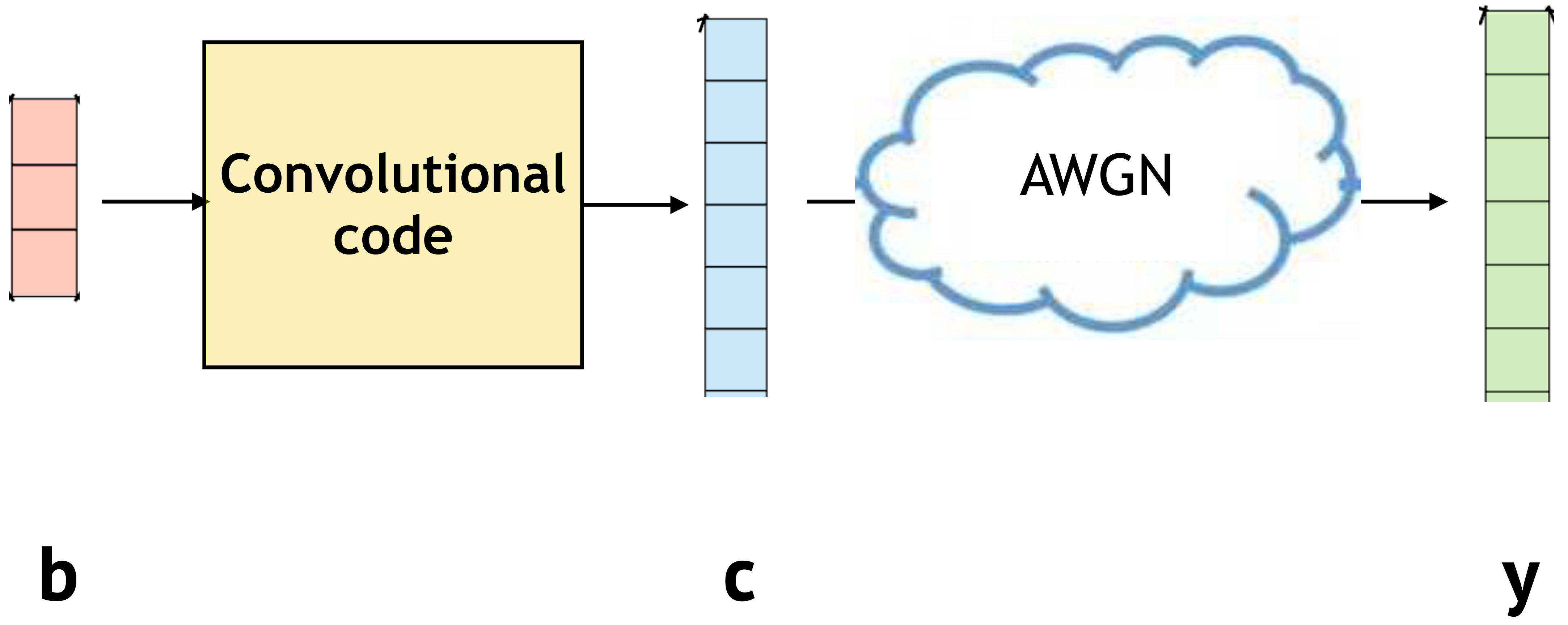
Training

- Supervised training with (noisy codeword **y**, message **b**)
- Binary cross-entropy loss $\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = -\mathbf{b} \log \hat{\mathbf{b}} - (1 - \mathbf{b}) \log(1 - \hat{\mathbf{b}})$



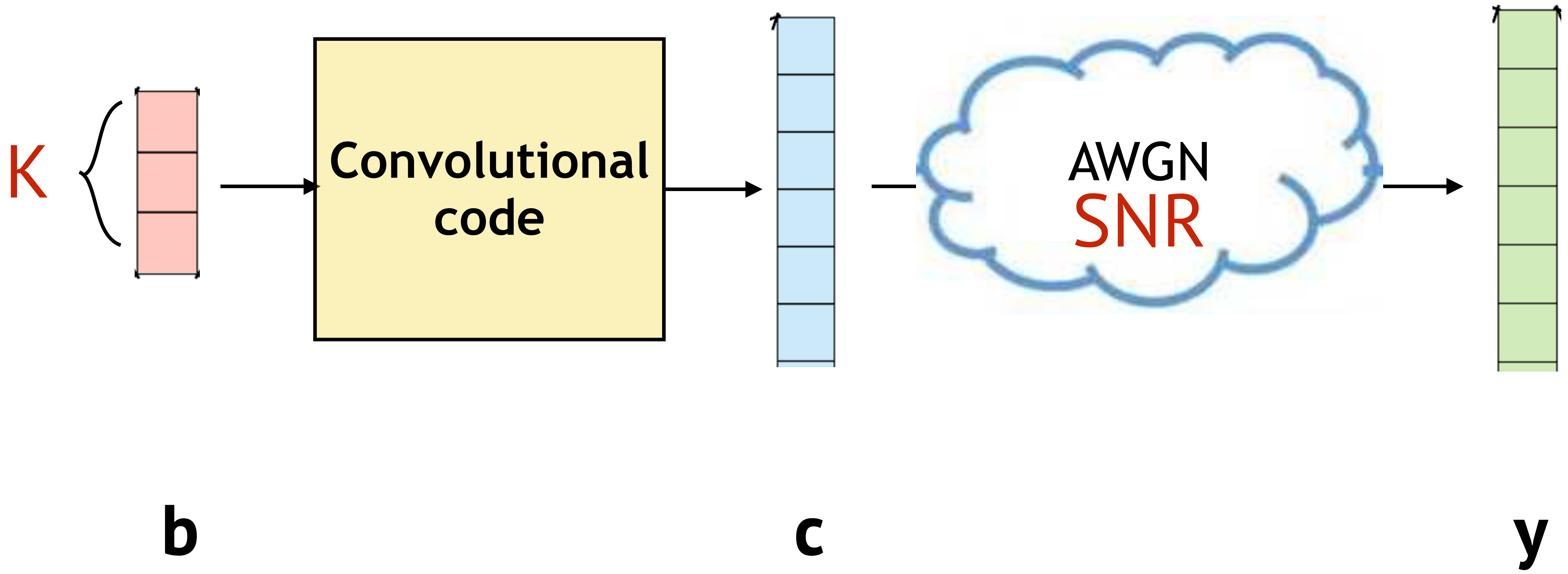
Choice of training examples

- Generate training examples (message **b**, noisy codeword **y**)



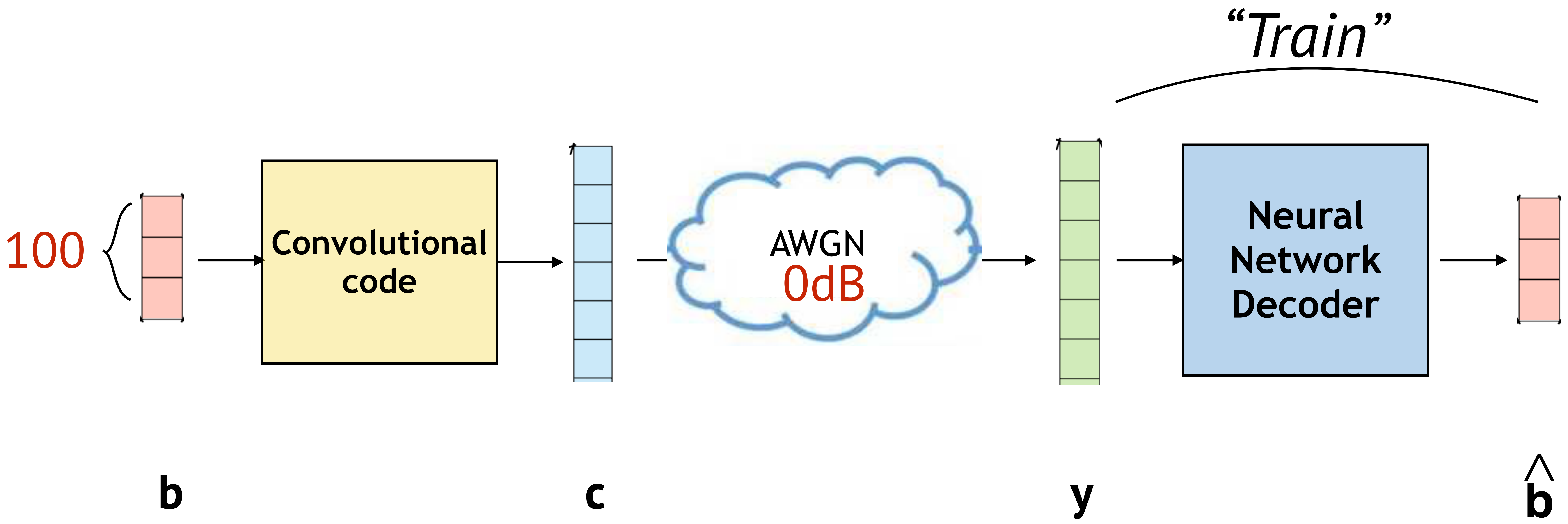
Choice of training examples

- Generate training examples (message **b**, noisy codeword **y**)
    - Length of message bits **b** = ($b_1, \ldots, b_K$)
    - SNR of the noisy codeword **y**



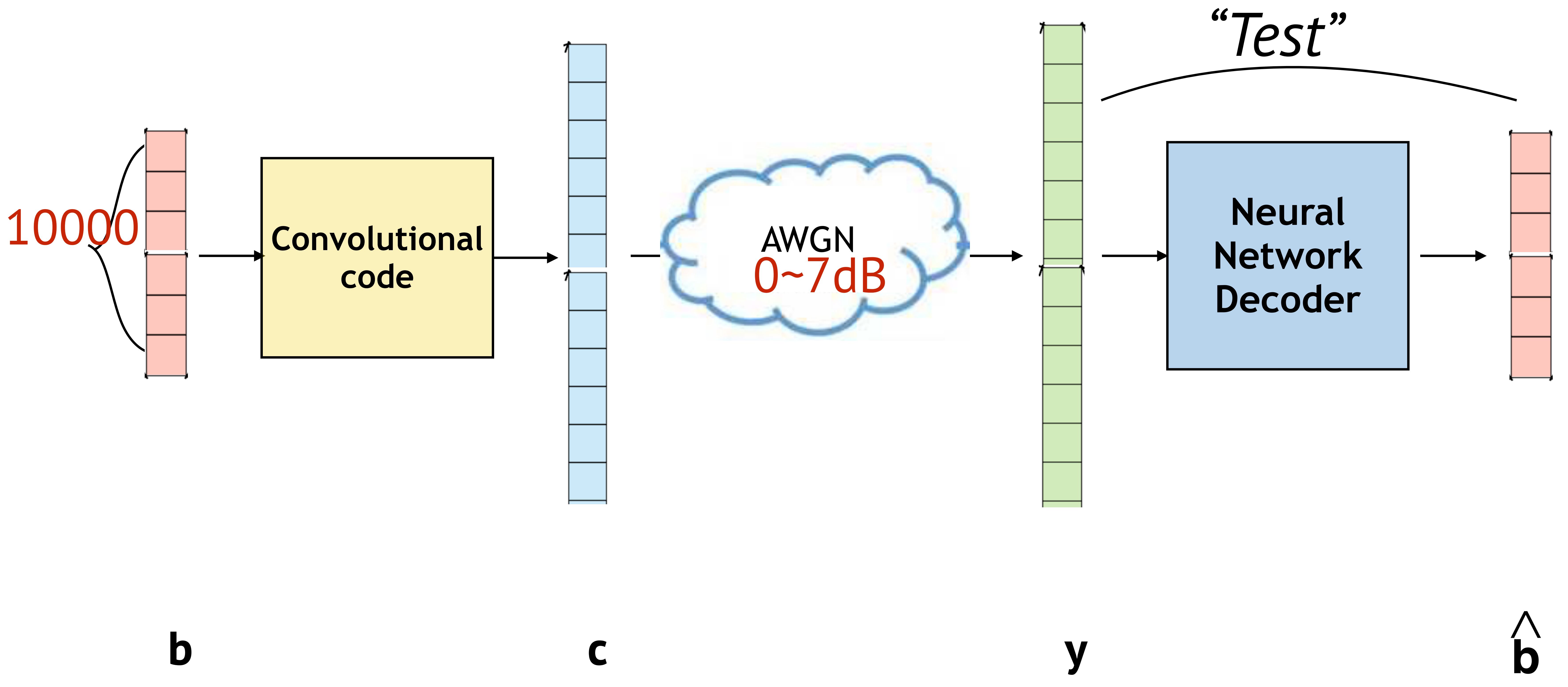
Choice of training examples

- Train at block length 100, fixed SNR (0dB)



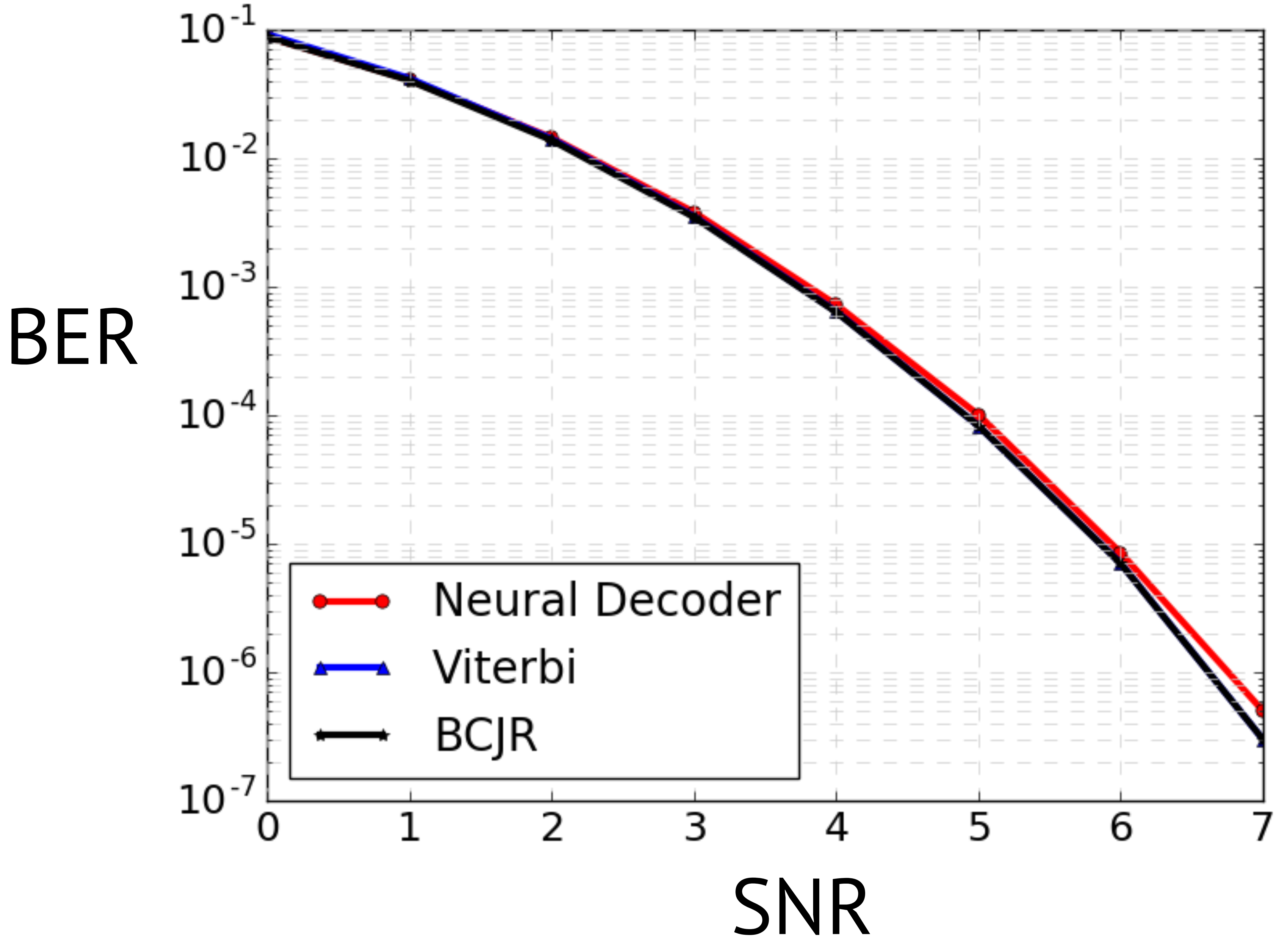
Strong generalization

- Train at block length 100, fixed SNR (0dB)
- Optimal performance for every test block lengths, SNR



Results: test block length 10000

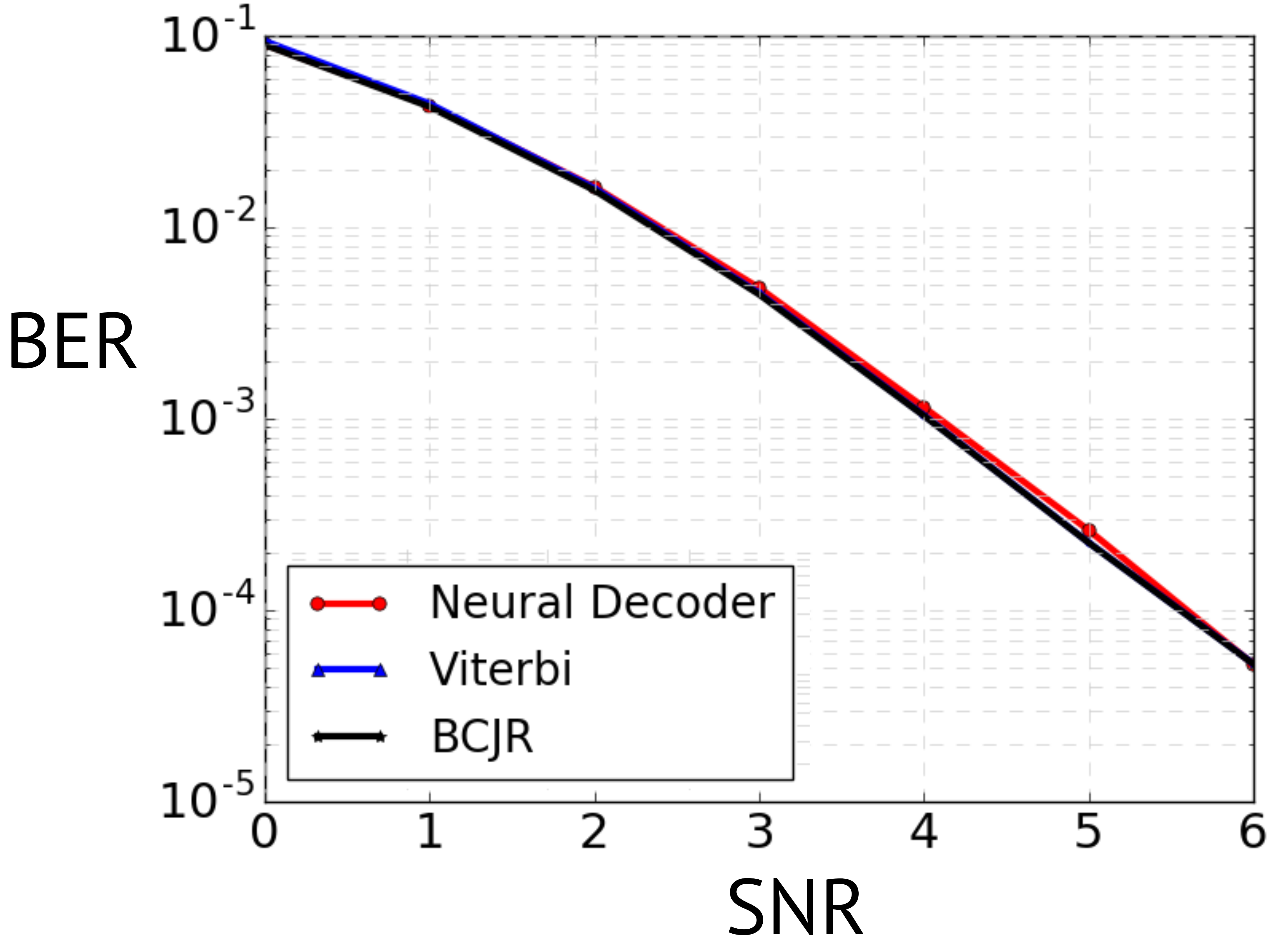
- Neural decoder is as reliable as an optimal decoder



Train: block length = 100, SNR=0dB

Results: test block length 100

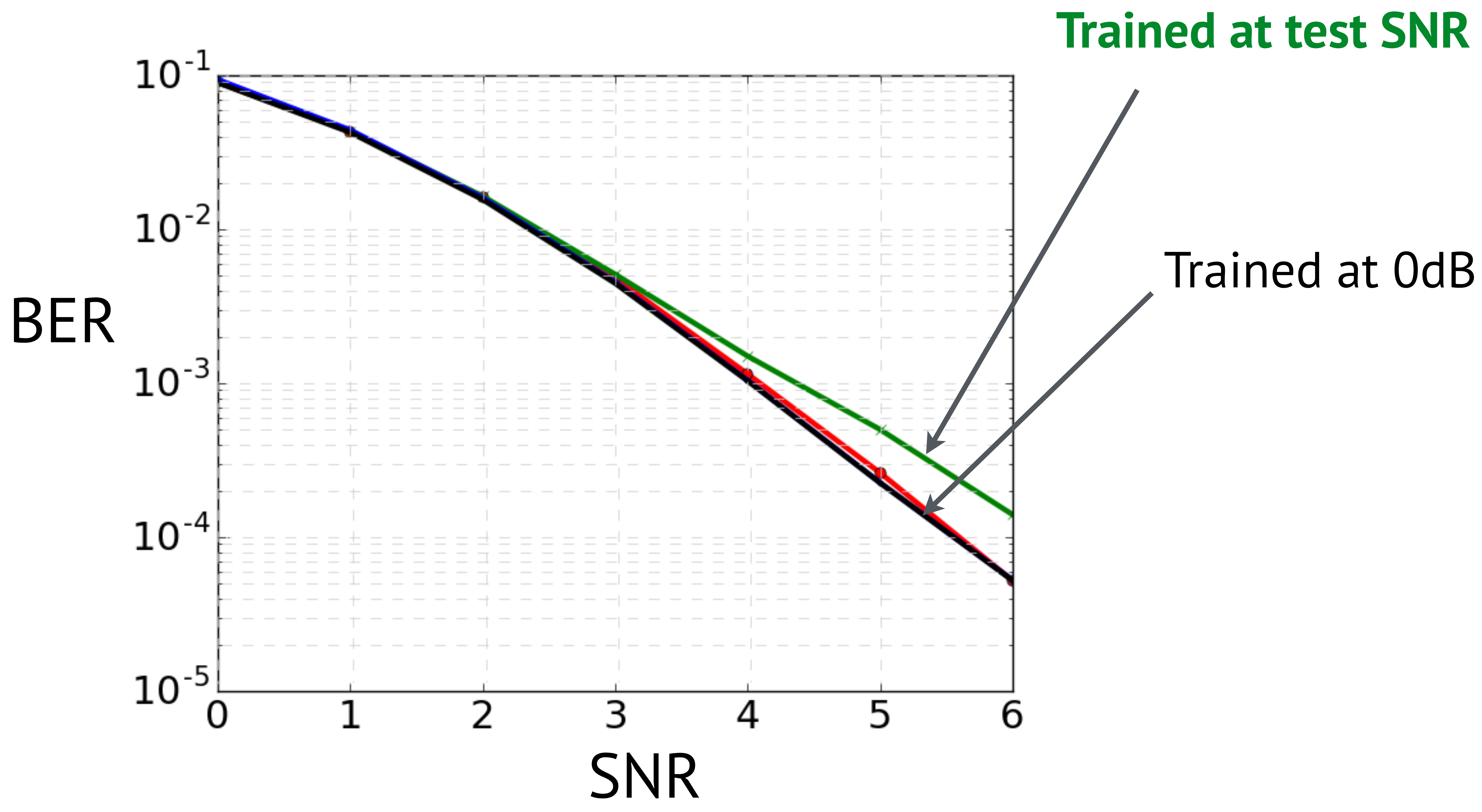
- Neural decoder is as reliable as an optimal decoder



Train: block length = 100, SNR=0dB

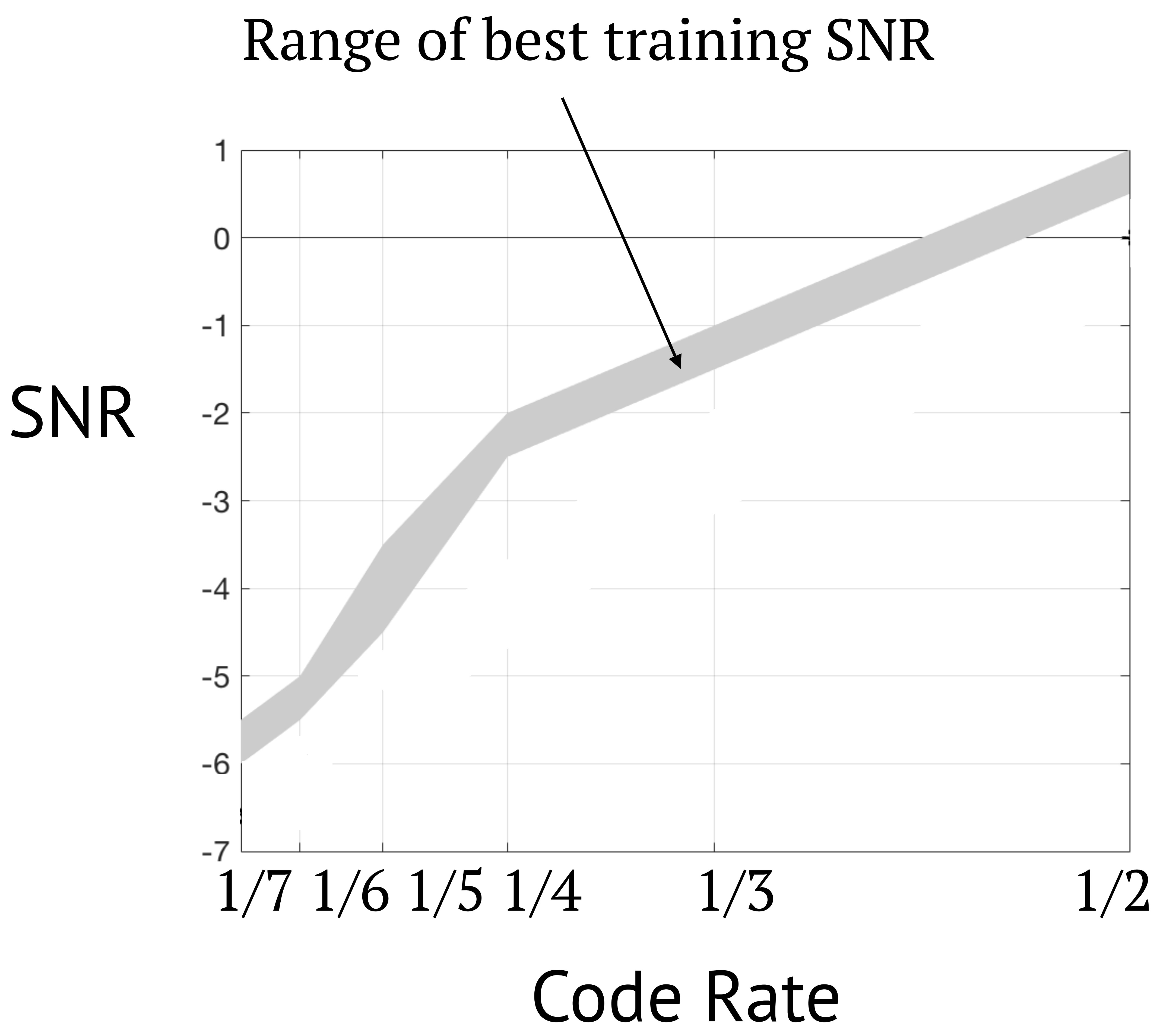
Choice of training examples

- What if we train with noisy codewords at test SNR?



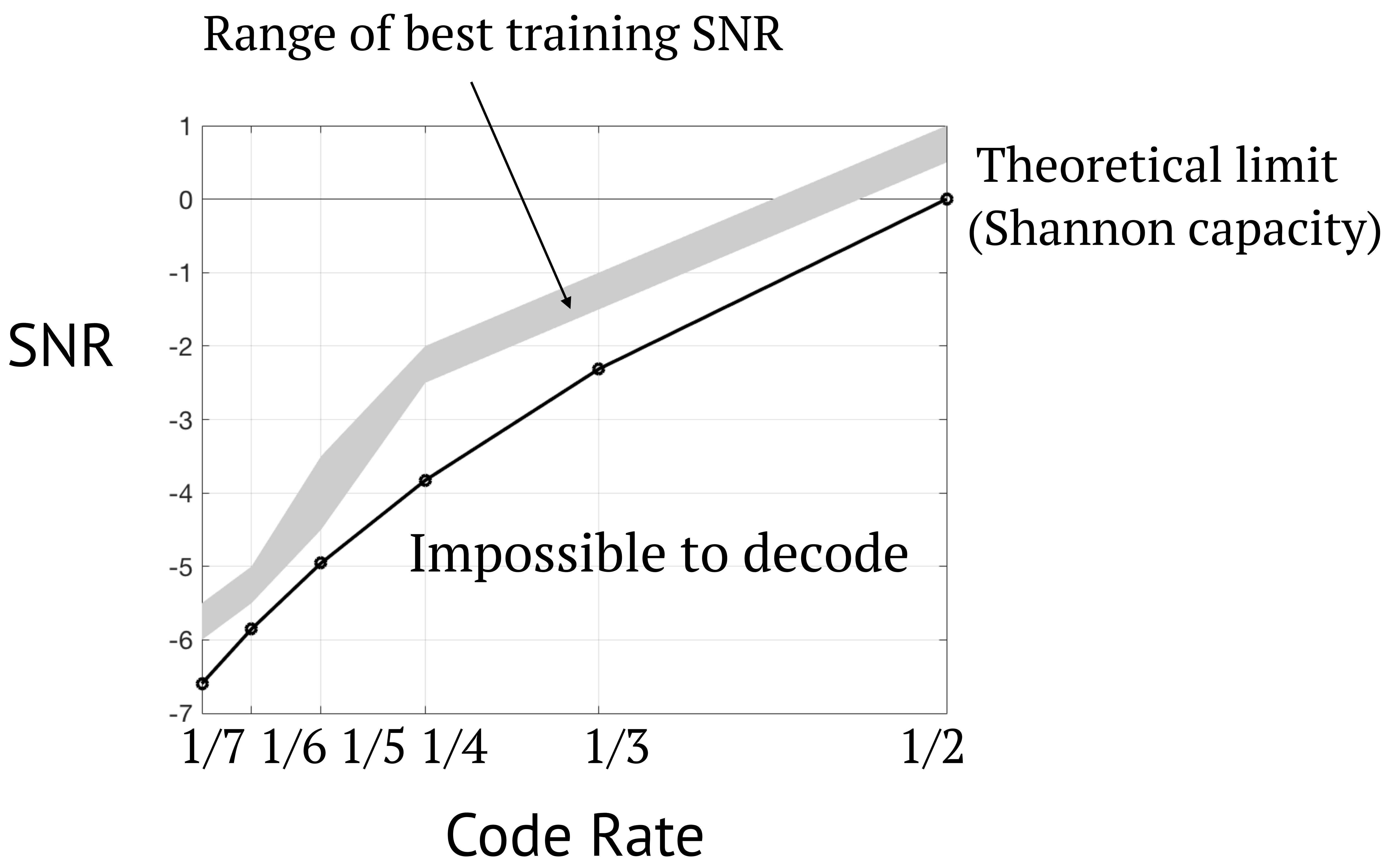
Choice of training examples

- Empirically find best training SNR for different code rates



Choice of training examples

- Hardest but decodable training examples



Hard training examples

- Idea of hardest but do-able training examples
  - Training with noisy examples
  - Applied to problems where training examples can be chosen

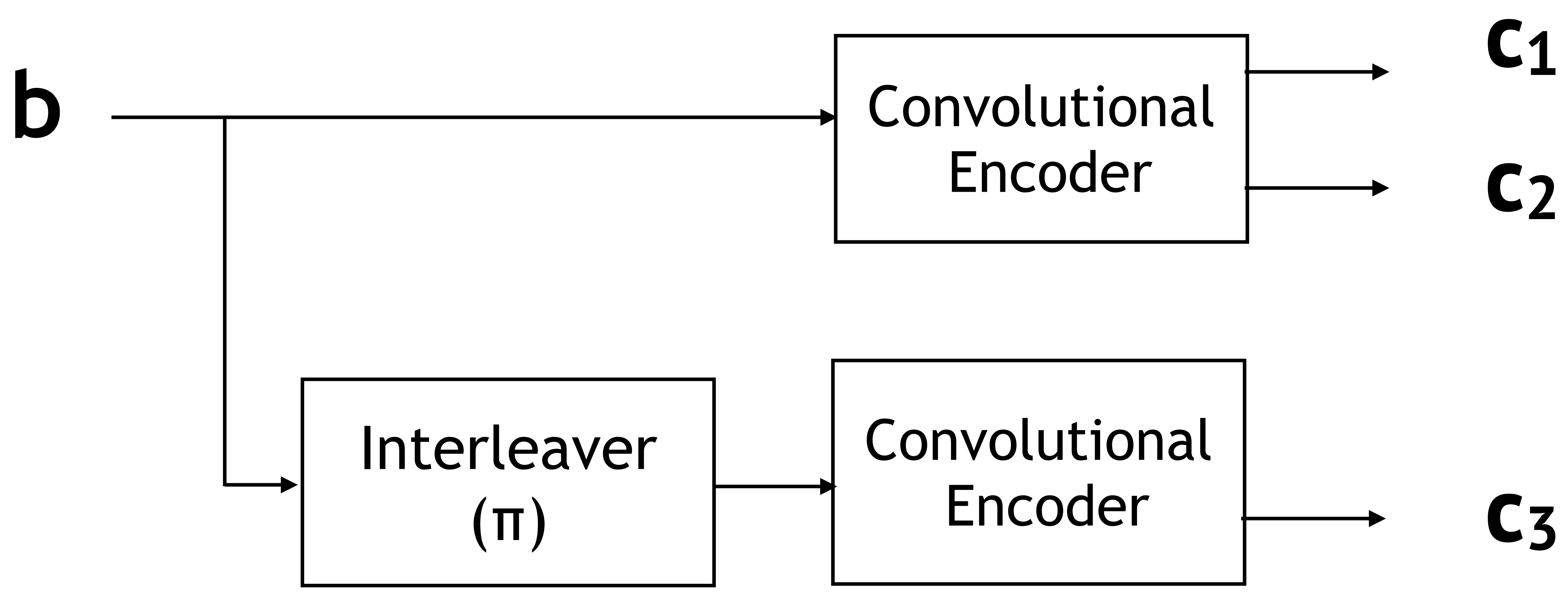
Outline - learning a decoder

1. Convolutional codes under AWGN channels

2. Turbo codes under AWGN channels

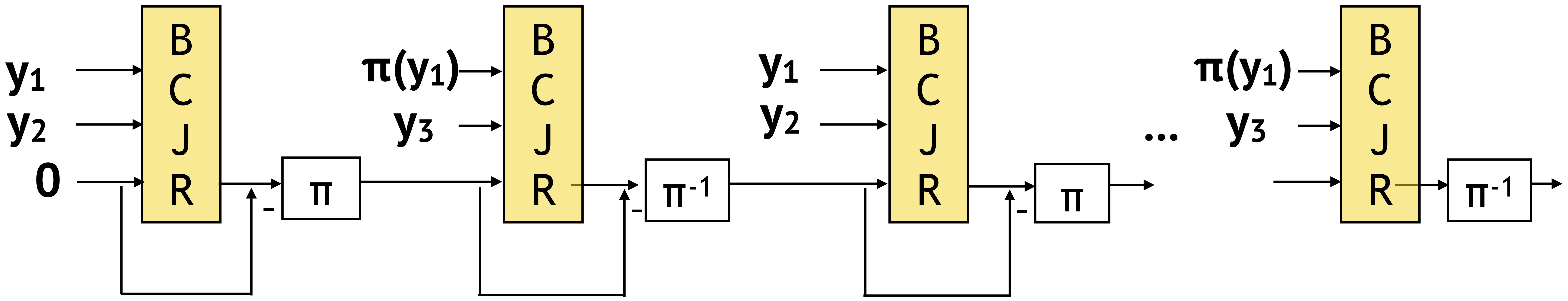
Turbo code

- Concatenate codewords from two convolutional encoders



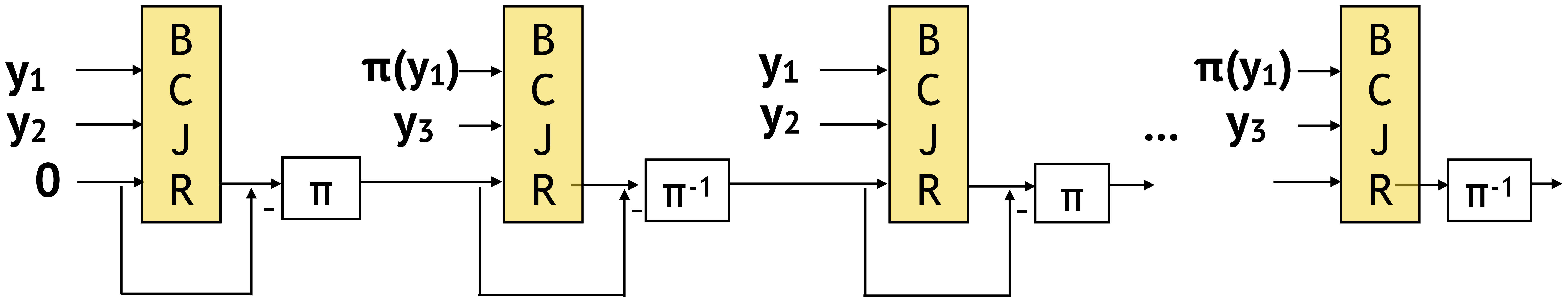
Belief propagation decoder

- Iterations of BCJR w. interleaver (π), de-interleaver (π^{-1})



Belief propagation decoder

- Iterations of BCJR w. interleaver (π), de-interleaver (π^{-1})
  - BCJR: maps (prior, noisy codewords) to posterior



Learning an iterative decoder for turbo

- Neural BCJR: If we could generate BCJR input-output pairs,
  can we learn a neural network based BCJR algorithm?

Learning an iterative decoder for turbo

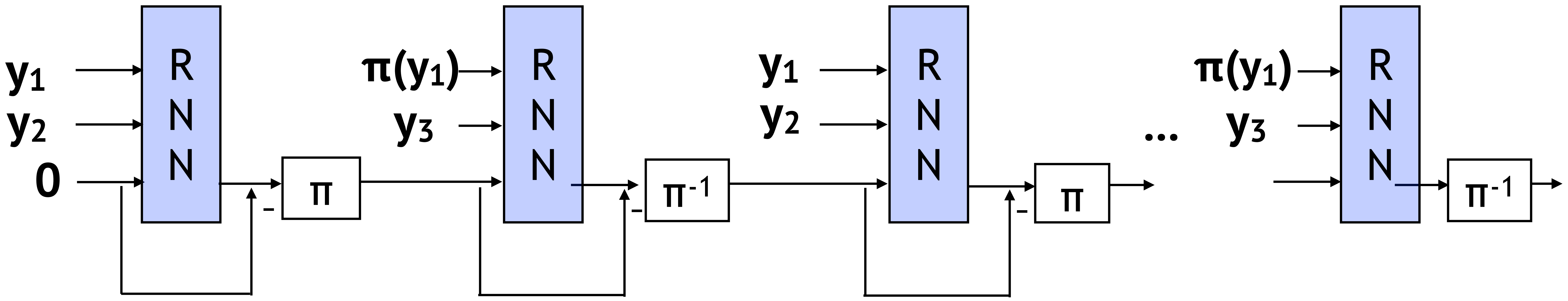
- Neural BCJR: If we could generate BCJR input-output pairs, can we learn a neural network based BCJR algorithm?
  - Would iteration of Neural BCJR decoders mimic the iterative decoder?

Learning an iterative decoder for turbo

- Neural BCJR: If we could generate BCJR input-output pairs,
  can we learn a neural network based BCJR algorithm?
  - Would iteration of Neural BCJR decoders mimic the iterative decoder?
- DeepTurbo: Can we learn an iterative decoder end-to-end?

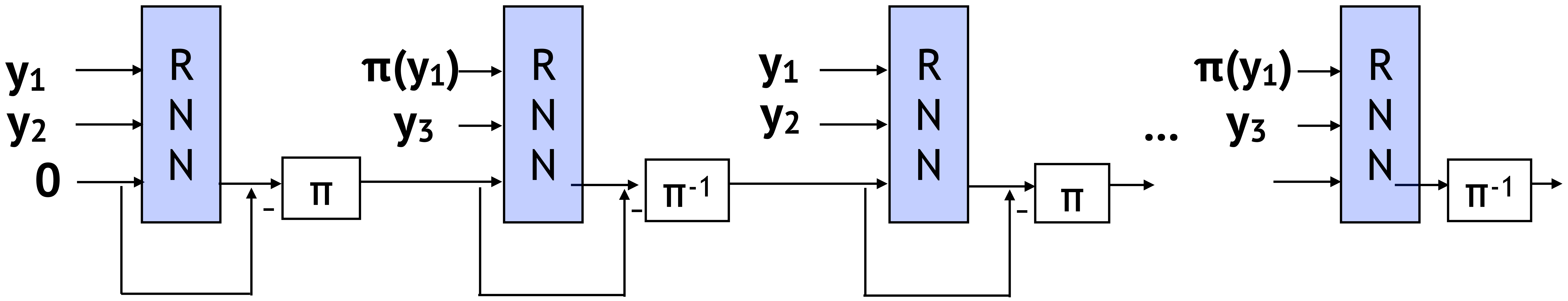
Decoder as a Recurrent Neural Network

- NeuralBCJR
  - Iterations of RNNs w. interleaver (π), de-interleaver (π^{-1})



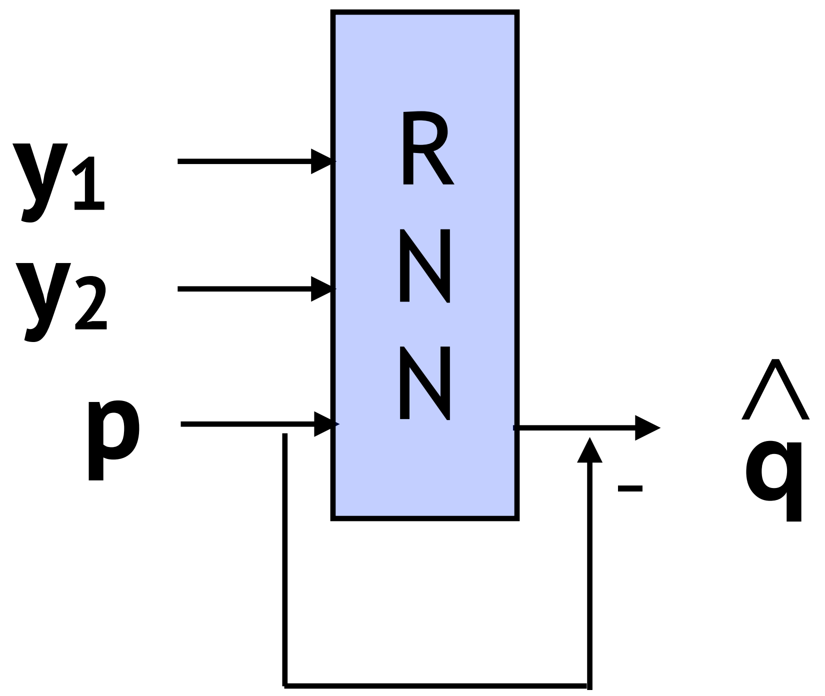
Decoder as a Recurrent Neural Network

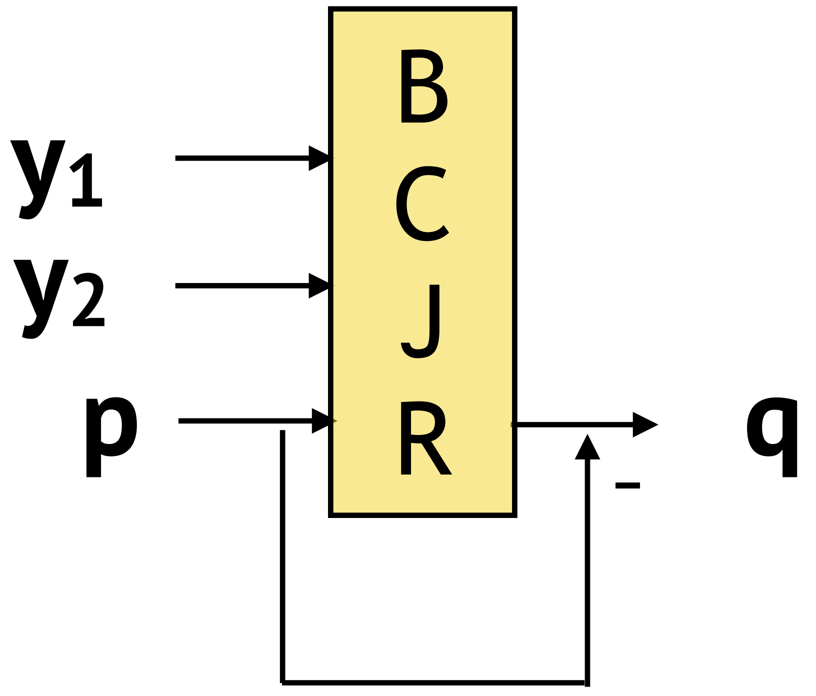
- NeuralBCJR
  - Each RNN trained to mimic BCJR



Training

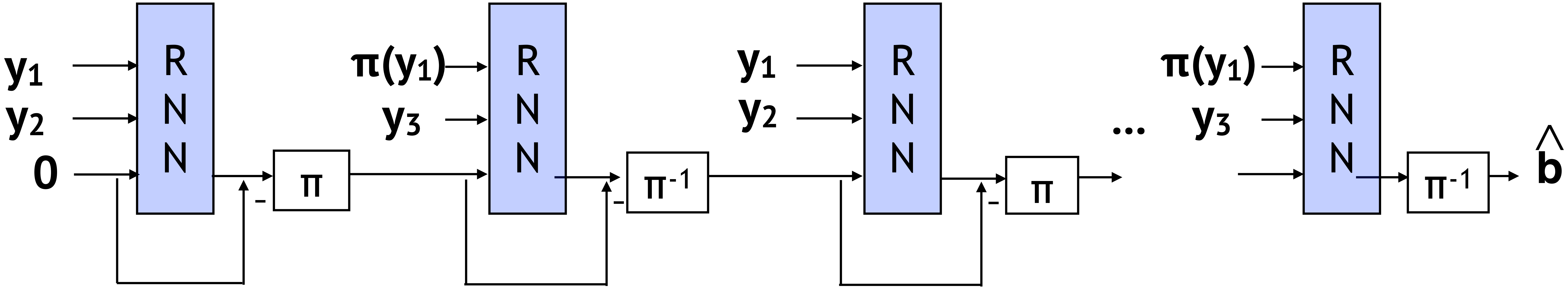
- Step 1: Neural BCJR learning
  - Supervised training with BCJR input-output under AWGN channels





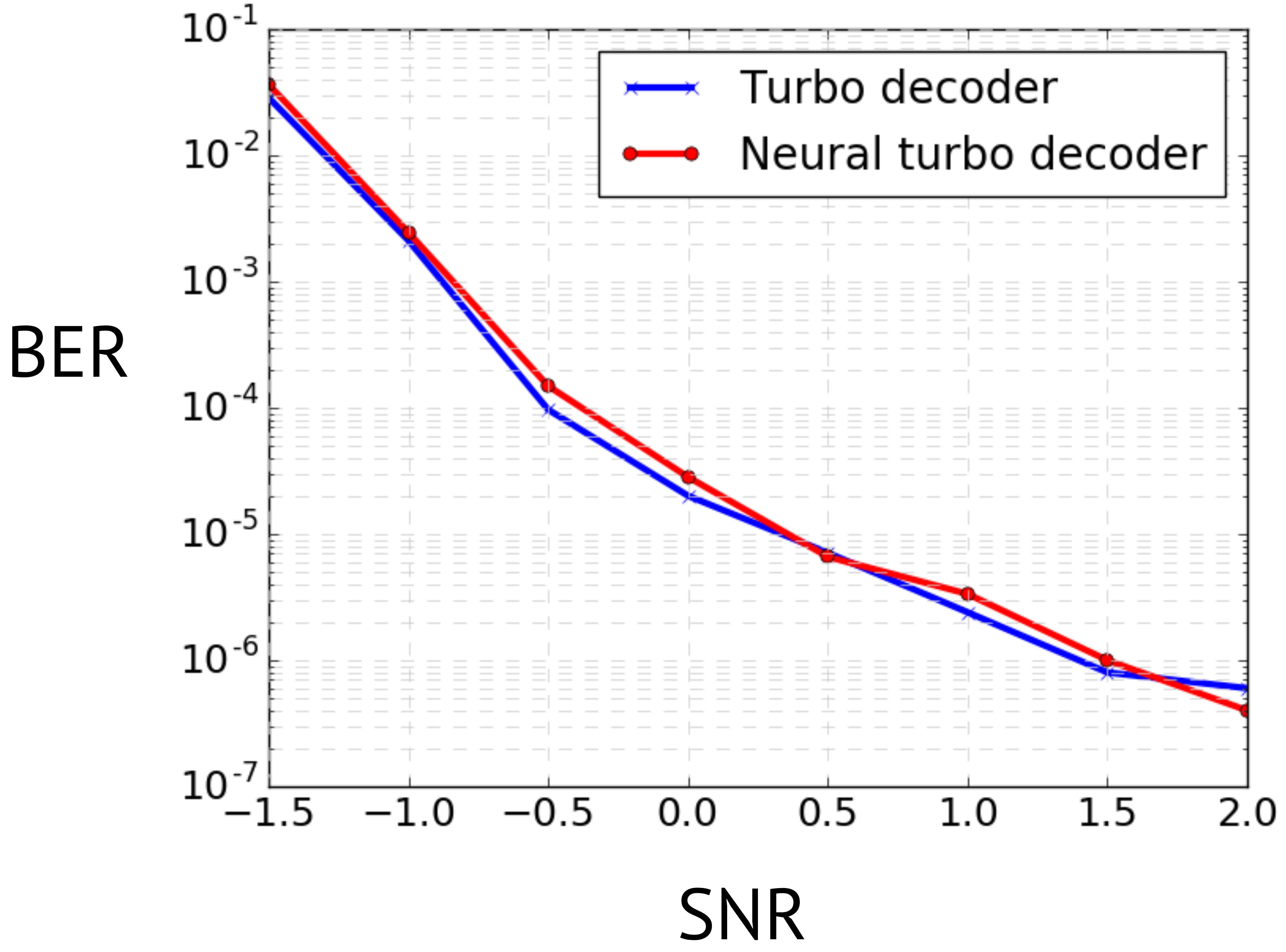
Training

- Step 2: E2E fine-tuning
  - Supervised training with (**y**,**b**) under AWGN channels



Decoding turbo codes: block length 1000

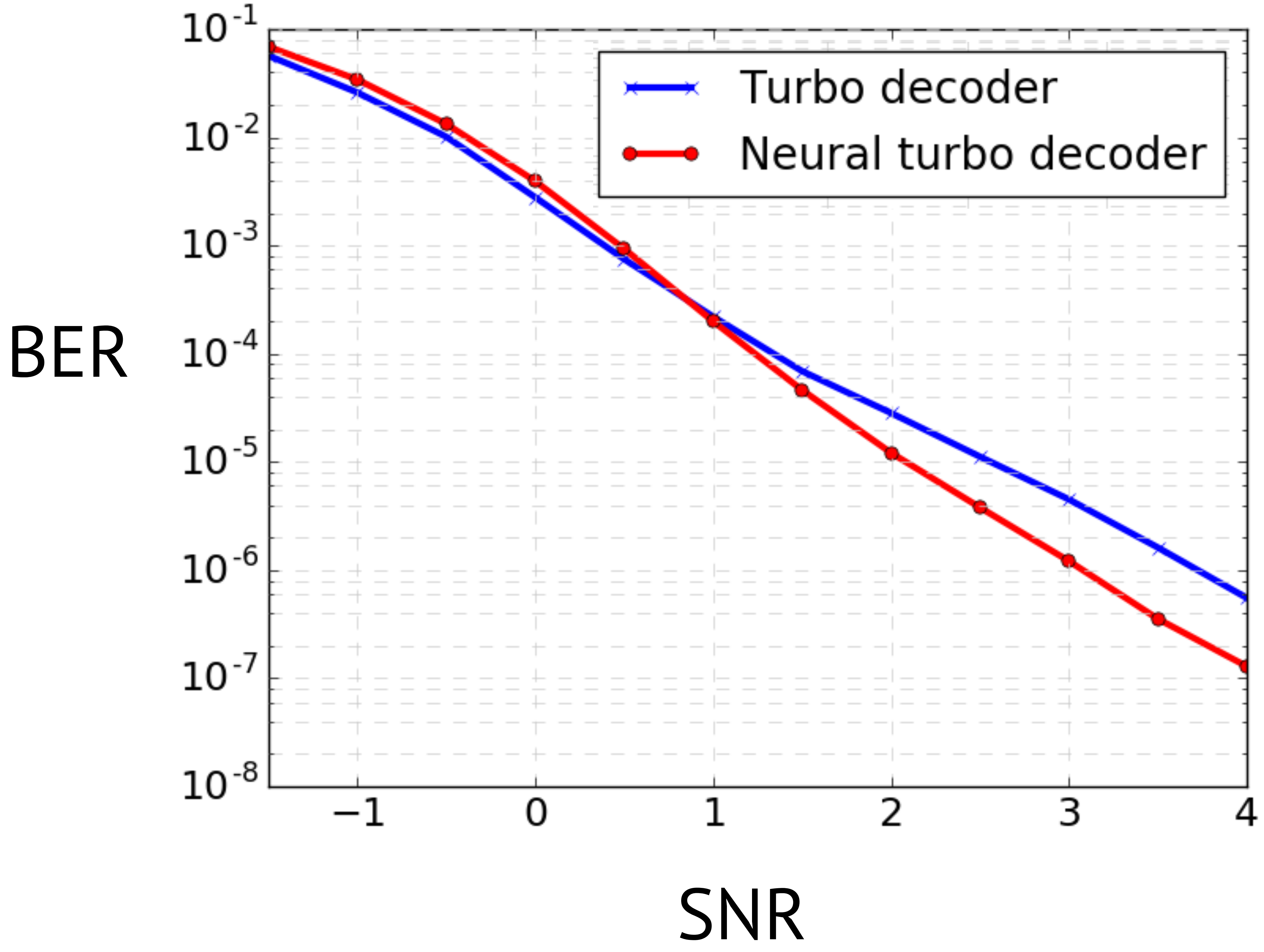
- Neural decoder is as reliable as traditional decoder



Rate 1/3 turbo code

Decoding turbo codes: block length 100

- Neural decoder is more reliable than traditional decoder



Rate 1/3 turbo code

Learning BP decoder from data alone

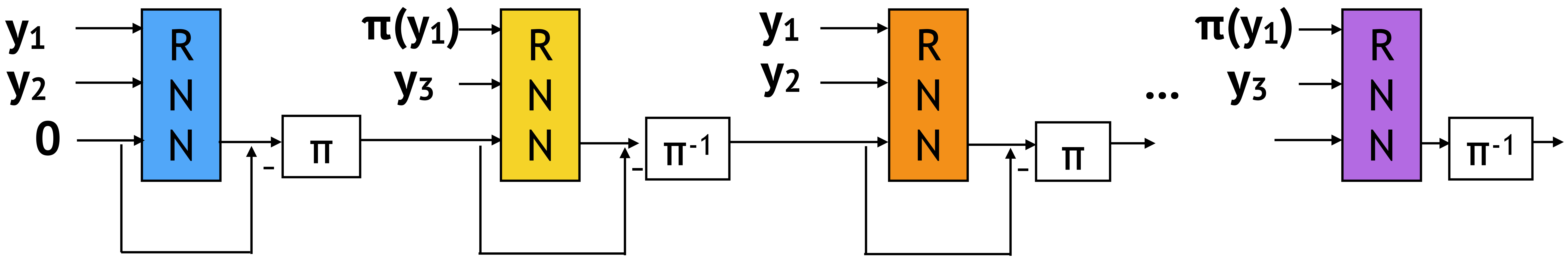
- What if BCJR input-output values are not available?

Learning BP decoder from data alone

- What if BCJR input-output values are not available?
  - DeepTurbo: learning the BP decoder from data alone (Jiang-Kim-Asnani-Kannan-Oh-Viswanath SPAWC '19)

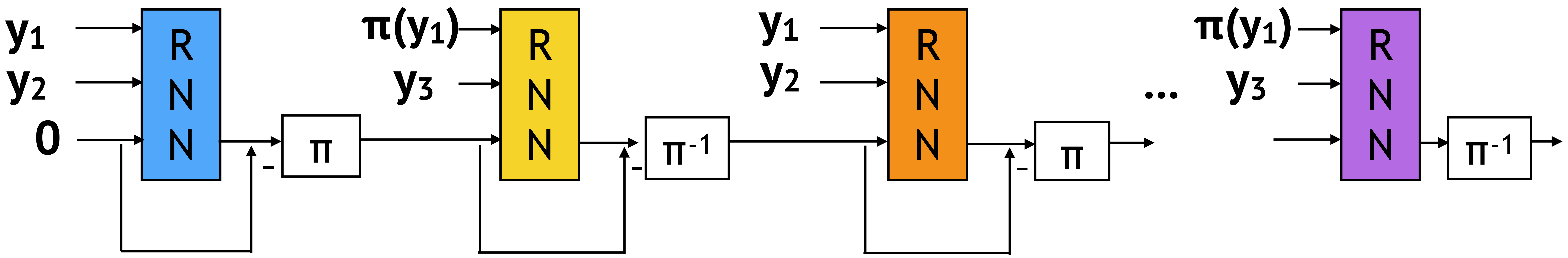
Learning BP decoder from data alone

- DeepTurbo: Iteration of 12 layers of bi-RNNs
  - De-coupled RNNs
  - Belief vectors



Training

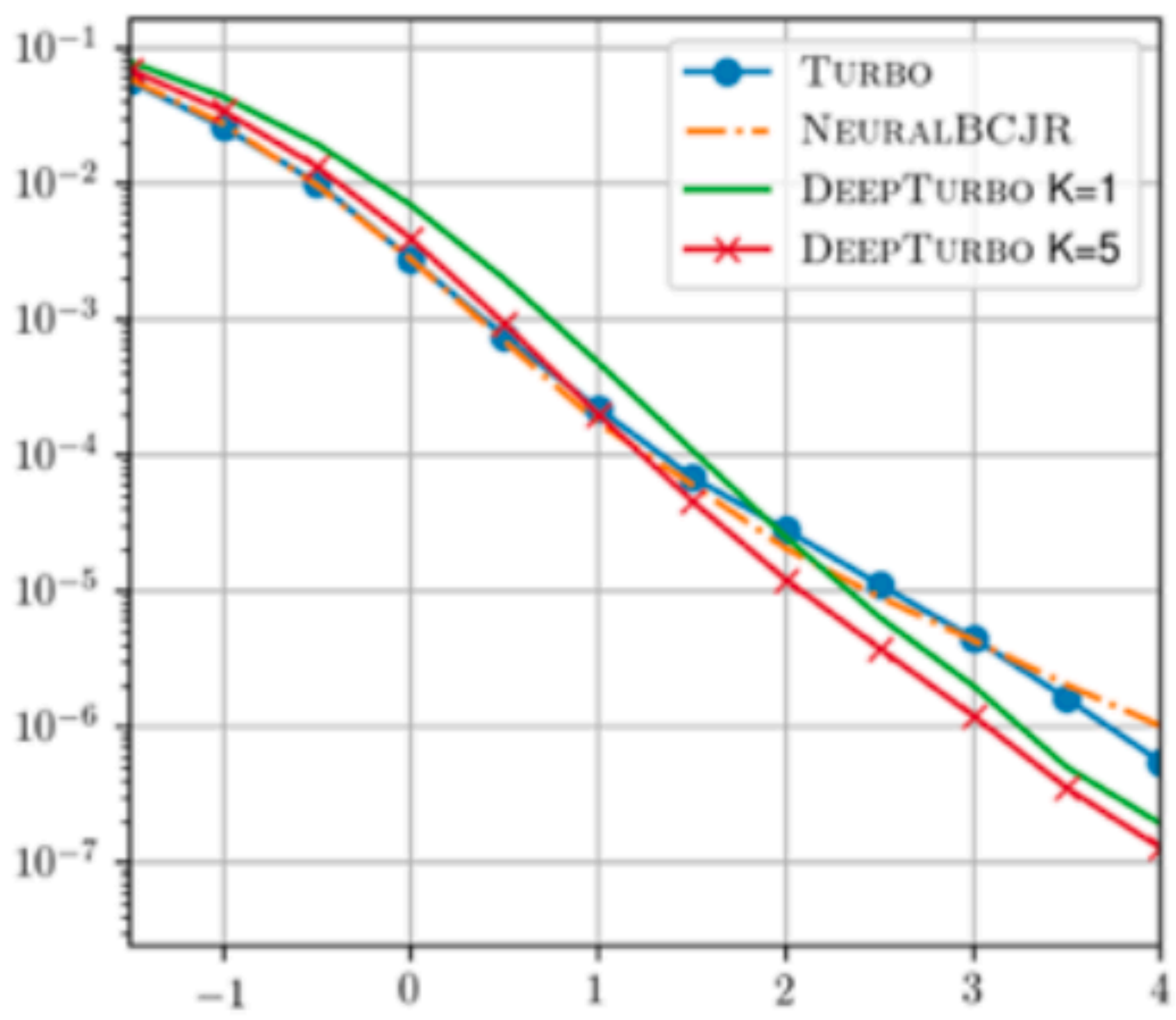
- Supervised training with (**y**,**b**) under AWGN channels
  - Training with hardest but do-able examples



Neural BCJR vs. DeepTurbo

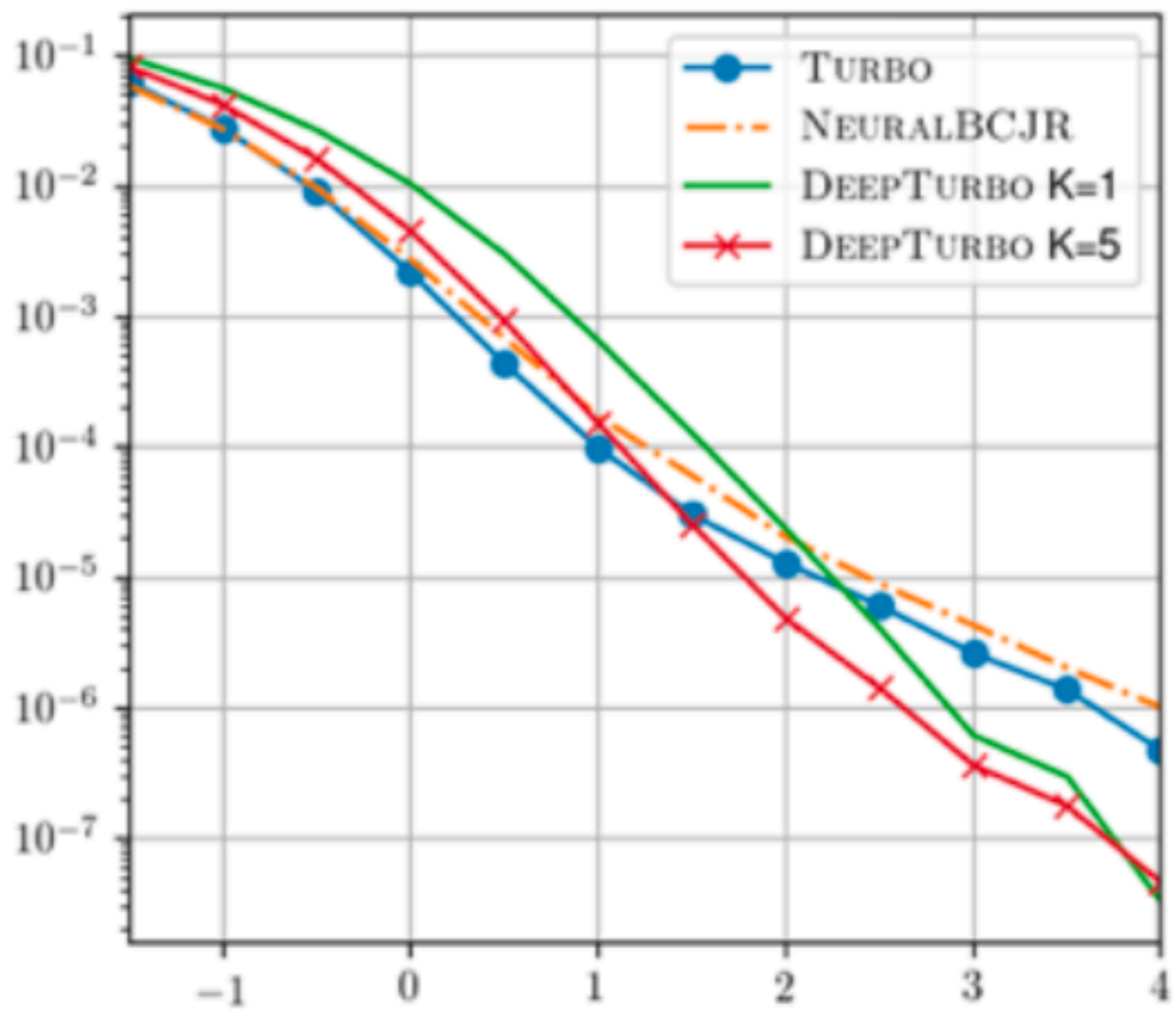
- Comparable performance

BER



SNR

Rate 1/3 turbo code w/ memory 2 (100 bits)



SNR

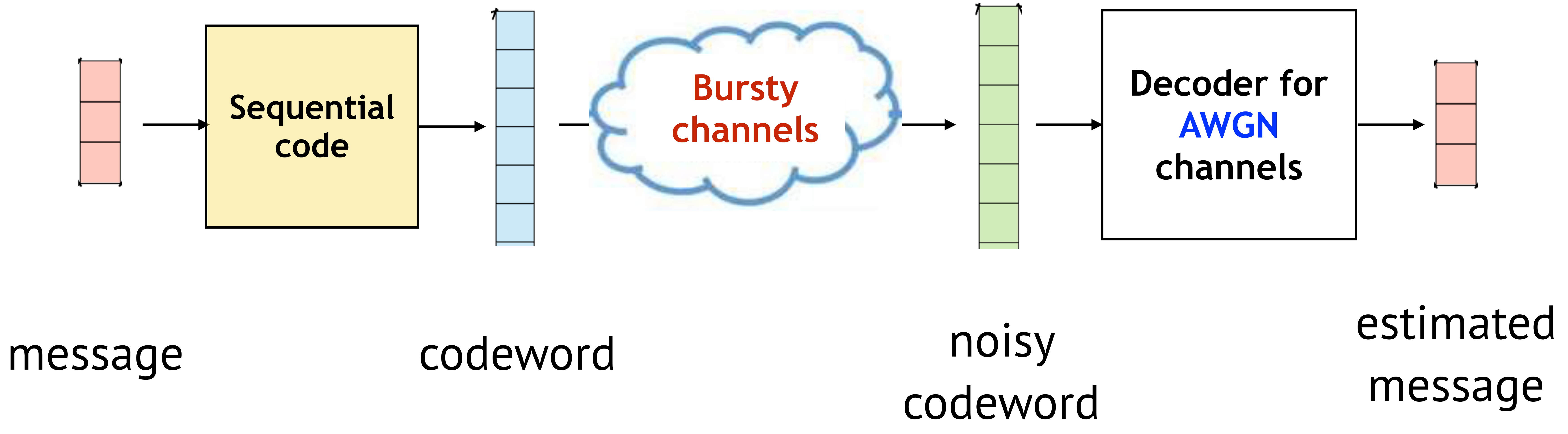
Rate 1/3 turbo code w/ memory 3 (100 bits)

Outline - learning a decoder

1. Convolutional codes under AWGN channels
2. Turbo codes under AWGN channels
3. Turbo codes under bursty channels

Robustness

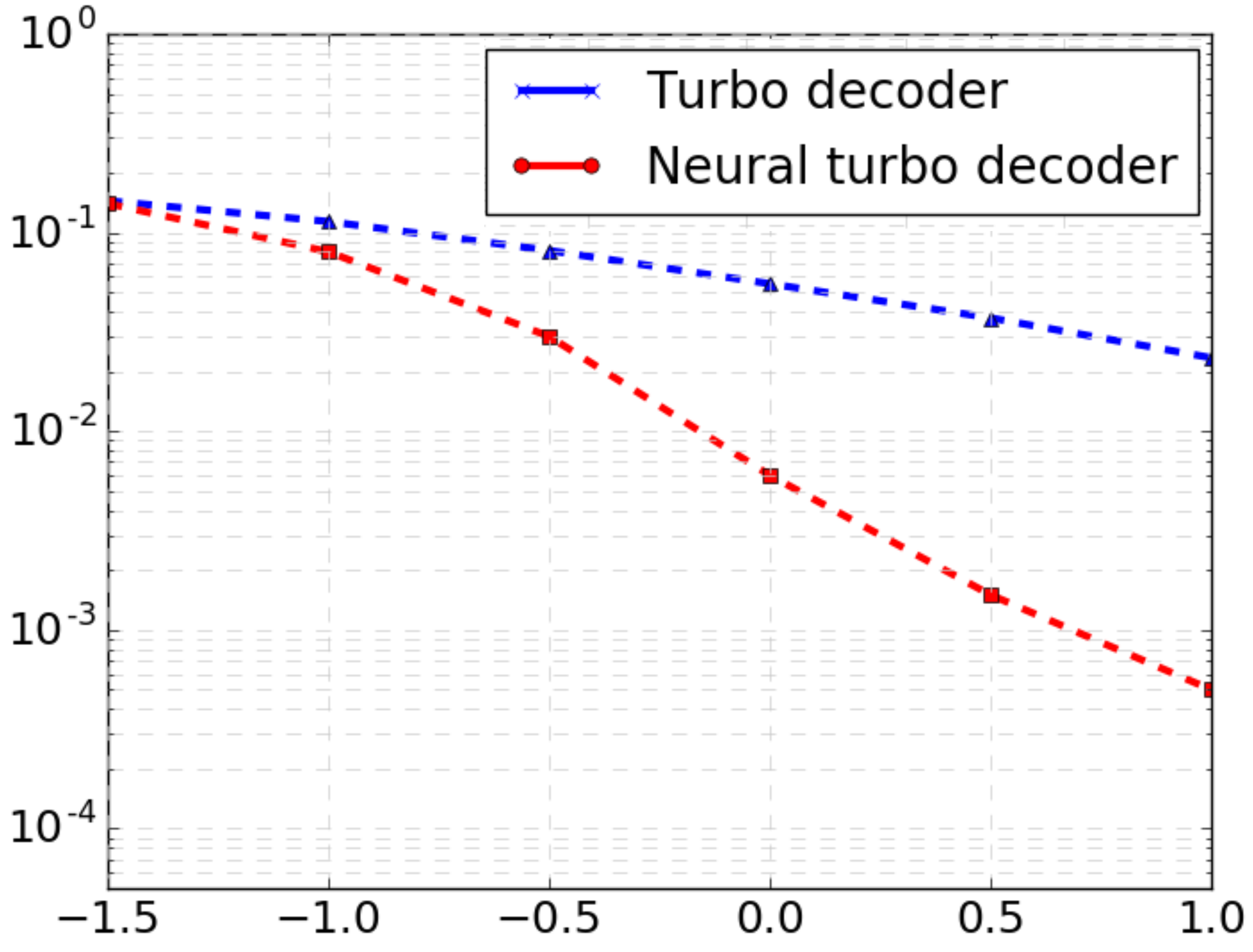
- Decoders for AWGN channel => test under bursty channels



Robustness

- Neural decoder for AWGN is more reliable under bursty

BER

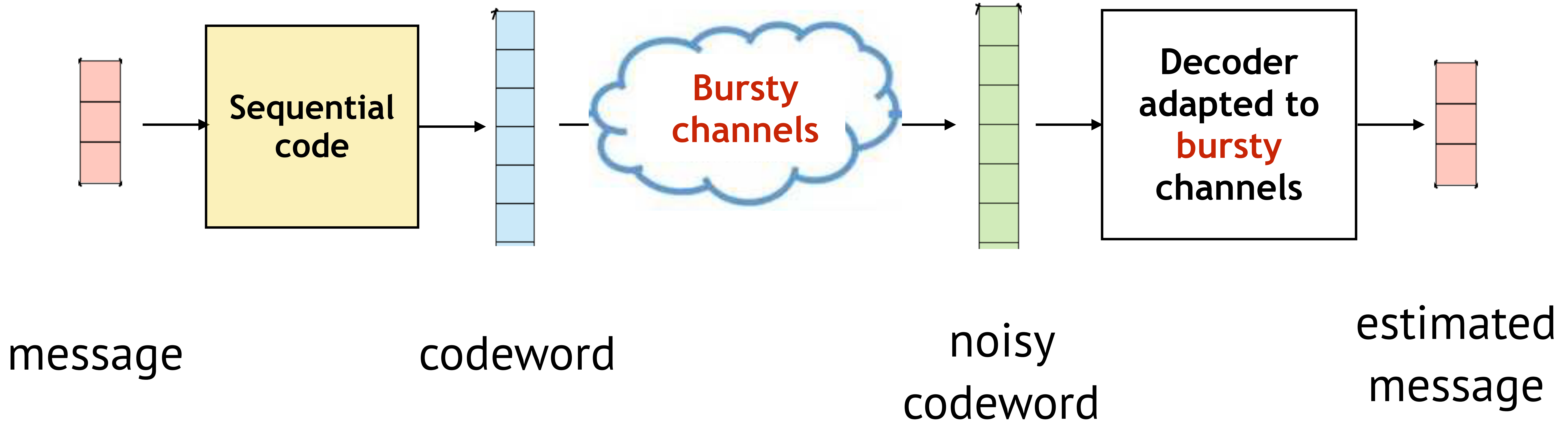


SNR of bursty noise channel

Rate 1/3, 1000 bits

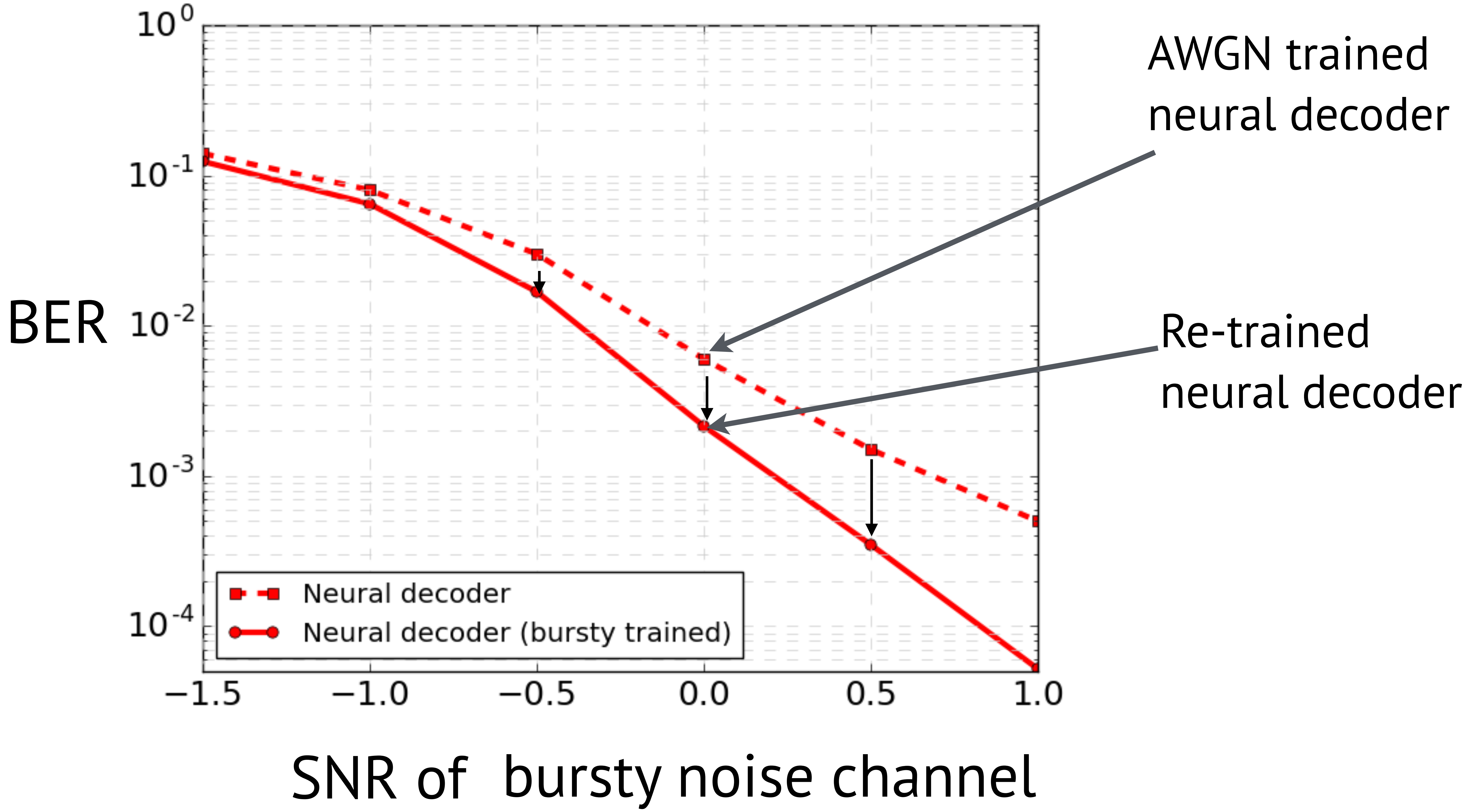
Adaptivity

- Decoder adapted to actual test channels



Adaptivity

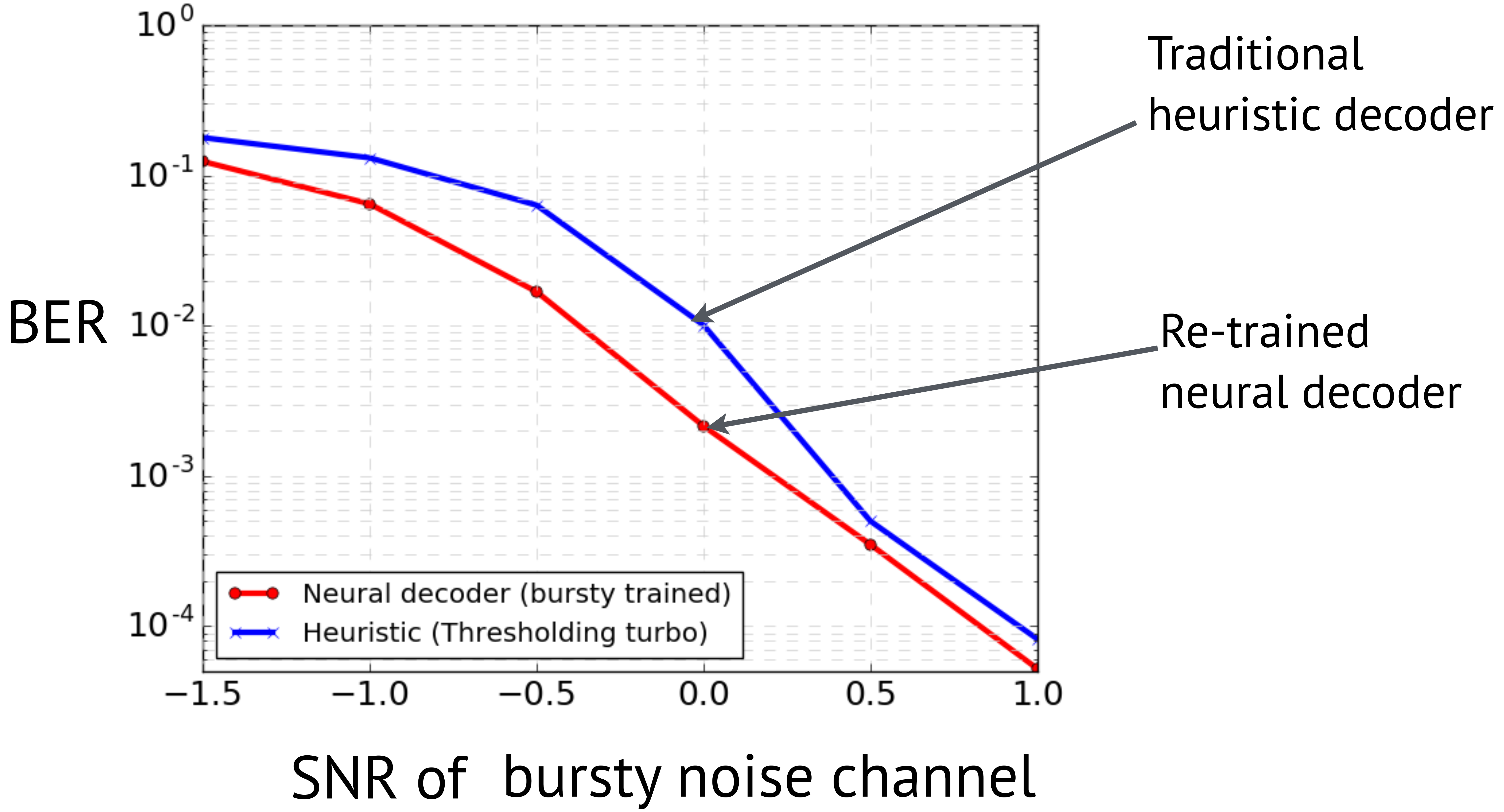
- Re-train neural decoder with bursty examples



Rate 1/3 turbo code, block length 1000

Adaptivity

- Re-trained neural decoder more reliable than heuristic dec.

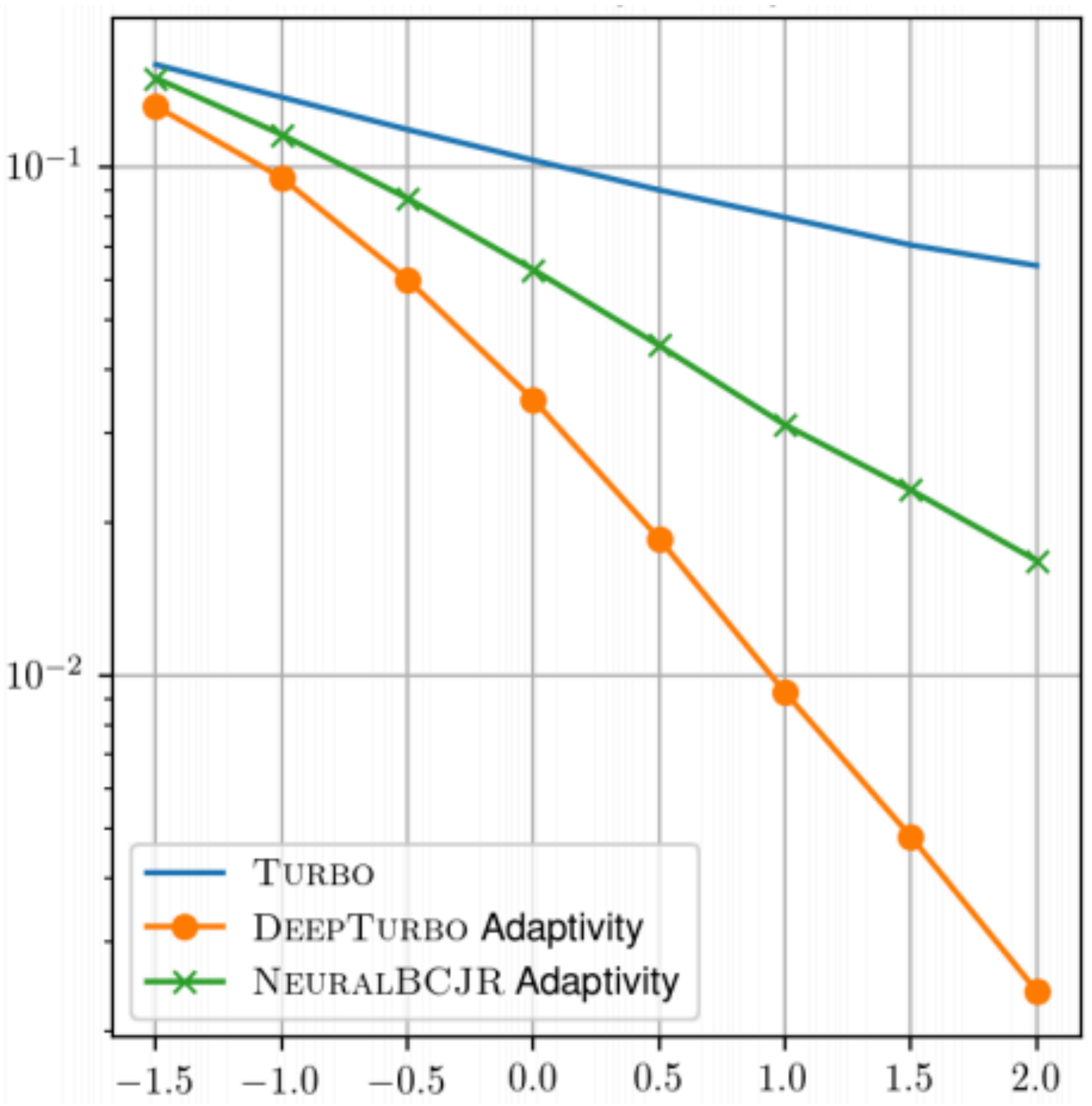


Rate 1/3 turbo code, block length 1000

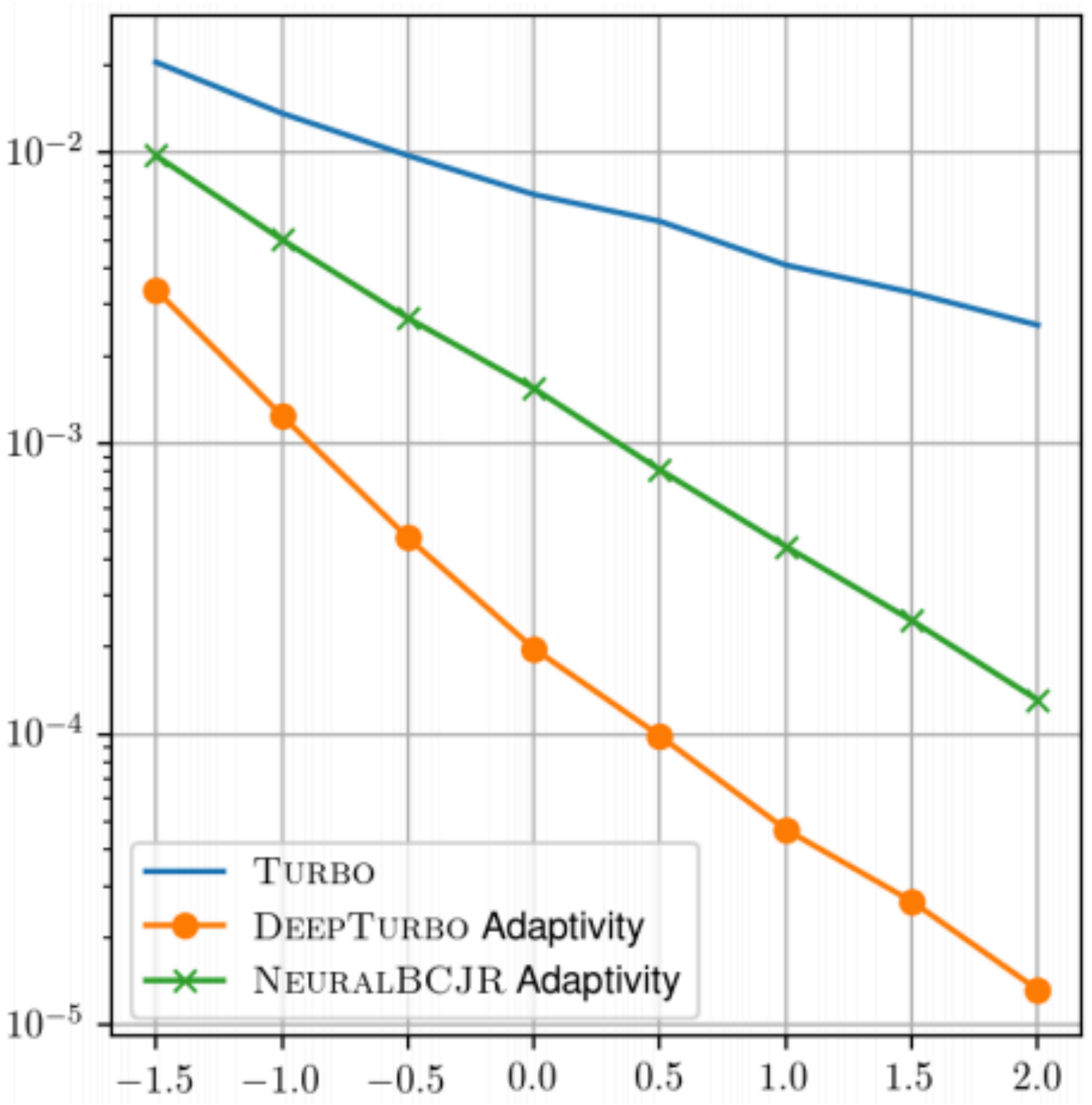
Neural BCJR vs. DeepTurbo - Adaptivity comparison

- DeepTurbo shows improved adaptivity

BER



SNR of bursty channel



SNR of T-distributed noise channel

Rate 1/3 turbo code (100 bits)

Summary

- Optimal decoders can be learned for AWGN channels
  - Training with hardest but decodable examples

Summary

- Optimal decoders can be learned for AWGN channels
  - Training with hardest but decodable examples

- Benefits
  - Robust to varying channel statistics
  - Adaptive when analytically designing a decoder is hard

Overview

- Inventing neural decoders
  - Example
    - Learning state-of-the-art decoders for AWGN channels
    - Improving the state-of-the-art decoders for non-AWGN channels
  - Literature
  - Open problems

Literature

- Learning a decoder from data alone vs. model-based

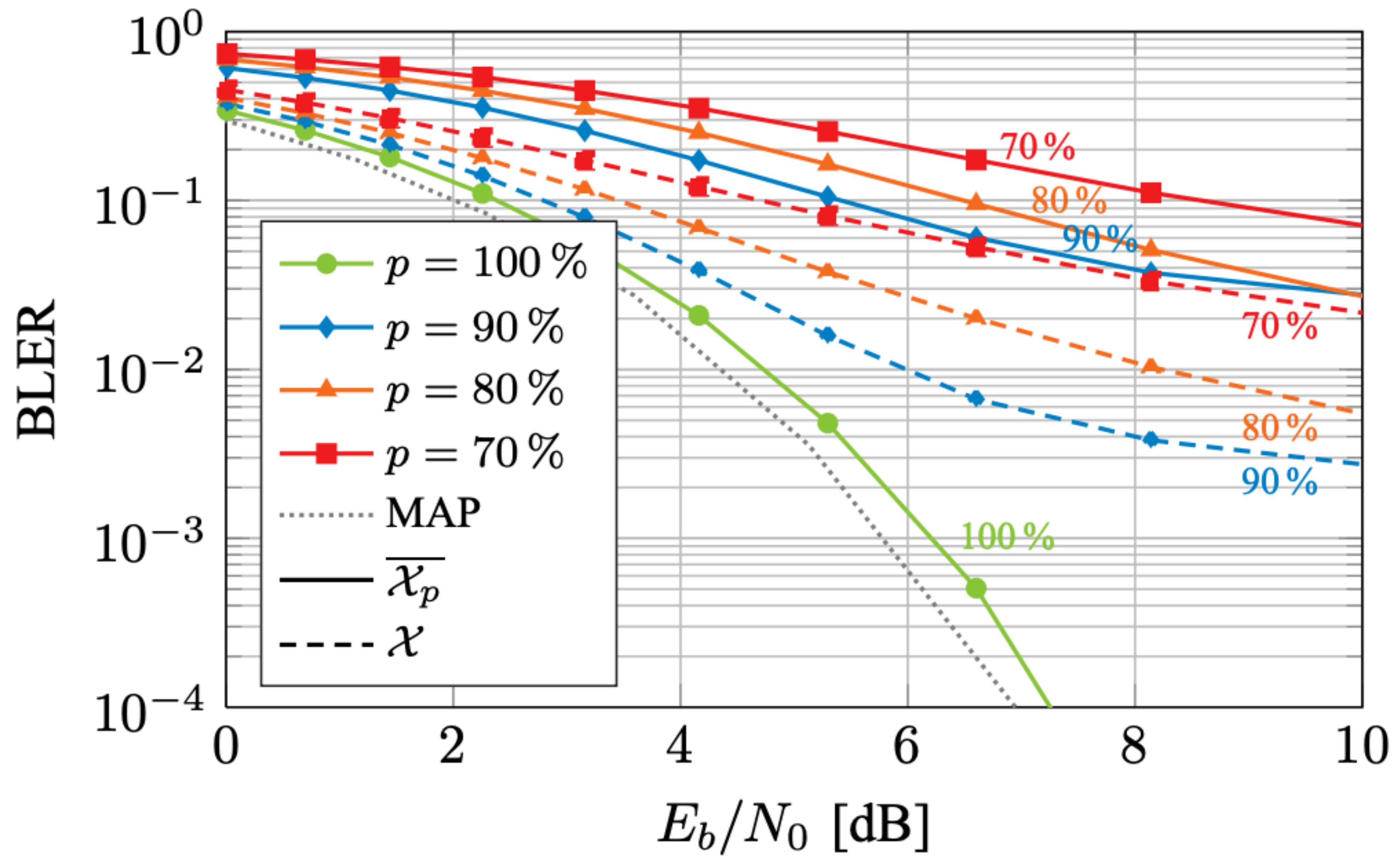
- Decoding sequential codes or linear block codes

Literature

- Learning a decoder from data alone
  - Polar codes for AWGN channels
    - Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, Stephan ten Brink, "***On deep learning-based channel decoding***", 2017

Literature

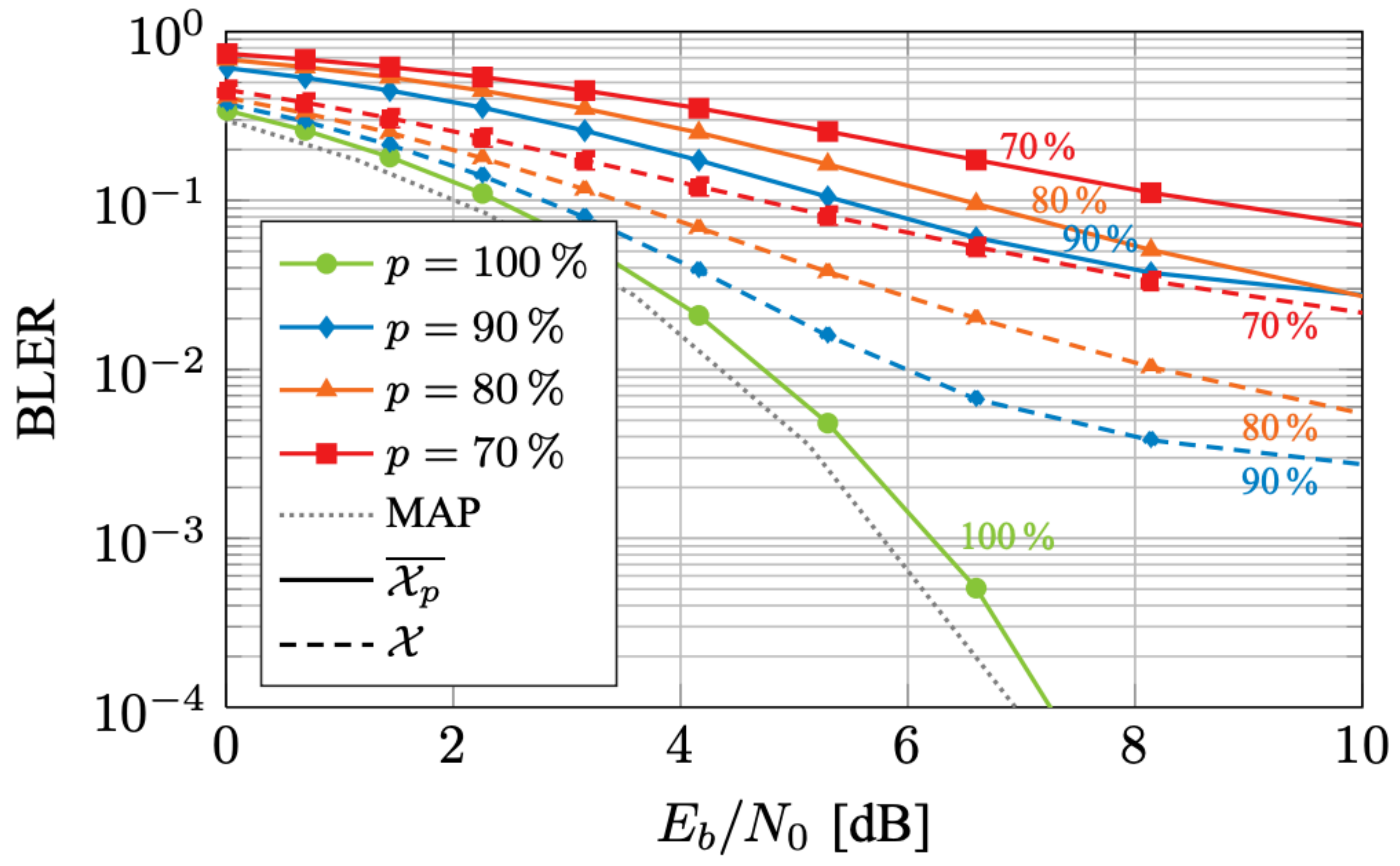
- Learning a decoder from data alone
  - Polar codes for AWGN channels
    - Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, Stephan ten Brink, "***On deep learning-based channel decoding***", 2017
    - Achieving strong generalization is challenging



Literature

- Learning a decoder from data alone
    - Polar codes for AWGN channels
        - Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, Stephan ten Brink, "***On deep learning-based channel decoding***", 2017
        - Achieving strong generalization is challenging
        - Scaling to longer block lengths

          Conventional iterative decoding algorithm w/ sub-blocks replaced by neural decoders



Literature

- Learning a decoder from data alone

  - Nonlinear channels (e.g., molecular channels)

    - Nariman Farsad, Andrea Goldsmith, "***Neural Network Detection of Data Sequences in Communication Systems***", 2018

Literature

- Learning a decoder from data alone
  - Nonlinear channels (e.g., molecular channels)
    - Nariman Farsad, Andrea Goldsmith, "***Neural Network Detection of Data Sequences in Communication Systems***", 2018
    - RNN-based detection (w/o CSI) achieves the reliability of Viterbi detector with CSI

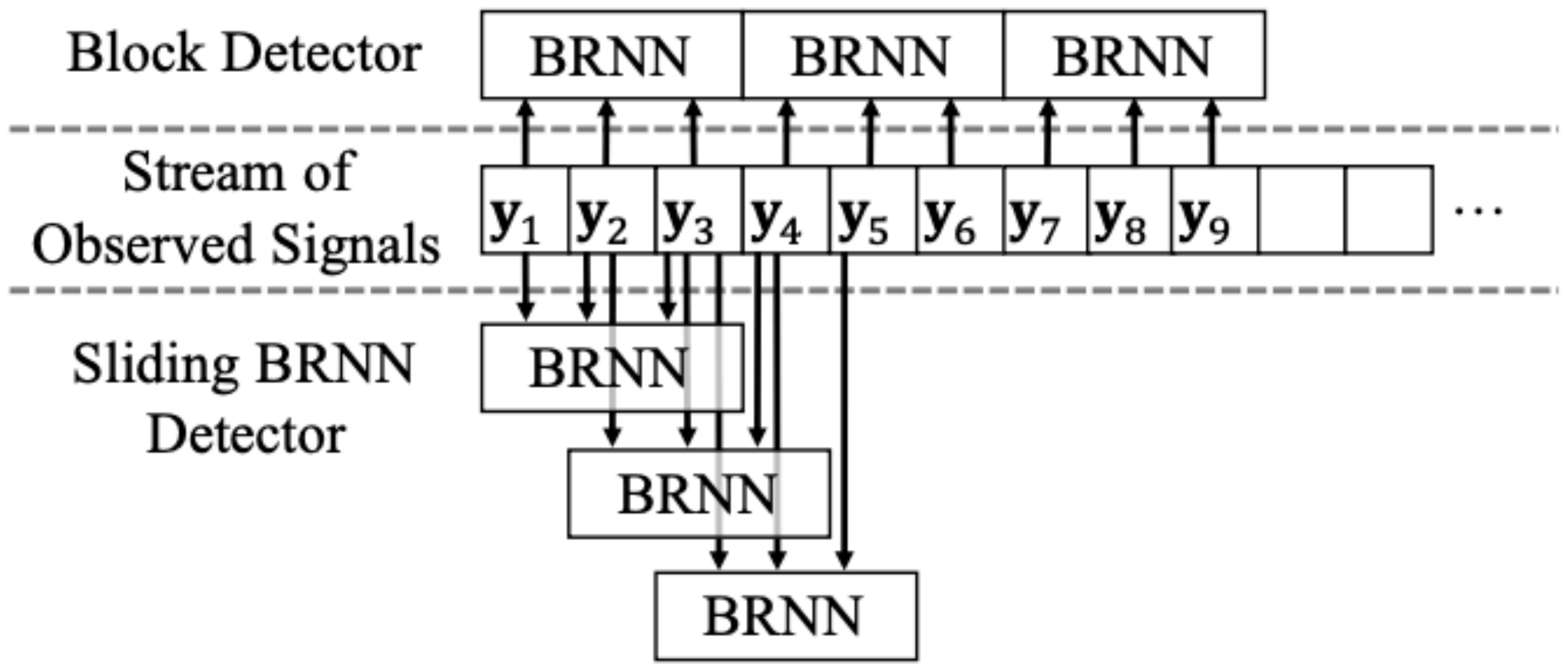
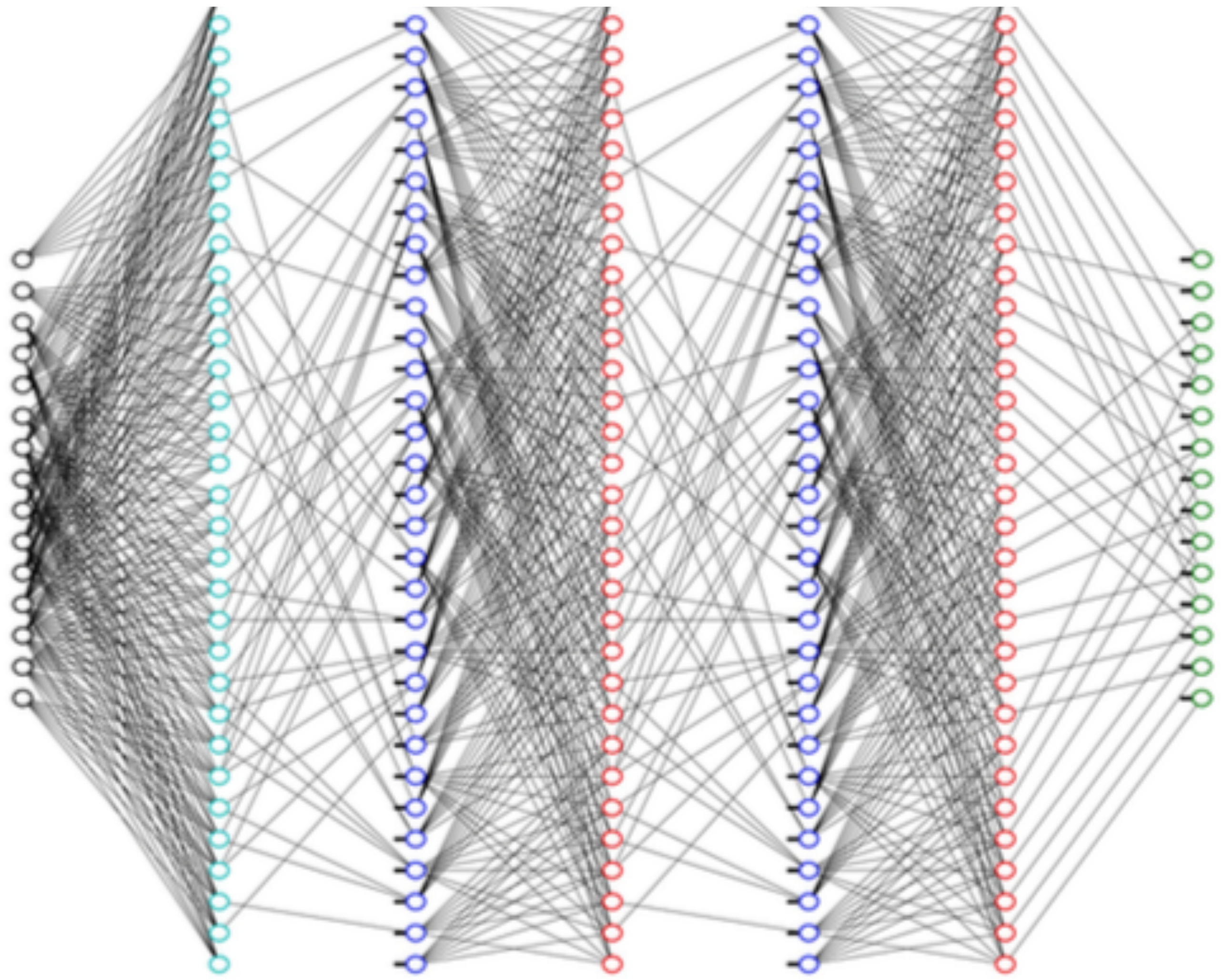


Fig. 4: The sliding BRNN detector.

Literature

- Model-based decoder with learnable variables

  - Weighted BP decoder for linear block codes



Eliya Nachmani, Yair Be'ery, David Burshtein,
"**Learning to decode linear codes using deep learning**", 2016

Eliya Nachmani, Yaron Bachar, Elad Marciano, David Burshtein, Yair Be'ery,
"**Near Maximum Likelihood Decoding with Deep Learning**", 2018

Literature

- Learnable variables (w) to the BP over the trellis graph

$$x_{i,e=(v,c)} = \tanh\left(\frac{1}{2}\left(w_{i,v} l_v + \sum_{e'=(v,c'),\, c' \neq c} w_{i,e,e'} x_{i-1,e'}\right)\right)$$

for odd i,

$$x_{i,e=(v,c)} = 2\tanh^{-1}\left(\prod_{e'=(v',c),\, v' \neq v} x_{i-1,e'}\right)$$

for even i, and

$$o_v = \sigma\left(w_{2L+1,v} l_v + \sum_{e'=(v,c')} w_{2L+1,v,e'} x_{2L,e'}\right)$$

Literature

- Training and generalization
  - Training with (noisy versions of) all-zero codewords is sufficient due to the symmetry in the decoder structure

Literature

- Model-based decoder with learnable variables
  - ViterbiNet for convolutional codes for non-AWGN channels

    N. Shlezinger, Y. C. Eldar, N. Farsad and A. J. Goldsmith,
    "**ViterbiNet: Symbol Detection Using a Deep Learning Based Viterbi Algorithm**,"
    2019 IEEE 20th SPAWC, Cannes, France, 2019

Literature

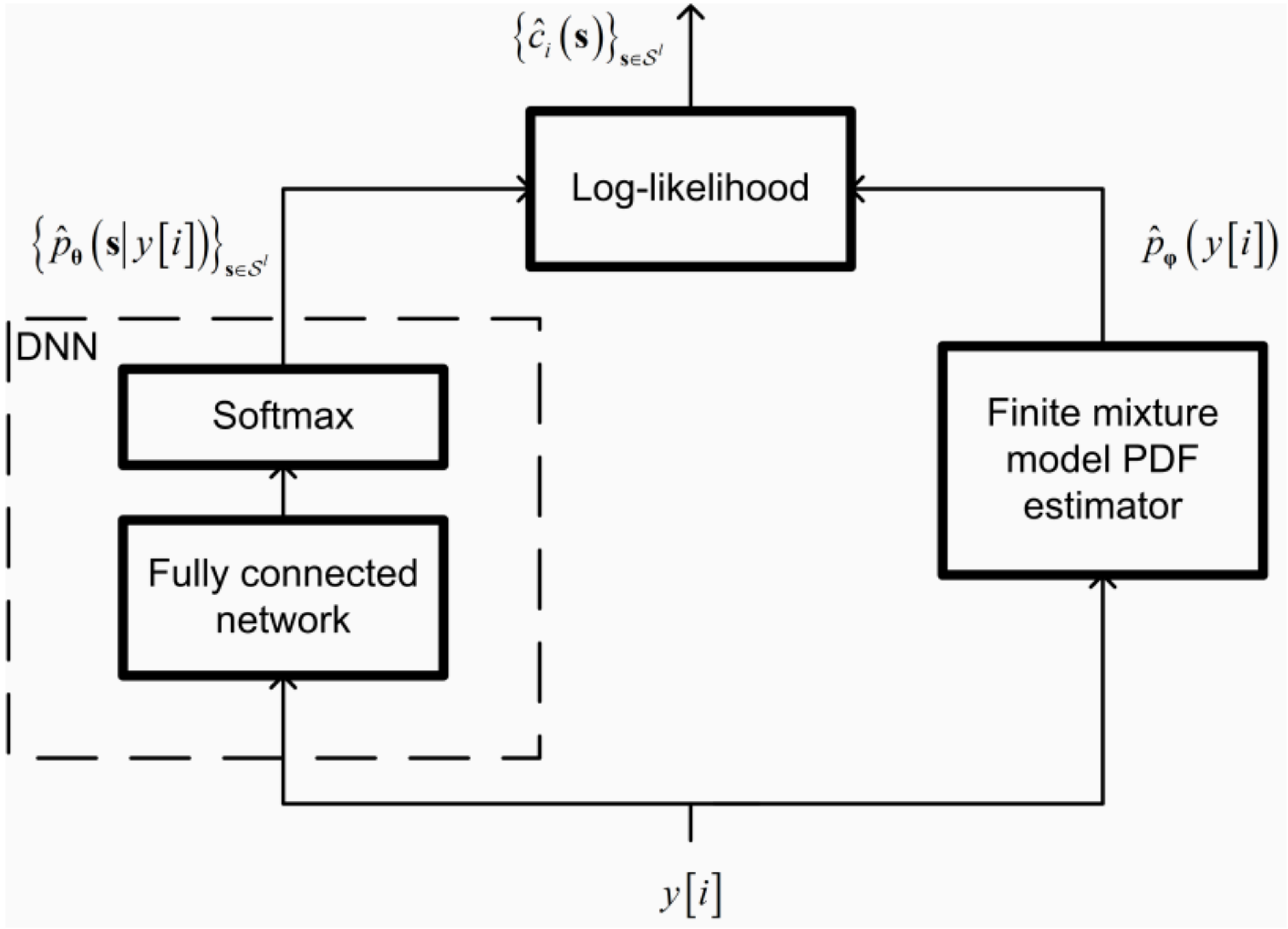
- Model-based decoder with learnable variables
  - ViterbiNet for convolutional codes for non-AWGN channels

    N. Shlezinger, Y. C. Eldar, N. Farsad and A. J. Goldsmith,
    "**ViterbiNet: Symbol Detection Using a Deep Learning Based Viterbi Algorithm**,"
    2019 IEEE 20th SPAWC, Cannes, France, 2019

  - Channels with memory: Inter-symbol-interference channels

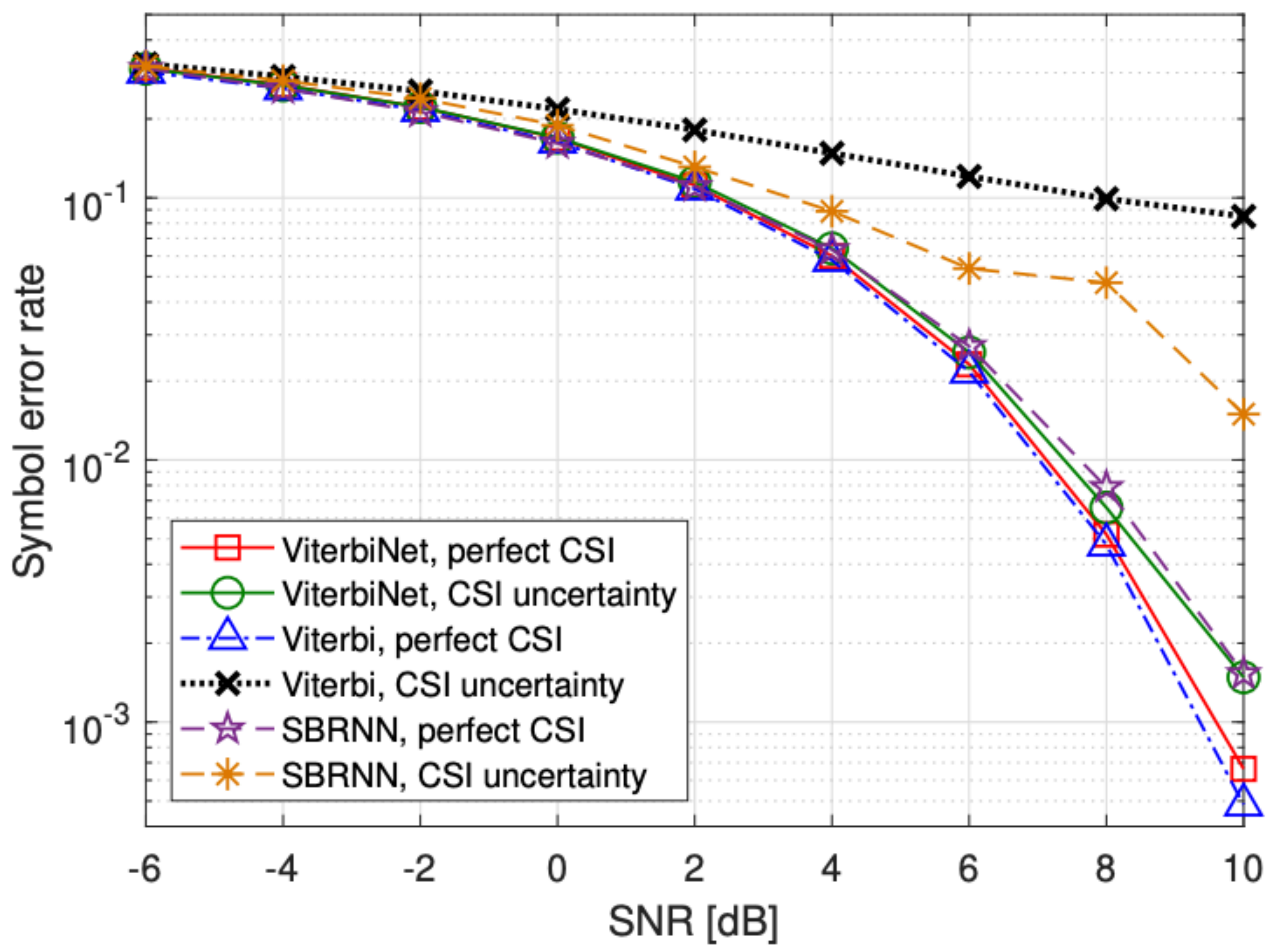
Literature

- ViterbiNet: Cost computation of Viterbi is learned from data



Literature

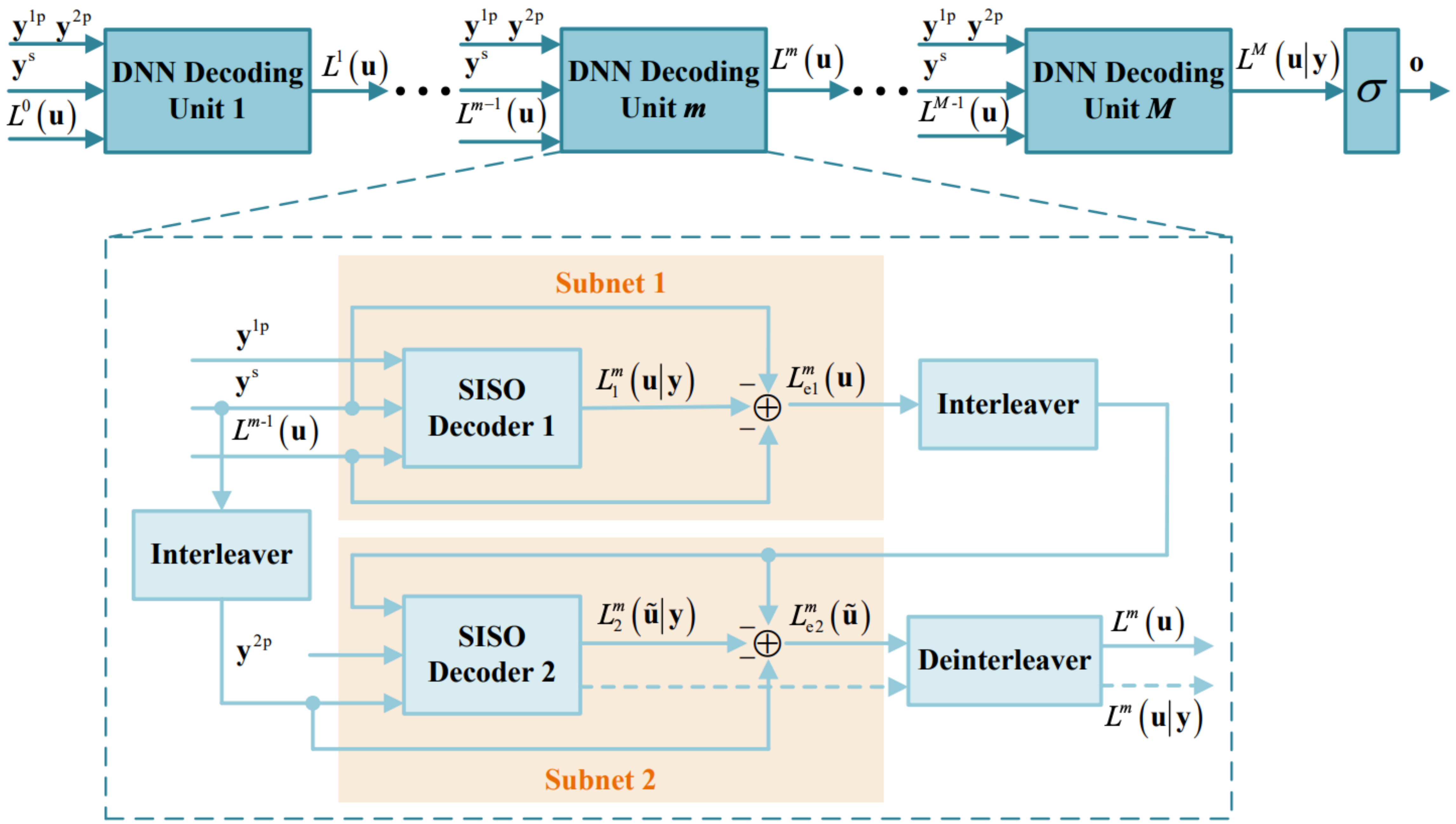
- ViterbiNet (w/o CSI) ~ Reliability of Viterbi detector w/ CSI



SER versus SNR, ISI channel with AWGN.

Literature

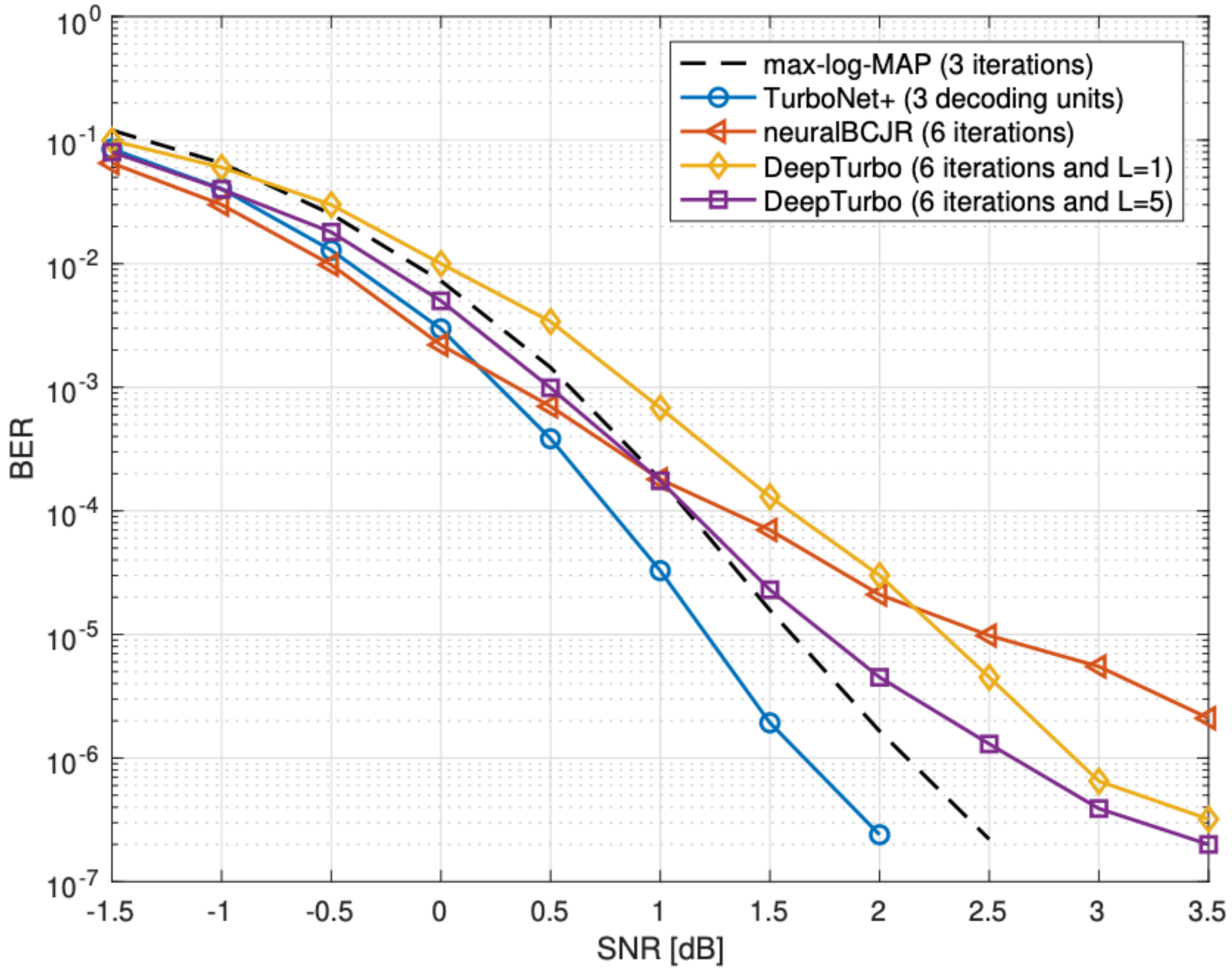
- Model-based decoder with learnable variables
  - TurboNet



Yunfeng He, Jing Zhang, Shi Jin, Chao-Kai Wen, Geoffrey Ye Li
"**Model-Driven DNN Decoder for Turbo Codes: Design, Simulation and Experimental Results,**"
arXiv June 2020

Literature

- TurboNet outperforms DeepTurbo at high SNR

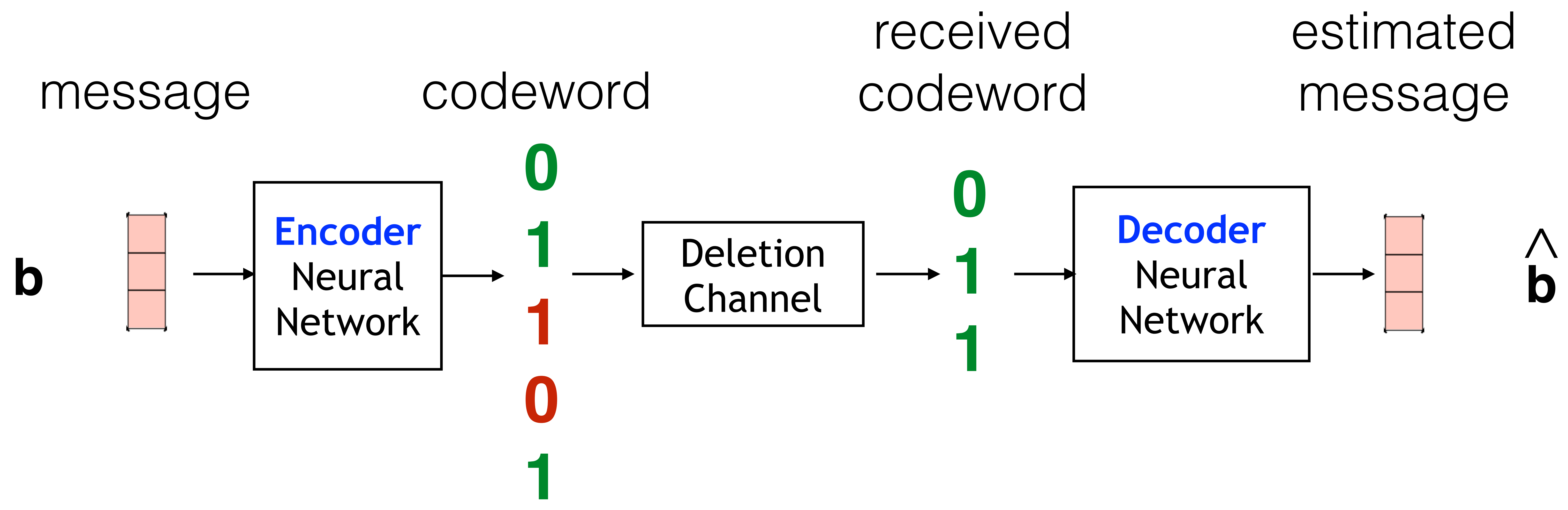


Open problems

- High SNR
- Large block lengths

Open problems

- Channels for which decoders can be improved
  - Nonlinear channels (e.g., low-precision ADC)
  - Deletion channels (e.g., nanopore sequencing)



Open problems

- Practical aspects of neural network based decoders
  - Complexity
  - Testing on real channels

Open problems

- Practical aspects of neural network based decoders
  - Complexity
  - Testing on real channels
  - Fast adaptation to varying channels
    - Meta-learning!

Adapting Decoder to Channel Variations

Joint work with Yihan Jiang, Himanshu Asani, Hyeji Kim

Adaptation Hierarchy

- Train on AWGN, Test on General: Robustness
  - Theory: Worst case noise
- Train on Complex channel, Test on Complex Channel: Generality
  - Theory: Optimal codes on non-AWGN
- Train on Set of channels, New channel (few training symbols), Test on New Channel

Adaptation: Goal

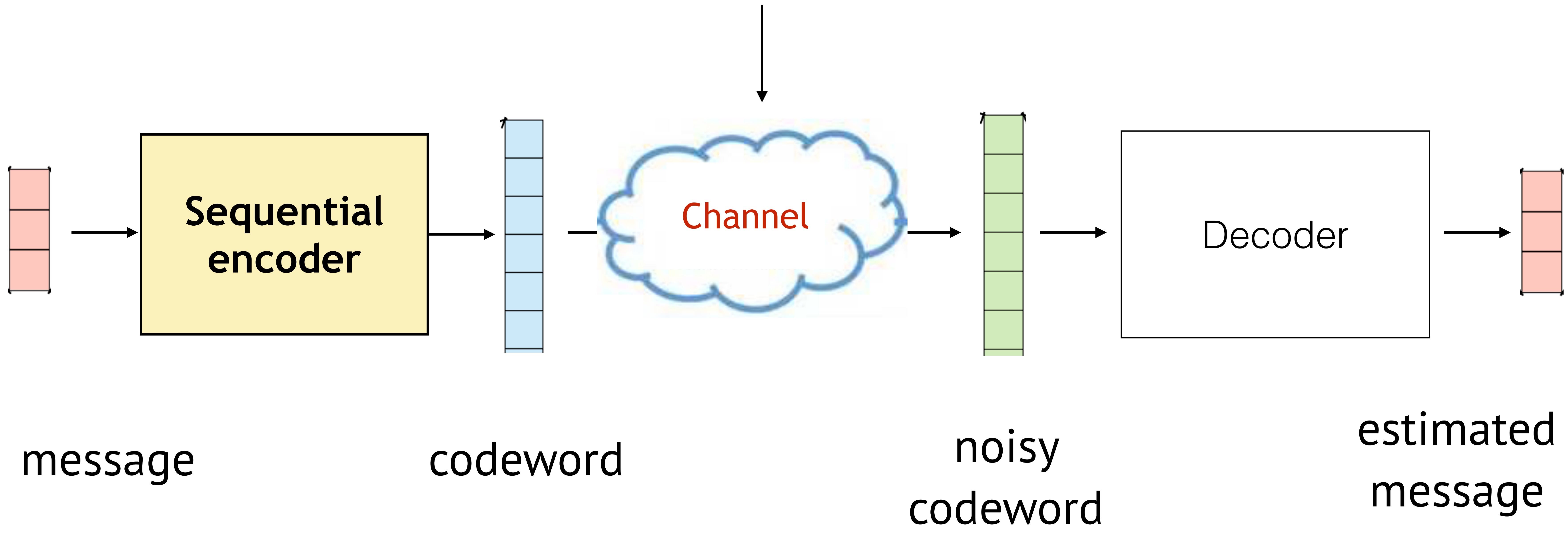
- Existing systems: Decoders adapt to channel variations by equalization
- Equalization is typically for *parametrizable multiplicative* effects
- Can we design a method to adapt to general channel variations
  - For example, noise is not Gaussian, channel is not iid,...

Setting

Test: Trained channels $\{c_1, c_2, \ldots, c_m\}$

Paradigm: Observe training symbols on c_i
Then Adapt

Training: Possible channels $\{c_1, c_2, \ldots, c_m\}$



Multi-task learning: General Idea

Setting: K Tasks with different (x_j,y_j) distributions

Train: Data from the K Tasks $$L_{T_i}(f_\theta) = \sum_{x^{(j)},y^{(j)} \sim T_i} BCE(f_\theta(x^{(j)}), y^{(j)}).$$

MTL
(Train for average
Of tasks)

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \in \{T\}} L_{T_i}(f_\theta).$$

Test phase: 1) Observe few samples from a random task
2) Update the model using SGD
3) Calculate test performance on the updated model

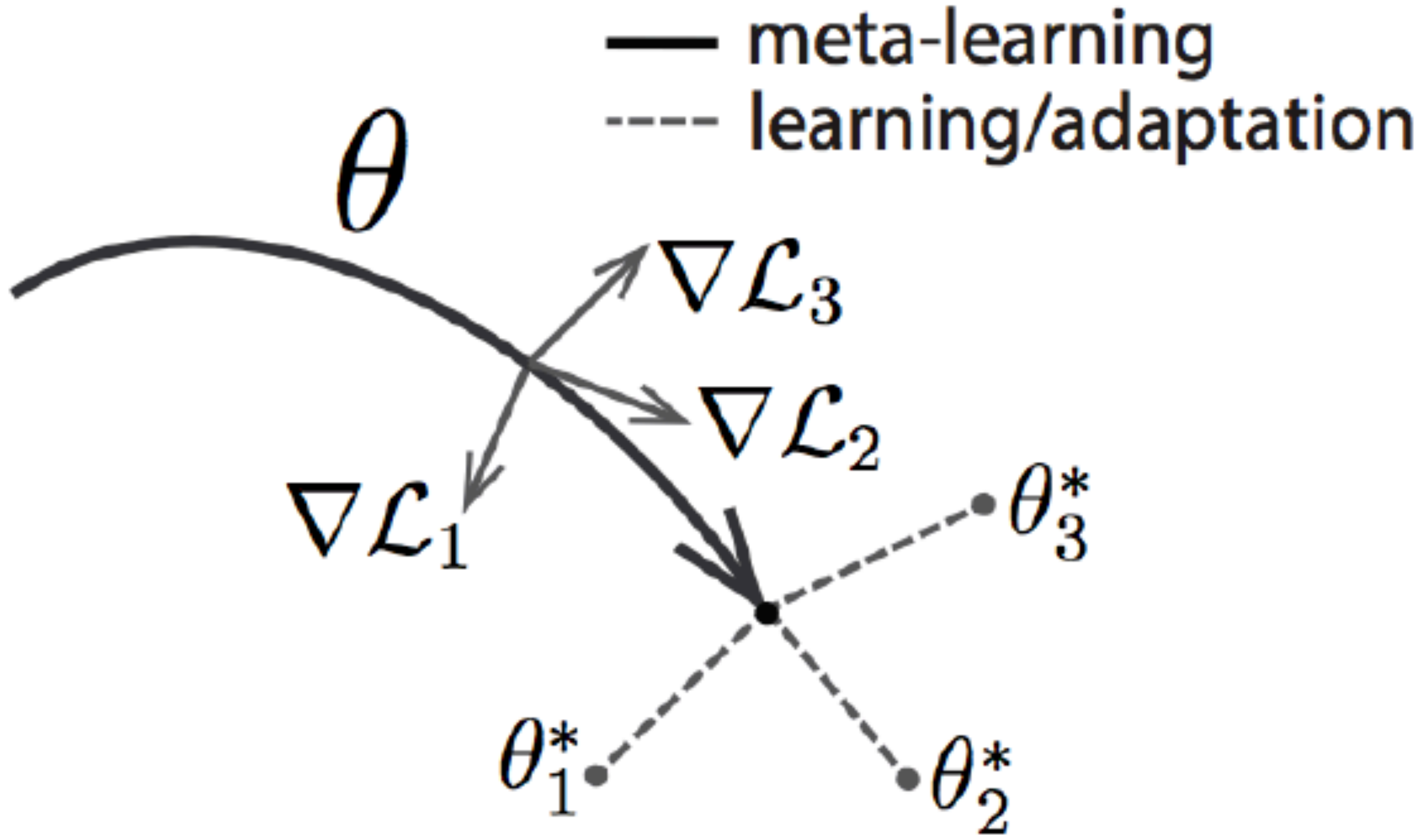
MAML: General Idea

Setting: K Tasks with different (x_j, y_j) distributions

Train: Data from the K Tasks $L_{T_i}(f_\theta) = \sum_{x^{(j)}, y^{(j)} \sim T_i} BCE(f_\theta(x^{(j)}), y^{(j)}).$

1-step Grad for task-i $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

$$\min_\theta \sum_i L_{T_i}(\theta_i') = \min_\theta \sum_i L_{T_i}(\theta - \alpha \nabla_\theta L_{T_i}(f_\theta))$$



Test: SGD on θ after observing a batch

MAML: For Channel Decoding

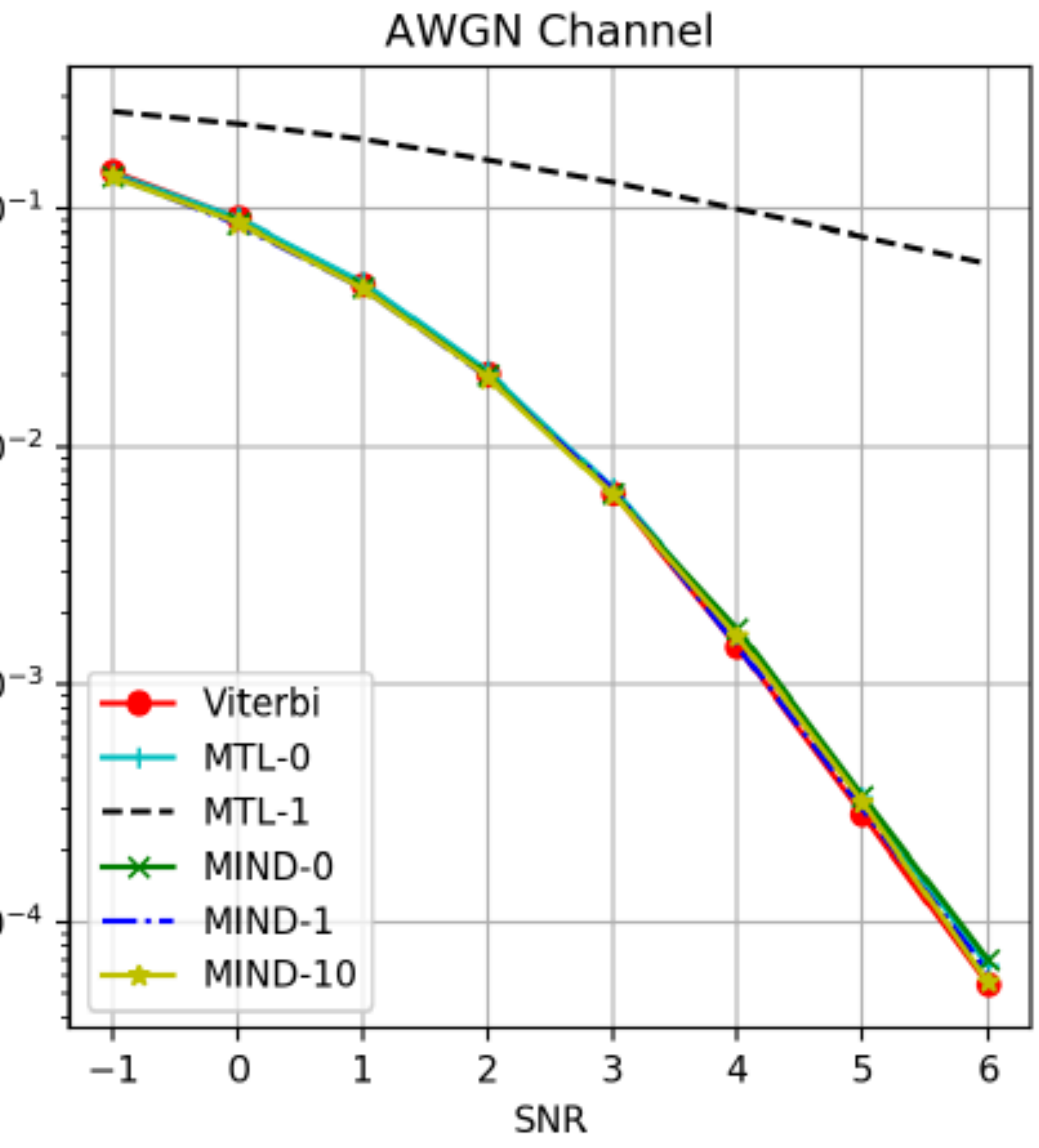
Setting: K Tasks = K Channels

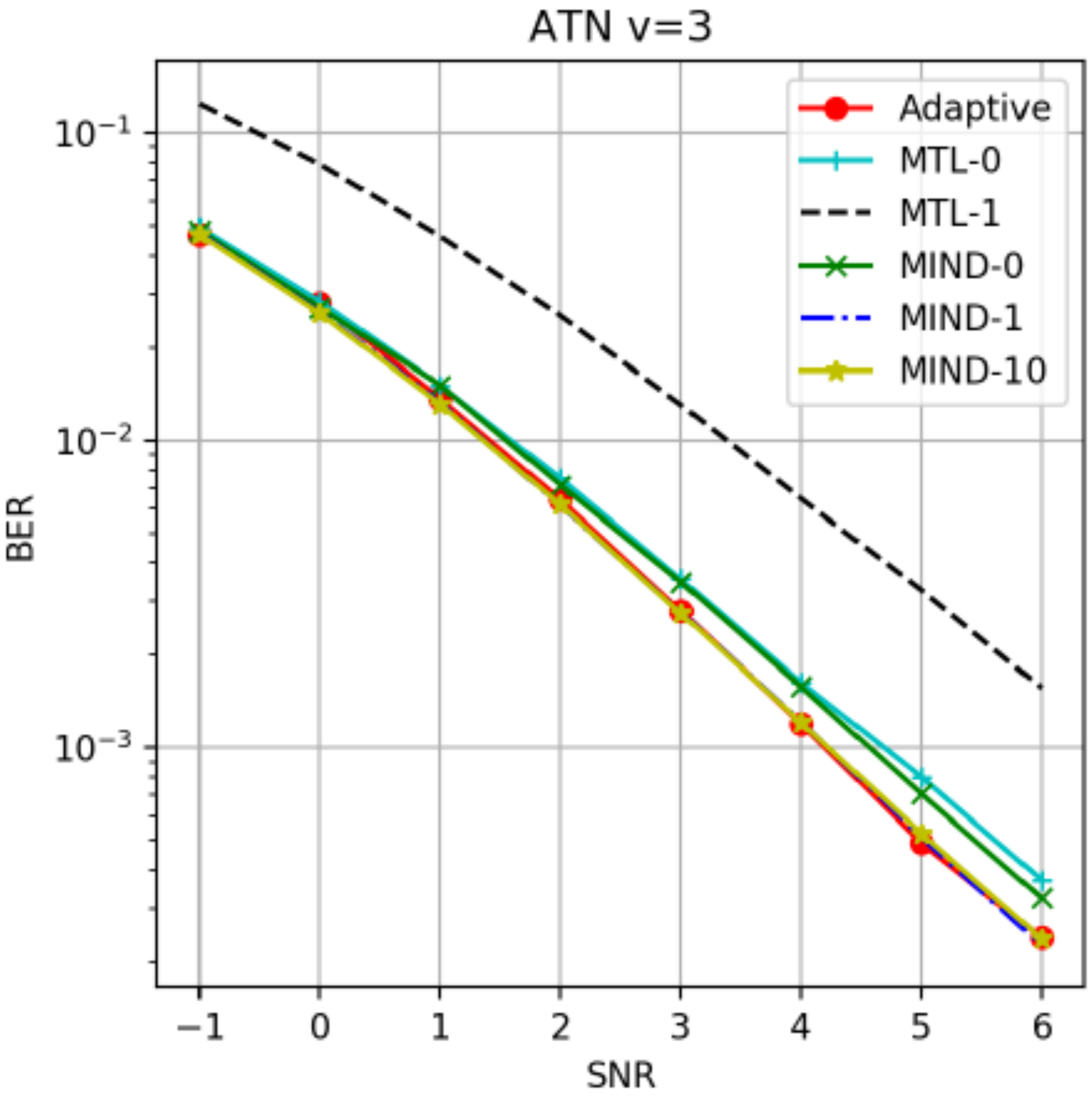
Train: Using simulated data from K channels

Test phase: 1) On the new channel: Observe some received symbols (with ground truth message known)
2) Update the decoder model using SGD
3) Calculate test performance on the updated model

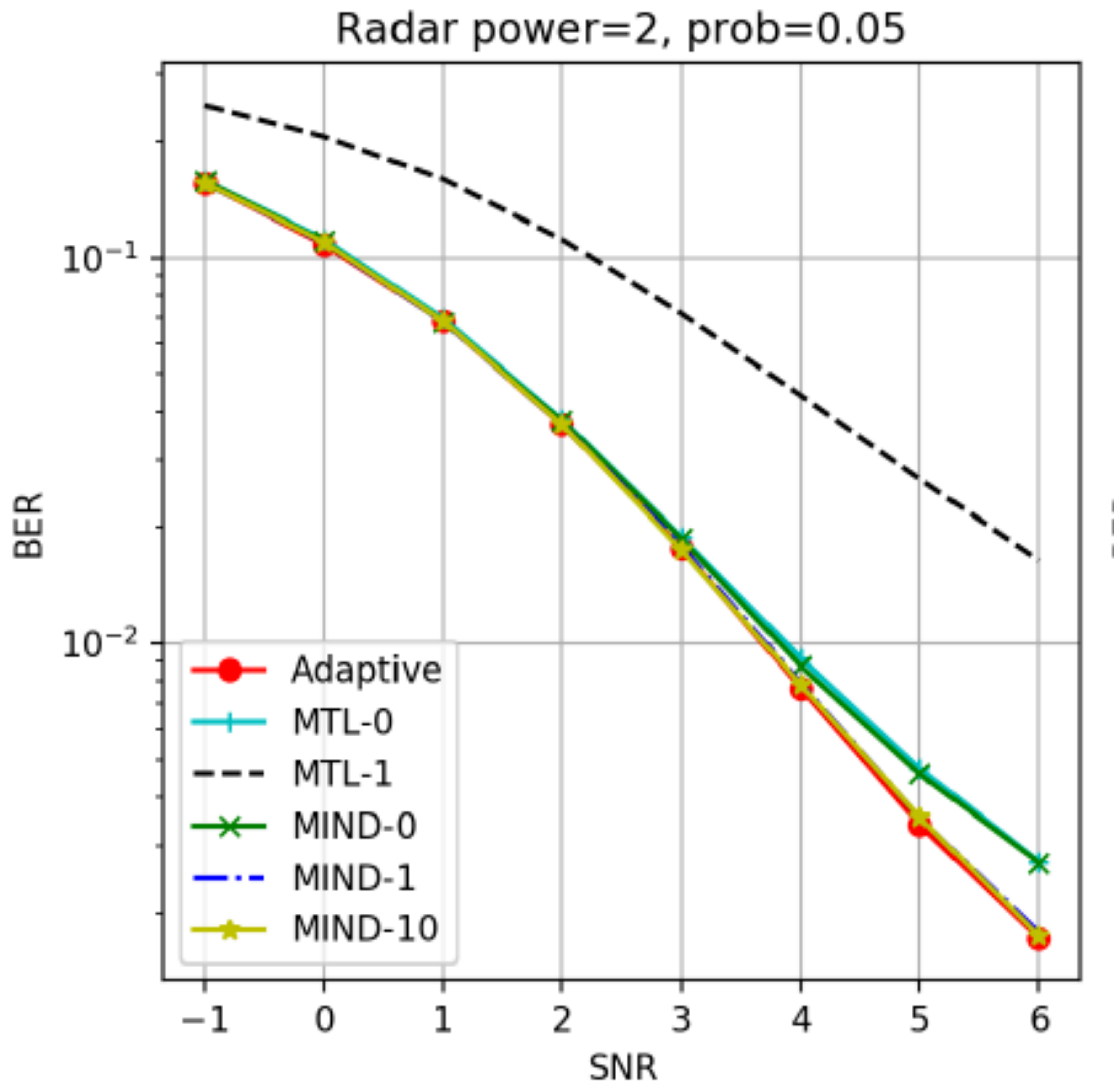
Two performance metrics: 1) How much data?
2) How much computation?

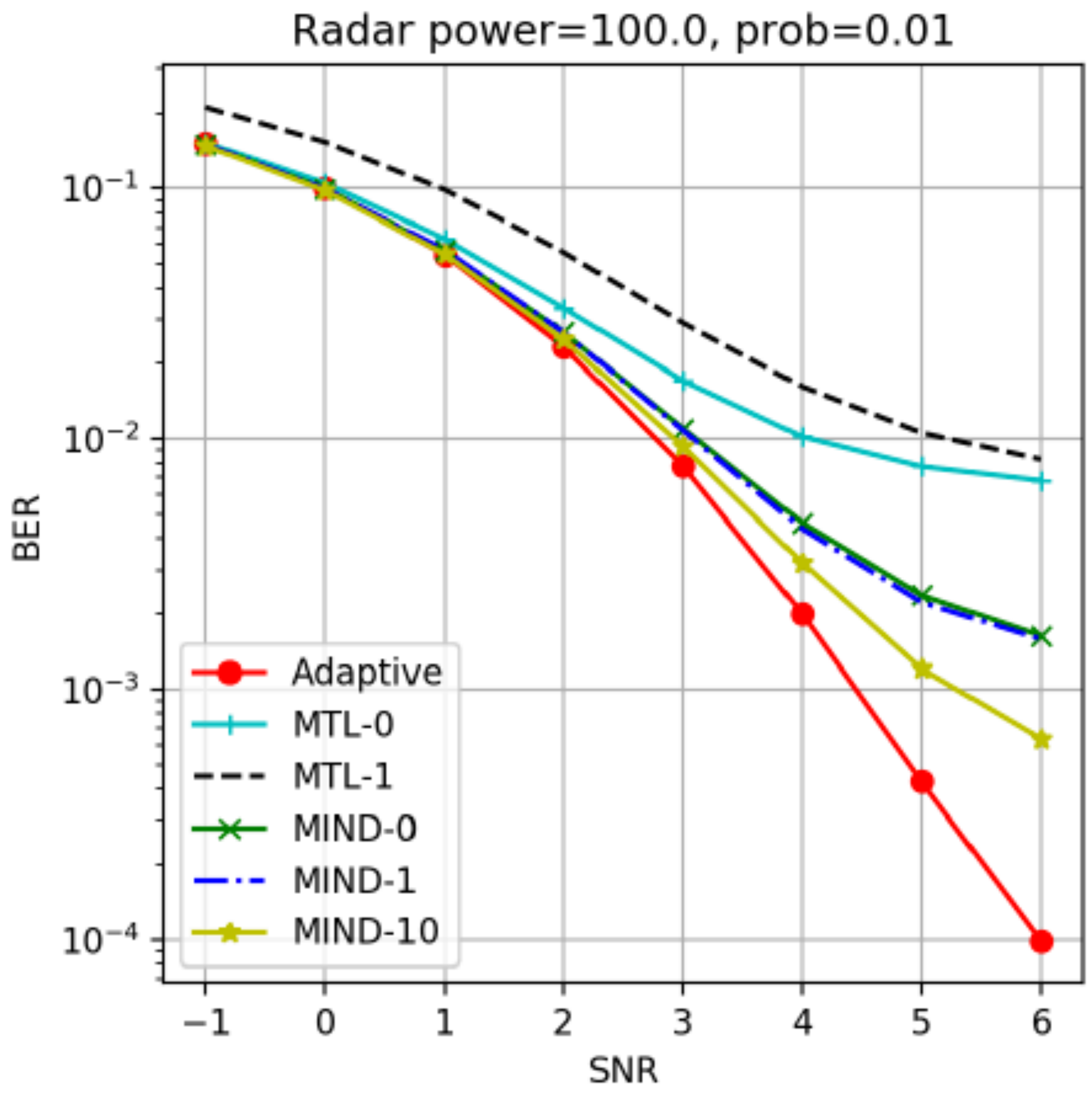
Convolutional Code + MAML





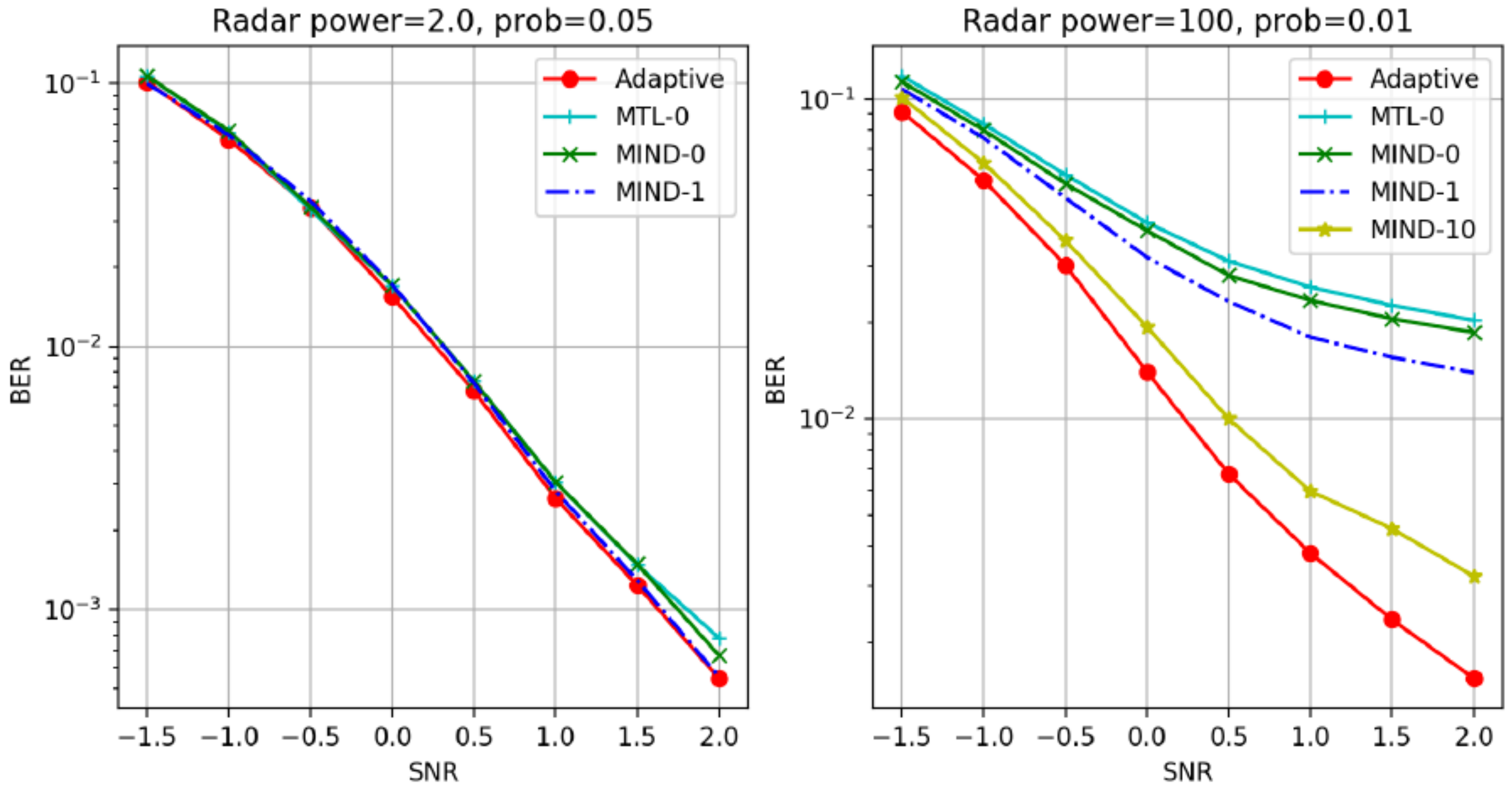
Convolutional Code + MAML





Untrained Channel

Turbo MAML Decoder



Trained Channel **Untrained** Channel

Parameters

Parameters	Convolutional Code	Turbo Code
Neural Decoder	2 layer bi-GRU	2 layer bi-GRU
Number of Neural Units	200	200
Batch Size B	100	100
Meta Batch Size P	10	10
Meta Learning Rate β	0.00001	0.00001
Adaptation Learning Rate α	0.001	0.0001
Number of Meta Update Steps	50000	50000
Block Length L	100	100
Train SNR	0 to 4dB	-1.5 to 2dB
Code Rate	1/2	1/3

Testing Method	Adaptation Data	Task Update Steps K
Fine-tune	1000000	10000
MIND-1 Meta Testing	100	1
MIND-10 Meta Testing	1000	10

Summary

- Showed that adaptation is possible with much fewer samples using MAML
- Still 100 batches of 100 training symbols each is required for adaptation
- Proposal for practice:
  - Use Equalizer to deal with multiplicative effects
  - Use MAML to adapt to *additive noise effects over slow-time scale*

Summary

- Showed that adaptation is possible with much fewer samples using MAML

- Still 100 batches of 100 training symbols each is required for adaptation

- Proposal for practice:

  - Use Equalizer to deal with multiplicative effects

  - Use MAML to adapt to <u>*additive noise effects over slow-time scale*</u>

Overview

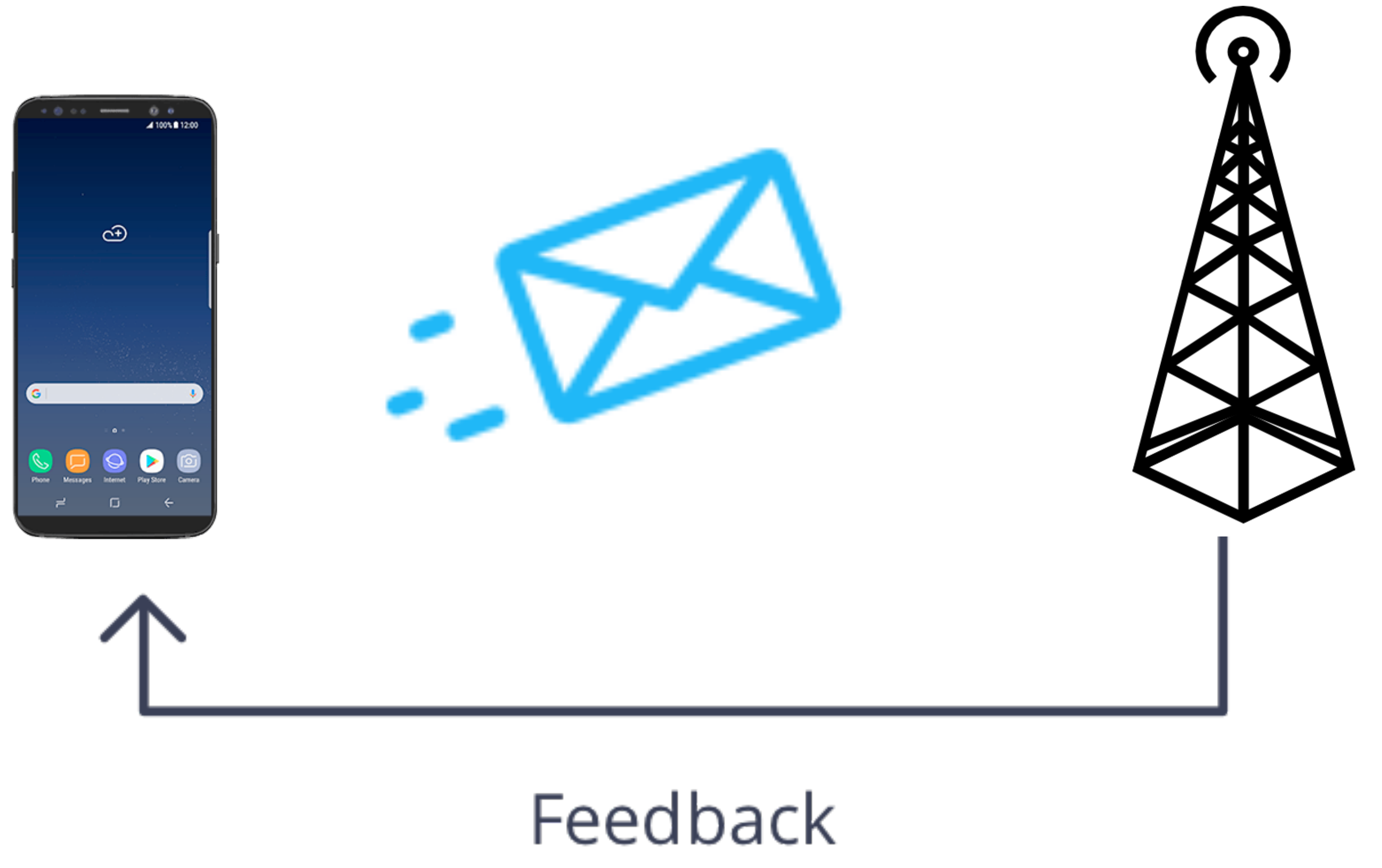
- Introduction to Neural Networks
- Inventing neural decoders
- Inventing neural codes
- Other applications of deep learning to information theory

Overview

- Inventing neural codes
  - Example
    - Learning a code for channels with output feedback
    - Learning Turbo codes
    - Coding for channels with block-wise output feedback
  - Literature
  - Open problems

Learning a code

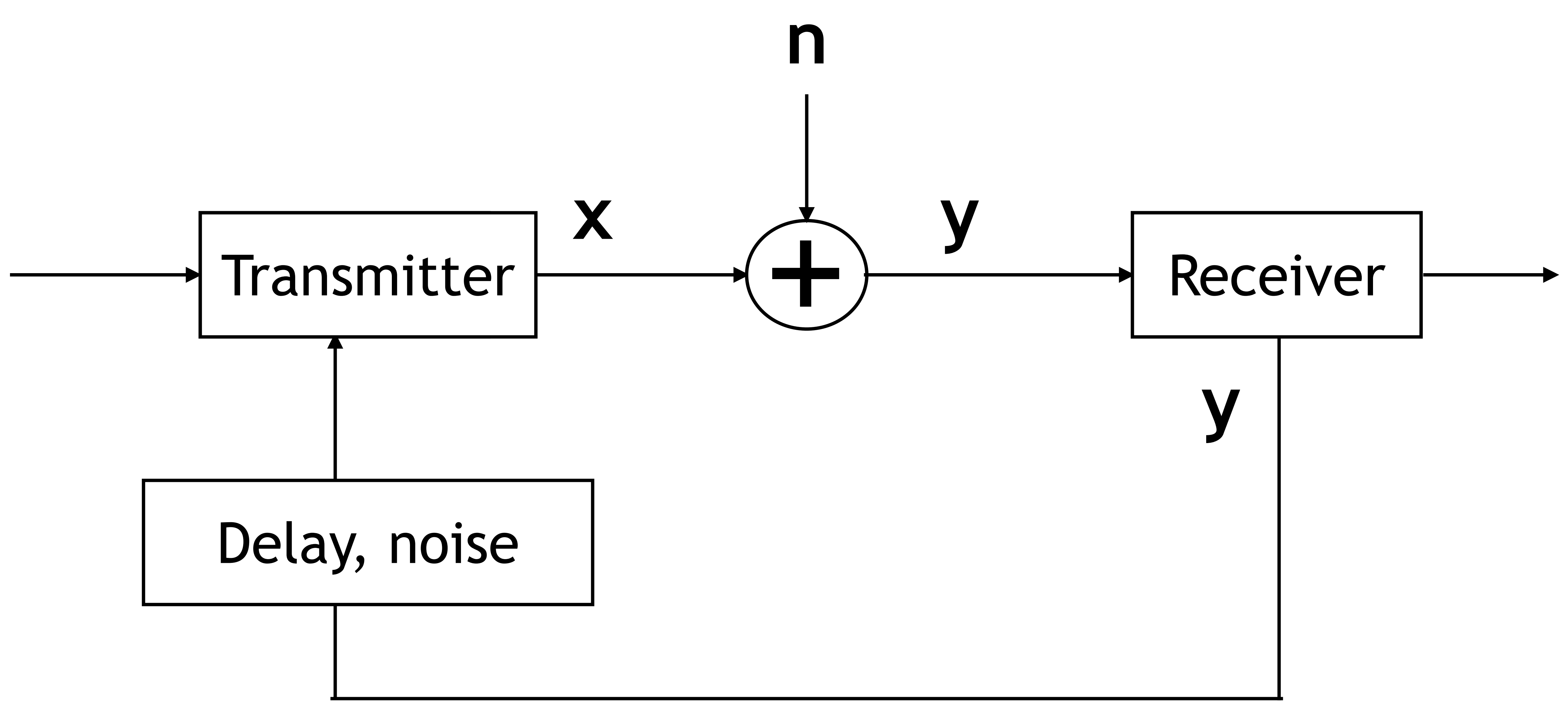
for channels with feedback



“*Deepcode: feedback codes via deep learning,*” K-Jiang-Kannan-Oh-Viswanath NeurIPS’18

AWGN channels with output feedback

- AWGN channel from transmitter to receiver
- Output fed back to the transmitter



Literature

- Noiseless feedback
  - Improved reliability
  - Coding schemes
    - Schalkwijk-Kailath scheme (Schalkwijk-Kailath ’66)
    - Posterior matching (Shayevitz-Feder ’09)

Literature

- Noisy feedback
  - Existing schemes perform poorly

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Concatenated coding (Chance-Love ’11)

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Concatenated coding (Chance-Love ’11)
  - Linear codes very bad (Kim-Lapidoth-Weissman ’07)

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Concatenated coding (Chance-Love ’11)
  - Linear codes very bad (Kim-Lapidoth-Weissman ’07)
- Nonlinear codes?

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Concatenated coding (Chance-Love ’11)
  - Linear codes very bad (Kim-Lapidoth-Weissman ’07)
- Nonlinear codes?

  “Deepcode”(Kim-Jiang-Kannan-Oh-Viswanath ’18)

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Concatenated coding (Chance-Love ’11)
  - Linear codes very bad (Kim-Lapidoth-Weissman ’07)

- Nonlinear codes?

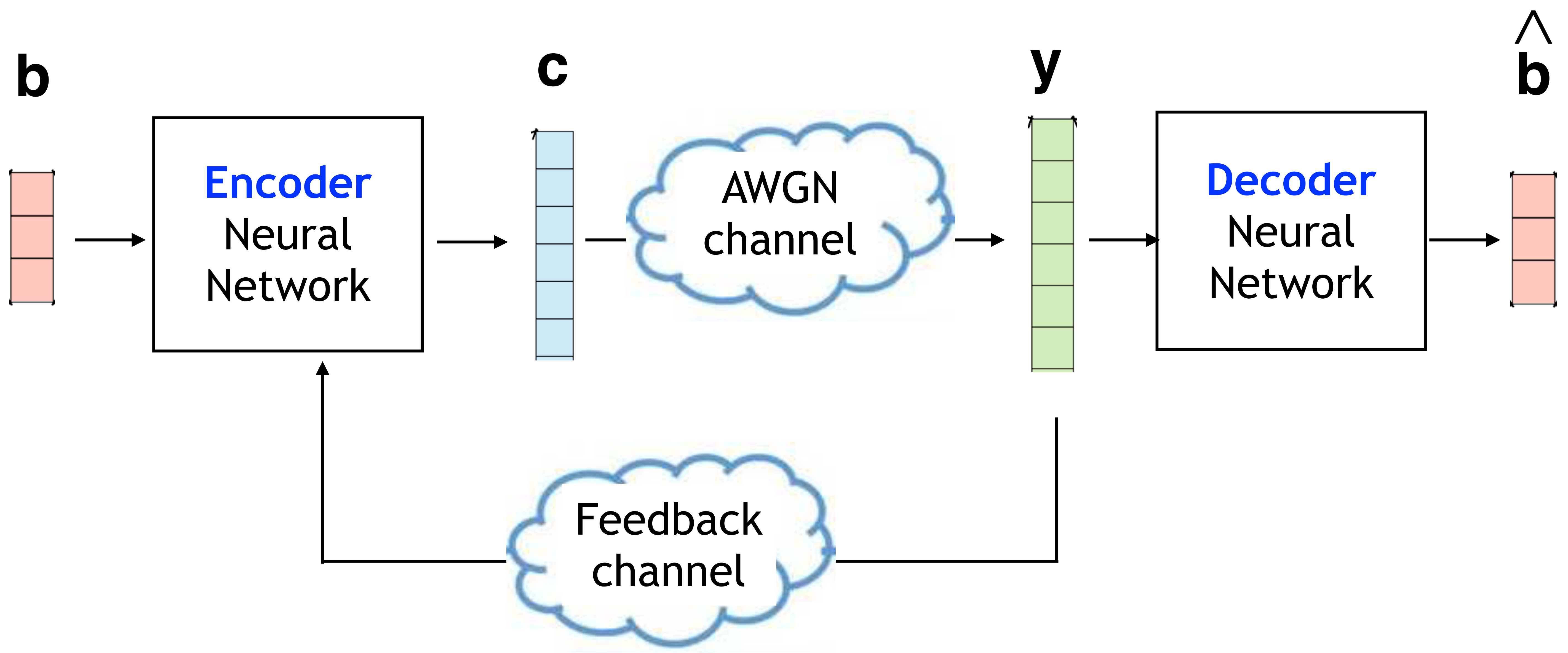
  “Deepcode”(Kim-Jiang-Kannan-Oh-Viswanath ’18)

- Challenge:

  How to combine noisy feedback and message causally?

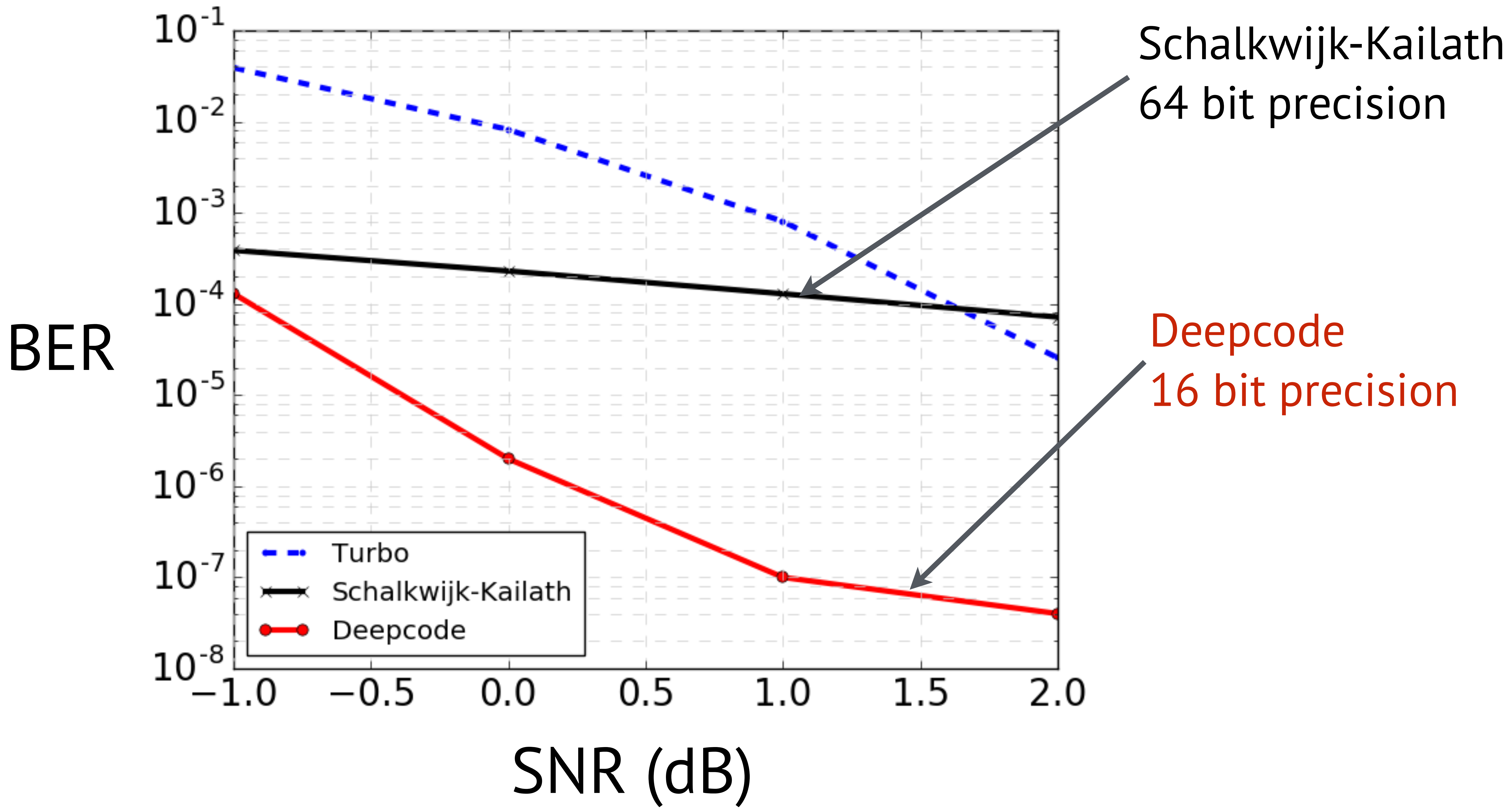
Our approach

- Deepcode
  - Model encoder and decoder as neural networks and learn



Main results

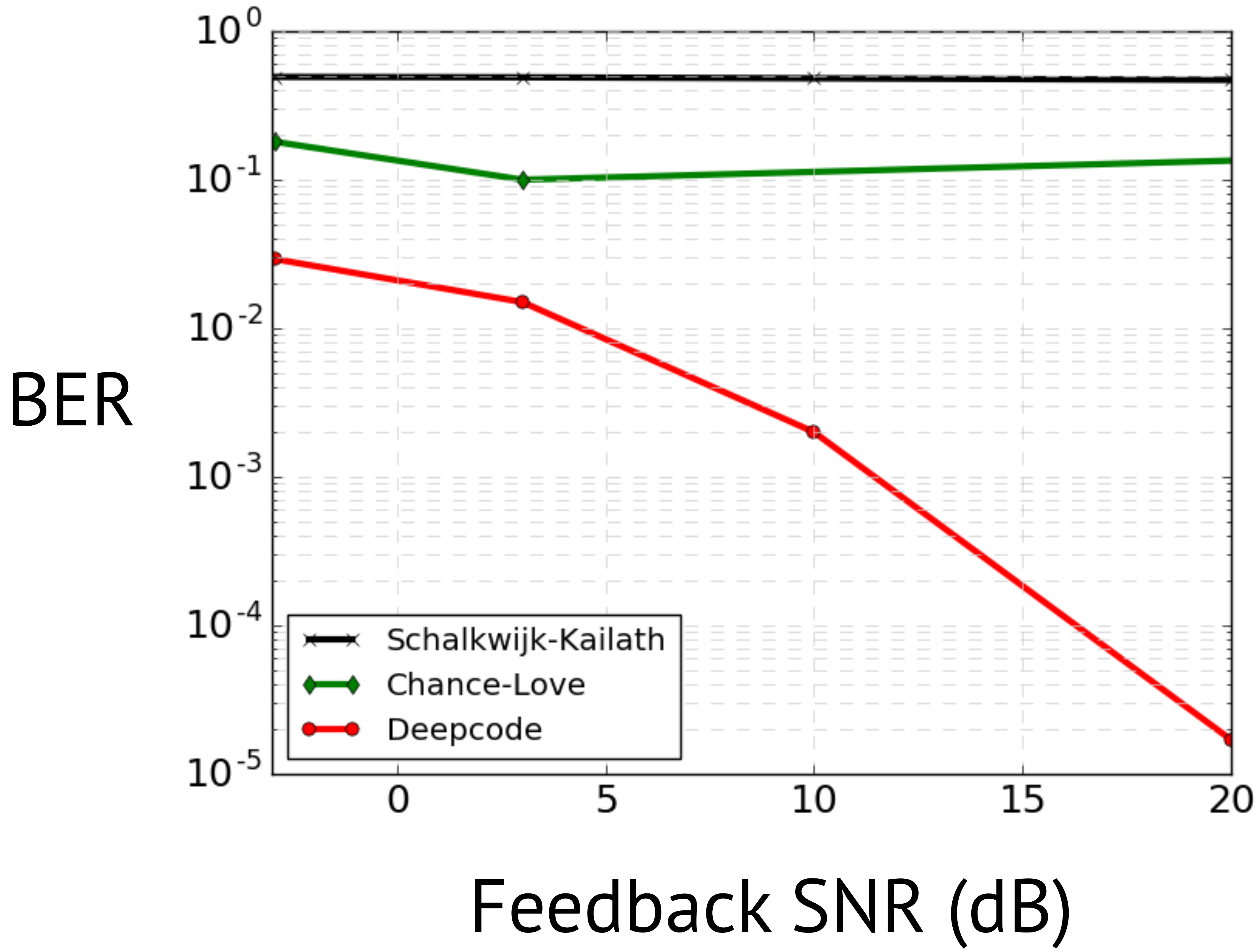
- 100x better reliability under noiseless feedback w. precision



(Rate 1/3, 50 bits)

Main results

- Outperforms state-of-the-art for AWGN feedback channel



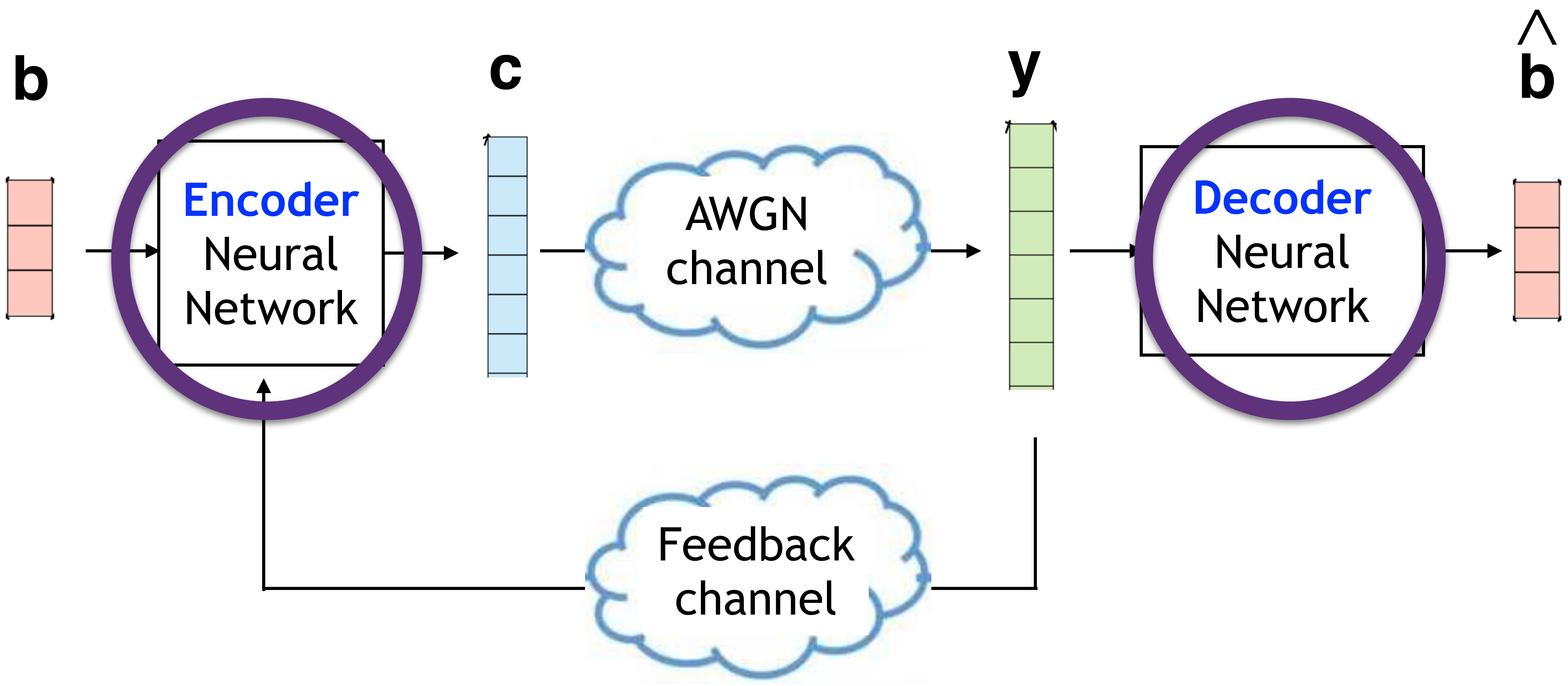
(Rate 1/3, 50 bits, SNR = 0dB)

Deepcode

Key: Architectural innovations, ideas from communications

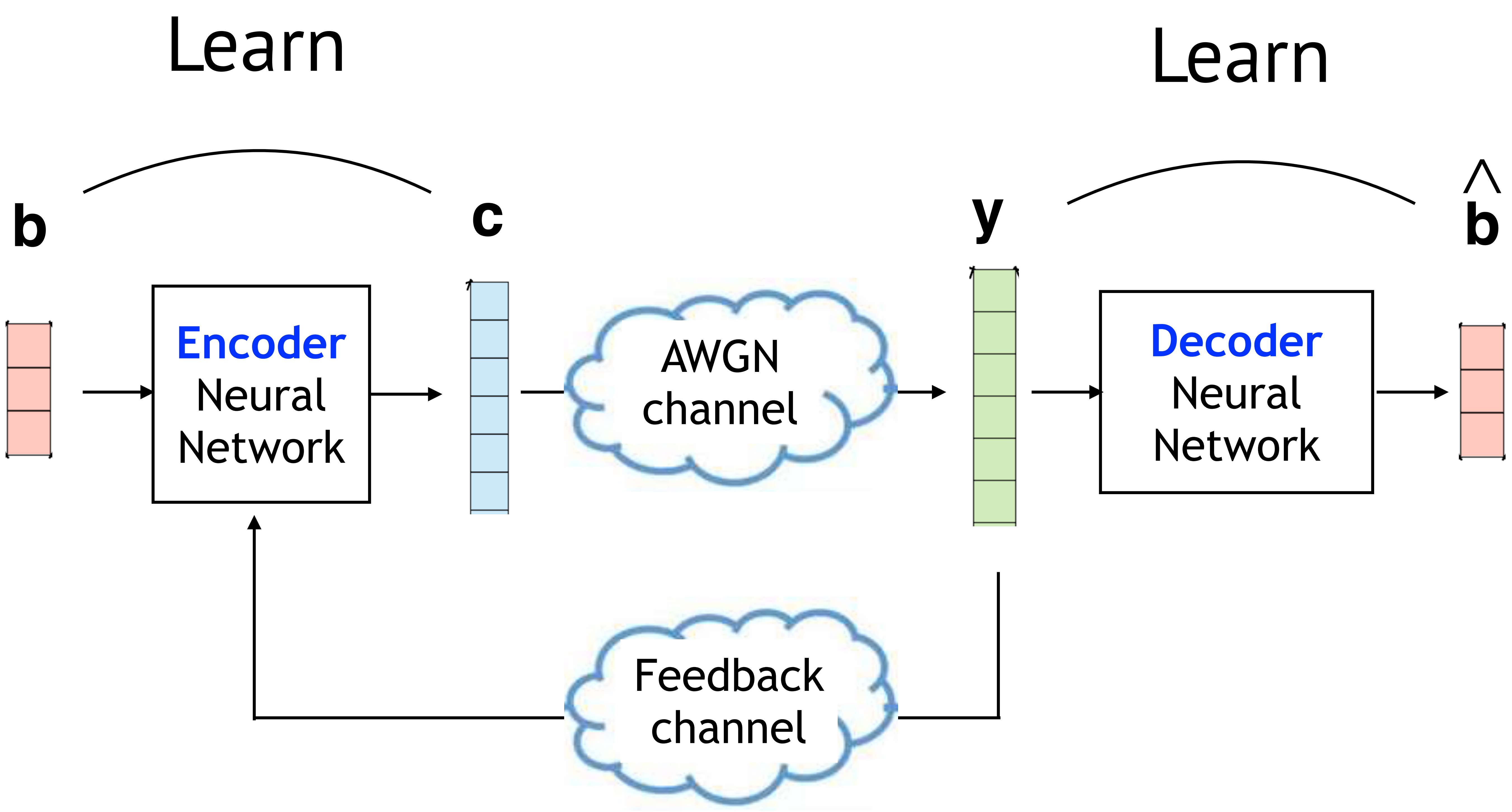
Outline - towards Deepcode

1. Neural network based encoder and decoder



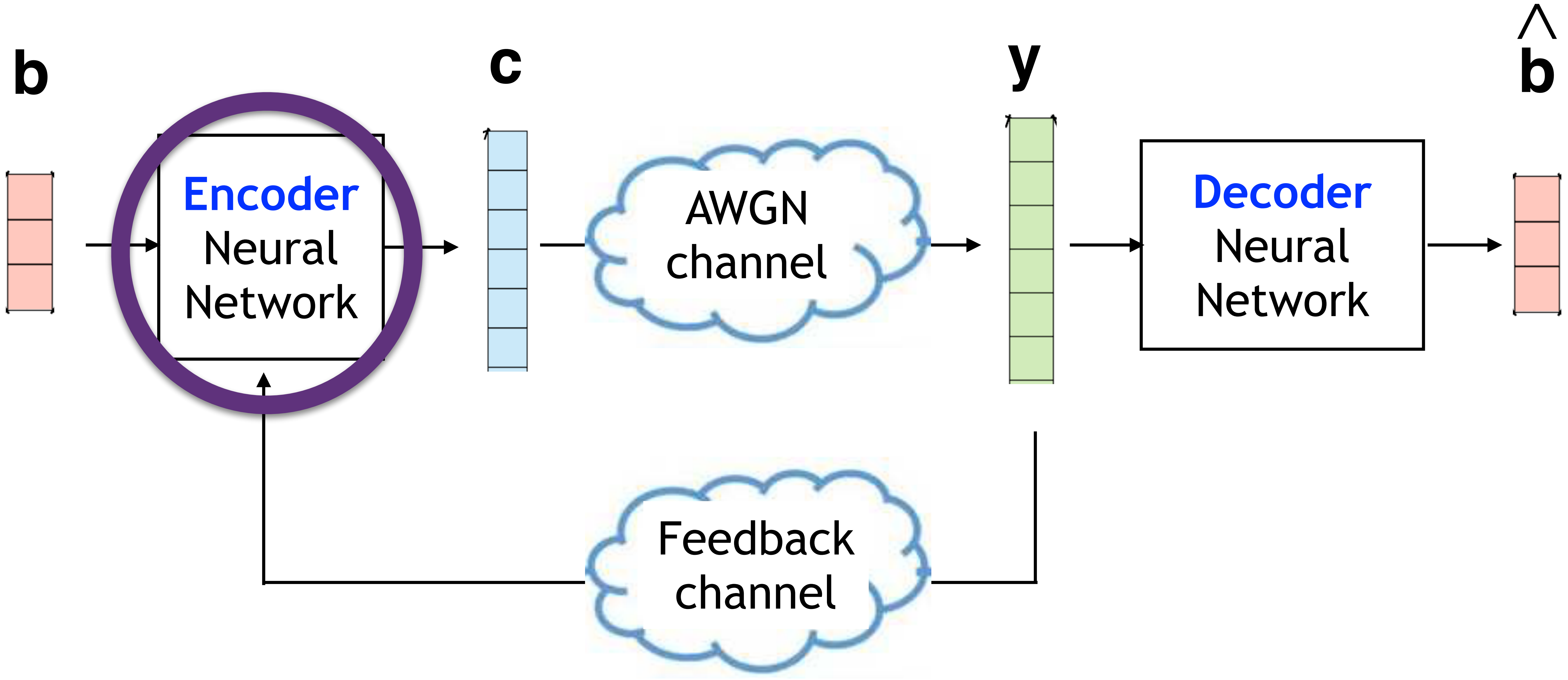
Outline - towards Deepcode

1. Neural network based encoder and decoder
2. Training



Outline - towards Deepcode

1. Neural network based encoder and decoder
2. Training
3. Modification on the encoder – "Deepcode"



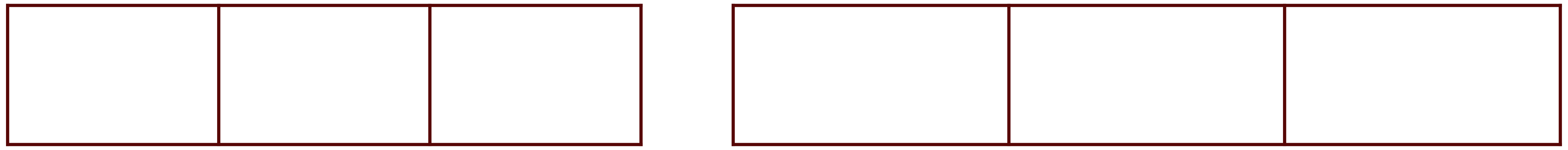
Encoder as a neural network

- Two-phase scheme
  - e.g. maps information bits b_1, b_2, b_3 to a length-6 code

Phase I.

Phase II.

codeword



Phase I: send information bits

Phase I.

codeword	b_1	b_2	b_3

Encoder receives feedback y_1 y_2 y_3

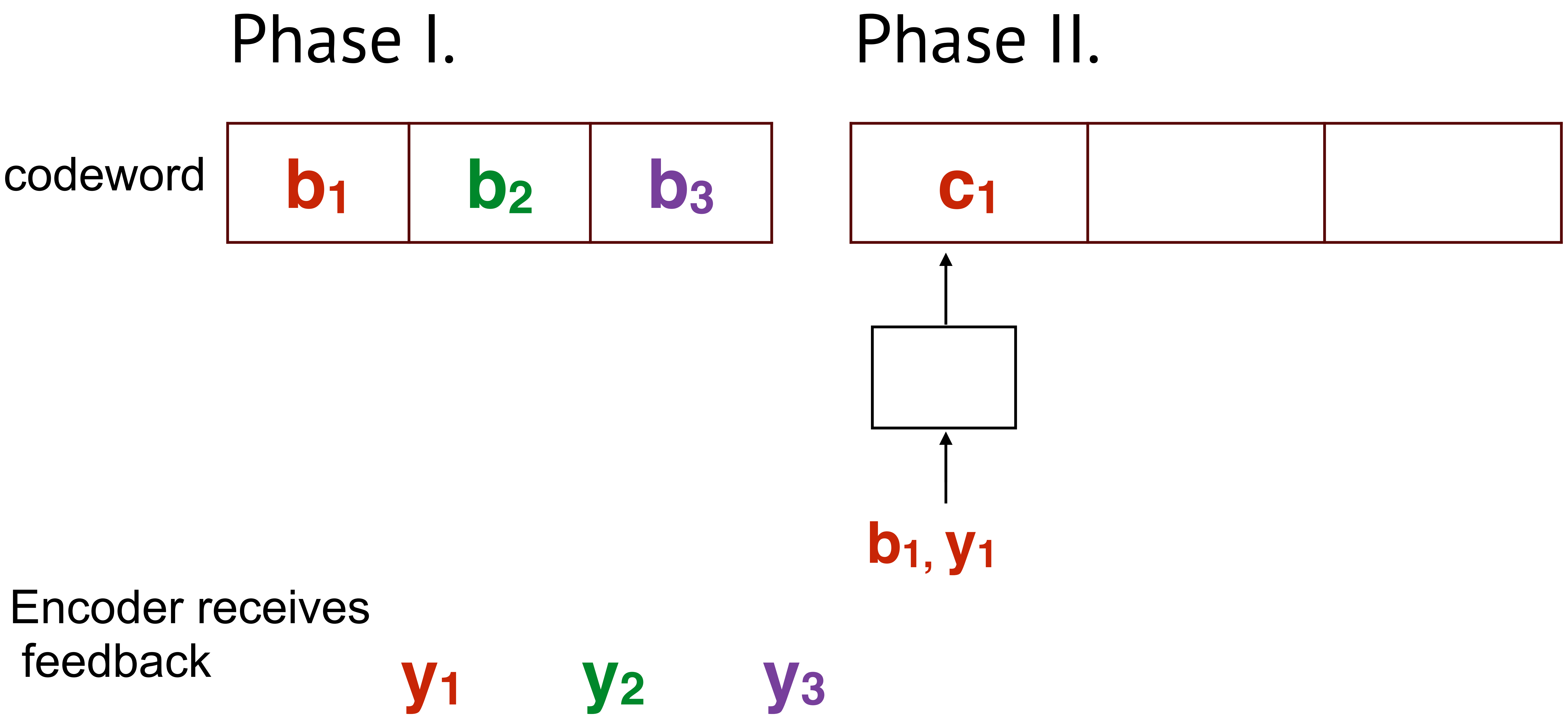
Phase II: use feedback to generate parity bits

	Phase I.			Phase II.		
codeword	b_1	b_2	b_3			

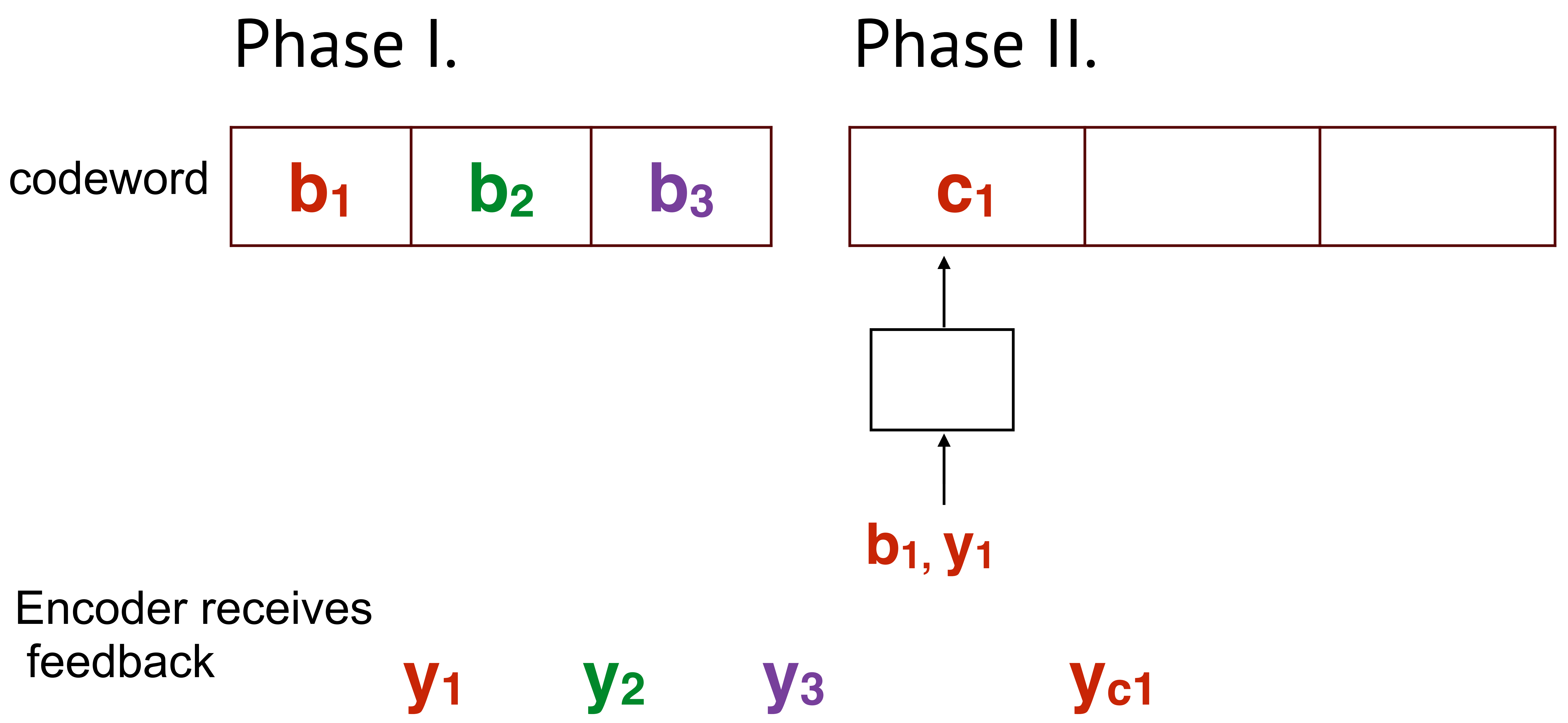
Encoder receives feedback y_1 y_2 y_3

Phase II: use feedback to generate parity bits

- Parity for b_1

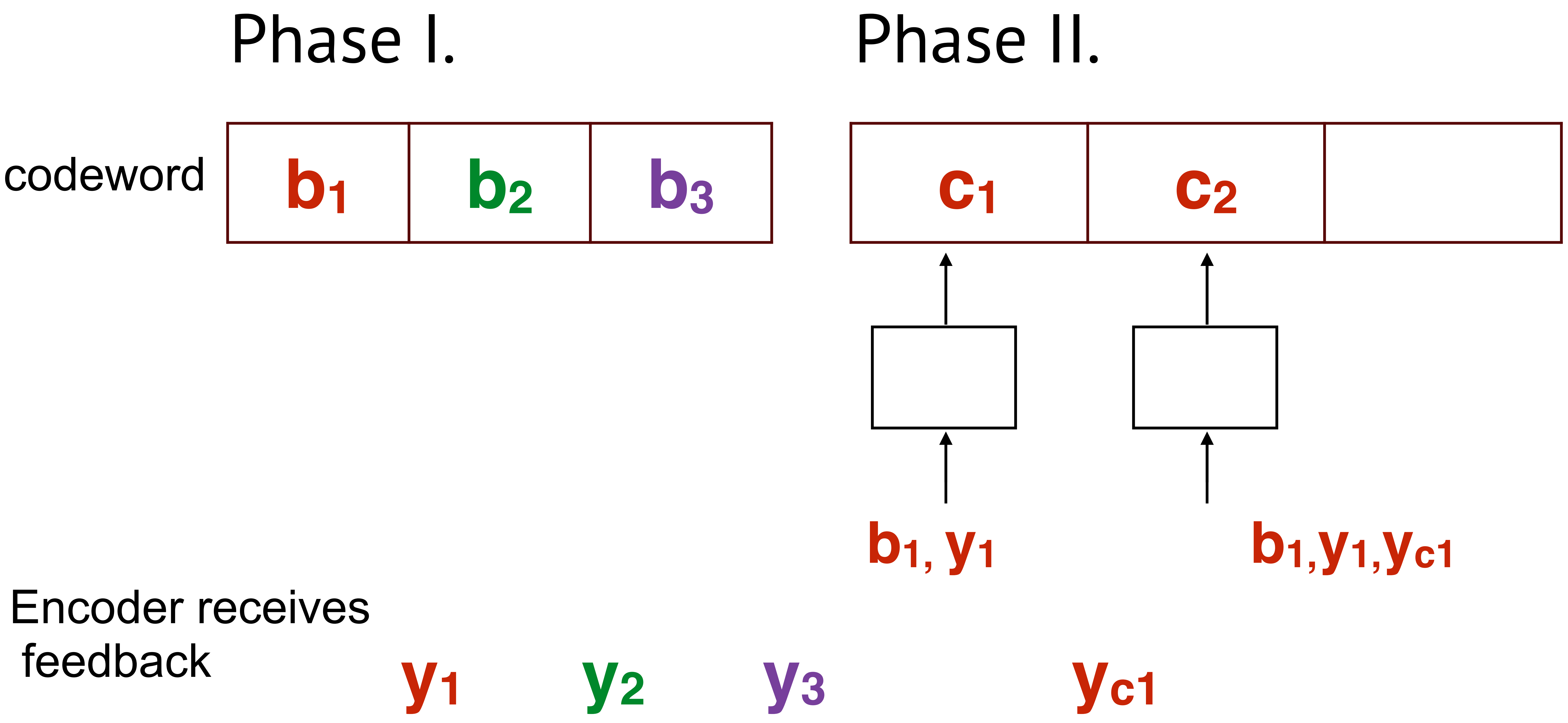


Phase II: use feedback to generate parity bits



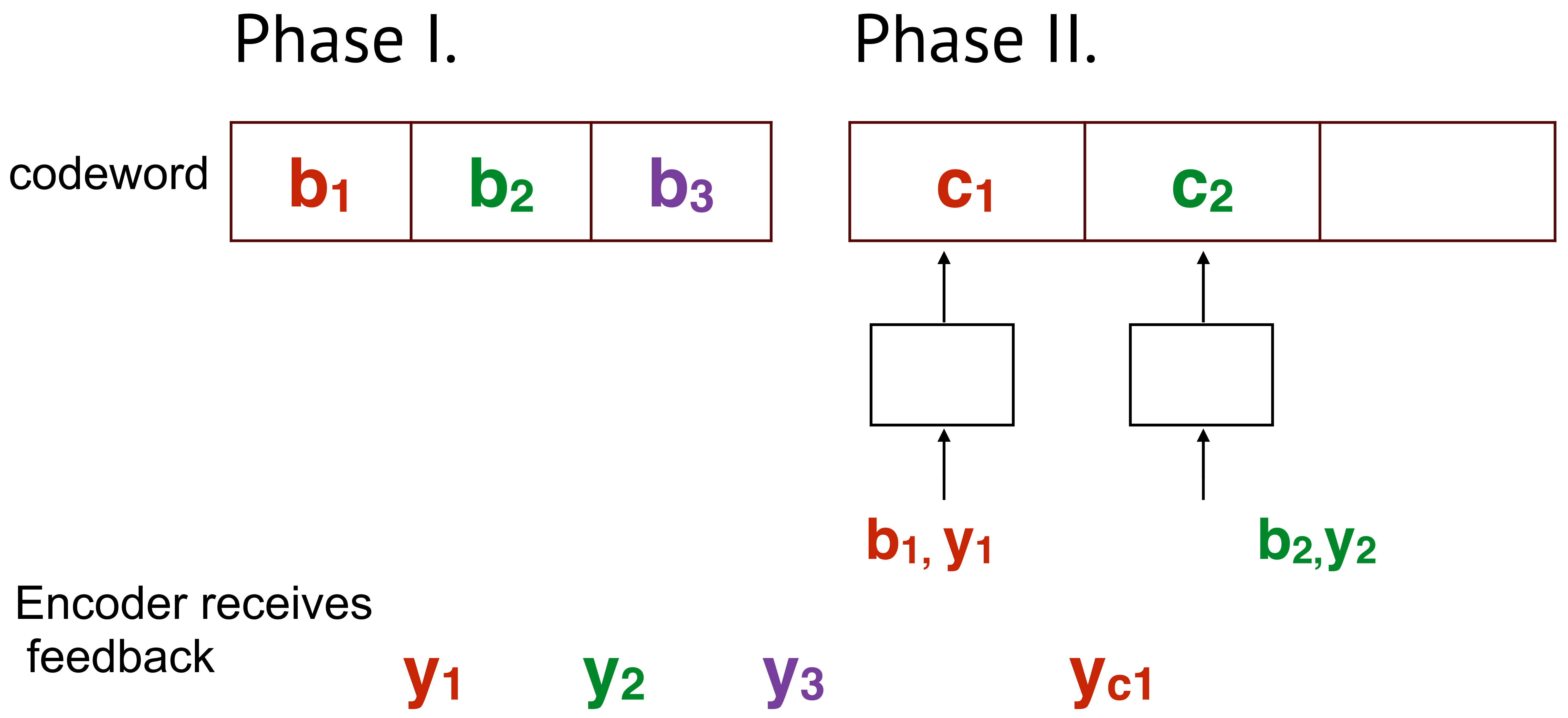
Phase II: use feedback to generate parity bits

- Another parity for b_1?



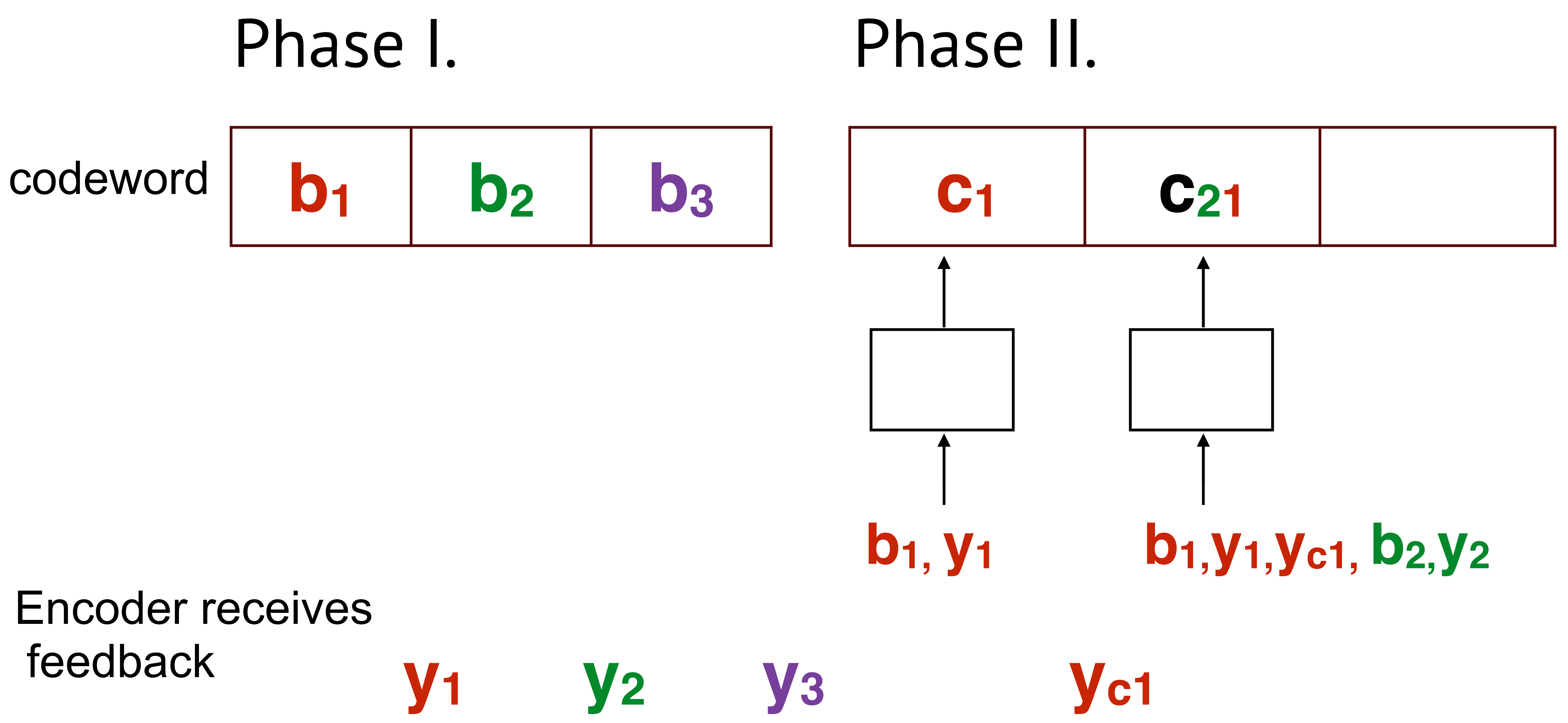
Phase II: use feedback to generate parity bits

- Parity for b_2?



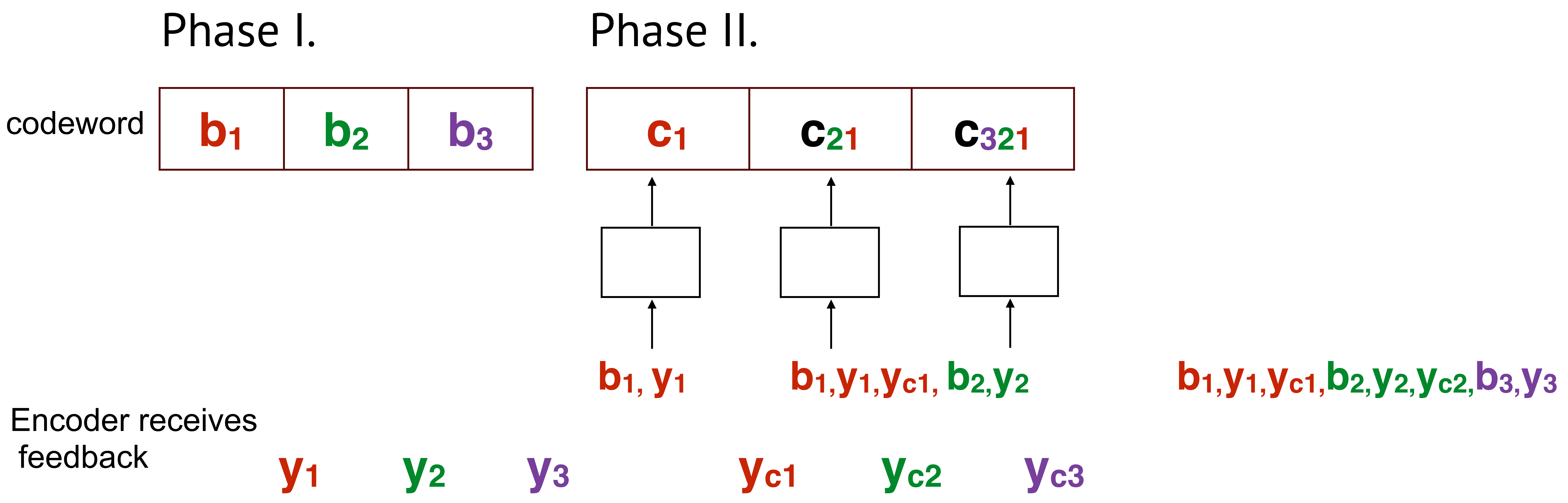
Phase II: use feedback to generate parity bits

- Parity for b_2 and b_1



Phase II: use feedback to generate parity bits

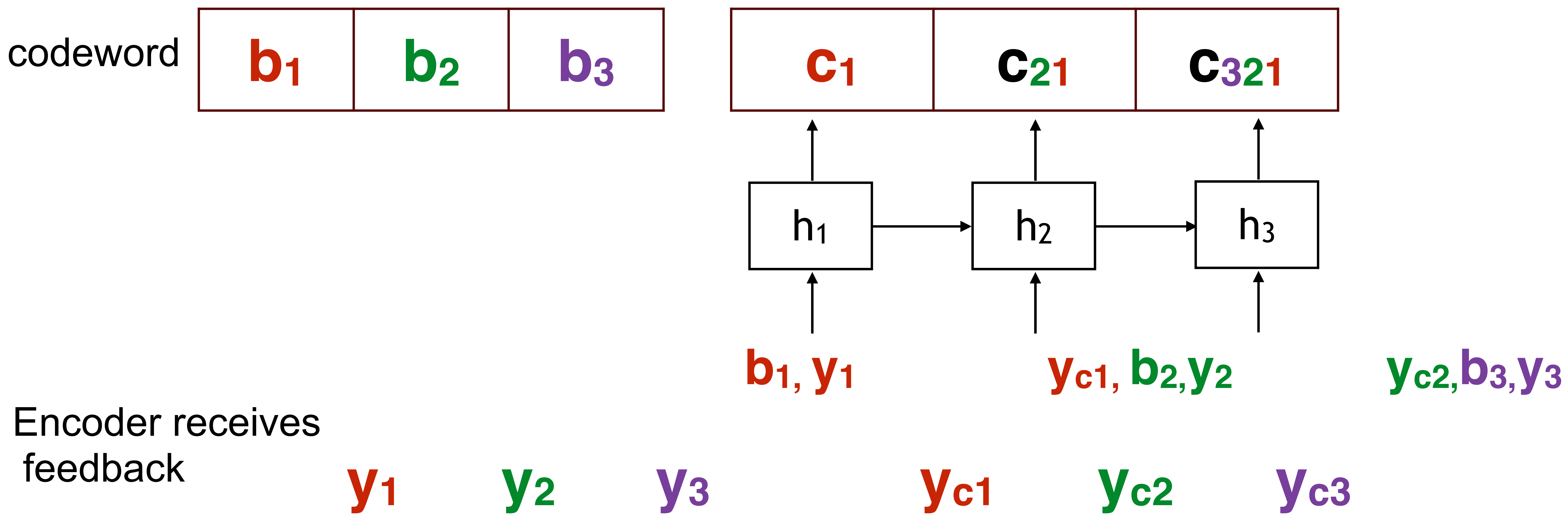
- Parity for b_3, b_2 and b_1



Recurrent Neural Network for parity generation

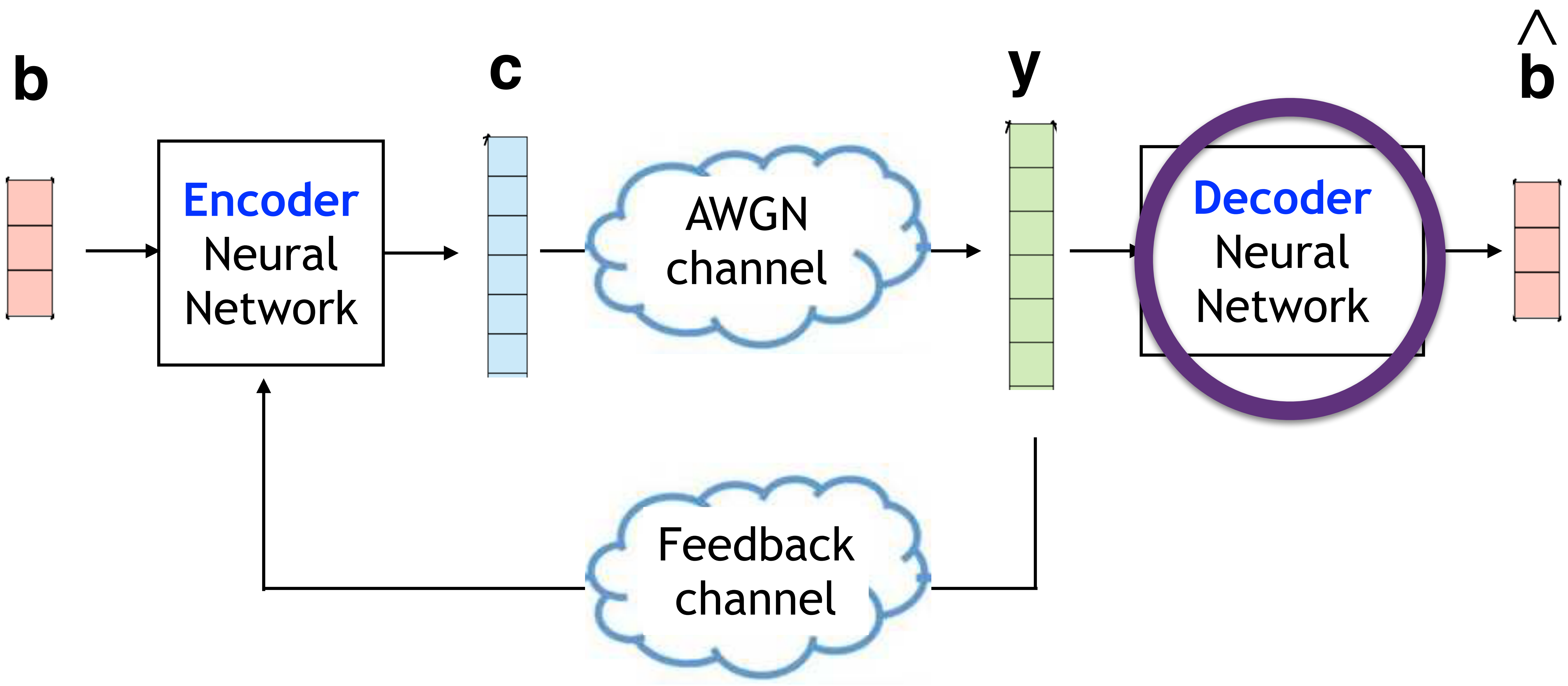
- Sequential mapping with memory

$$h_i = f(h_{i-1}, \mathrm{Input}_i)$$
$$\mathrm{Output}_i = g(h_i)$$



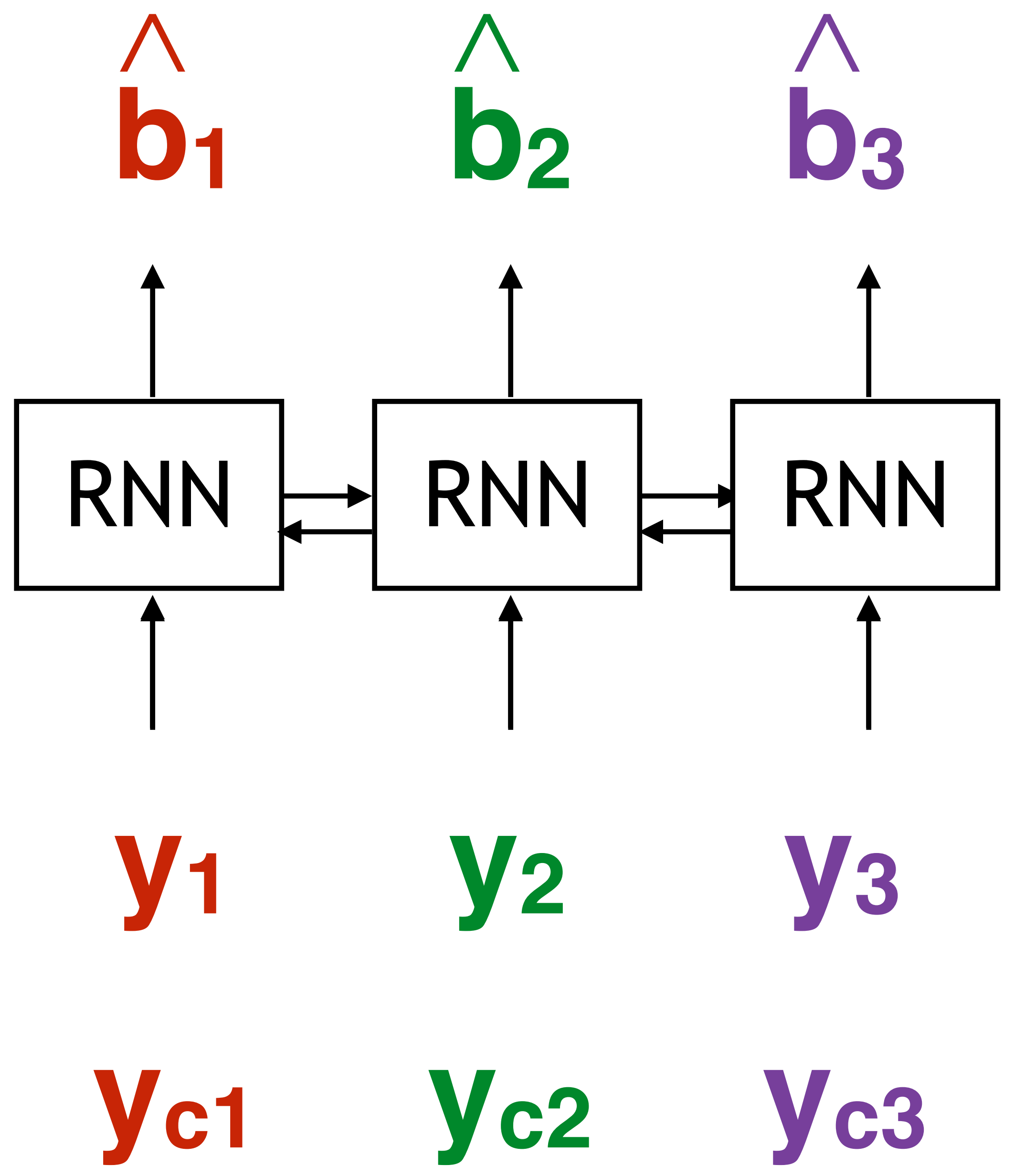
Outline - towards Deepcode

1. Neural network based encoder and decoder



Decoder as a recurrent neural network

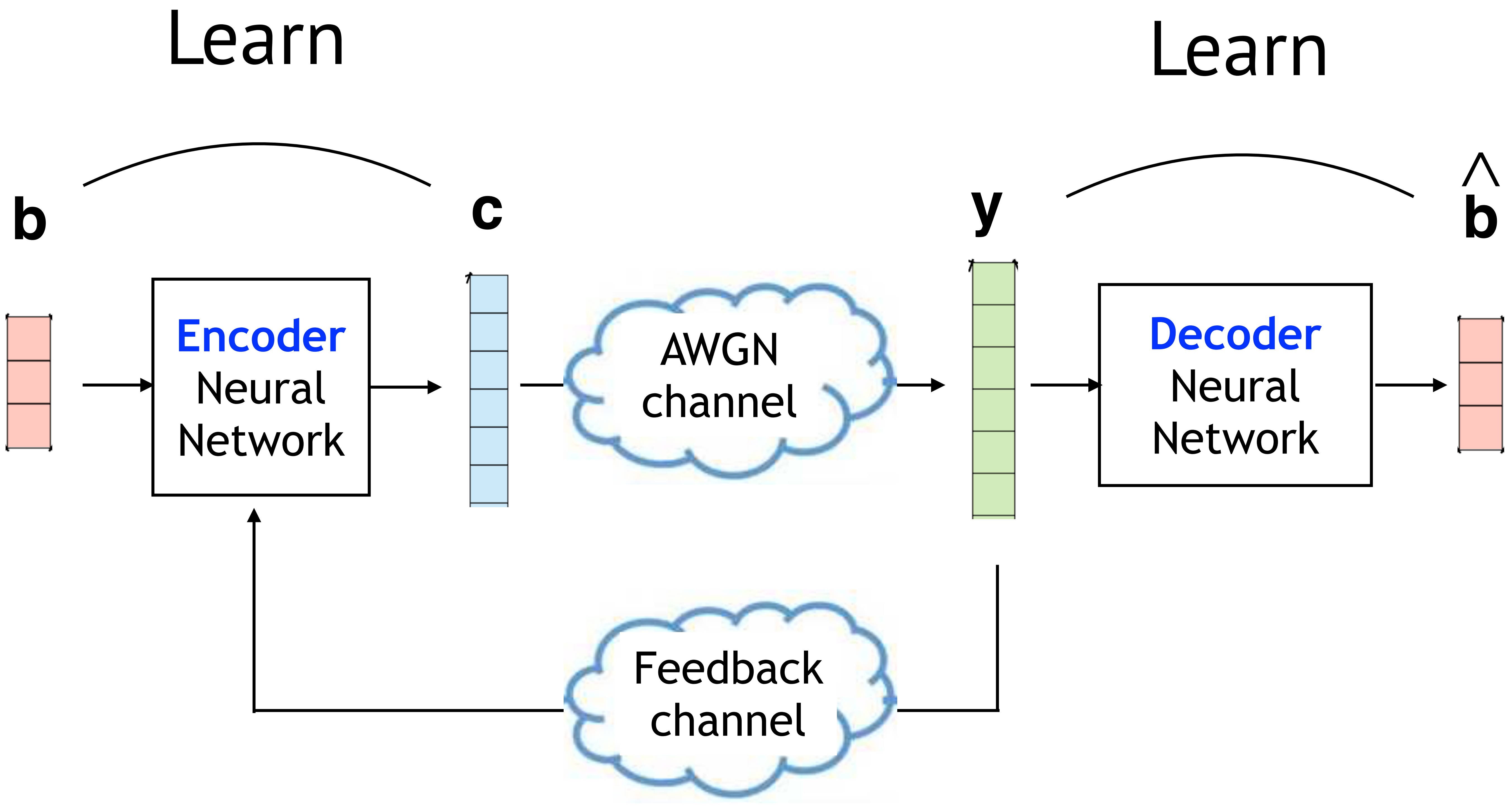
- Maps $(y_1, y_2, y_3, y_{c1}, y_{c2}, y_{c3})$ -> $\hat{b}_1, \hat{b}_2, \hat{b}_3$ via bi-directional RNN



Outline - towards Deepcode

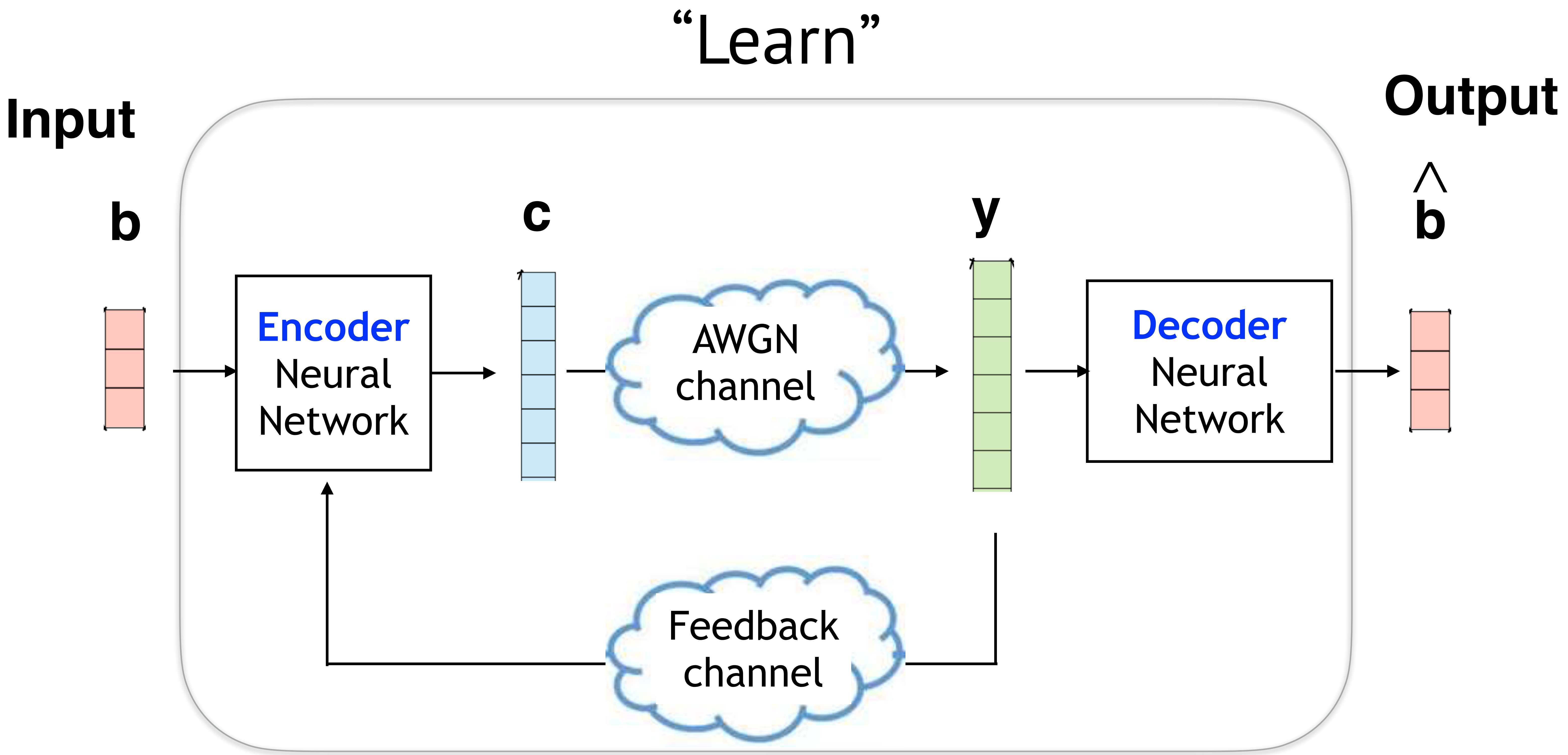
1. Neural network based encoder and decoder

2. Training



Training

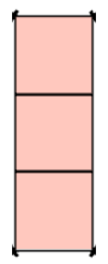
- Learn encoder and decoder jointly– autoencoder training



Training

- Generate random bit sequences **b** of length K

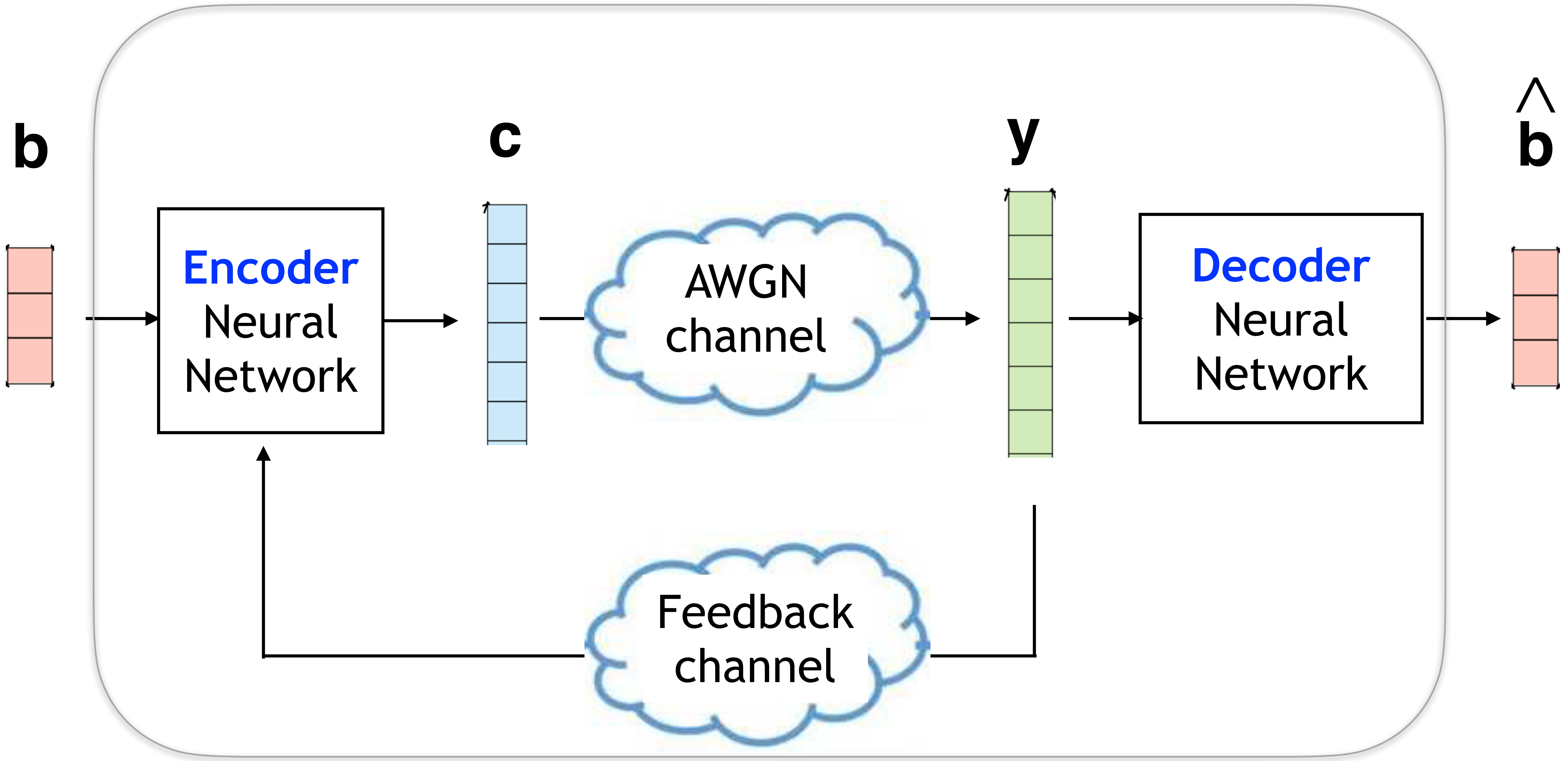
b



Training

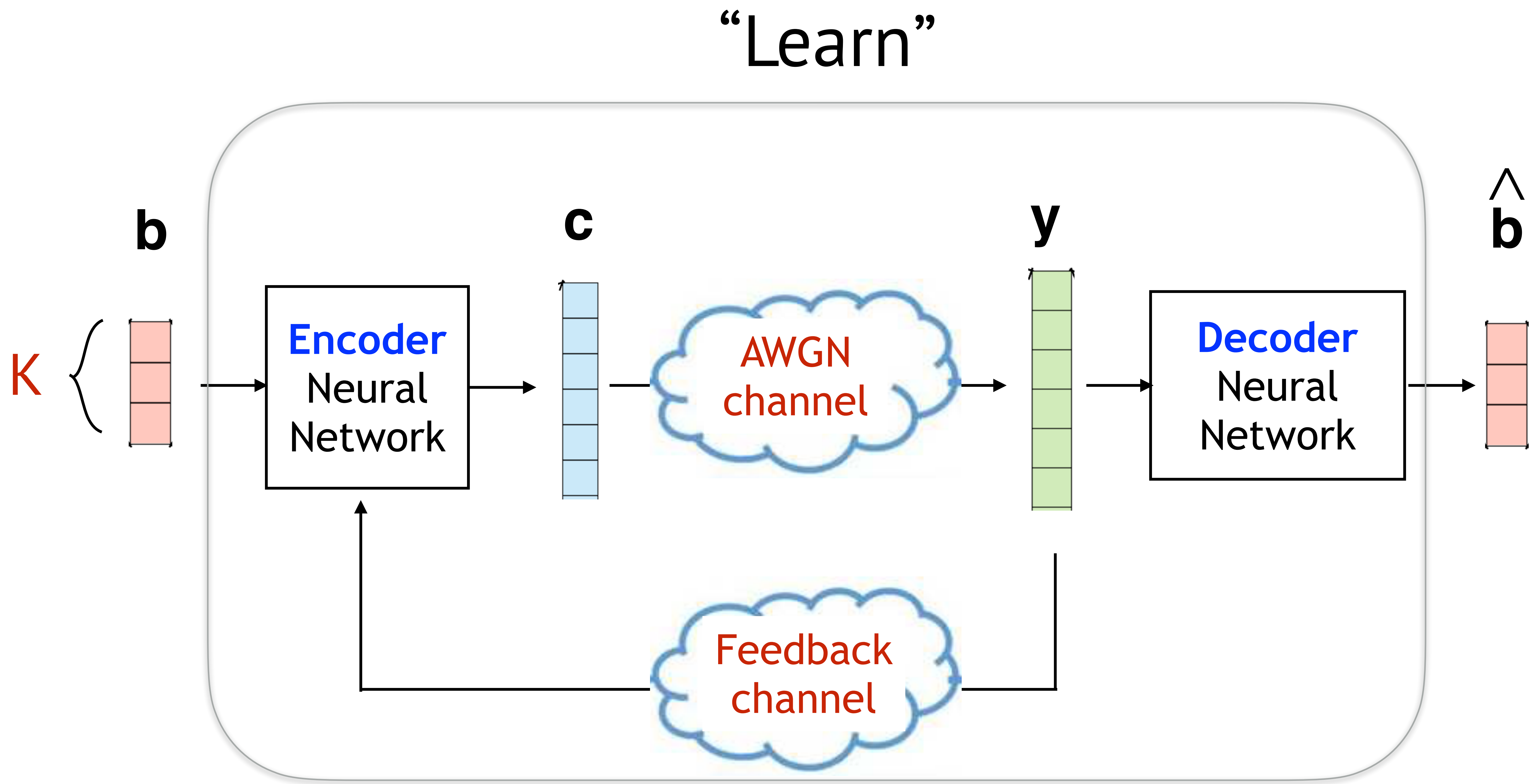
- Auto-encoder training : (input,output) = (**b**,**b**)
- Loss : binary cross entropy $\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = -\mathbf{b} \log \hat{\mathbf{b}} - (1-\mathbf{b}) \log(1-\hat{\mathbf{b}})$

“Learn”



Choice of training examples

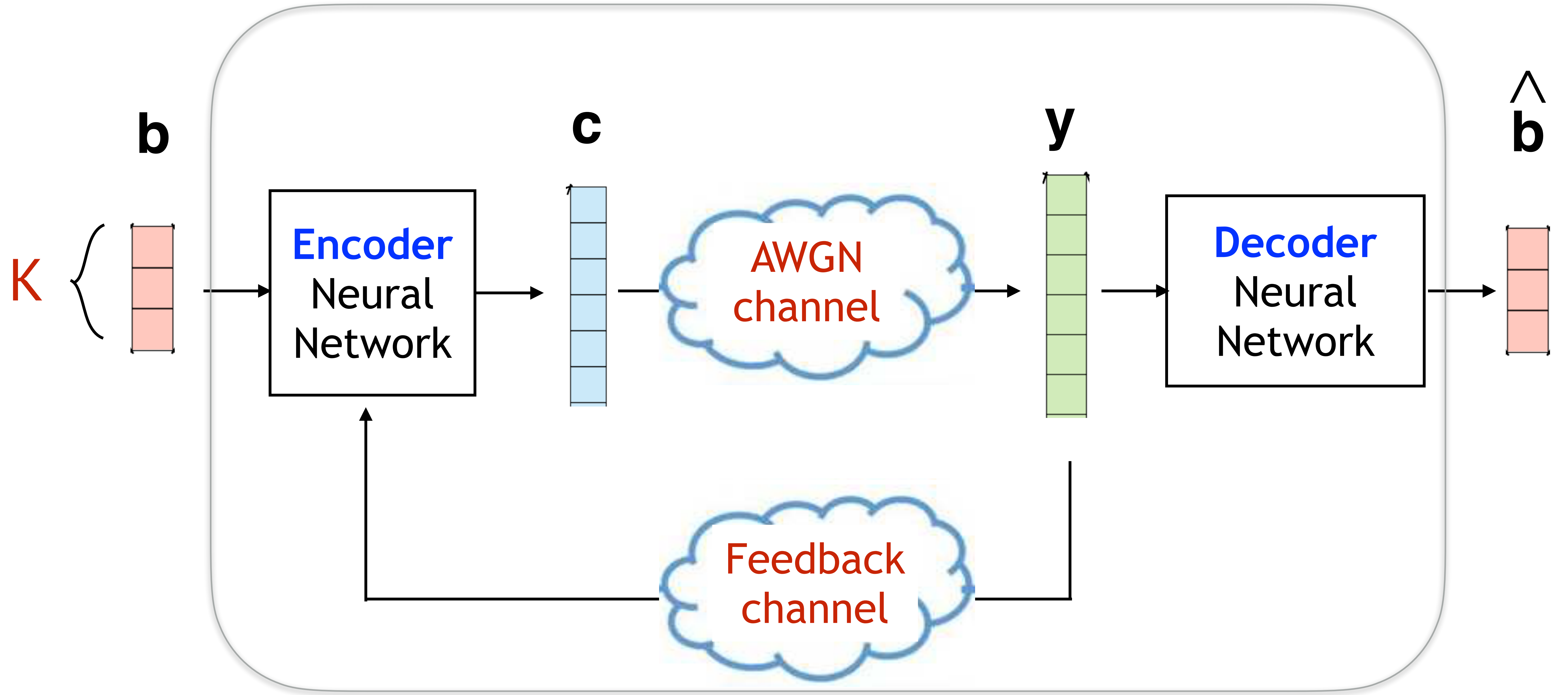
- Length of binary bit sequence **b**
- SNR of AWGN/feedback channels



Choice of training examples

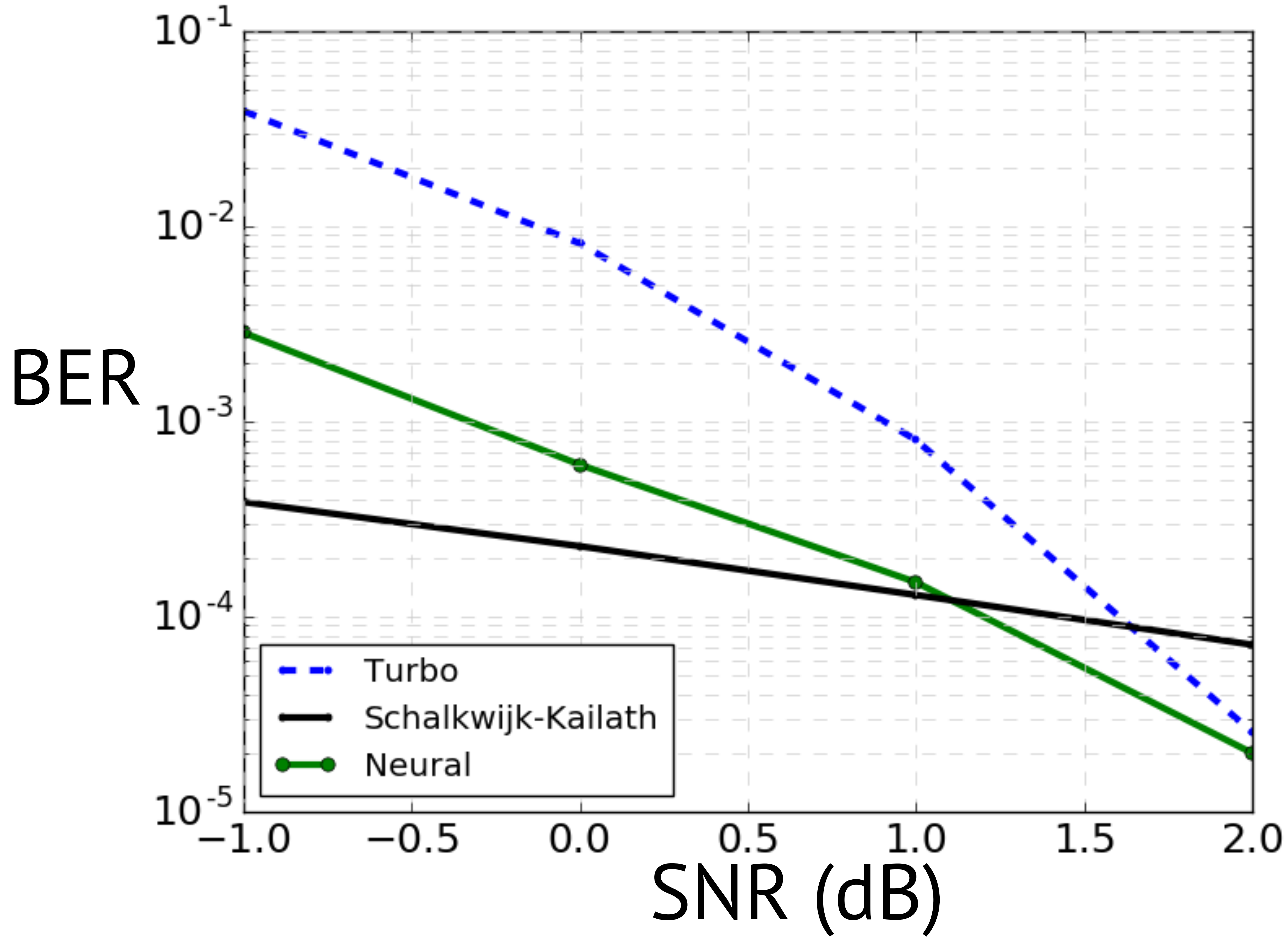
- Length of binary bit sequence **b** — K = 100
- SNR of AWGN/feedback channels — matched to test SNR

“Learn”



Intermediate result

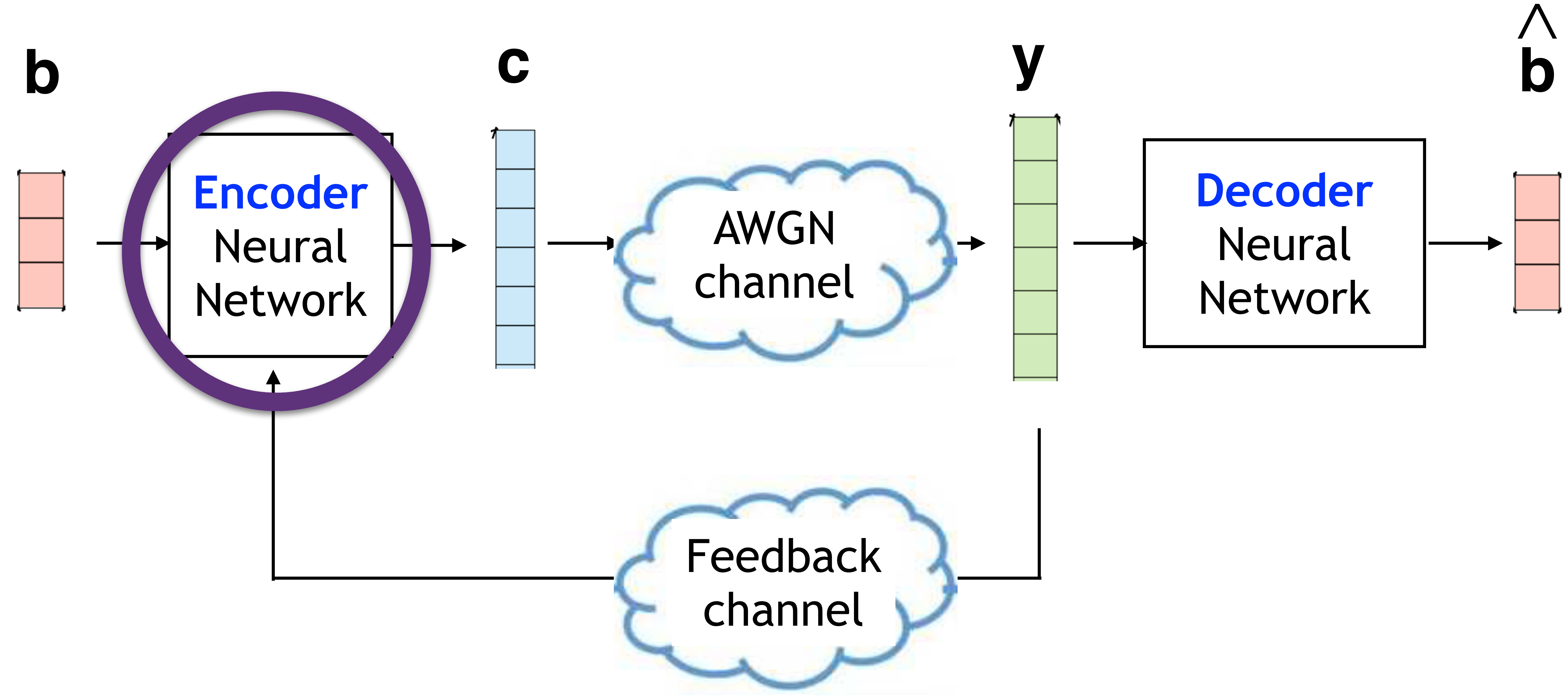
- Outperforms S-K for a small range of SNR



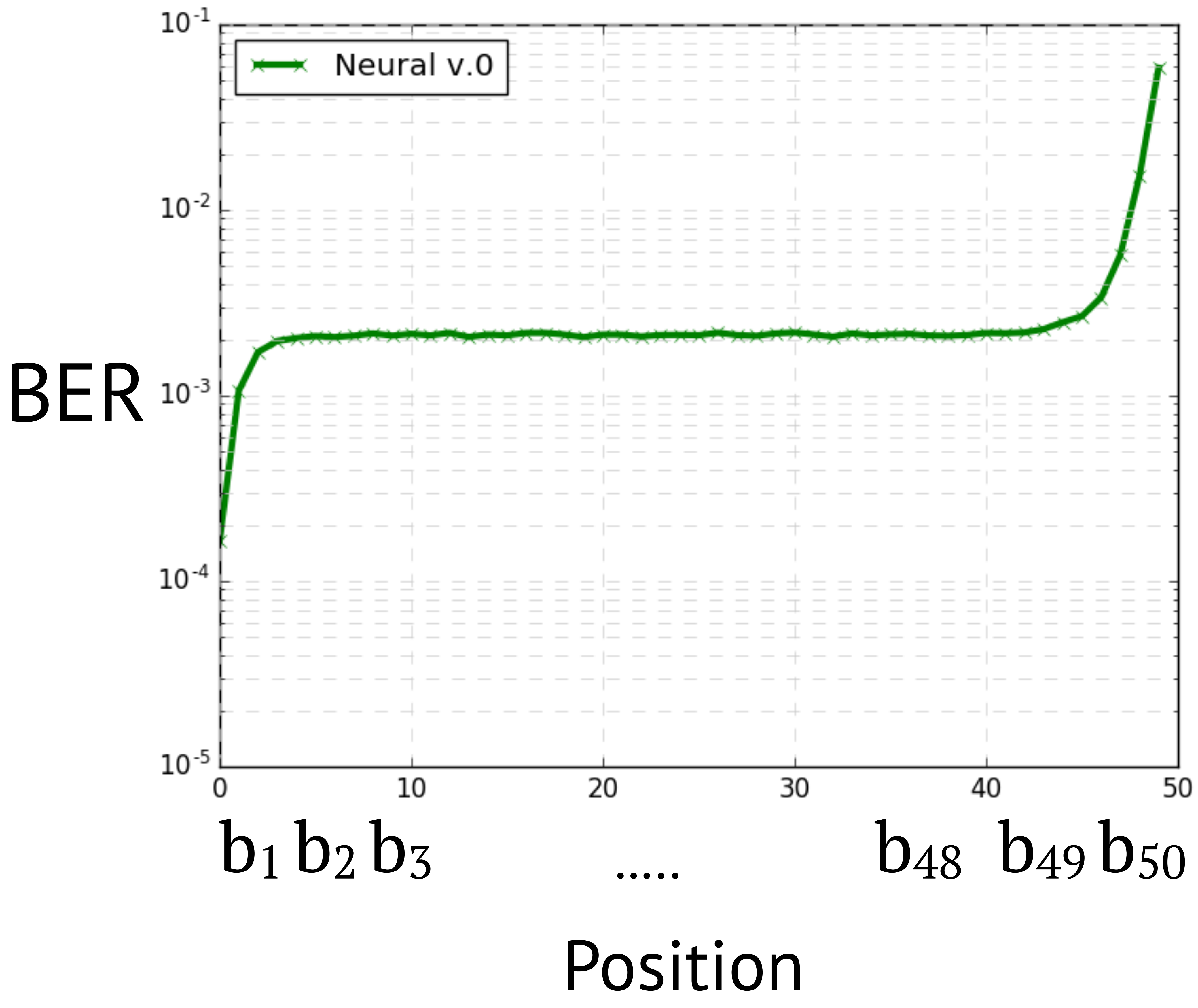
(50 bits, rate 1/3, noiseless feedback)

Outline - towards Deepcode

1. Baseline encoder and decoder
2. Training
3. Modification on the encoder - “Deepcode”

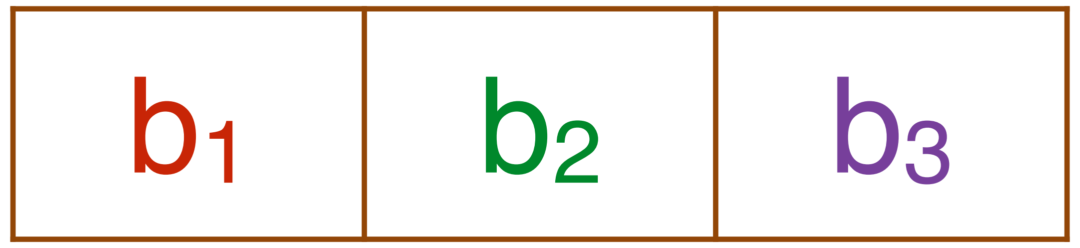


High error in the last bits

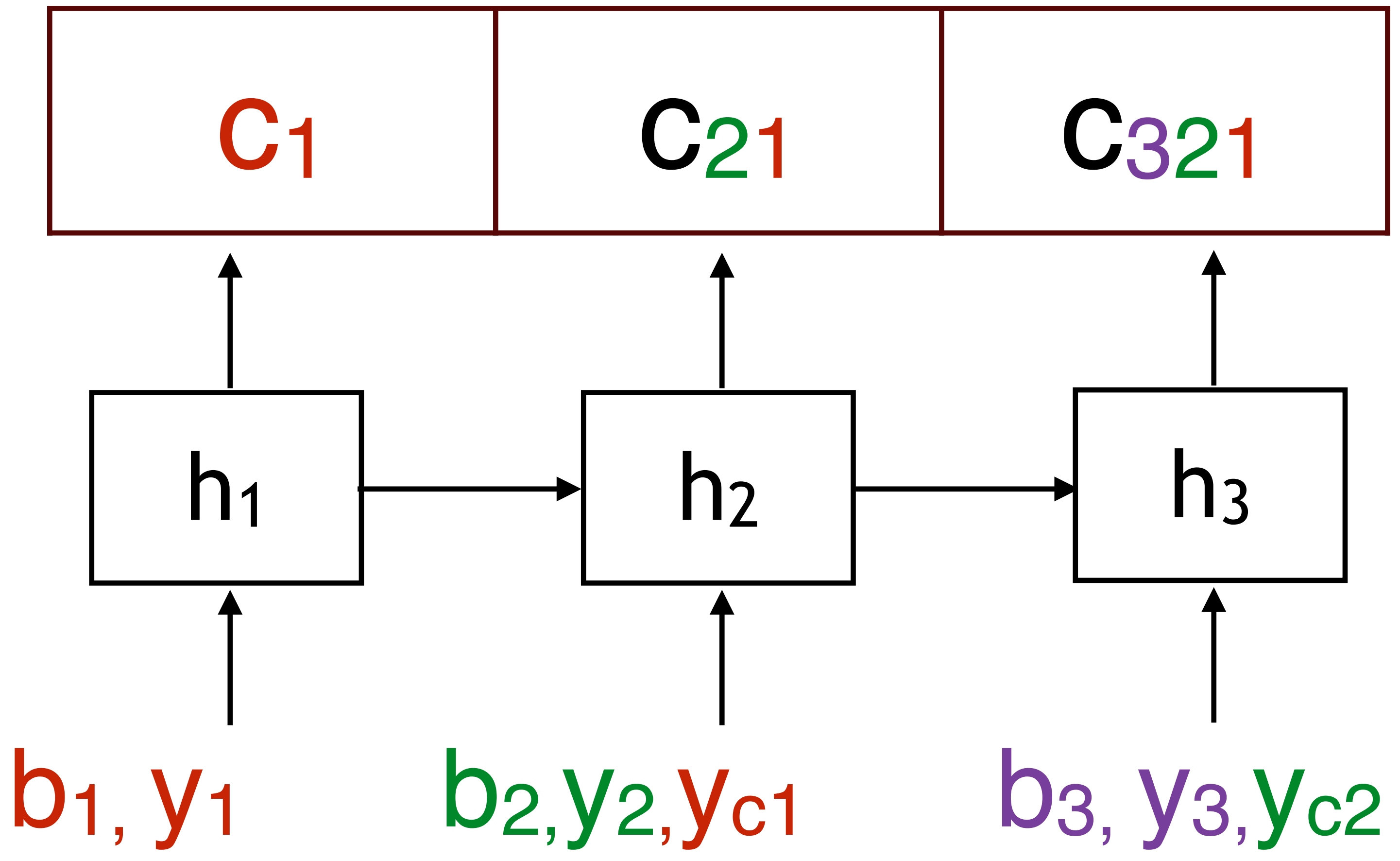


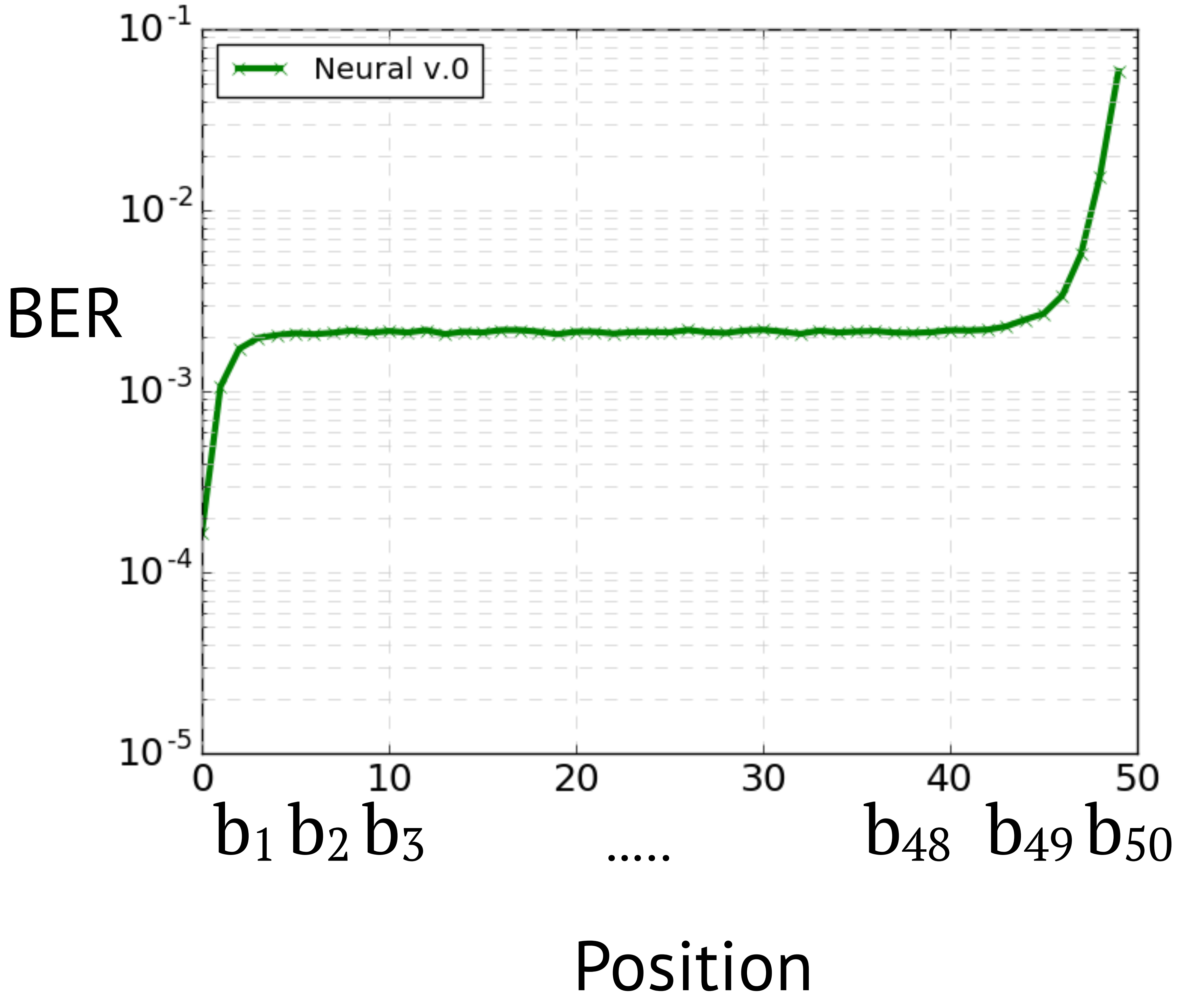
High error in the last bits

Phase I.



Phase II.



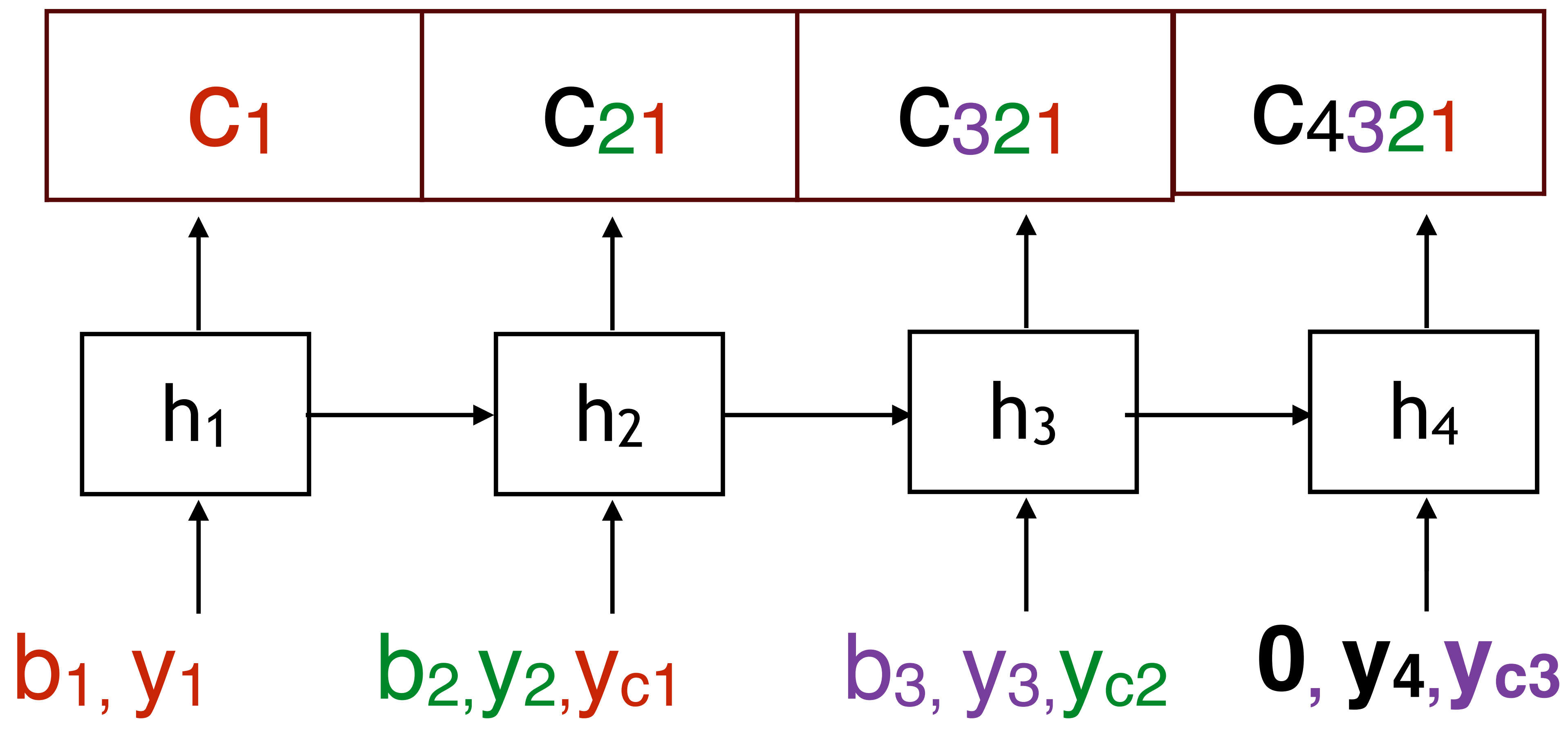


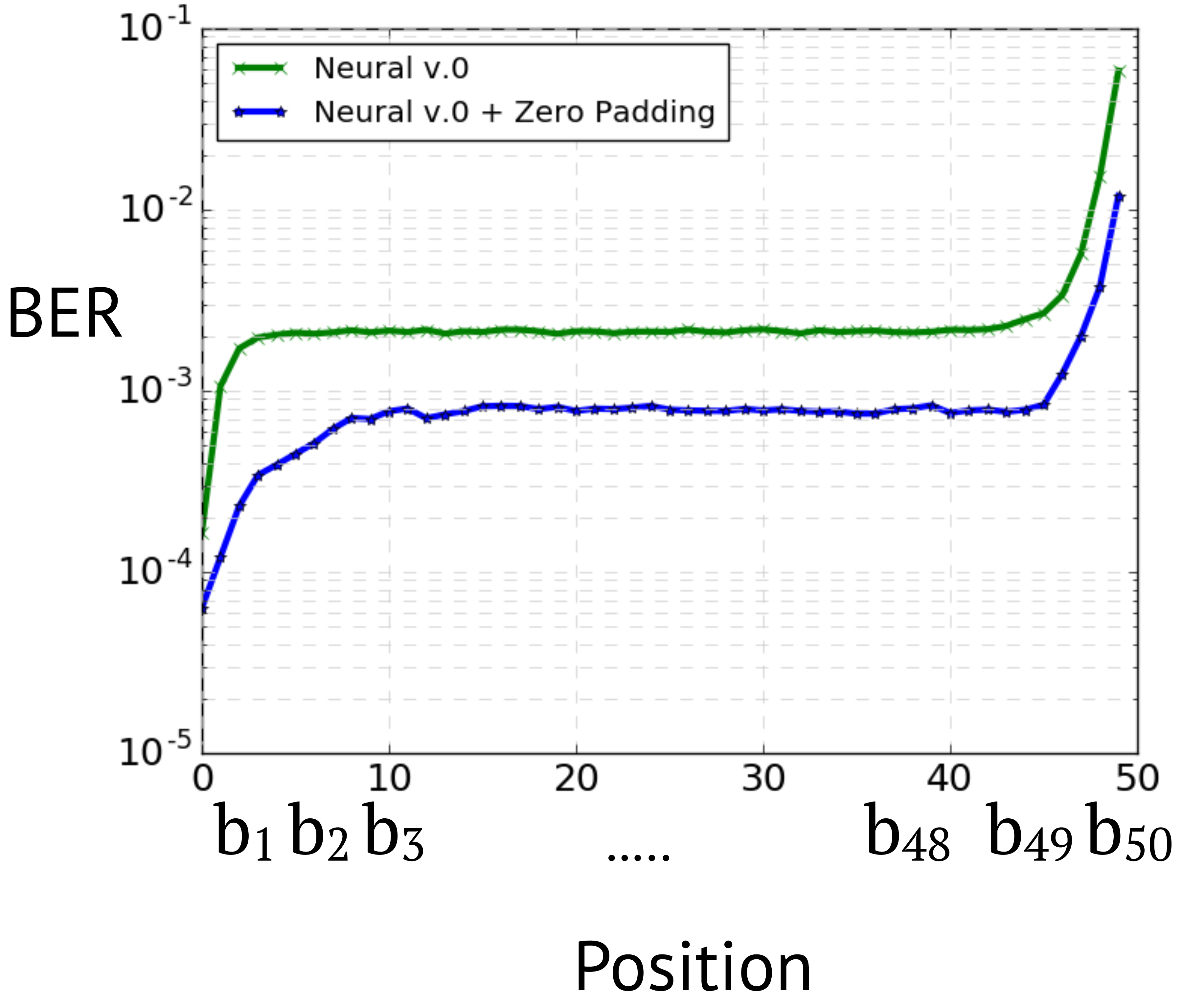
Encoder modification 1. Zero padding

Phase I.

b1	b2	b3	0

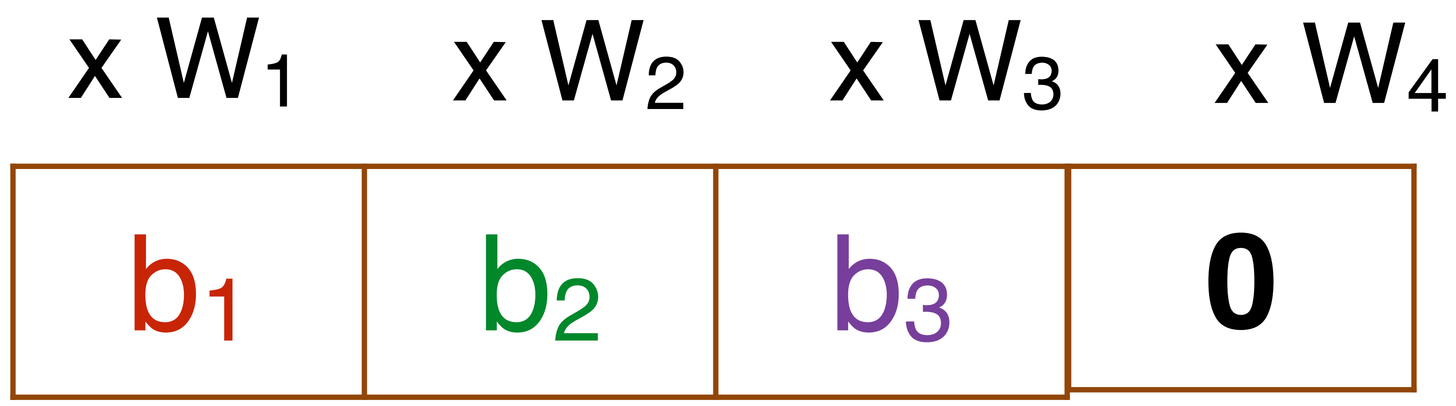
Phase II.



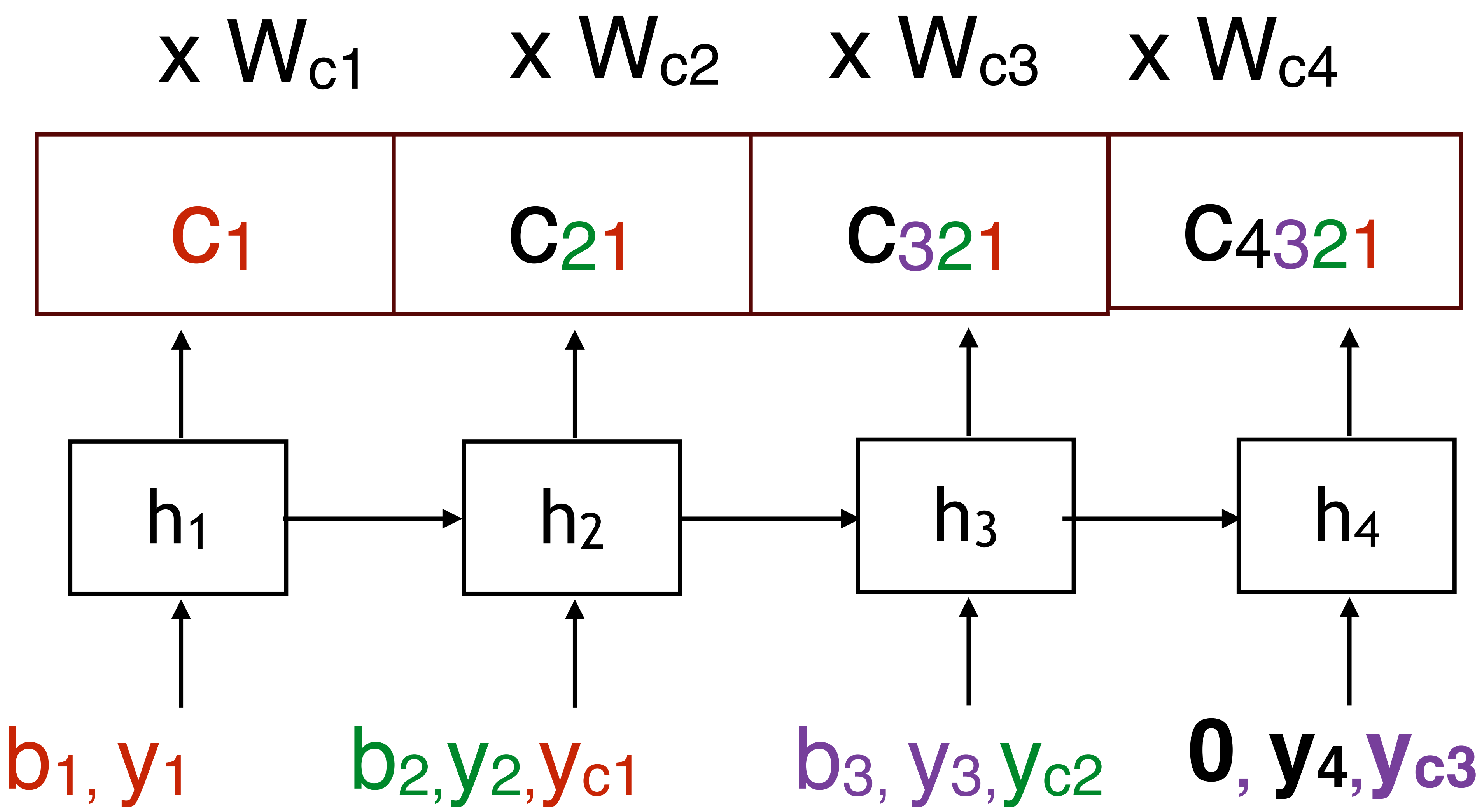


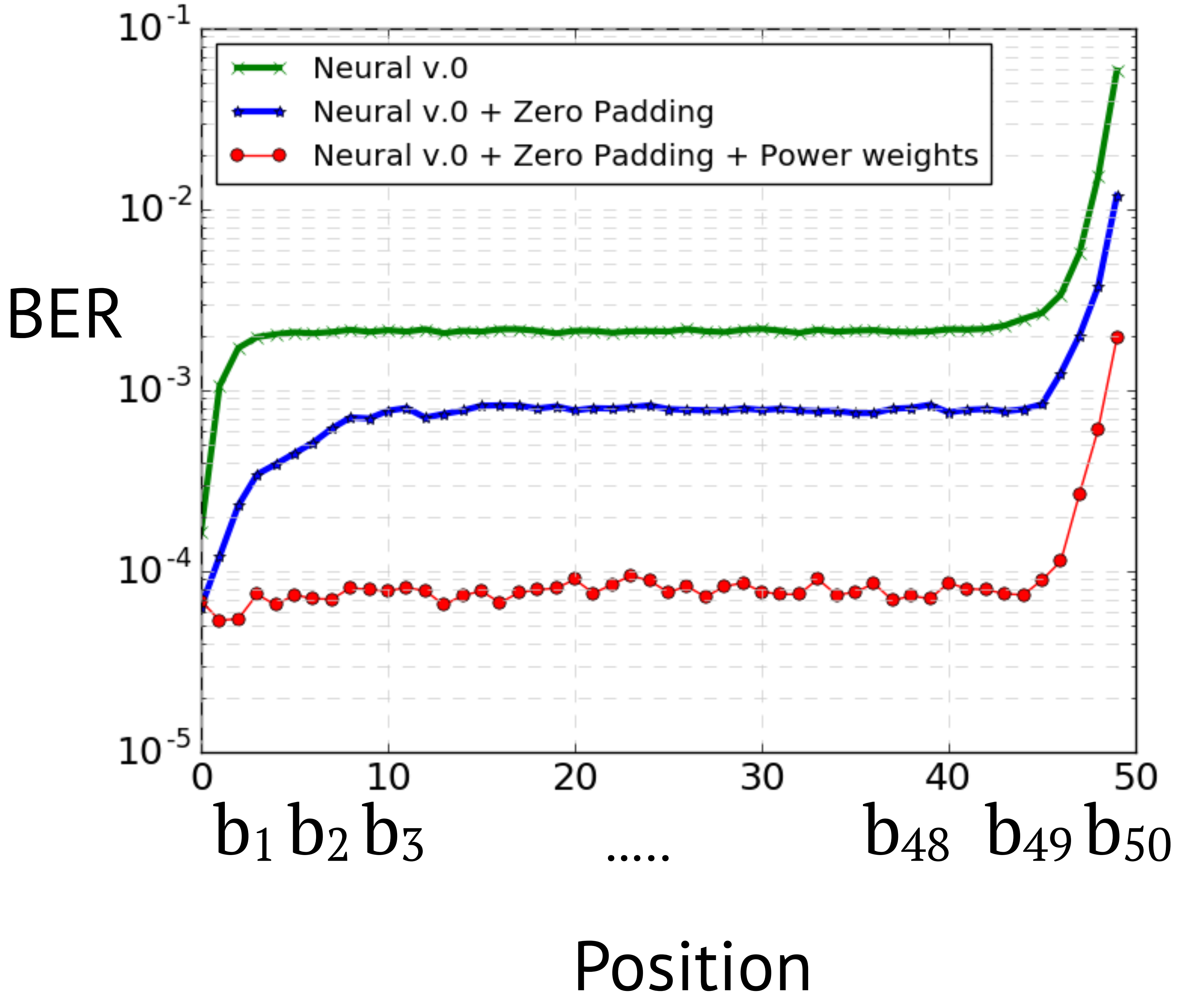
Encoder modification 2. Power allocation

Phase I.



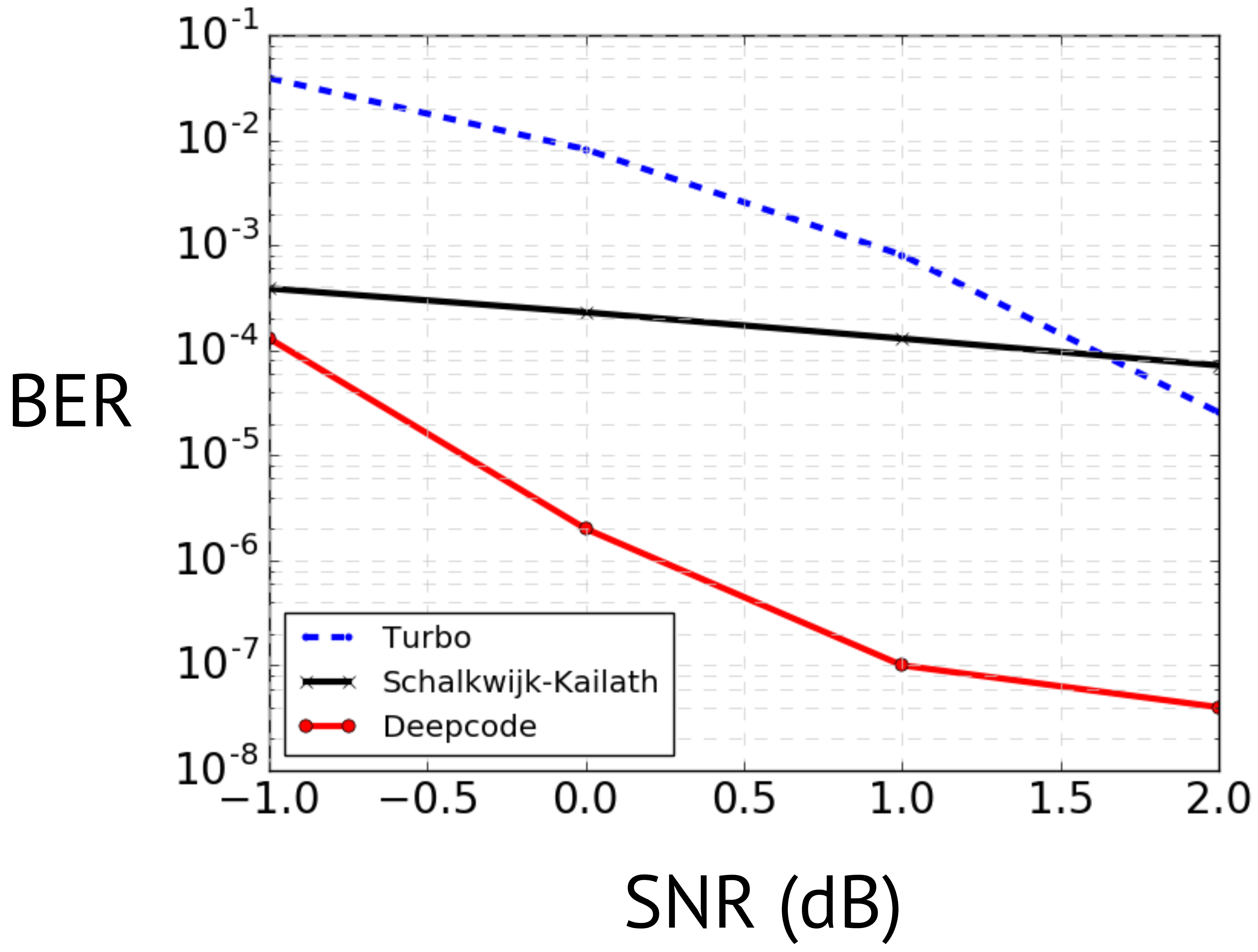
Phase II.





Final results

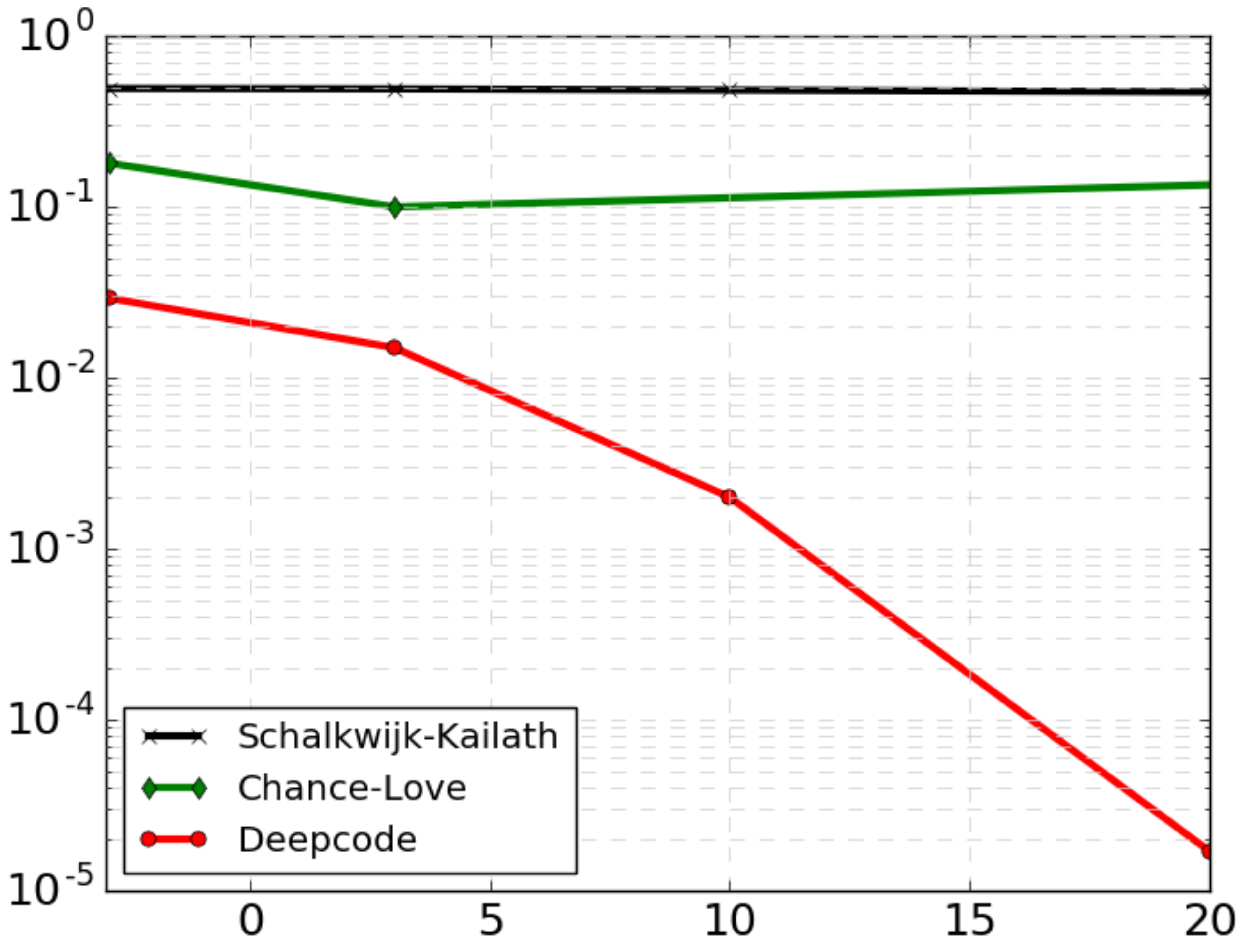
- 100x improvement for noiseless feedback w. precision



Final results

- Outperforms state-of-the-art for AWGN feedback channel

BER

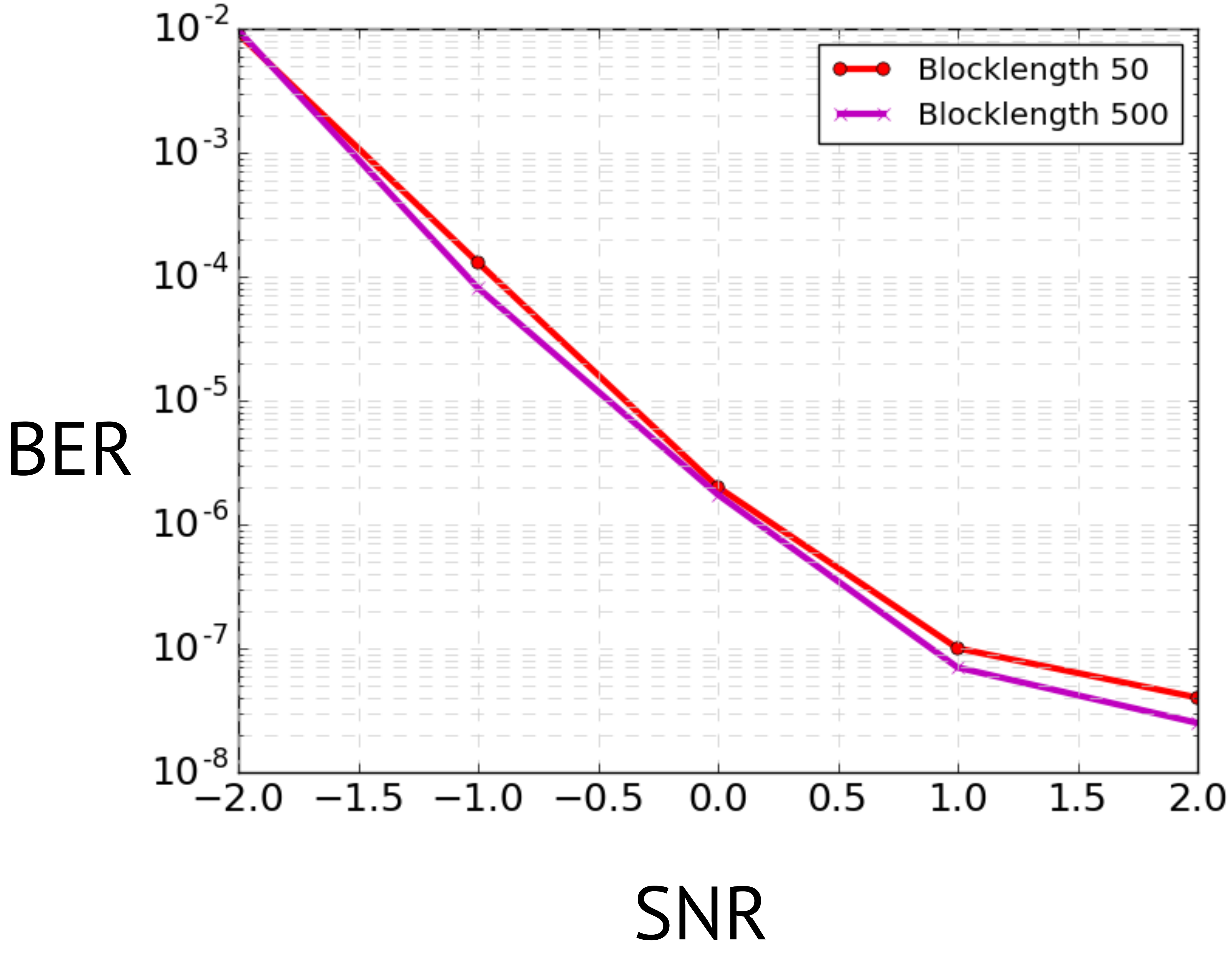


Feedback SNR (dB)

(Rate 1/3, 50 bits, forward SR = 0dB)

Generalization: block lengths

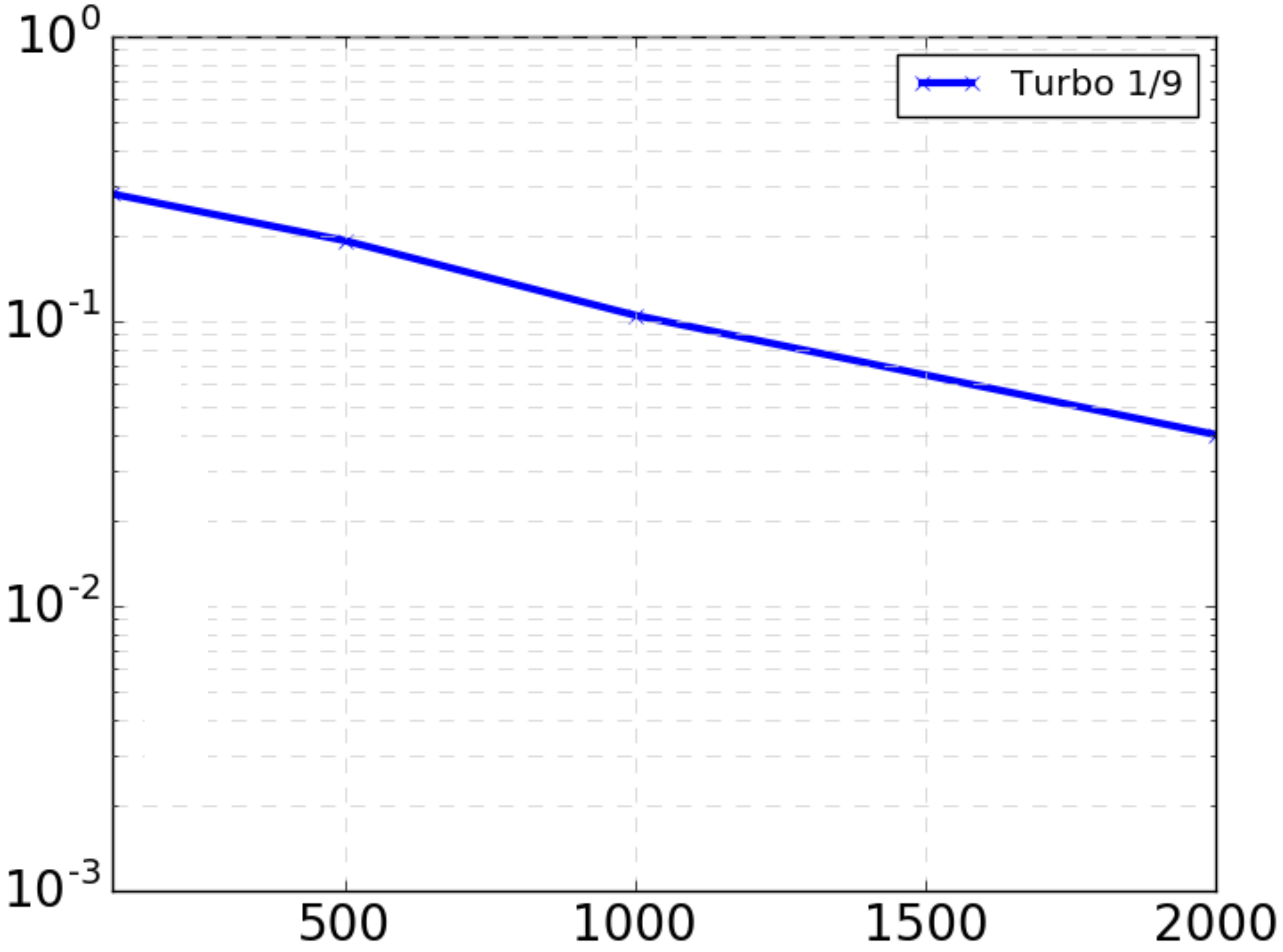
- Train on block length 100. Test on block lengths 50 & 500



Improved error exponents

- Non-feedback scheme: BLER ↓ as block length ↑

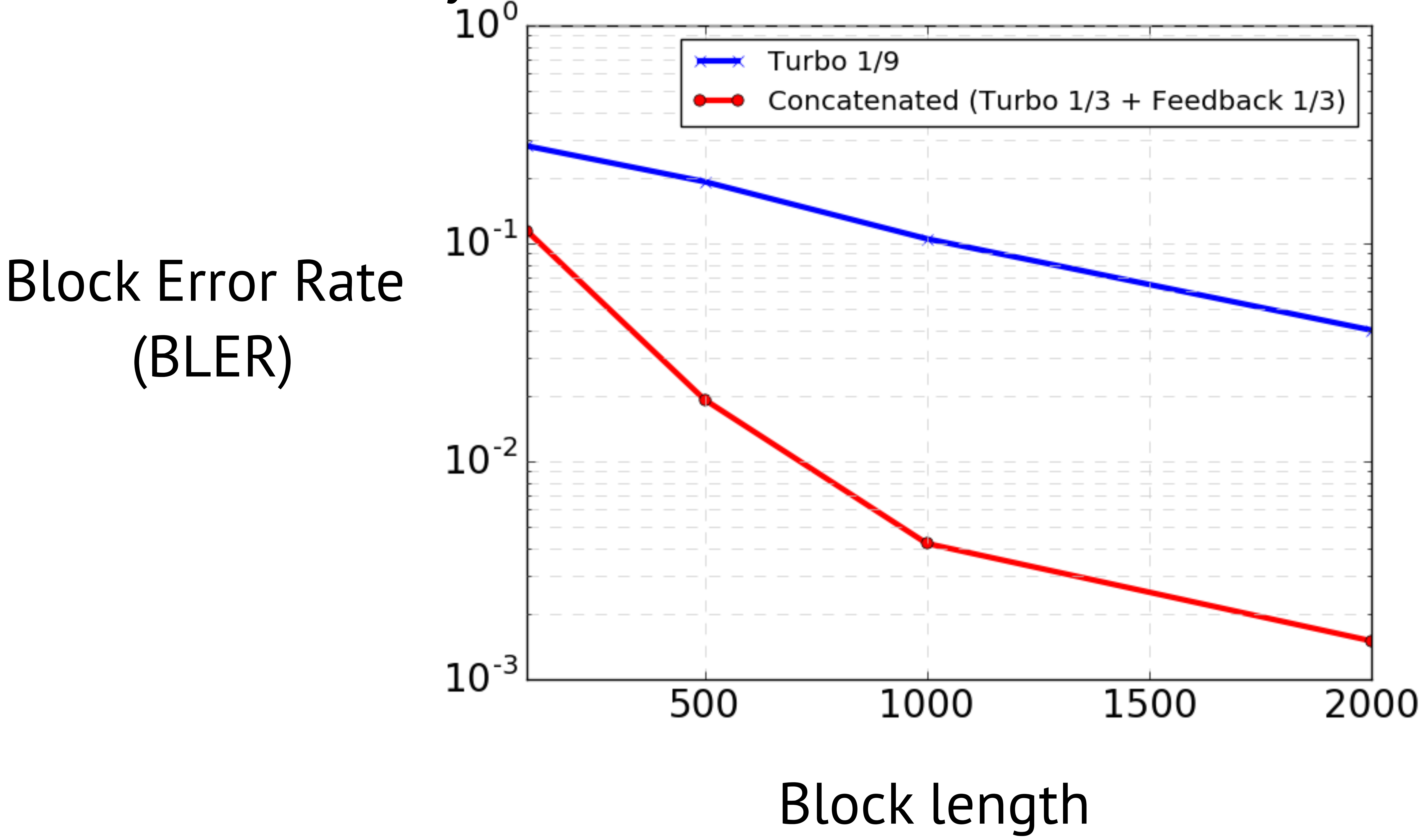
Block Error Rate (BLER)



Block length

Improved error exponents

- Concatenated code: turbo + Deepcode
  - BLER decays faster



Summary

- Deep learning based code (encoder-decoder)
  - Significantly more reliable for channels with feedback
  - Key: neural architecture

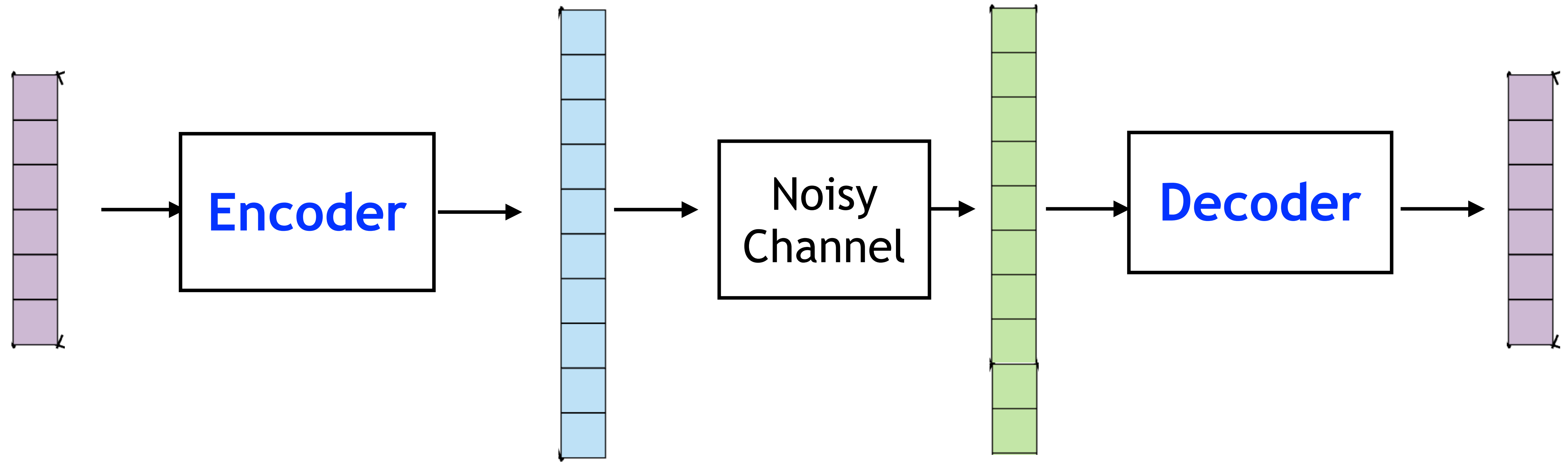
    two phase scheme + ideas from communications

Turbo Autoencoder:
AWGN Channel Codes via Deep Learning

Yihan Jiang, Hyeji Kim, Himanshu Asani, Sreeram Kannan, Sewoong Oh, Pramod Viswanath

Channel coding as an autoencoder

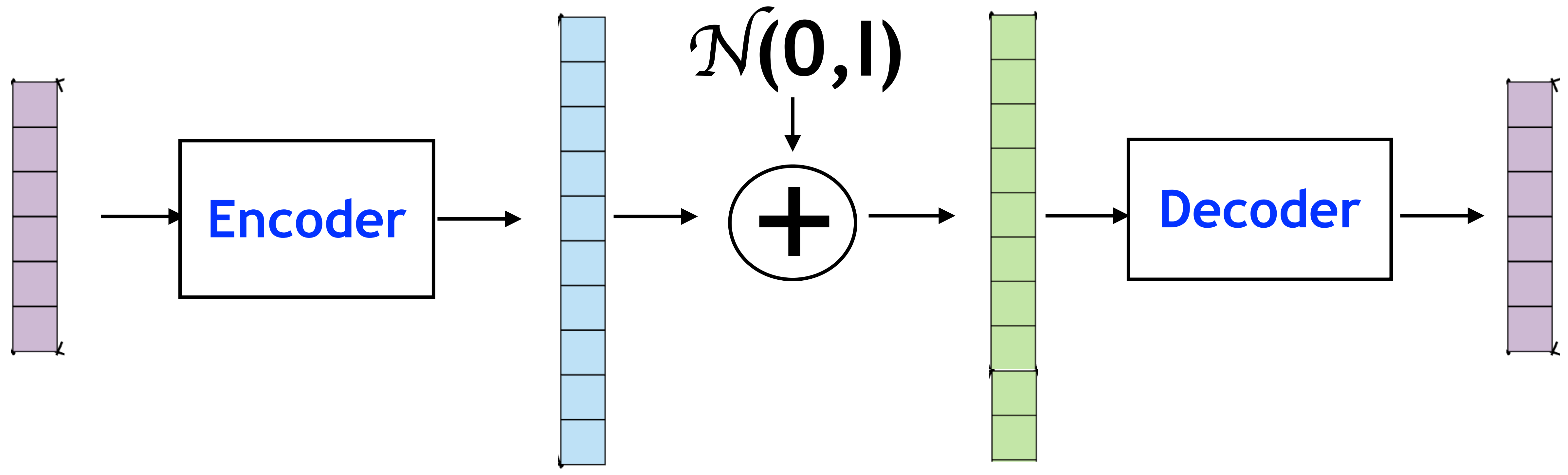
- Channel coding



- Autoencoder that learns to copy its input to its output

Literature

- Deep learning (DL) based code for AWGN channels

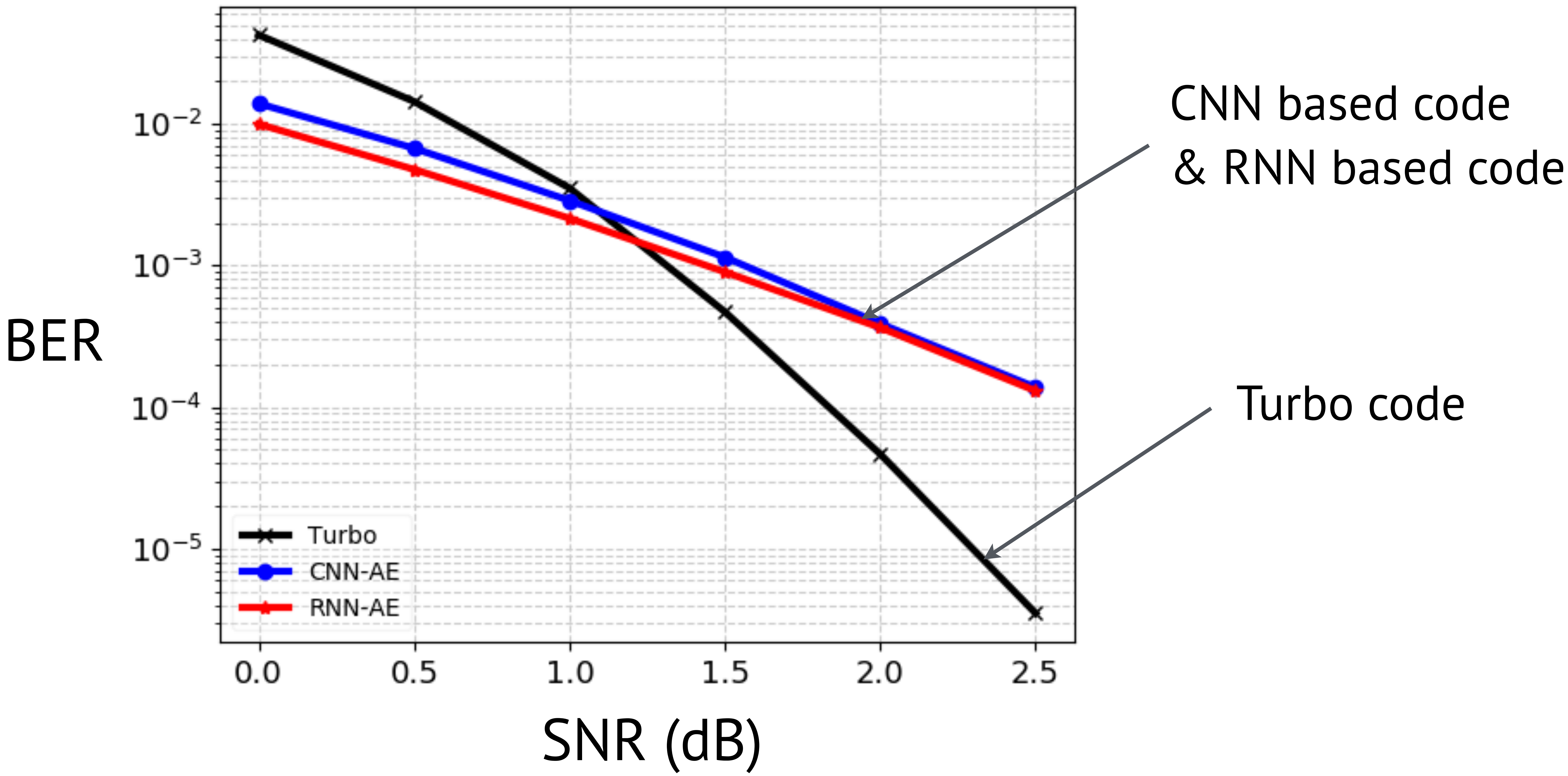


Literature

- Deep learning (DL) based code for AWGN channels
- DL based code achieves the reliability of (7,4) Hamming code (O'Shea, Hoydis '17)
- Blocklength 100 => $2^{100.}$ codewords!

Literature

- Challenge: generalization to longer block lengths



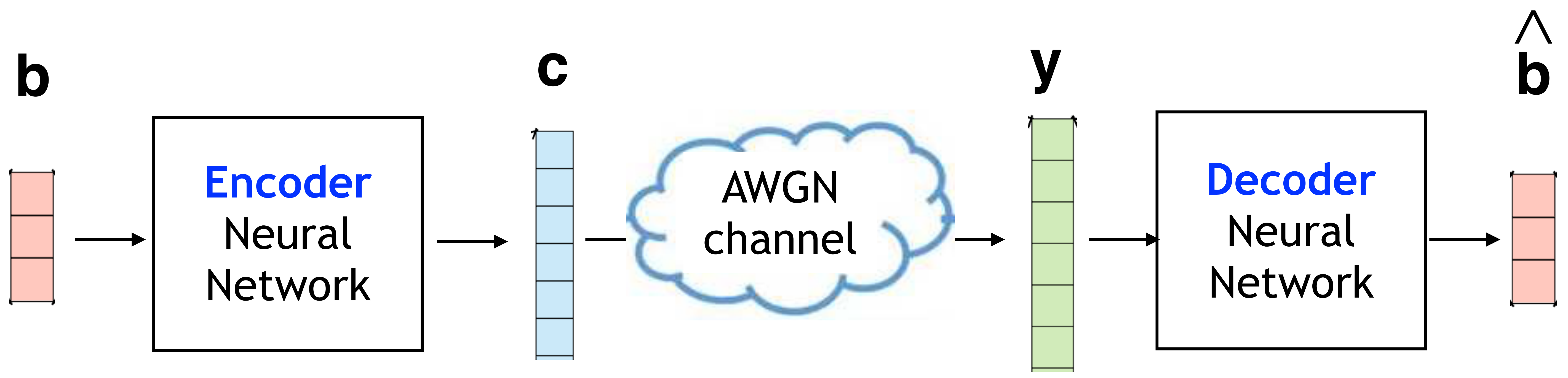
(100 bits, rate 1/3)

Turbo AutoEncoder

Key: Architectural innovations, novel training methodology

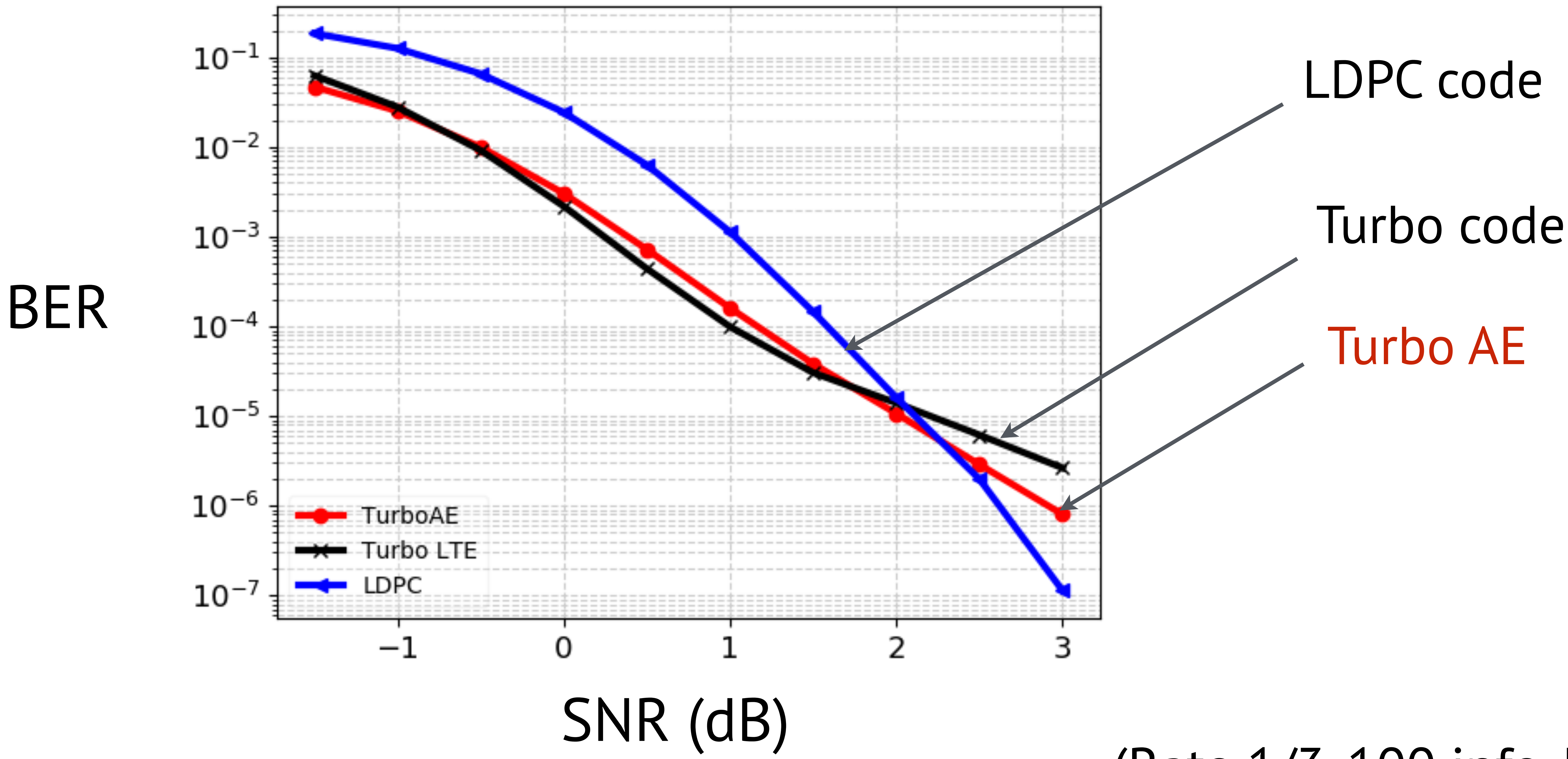
Our approach

- Turbo AutoEncoder (TurboAE)
  - ENC and DEC as neural networks inspired by turbo code



Main results

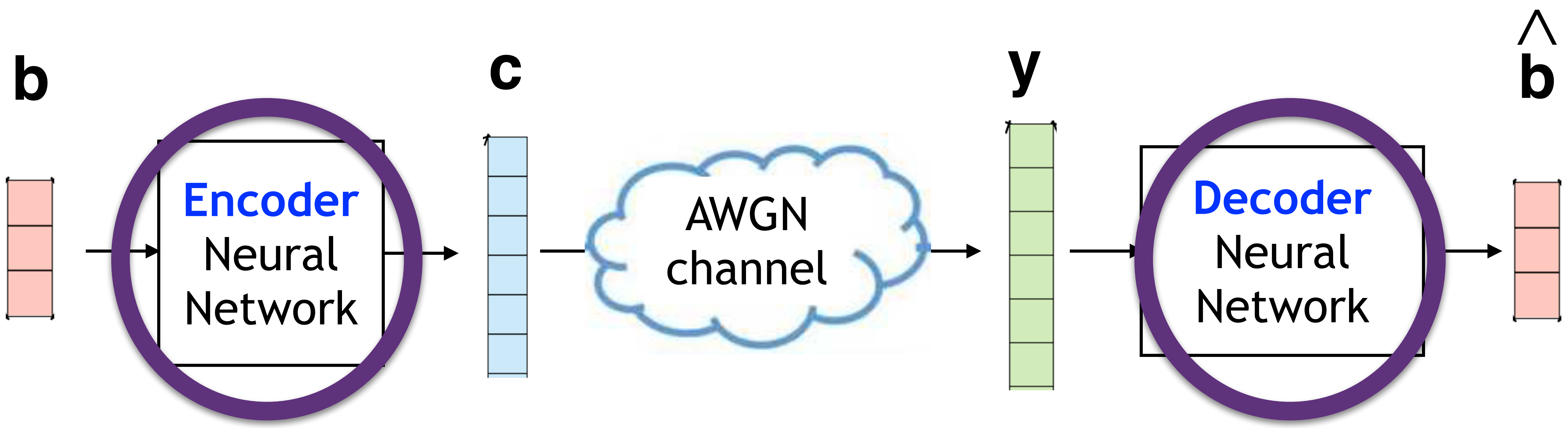
- TurboAE is comparable to turbo codes for block length 100



(Rate 1/3, 100 info. bits)

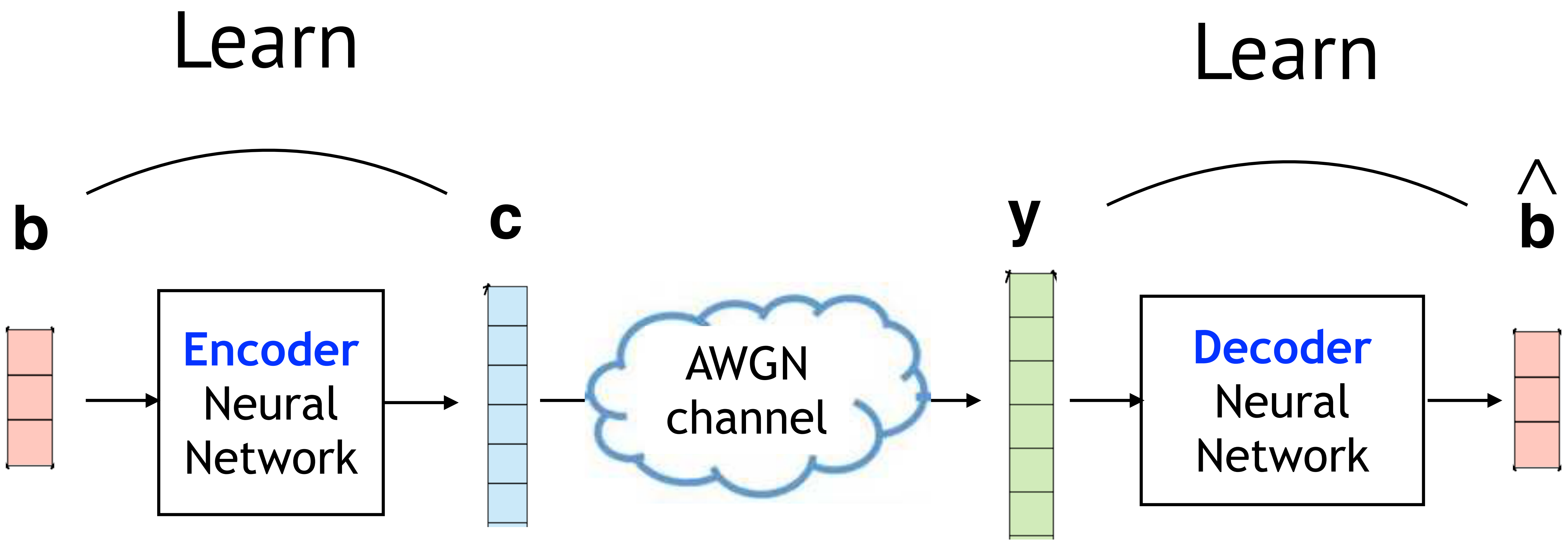
Outline

1. Neural network based encoder and decoder



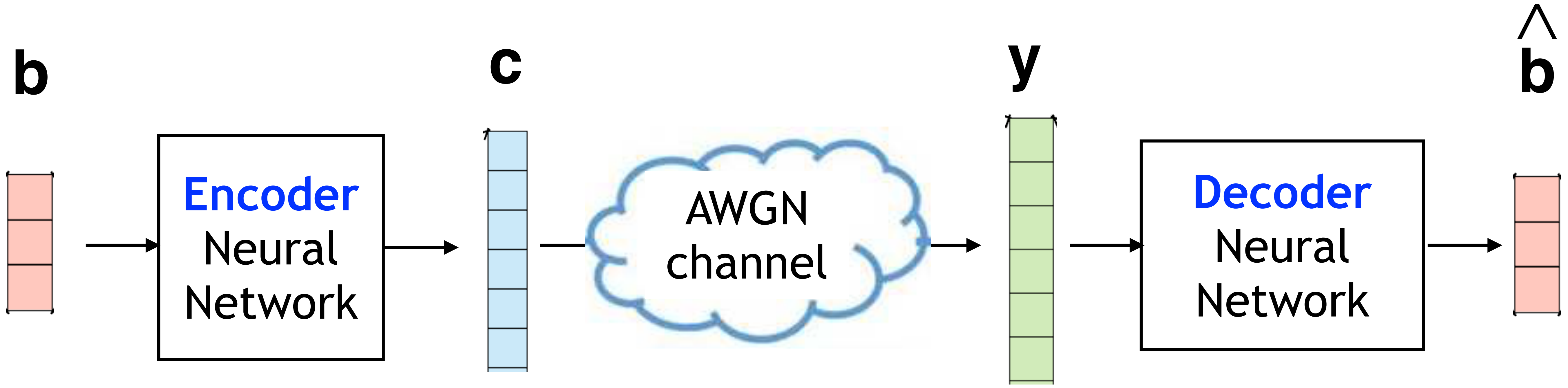
Outline

1. Neural network based encoder and decoder
2. Training



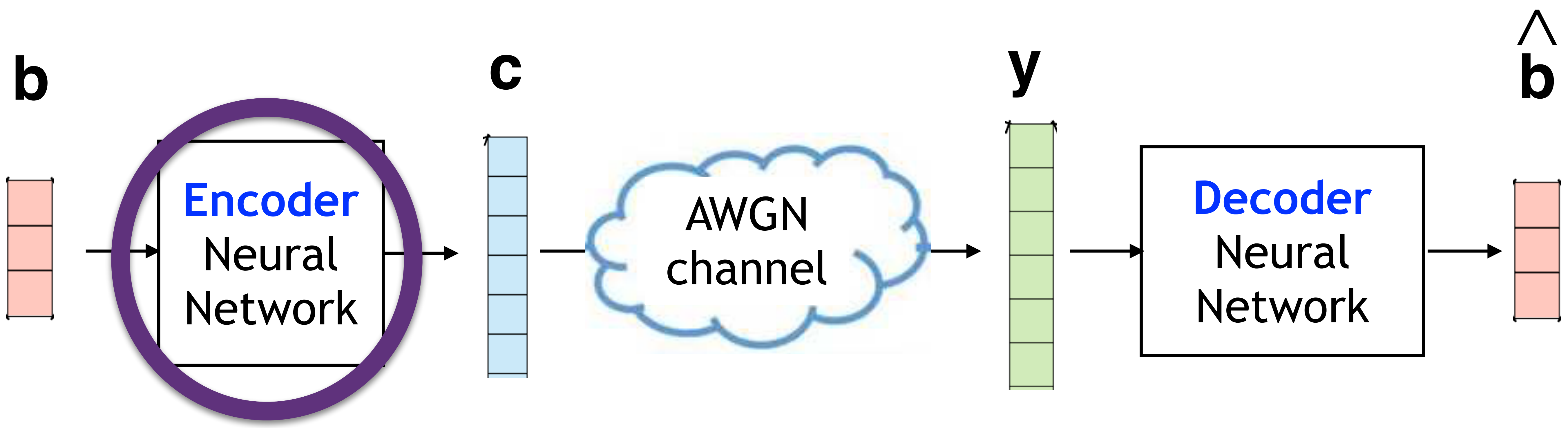
Outline

1. Neural network based encoder and decoder
2. Training
3. Binarization of learned (TurboAE) codewords



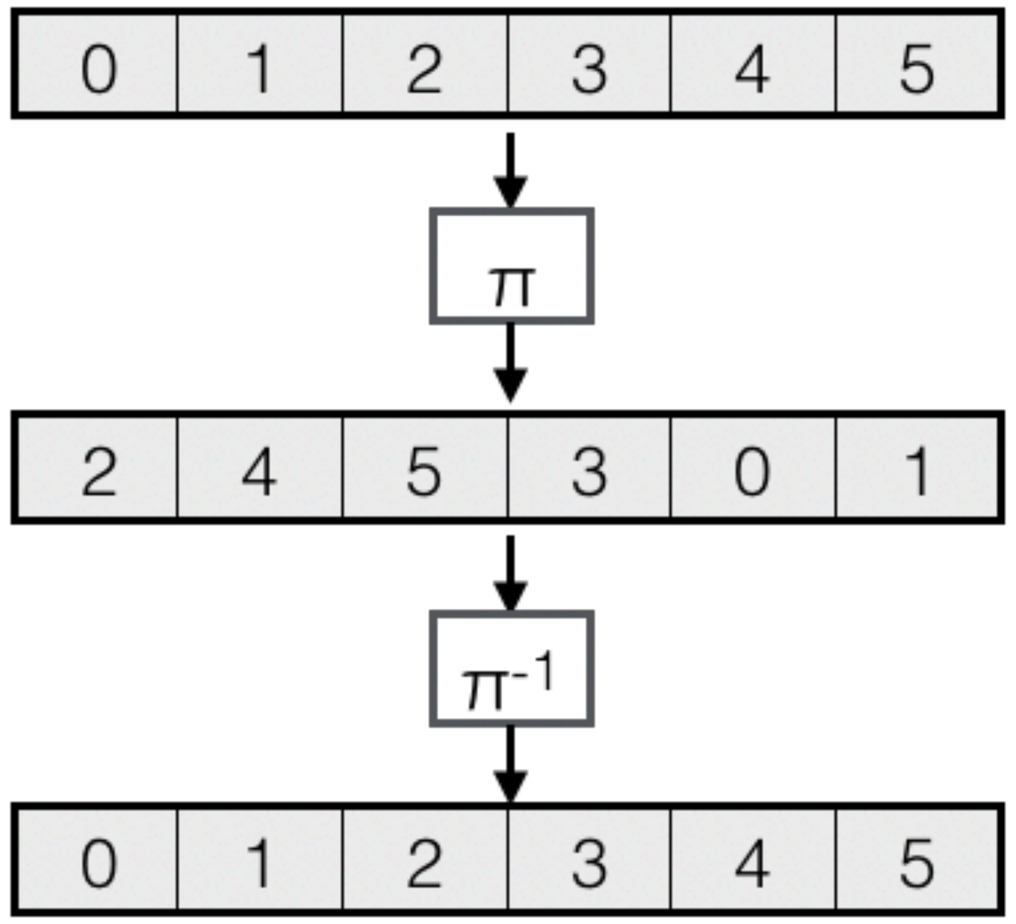
Outline

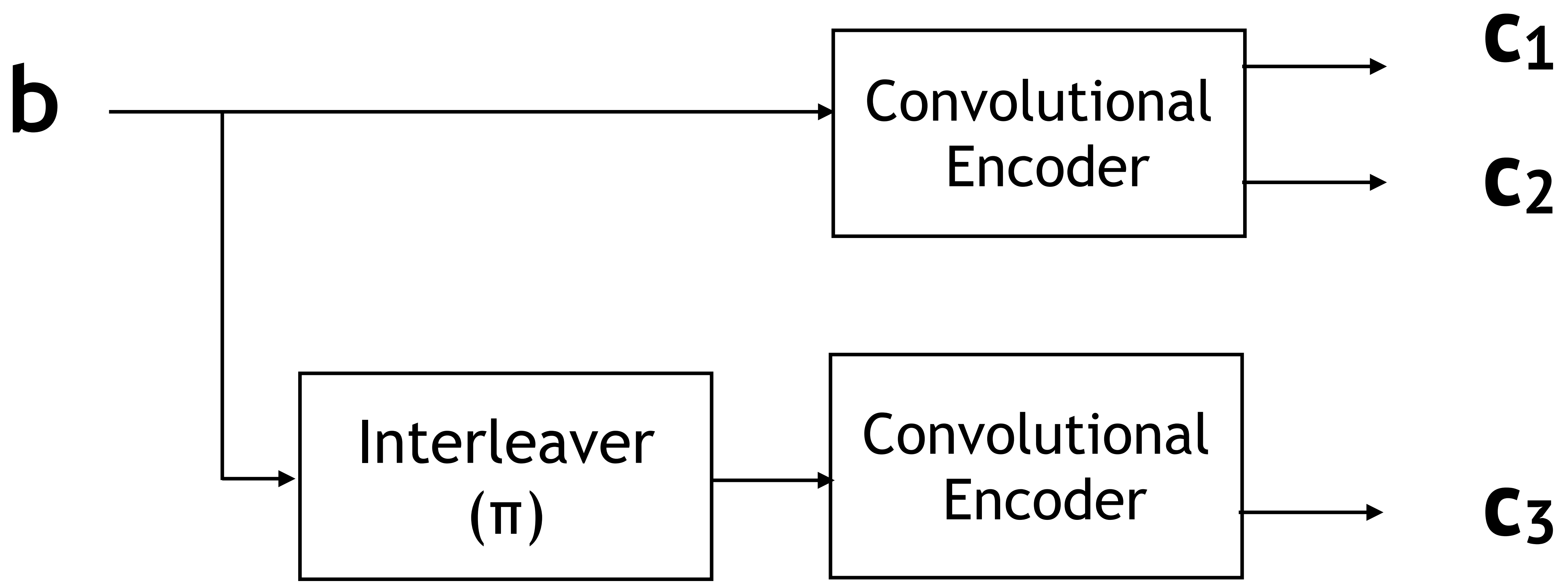
1. Neural network based encoder and decoder



Inspiration from Turbo code

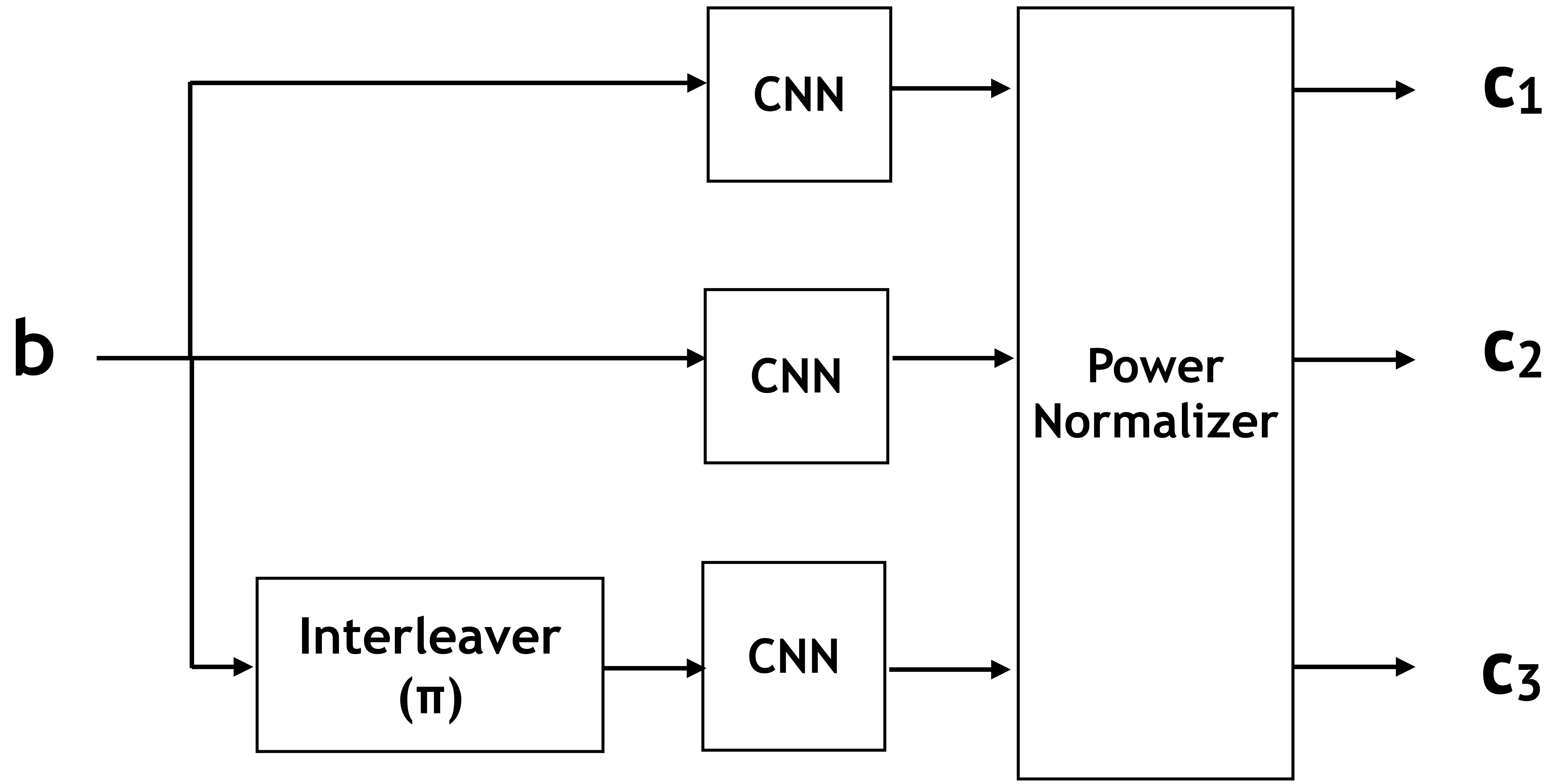
- Concatenate codewords from two convolutional encoders
  - Long term memory via interleaver





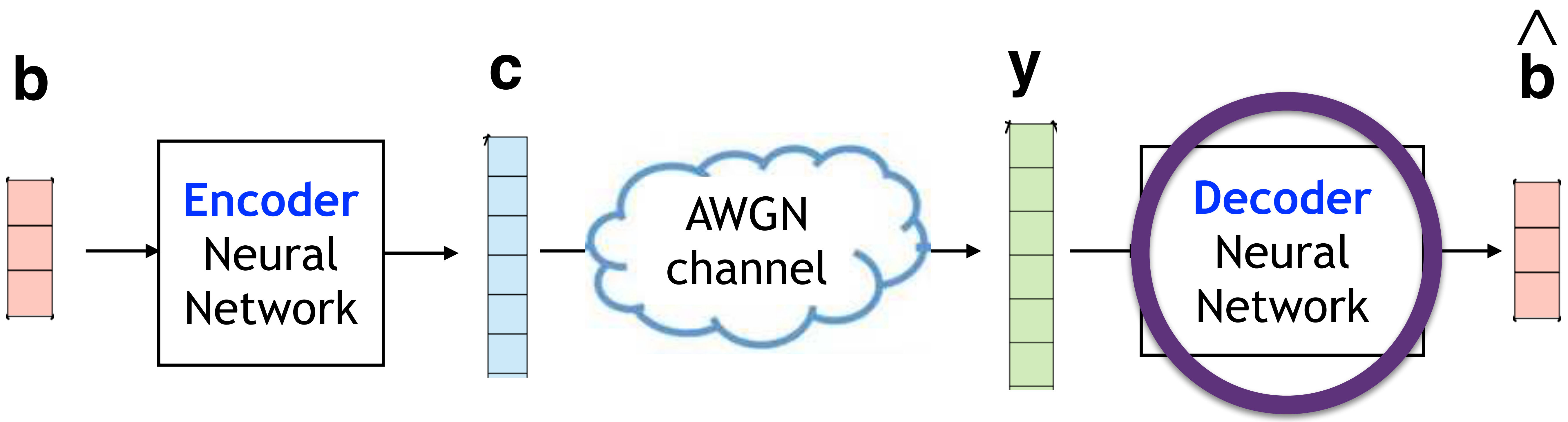
Encoder as a CNN with an interleaver

- 1D convolutional neural network



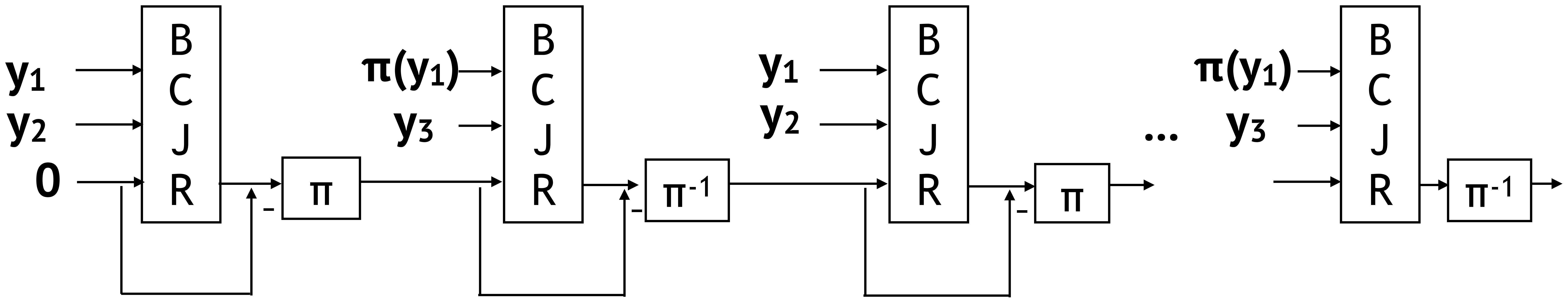
Outline

1. Neural network based encoder and decoder



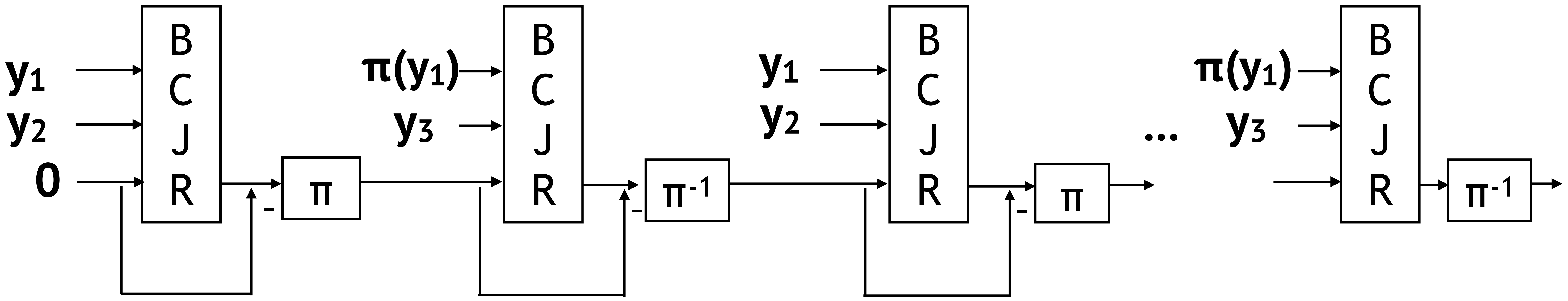
Belief Propagation Decoding

- Iterations of BCJR w. interleaver (π), de-interleaver (π^{-1})
  - BCJR: maps (prior, noisy codewords) to posterior



Belief Propagation Decoding

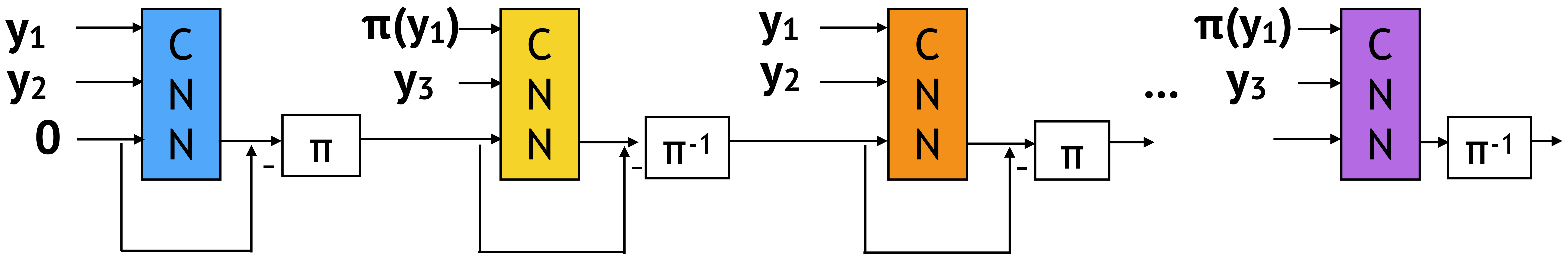
- Iterations of BCJR w. interleaver (π), de-interleaver (π^{-1})
  - BCJR: maps (prior, noisy codewords) to posterior



- Earlier: RNN decoder can be learned solely from data (Jiang-Kim-Asani-Kannan-Oh-Viswanath '19)

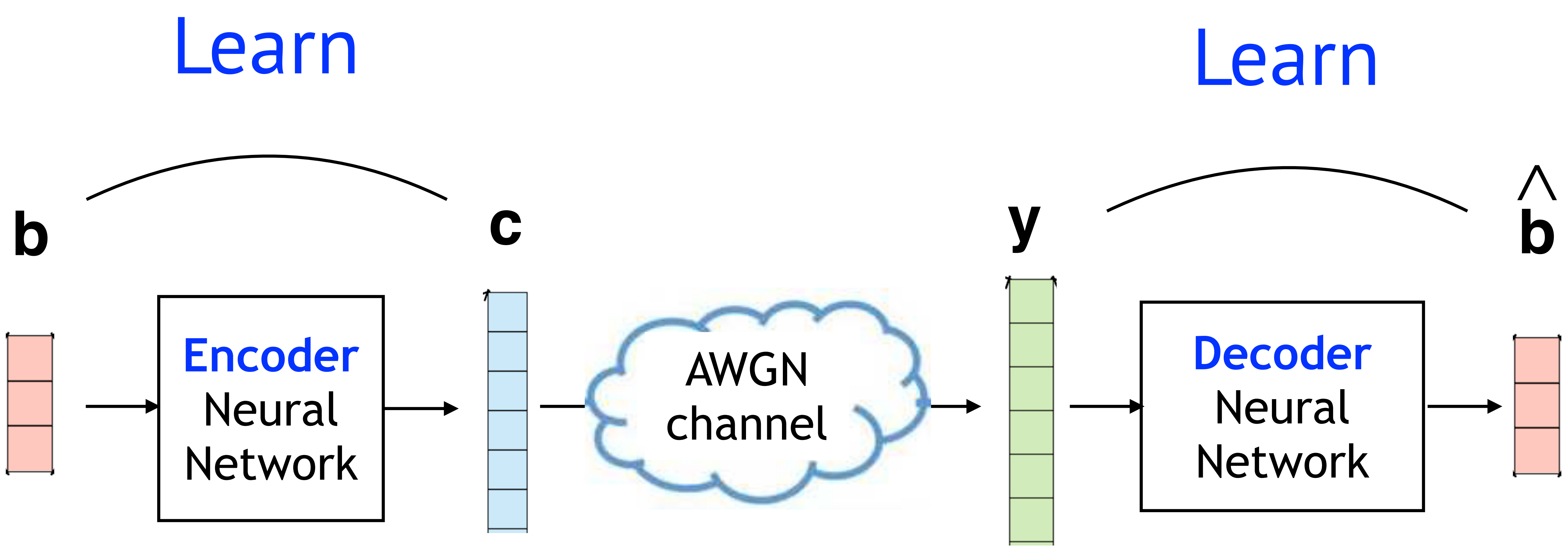
Decoder as a CNN

- Model decoder as a CNN with a vector belief propagation



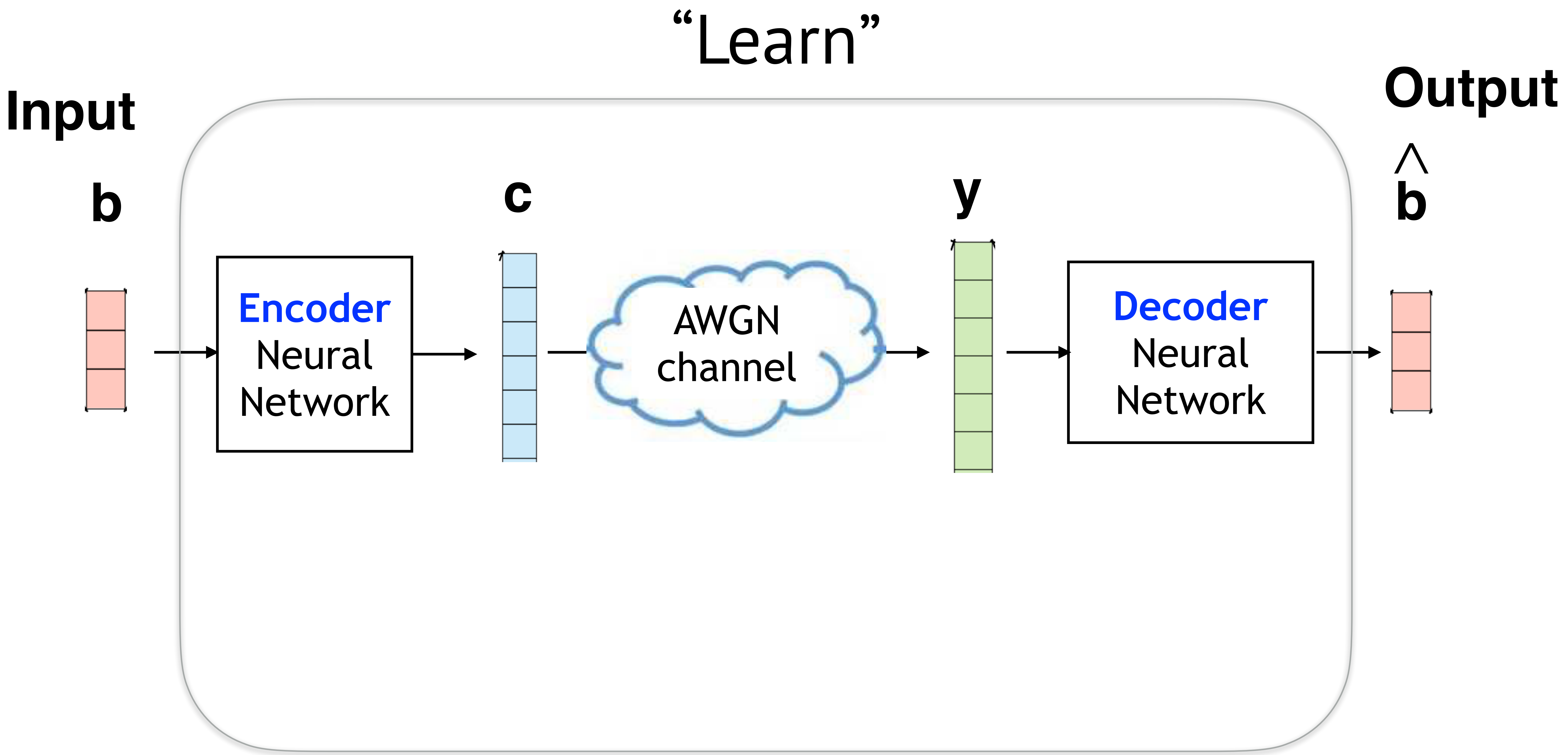
Outline

1. Neural network based encoder and decoder
2. Training



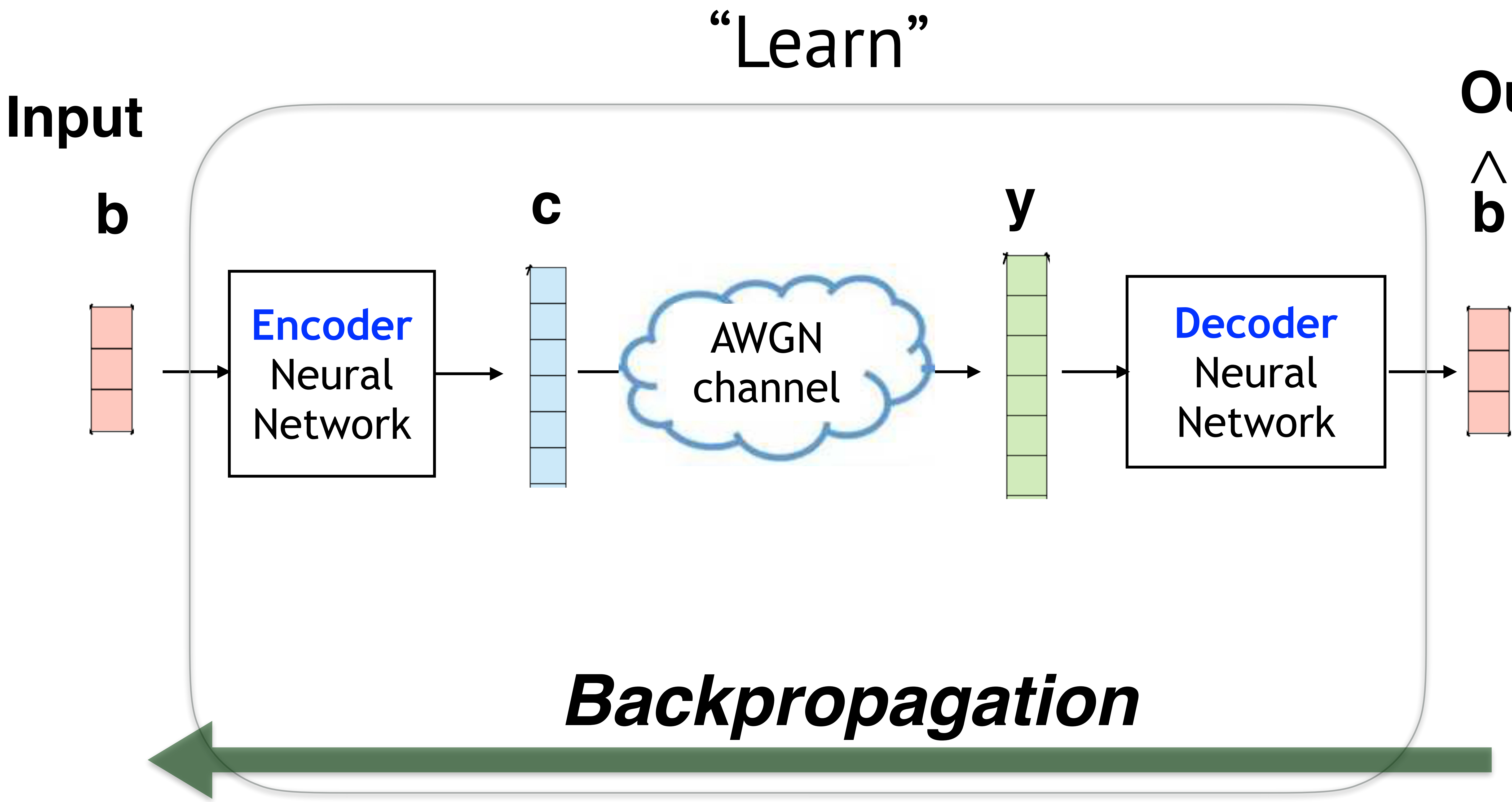
Training

- Generate random bit sequences **b** of length 100
- Simulate the AWGN channels



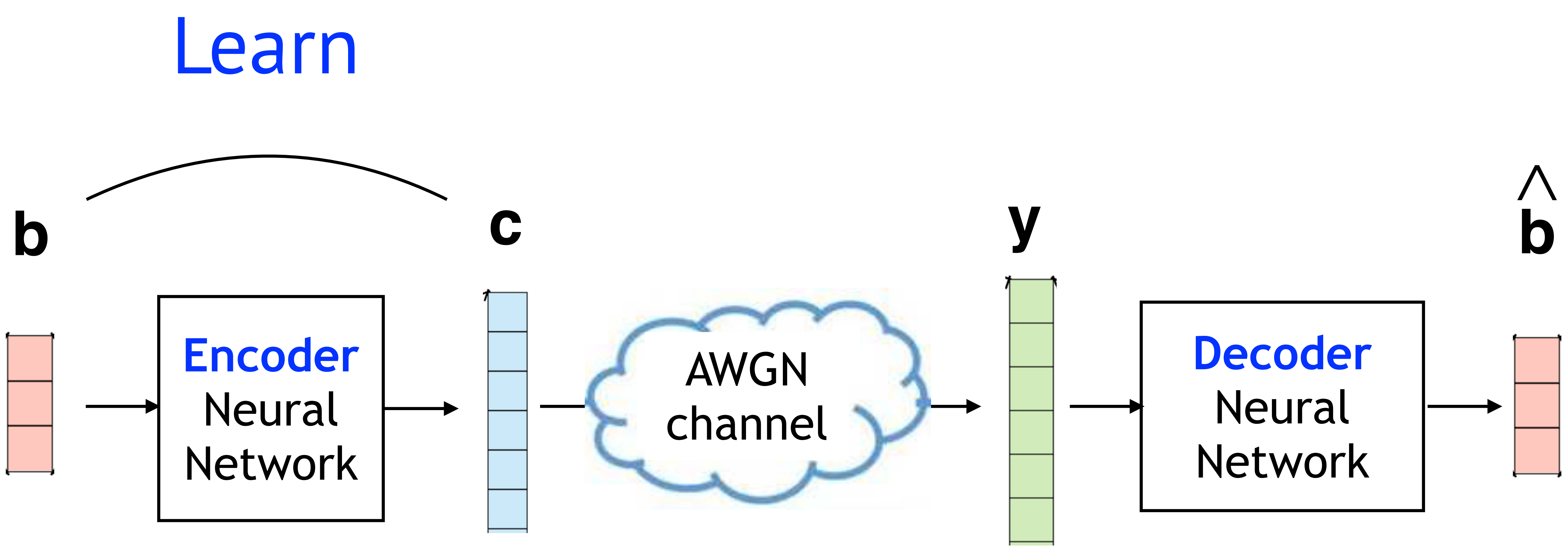
Training

- Auto-encoder training : (input,output) = (**b**,**b**)
- Loss : binary cross entropy $\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = -\mathbf{b} \log \hat{\mathbf{b}} - (1 - \mathbf{b}) \log(1 - \hat{\mathbf{b}})$



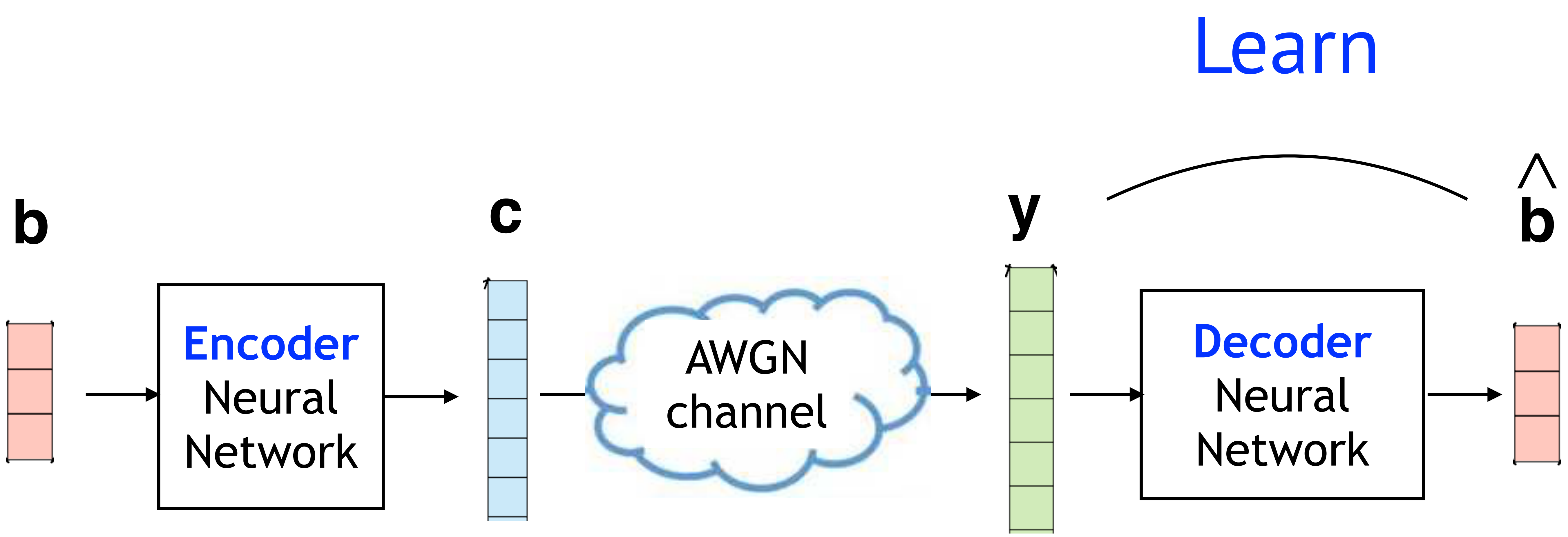
Training

- Alternate training of encoder and training of decoder



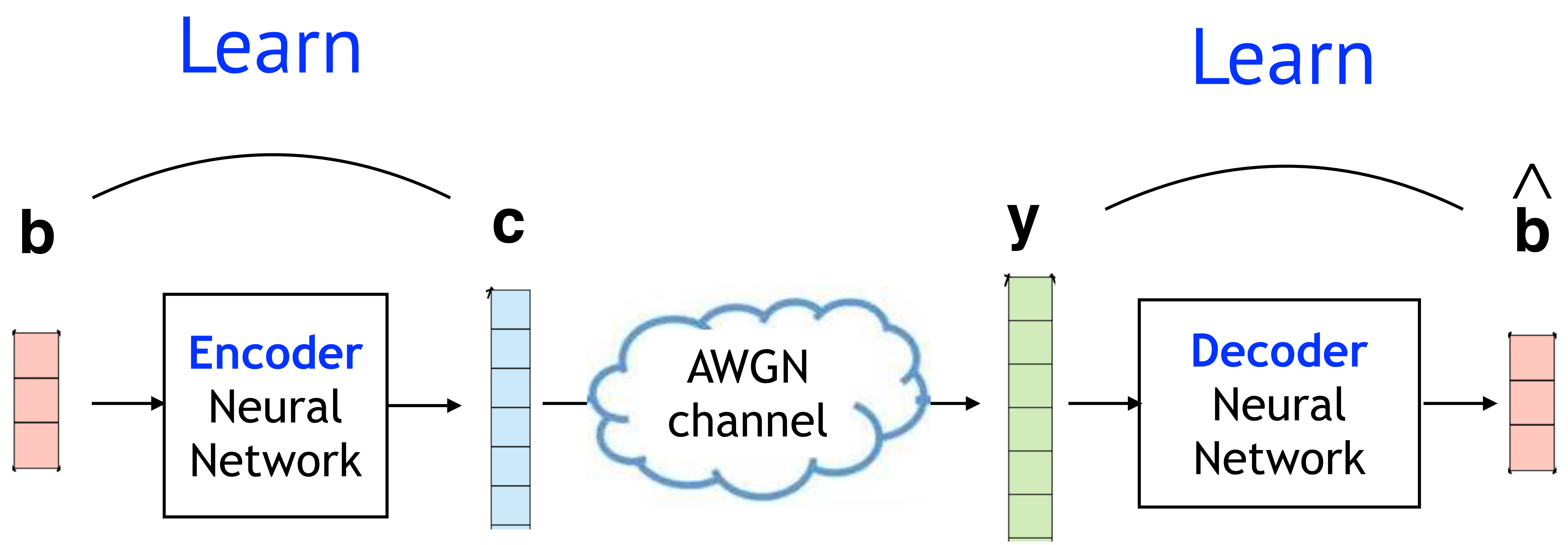
Training

- Alternate training of encoder and training of decoder



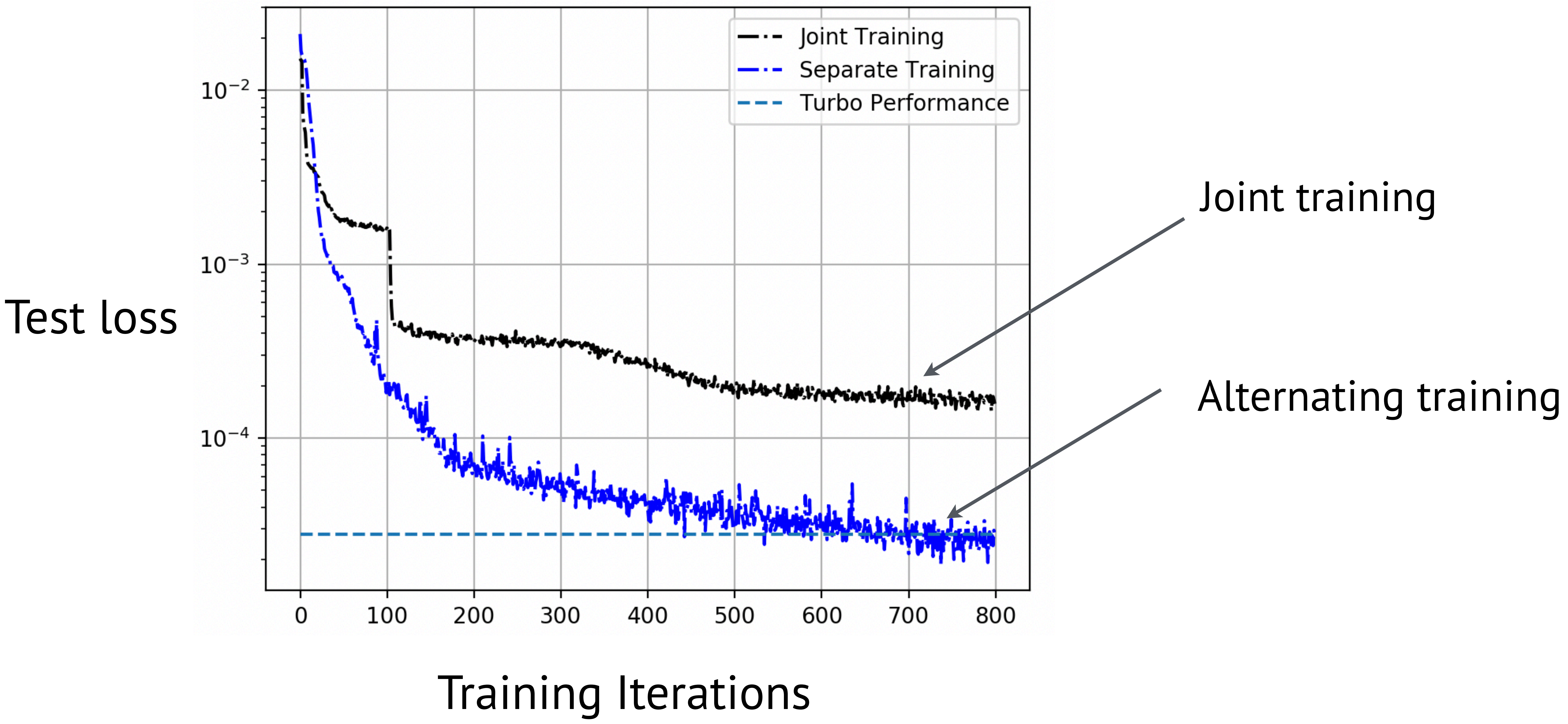
Training

- Alternate training of encoder and training of decoder
  - Encoder 100 times & decoder 500 times
  - Principle: (For each code, learn near-optimal decoder)



Training choice 1. Alternating training

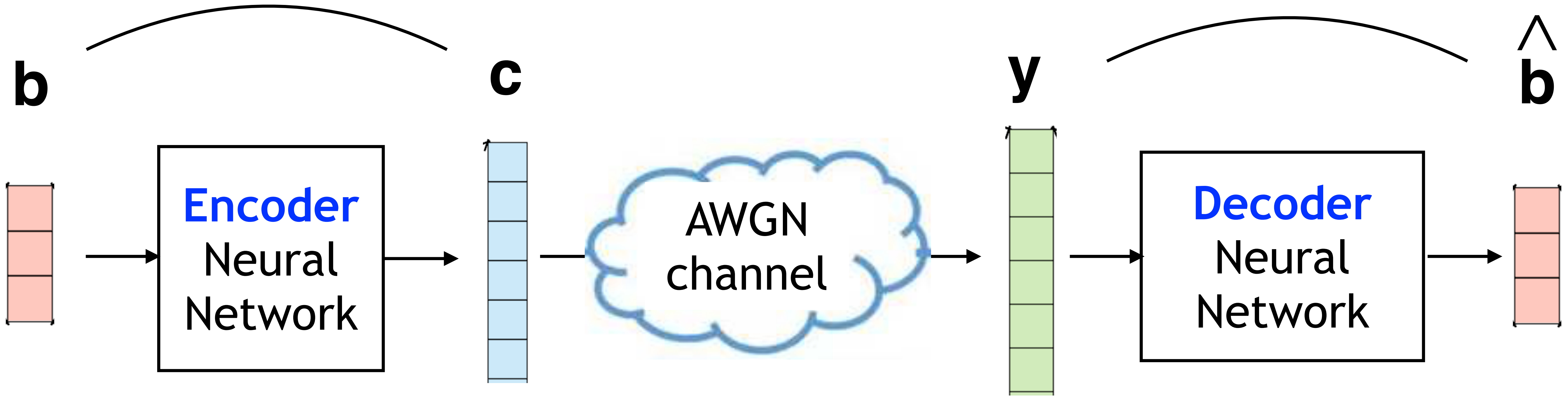
- Joint training of encoder and decoder results in a local optima



Training choice 2. SNR

- SNR of AWGN channels

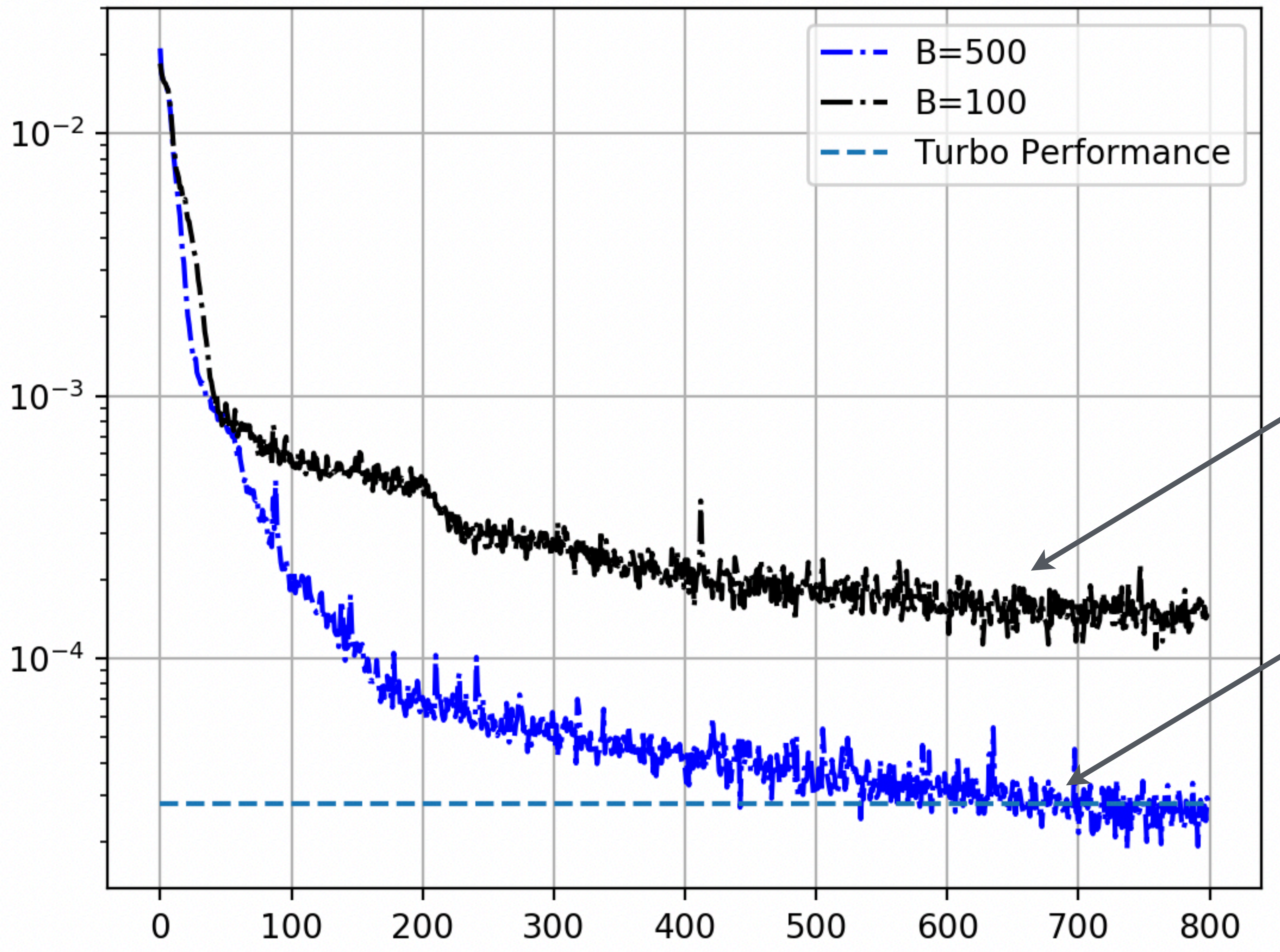
Learn at test SNR *Learn at Mixture SNRs (-1.5 to 2dB)*



Training choice 3. Batch size

- Batch size is critical
  - Large batch size is necessary (>500!)

Test loss



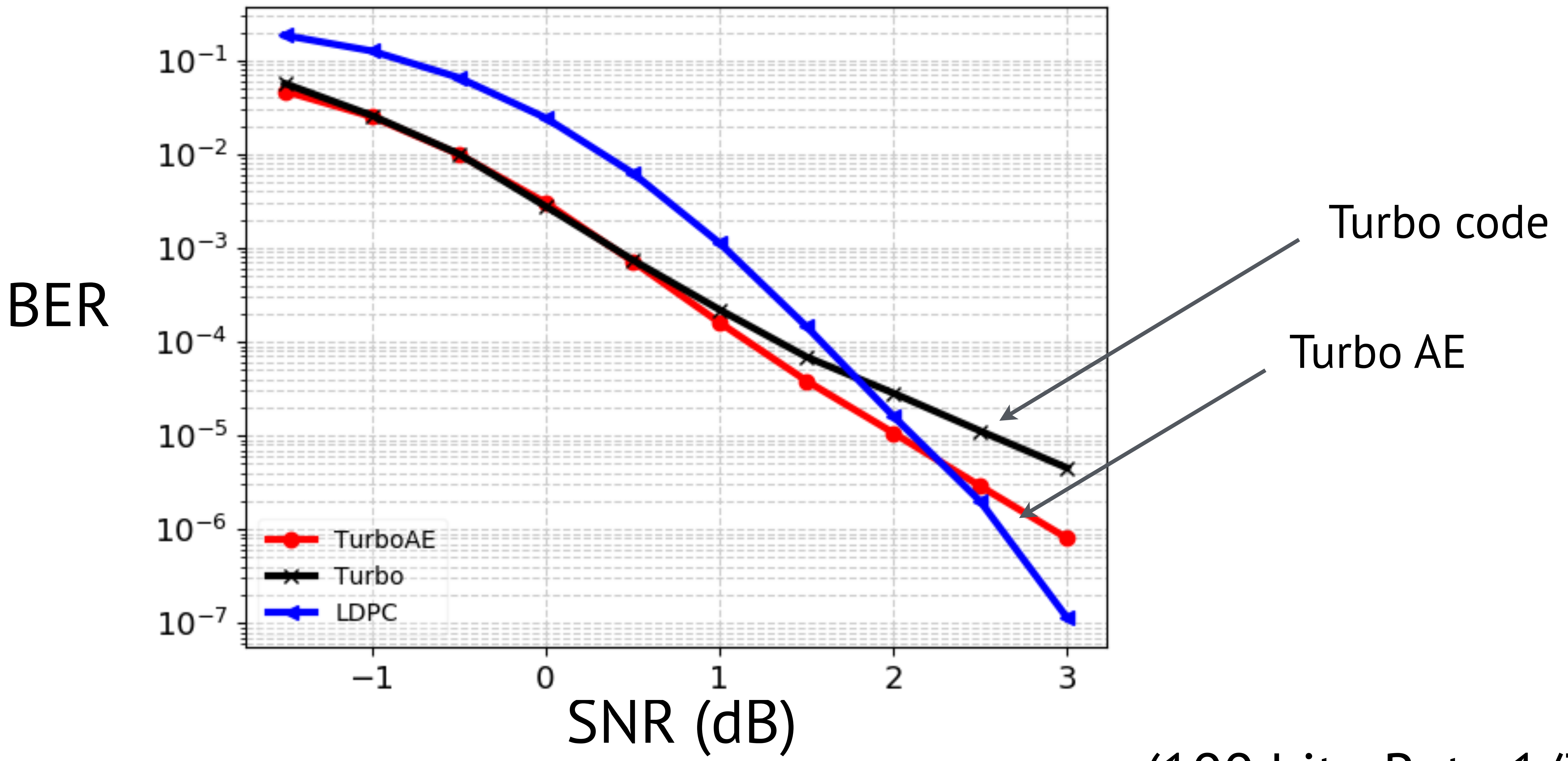
Training iterations

Batch size 100

Batch size 500

Main result

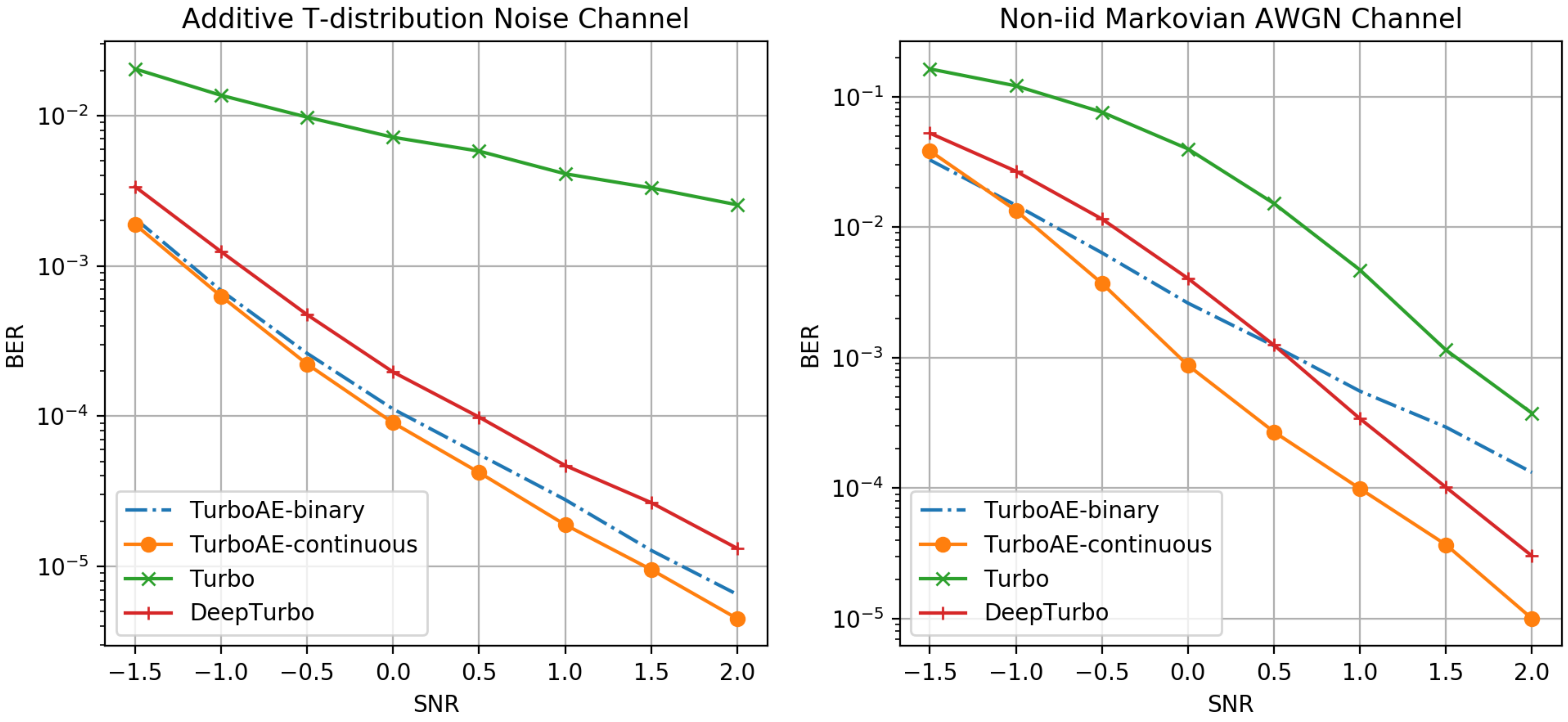
- Achieve the reliability of turbo codes for block length 100



(100 bits, Rate 1/3)

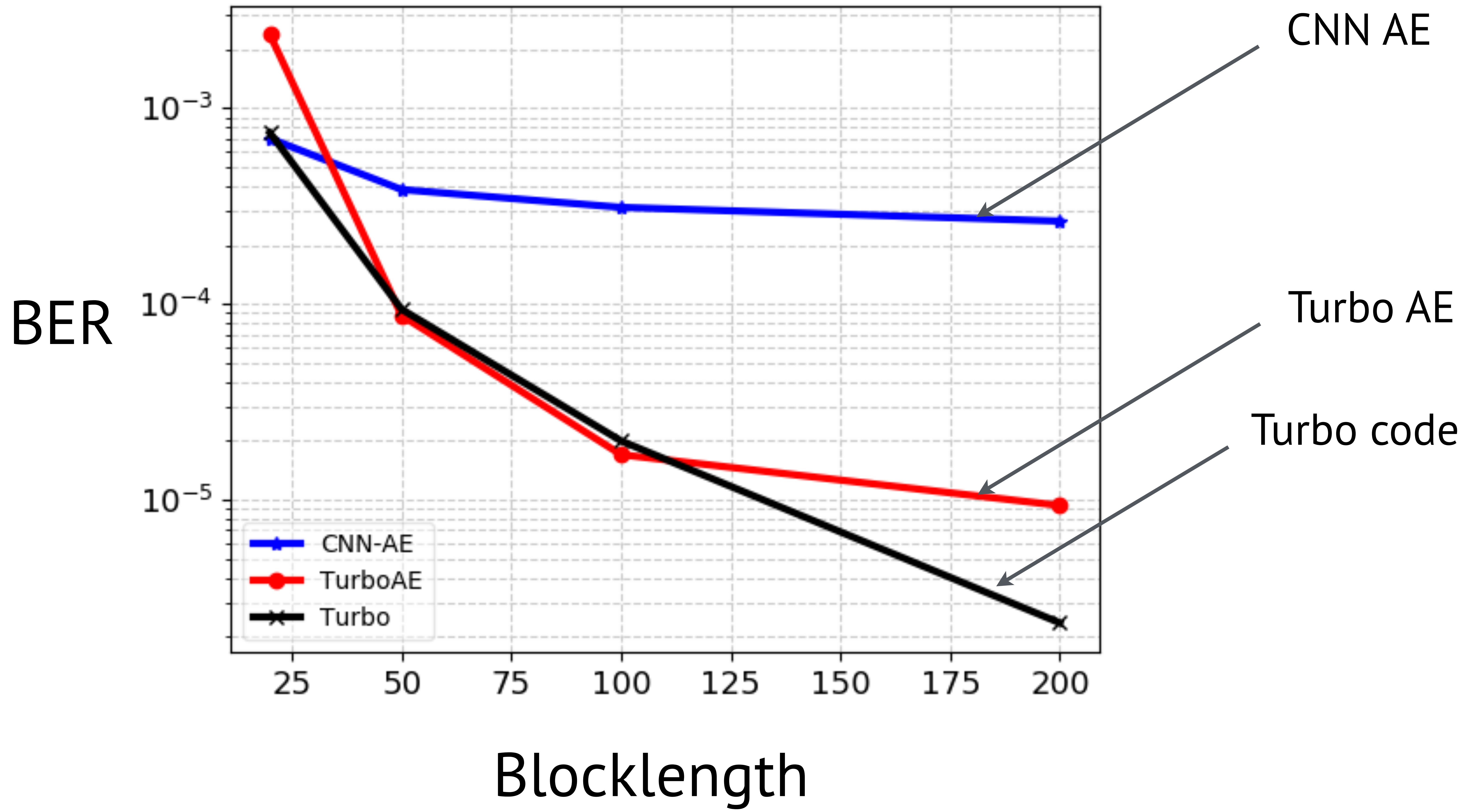
TurboAE: Results

- non-AWGN: TurboAE harvests encoder flexibility:
  - iid non-Gaussian Channel (ATN)
  - non-iid Markovian AWGN channel



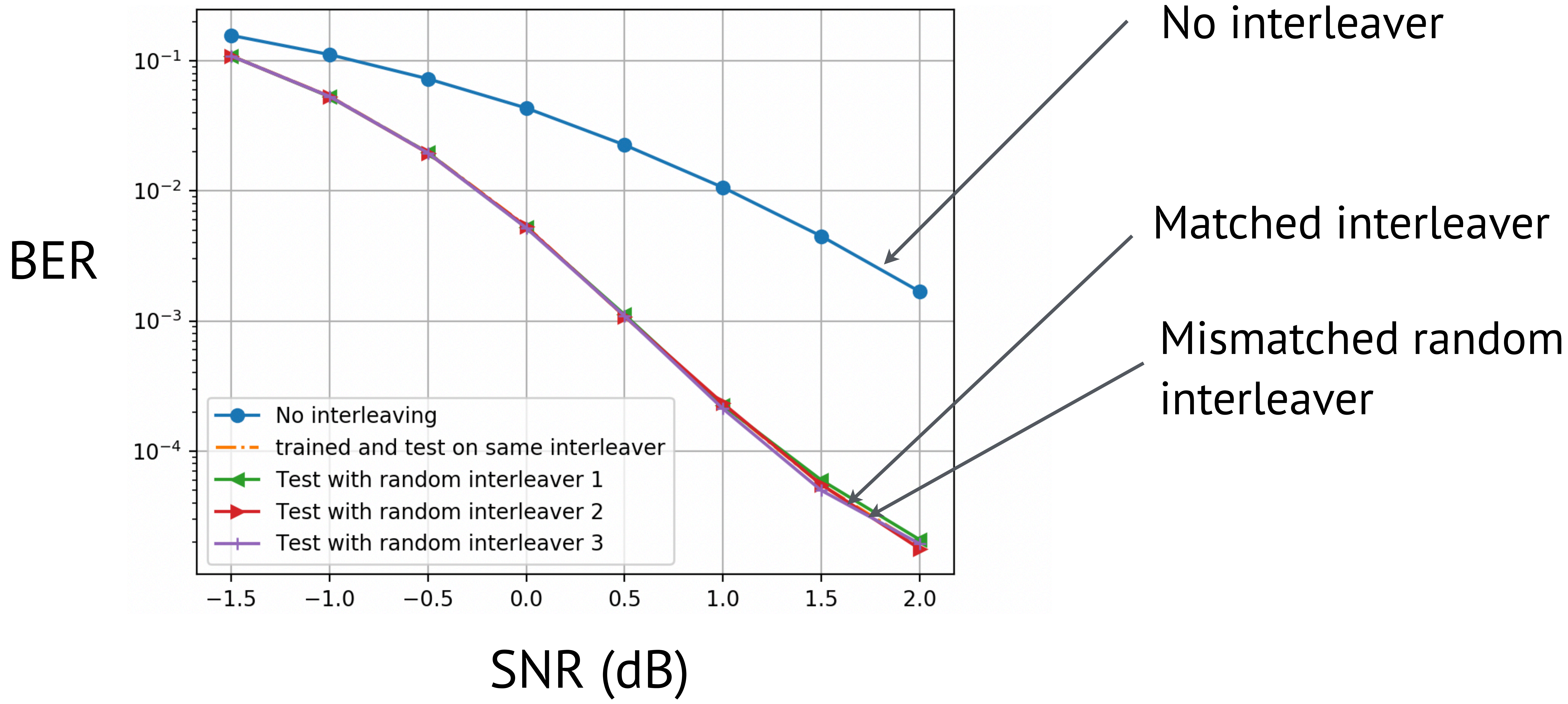
Block length gain

- TurboAE has a block length gain



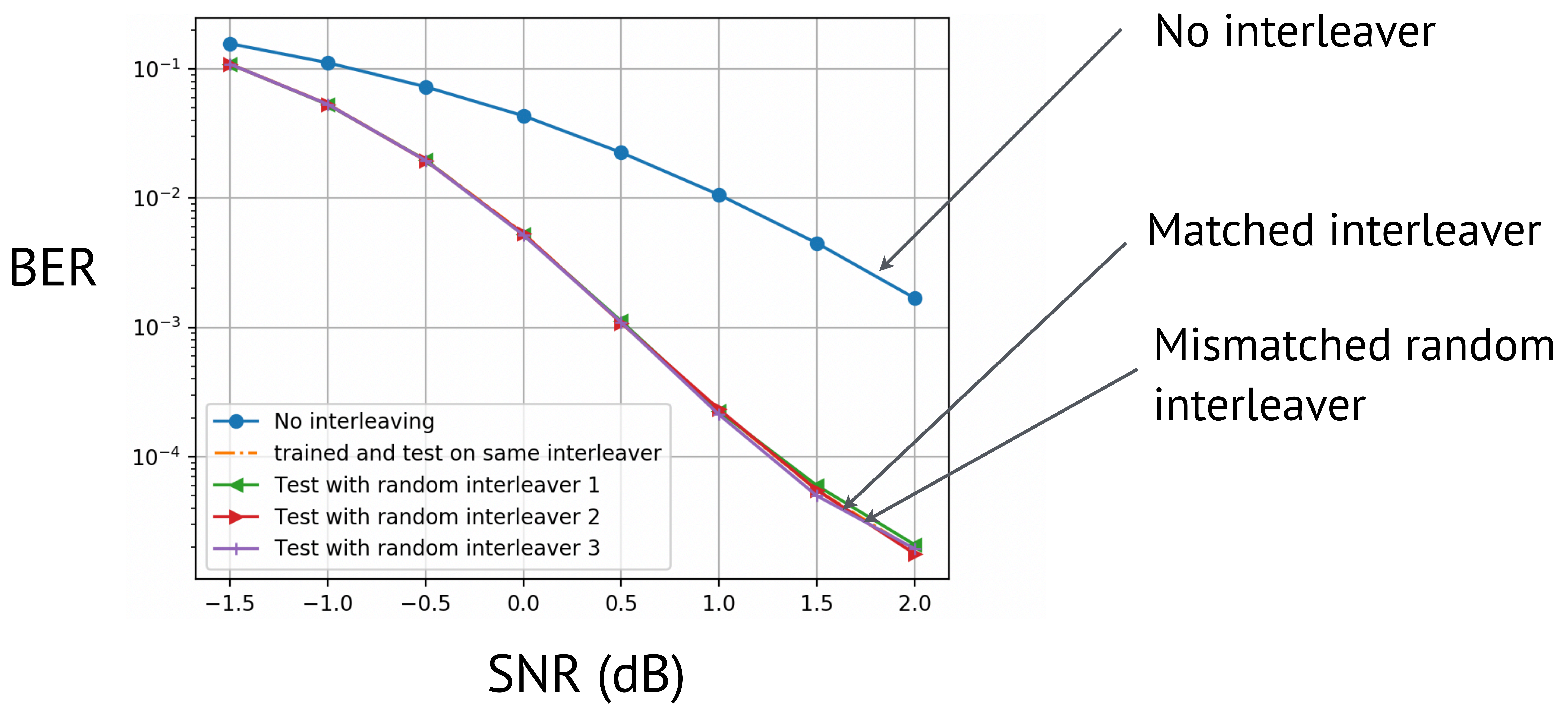
Generalization across interleavers

- Fix an interleaver during training, and test with various interleavers



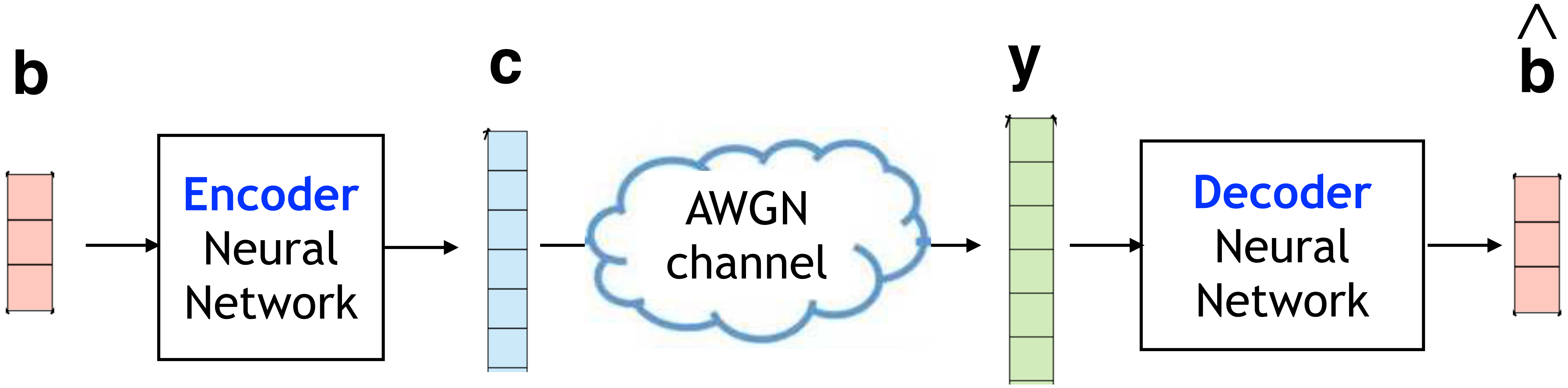
Generalization across interleavers

- Fix an interleaver during training, and test with various interleavers
  - No overfitting to the interleaver used in the training



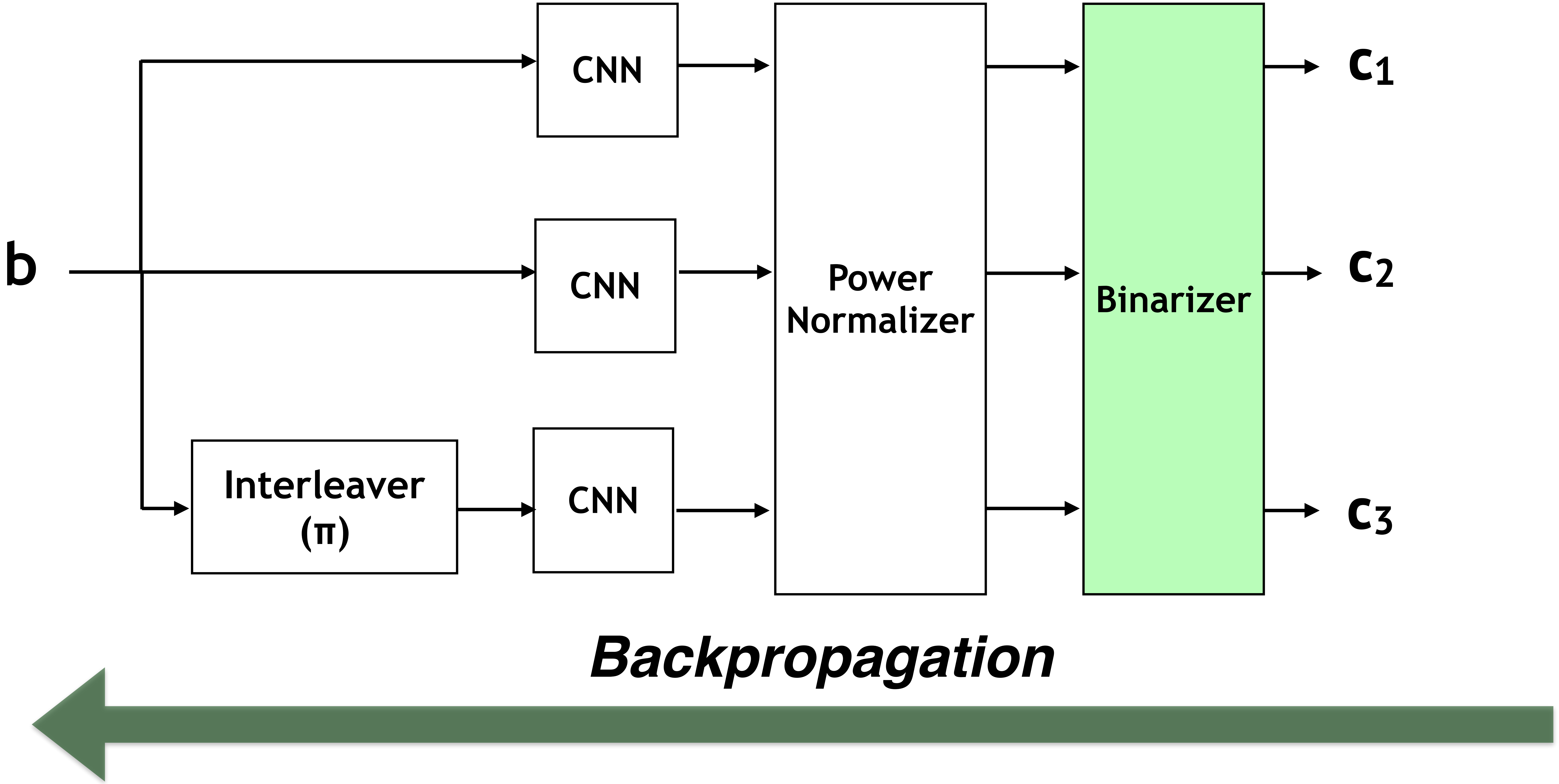
Outline

1. Neural network based encoder and decoder
2. Training
3. Binarization of TurboAE code

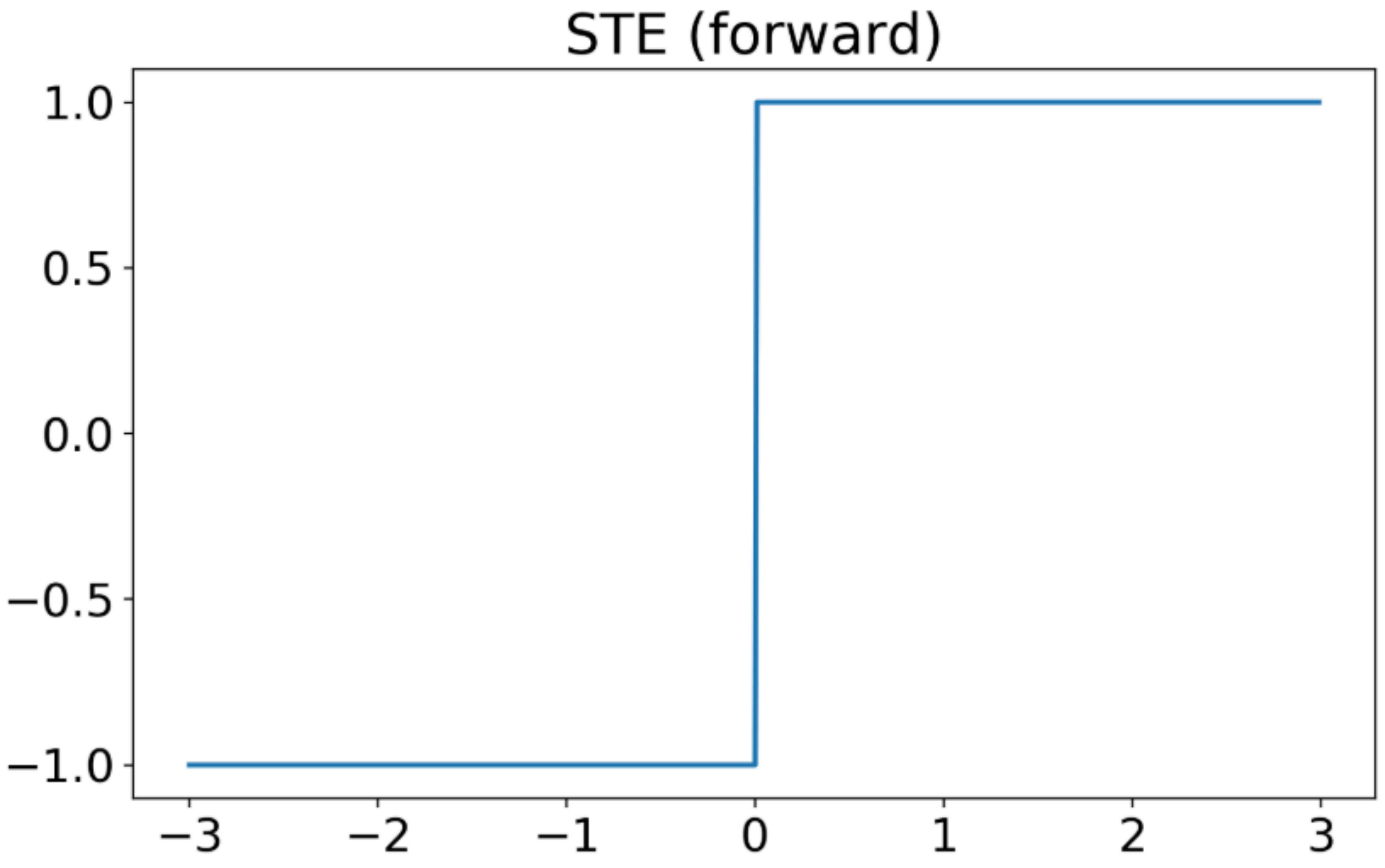


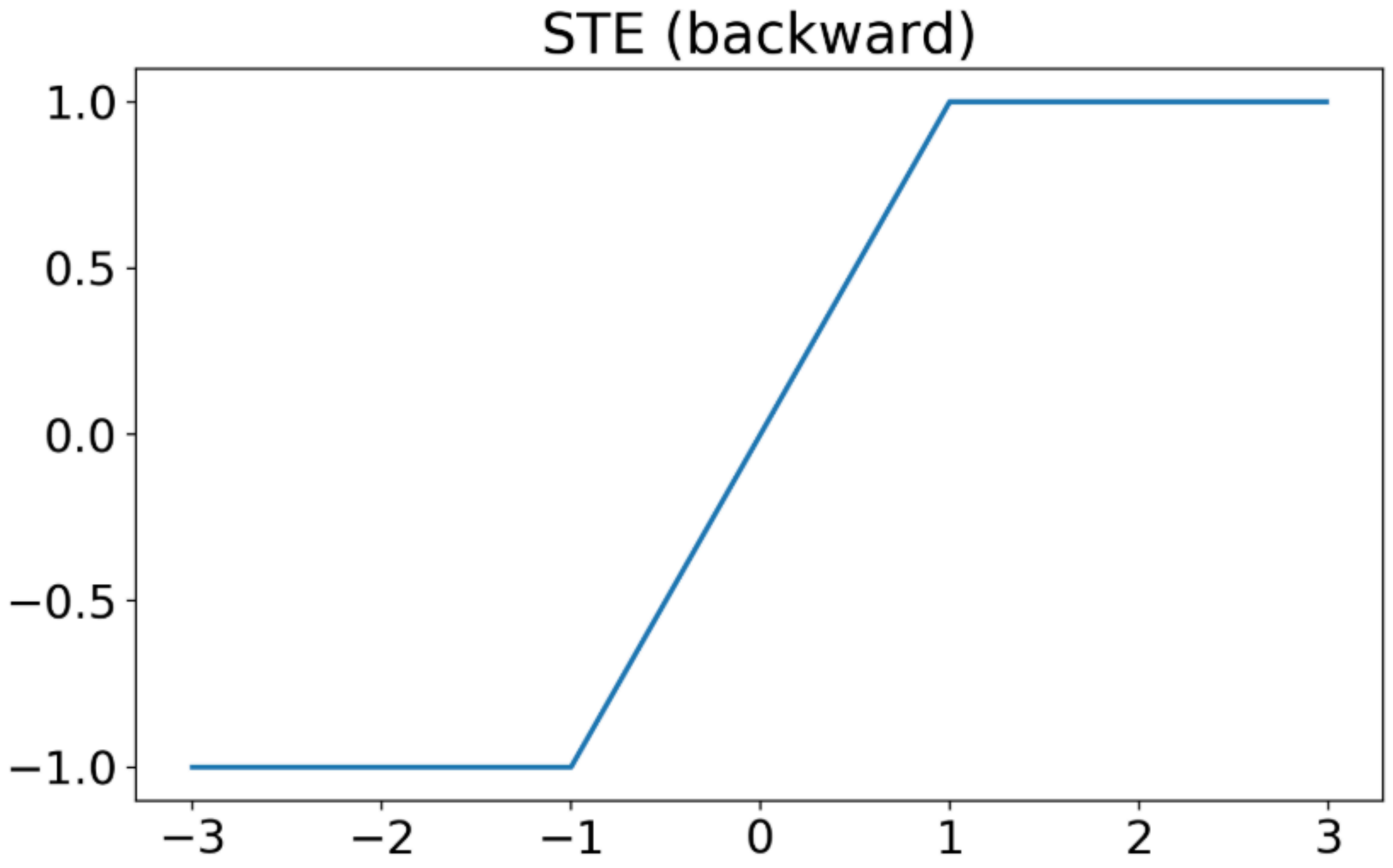
Binarization of TurboAE

- Binarizer: Output = sign(Input)



Training through Straight-Through-Estimator





$$x = sign(\hat{b})$$

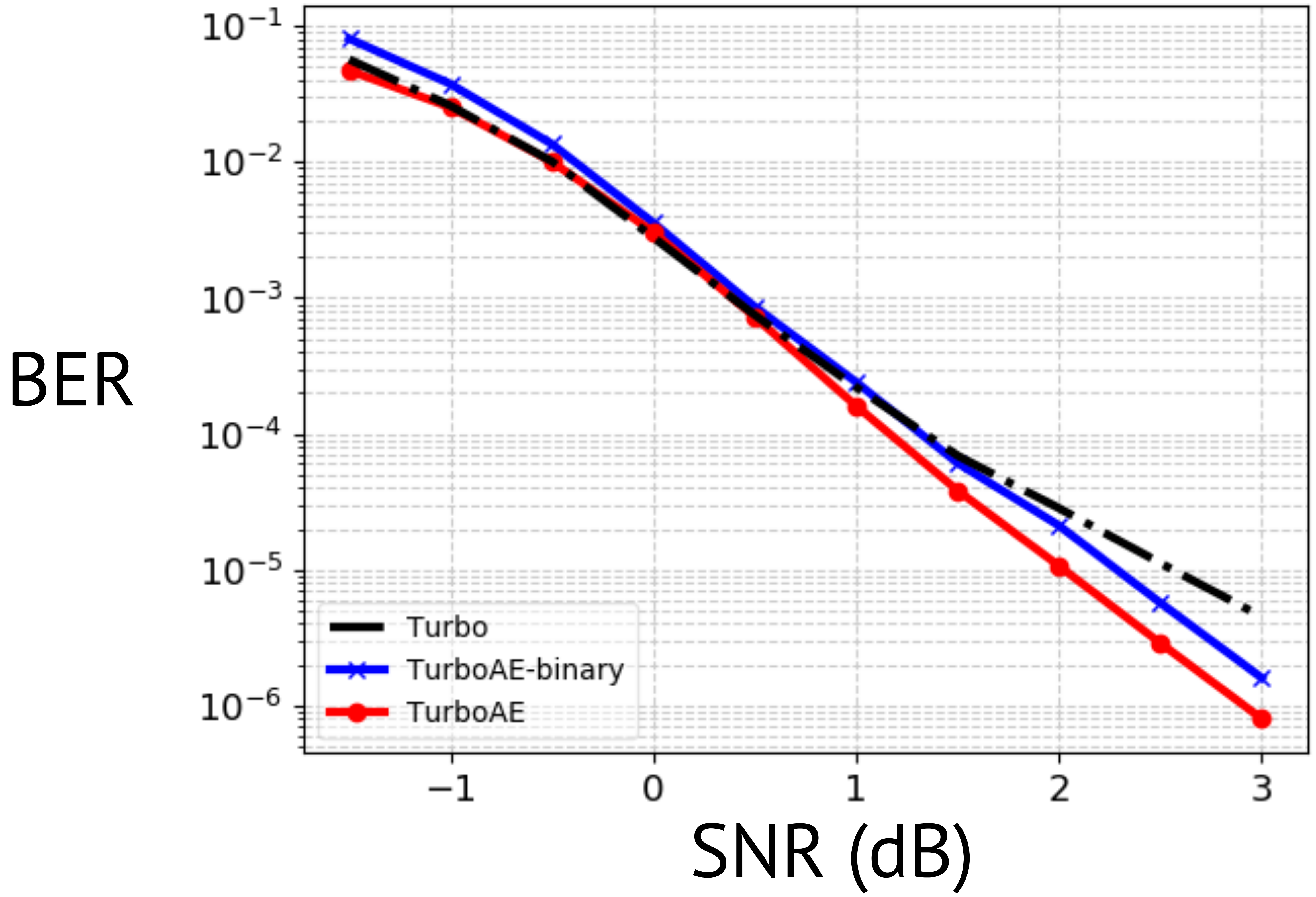
$$\frac{\partial x}{\partial \hat{b}} = 1_{|\hat{b}| \leq 1}$$

No Gradient

Non-trivial Gradient

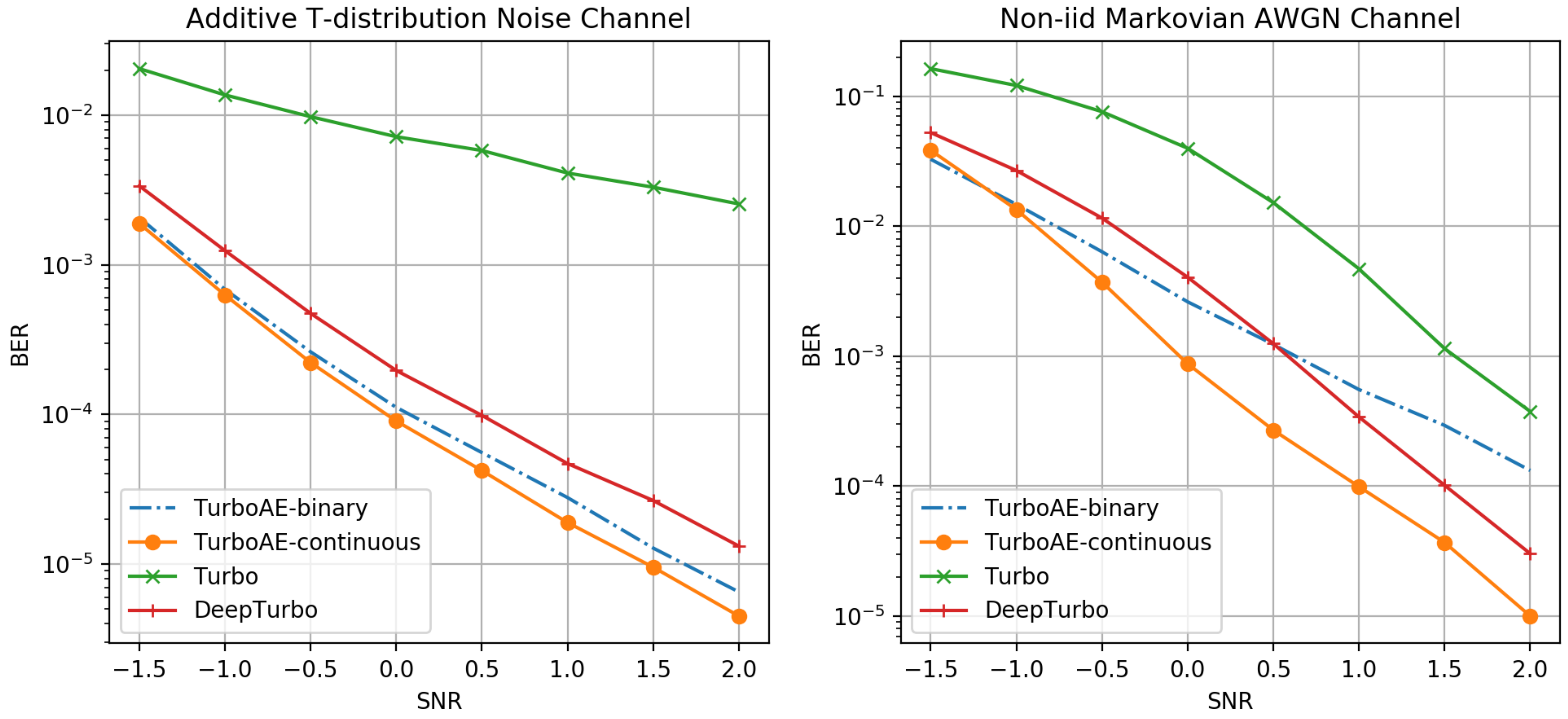
Effect of binarization

- Reliability remains almost the same after binarization



TurboAE: Results

- non-AWGN: TurboAE harvests encoder flexibility:
  - iid non-Gaussian Channel (ATN)
  - non-iid Markovian AWGN channel



Summary

- TurboAE
  - Reliability comparable to modern codes at block length 100
  - Key: Architectural innovation (long term memory by interleaving) & Training methodology

Summary

- TurboAE
  - Reliability comparable to modern codes at block length 100
  - Key: Architectural innovation (long term memory by interleaving) & Training methodology
  - Improves reliability for non-AWGN channels

Open problem

- Longer block length, High SNR

Open problem

- Longer block length, High SNR
- Interpretation

Open problem

- Longer block length, High SNR
- Interpretation
- Extension of TurboAE architecture to other applications

Remark

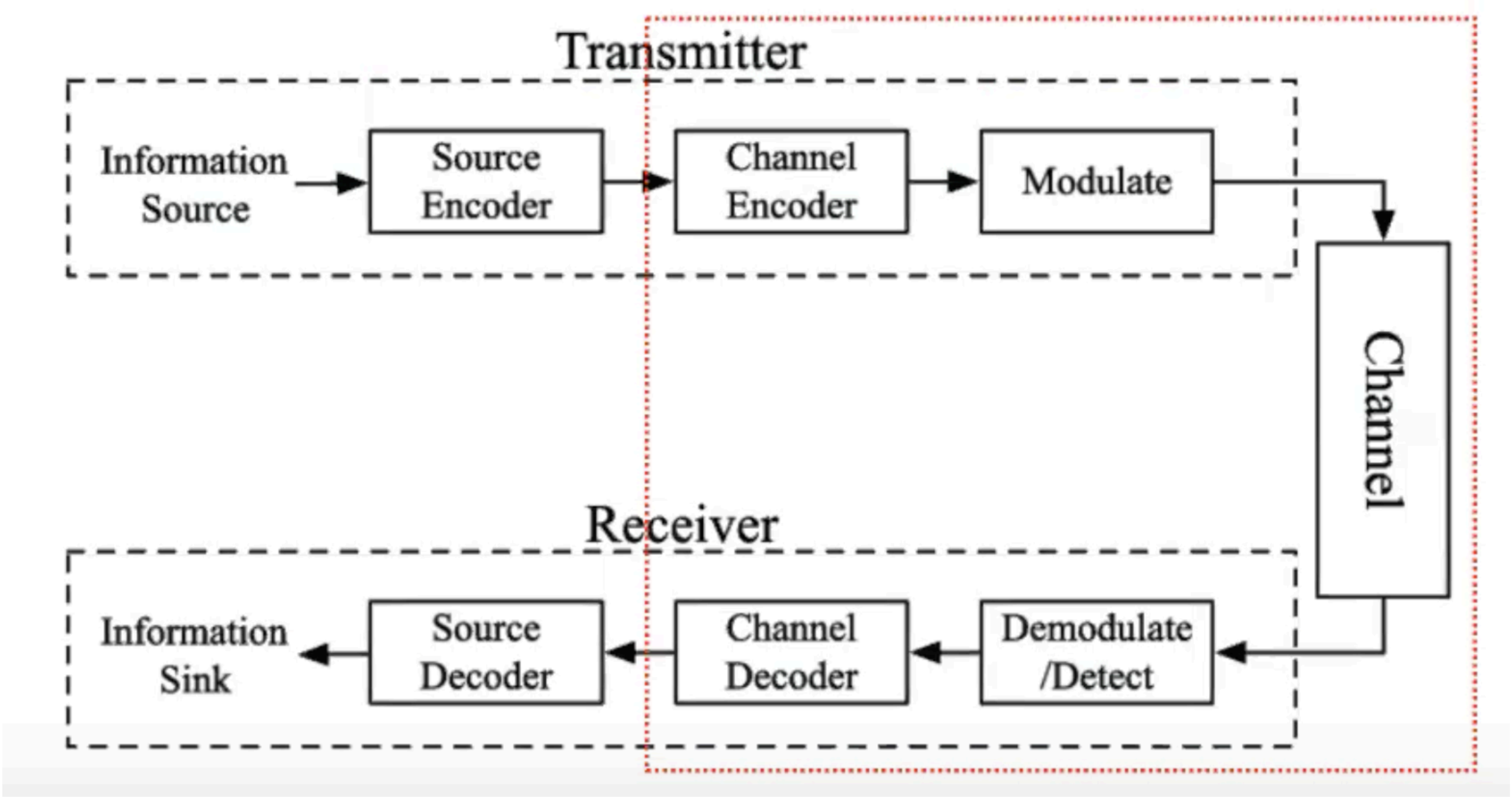
- Source code
  - https://github.com/yihanjiang/turboae
- Paper will be available soon
  - *"TurboAE: channel codes via deep learning"*

    Y. Jiang, H.Kim, H. Asani, S. Kannan, S. Oh, and P. Viswanath, NeurIPS '19

TurboAE: Joint Modulation and Coding

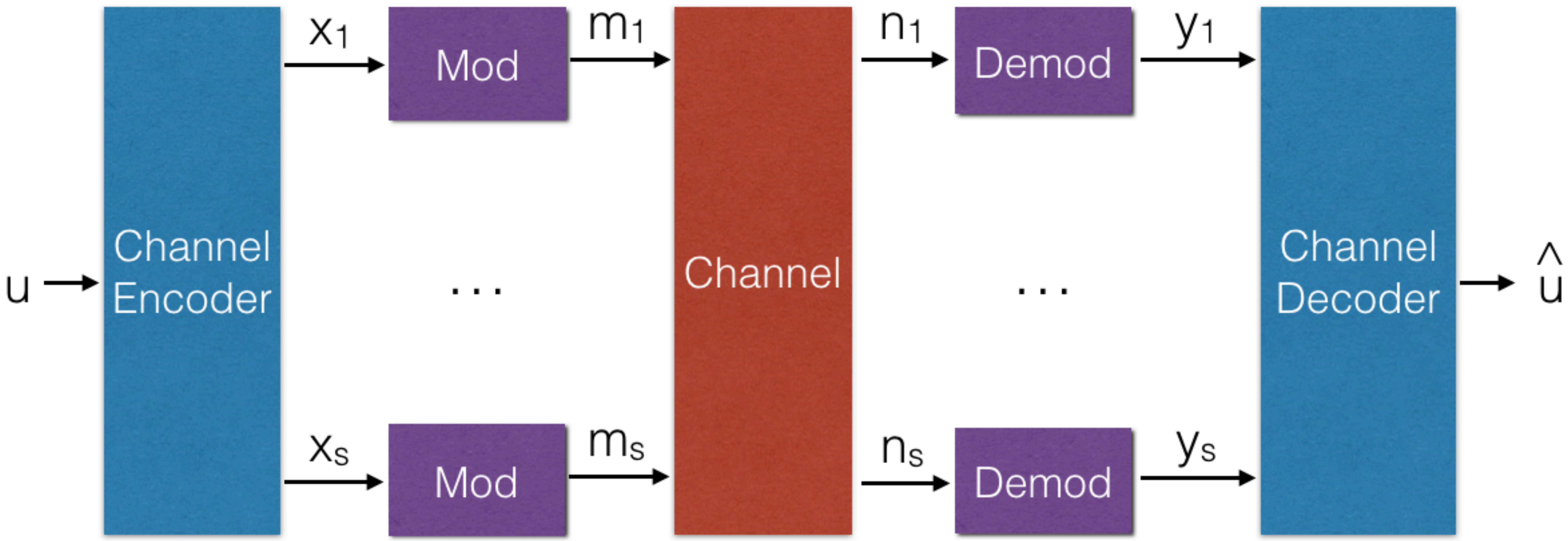
Joint Coding and Modulation

- TurboAE automatically learns coding + mod together.
  - But Learning separately allows rate adaptation using same code.



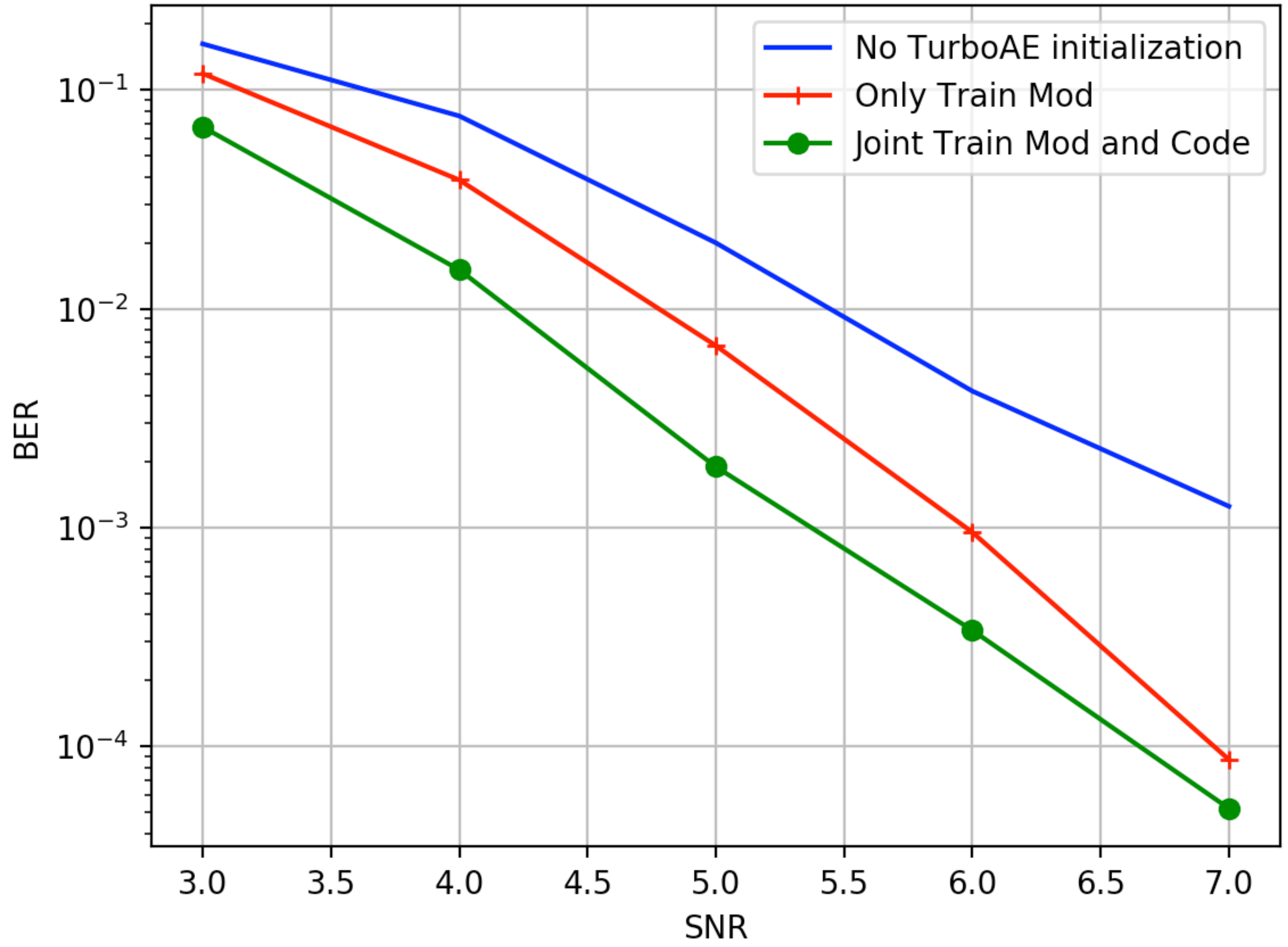
TurboAE with modulation

- All modules are neural network
  - Mod/Demod is small FCNN.
  - TurboAE is CNN+interleaver.



Training Algorithm

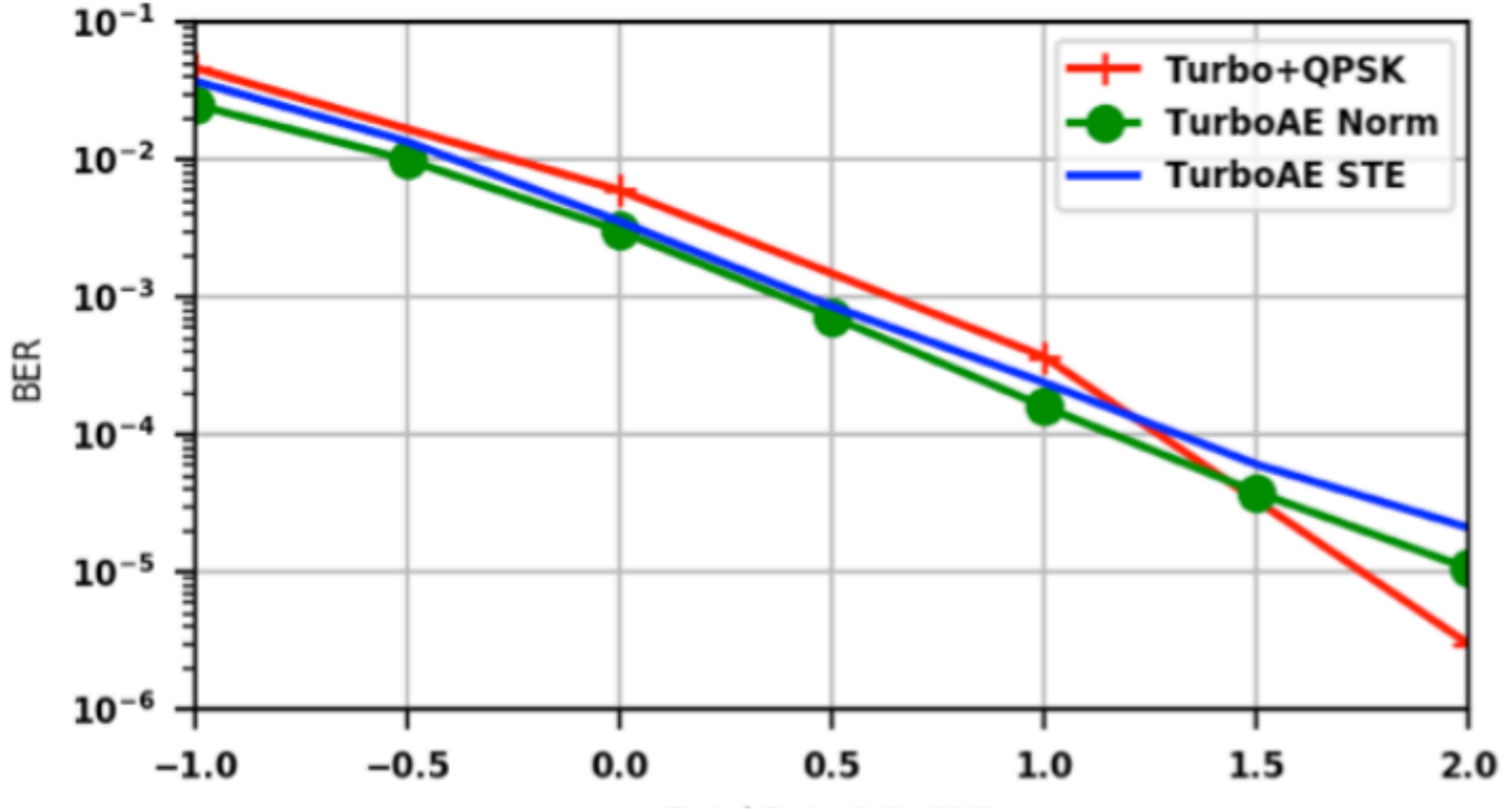
- If no TurboAE initialization, the performance drops.
- Just training modulation is suboptimal.
- Joint optimization lead to best performance



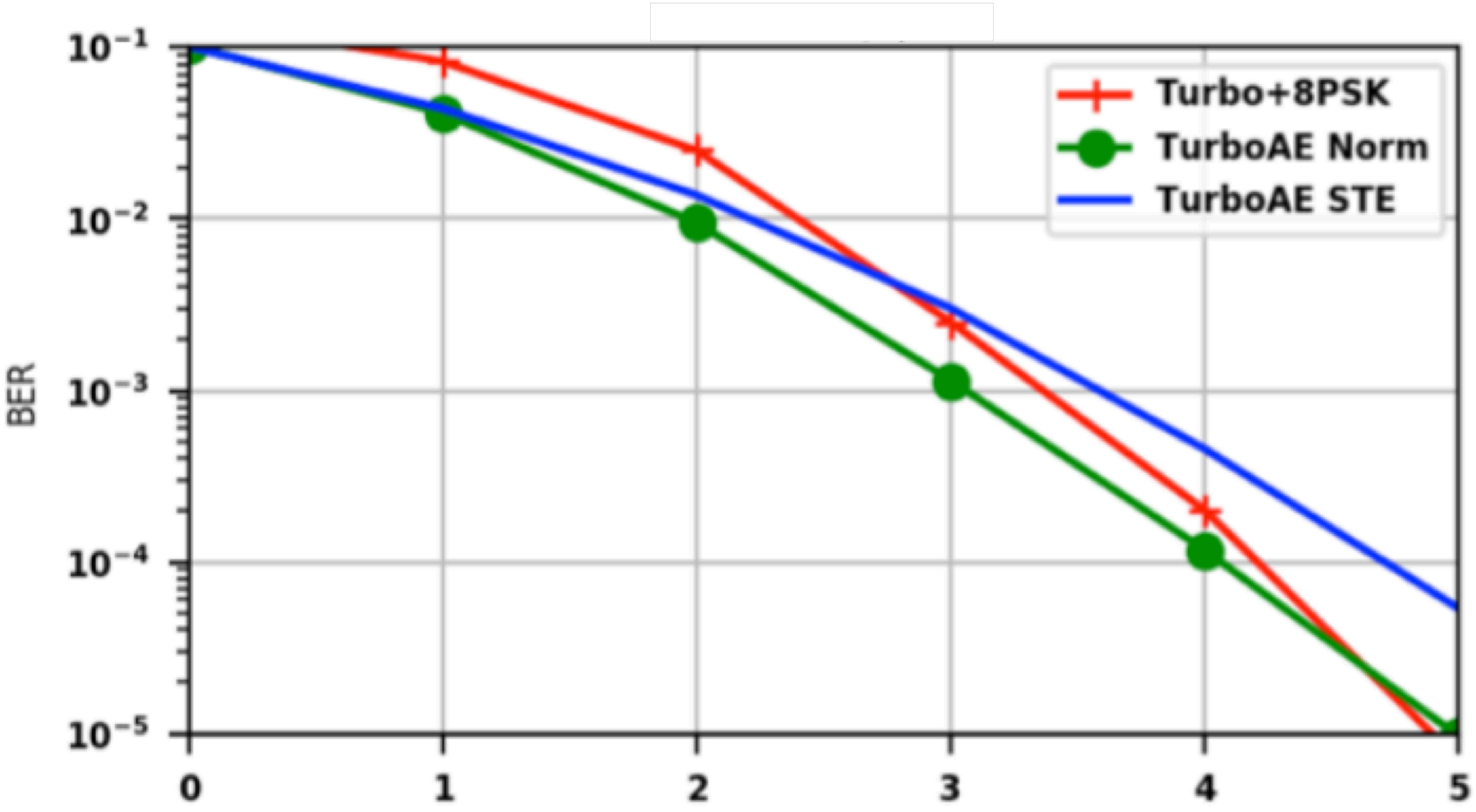
AWGN Performance

- Benchmarks:
  - Turbo+QPSK/8PSK/16QAM
- TurboAE-STE:
  - Still use QPSK/8PSK/16QAM
- TurboAE Norm:
  - Learned constellation.
- Better at low SNRs.
  - High SNR performance not good

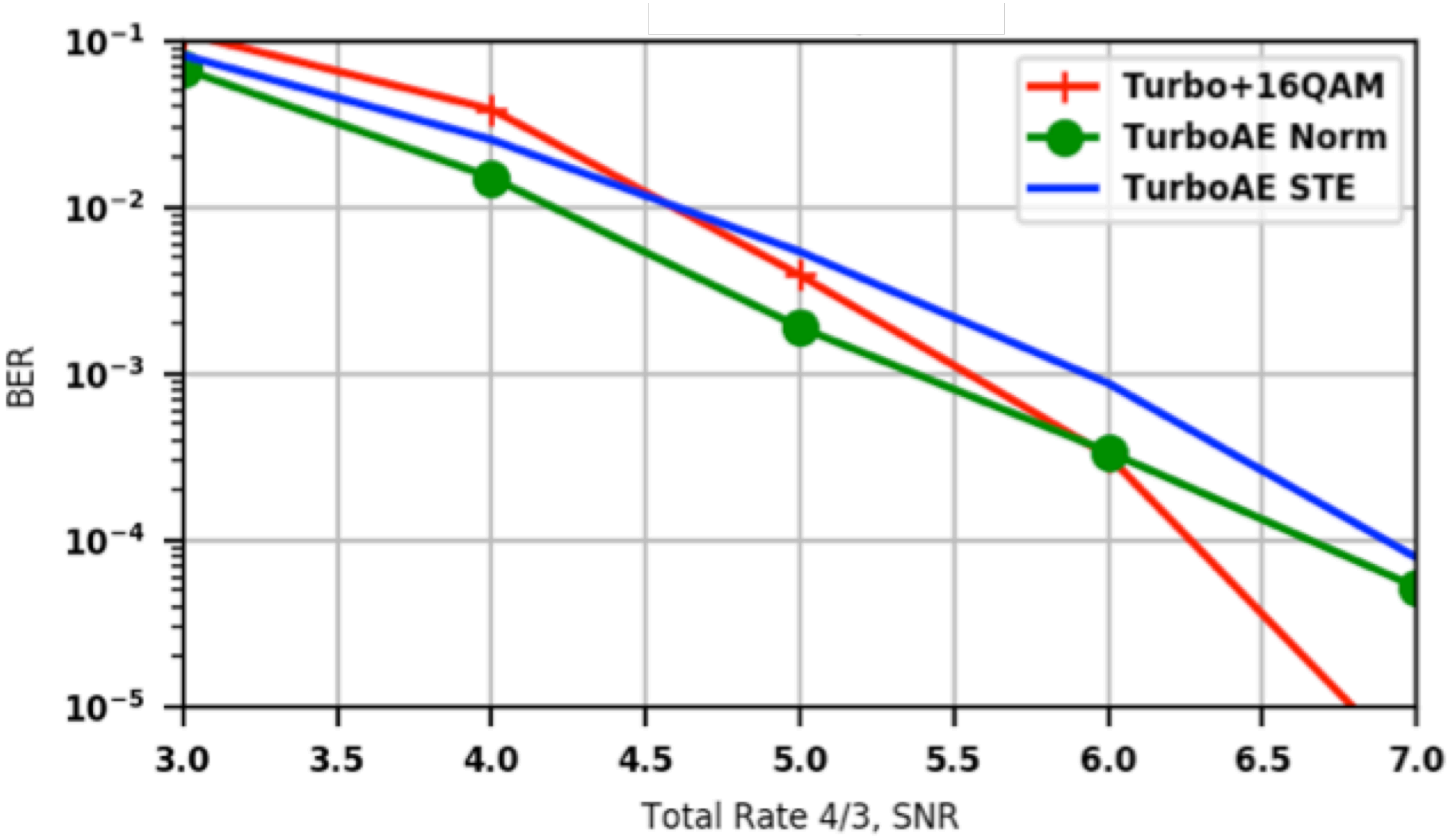
AWGN Performance



AWGN Performance

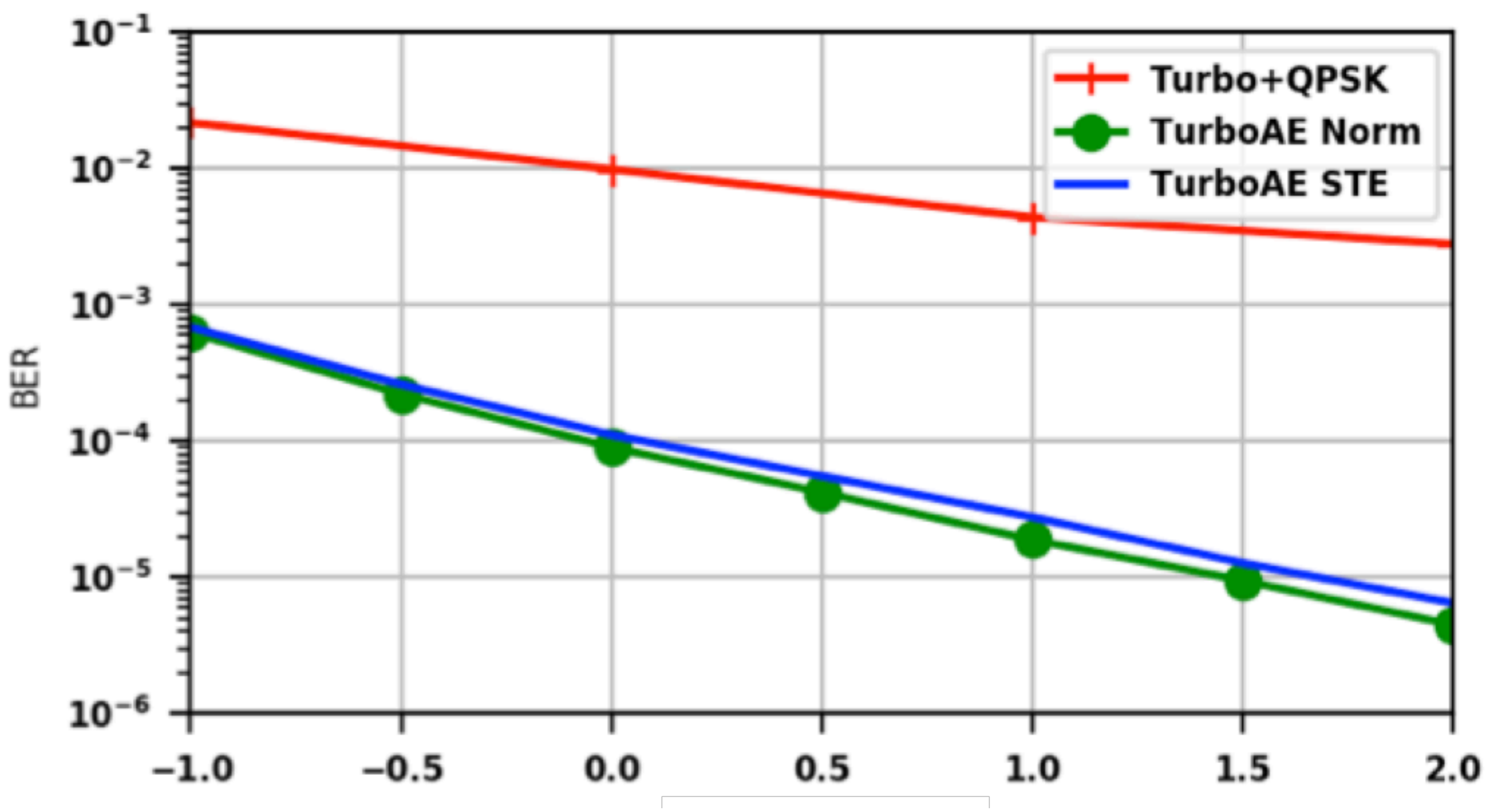


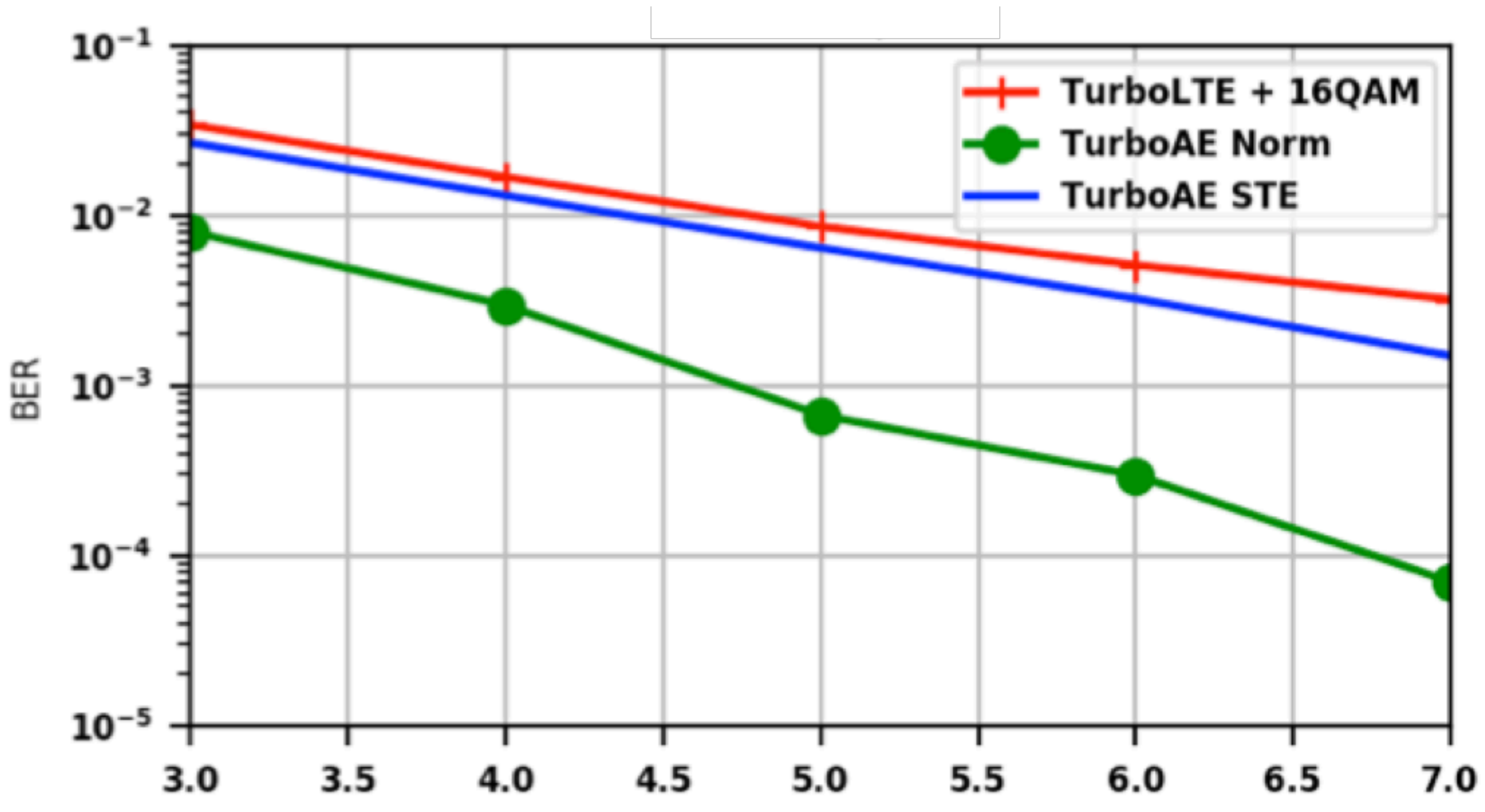
AWGN Performance



Non-AWGN performance

- ATN is much better than AWGN.
- Joint optimization is good:

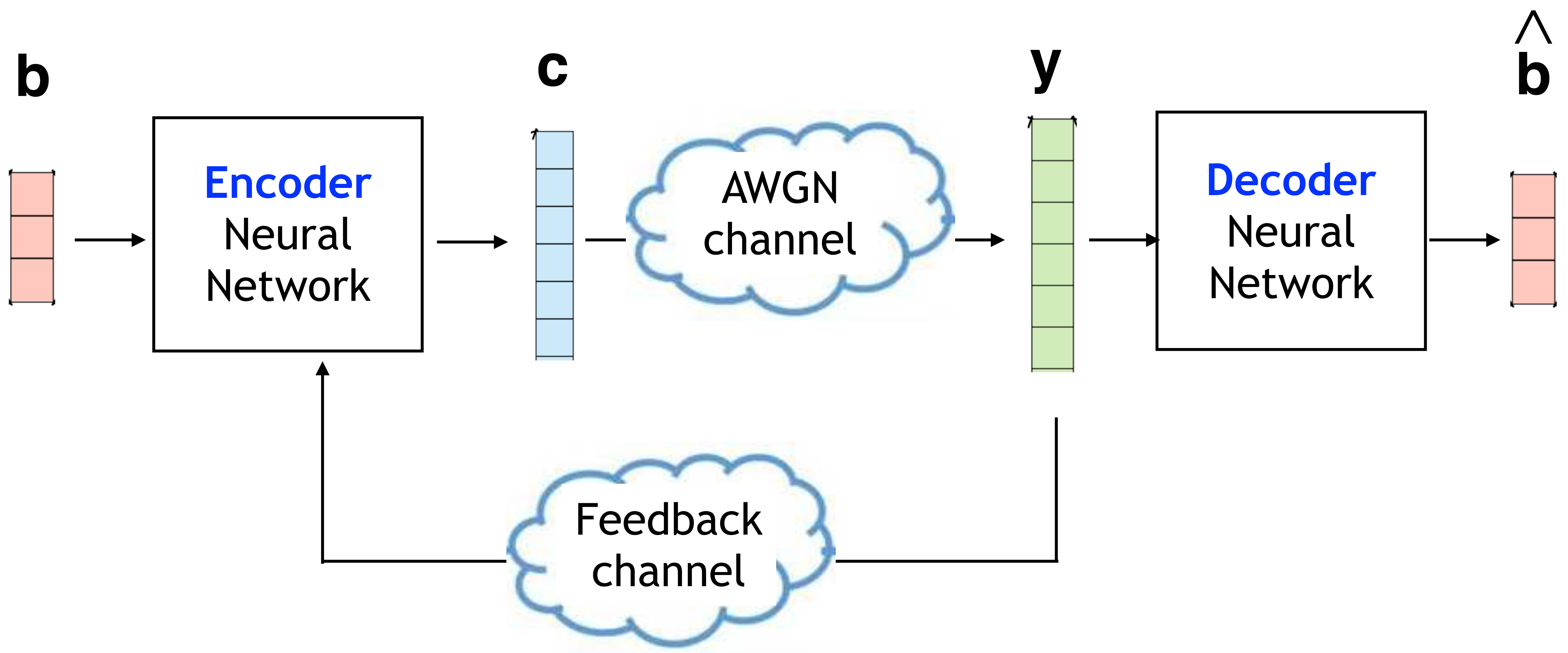




Feedback code with block output feedback

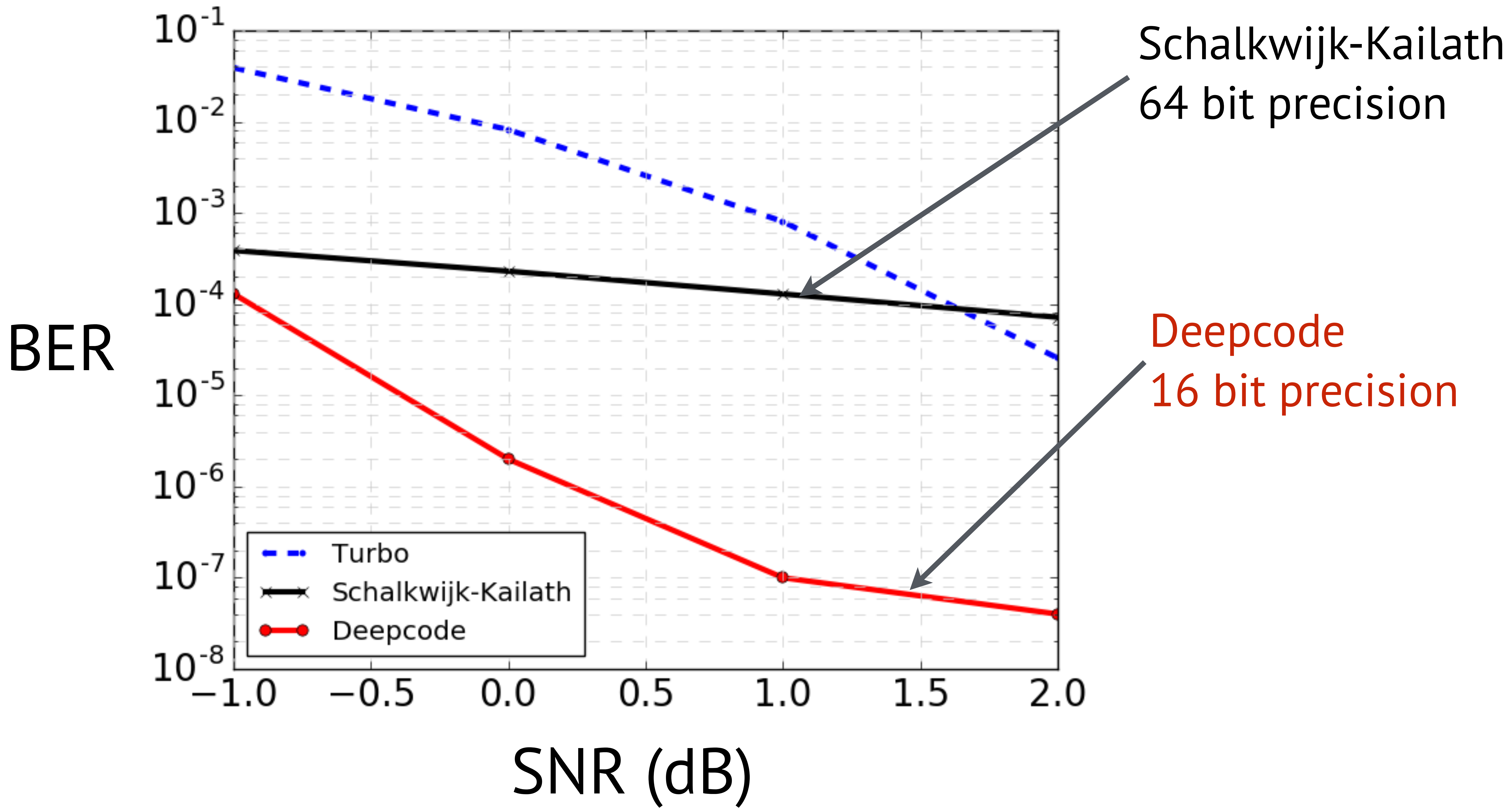
Deepcode: Recall

- Deepcode
  - Model encoder and decoder as recurrent neural networks and learn them jointly



Deepcode: Performance

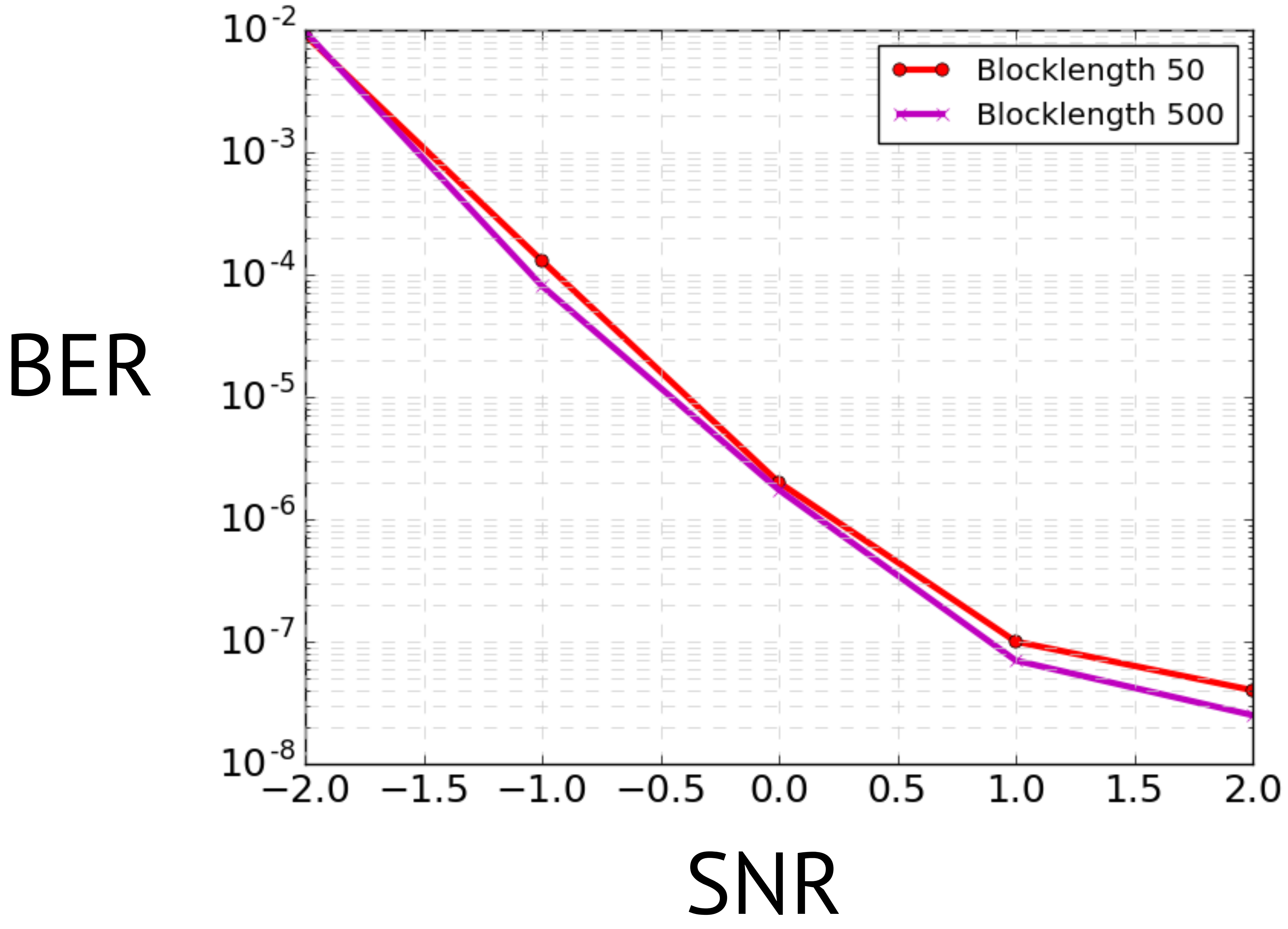
- 100x better reliability under noiseless feedback w. precision



(Rate 1/3, 50 bits)

Deepcode: Limitation

- Limitation of deepcode
  - Deepcode does not have a block-length gain



Deepcode: Limitation

- Limitation of deepcode
  - Deepcode does not have a block-length gain
  - Requires feedback with unit-step delay

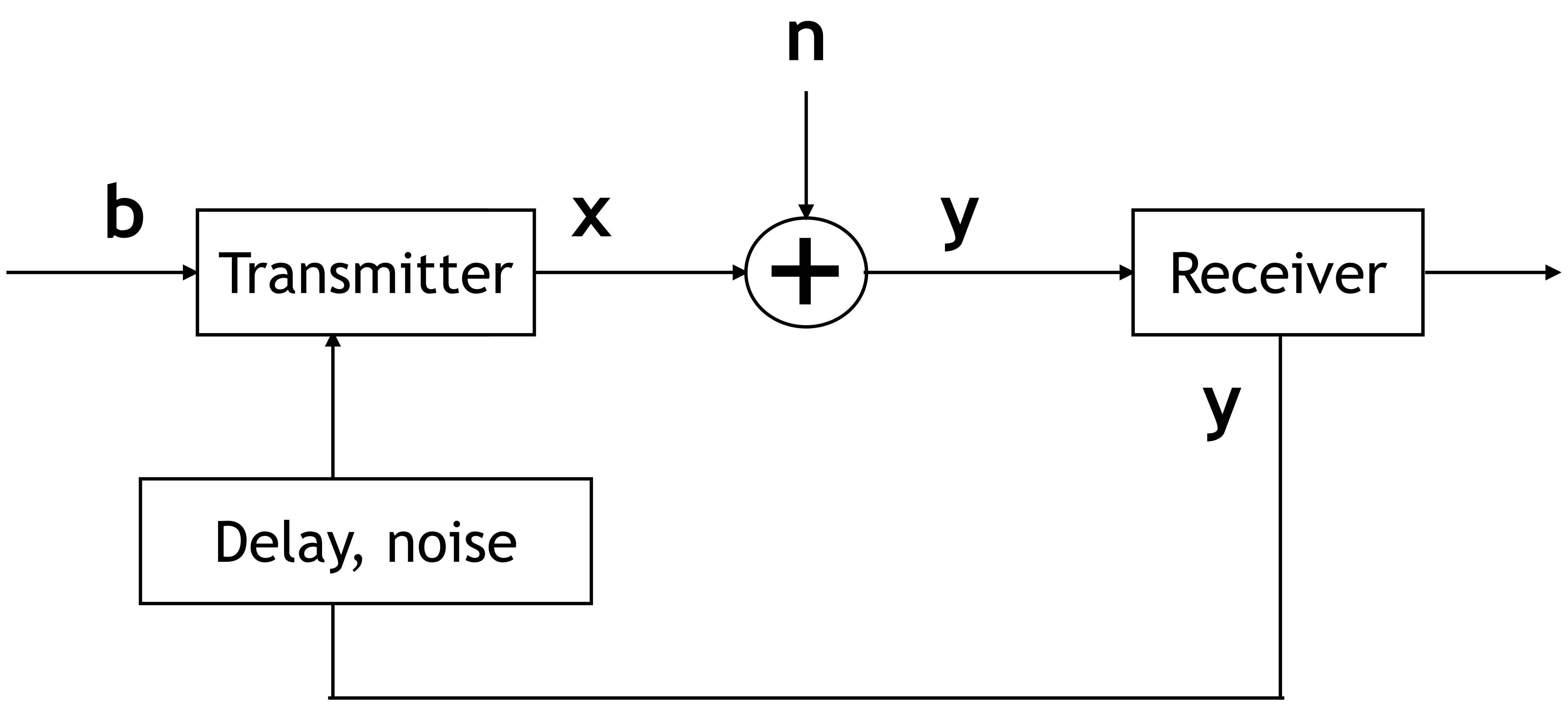
Deepcode: Limitation

- Limitation of deepcode
  - Deepcode does not have a block-length gain
  - Requires feedback with unit-step delay

- Feedback Turbo Autoencoder addresses these limitations!

Block-wise output feedback

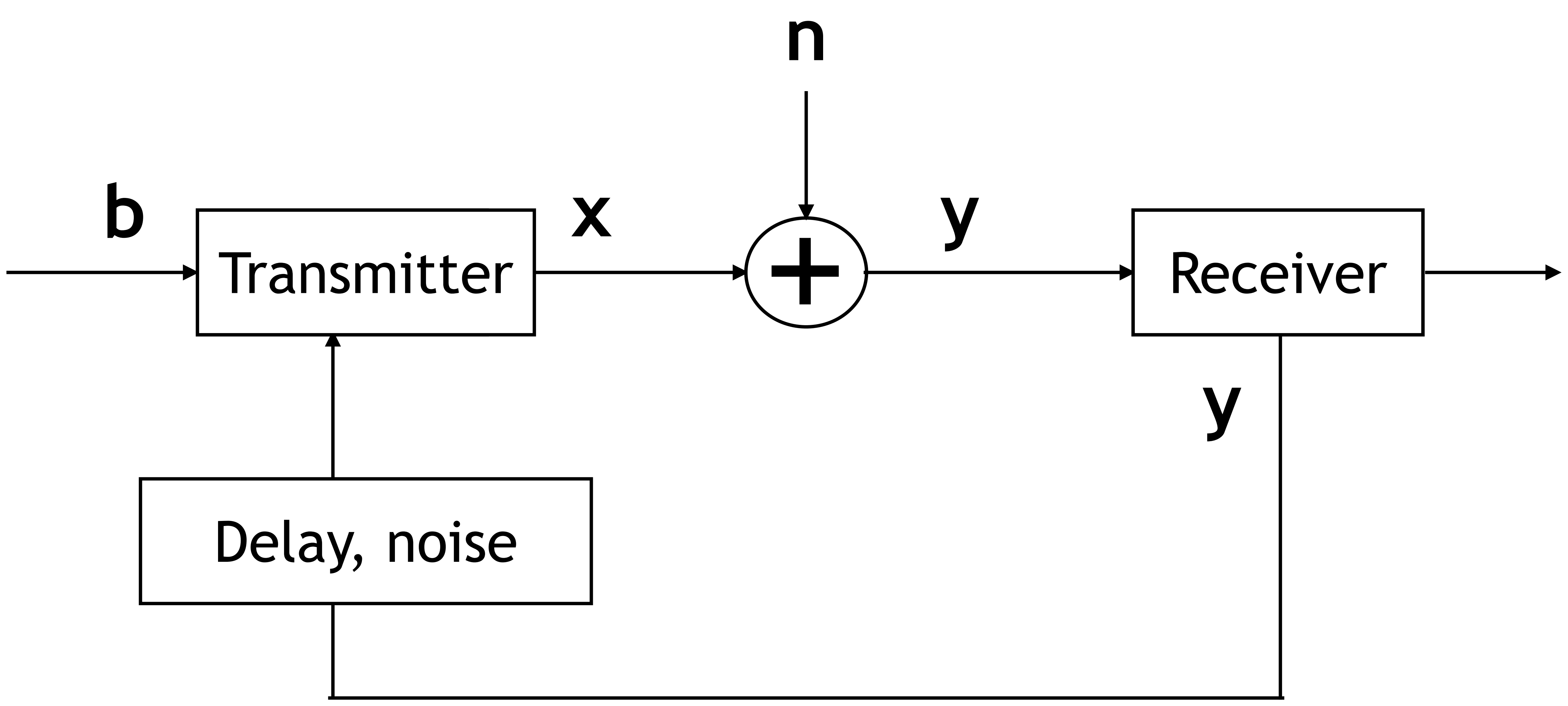
- Message: **b** = binary sequence of length k
- Code rate 1/3: $\mathbf{x} = (x_1, \ldots, x_{3k})$



Block-wise output feedback

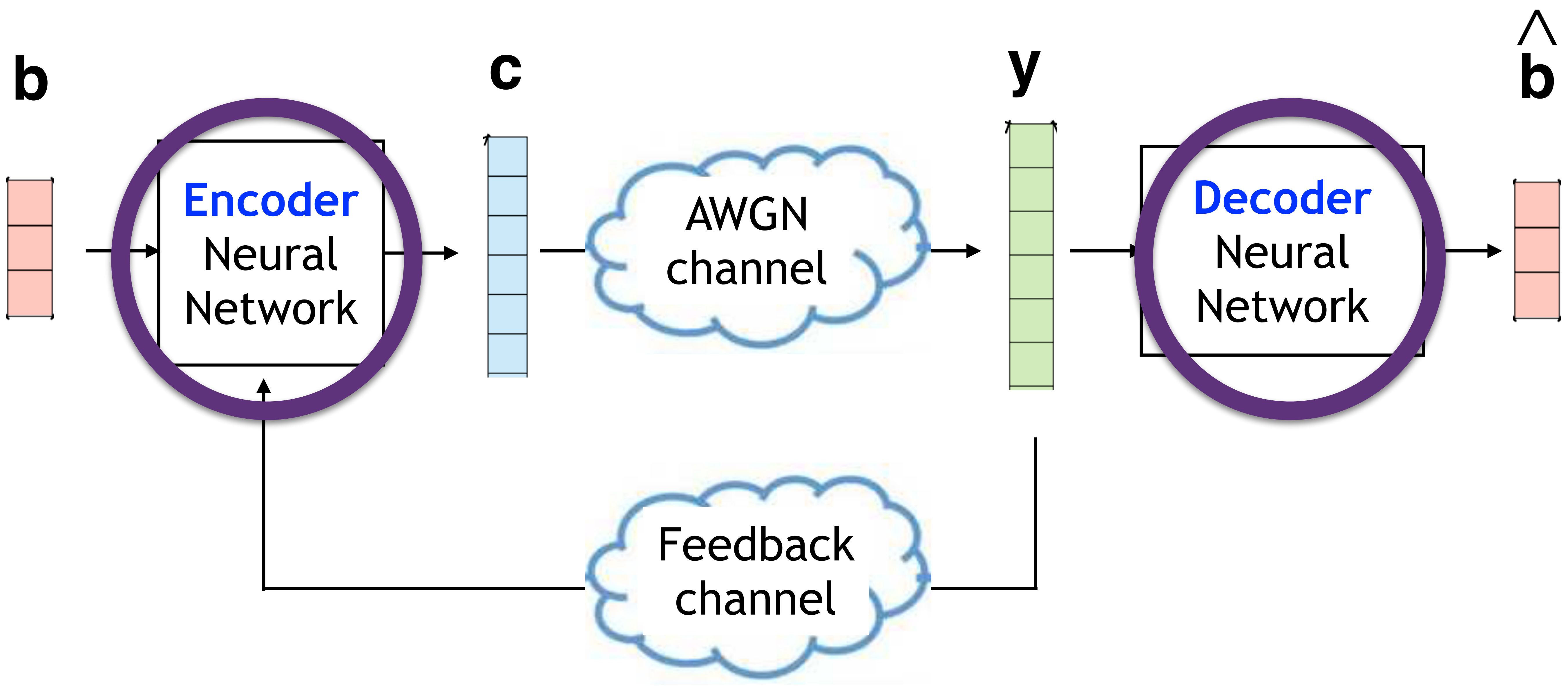
- Message: **b** = binary sequence of length k
- Code rate 1/3: $\mathbf{x} = (x_1, \dots, x_{3k})$
- Block feedback:

  At time k+1, Tx gets $(y_1, \dots, y_k)$; At time 2k+1, Tx gets $(y_{k+1}, \dots, y_{2k})$



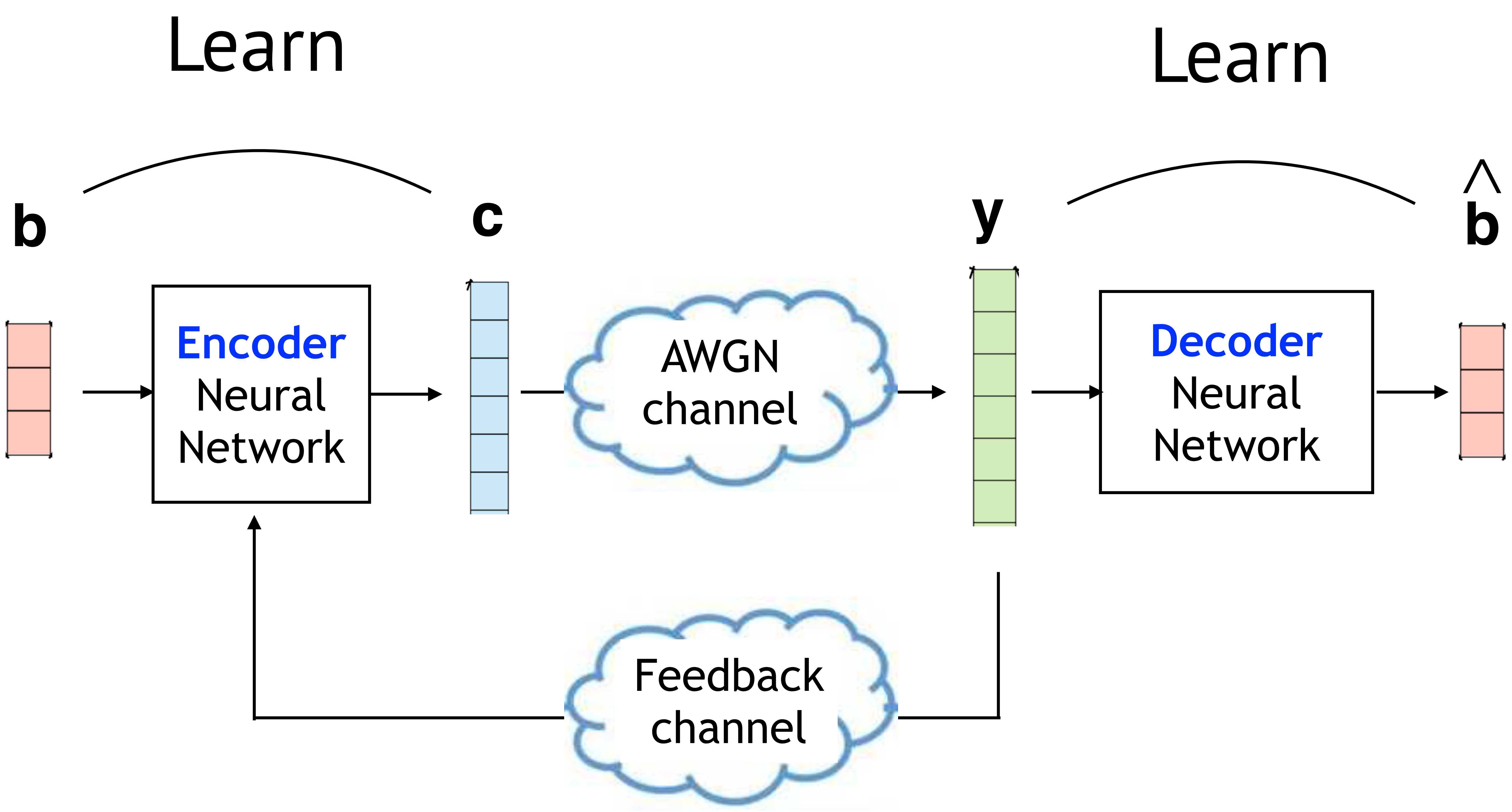
Outline

1. Neural network based encoder and decoder



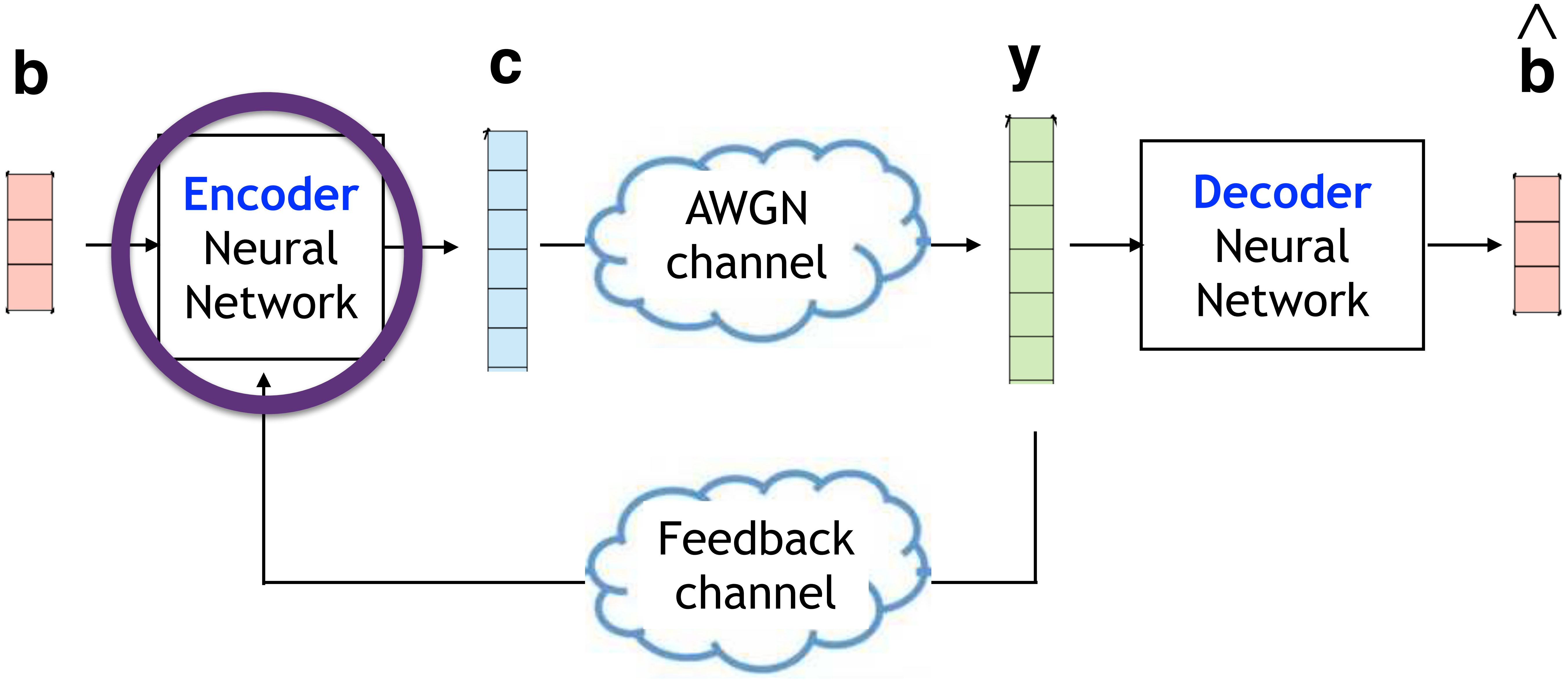
Outline

1. Neural network based encoder and decoder
2. Training



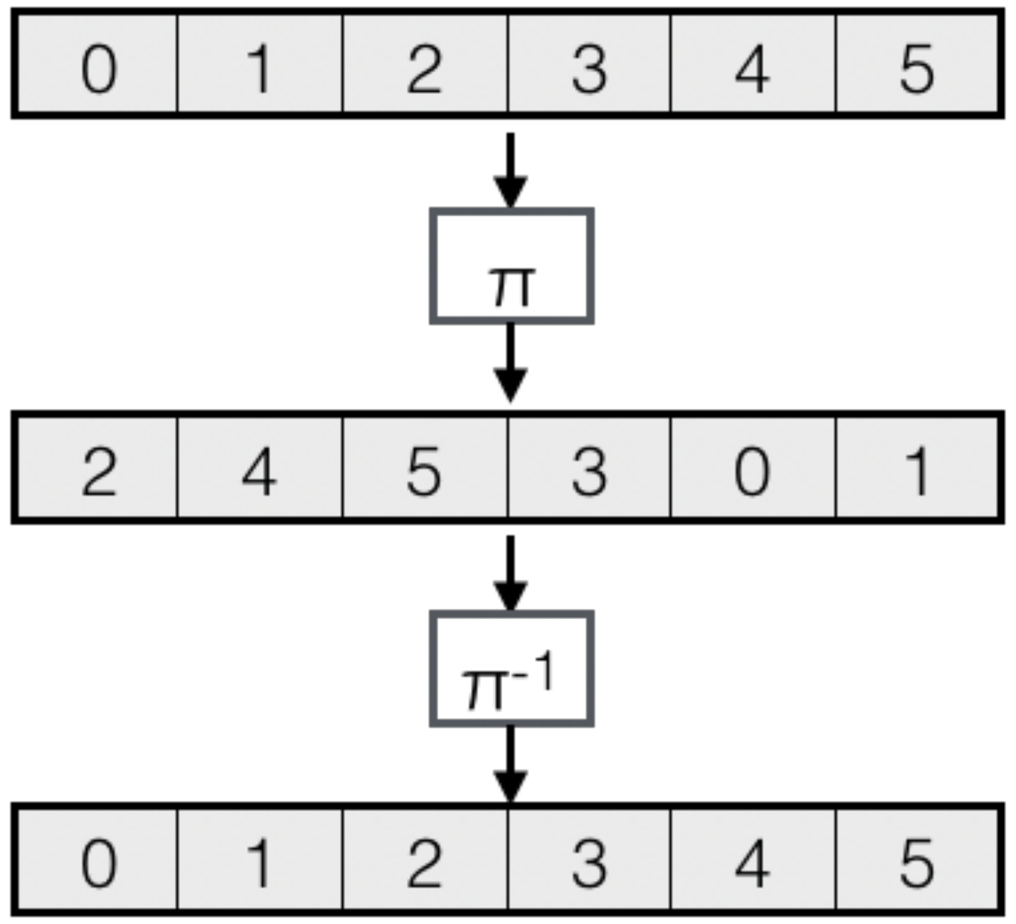
Outline

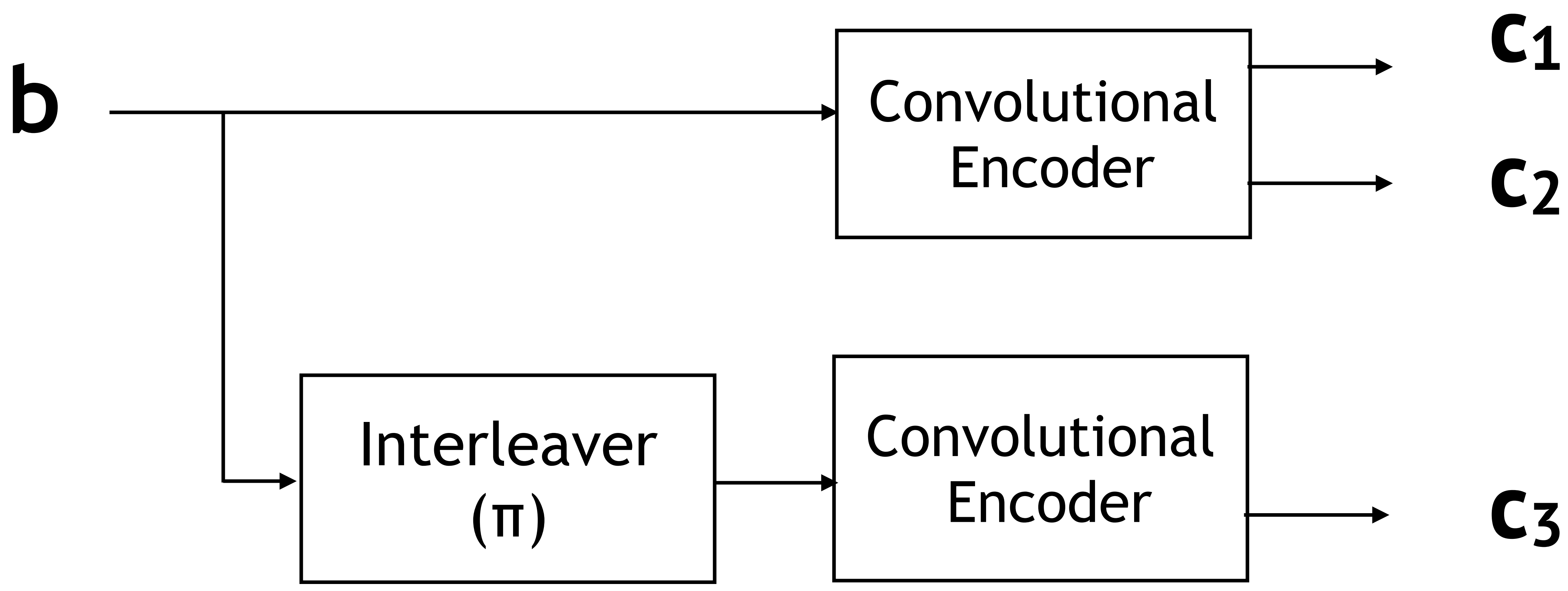
1. Neural network based encoder and decoder



Inspiration from Turbo code

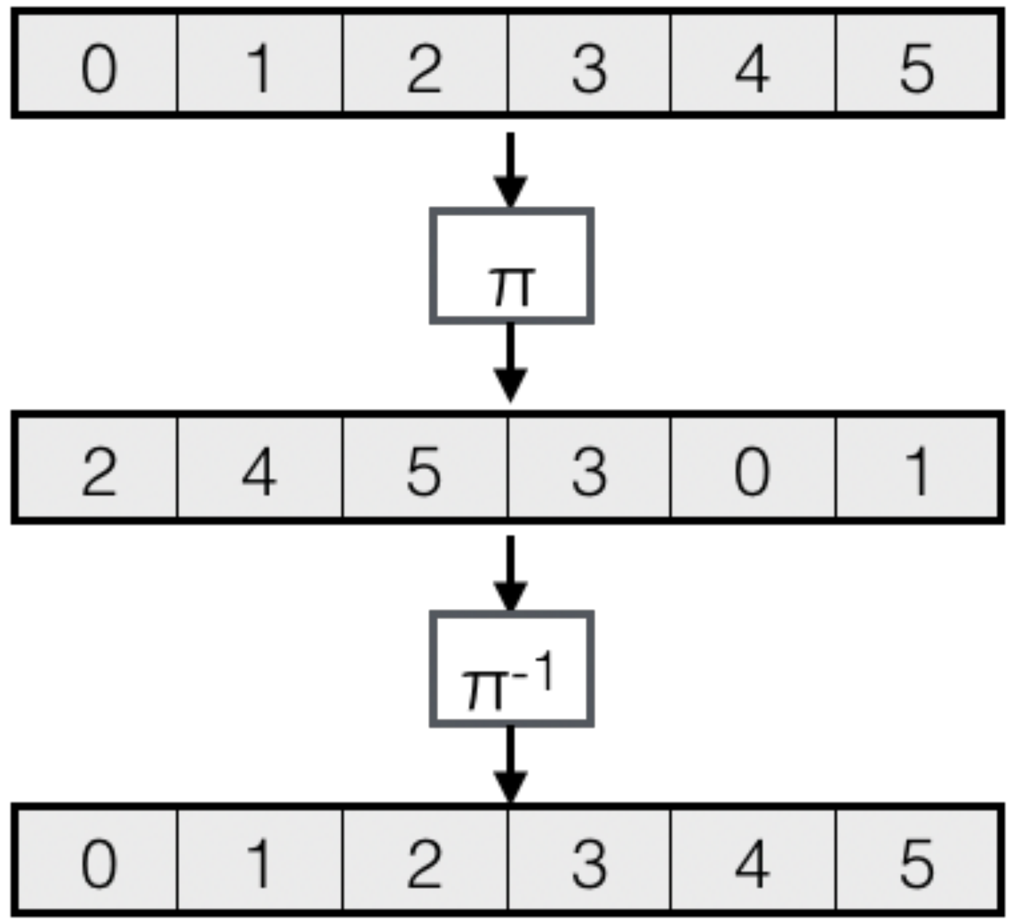
- Concatenate codewords from two convolutional encoders

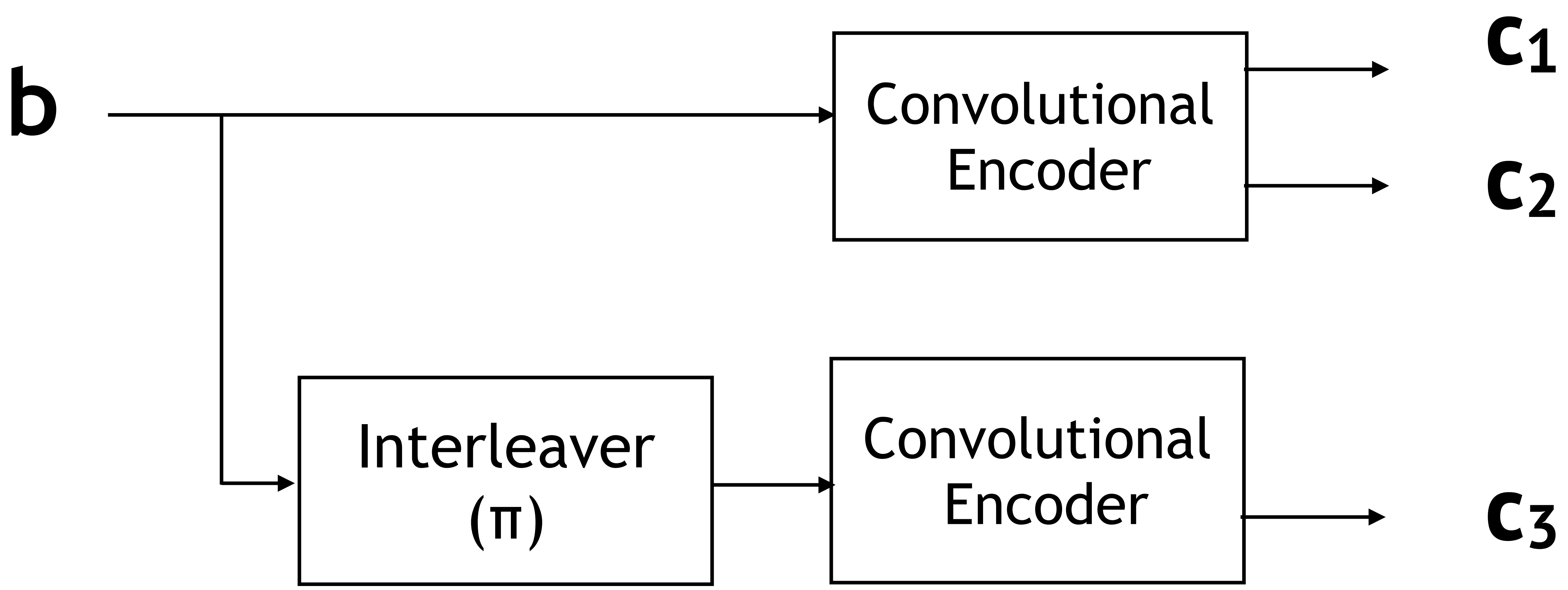




Inspiration from Turbo code

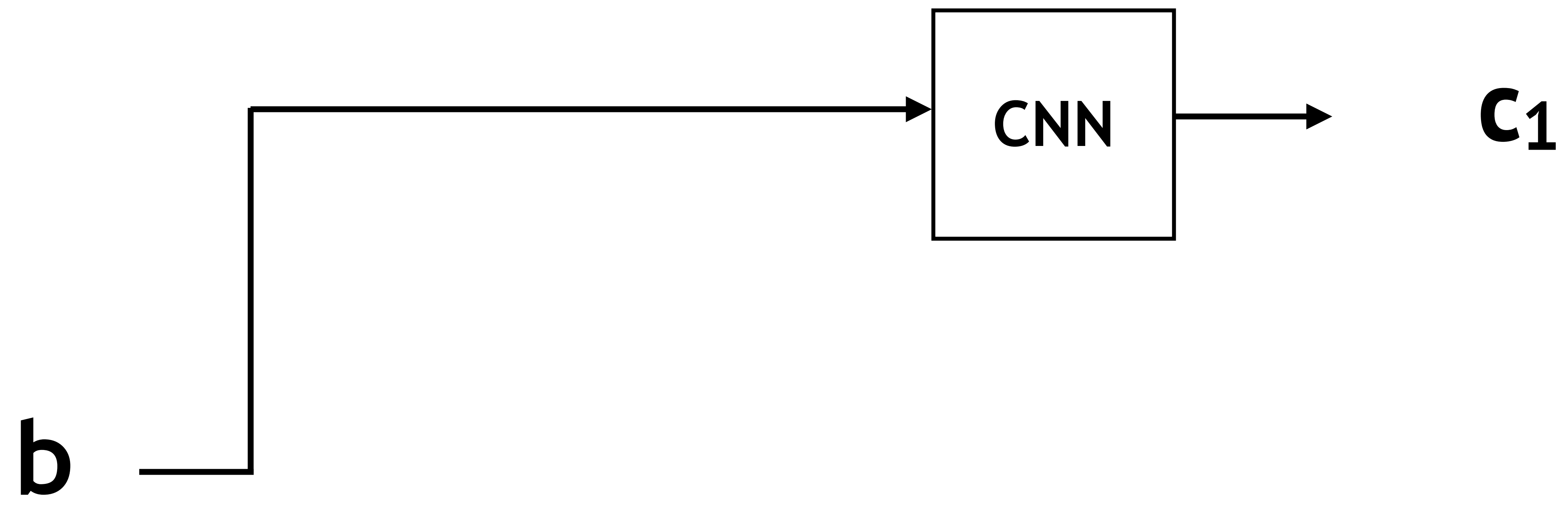
- Concatenate codewords from two convolutional encoders
  - Long term memory via interleaver





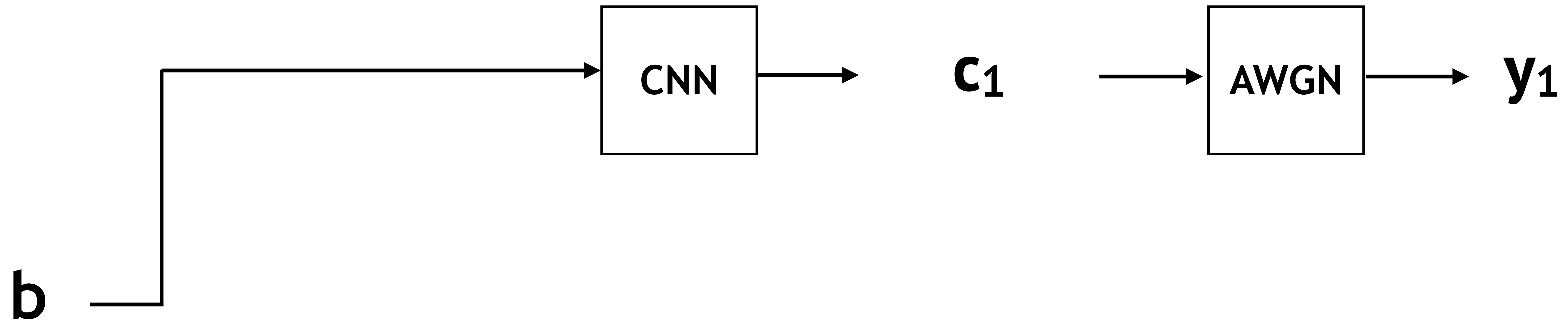
Encoder as a CNN with an interleaver

- 1D convolutional neural network



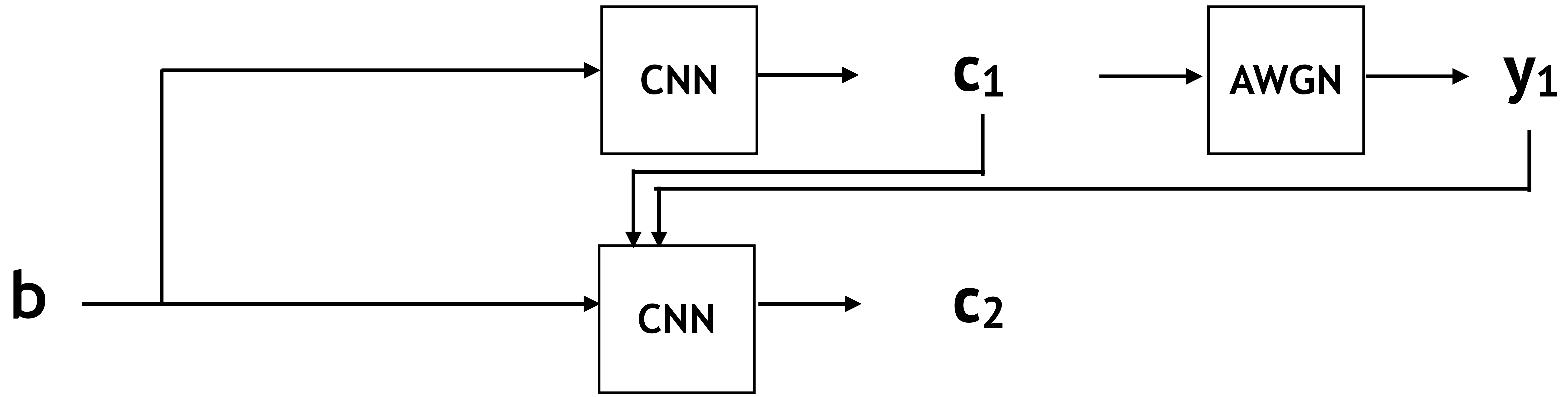
Encoder as a CNN with an interleaver

- 1D convolutional neural network



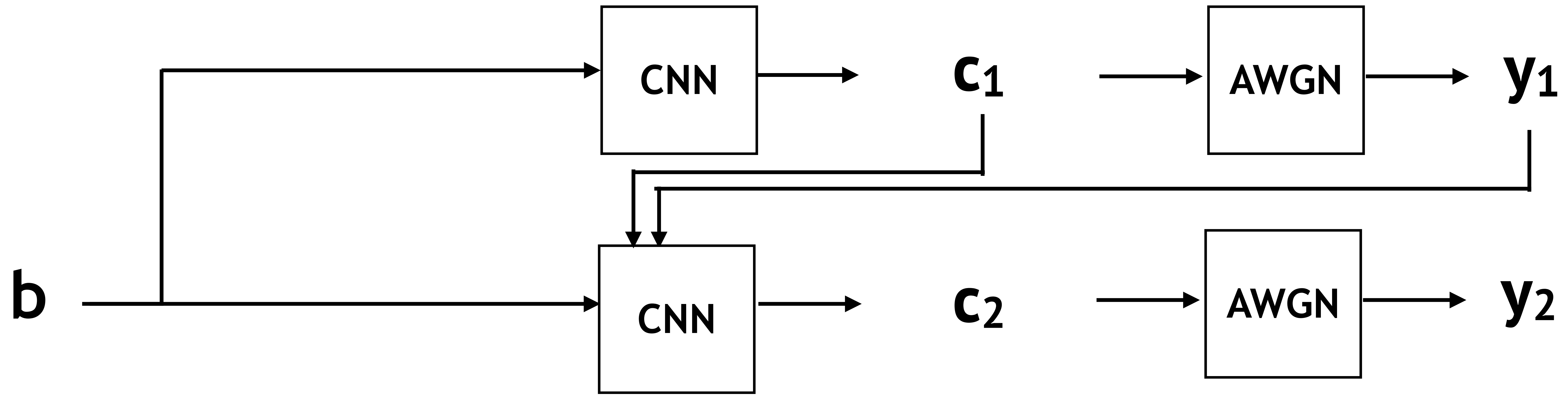
Encoder as a CNN with an interleaver

- 1D convolutional neural network



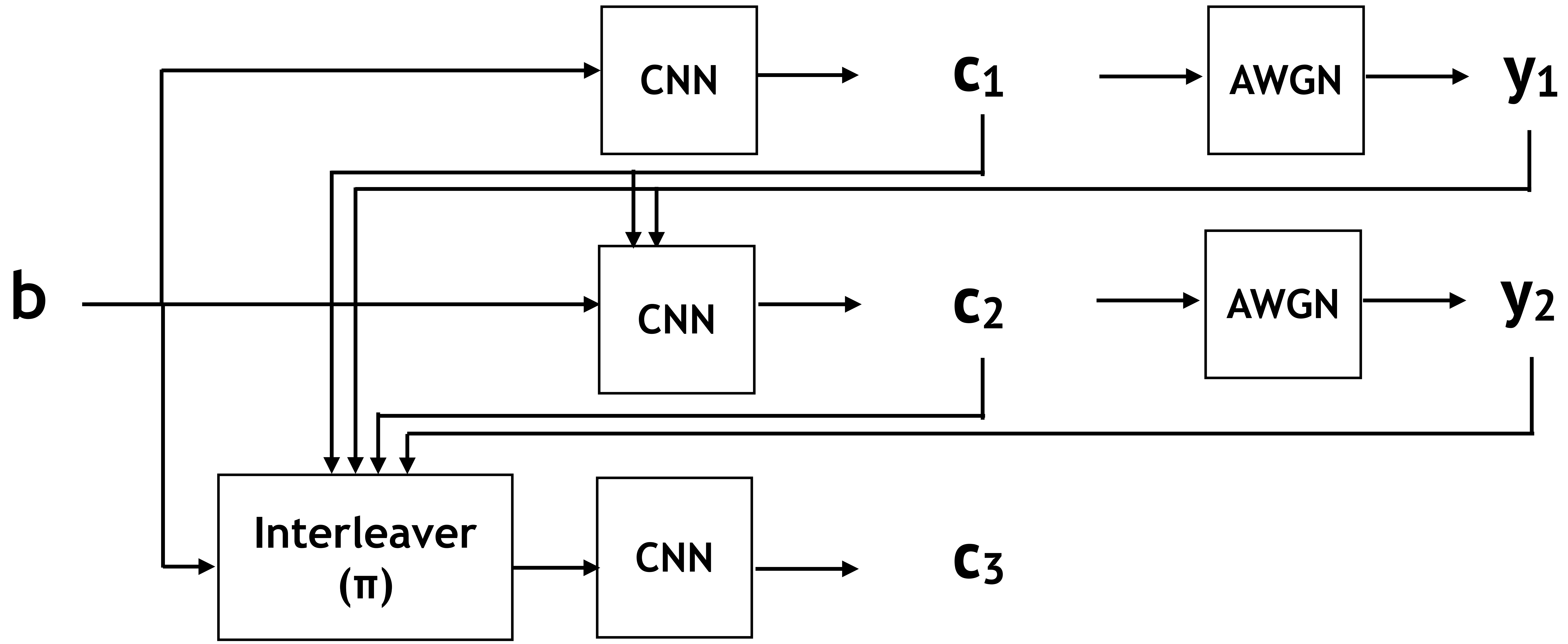
Encoder as a CNN with an interleaver

- 1D convolutional neural network



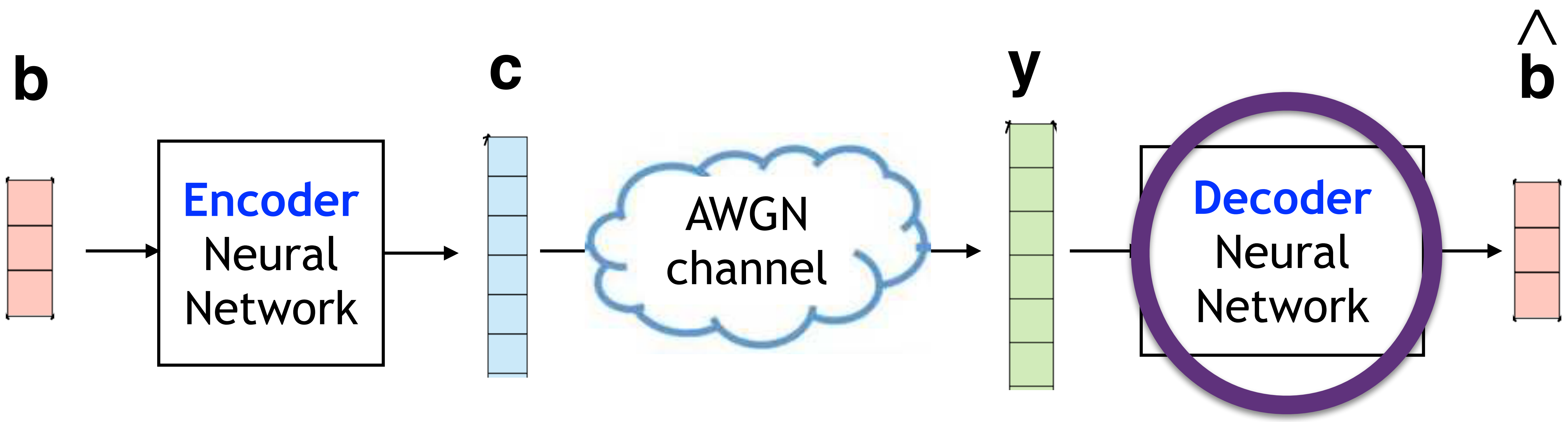
Encoder as a CNN with an interleaver

- 1D convolutional neural network



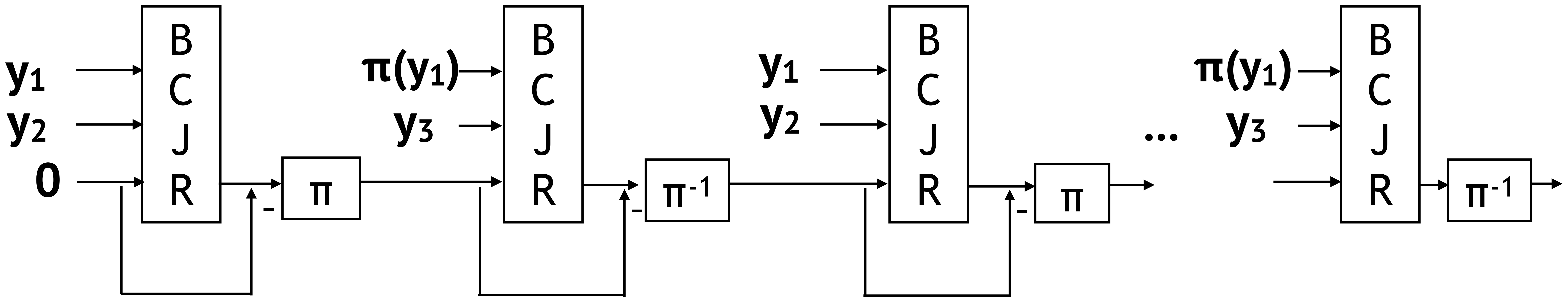
Outline

1. Neural network based encoder and decoder



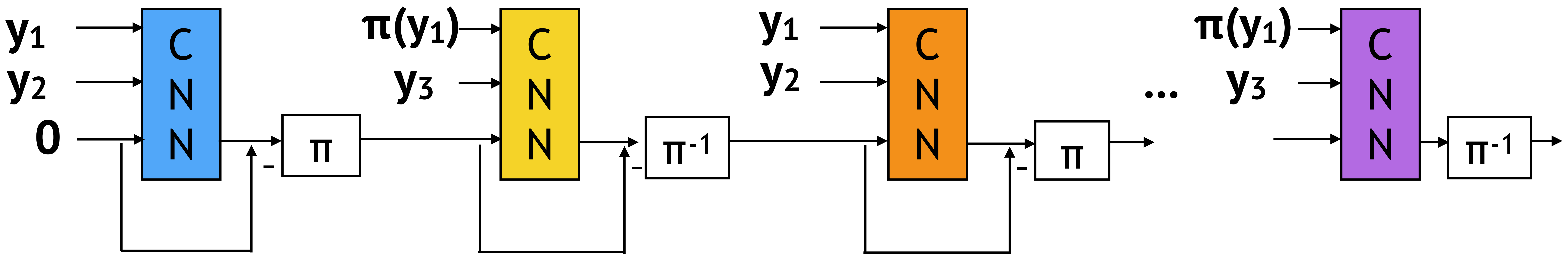
Belief Propagation Decoding

- Iterations of BCJR w. interleaver (π), de-interleaver (π^{-1})
  - BCJR: maps (prior, noisy codewords) to posterior



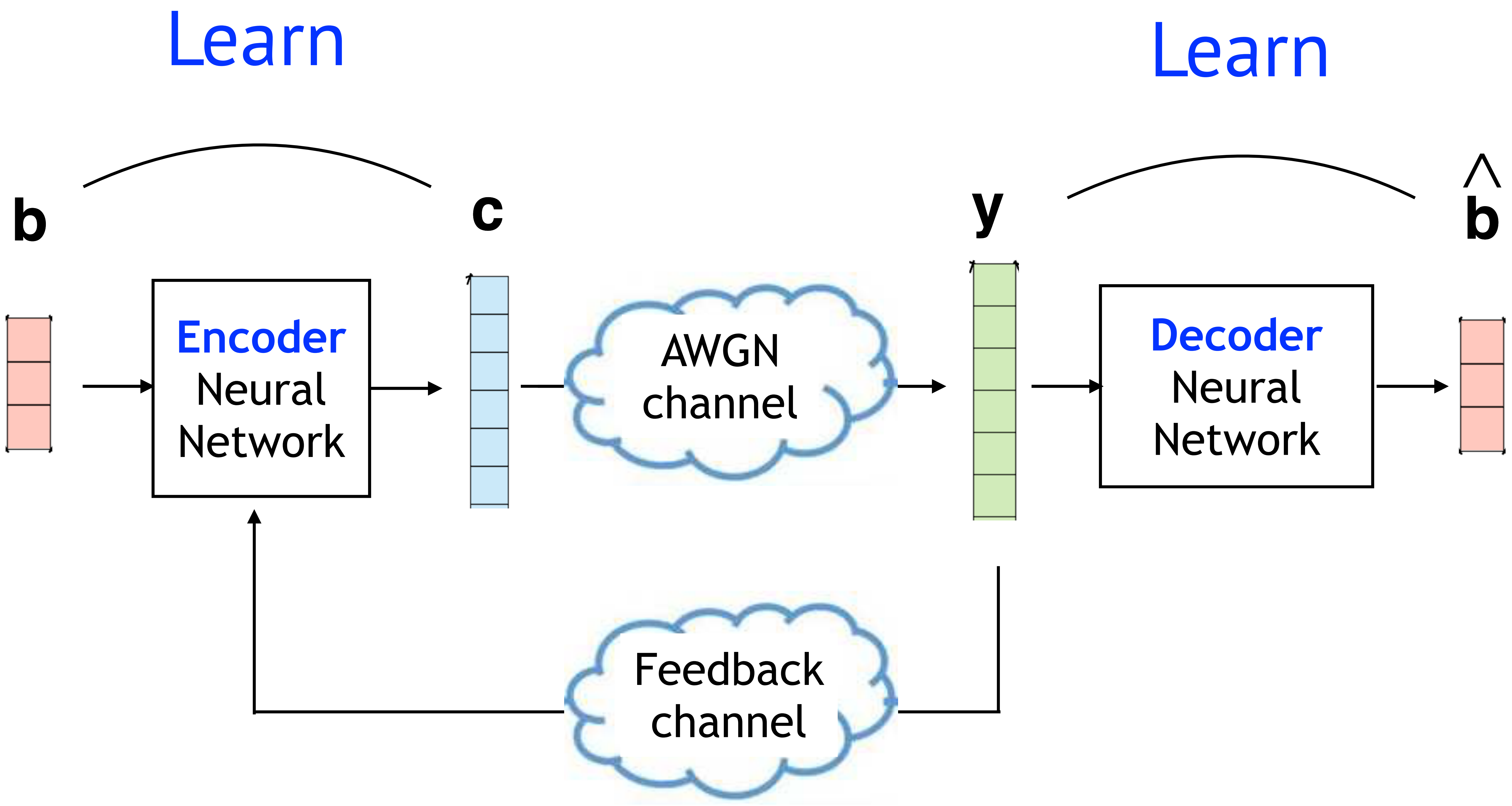
Decoder as a CNN

- Model decoder as a CNN with a vector belief propagation



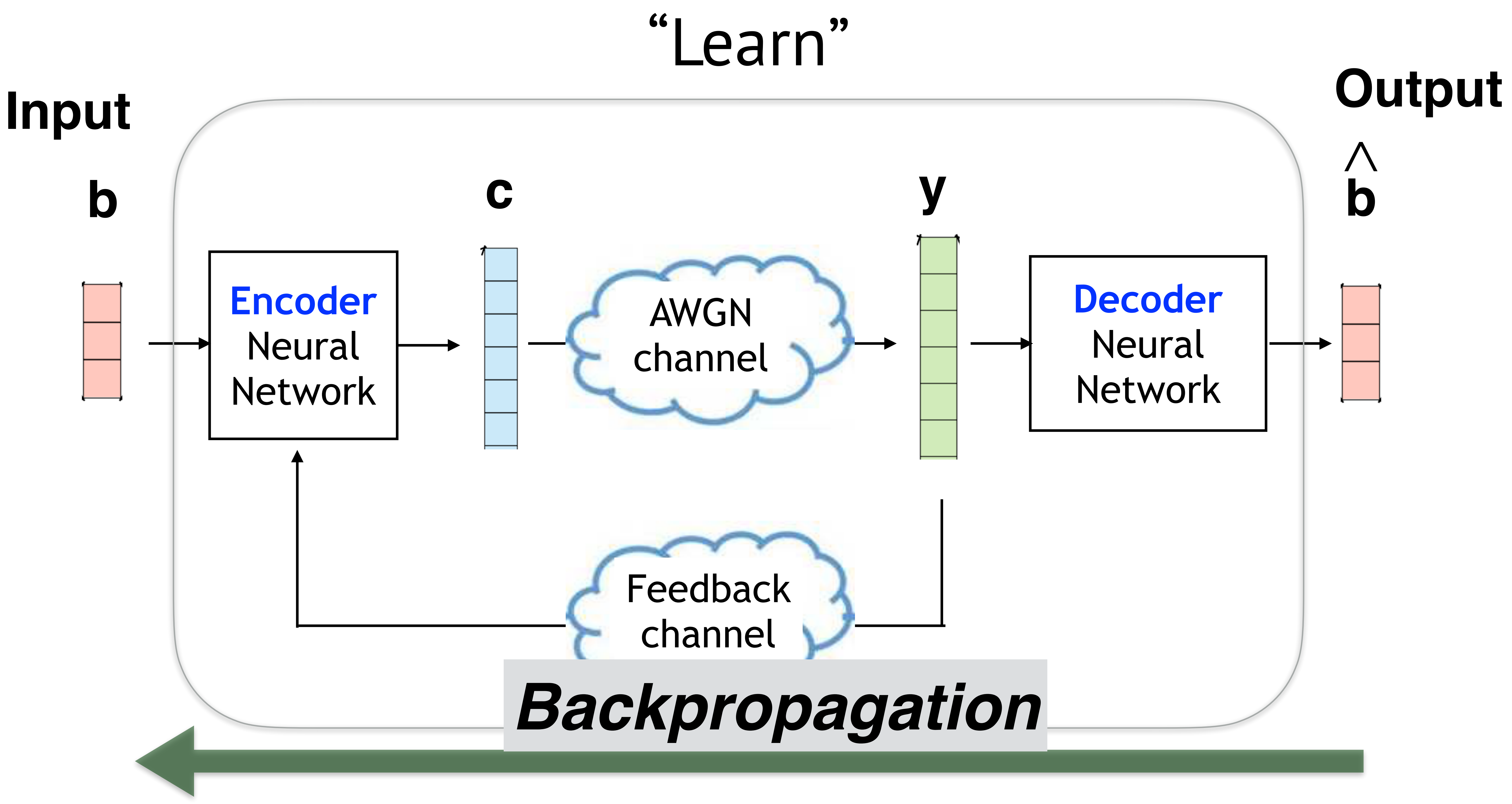
Outline

1. Neural network based encoder and decoder
2. Training



Training

- Auto-encoder training : (input,output) = (**b**,**b**)
- Loss : binary cross entropy $\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = -\mathbf{b} \log \hat{\mathbf{b}} - (1 - \mathbf{b}) \log(1 - \hat{\mathbf{b}})$



Two stage training

1. Train FTAE without feedback until convergence
2. Train FTAE with feedback

Two stage training

1. Train FTAE without feedback until convergence
    - Let feedback signal $y \sim N(0, 0.01^2)$

Two stage training

1. Train FTAE without feedback until convergence
   - Let feedback signal $y \sim N(0, 0.01^2)$
   - Training techniques
     - Large batch size
     - Alternating training of the encoder and decoder (5x)

Two stage training

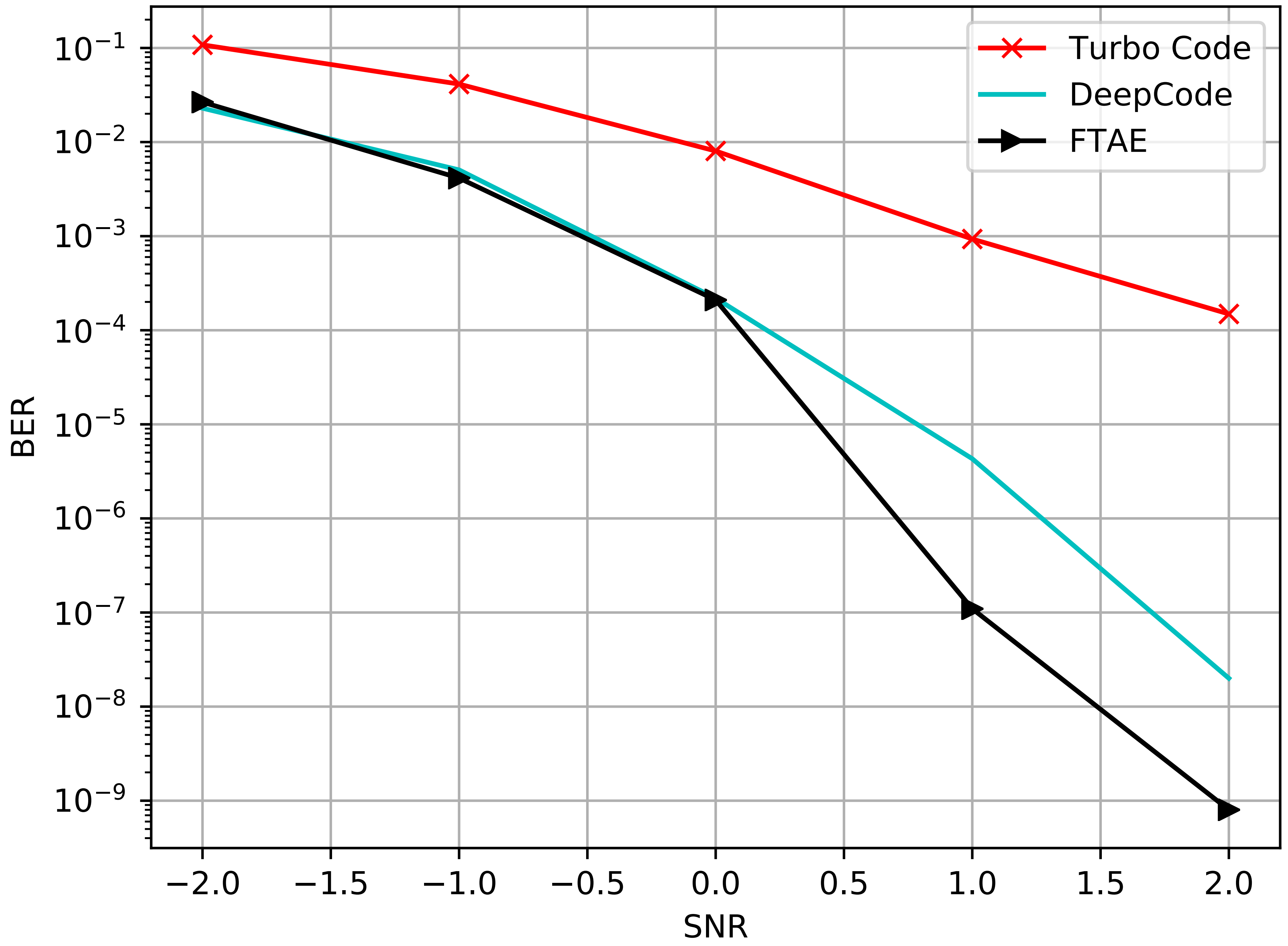
1. Train FTAE without feedback until convergence
2. Train FTAE with feedback

Two stage training

1. Train FTAE without feedback until convergence
2. Train FTAE with feedback
   - Training techniques
     - Large batch size
     - Alternating training of the encoder and decoder (equally)

Results

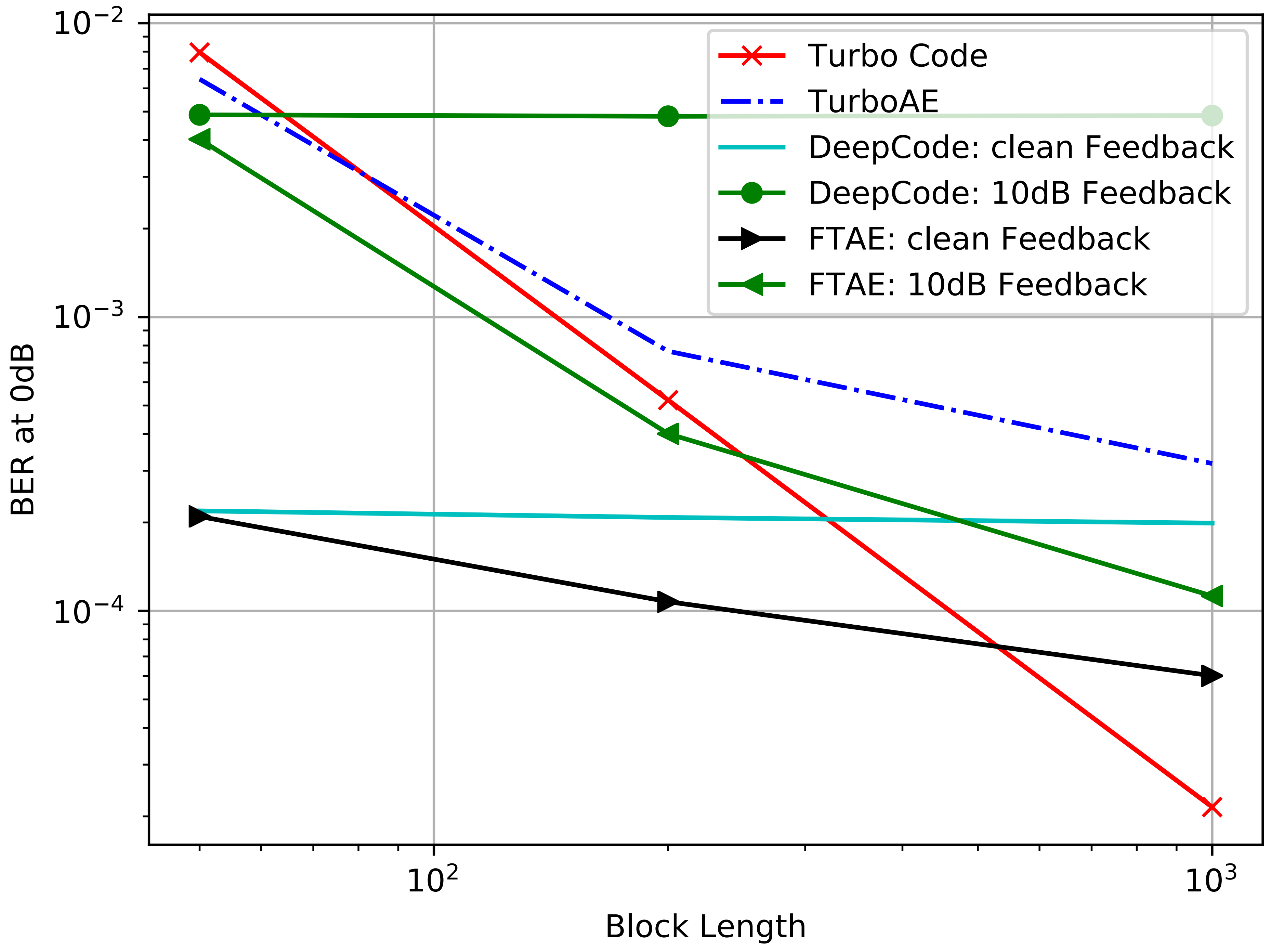
- FTAE outperforms Deepcode at high SNR



(Rate 1/3, 50 bits, noiseless feedback)

Results

- FTAE demonstrates block length gain:
  - BER ↓ as block length ↑



(Rate 1/3)

Conclusion

- Feedback Turbo Autoencoder
  - CNN based code for channels with feedback, inspired by turbo codes
  - Two-stage training

Conclusion

- Feedback Turbo Autoencoder
  - CNN based code for channels with feedback, inspired by turbo codes
  - Two-stage training
  - Outperforms existing codes in reliability
  - Achieves a block length gain

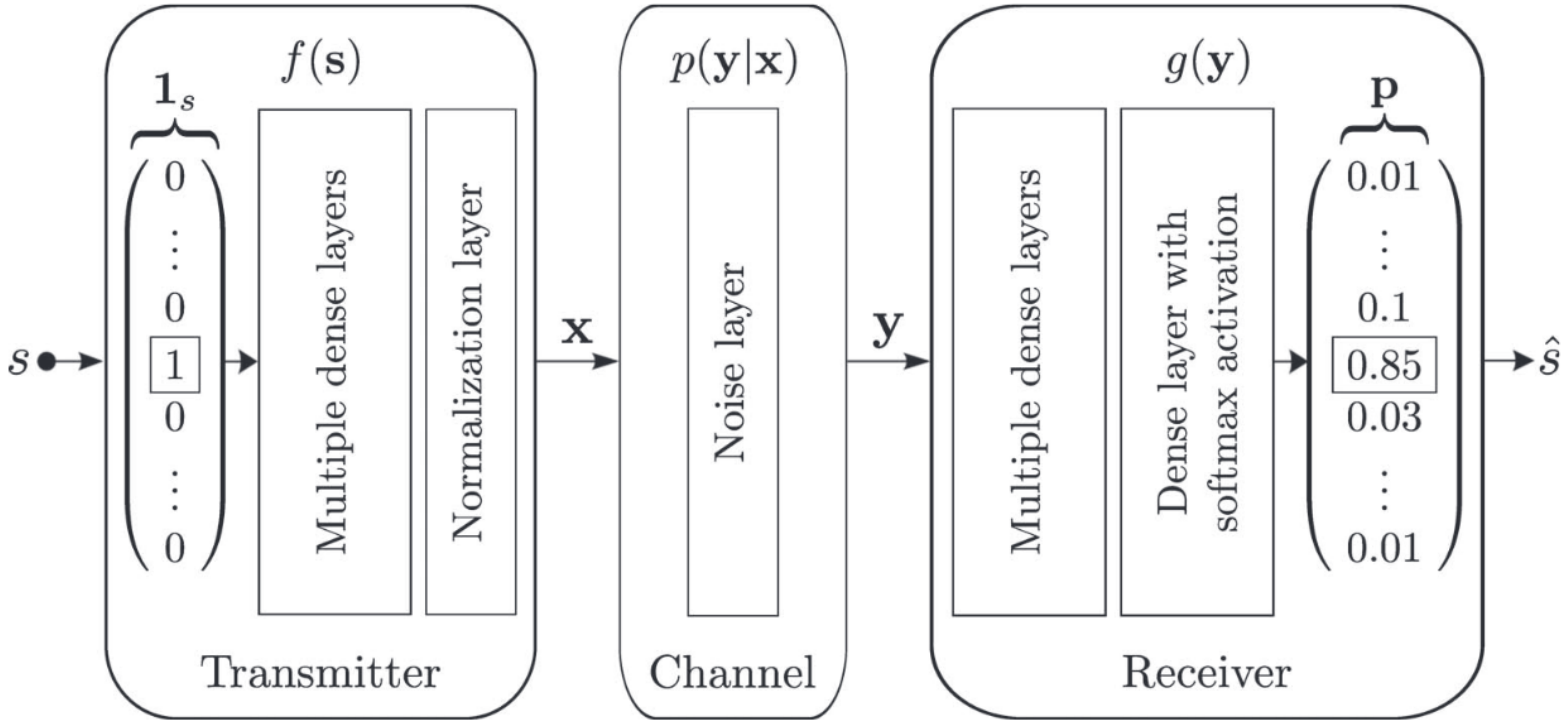
Conclusion

- Feedback Turbo Autoencoder
  - CNN based code for channels with feedback, inspired by turbo codes
  - Two-stage training
  - Outperforms existing codes in reliability
  - Achieves a block length gain

Survey of Other Directions to Invent Code

Discovering neural codes

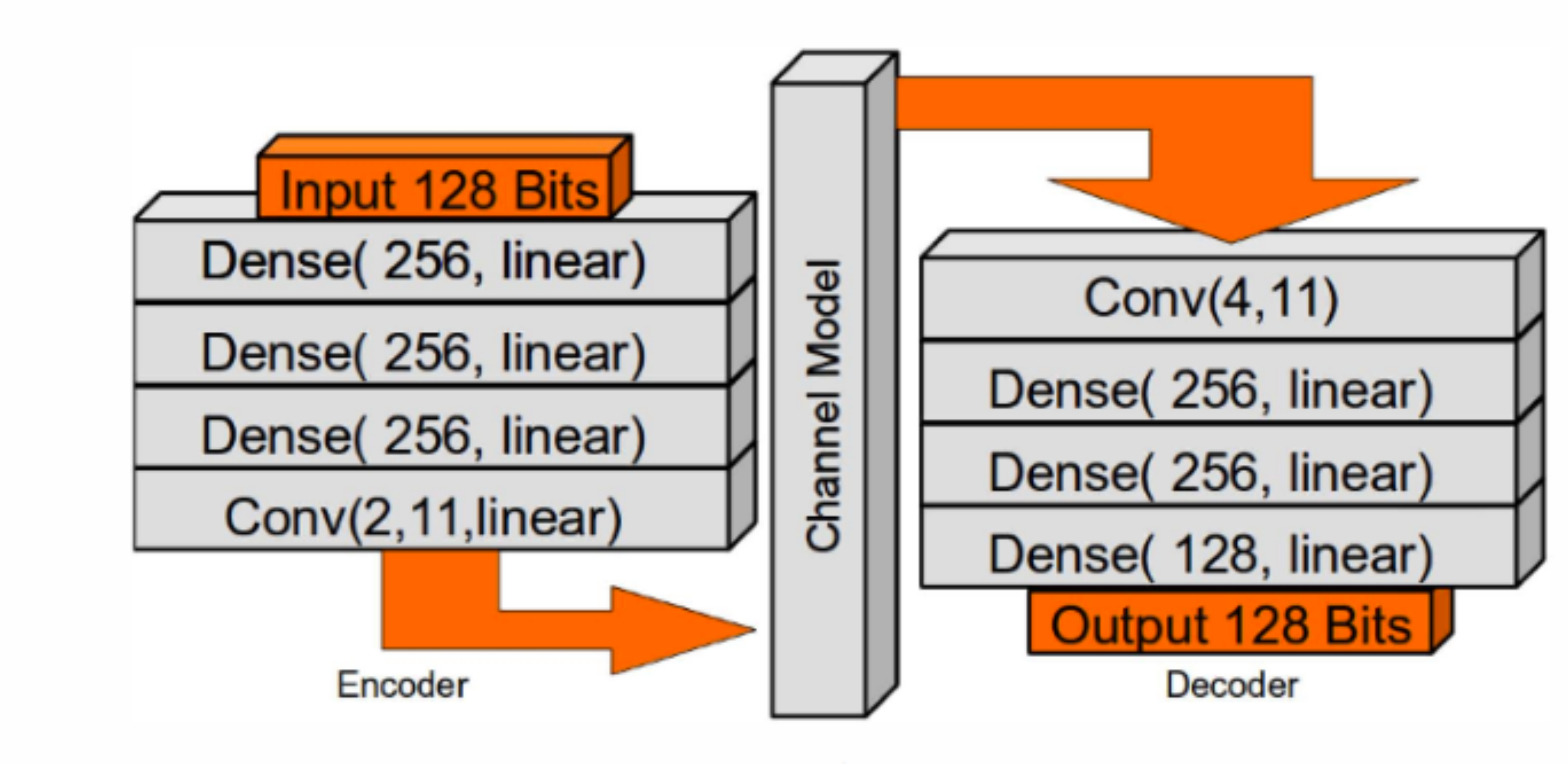
- AWGN
  - Neural (7,4) code: BER ~ BER of (7,4) Hamming code

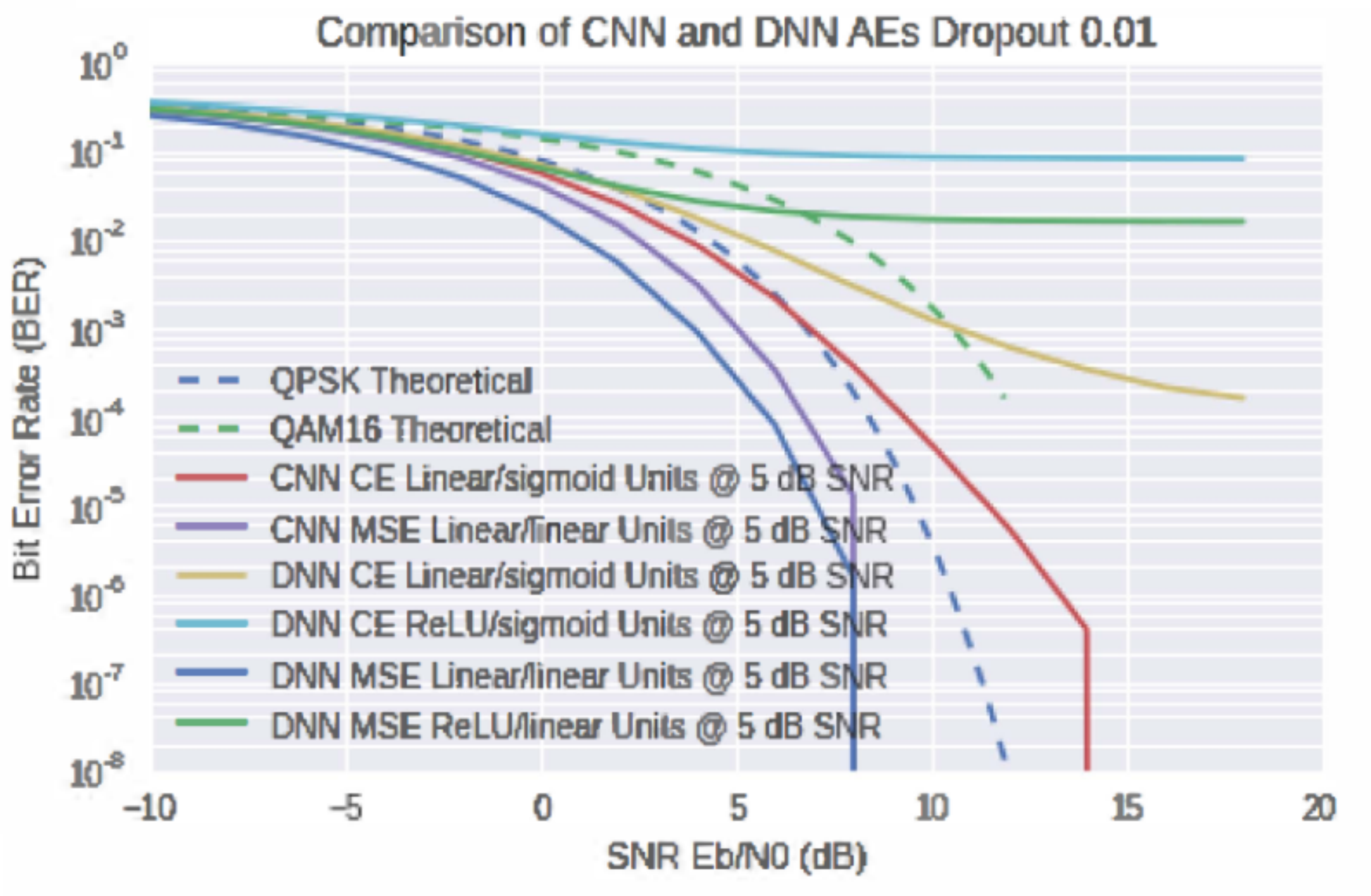


T. O'Shea, J. Hoydis, "***An Introduction to Deep Learning for the Physical Layer***" 2017

Discovering neural codes

- AWGN
  - Rate 1 (128 info. bits.) BER ~ 5dB better than QPSK

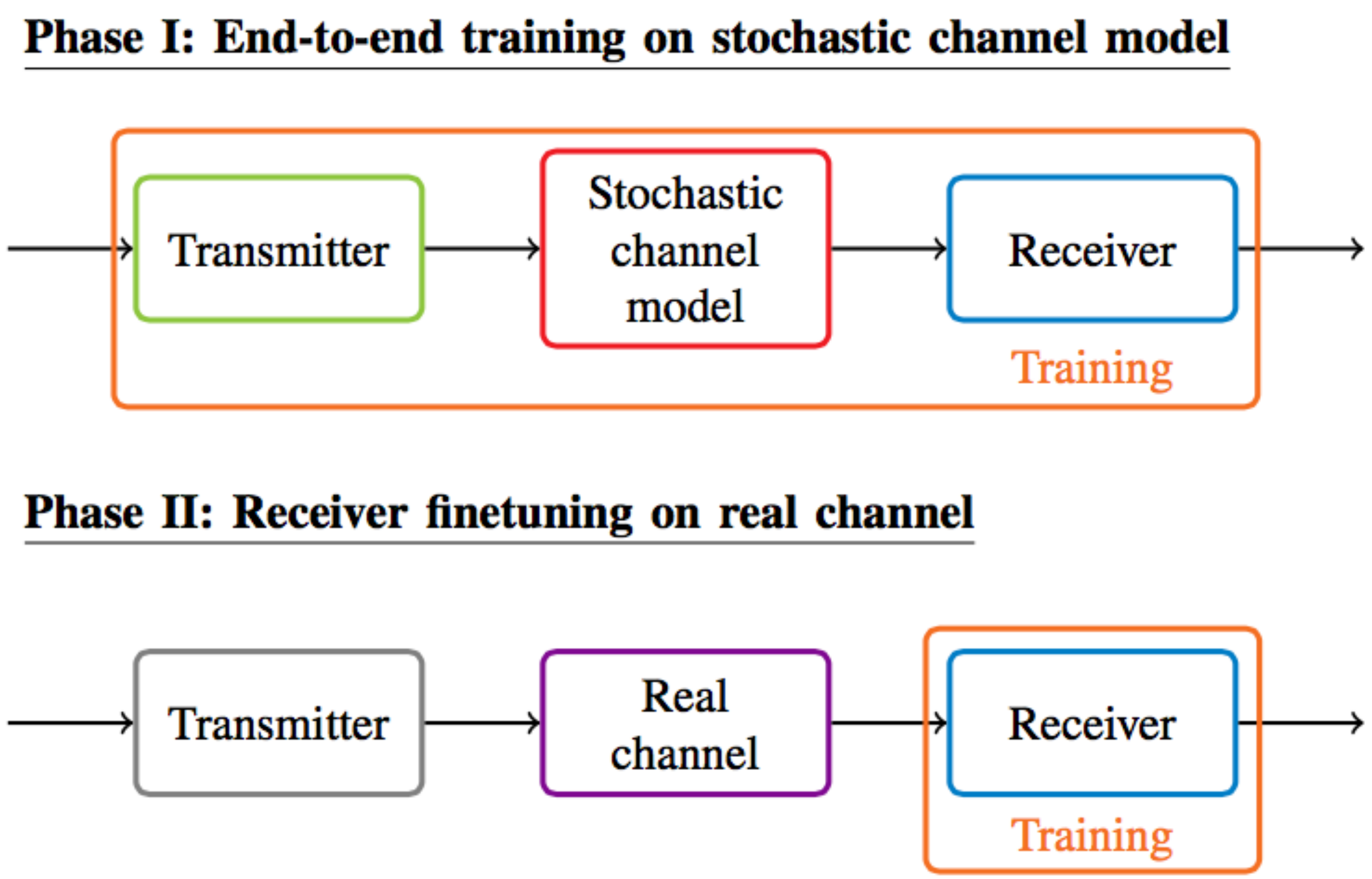


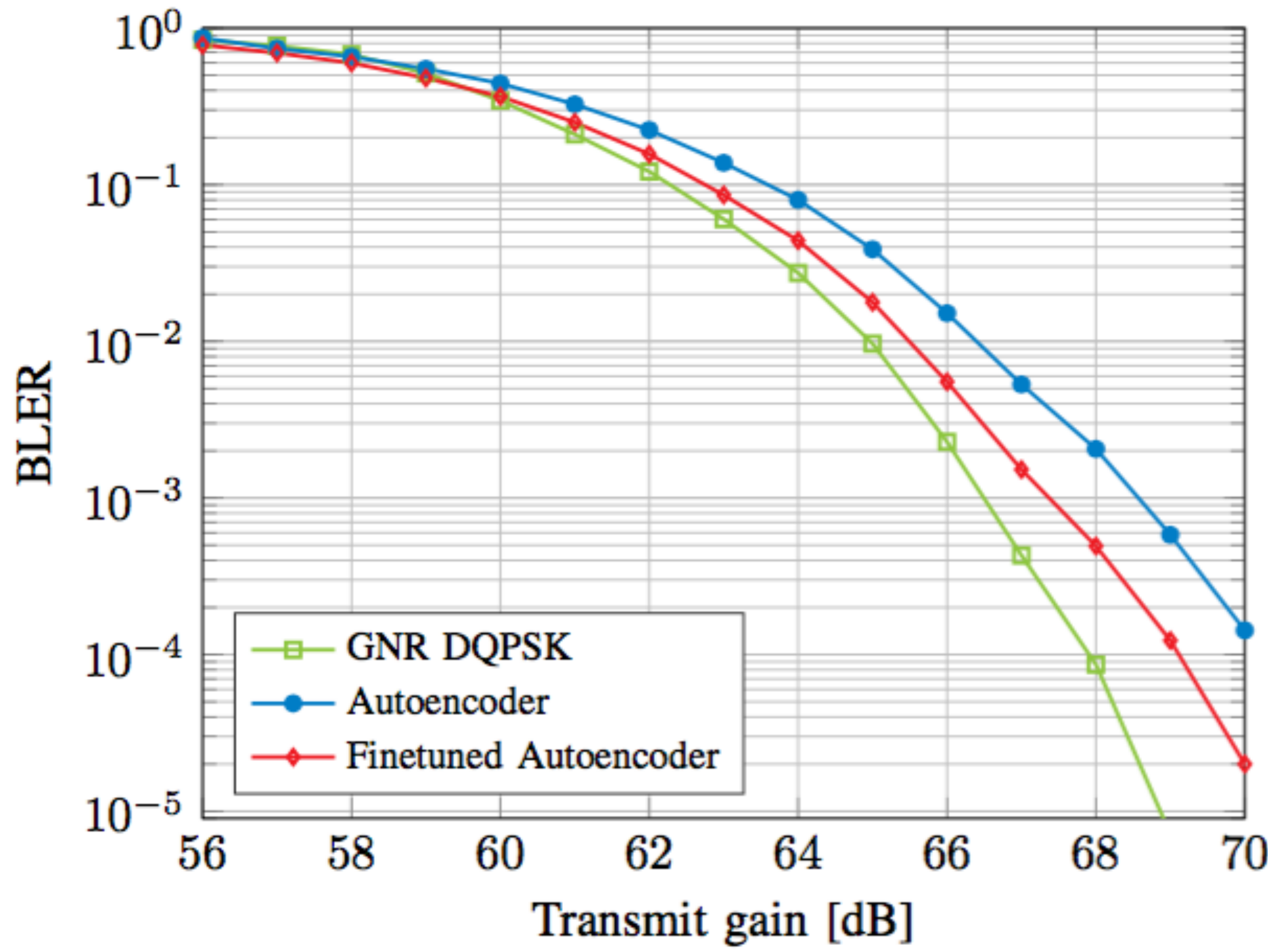


T. O'Shea, K. Karra, and T. C. Clancy, "***Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention***" 2016

Discovering neural codes

- No clean model: variation of AWGN channels





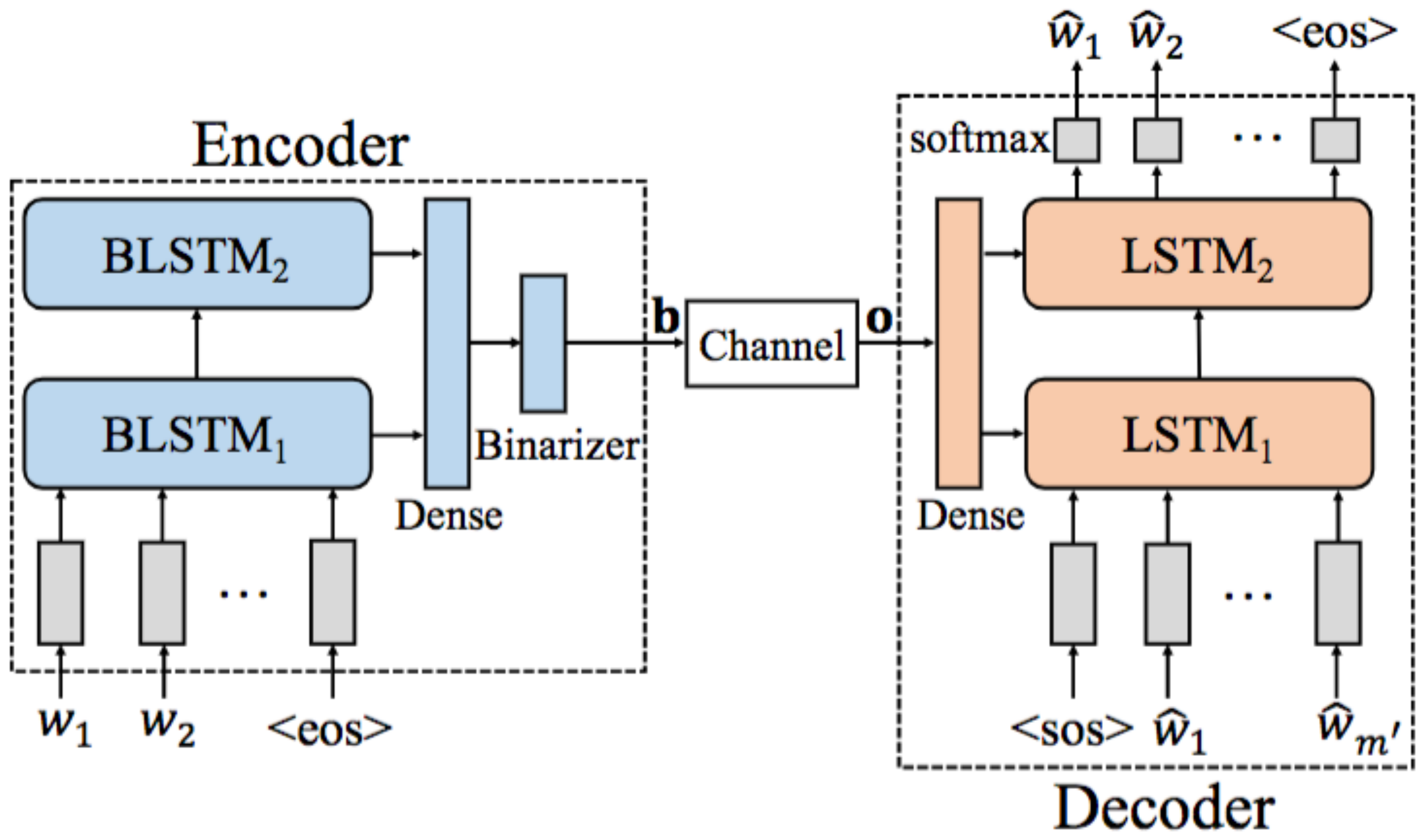
8 bits, 4 (complex) symbols under a wireless channel

S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "***Deep learning-based communication over the air***", 2017

Aoudia and Jakob Hoydis, "***End-to-End Learning of Communications Systems Without a Channel Model***" 2018

Discovering neural codes

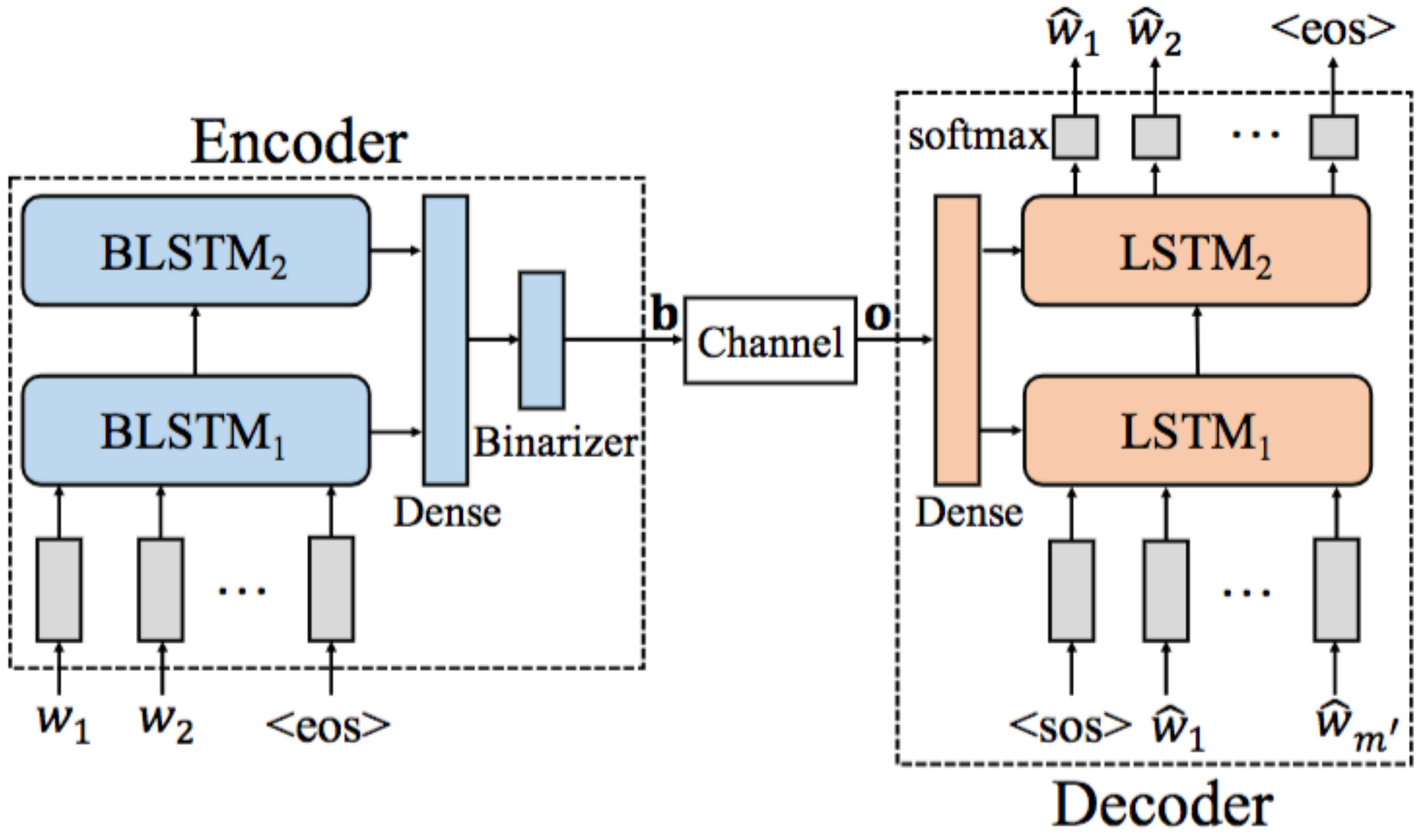
- Clean channel (erasure) / source is complicated (text)
  - Joint source channel coding



N. Farsad, M. Rao, and A. Goldsmith, "***Deep Learning for Joint Source-Channel Coding of Text***" 2018

Discovering neural codes

- Clean channel (erasure) / source is complicated (text)
  - Joint source channel coding
  - Improved reliability, evaluated by human



N. Farsad, M. Rao, and A. Goldsmith, "***Deep Learning for Joint Source-Channel Coding of Text***" 2018

Deep Learning Polar Design

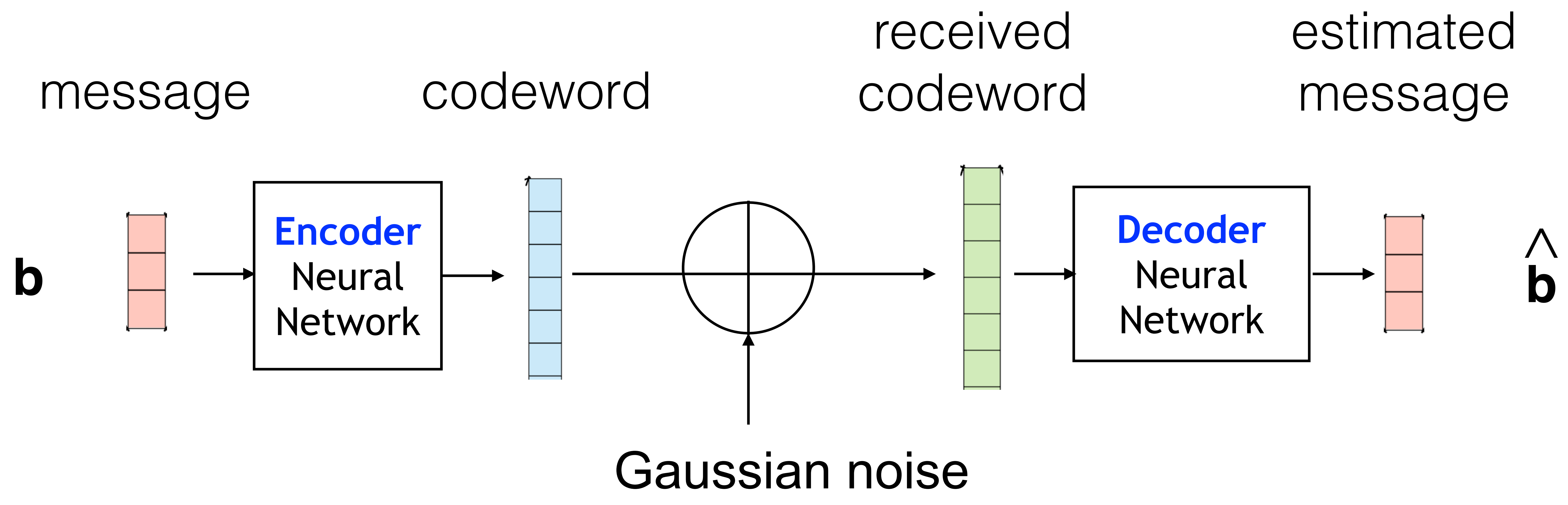
- Idea-1: Fix the code and learn which bits are good
  - Ebada, et al, “Deep Learning-based Polar Code Design”
- Idea-2: Use Reinforcement Learning to optimize the starting code for polar code
  - Huang, et al, “Reinforcement Learning for Nested Polar Code Construction”

Discovering neural codes

- Coded computation
  - J. Kosaian, K.V. Rashmi, and S. Venkataraman, "***Learning a Code: Machine Learning for Approximate Non-Linear Coded Computation***", 2018
- Orthogonal frequency-division multiplexing (OFDM)
  - A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. ten Brink, "***OFDM-Autoencoder for end-to-end learning of communications systems***", 2018
  - M. Kim, W. Lee, and D. H. Cho, ***"A novel PAPR reduction scheme for OFDM system based on deep learning"***, 2018
- Multiple-Input Multiple-Output (MIMO)
  - T. J. O'Shea, T. Erpek, and T. C. Clancy, ***"Physical layer deep learning of encodings for the MIMO fading channel"***, *2017*

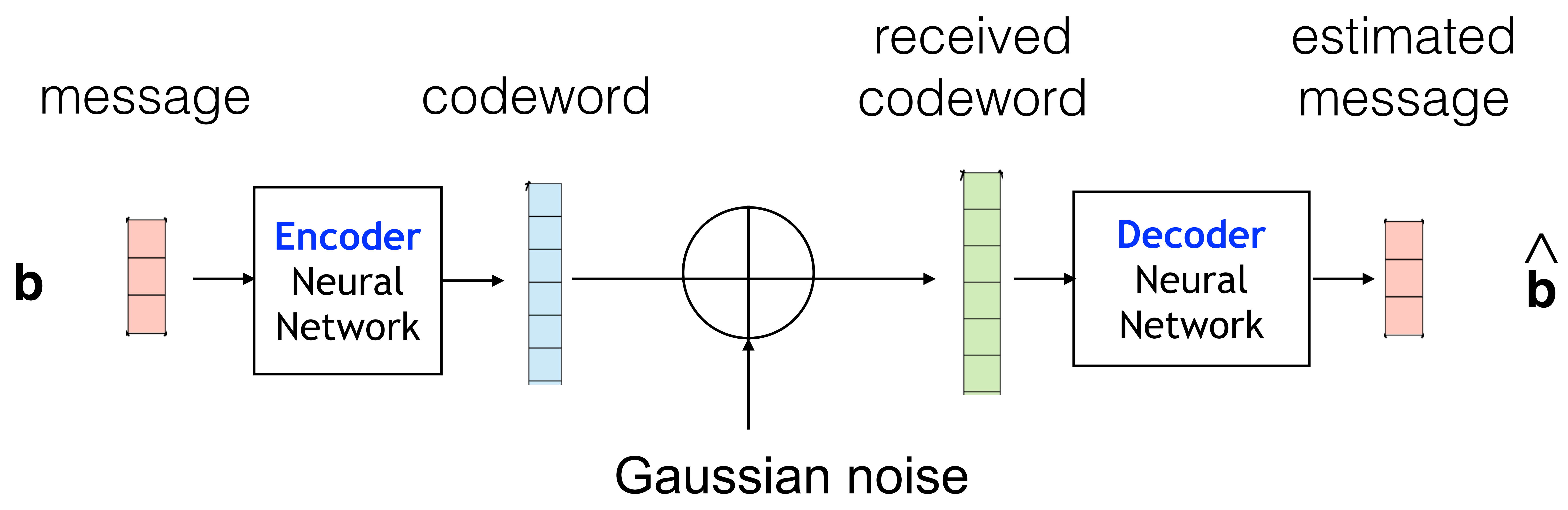
Open problems

- Canonical and benchmark : AWGN



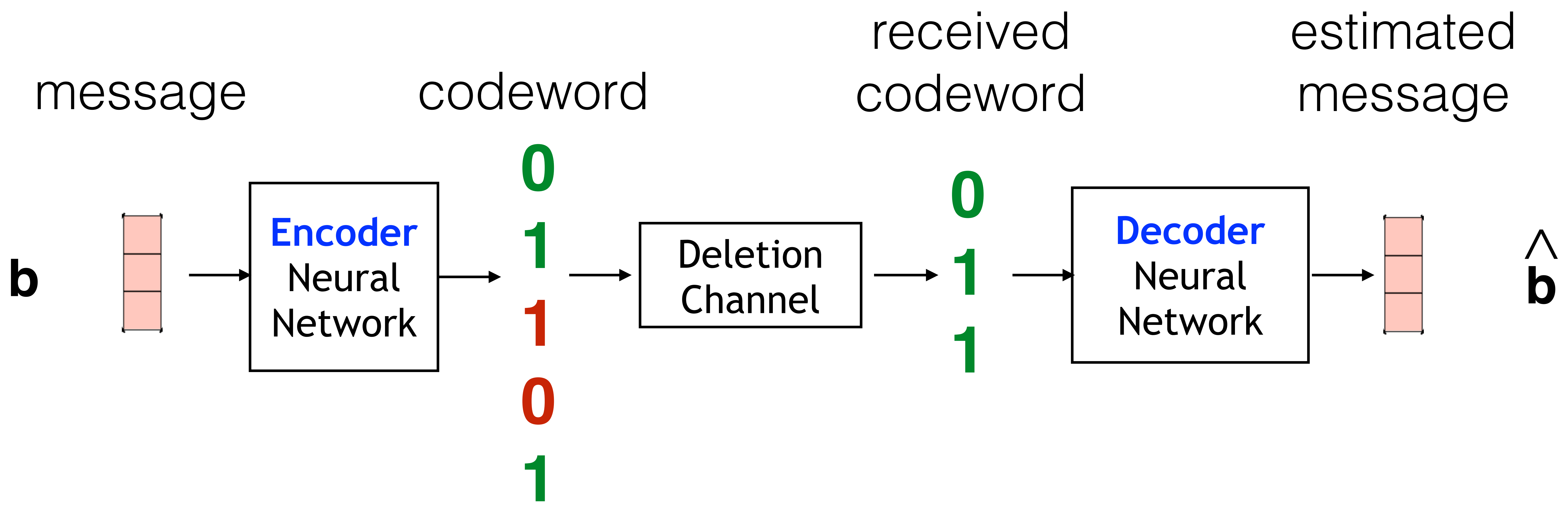
Open problems - 1

- Canonical and benchmark : AWGN
  - Challenge 1. neural code that has a long range memory
  - Challenge 2. Error floor at high SNR



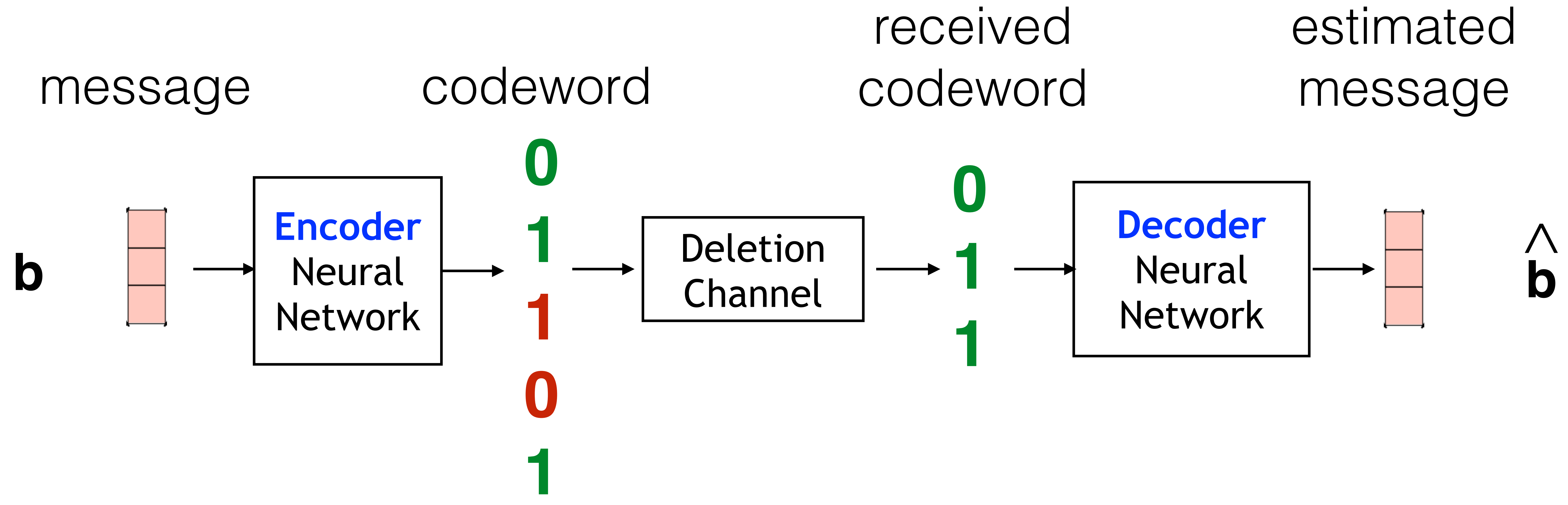
Open problems - 2

- Channels with no good codes: deletion channel
  - Practical (e.g. lack of synchronization, DNA sequencing)



Open problems - 2

- Channels with no good codes: deletion channel
  - Practical (e.g. lack of synchronization, DNA sequencing)
  - Optimal codes known only if deletion probability v. small
  - No practical code exists; capacity unknown in general



Open problems - 3

- Network information theory setting
    - Relay, interference, Coordinated Multipoint (CoMP)

Applications of Deep Learning to Information Theory

Applications of Deep Learning to Info Theory

- Compressed sensing
  - DeepCodec: Adaptive Sensing and Recovery via Deep Convolutional Neural Networks
- Mutual Information estimation
  - MINE - Mutual Information Neural Estimation
  - CCMI - Classifier-based mutual information estimation
- Low Rank Matrix Decomposition
  - Indyk et al, “Learning-Based Low-Rank Approximations”
- Coded computation

Reference

- Learning a decoder

  - H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, P. Viswanath, "Communication Algorithms via Deep Learning," ICLR, 2018
  - Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, P. Viswanath, "DeepTurbo: Deep Turbo Decoder," SPAWC, 2019
  - Y. Jiang, H. Kim, H. Asnani, S. Kannan, "MIND: Model Independent Neural Decoder," SPAWC, 2019
  - Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "LEARN Codes: Inventing low-latency codes via recurrent neural networks," ICC 2019

Reference

- Learning a modulation
  - Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Joint channel coding and modulation via deep learning," SPAWC 2020

Reference

- Learning an encoder

  - H. Kim, Y. Jiang, S. Kannan, S. Oh, P. Viswanath, "Deepcode: Feedback Codes via Deep Learning" NeurIPS, 2018

  - Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, P. Viswanath, "Turbo Autoencoder: Deep learning based channel code for point-to-point communication channels," NeurIPS, 2019

  - Y. Jiang, H. Kim, H. Asnani, S. Oh, S. Kannan, P. Viswanath, "Feedback Turbo Encoder," ICASSP, 2020

Source codes

- Communication Algorithms via Deep Learning
  International Conference on Learning Representations (ICLR), Vancouver, April 2018
  by H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, P. Viswanath
  Code: https://github.com/yihanjiang/Sequential-RNN-Decoder

- Deepcode: Feedback Codes via Deep Learning
  IEEE Journal on Selected Areas in Information Theory, 2020
  Conference on Neural Information Processing Systems (NeurIPS), Montreal, December 2018
  by H. Kim, Y. Jiang, S. Kannan, S. Oh, P. Viswanath
  Code (Keras): https://github.com/hyejikim1/Deepcode
  Code (PyTorch): https://github.com/yihanjiang/feedback_code

- Turbo Autoencoder: Deep learning based channel code for point-to-point communication channels
  Conference on Neural Information Processing Systems (NeurIPS), Vancouver, December 2019
  by Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, P. Viswanath
  Code: https://github.com/yihanjiang/turboae