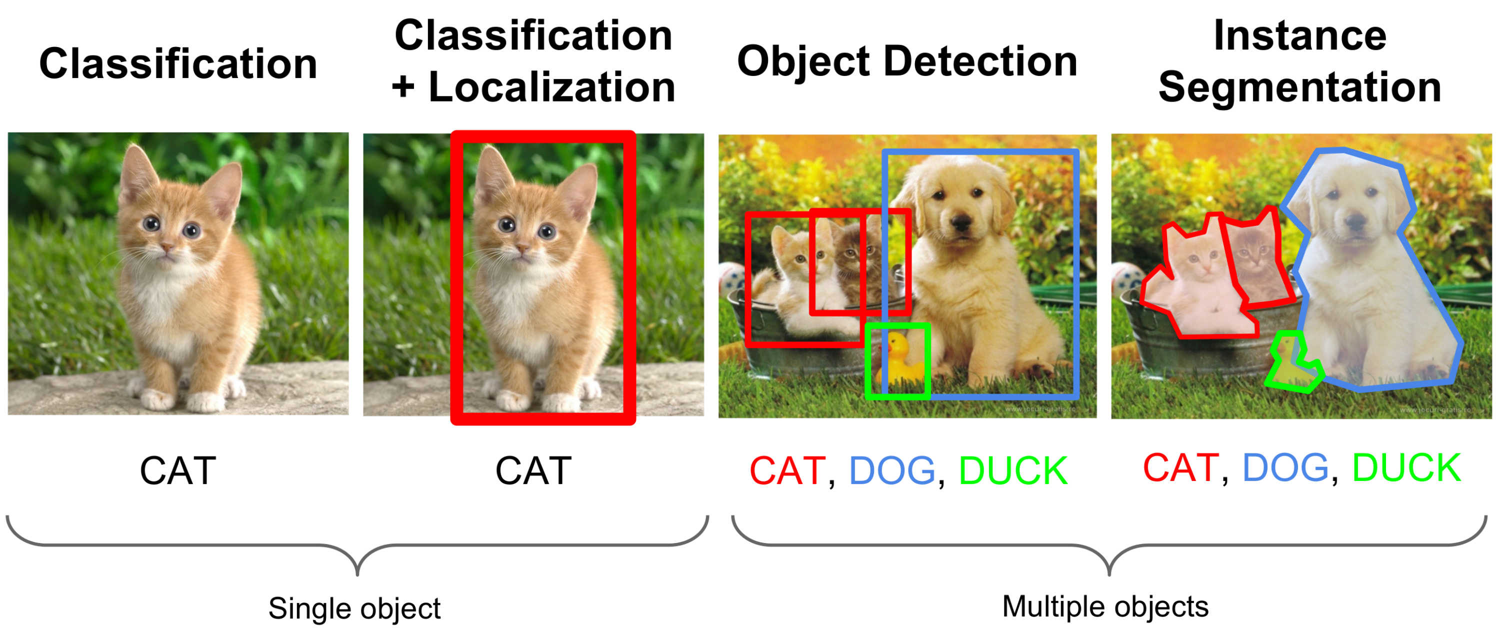
Communication Algorithms via Deep Learning

Pramod Viswanath

University of Illinois

Deep learning is part of daily life



Computer Vision

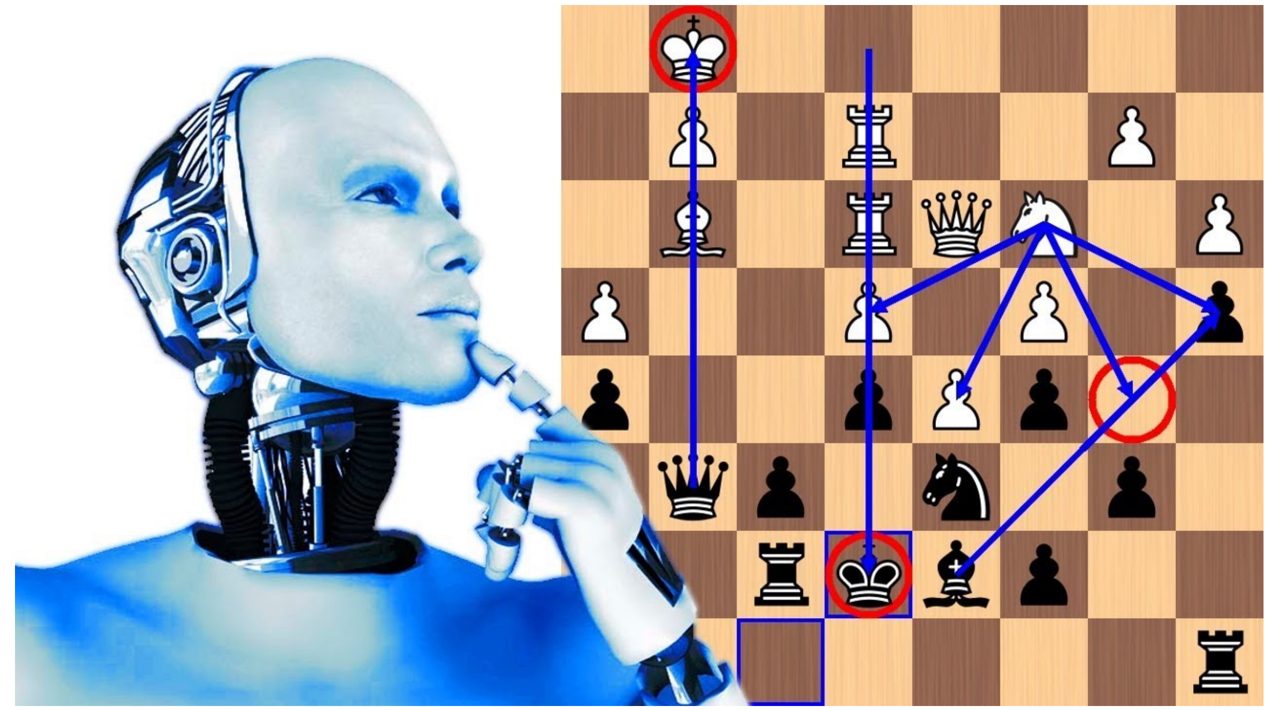
natural language processing

Model complexity

- Data (image, languages): **hard to model**
- Inference problems : **hard to model**
- Evaluation metrics: empirical, **human-validated**
- **Deep learning learns efficient models**

Algorithmic complexity

- Models are simple
  - example: chess



  - unlimited training data

  - clear performance metrics

- Challenge: space of algorithms very large

- Deep learning learns sophisticated algorithms (Alphazero)

Communication Algorithms

- Problems of great scientific/engineering relevance
  - mathematical models
  - precise performance metrics

Two Goals

- New (deep learning) tools for classical problems
  - new state of the art
  - inherent practical value
- Insight into deep learning methods
  - no overfitting
  - interpretability

One Lens

- **Scalability**
  - train on small settings
  - test on much larger settings (100x)

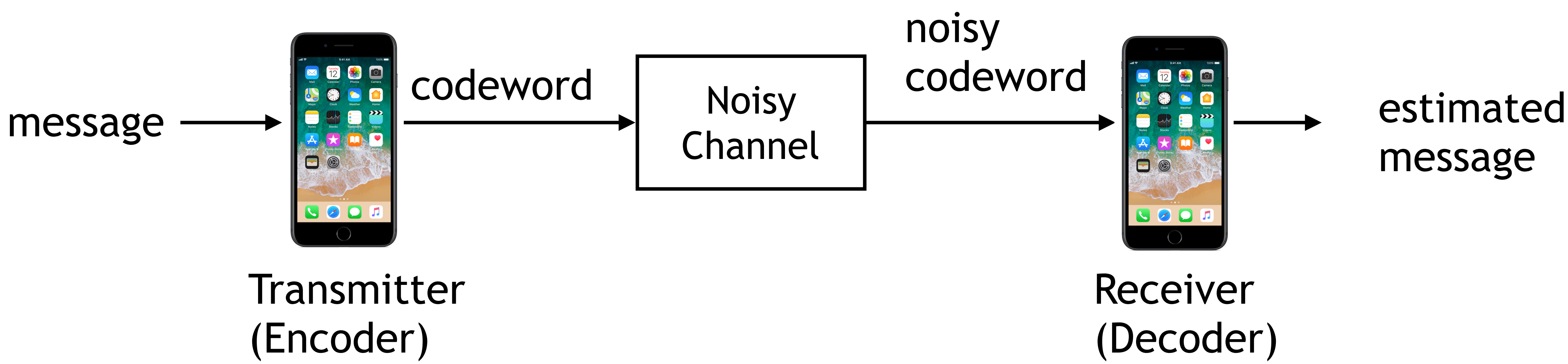
Two Deep Learning Components

- **Recurrent neural networks**
  - in-built capability for recursion

- **Gated neural networks**
  - attention, GRU, LSTM

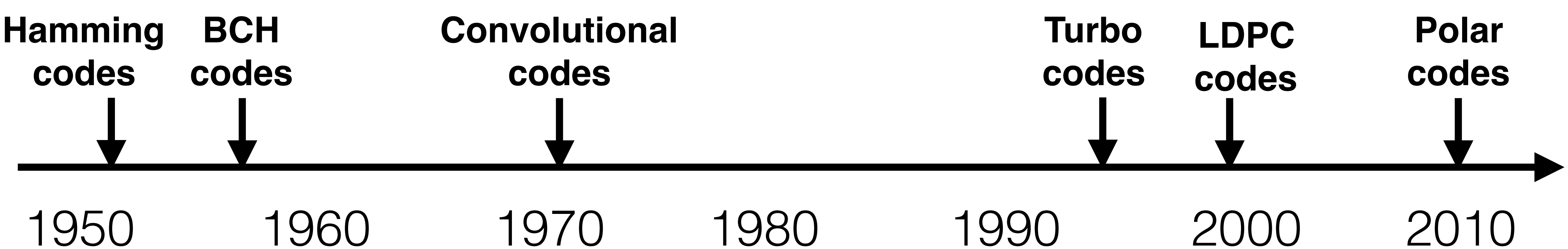
Communication

- Models are simple
- Performance metrics are clear (e.g. Bit Error Rate)
- Designing a robust encoder/decoder is critical
- Challenge : space of encoder/decoder mappings very large



Design of codes

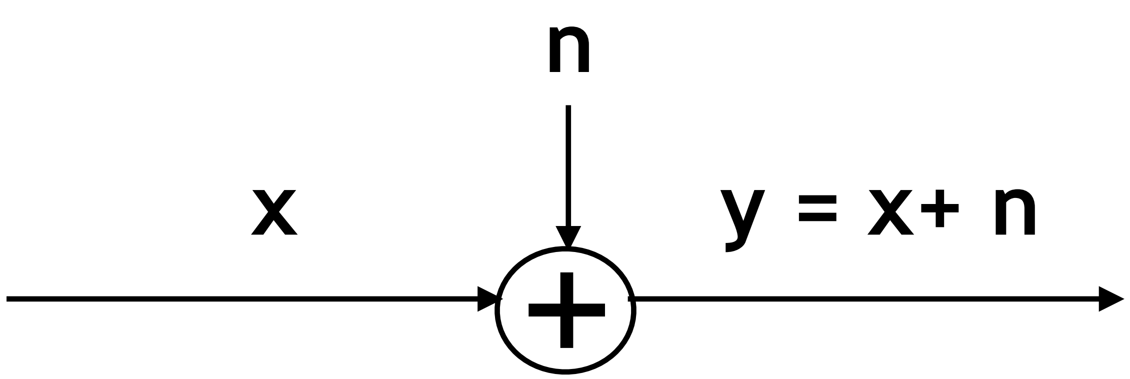
- Technical Communities
- Eureka moments



- Huge practical impact

Classical Model

- Additive White Gaussian Noise (AWGN) channels



- **Very good codes**
  - turbo, LDPC, polar codes

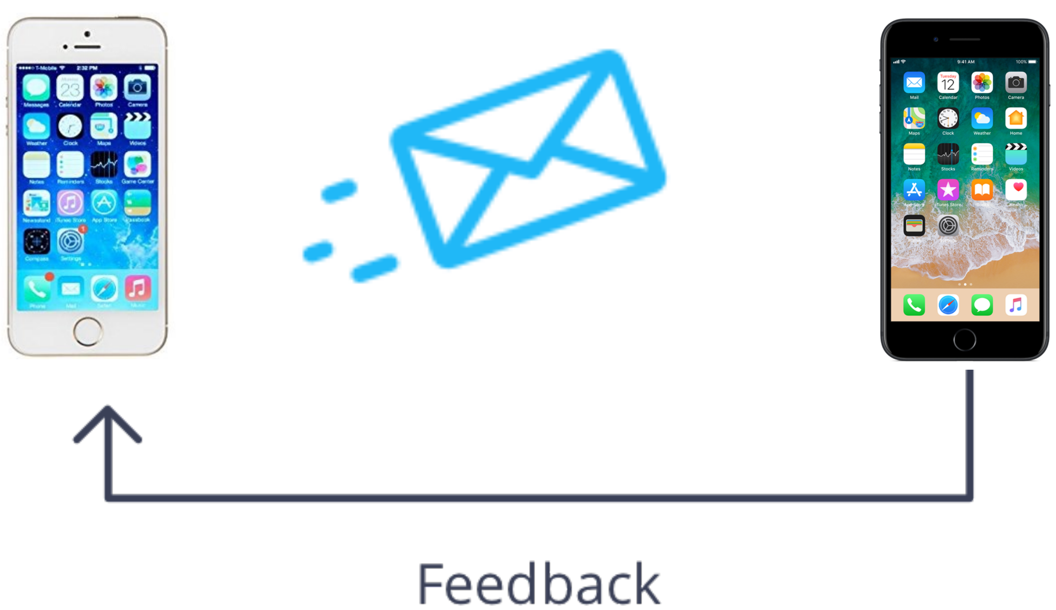
Central goal

Automate the search for codes and decoders via deep learning

But there are many challenges. Fundamentally different from other machine learning problems.

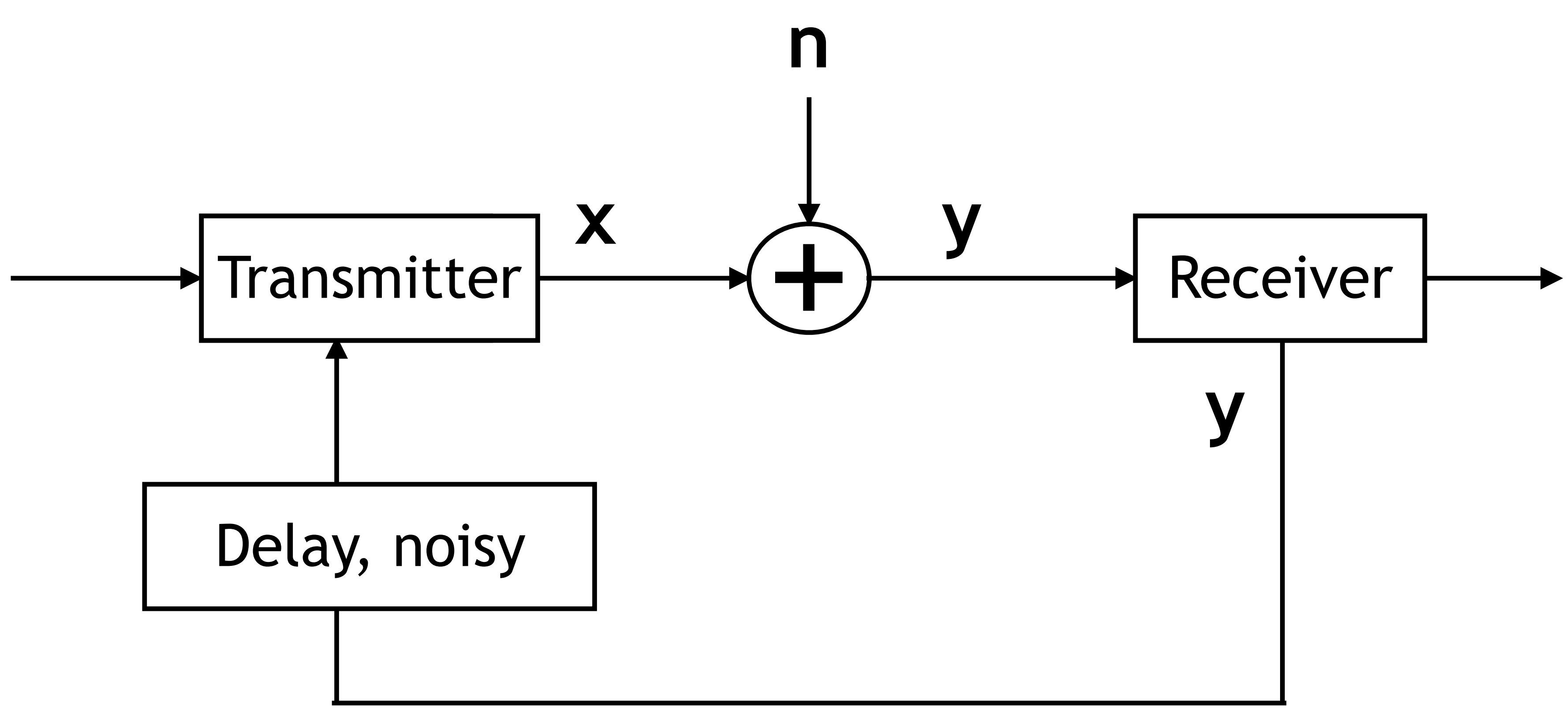
Channel with Feedback

- Learning the **encoder**
- Learning the **decoder**



AWGN channels with feedback

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



Literature

- ARQ
  - practical, minimal feedback

- Noiseless output feedback
  - Improved reliability
  - Coding schemes
    - Schalkwijk-Kailath, '66

Literature

- Noisy feedback
  - Existing schemes perform poorly
  - Negative results
  - Linear codes very bad (Kim-Lapidoth-Weissman, '07)

- Wide open

Focus of this talk

- AWGN channels with noisy feedback

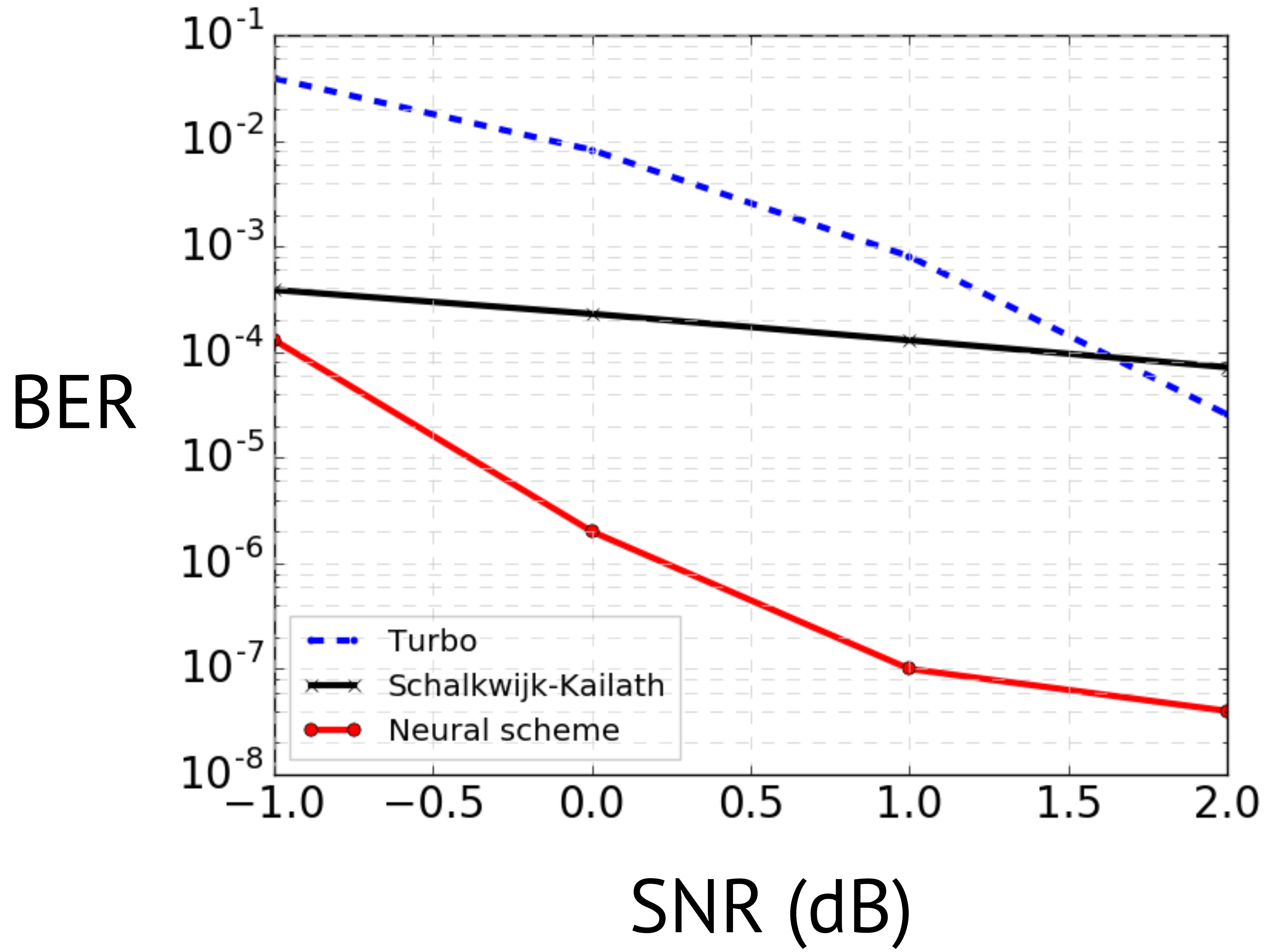
- Challenge:

  How to combine noisy feedback and message causally?

- Model encoder and decoder as neural networks and train

Quick Summary: 100x Better Reliability

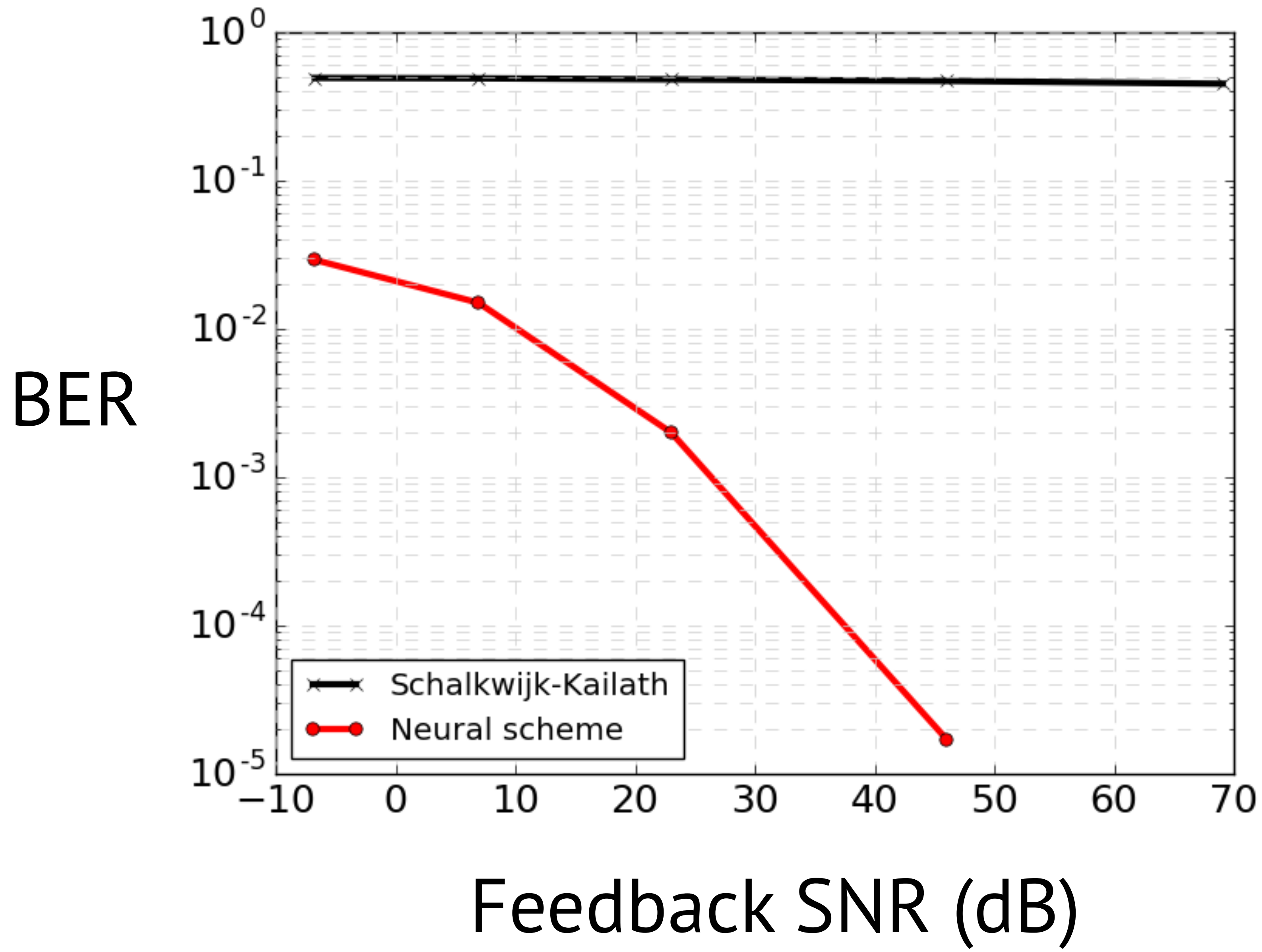
- Feedback with machine precision



(Rate 1/3, 50 bits)

Robustness

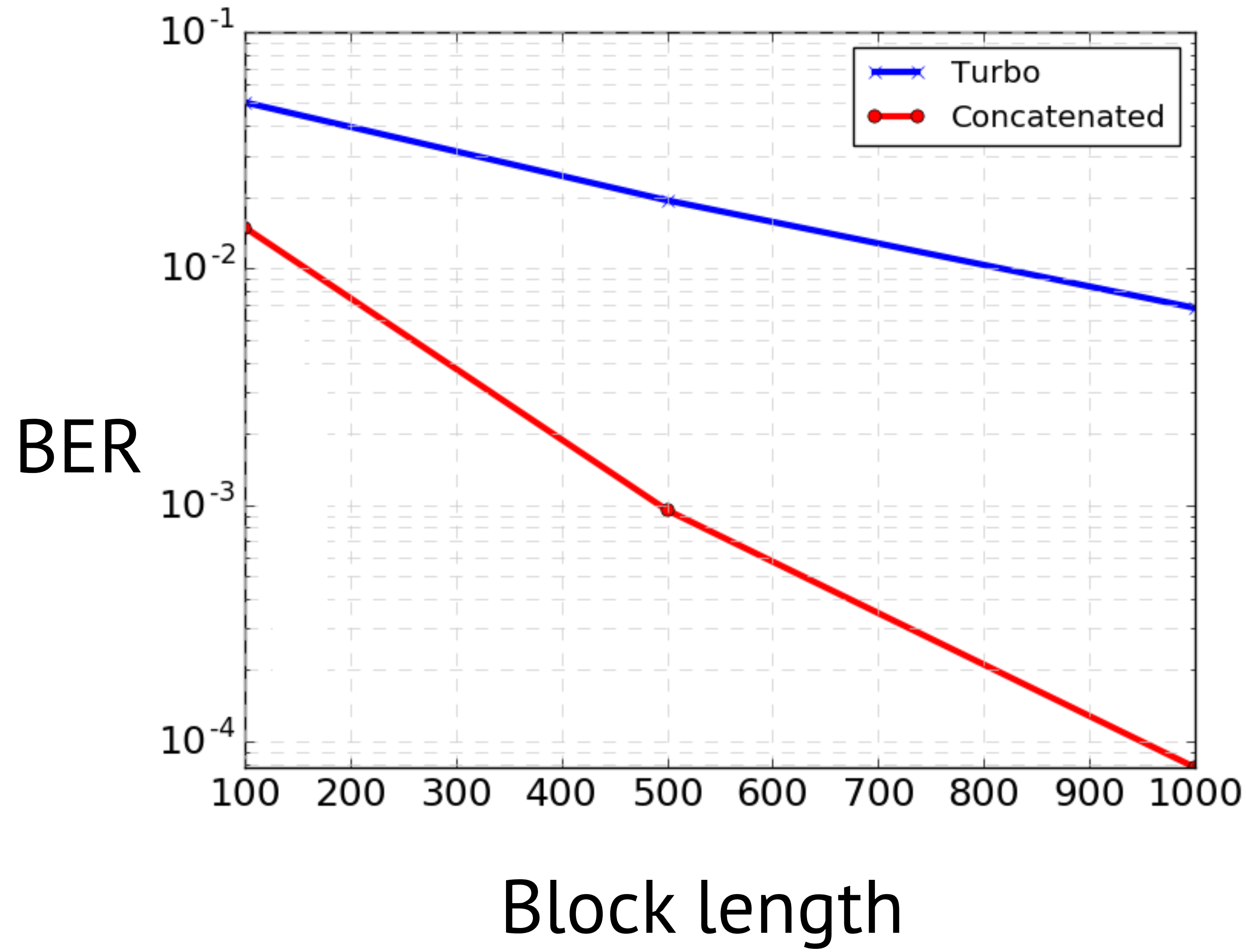
- varying noise in the feedback



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

Improved error exponents

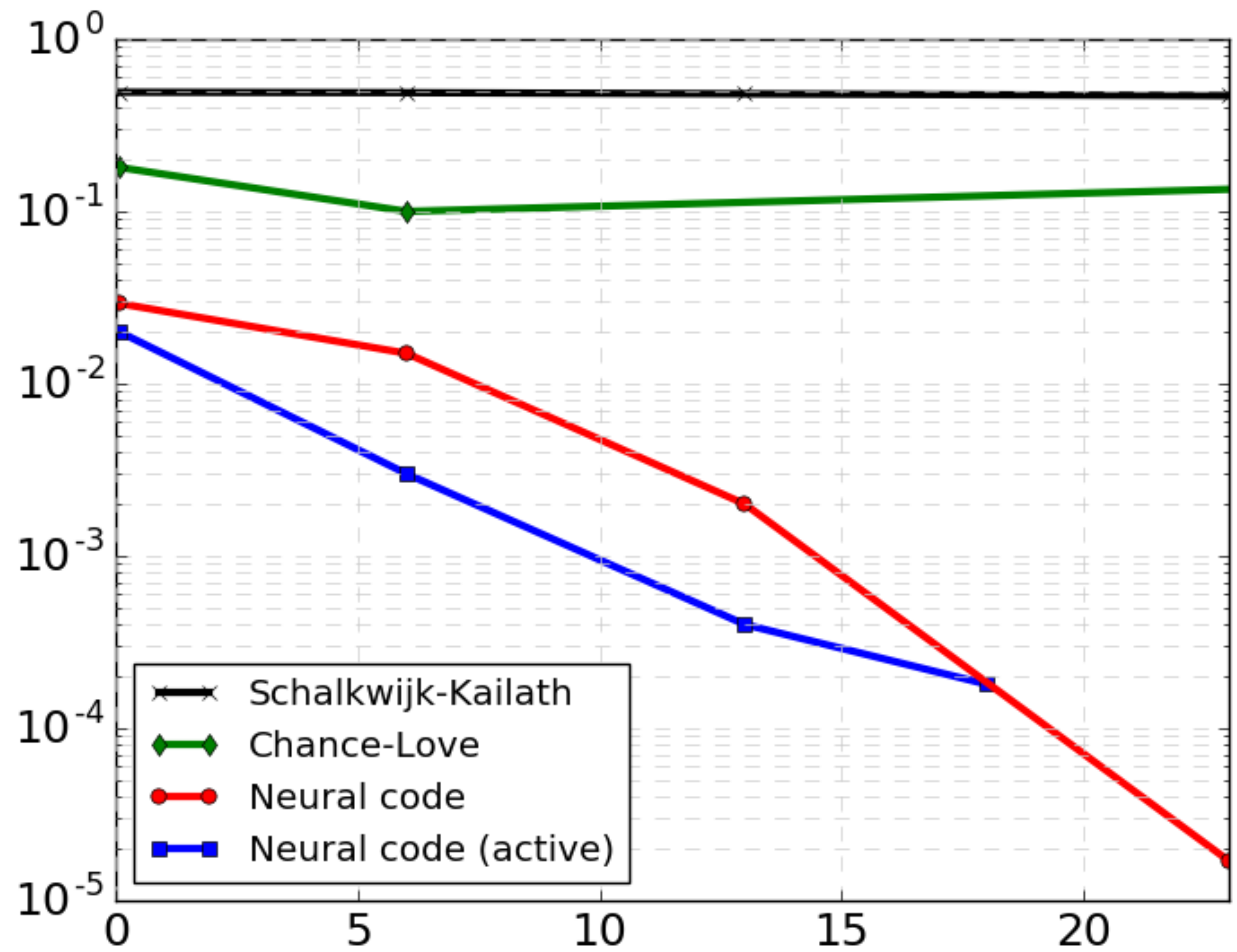
- BER decays faster



Coded Feedback

- BER decays even faster

BER



feedback SNR

Practical Considerations

- **Delay in feedback**
  - interleaving; can handle large delays
- Opportunistic use of feedback capability
  - cellular and WiFi environments
- Composes with ARQ

Neural feedback code

Architectural innovations

Ideas from communications

Computationally simple

Other open problems

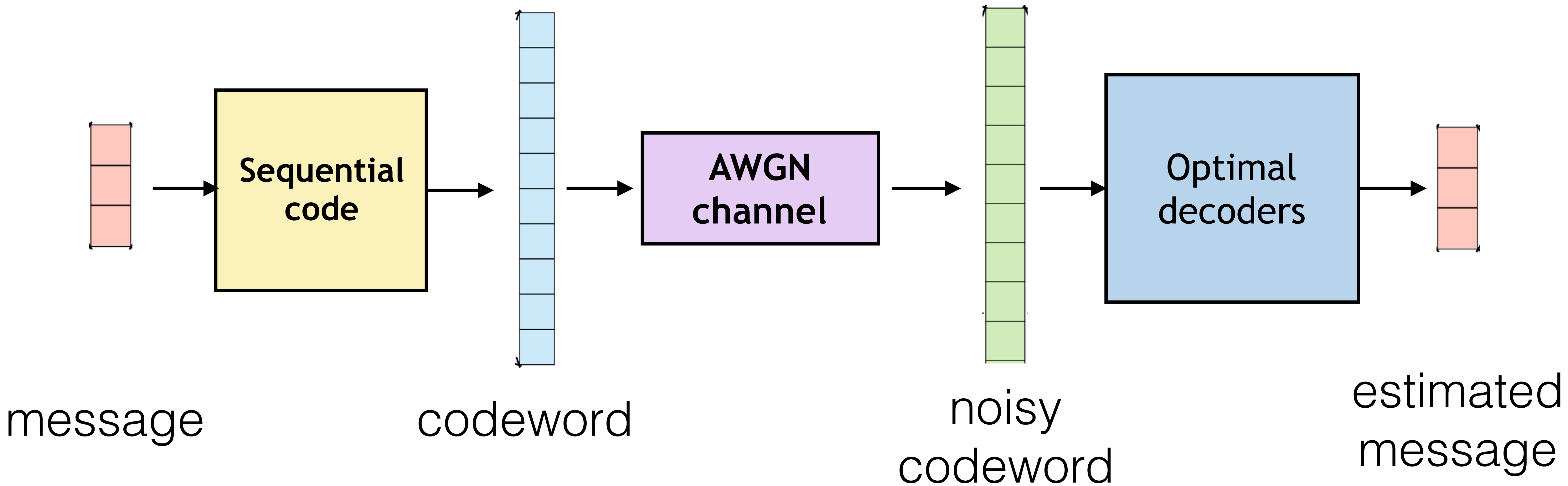
- Encoder is fixed (e.g. standardization)
- Practical channels are not always AWGN
- How to adapt the decoder to practical channels?

Sequential codes

- Convolutional codes, turbo codes
  - 3G/4G mobile communications (e.g., in UMTS and LTE)
  - (Deep space) satellite communications
- Provably achieve performance close to fundamental limit
- Natural recurrent structure aligned with RNN

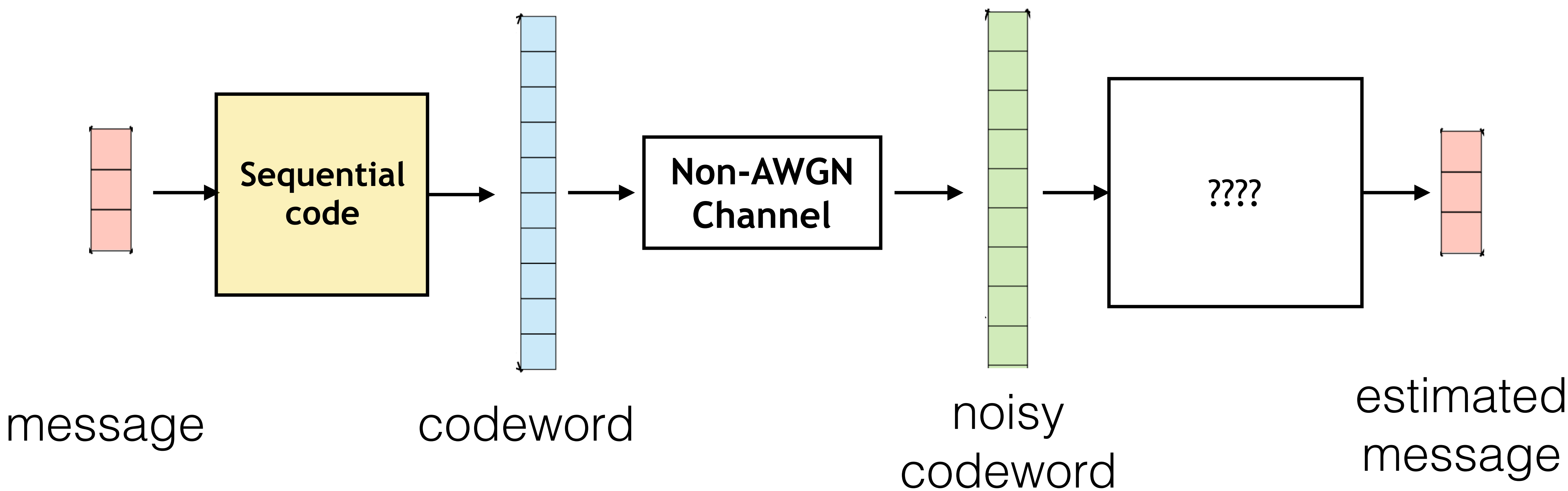
Sequential codes under AWGN

- Optimal decoders under AWGN
  - e.g. Viterbi, BCJR decoder for convolutional codes



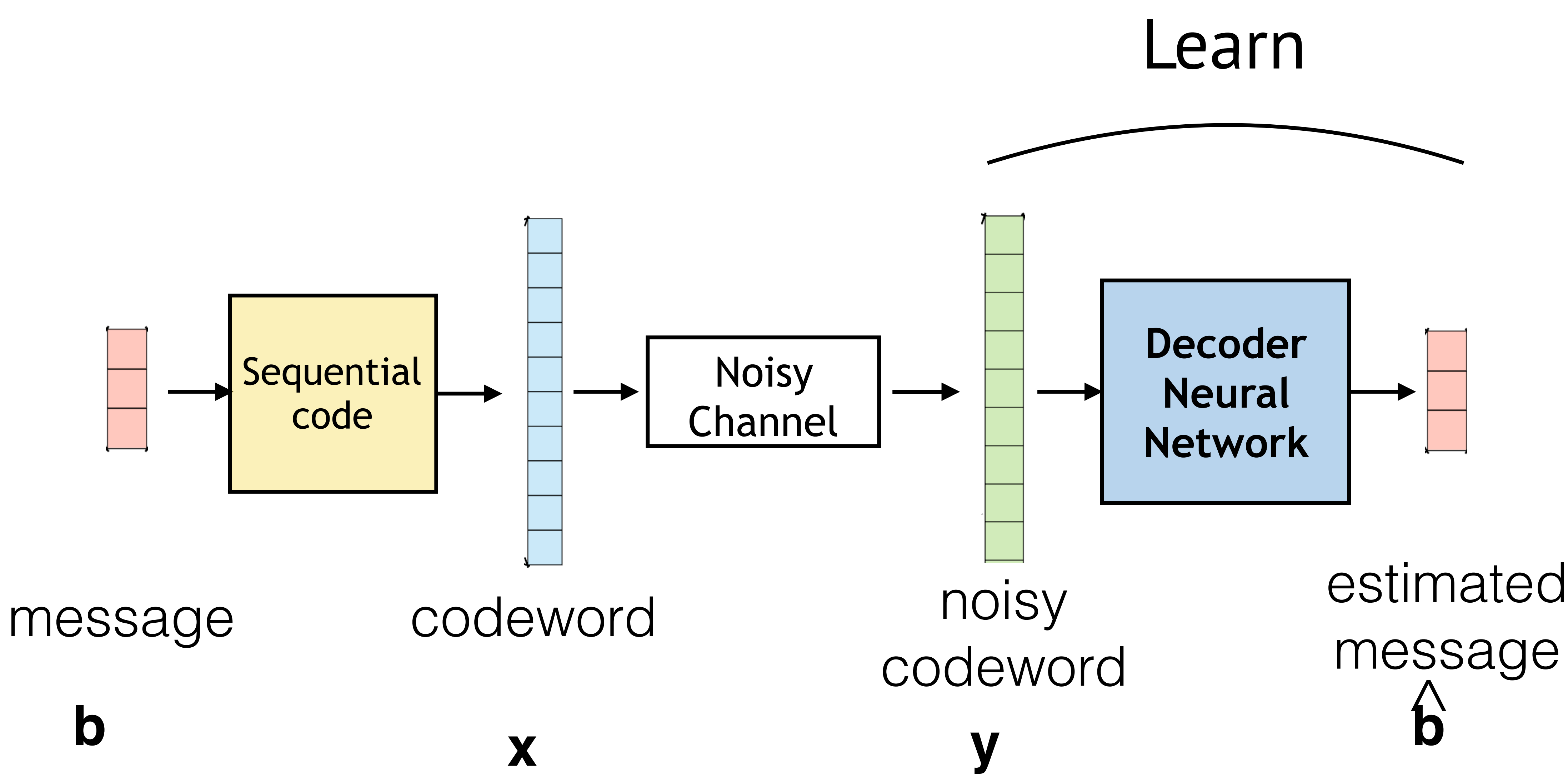
Non-AWGN channel

Bursty noise



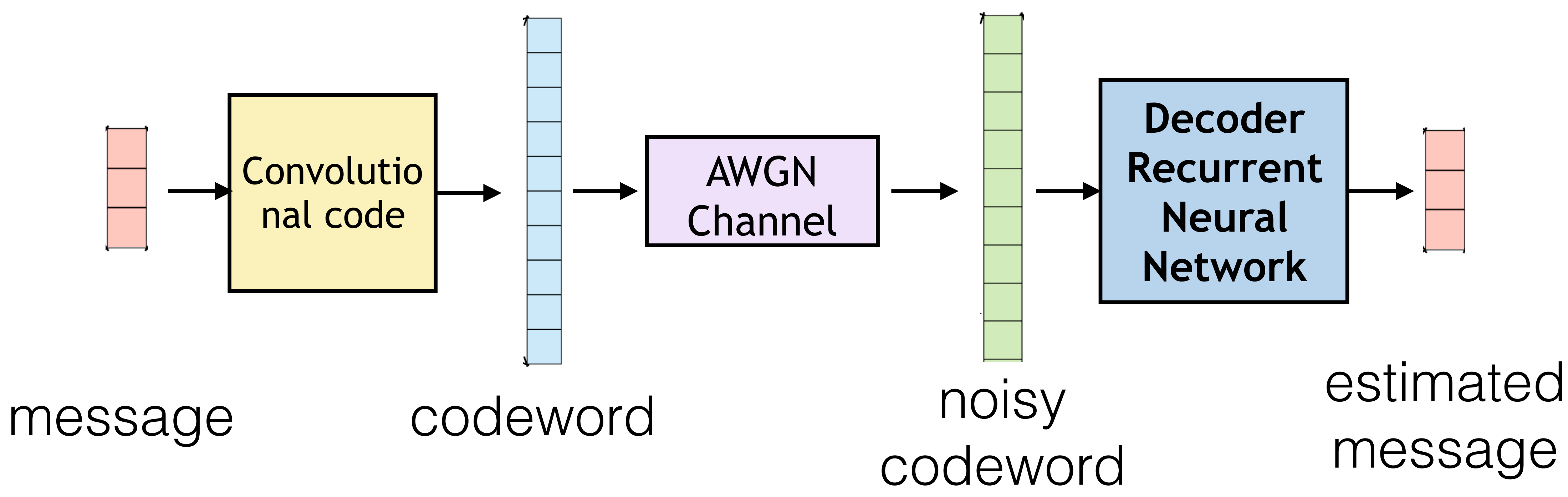
Neural decoder

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



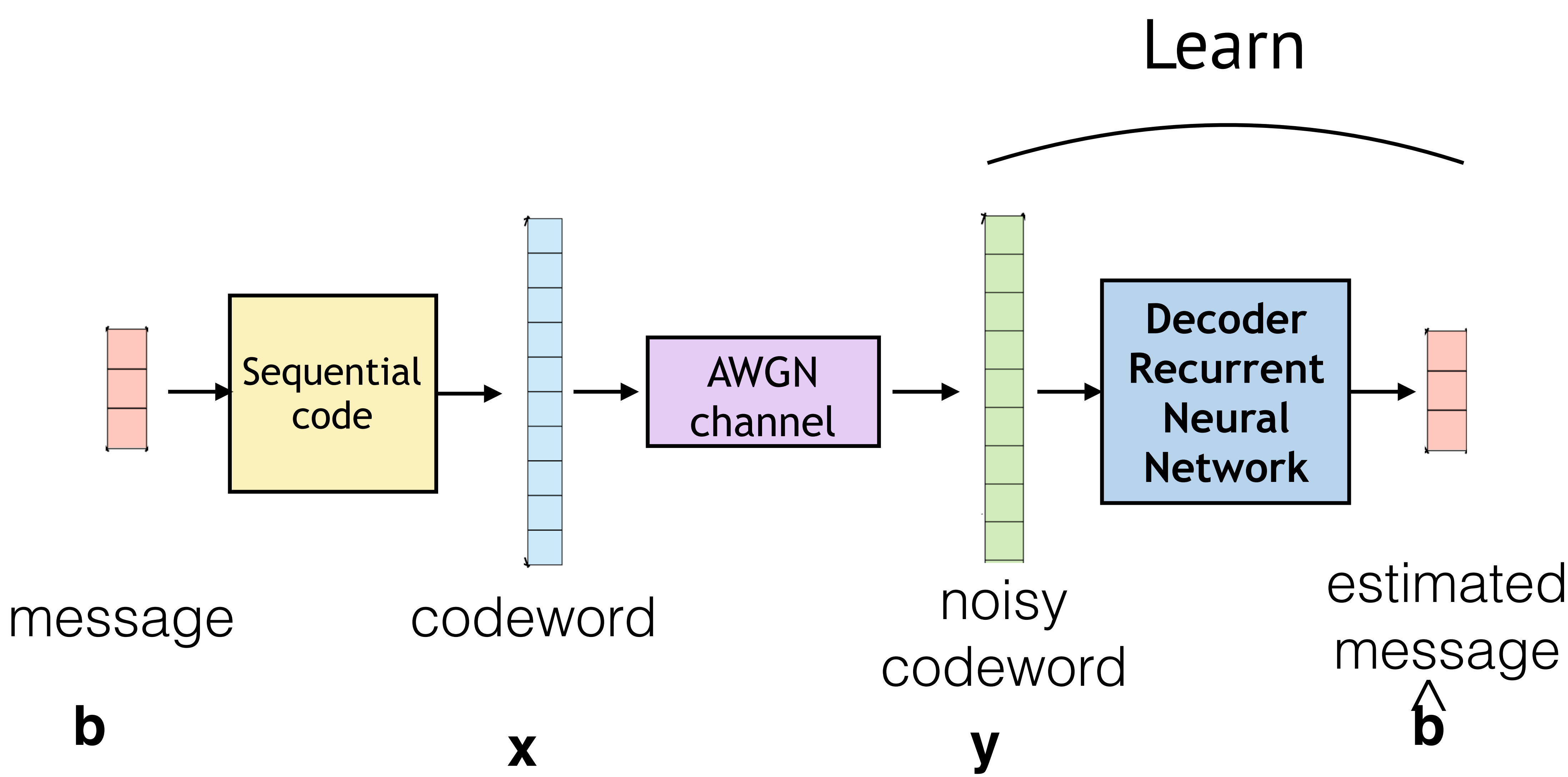
Neural decoder under AWGN

- Model decoder as a Recurrent Neural Network



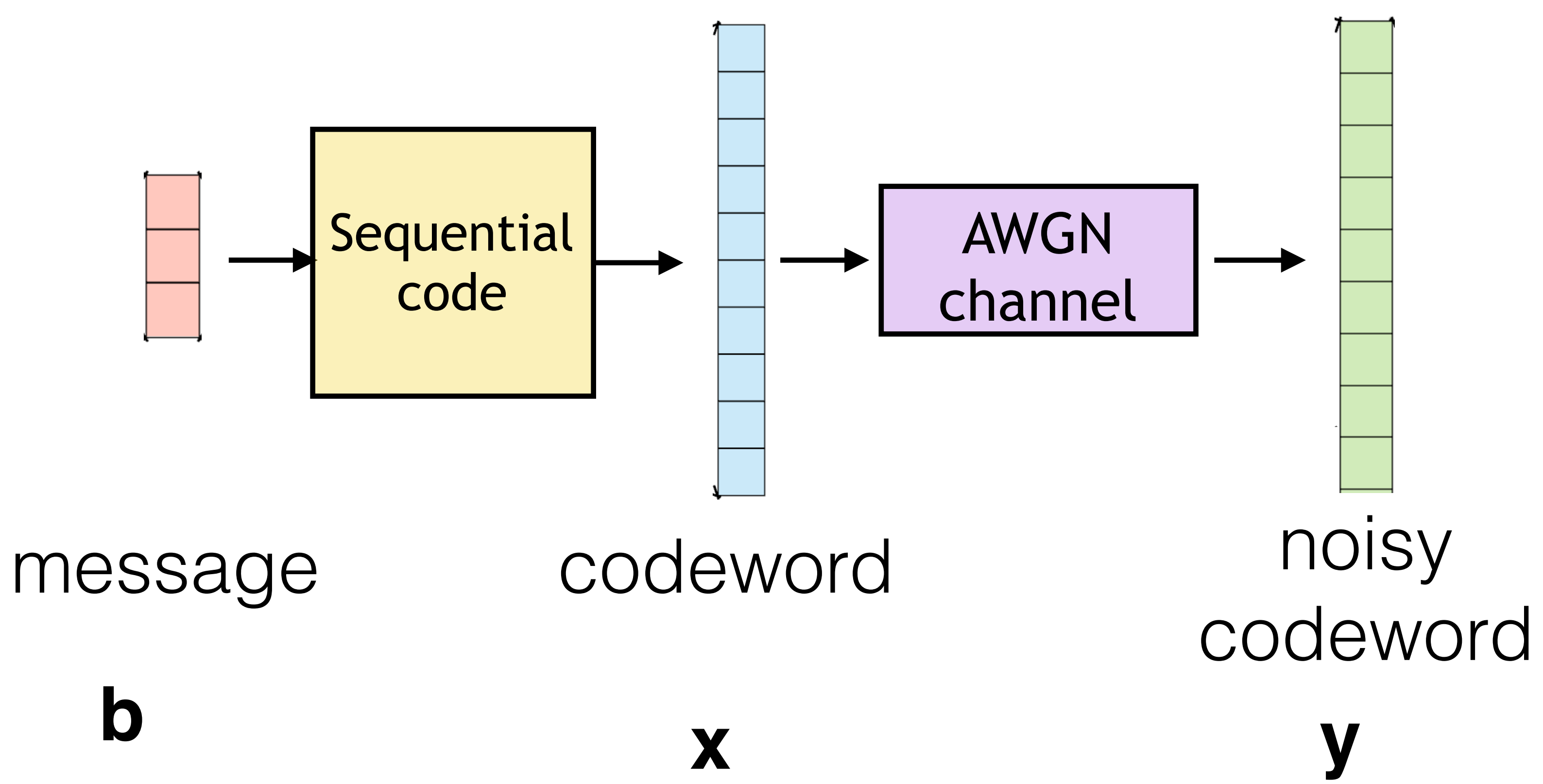
Training

- Supervised training with (noisy codeword **y**, message **b**)
- Loss $\mathrm{E}[(\mathbf{b} - \hat{\mathbf{b}})^2]$



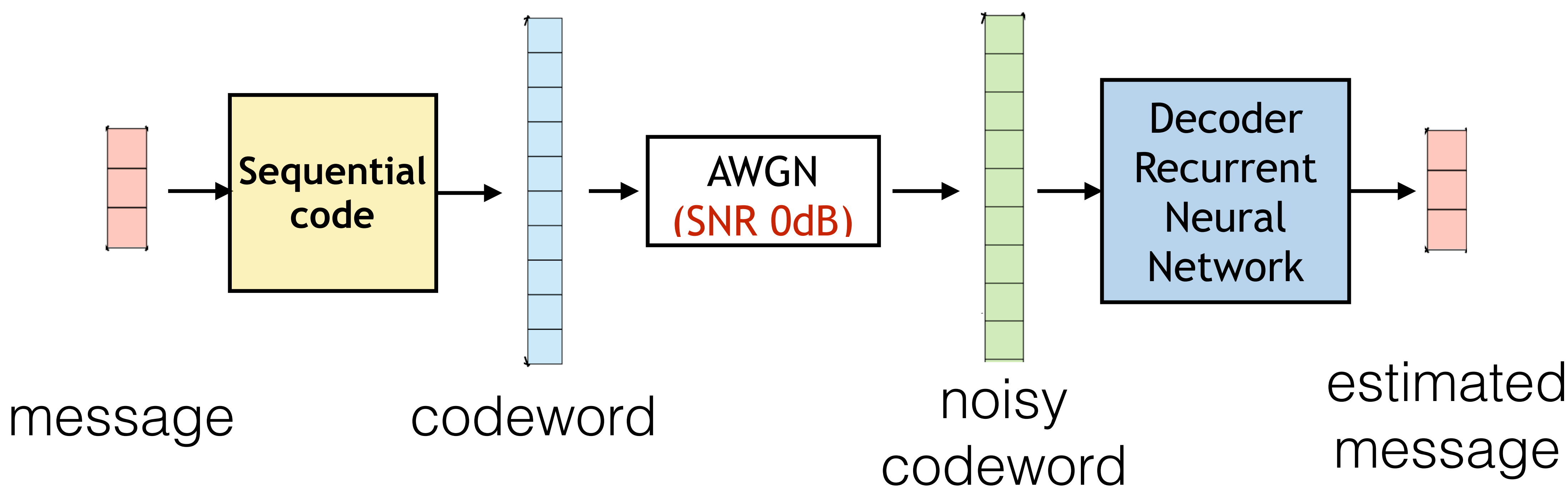
Choice of training examples

- Training examples $(\mathbf{y}, \mathbf{b})$:
  - Length of message bits $\mathbf{b} = (b_1, \ldots, b_K)$
  - SNR of the noisy codeword $\mathbf{y}$



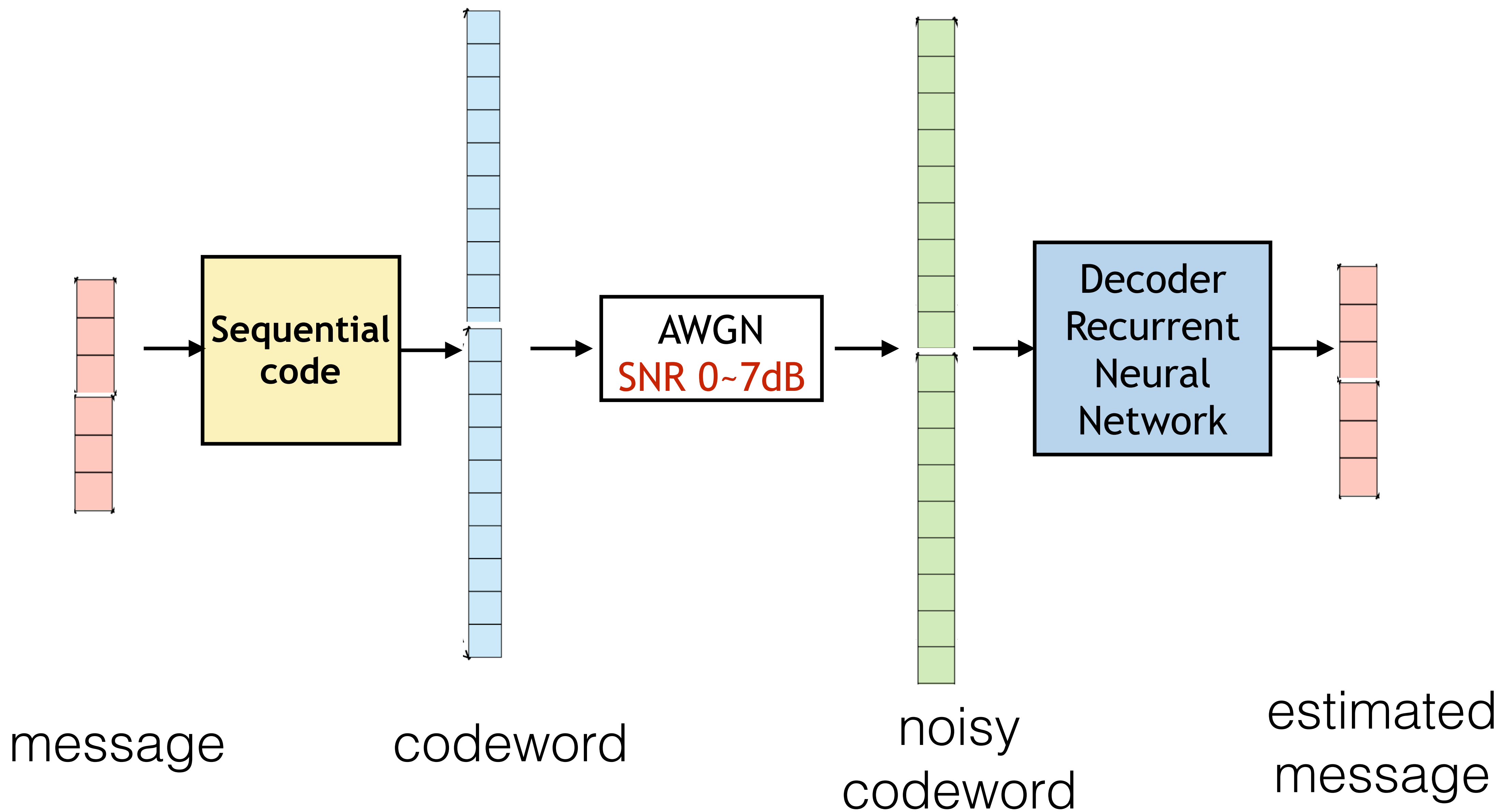
Choice of training examples

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

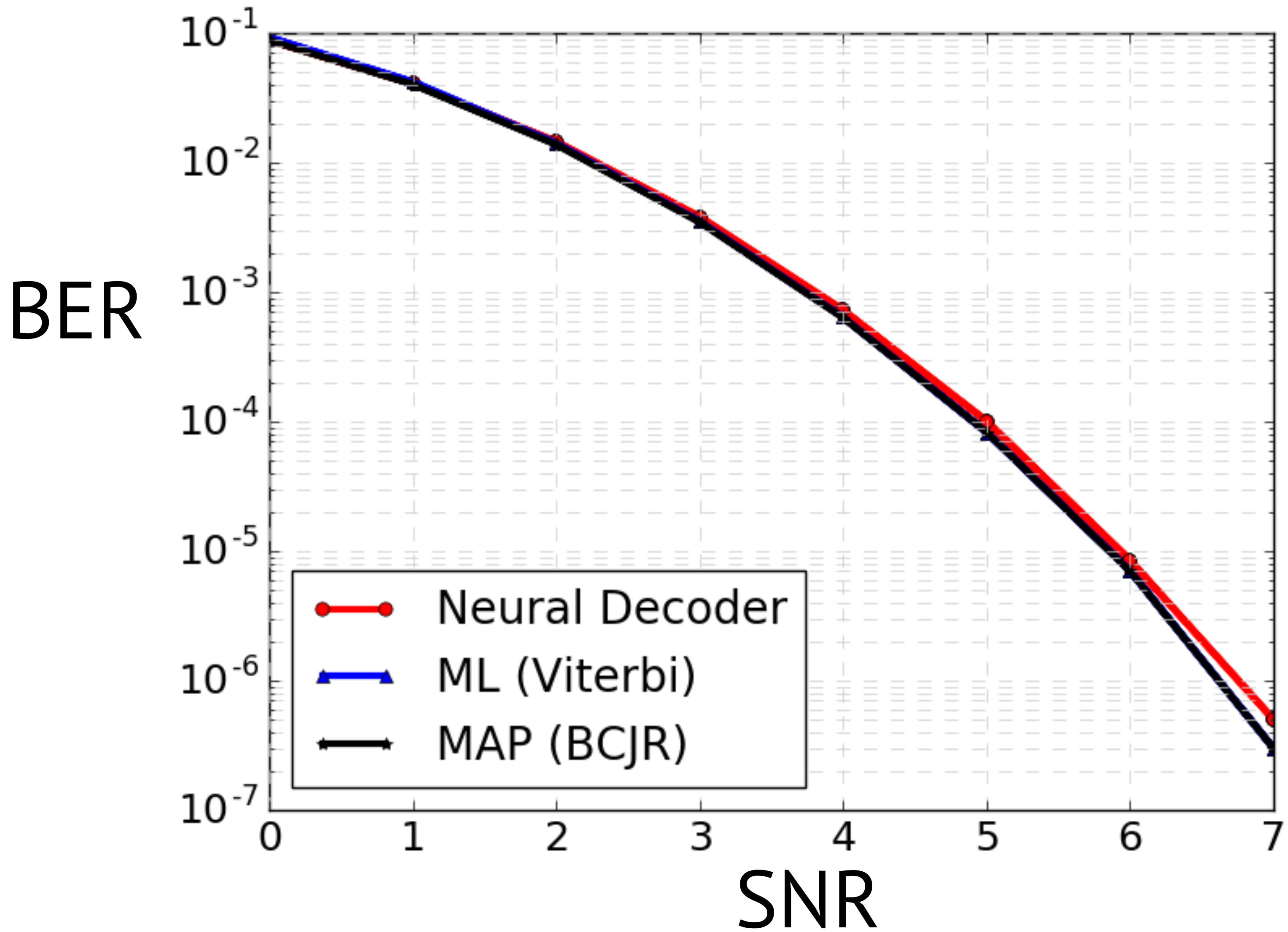


Choice of training examples

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



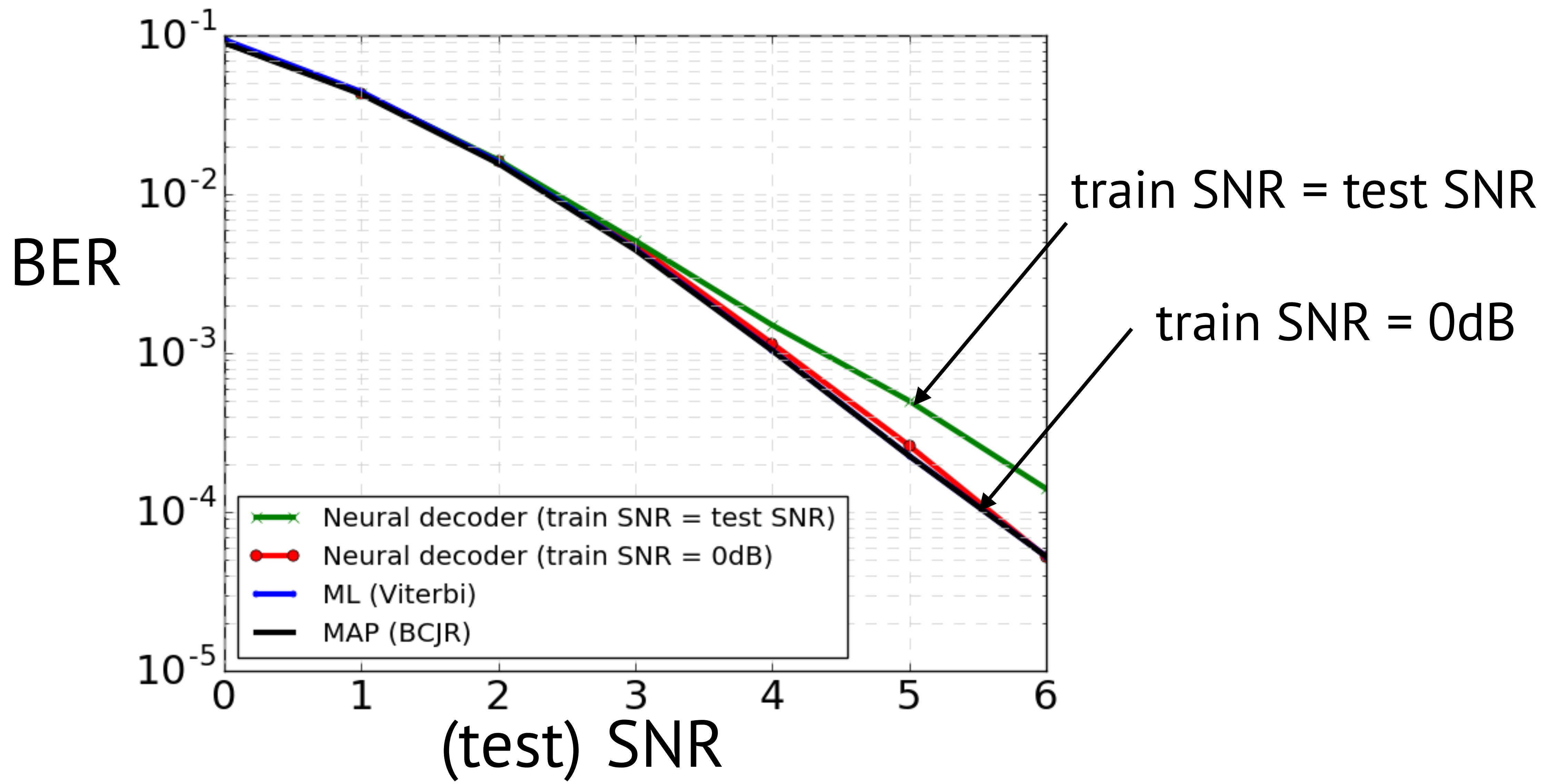
Results: 100x scalability



Train: block length = 100, SNR=0dB   Test: block length = 10K

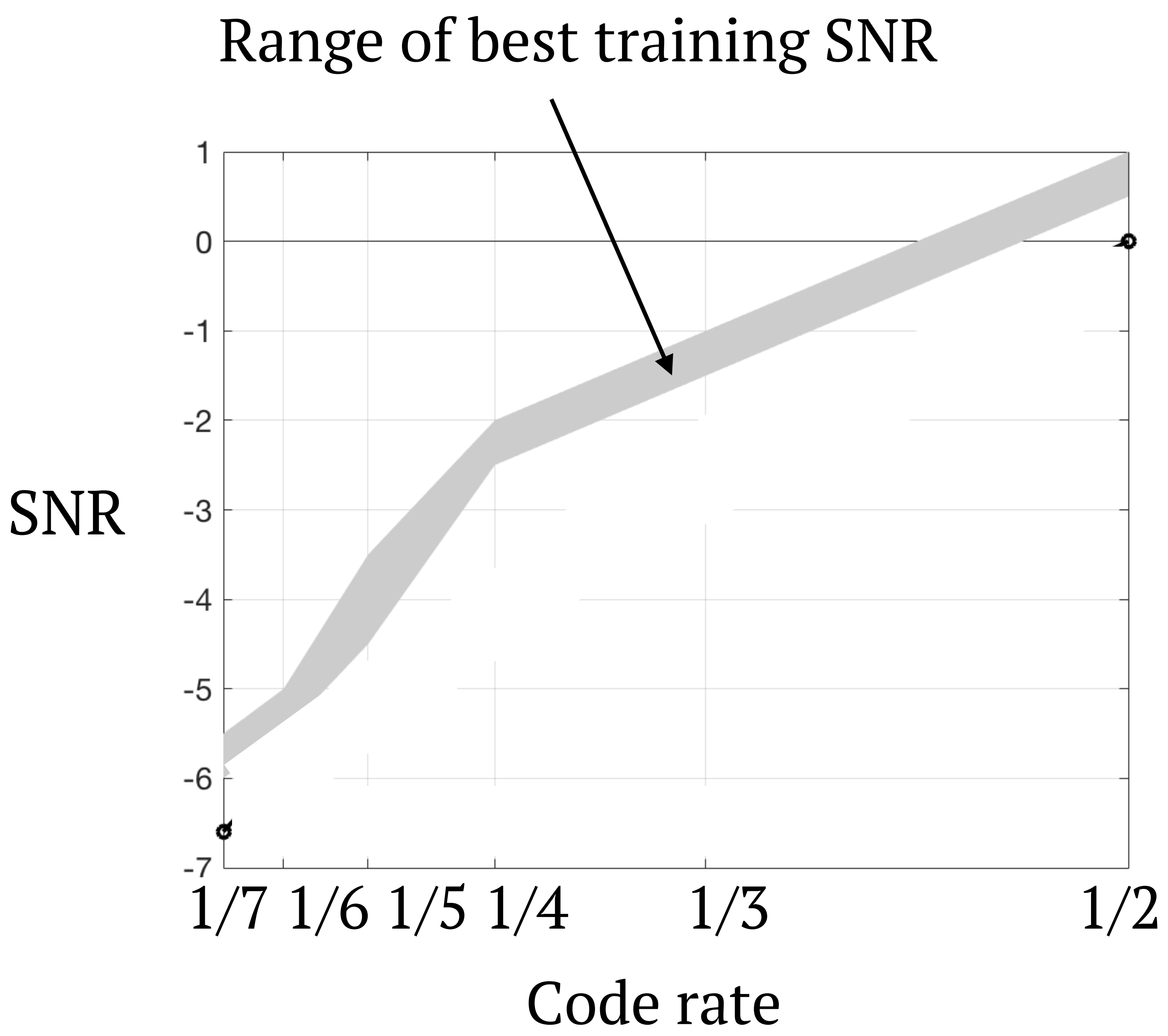
Choice of training examples

- Training equal to test SNR?



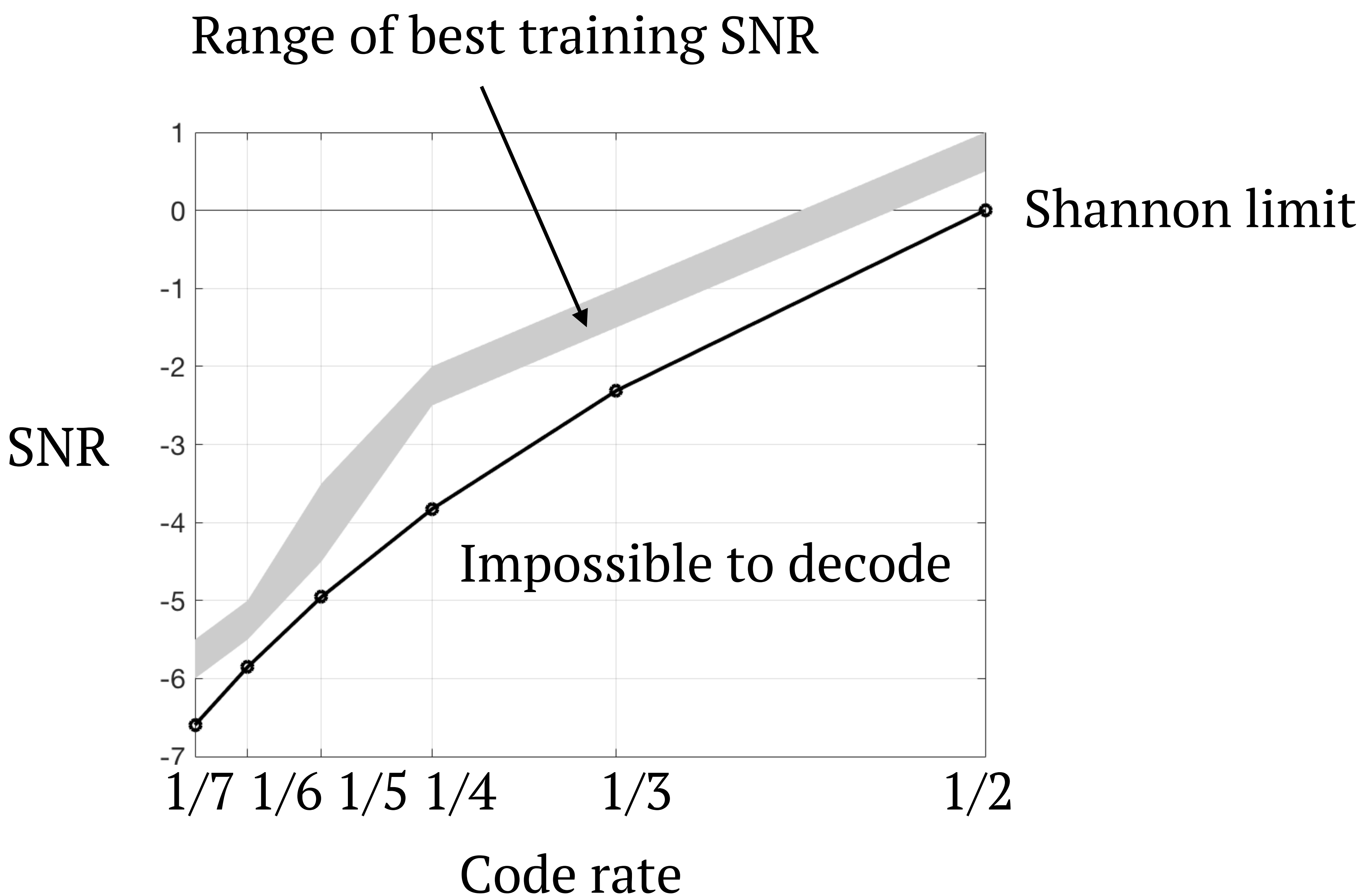
Choice of training examples

- Empirically find best training SNR for different code rates



Choice of training examples

- Hardest training examples



Training with Hard Examples

- **Idea of hardest training examples**
  - Training with noisy labels
  - Applies to problems where training examples can be chosen

Lessons

- Use hard examples for training

- Neural network robust to model mismatch

Many open problems

- Channels with feedback

- Network settings

Theoretical Agenda

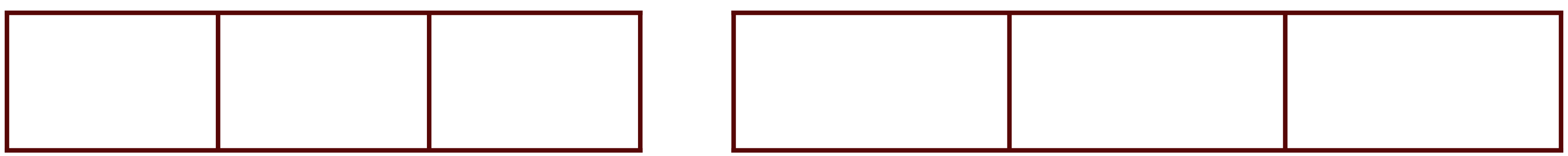
- Underpinning this talk
  - **Gated Recurrent Neural Networks**
  - Nonlinear dynamical systems
    - switched linear system
- **Learning theory meets Switched Dynamical Systems**
  - many open questions
  - basic technical/mathematical value

Collaborators

Neural encoder

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

Phase I. Phase II.

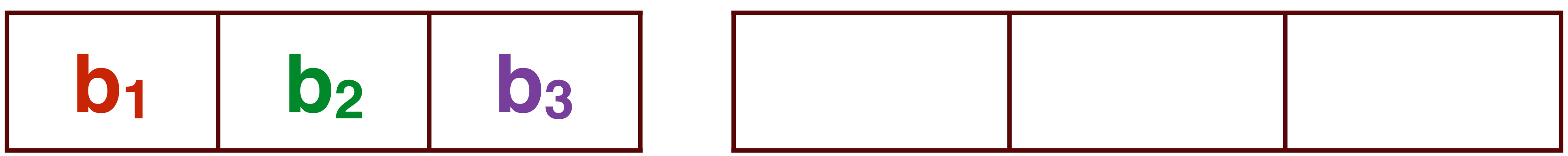


Phase I: send information bits

b_1	b_2	b_3

Encoder gets y_1 y_2 y_3

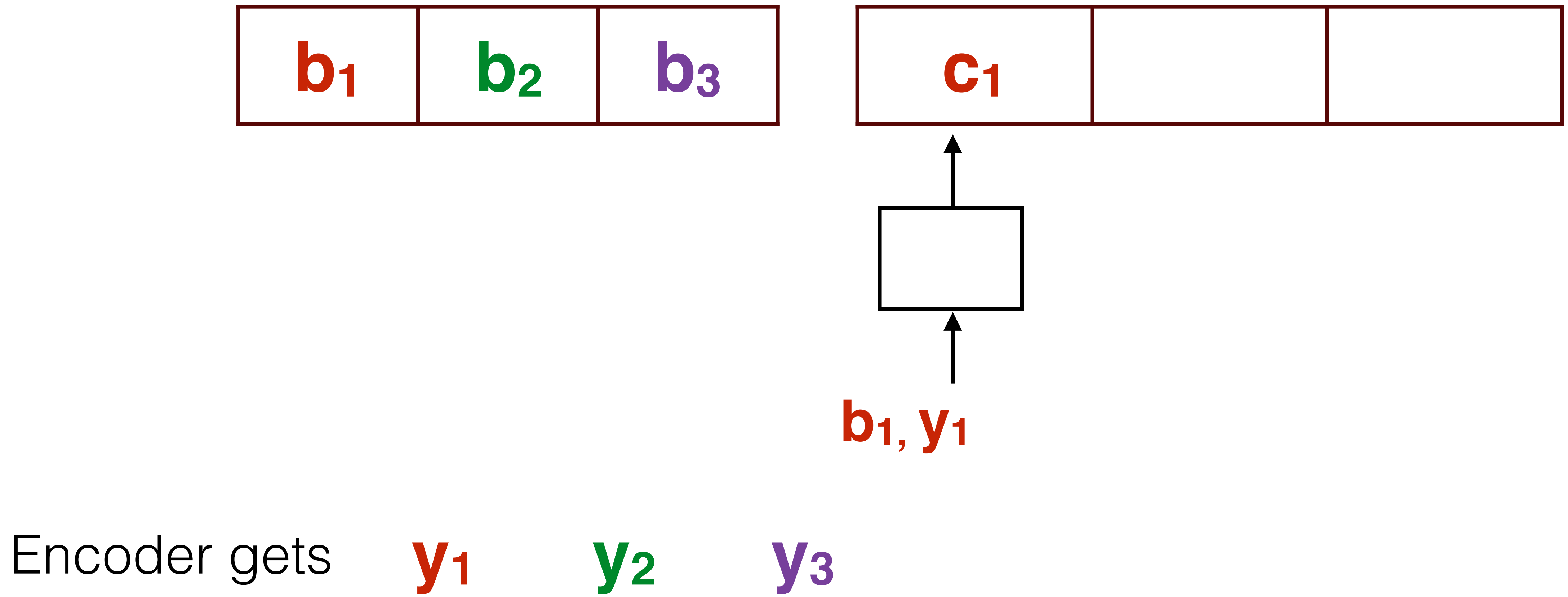
Phase II: use feedback to generate parity bits



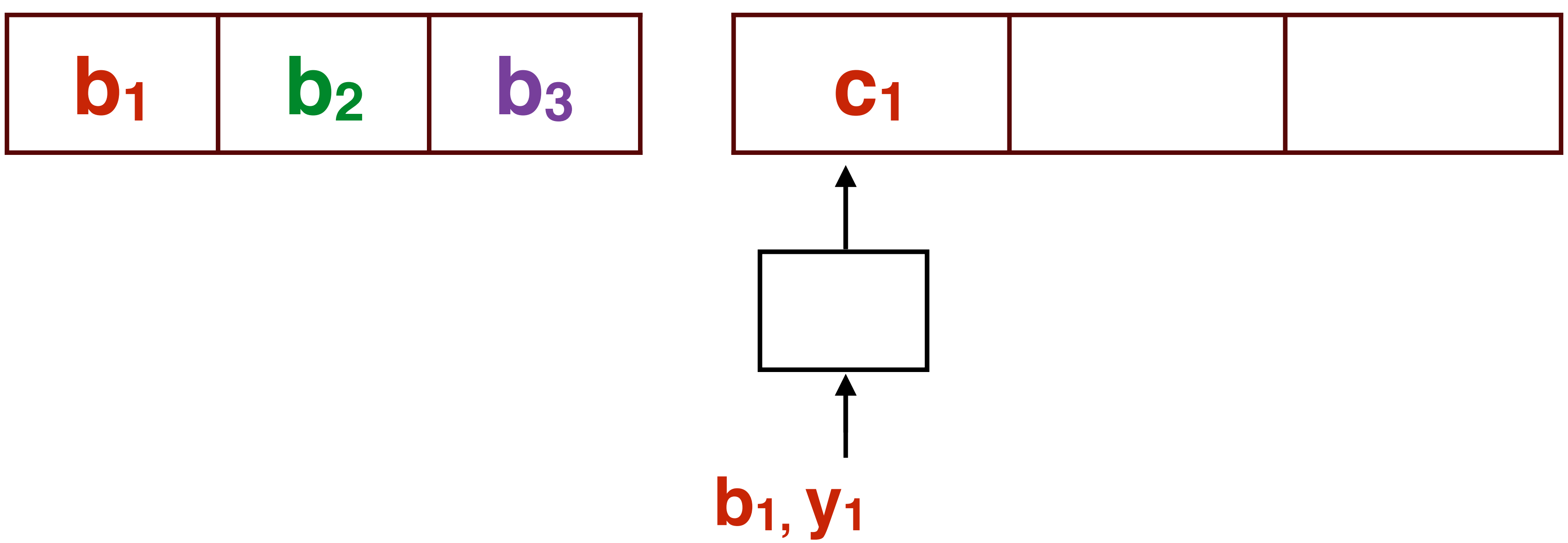
Encoder gets y_1 y_2 y_3

Phase II: use feedback to generate parity bits

- Parity for b_1



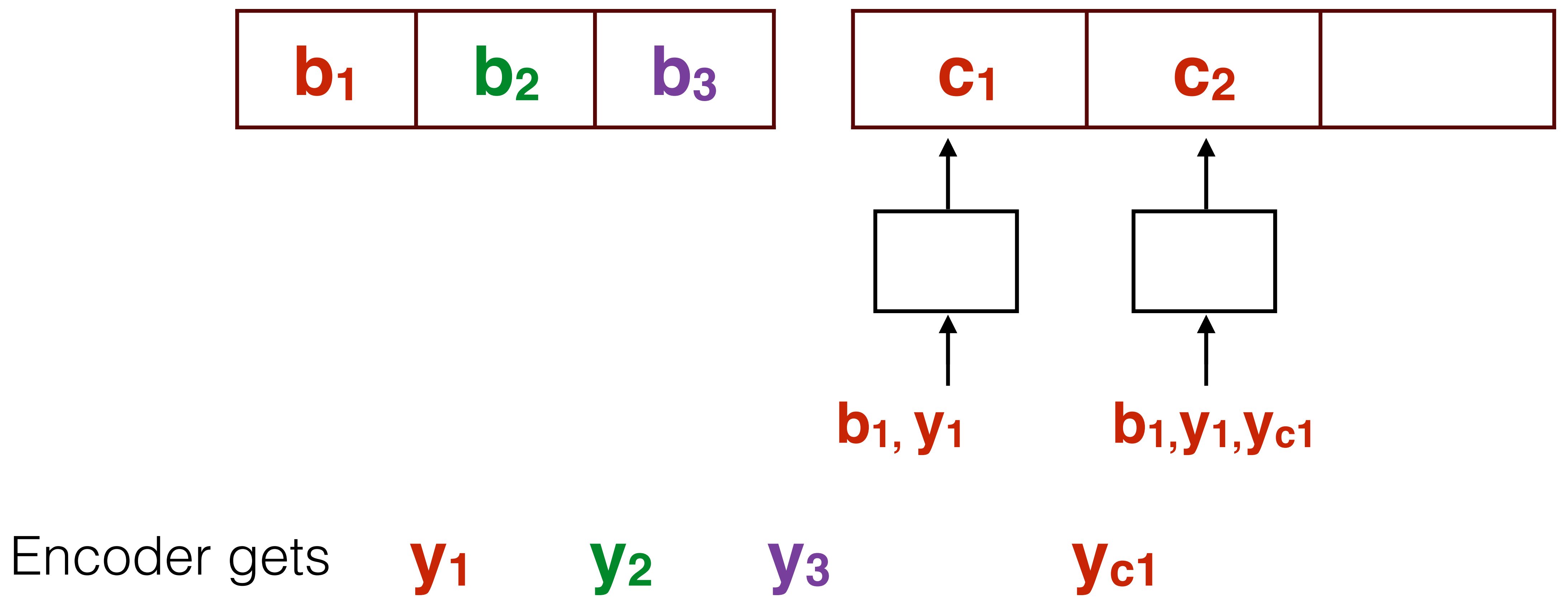
Phase II: use feedback to generate parity bits



Encoder gets y_1 y_2 y_3 y_{c1}

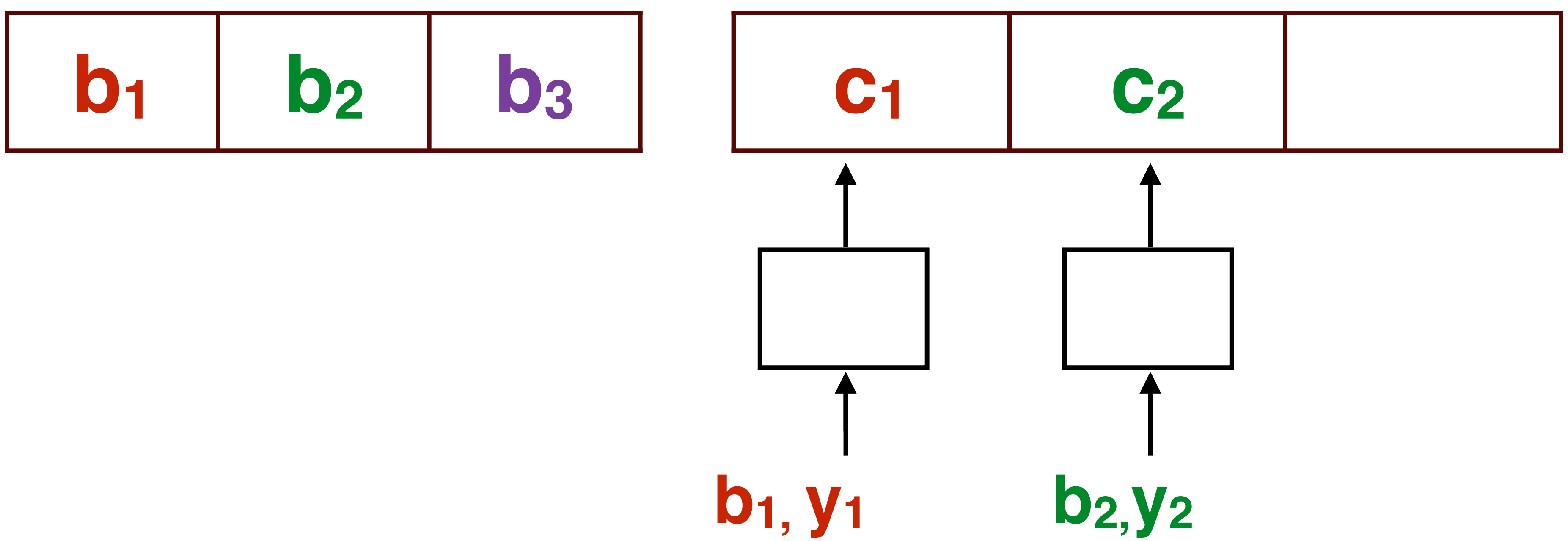
Phase II: use feedback to generate parity bits

- Another parity for b_1?



Phase II: use feedback to generate parity bits

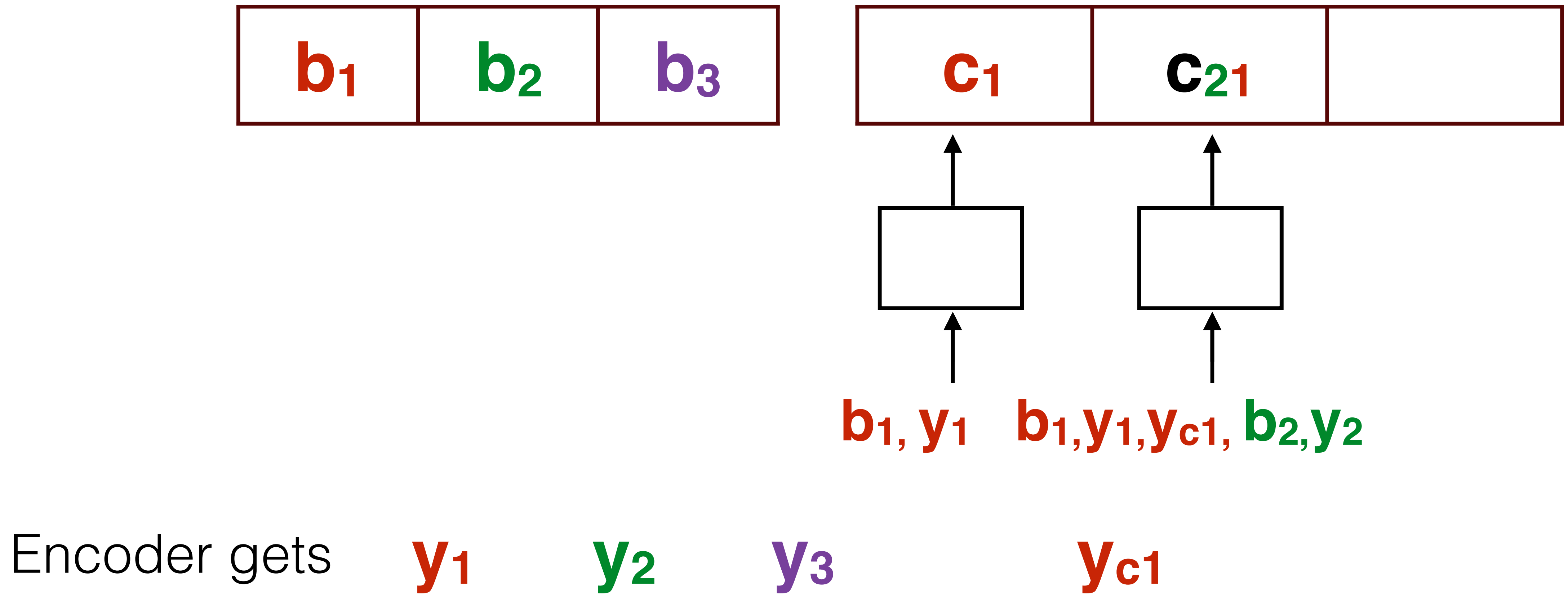
- Parity for b_2?



Encoder gets y_1 y_2 y_3 y_{c1}

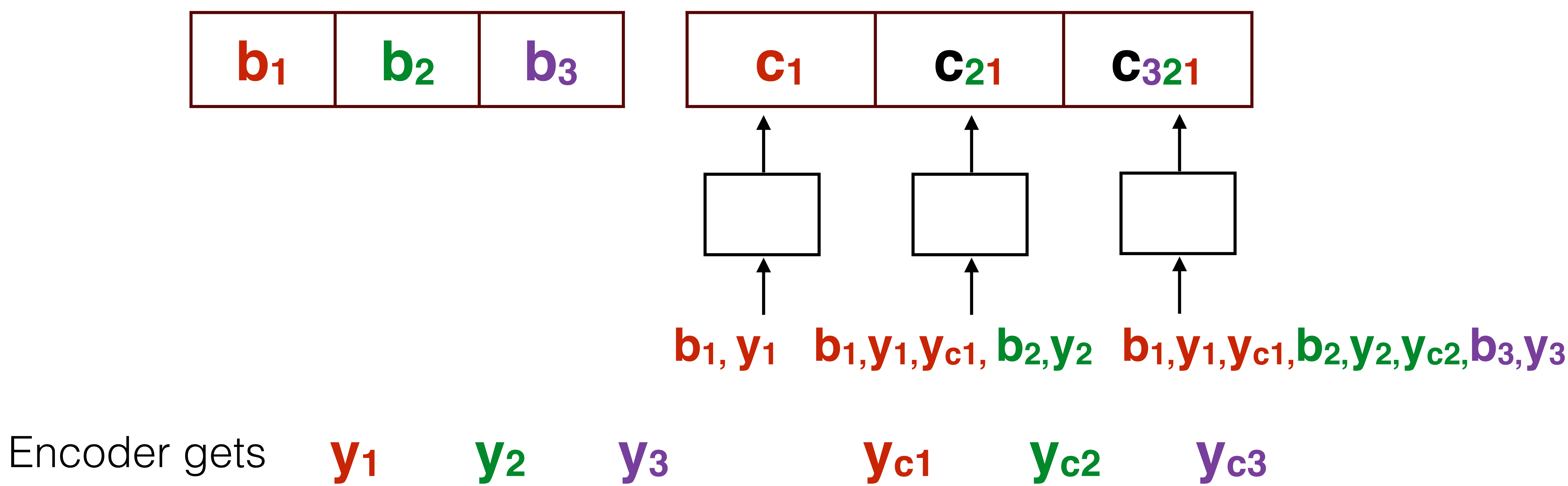
Phase II: use feedback to generate parity bits

- Parity for b_2 and b_1



Phase II: use feedback to generate parity bits

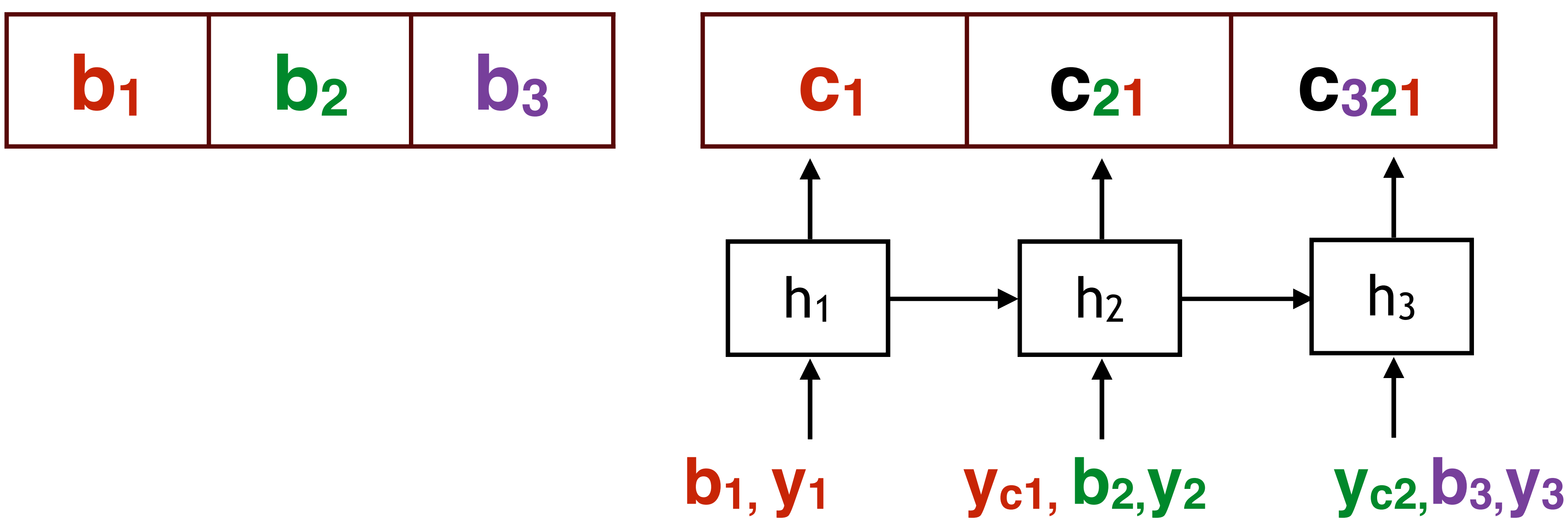
- Parity for b_3, b_2 and b_1



Recurrent Neural Network for parity generation

- Sequential mapping with memory

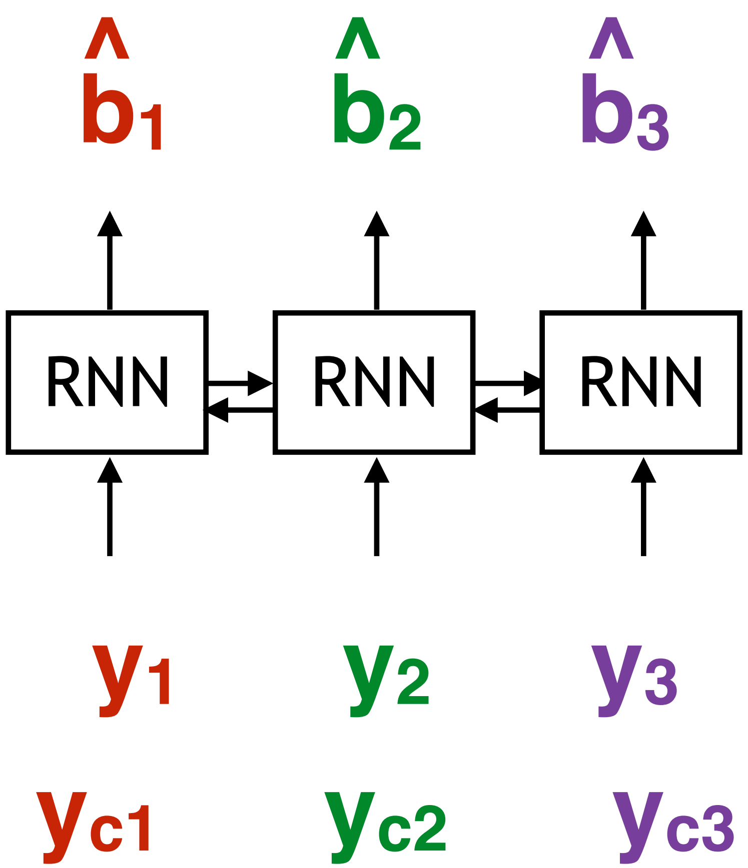
$$h_i = f(h_{i-1}, \text{Input}_i)$$
$$\text{Output}_i = g(h_i)$$



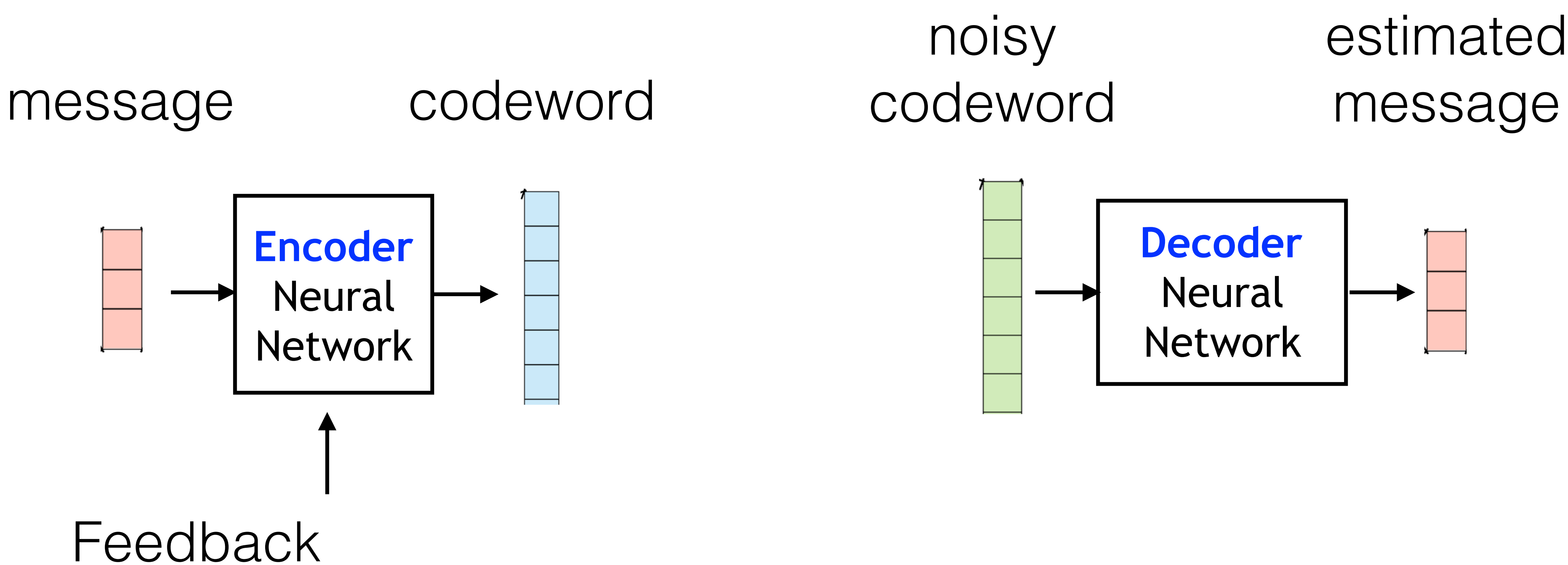
Encoder gets y_1 y_2 y_3 y_{c1} y_{c2} y_{c3}

Neural decoder

- 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

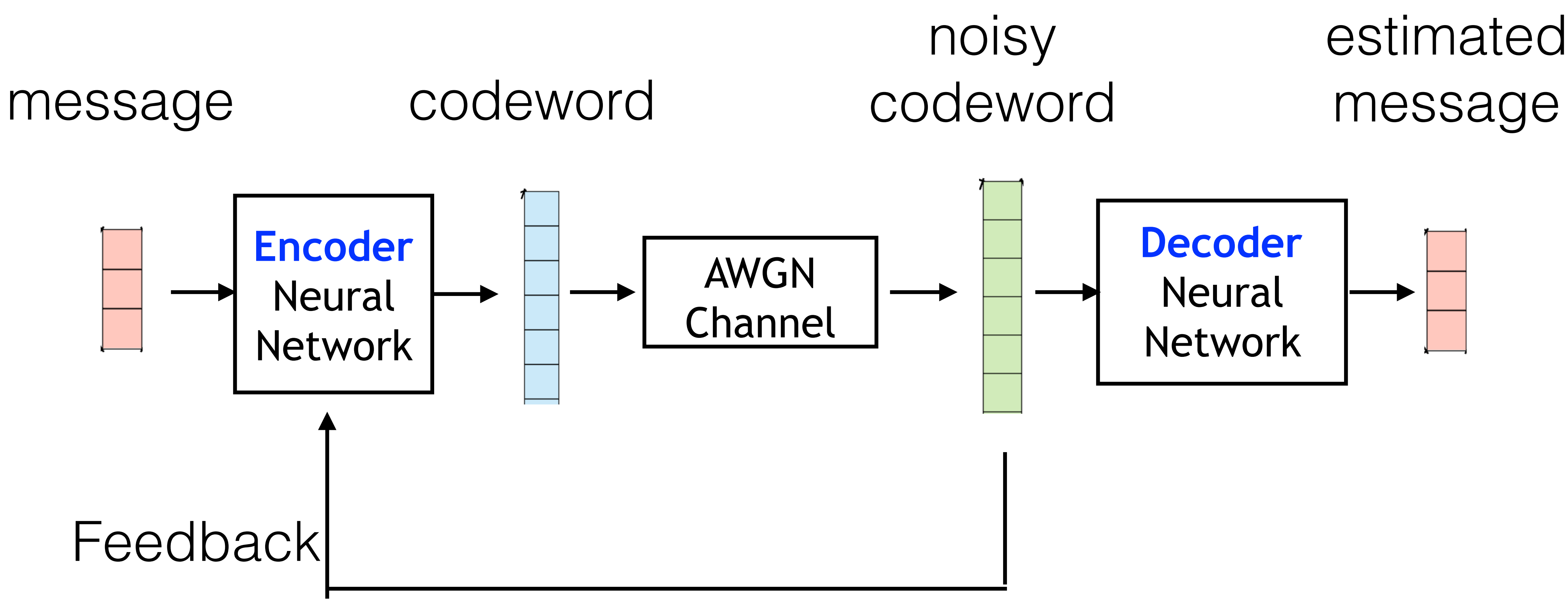


Neural encoder and decoder



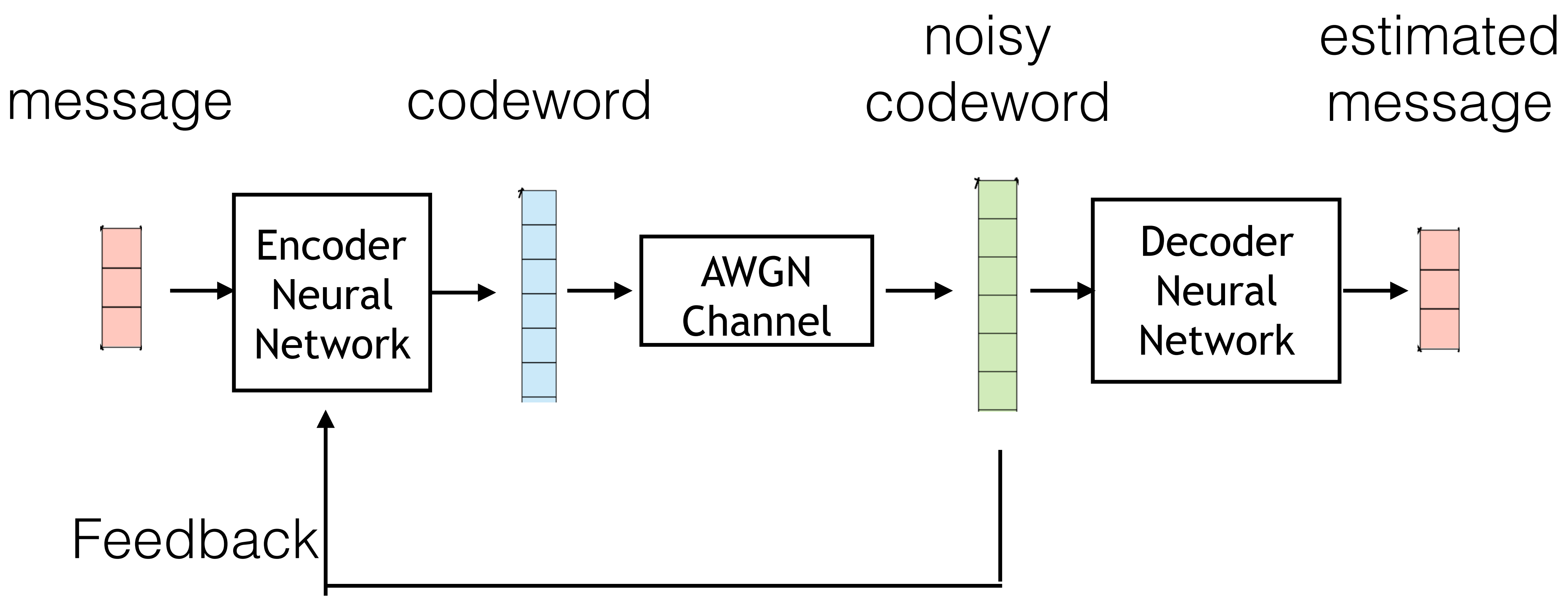
Training

- Learn the encoder and decoder jointly



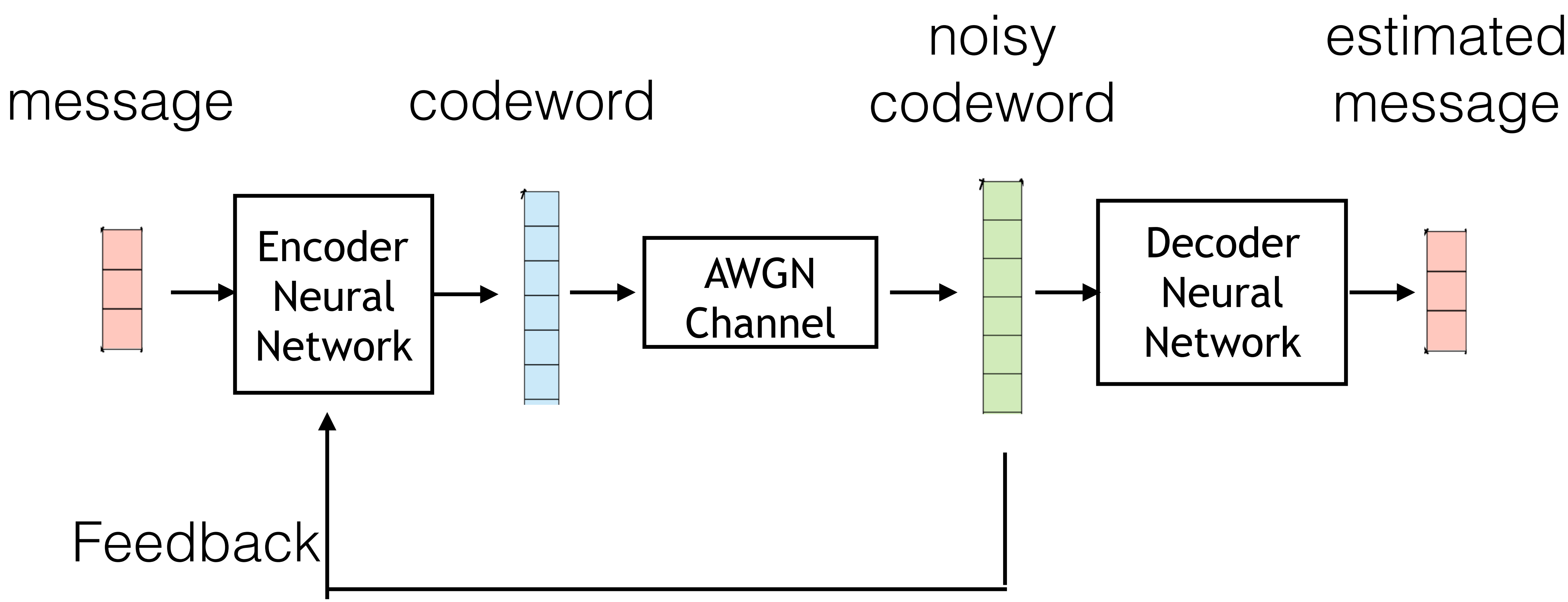
Training

- Learn the encoder and decoder jointly
  - Challenge: Gradients have to pass through decoder



Training

- Learn the encoder and decoder jointly
  - Challenge: Gradients have to pass through decoder



  - **Encoder simpler than decoder**

Supervised Training

- Auto-encoder training : (input,output) = (**b**,**b**)

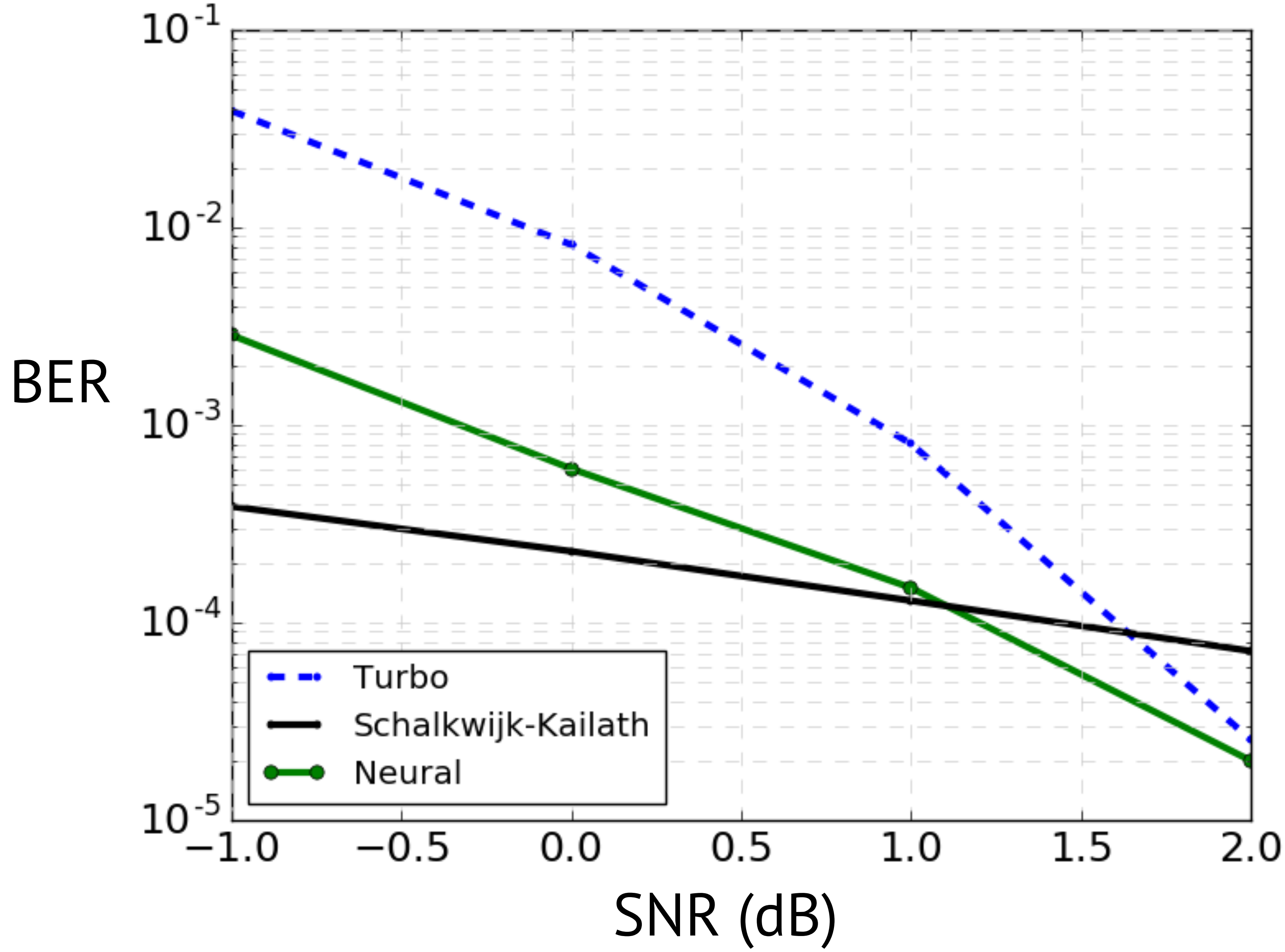
$$\mathbf{b} = (b_1, b_2, \cdots, b_K)$$

- Loss : binary cross entropy

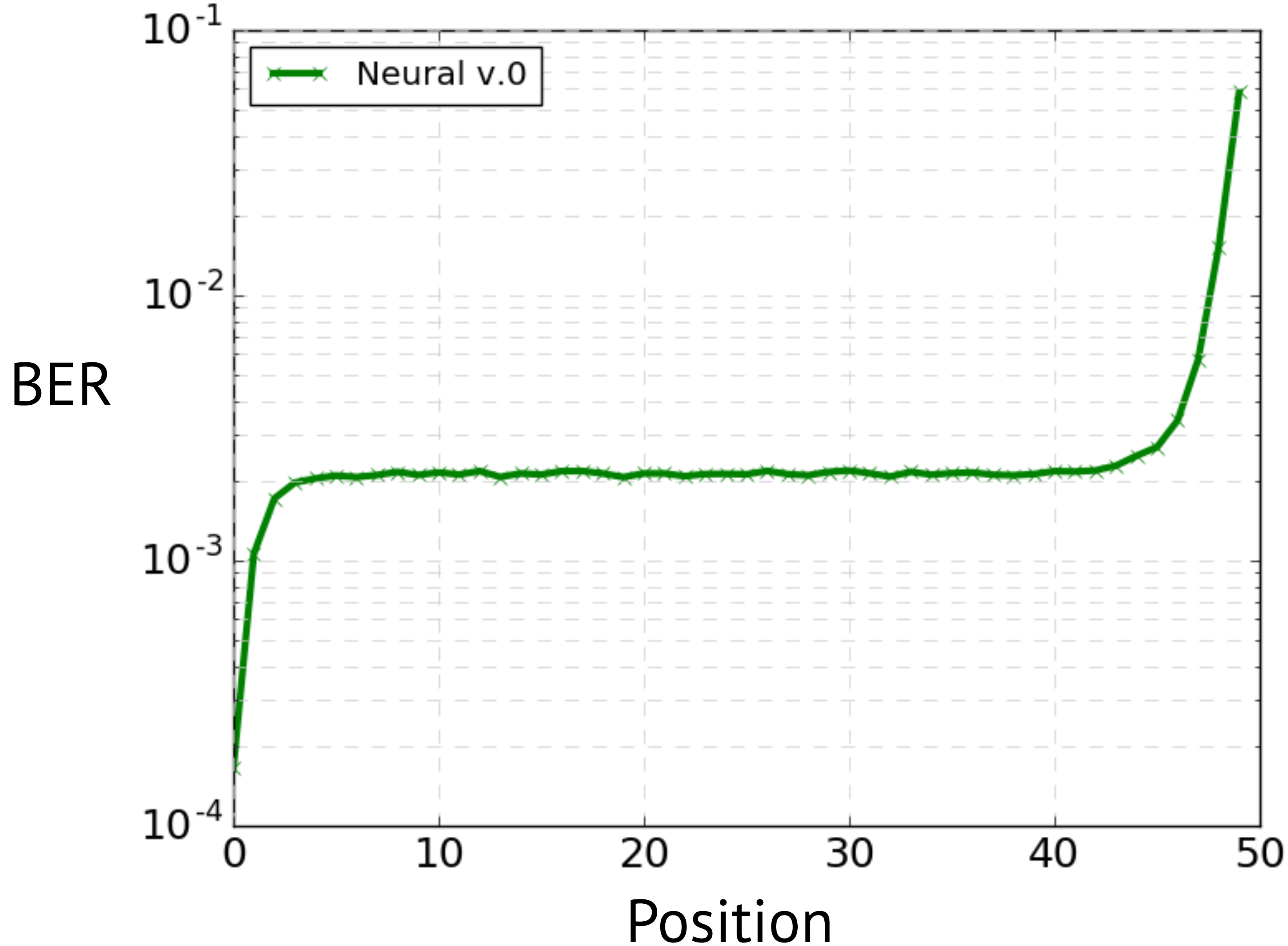
$$\mathcal{L}(\mathbf{b}, \hat{\mathbf{b}}) = -\mathbf{b}\log\hat{\mathbf{b}} - (1-\mathbf{b})\log(1-\hat{\mathbf{b}})$$

- Training code length :
  - long enough (100)

Intermediate results



High error in the last bits

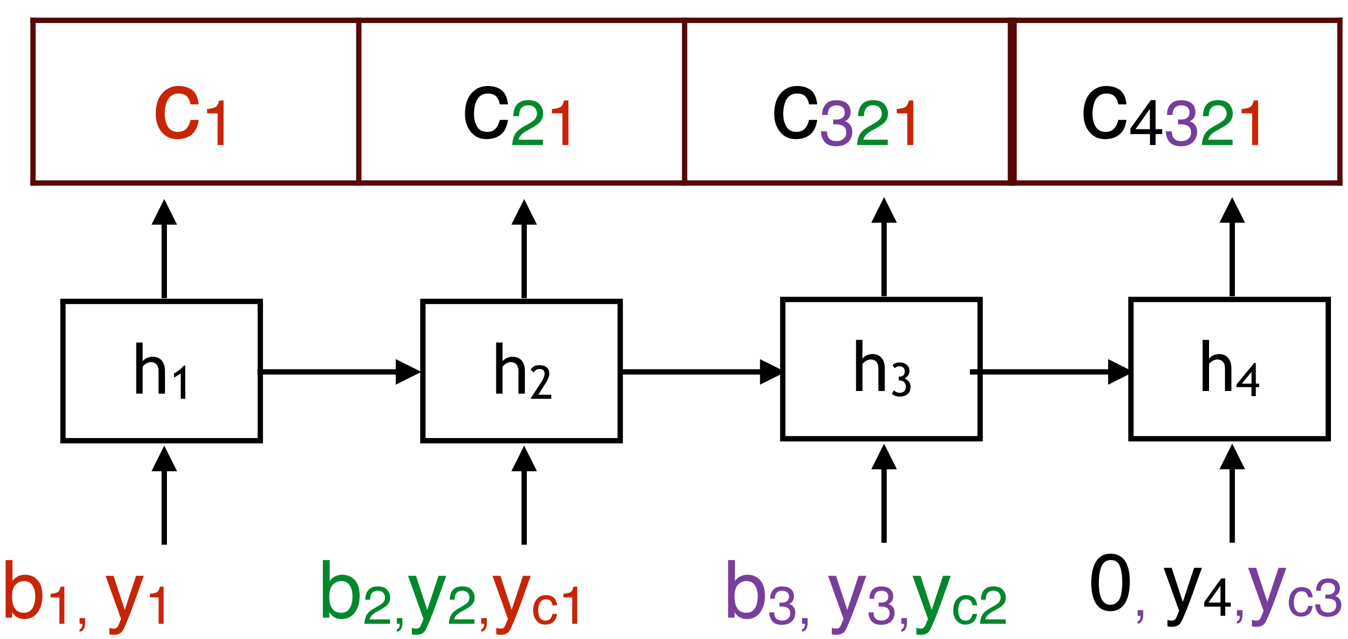


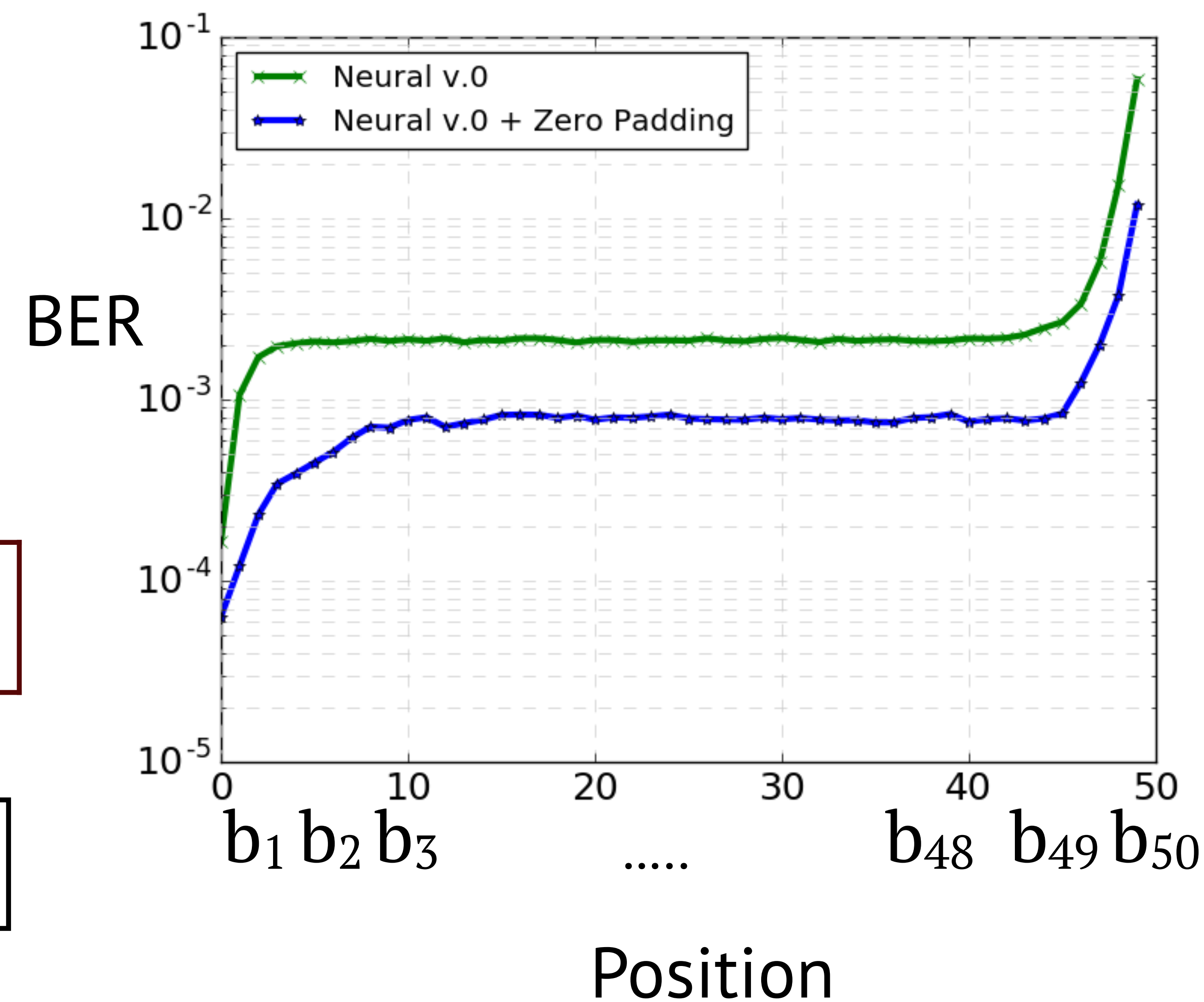
Idea 1. Zero padding

Phase I.

b_1	b_2	b_3	**0**

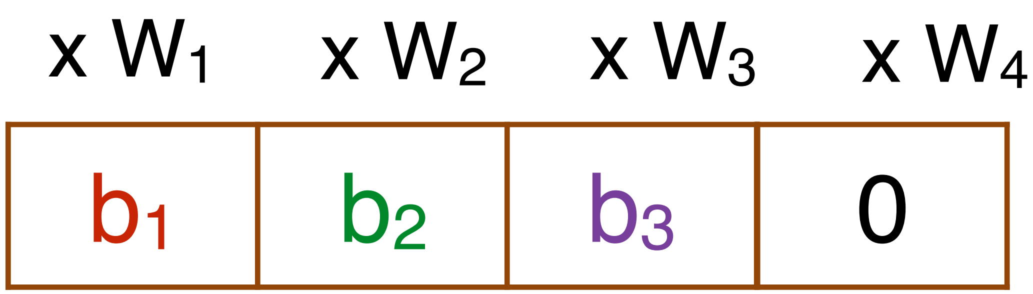
Phase II.



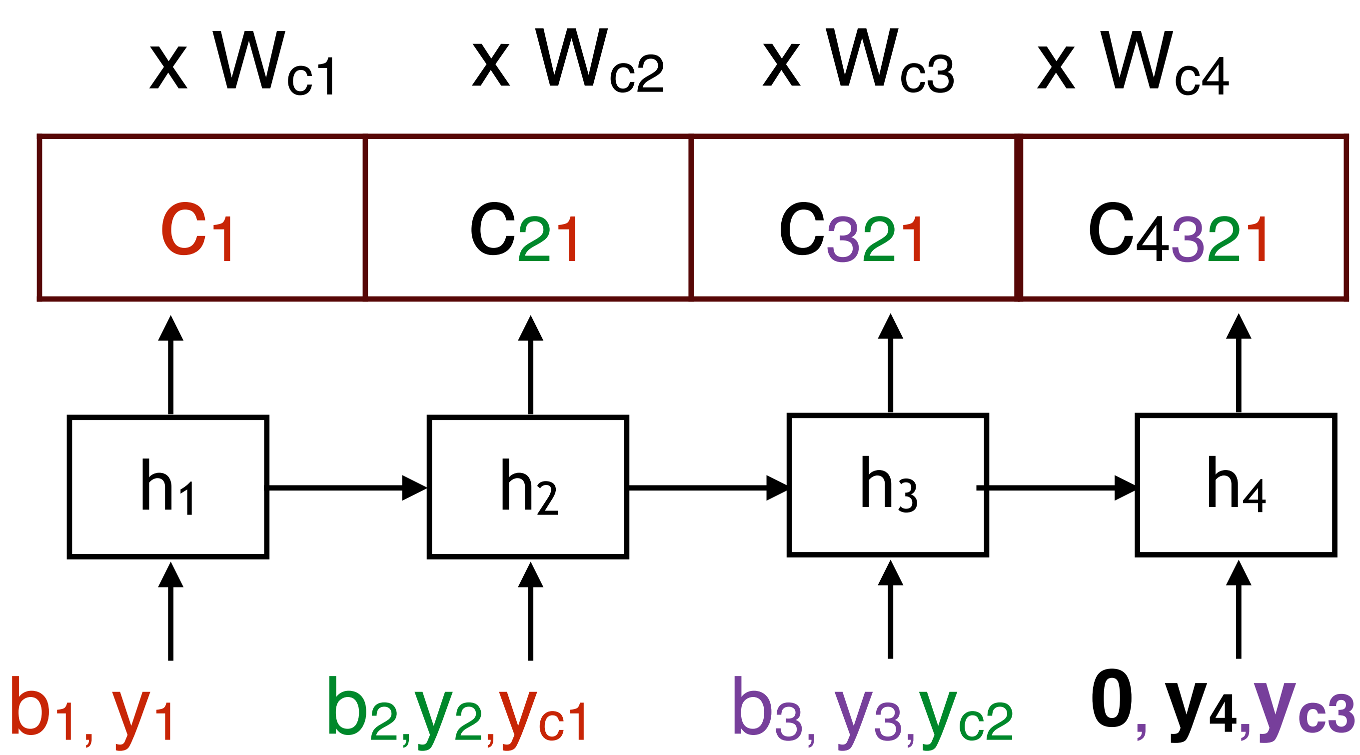


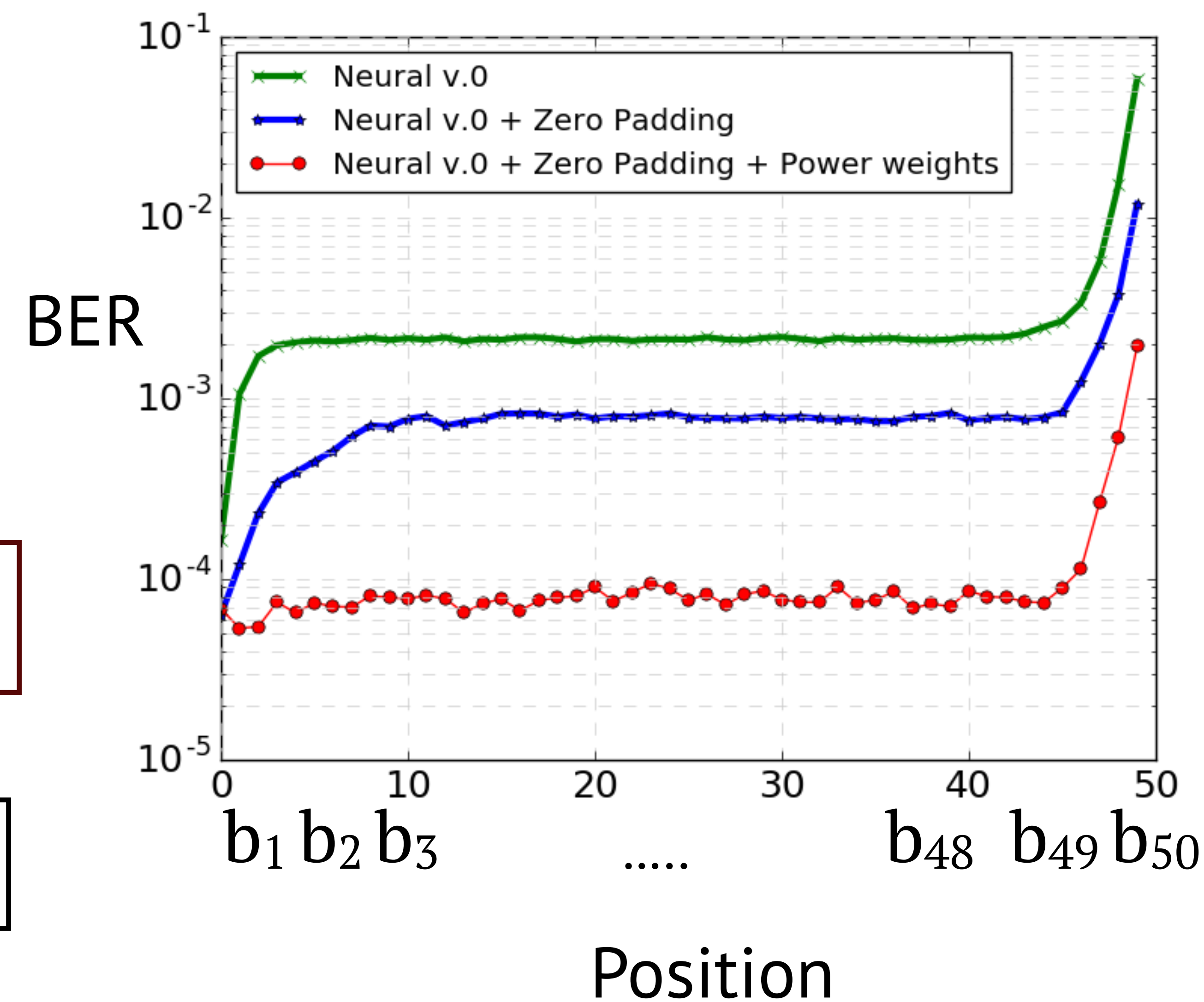
Idea 2. Power allocation

Phase I.



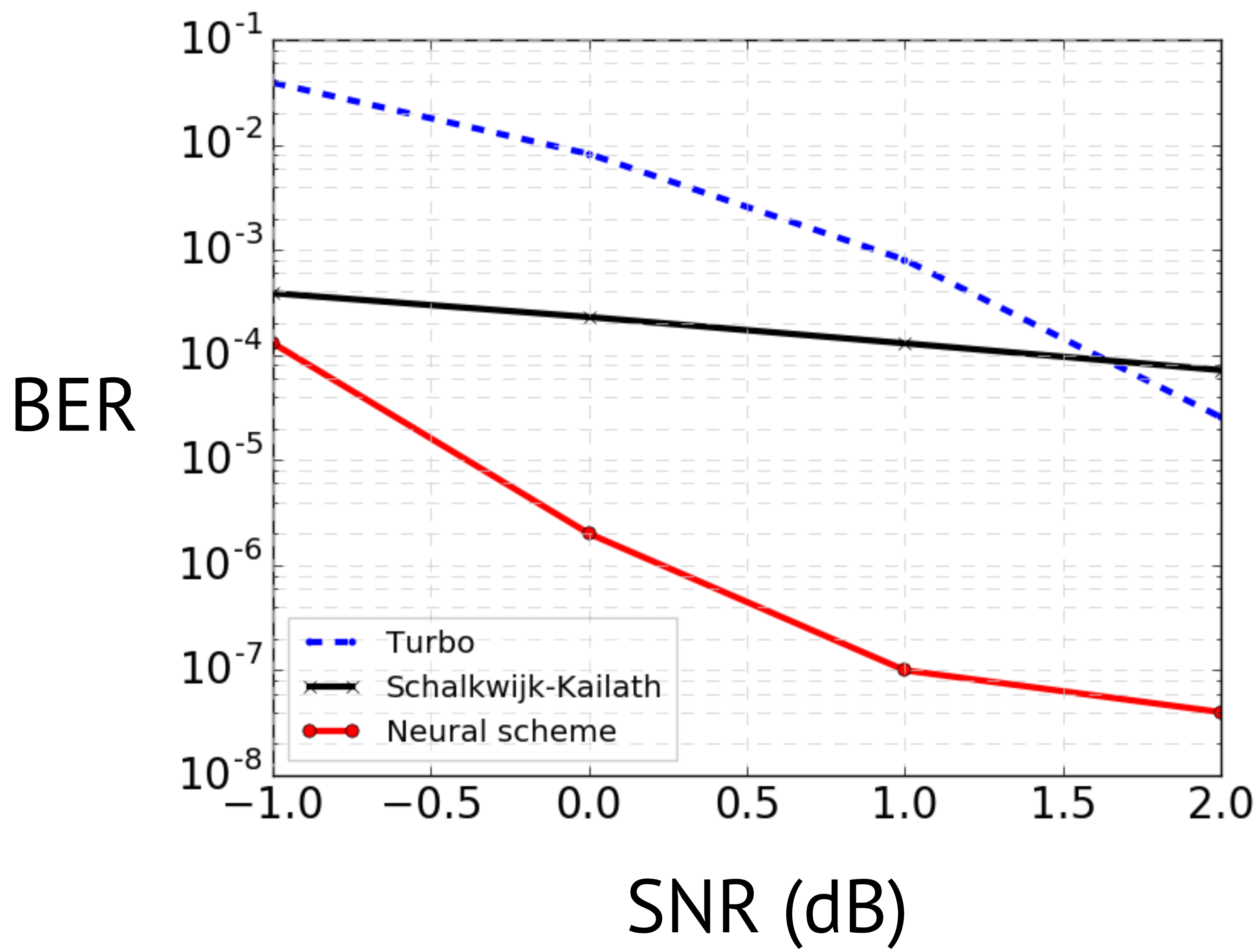
Phase II.





Results

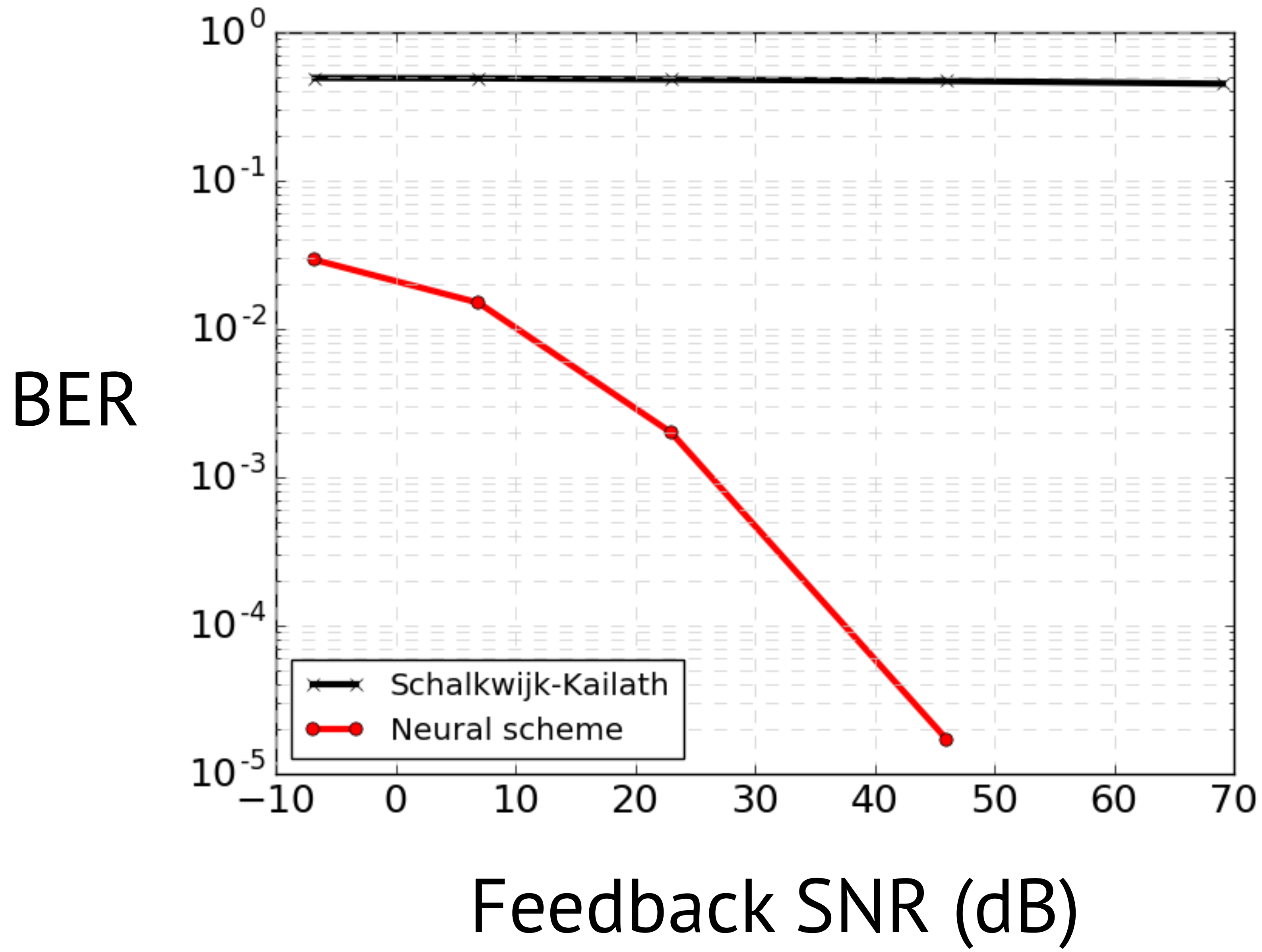
- 100x better reliability under feedback w. machine precision



(Rate 1/3, 50 bits)

Results

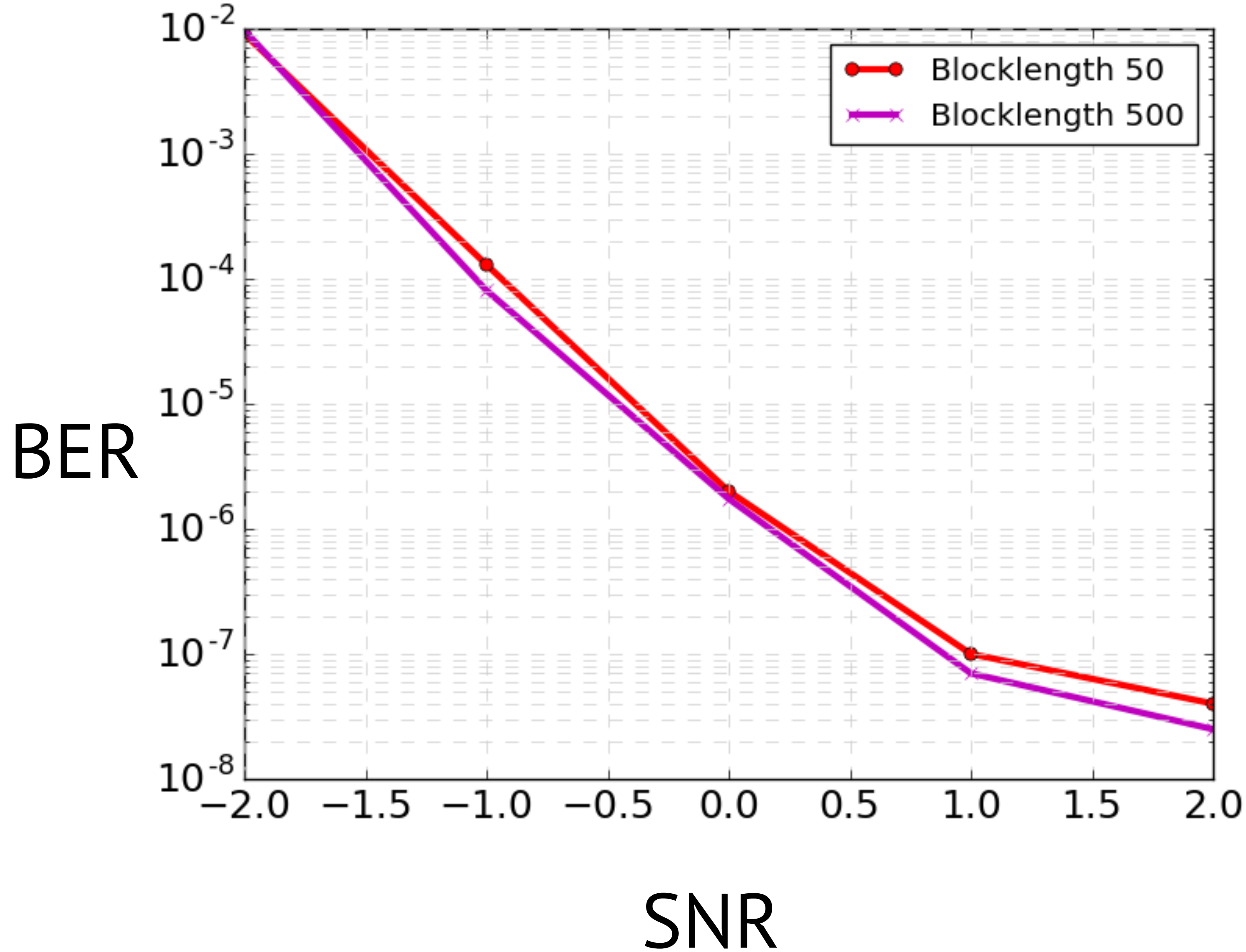
- Robust to noise in the feedback



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

Generalization : block lengths

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



Improved error exponents

- Concatenated code : turbo + neural feedback code
  - BER decays faster