第９回 誤り訂正符号のワークショップ

Learning-based approach for designing error-correcting codes

Shan LU
Gifu University

２０２０年９月２日～９月３日
＠オンライン開催

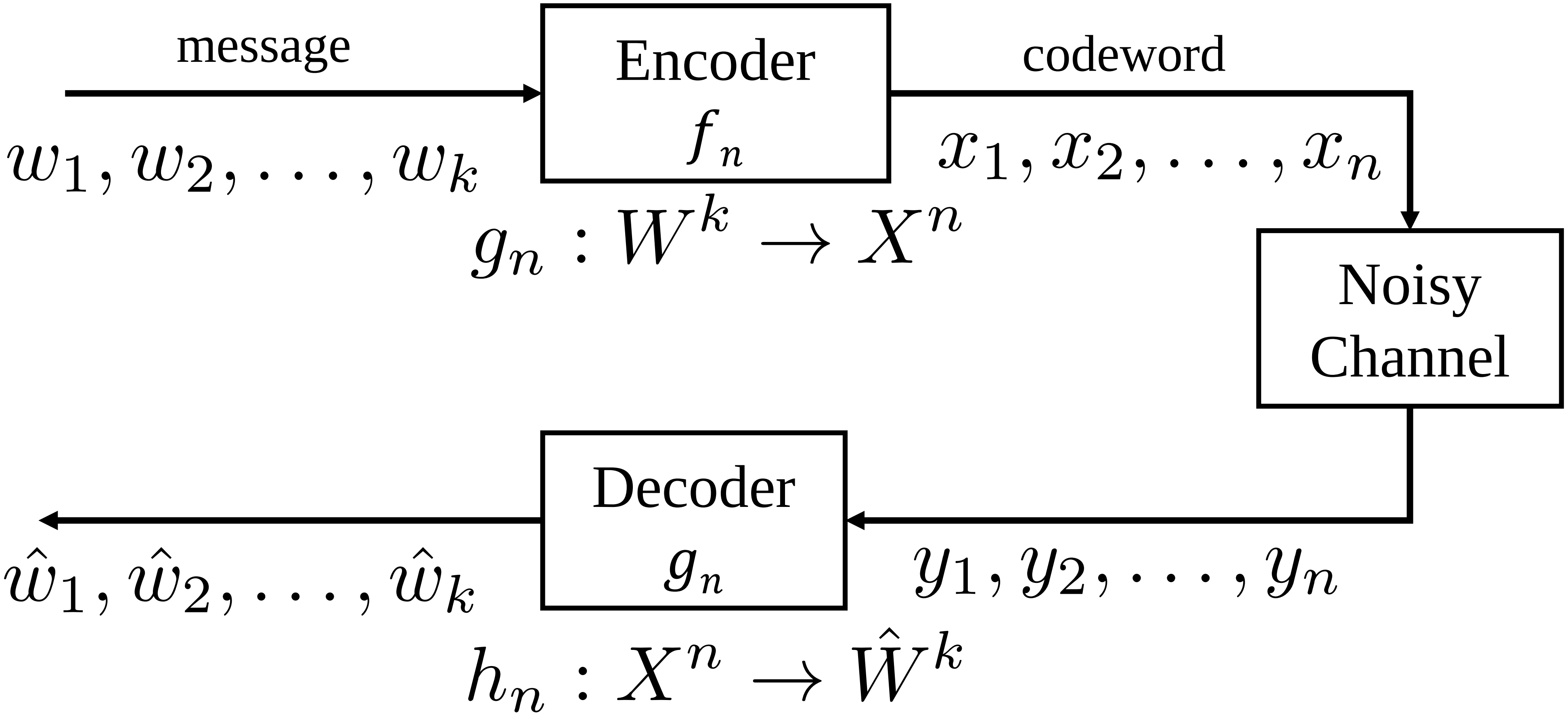
Noisy-Channel Coding Theorem

Noisy-Channel Coding Theorem

Given a noisy channel with <u>channel capacity</u> C,
for arbitrary small $\epsilon > 0$, if information transmitted rate $R < C$ and code length n *is* sufficiently large,
there exists an $[n, k]$ code of rate $k/n \geq R$ with error probability p

$$p \leq \epsilon.$$

The model of coding system



Traditional error-correcting codes (linear codes)

- Generator Matrix G

$$\boldsymbol{x} = \boldsymbol{w} \cdot G$$

- Parity-Check Matrix

$$G \cdot H^{\mathrm{T}} = 0$$

G: generator matrix

w: message vector

x: codeword

- Fixed design of error-correcting codes

Short code (design Generator Matrix or Parity-Check Matrix)

Hamming Distance: block codes using finite field algebra

Such as: Hamming codes, Golay codes, RM codes, BCH codes, RS codes, etc.

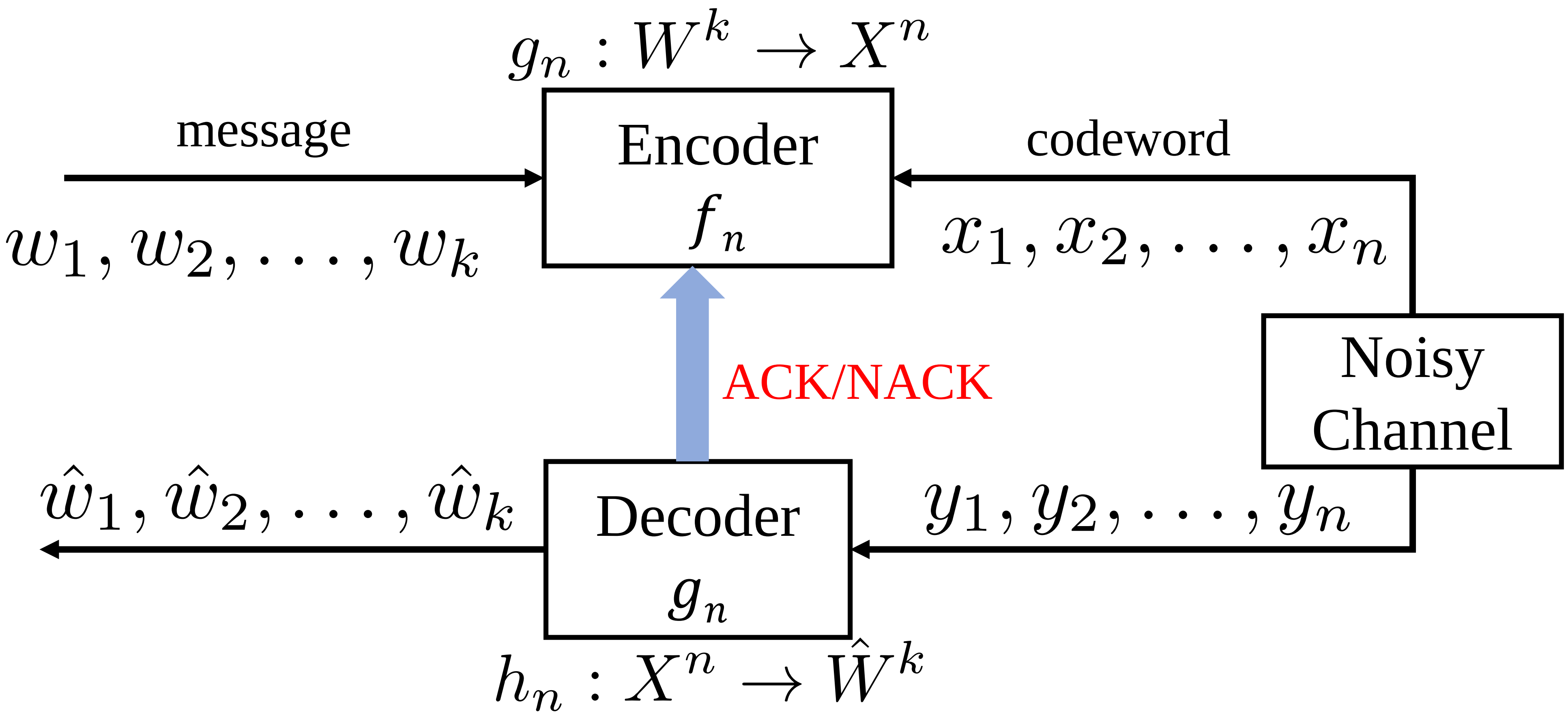
Free distance: convolutional codes by increasing memory order/selecting polynomials

Such as: convolutional code/ Turbo code

Long code: design the property for code ensemble, choose one code from code ensemble.

(If failed to transmit, ARQ (Automatic repeat-request) is used.)

Flexible design of error-correcting codes



- HARQ:(Hybrid Automatic repeat-request)
  - Rateless code
  - Rate-compatibility code

Learning design of error-correcting code

- AI techniques: machine learning/ deep learning/Reinforcement learning
- AI techniques is a natural choice for learning the encoding and decoding functions due to their ability to perform universal function approximation.

- How: Neural network + optimization algorithm

  (ニューラルネットワーク + 最適化アルゴリズム)

- fixed design of error-correcting codes
  - **Supervised learning:** Learning with a **labeled training** set
- flexible design of error-correcting codes
  - **Reinforcement learning**: Learn to **act** based on **feedback/reward**

Design of error-correcting codes

	Fixed design	**Flexible design**
traditional design	Differential evolution algorithm	Rateless code Rate-compatibility code
Design by learning	Supervised learning	Reinforcement learning

Traditional design of error-correcting code

Fixed design	Flexible design
Differential evolution algorithm 差分進化法 (Example of LDPC)	Rateless code Rate-compatibility code

LDPC code：Low-Density Parity-Check code

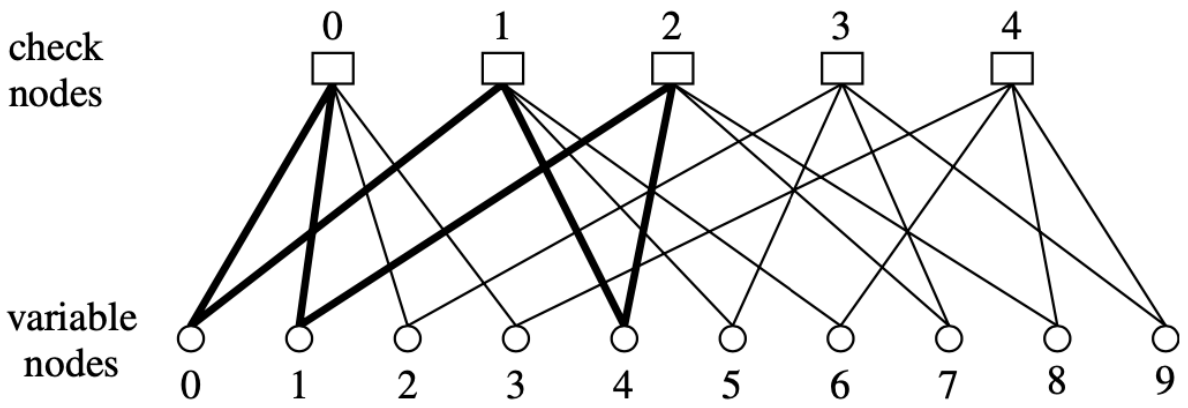
Design a code with coderate $R = k/n$ is to find a parity check matrix $H \in \{0,1\}^{(n-k)\times n}$

Sparse parity check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Find a sparse H or a sparse Tanner graph

sparse Tanner graph



Idea: fixed design of code ensemble

- **Long code:** design the code ensemble
- Code ensemble: a set of code with same property.

Example of property: convolutional/Turbo code : weight enumerator

LDPC code : degree/ degree distribution/girth

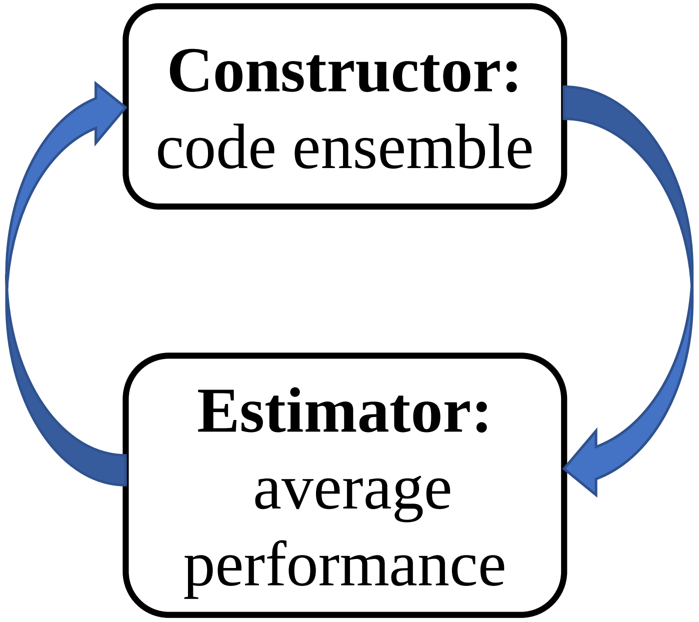
Idea of design code ensemble:

① Random choose one parameter of code ensemble.

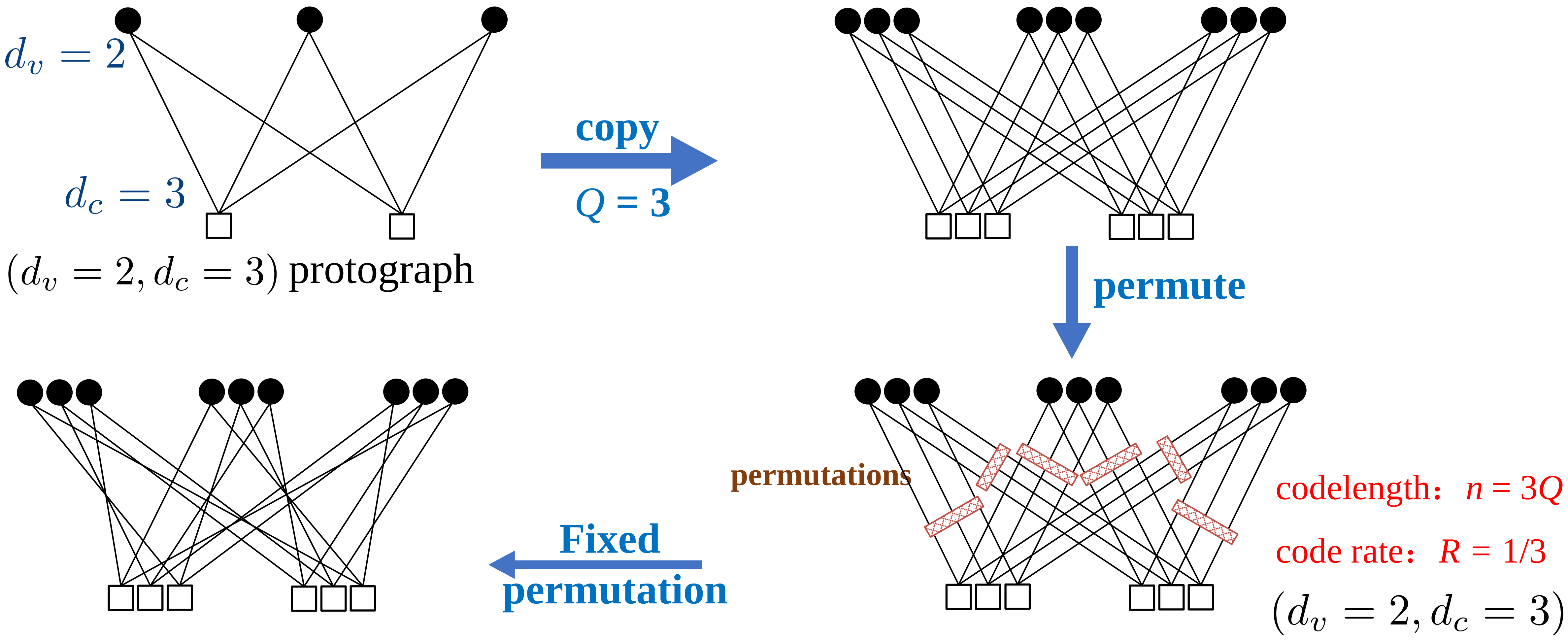
② Estimate the average performance of code ensemble.

③ Iteratively update the parameter of the code ensemble until obtain the excellent performance.

④ Pick a code at random from the ensemble and expect excellent performance.



Example: Regular (d_c, dv)-LDPC Code Ensemble



Tanner graph
(a specific code)

(code ensemble: codes set with all of possible permutations)

Example: Matrix presentation of regular LDPC Code Ensemble

$$\boldsymbol{B} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$(d_v = 2, d_c = 3)$ Base matrix

$$\boldsymbol{B} \xrightarrow[Q=3]{\text{copy}} \boldsymbol{H}_{\mathrm{c}} = \begin{pmatrix} \boldsymbol{I}_3 & \boldsymbol{I}_3 & \boldsymbol{I}_3 \\ \boldsymbol{I}_3 & \boldsymbol{I}_3 & \boldsymbol{I}_3 \end{pmatrix}$$

permute

$$\boldsymbol{H}_{\mathrm{p}} = \begin{pmatrix} \boldsymbol{P}_3^{(00)} & \boldsymbol{P}_3^{(01)} & \boldsymbol{P}_3^{(02)} \\ \boldsymbol{P}_3^{(10)} & \boldsymbol{P}_3^{(11)} & \boldsymbol{P}_3^{(12)} \end{pmatrix}$$

(code ensemble: all of possible permutations)

Fixed permutation

$$\boldsymbol{H} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right)$$

Tanner graph
(a specific code)

codelength： $n = 3Q$

code rate： $R = 1/3$

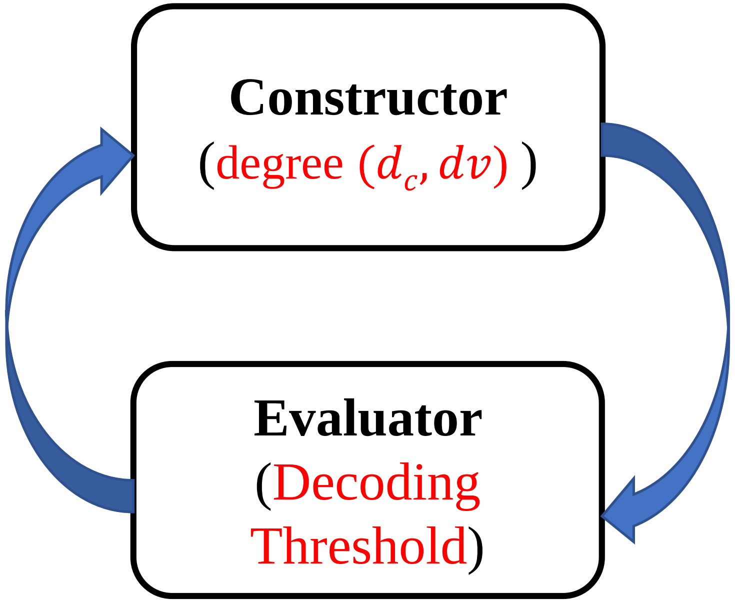
Example: Differential Evolution Algorithm (差分進化法)

① Initiation: Given a degree (d_c, dv) of LDPC code.

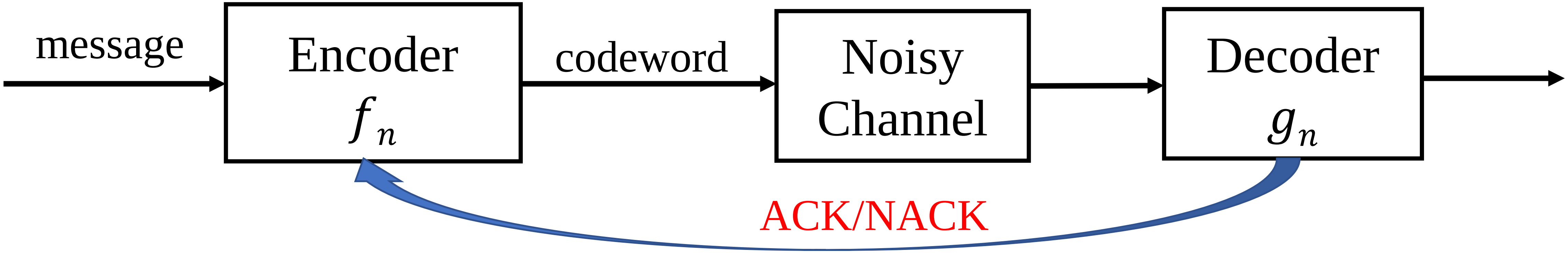
② Estimate the **average performance (**Decoding Threshold**: in terms of the noise standard deviation)** by **density evolution(密度進化)/EXIT chart** over a **code ensemble (d_c, dv)**

③ Update the degree (d_c, dv) **,** find the parameter of code ensemble with excellent performance.

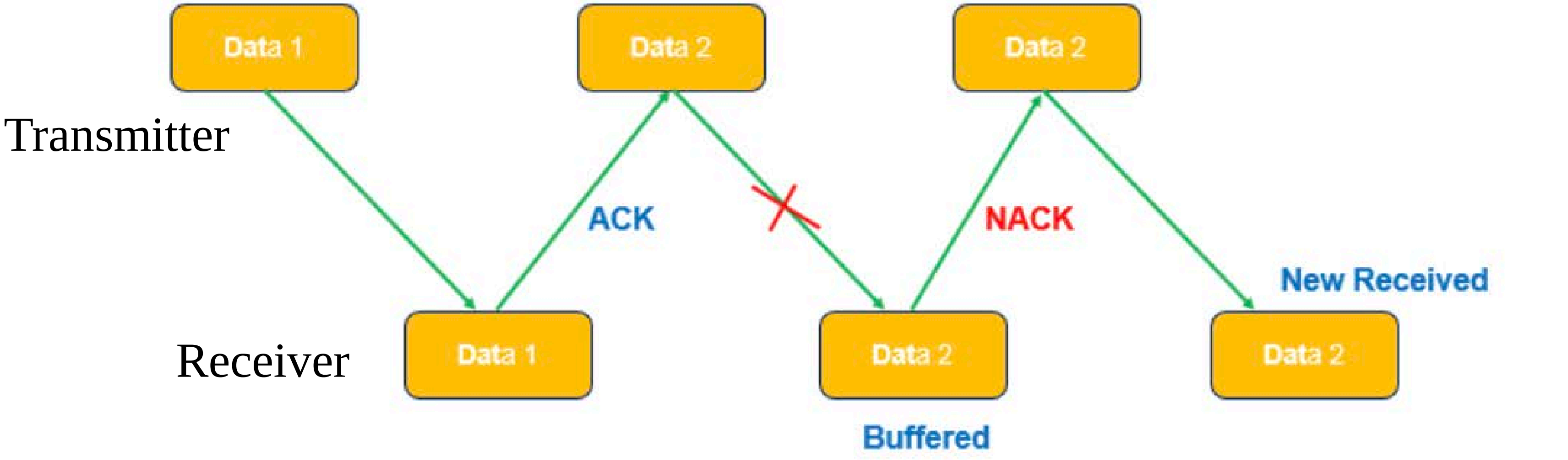
④ Pick a code at random from the ensemble and expect excellent performance.



Idea: Flexible Design of error-correcting code (HARQ)

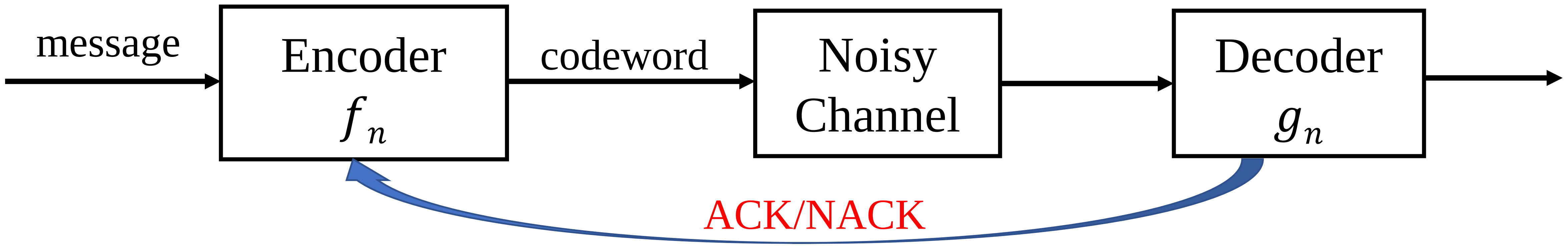


HARQ

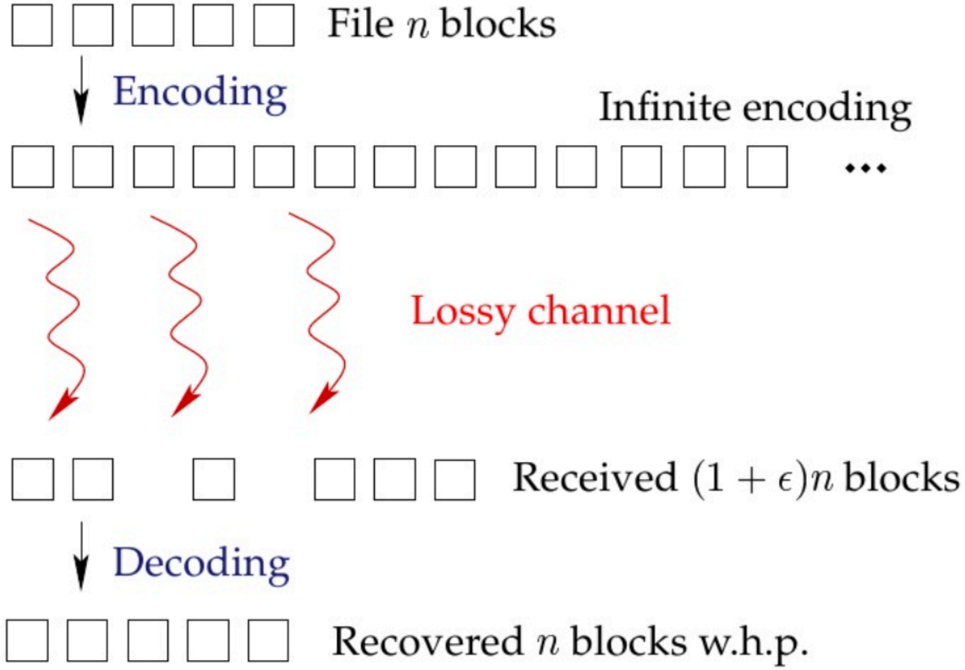


ACK： acknowledgement　肯定応答
NACK: negative-acknowledgement, 否定応答

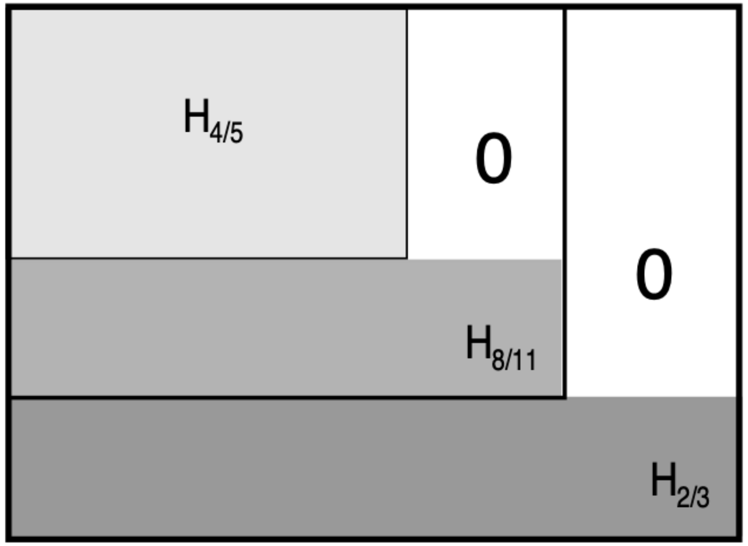
Flexible design: Rateless code/Rate-compatibility code



Rateless code (fountain code)



Parity check matrix of Rate-compatibility code



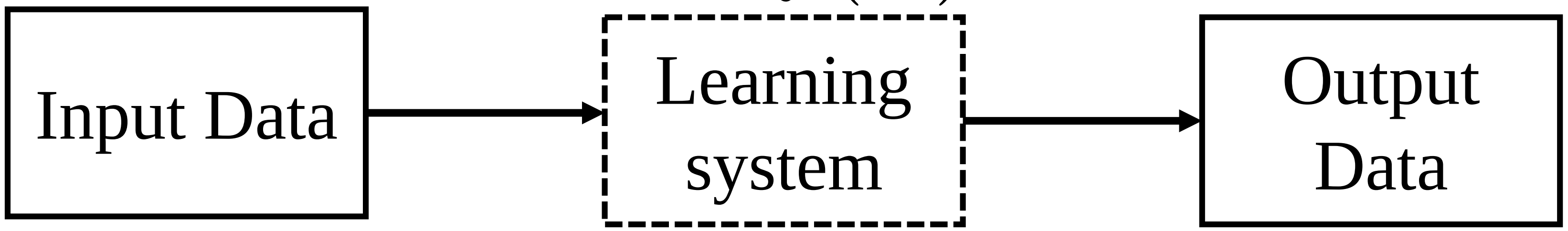
Learning-based approach of designing error-correcting codes

Fixed design:
Supervised Learning (Autoencoder)

Supervised Learning Basic: Training and Testing

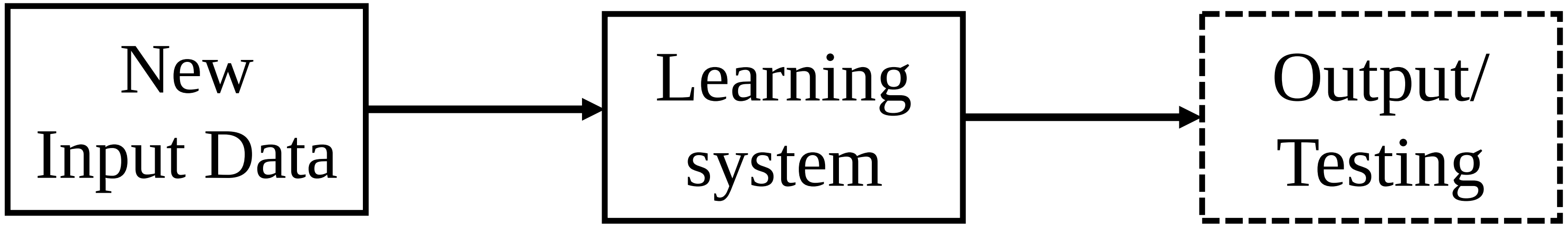
- Training stage: (Learning with a **labeled training set:** input data, desired results)

$$\boldsymbol{x} = f(\boldsymbol{w})$$

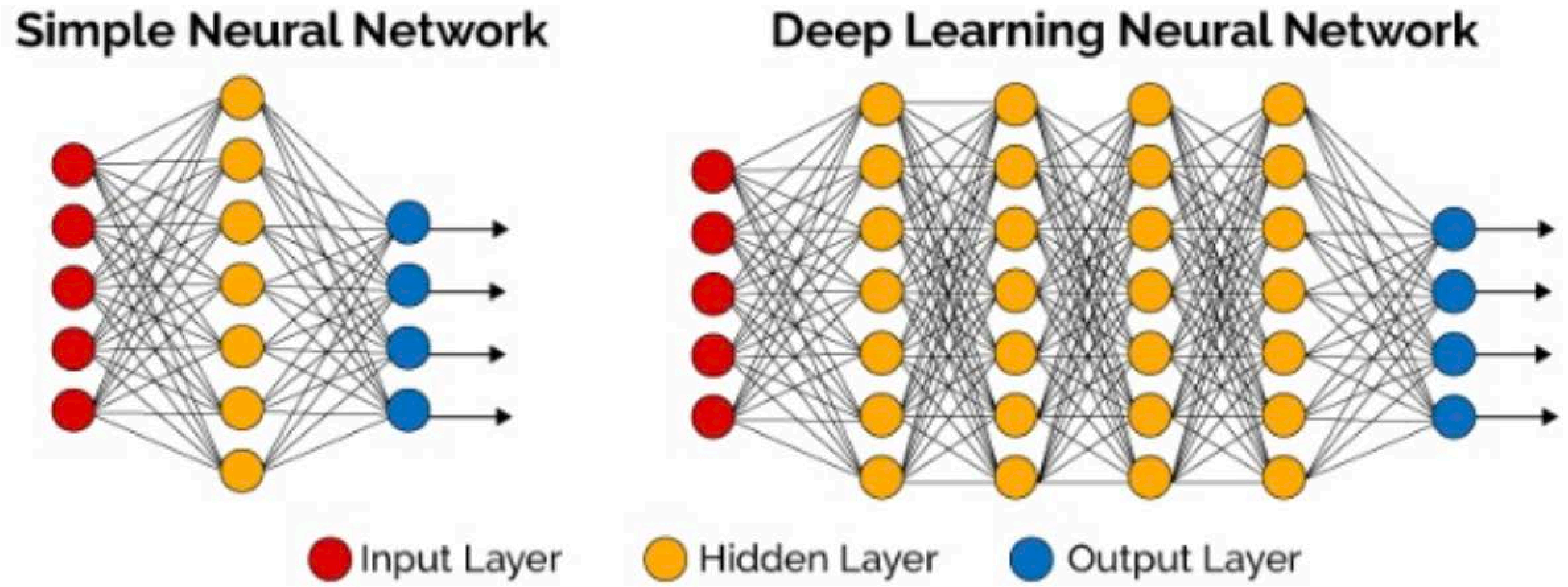


Neural Network(NN)
ニューラルネットワーク

- Testing stage:



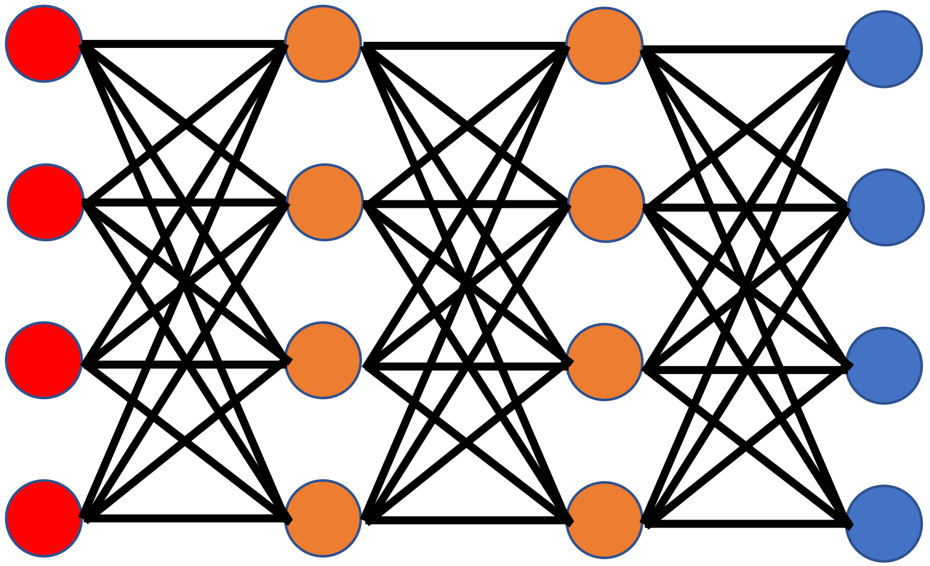
Neural Network



The operation in the neural network

Weights	W_1, b_1	W_2, b_2	W_3, b_3
Activation functions	h_1	h_2	h_3

Input
x



Iteration of "Linear" and "non-linear" operation

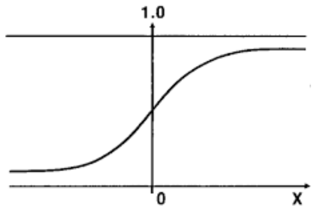
Example: activation functions (活性化関数) (non-linear)

ReLU(Rectified Linear Unit):

$$h(u) = \max\{0, u\}$$

Sigmoid function:

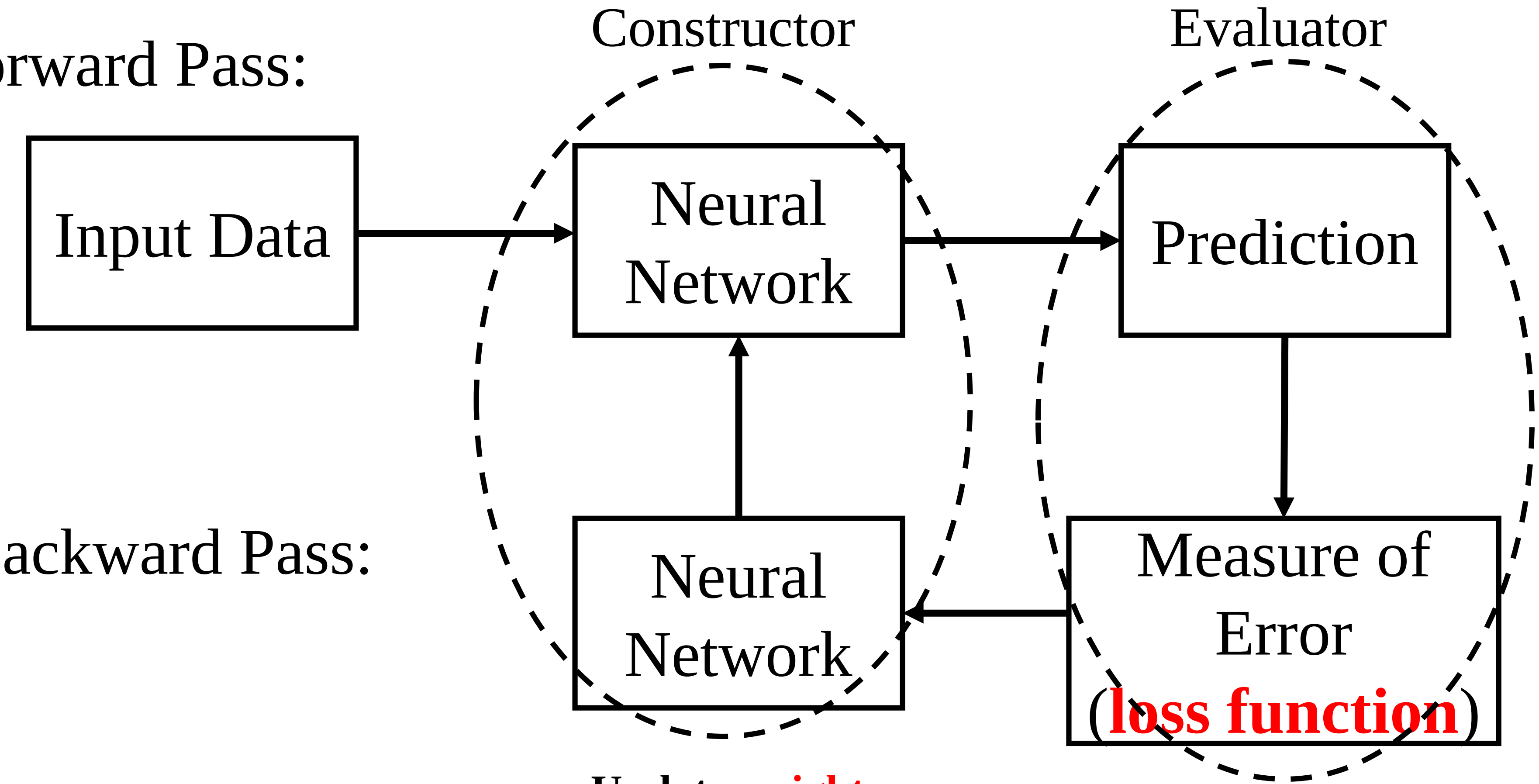
$$h(u) = \frac{1}{1 + e^{-u}}$$



x	$W_1 x + b_1$	$h_1(W_1 x + b_1)$	$W_2 h_1(W_1 x + b_1) + b_2$	$h_2(W_2 h_1(W_1 x + b_1) + b_2)$	……
Input	**Linear Transform**	**Non-Linear Transform**	**Linear Transform**	**Non-Linear Transform**	

How Neural Network Learning: Backpropagation
誤差逆伝播法

- Forward Pass:
- Backward Pass:



Update weights by SGD to decrease loss function

quantifies gap between prediction and desired results.

SGD: stochastic gradient descent
確率的勾配降下法

Loss function

For regression:

Mean Squared Error

$$MSE = \frac{1}{N}\sum(t_i - s_i)^2$$

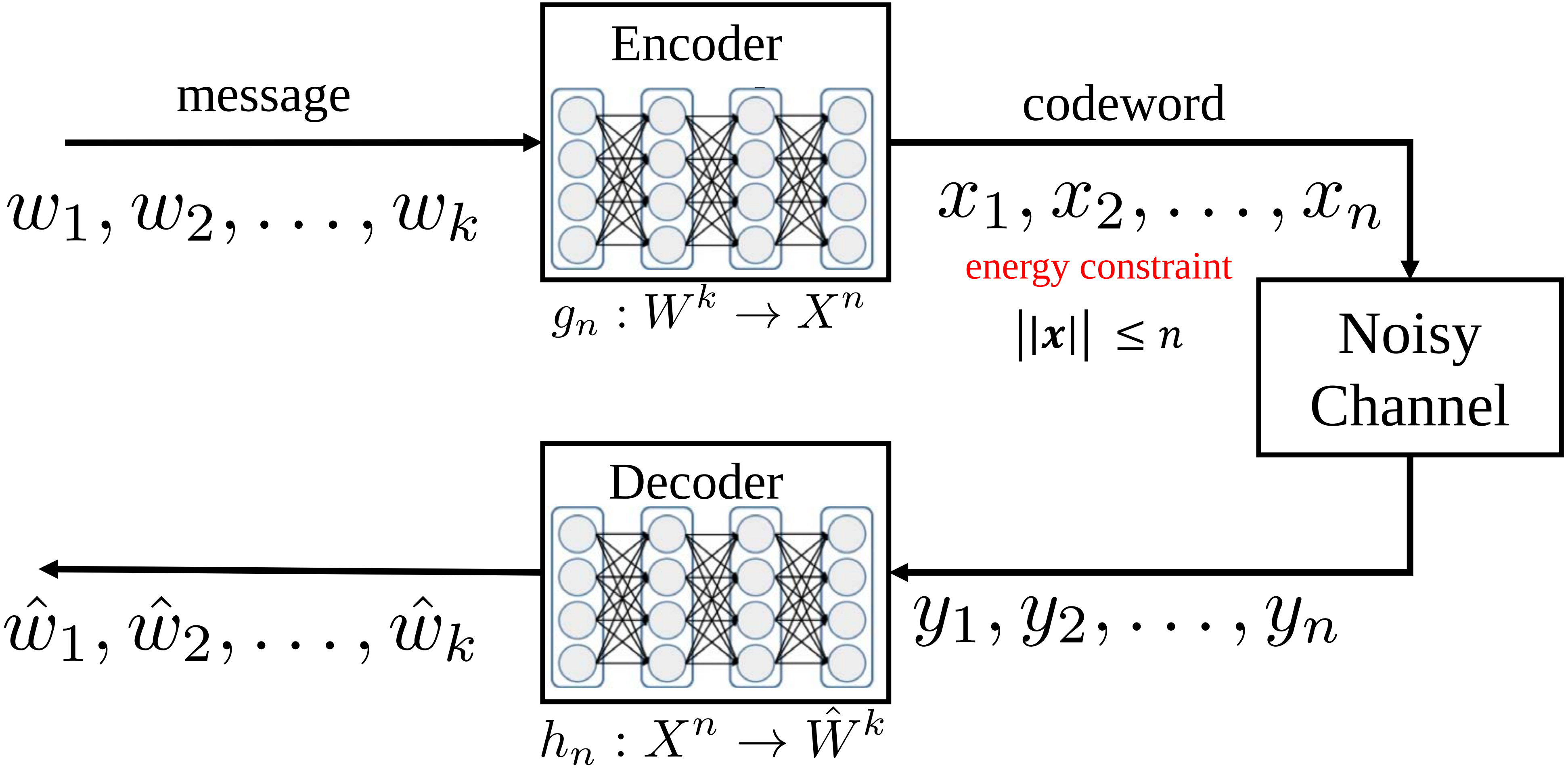
For classification:

Cross Entropy Loss

$$CE = -\sum_{i}^{C} t_i \log(s_i)$$

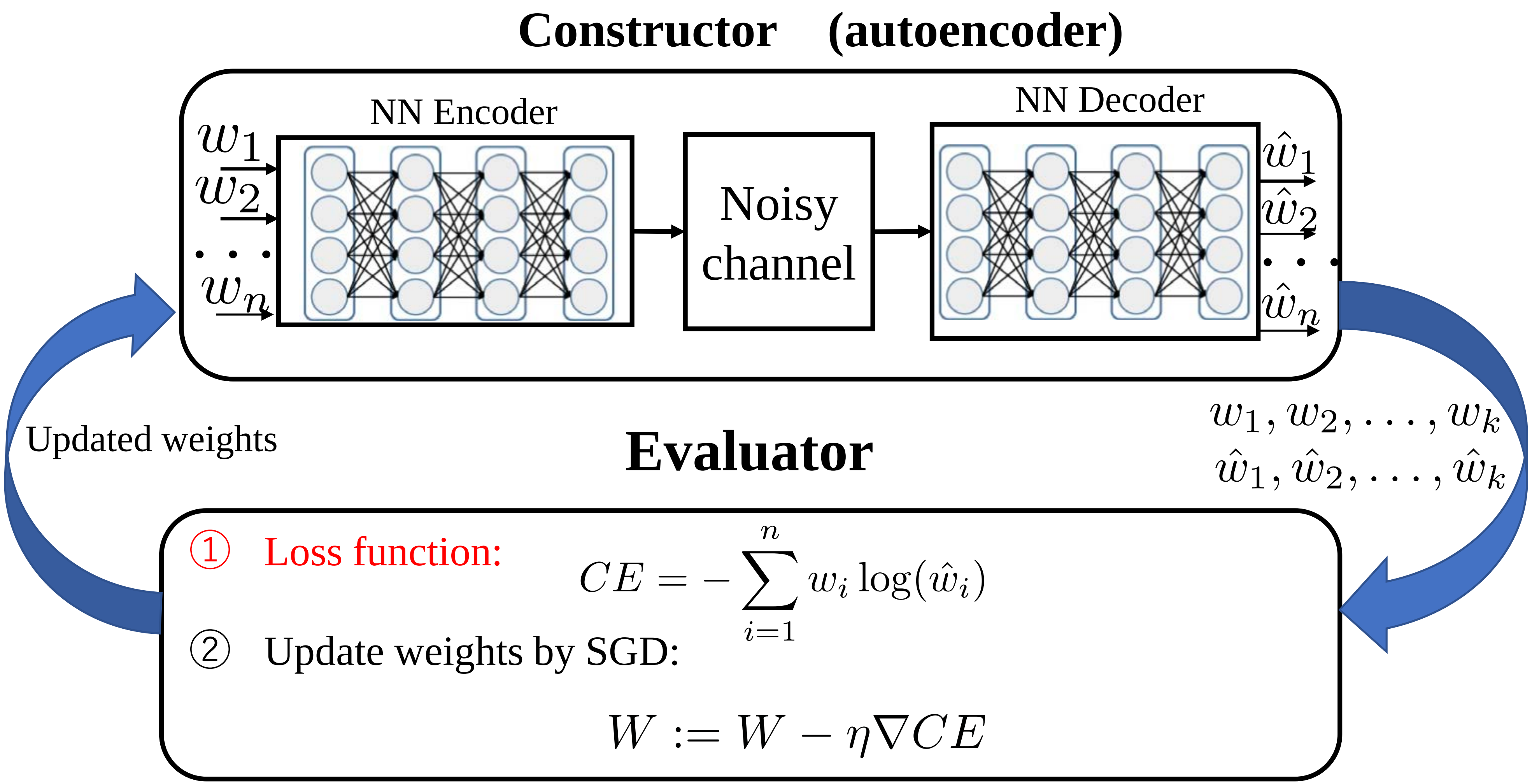
Autoencoders of error-correcting codes system

- An autoencoder is a neural network that is trained to attempt to copy its input to its output.



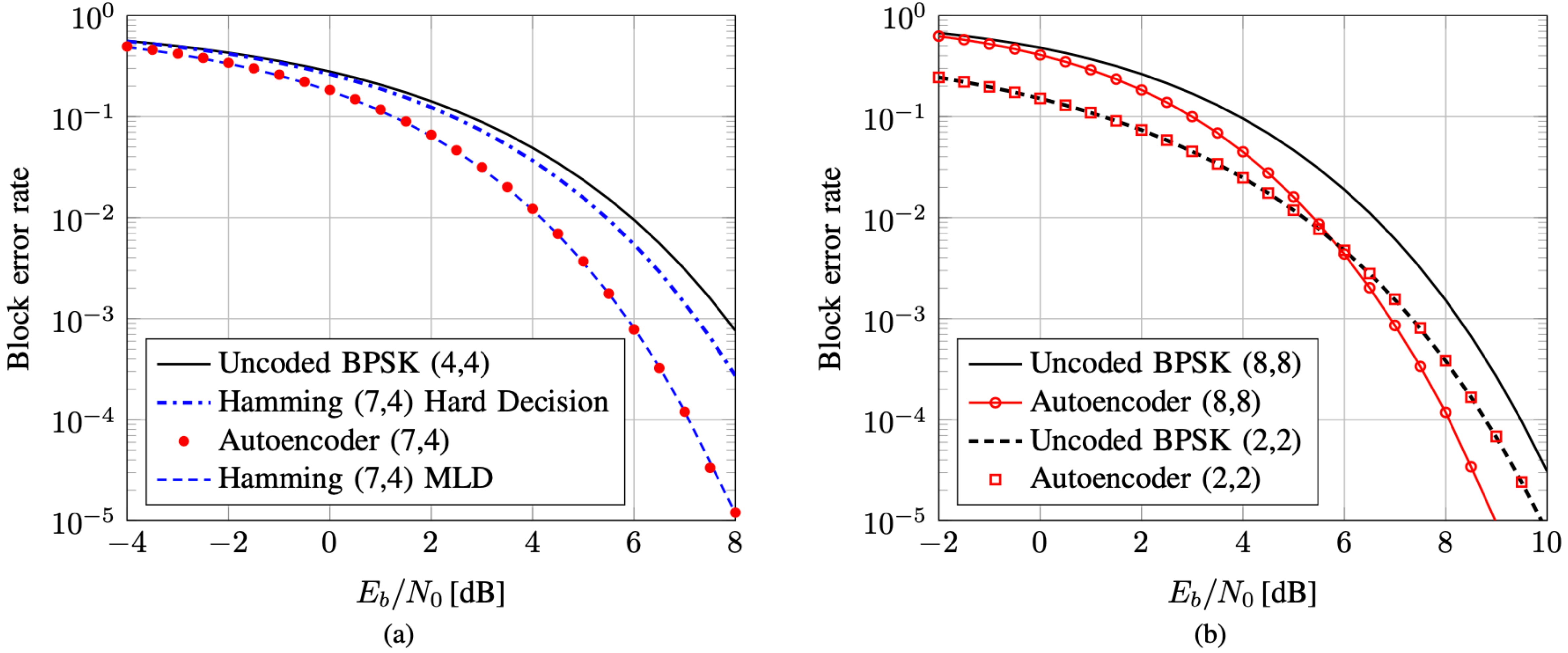
Tim O'Shea, and Jakob Hoydis, "An Introduction to Deep Learning for the Physical Layer", IEEE Transactions on Cognitive Communications and Networking, Vol. 3-4, PP. 563-575, Dec. 2017.

Autoencoders of error-correcting codes system (training process)



BLER vs Eb/N0 for the autoencoder and baseline communication schemes

Training Eb/N0 = 7dB



Tim O'Shea, and Jakob Hoydis, "An Introduction to Deep Learning for the Physical Layer", IEEE Transactions on Cognitive Communications and Networking, Vol. 3-4, PP. 563-575, Dec. 2017.

Autoencoder for NN Error-Correcting Systems

Advantages:
- Possible to design non-linear code with good performance
  (suitable for coding and decoding for multi-access channel)

- Simper design for construction
  (suitable for designing decoding algorithm of the code)

Disadvantage:
- Difficult to design long code
  (try to design the parameters of the code ensemble)

https://mlc.committees.comsoc.org/research-library/

Further design of NN for error-correcting systems

Decoding algorithm

- E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp.119-131, February 2018.
- L. Lugosch and W. J. Gross, "Neural offset min-sum decoding," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2017.
- F. Liang, C. Shen and F. Wu, "An iterative BP-CNN architecture for channel decoding," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 144-159, February 2018.
- W. Xu, X. You, C. Zhang and Y. Be'ery, "Polar decoding on sparse graphs with deep learning," in Proc. IEEE Asilomar Conference on Signal, System, Computers, October 2018.

Coding and decoding for multi-access channel

- L. WEI, S. Lu, H. Kamabe, J. Cheng, "User Identification and Channel Estimation by DNN-Based Decoder on Multiple-Access Channel," to be presented in 2020 IEEE Global Communications Conference.
- S. Takabe, Y. Yamauchi, and T. Wadayama, "Trainable projected gradient detector for sparsely spread code division multiple access," *preprint arXiv:1910.10336*, 2019.
- J. Lin, S. Feng, Z. Yang, Y. Zhang and Y. Zhang, "A novel deep neural network-based approach for sparse code multiple access," *preprint arXiv:1906.03169*, 2019.
- I. Abidi, M. Hizem, I. Ahriz, M. Cherif and R. Bouallegue, "Convolutional neural networks for blind decoding in sparse code multiple access," in *Proc.International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2019.

Joint design of source-channel coding

- Y. M. Saidutta, A. Abdi and F. Fekri, "M to 1 joint source-channel coding of Gaussian sources via dichotomy of the input space based on deep learning," *in Proc. Data Compression Conference (DCC)*, 2019.

Learning-based approach of designing error-correcting codes

Reinforcement learning（強化学習） (Flexible design of error-correcting codes)

Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child’s ?

——Alan Turing

Reinforcement Learning

- Task
  - Learn how to behave successfully to achieve a goal while interacting with **an external environment**
- How to design?

  Reinforcement Learning problems can be modeled by a so-called *Markov Decision Process* (MDP)　(マルコフ決定プロセス)

- Examples
  - Game playing: player knows whether it win or lose, but not know how to move at each step
  - Control: a traffic system can measure the delay of cars, but not know how to decrease it.
  - Error-correcting codes: a system knows how to measure the performance of the codes, but not know how to construct it at each system .

Markov Decision Process (MDP) model

- **State:** s / S: state set
- **Action:** a / A: actions space

Agent

- **Policy**(方針): $\boldsymbol{\pi(a|s)}$

function to map states s to actions a

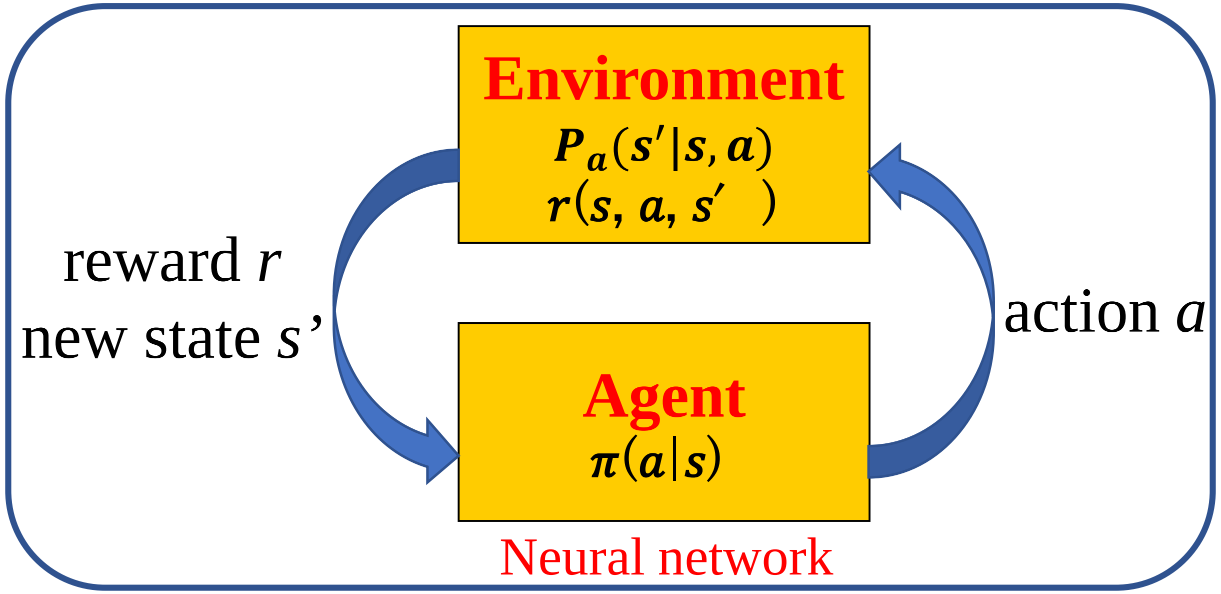
Environment

- $\boldsymbol{P_a(s'|s,a)}$

probability of action a in state s at time t to state s' at time $t+1$.

- $\boldsymbol{r(s, a, s')}$

immediate reward(即時報酬): feedback after transitioning from state s to state s', triggered by action a.



Return (long-run reward長期報酬)

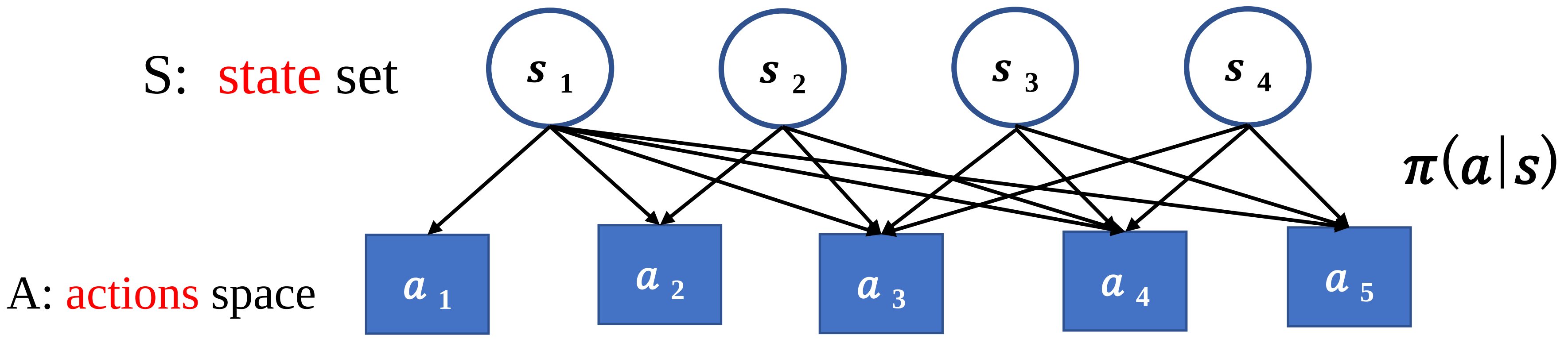
$$G(\gamma) = \sum r(s, a, s')$$

MDP model to reinforcement learning

The agent task:

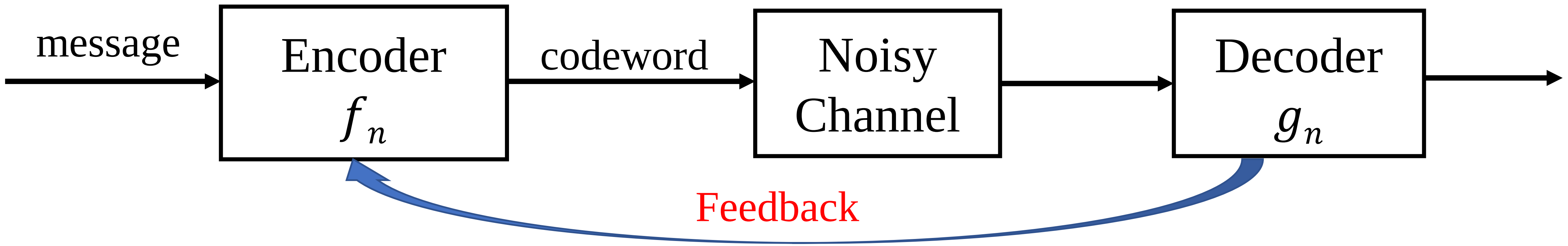
To find an optimal policy $\pi(a|s)$ that maximize Return (long-run reward)

$$G(\gamma) = \sum r(s, a, s')$$



- How to define the **rewards** $r(s, a, s')$ and **return**
- How to **change the policy** based on experience
- How to **change transitions** $p(s'|s, a)$

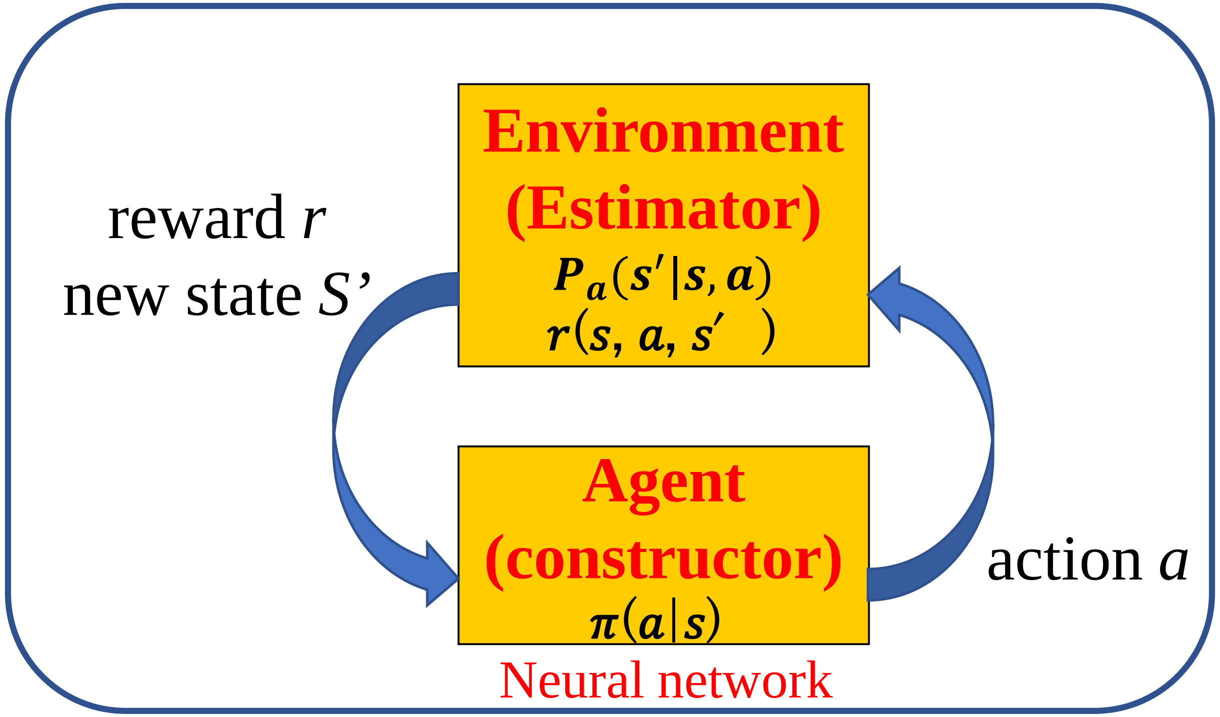
Example of RL for designing error-correcting code



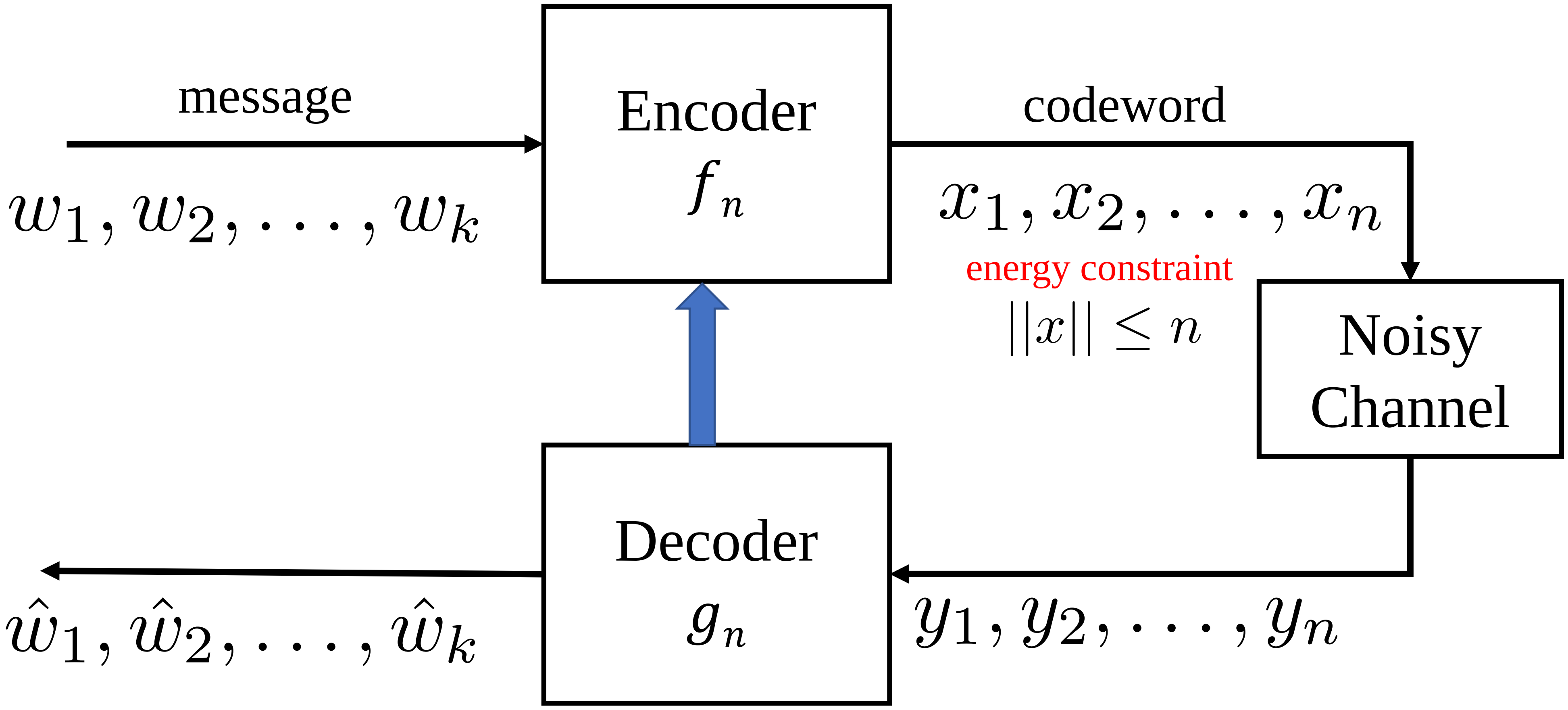
- S (state set): SNR (by step)
- A: actions space: parity check matrix $\boldsymbol{H}$
  (“1”’s positions of P in $\boldsymbol{H}=[\boldsymbol{I}_K, \boldsymbol{P}]$)

- Reward $r(s, a, s')$: BER
- Return: $G(\gamma) = \sum r(s, a, s')$

- $\pi(a|s)$: depend parity check matrix by NN
- $p(s'|s, a)$: depend SNR by NN
- How to change the $p(s'|s, a)/\pi(a|s)$:
  update based on SGD…

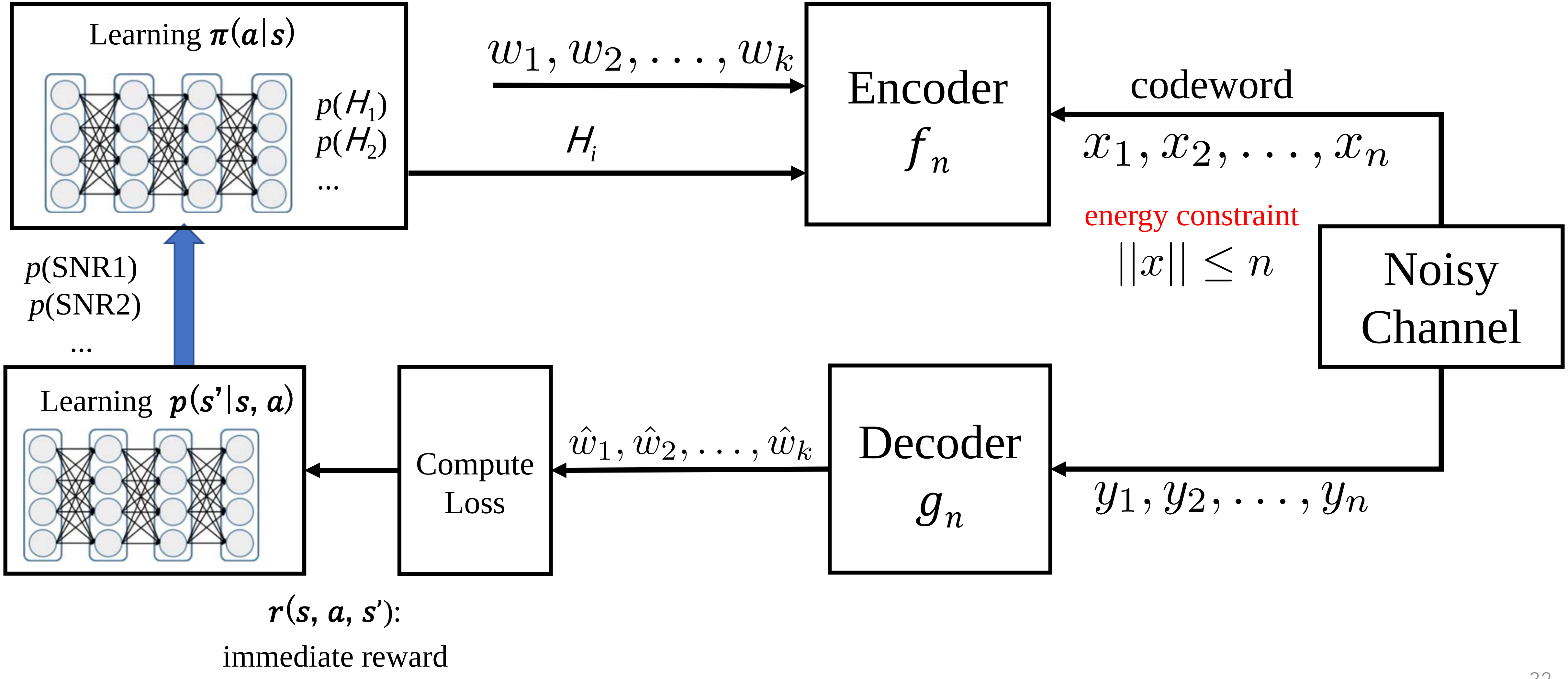


System model of error-correcting codes with feedback



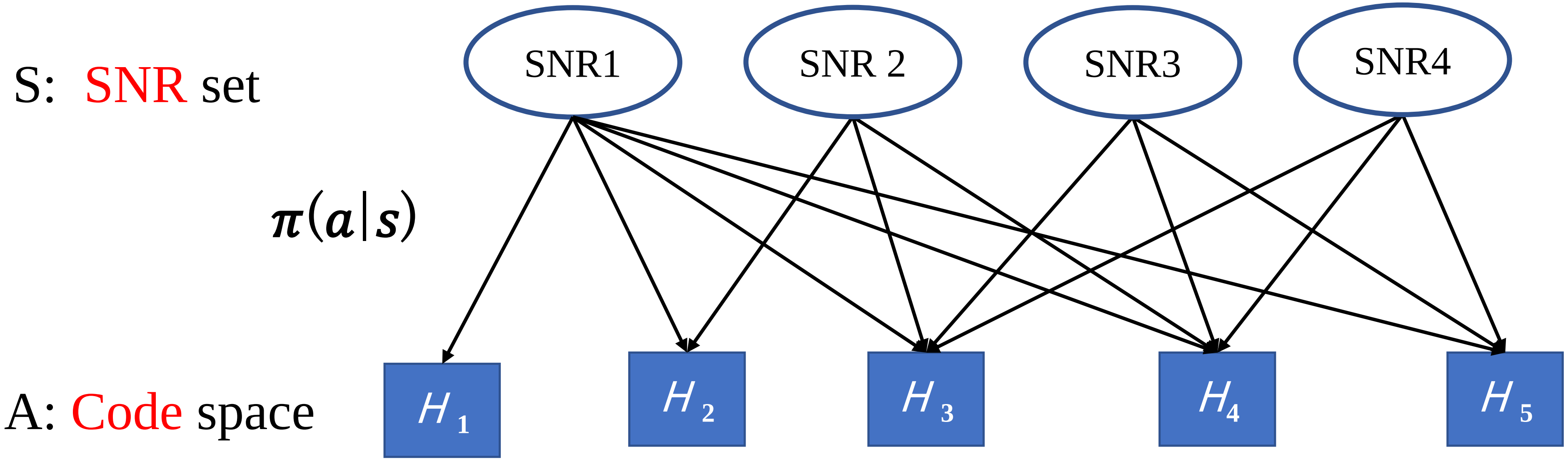
Error-correcting codes RL model (training)

Supervised Learning



Unsupervised Learning system also can be constructed.

Error-correcting codes after training



Extension of RL for error-correcting systems

Advantages:

- Possible to design code corresponding **state of channel information(SCI)**

Polar codes: (such as frozen positions of polar codes)

- L. Huang, H. Zhang, R. Li, Y. Ge and J. Wang, "Reinforcement learning for nested polar code construction," preprint arXiv:1904.07511, 2019
- F. Carpi, C. Häger, M. Martalò, R. Raheli and H. D. Pfister, "Reinforcement learning for channel coding: Learned bit-flipping decoding," in *Proc. 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019.

MIMO system (state of channel information)

- Y.-S. Jeon, J. Li, N. Tavangaran, and H. V. Poor, "Data-Aided Channel Estimator for MIMO Systems via Reinforcement Learning," *preprint arXiv:2003.10084*, 2020.
- Y.-S. Jeon, N. Lee and H. V. Poor, "Robust data detection for MIMO systems with one-bit ADCs: A reinforcement learning approach," preprint arXiv:1903.12546, 2019.
- M. Goutay, F. Ait Aoudia and J. Hoydis, "Deep reinforcement learning autoencoder with noisy feedback," *preprint arXiv:1810.05419*, 2018.

Disadvantage:

- Difficult to design long code

Conclusion

	fixed design	flexible design
Traditional design	Differential evolution algorithm	Rateless code/ Rate-compatibility code
Design by learning • perform universal function approximation • Automatic design	Supervised learning	Reinforcement learning (for feedback channel)

- Code design is a code property optimization problems.

Constructor: code/ code ensemble (G or NN)
Estimator: BER/threshold performance (G or NN)