NOKIA Bell Labs

The Physical Layer as an Autoencoder

Fayçal Ait Aoudia, Nokia Bell Labs

EURECOM

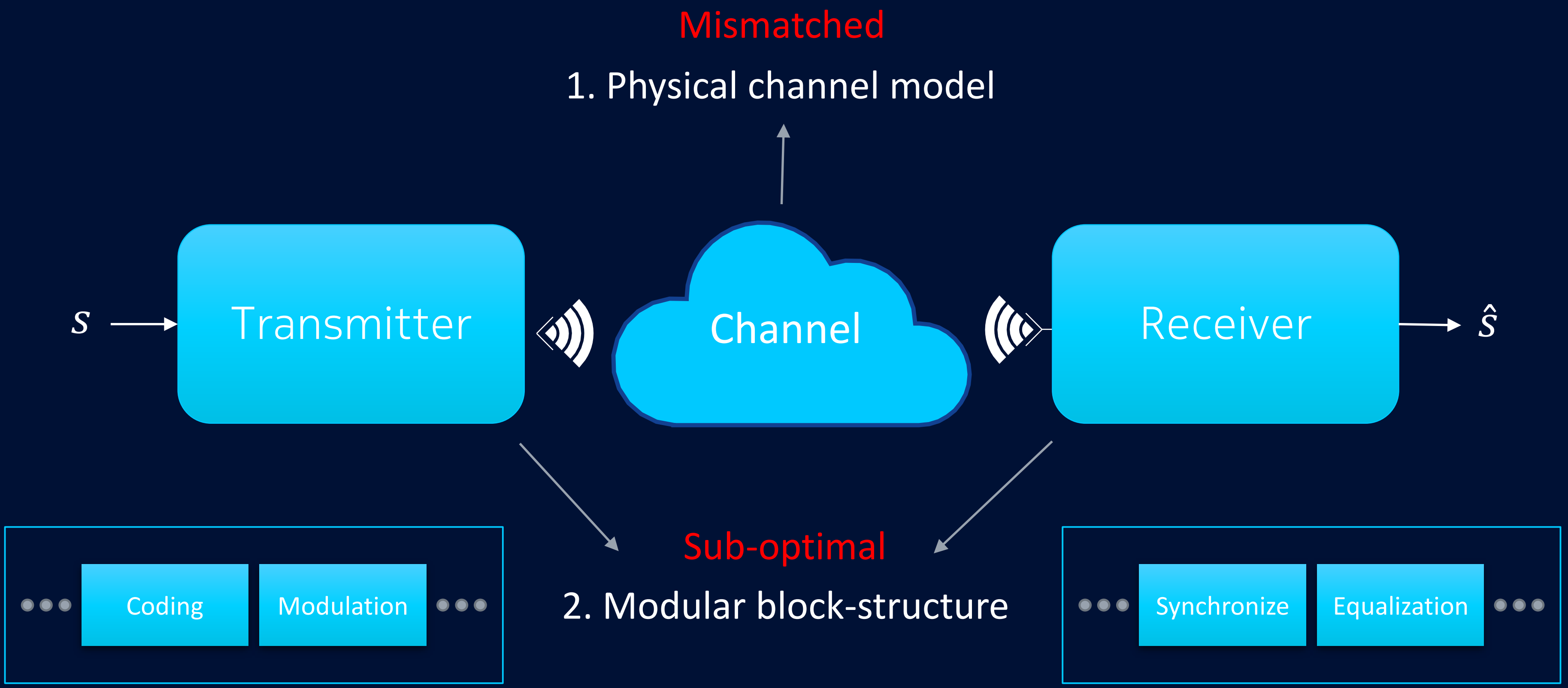
November 4th, 2019

The communication problem

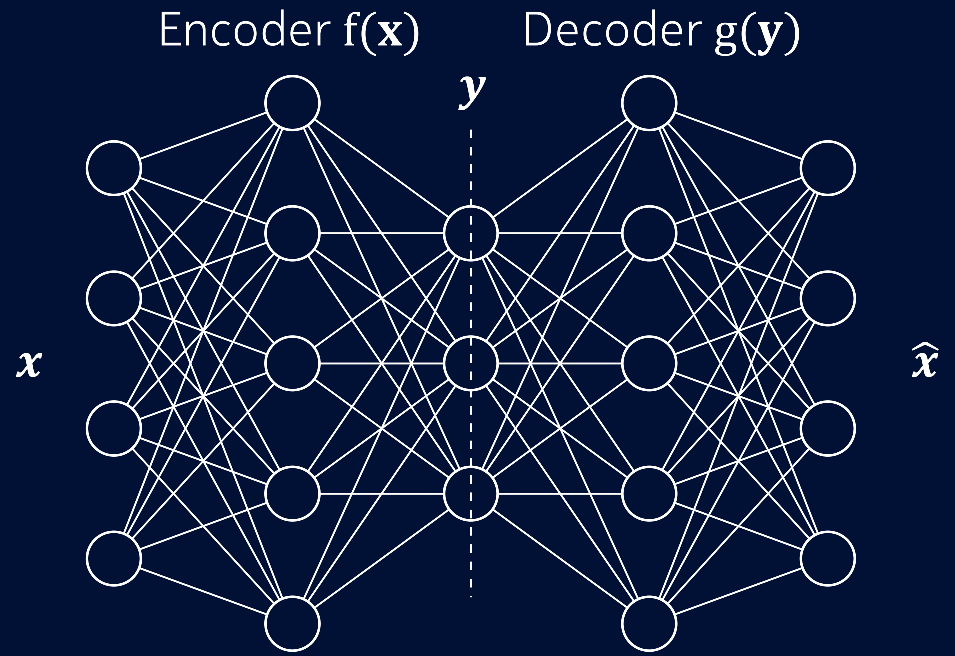
Goal: Minimize $Pr(\hat{s} \neq s)$

- $s \in \mathcal{M} = \{1, \cdots, M\}$, $\mathrm{k} = \log_2 M$
- $\boldsymbol{x} \in \mathbb{C}^n$ with $\mathrm{E}[||\boldsymbol{x}||^2] \leq n$
- $\boldsymbol{y} \in \mathbb{C}^n \sim p(\boldsymbol{y}|\boldsymbol{x})$
- $\hat{s} \in \mathcal{M}$
- $R = \frac{k}{n}$ bits/channel use

How we have solved the problem until now

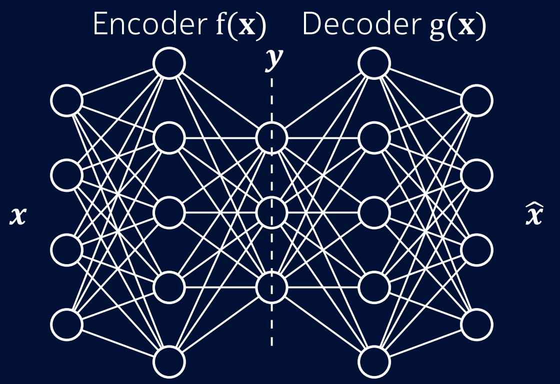


Primer on Autoencoders



Find a useful representation $\boldsymbol{y} \in \mathbb{R}^{n}$ of $\boldsymbol{x} \in \mathbb{R}^{r}$ at some intermediate layer through learning to reproduce the input at the output

Autoencoder terminology

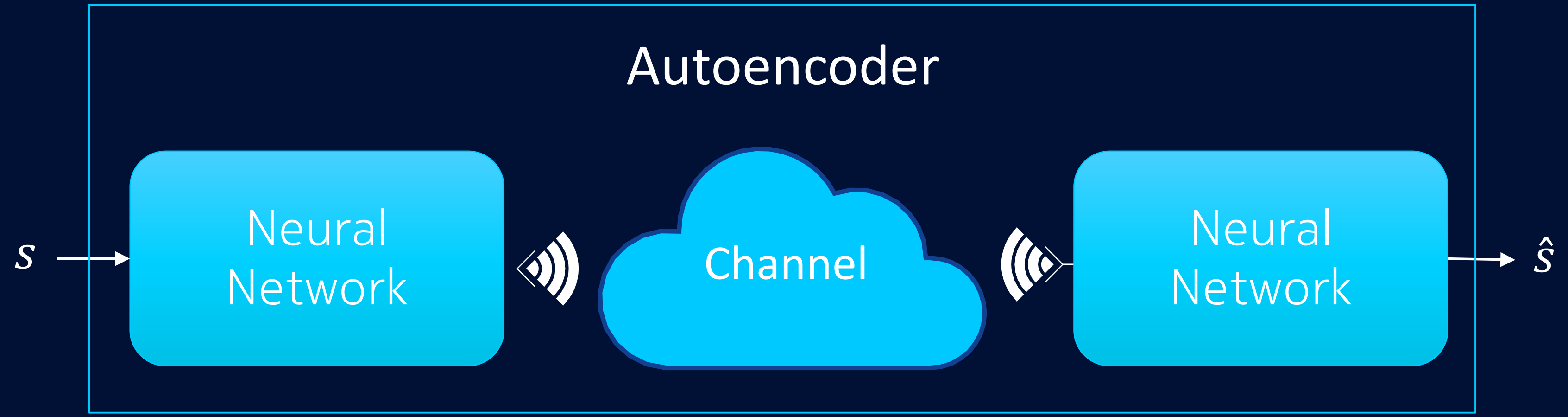


- Encoder and decoder are separated by a **penalty** which is either a dimensionality constraint or regularization

- **Incomplete** autoencoder ($n < r$):
  - Capture only the most important features of x
  - Typically used for compression/dimensionality reduction

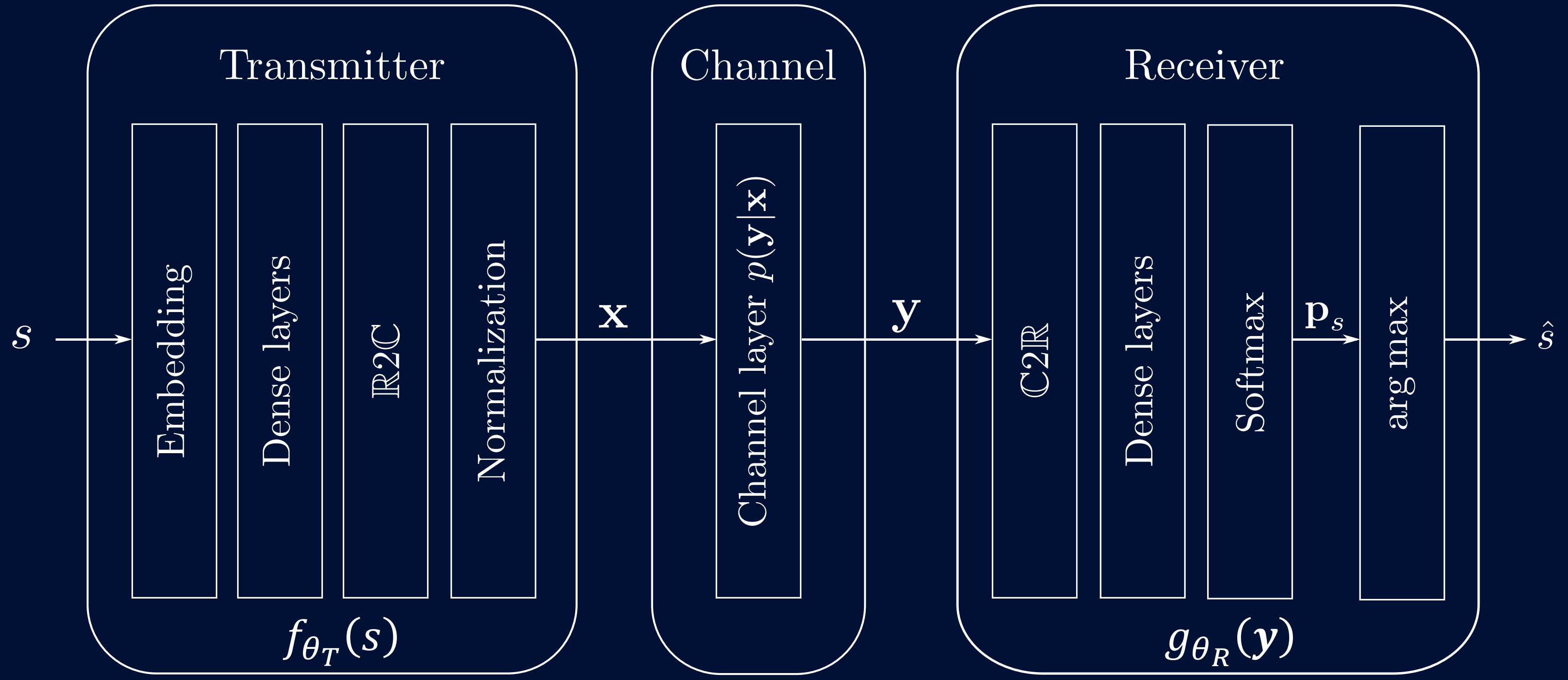
- **Overcomplete** autoencoder ($n \geq r$):
  - Could learn the identity function (regularization can avoid this)
  - Adds some form of redundancy to $\boldsymbol{y}$

Key idea: Communication seen as an autoencoder



- Learns a robust message representation
- Trainable from end-to-end to minimize $Pr(\hat{s} \neq s)$
- Universal concept which applies to any channel $p(\boldsymbol{y}|\boldsymbol{x})$

Neural network structure for a simple channel model



arxiv:1702.00832

Embeddings

- **Embeddings** map integers to vectors, i.e., essentially, a lookup table that returns columns $s \in \mathcal{M}$ of matrix $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_M]$

- Simply a more efficient implementation of a dense layer with **one-hot** encoded inputs:

$$\boldsymbol{W} \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} = \boldsymbol{w}_s$$

- $\boldsymbol{W}$ is trainable like the weight matrix of a dense layer

How to deal with complex values?

- In communications, we typically deal with complex numbers, but most deep learning libraries work with real numbers
- Obtain real-valued representations through the transformations:

$$\mathbb{R}2\mathbb{C}\colon \mathbb{R}^n \mapsto \mathbb{C}^{n/2} \qquad \mathbb{R}2\mathbb{C}(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ \vdots \\ x_{\frac{n}{2}-1} \\ \vdots \\ x_{n-1} \end{bmatrix} + j \begin{bmatrix} x_2 \\ \vdots \\ x_{\frac{n}{2}} \\ \vdots \\ x_n \end{bmatrix}$$

$$\mathbb{C}2\mathbb{R}\colon \mathbb{C}^{n/2} \mapsto \mathbb{R}^n \qquad \mathbb{R}2\mathbb{C}(\boldsymbol{x}) = \begin{bmatrix} \Re(\boldsymbol{x}) \\ \Im(\boldsymbol{x}) \end{bmatrix}$$

- Extensions to complex neural networks exist, e.g., https://arxiv.org/abs/1705.09792, but gains unclear, ongoing reserach

Normalization layer

- Normalization is necessary to ensure that constraints on $\boldsymbol{x}$ are met
- Can be seen as a neural network layer without any trainable parameters, i.e., a differentiable operation
- Instantaneous normalization: $\frac{\boldsymbol{x}}{|\boldsymbol{x}|}$
- Constraint on symbol amplitude: $\min(\max(x_i, x_{min}), x_{max})$
- Average power normalization:

$$\frac{\boldsymbol{x}(s)}{\sqrt{\frac{1}{\frac{M}{2}} \sum_{s=1}^{M} \|\boldsymbol{x}(s)\|^2}} \approx \frac{\boldsymbol{x}(s)}{\sqrt{\frac{1}{\frac{N}{2}} \sum_{i=1}^{N} \|\boldsymbol{x}_i\|^2}}$$

- Even (pseudo) quantization of $\boldsymbol{x}$ can be done

Channel layer

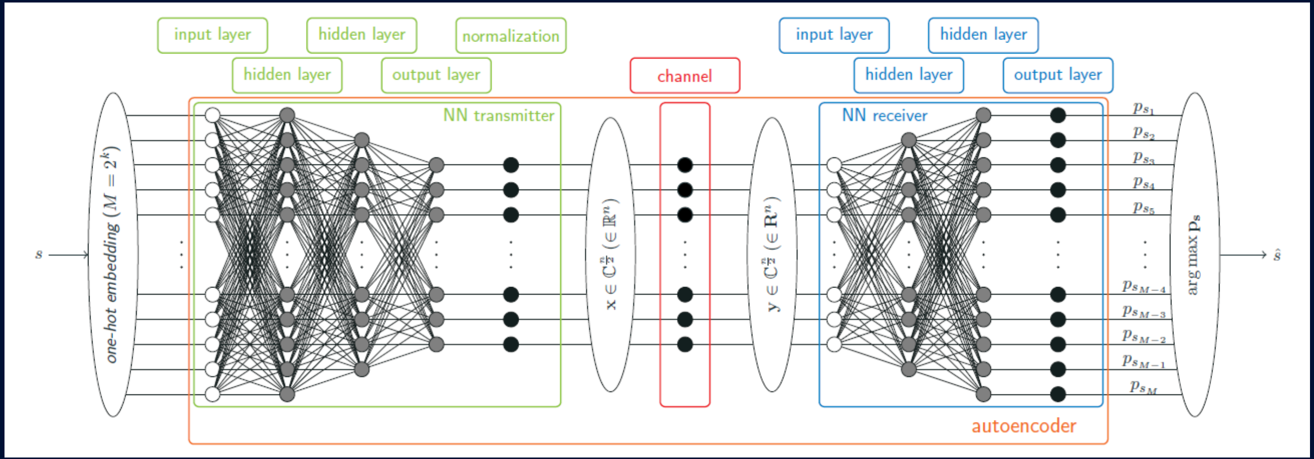
- We require a differentiable generative model for $p(\boldsymbol{y}|\boldsymbol{x})$, i.e.,
  $\nabla_{\boldsymbol{x}} y_i \ \forall i$ must be known

- No trainable parameters, stochastic transformation of the input

- Autoencoder penalty layer: e.g., regularization by adding noise
  →Encoder is forced to learn robust message representations

- Examples:
  - Additive white Gaussian noise channel: $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n}$
    $\nabla_{\boldsymbol{x}} y_i = \mathbf{1}$
  - Memoryless fading channel: $\boldsymbol{y} = \mathrm{h}\boldsymbol{x} + \boldsymbol{n}$
    $\nabla_{\boldsymbol{x}} y_i = \mathrm{h}\mathbf{1}$
  - Multi-tap fading channel: $y_i = \sum_{l=1}^{L} h_l x_{i-l+1} + n_i$
    $\nabla_{\boldsymbol{x}} y_i = [\cdots \quad 0 \quad h_L \quad \cdots \quad h_1 \quad \cdots]$

End-to-end training

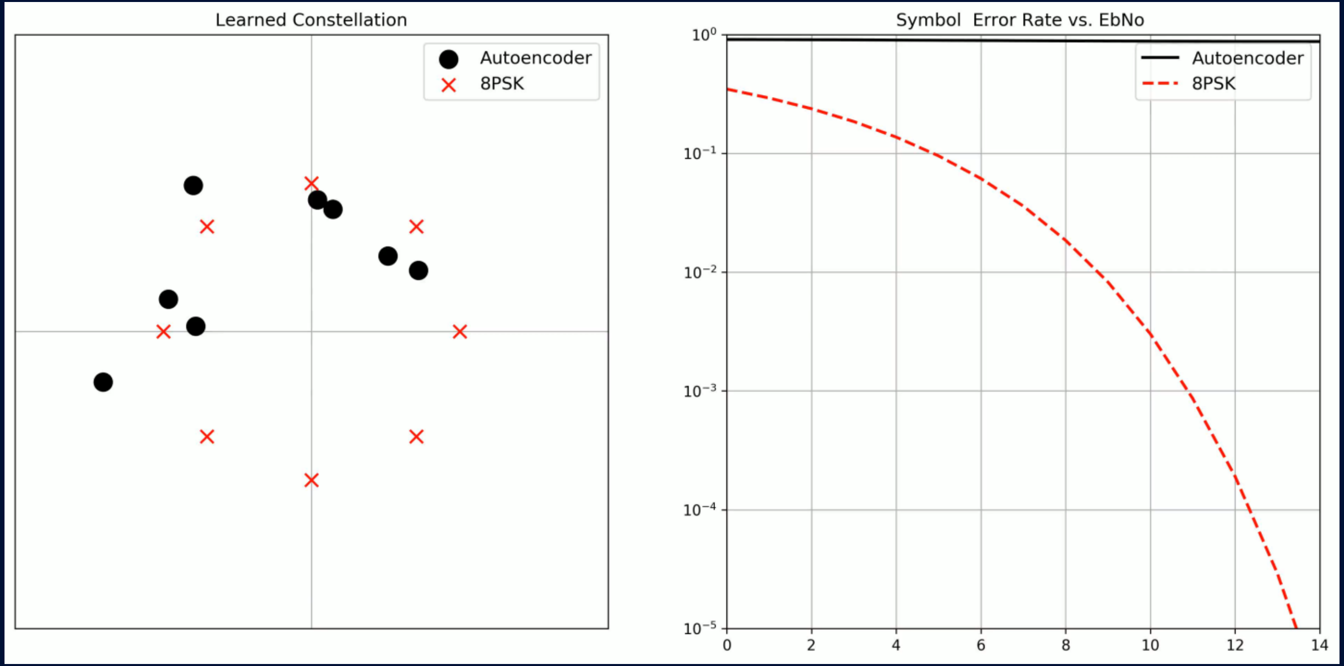


Training process:

- Classification task: (categorical) cross-entropy loss
- Channel model $p(\boldsymbol{x}|\boldsymbol{y})$ is…
  - stochastic: infinite amount of training data
  - differentiable: gradient can be computed through the channel

$\mapsto$ SGD can optimize transmitter and receiver jointly!

Learning process over an AWGN channel: $p(\boldsymbol{y}|\boldsymbol{x}) = CN(\boldsymbol{x}, \sigma^2 \boldsymbol{I})$



Compare with Fig.7 (c) G. Foschini et al. "Optimization of Two-Dimensional Signal Constellations in the Presence of Gaussian Noise," *IEEE Trans. Commun.*, 1974.

Try it yourself on Google Collab

Information theory perspective

Loss function is the symbolwise cross-entropy:

$$\min_{\theta_M,\theta_D} \mathcal{L}(\theta_M,\theta_D) := \min_{\theta_M,\theta_D} \mathbb{E}_{s,y}\left[-\log\left(\tilde{p}_{\theta_D}(s|y)\right)\right], y\sim p\left(y|f_{\theta_M}(s)\right)$$

$$\approx \min_{\theta_M,\theta_D} \frac{1}{B}\sum_{i=1}^{B} -\log\left(\tilde{p}_{\theta_D}\left(s^{(i)}|y^{(i)}\right)\right)$$

$$= \min_{\theta_M,\theta_D} H(X) - \underbrace{I_{\theta_M}(X;Y)}_{\text{Maximize the mutual information}} + \underbrace{\mathbb{E}_y\left[D_{\mathrm{KL}}\left(p_{\theta_M}(s|y)\|\tilde{p}_{\theta_D}(s|y)\right)\right]}_{\text{Minimize KL divergence to posterior}}$$

Batch-size

Shannon's Theorem: $C = \max_{p(X)} I(X;Y)$

Information theory perspective

We want to transmit bits !

$$I(X;Y) = I(\boldsymbol{B};Y) = \sum_{i=1}^{m} H(B_i | B_{i-1}, \dots, B_1) + \sum_{i=1}^{m} H(B_i | Y, B_{i-1}, \dots, B_1)$$

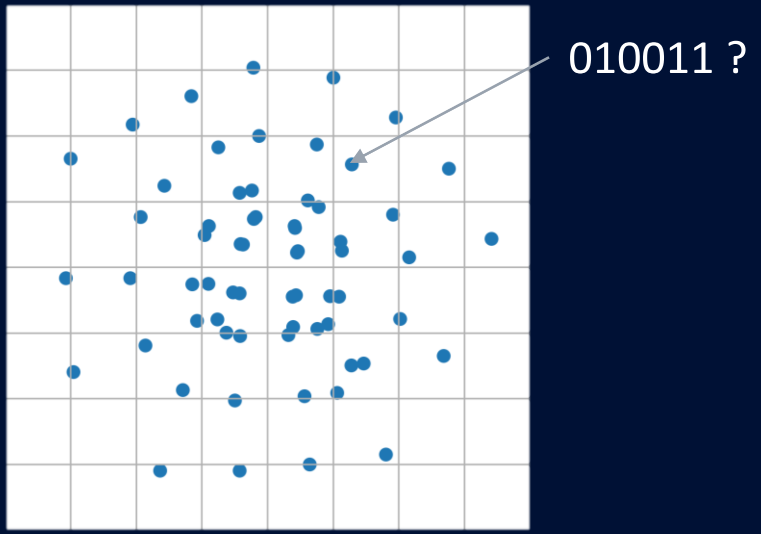
Maximizing $I(X;Y)$ requires:

- Multilevel coding at the transmitter
- Multistage decoding at the receiver

Not practical !

arxiv:1502.02733

The labelling problem



How to optimally label constellation points with bits?

Information theory perspective

$$I(X;Y) = I(\boldsymbol{B};Y) = \sum_{i=1}^{m} H(B_i|B_{i-1},\ldots,B_1) - \sum_{i=1}^{m} H(B_i|Y,\cancel{B_{i-1},\ldots,}B_1)$$

$$R := H(\boldsymbol{B}) - \sum_{i=1}^{m} H(B_i|Y)$$

R achievable using:

- Bit interleaved coded modulation at the transmitter
- Bit-metric decoding at the receiver

Practical and widely used !

arxiv:1410.8075

Information theory perspective

We want to maximize $R \coloneqq H(\boldsymbol{B}) + \sum_{i=1}^{m} H(B_i|Y)$

R is closely related to the total binary cross-entropy:

$$\min_{\theta_M,\theta_D} \mathcal{J}(\theta_M,\theta_D) \coloneqq \min_{\theta_M,\theta_D} \sum_{i=1}^{m} \mathbb{E}_{b_i,y}\left[-\log\left(\tilde{p}_{\theta_D}(b_i|y)\right)\right], y\sim p\left(y|f_{\theta_M}(\boldsymbol{b})\right)$$

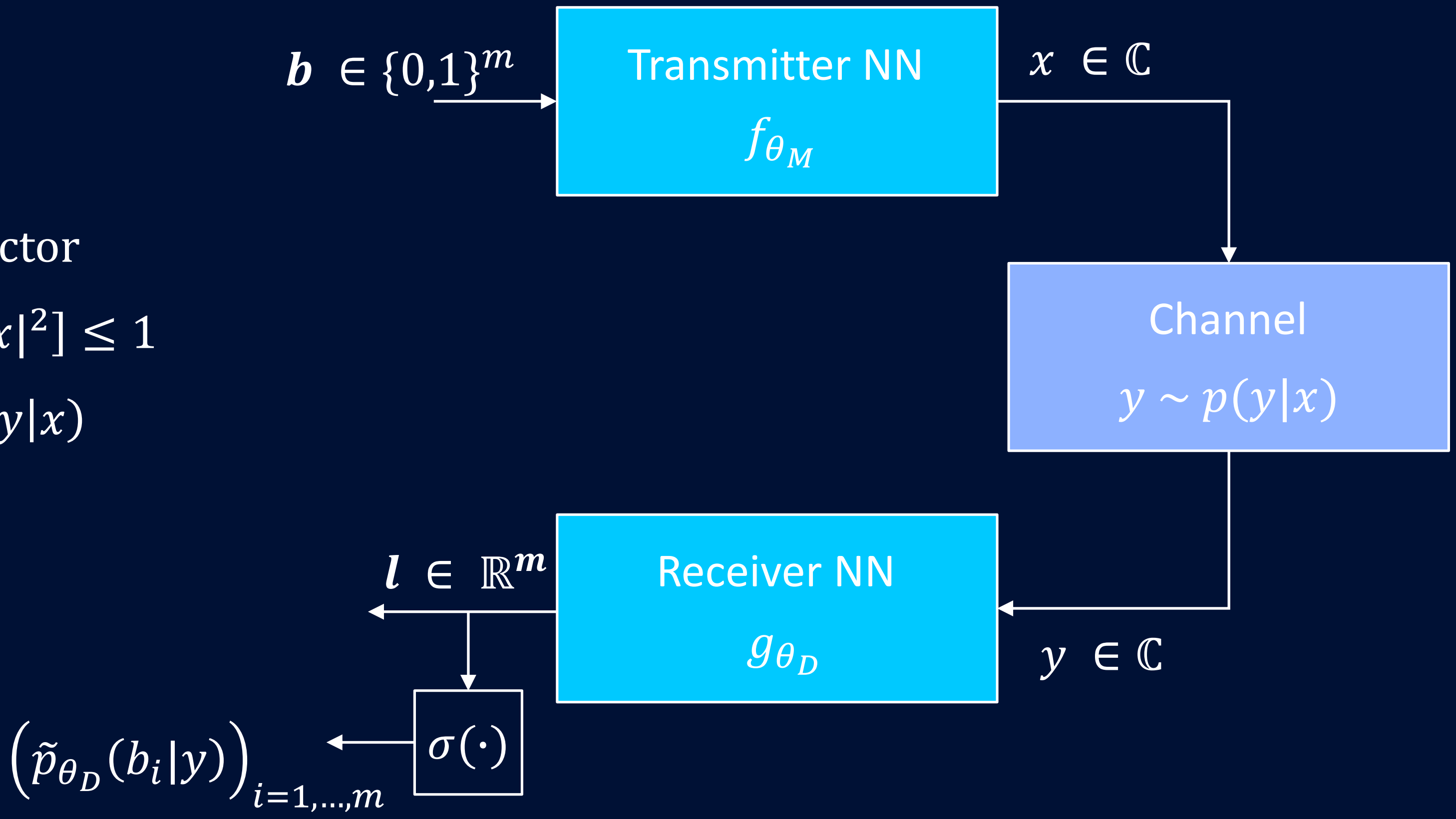
$$= \min_{\theta_M,\theta_D} H(\boldsymbol{B}) - R + \sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{y}}\left[\mathrm{D}_{\mathrm{KL}}\left(p_{\theta_M}(b_i|y)\|\tilde{p}_{\theta_D}(b_i|y)\right)\right]$$

Maximize R

Minimize
KL divergence
to posterior

Bitwise autoencoder

- $\boldsymbol{b}$: m-bit vector
- $x \in \mathbb{C}$, $\mathbb{E}[|x|^2] \leq 1$
- $y \in \mathbb{C} \sim p(y|x)$
- $\boldsymbol{l}$: LLRs

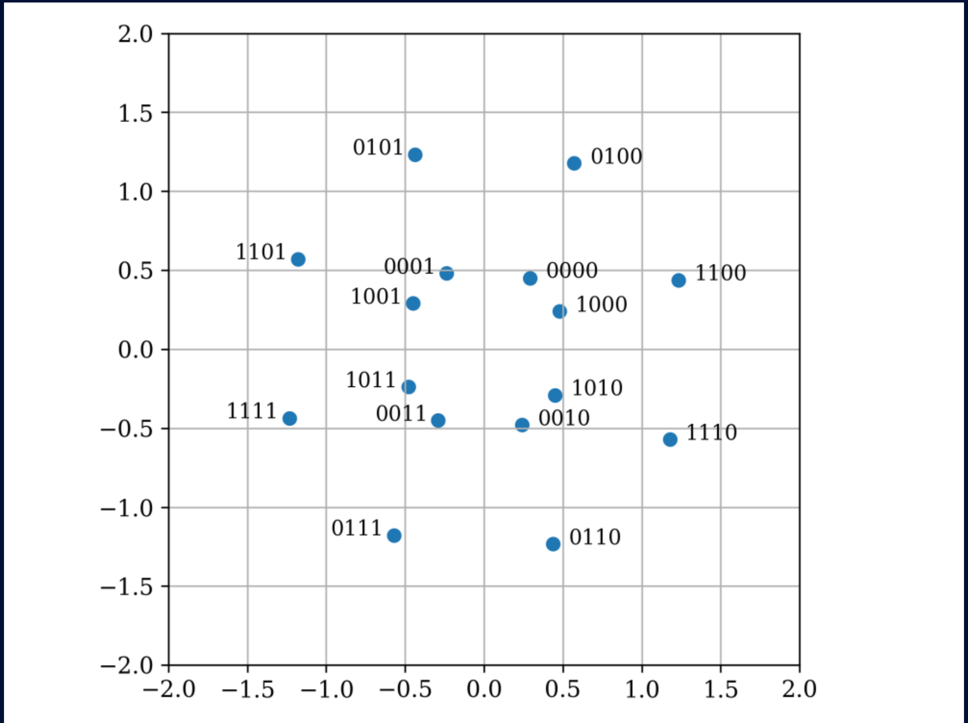


Train in practice using an estimate of the total binary cross-entropy:

$$\mathcal{J}(\theta_M, \theta_D)$$
$$\approx -\frac{1}{B}\sum_{j=1}^{B}\sum_{i=1}^{m}\left(b_i^{(j)}\log\left(\tilde{p}_{\theta_D}\left(b_i^{(j)}|y^{(j)}\right)\right) + \left(1 - b_i^{(j)}\right)\log\left(1 - \tilde{p}_{\theta_D}\left(b_i^{(j)}|y^{(j)}\right)\right)\right)$$

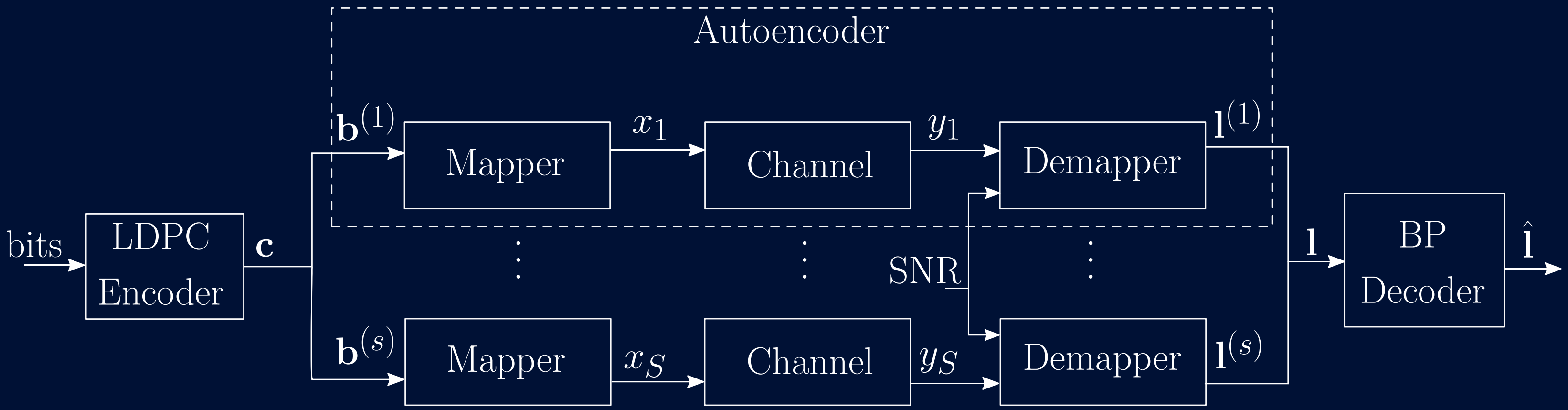
Joint optimization of the constellation geometry and labelling

Constellation learned by the bitwise autoencoder



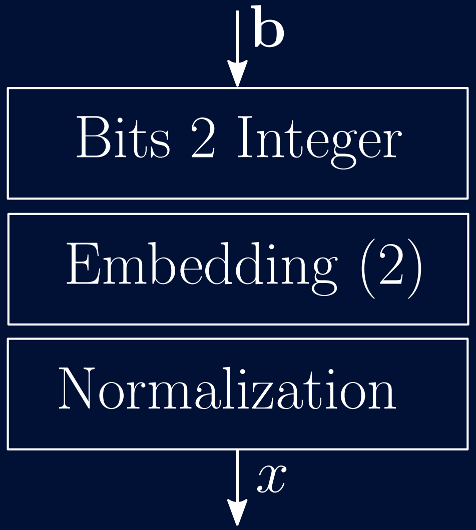
Learned constellation and its corresponding labelling
AWGN channel with $m = 4$

Integration with ECC

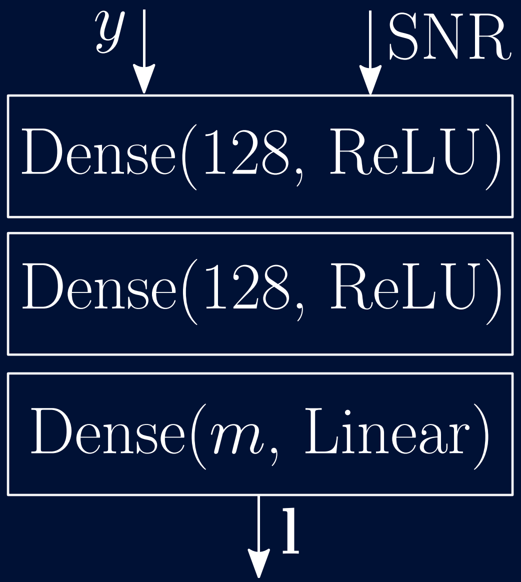


- Mapper and Demapper implemented as NN
- AWGN channel
- SNR fed to the demapper
- Standard IEEE 802.11n LDPC code. Rate = 0.5, length = 1296 bit
- Belief-propagation decoding with 15 iterations

Evaluation setup



Mapper



Demapper

Learning probabilistic shaping

From geometric to probabilistic constellation shaping

$$C = \max_{p(x),\, \theta_M} I(X;Y)$$

Probabilistic shaping

With which probability should each symbol be sent?

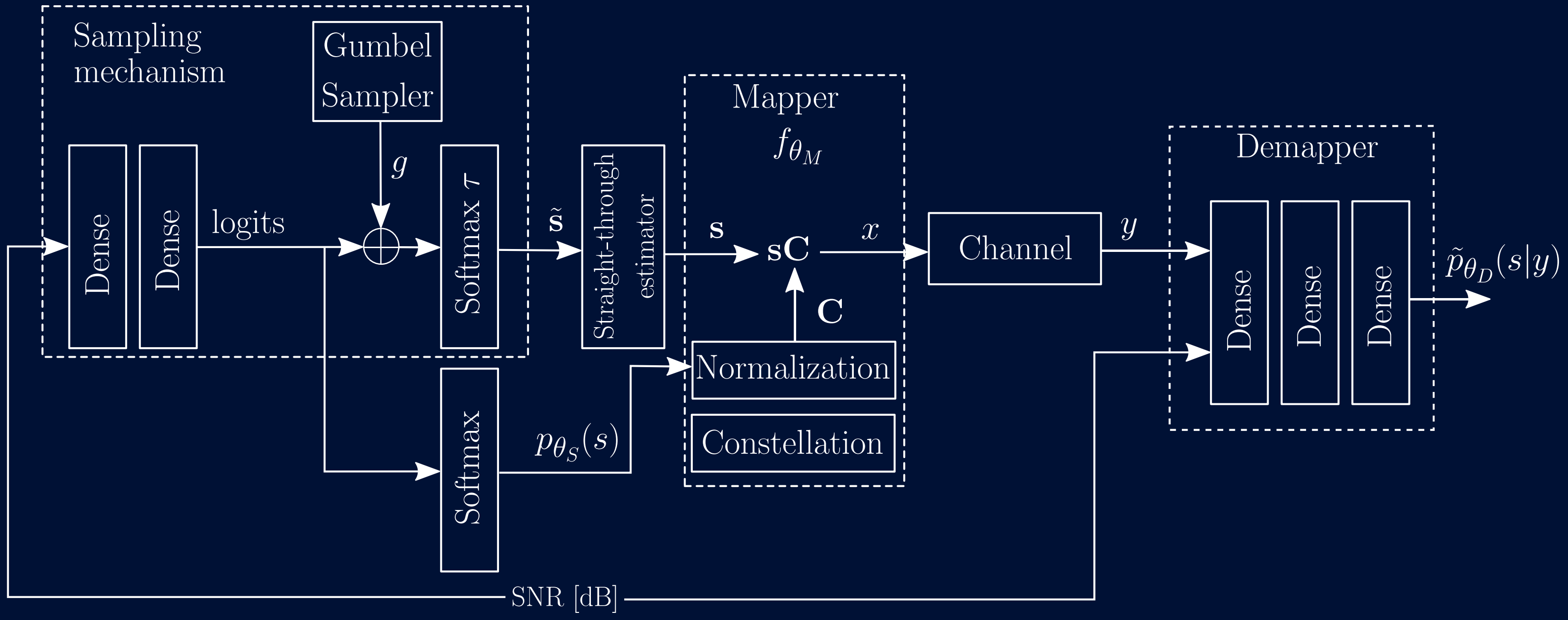
Geometric shaping

Where to place the constellation points?

Can we jointly learn probabilistic and geometric shaping?

https://arxiv.org/abs/1906.07748

Neural Network Architecture



How to sample from an arbitrary discrete distribution?

$$p(s), s \in \mathcal{M} = \{1, \cdots, M\}$$

- For any unconstrained representation $\gamma_s = \log(p(s)) + \alpha, \alpha \in \mathbb{R}$ we can recover $p(s)$ through the softmax function
$$p(s) = \frac{e^{\gamma_s}}{\sum_i e^{\gamma_i}}$$

- We can create samples from $p(s)$ according to
$$s = \underset{i \in \mathcal{M}}{\operatorname{argmax}}(\gamma_i + g_i)$$
$$g_i = -\log(-\log(u_i)), u_i \sim Uniform(0,1)$$

- This is called the Gumbel-Max trick (https://arxiv.org/abs/1206.6410) since the g_i are i.i.d. standard Gumbel r.v.'s

How to make the argmax operator differentiable?

- Generate a vector $\tilde{\boldsymbol{s}}$ with elements
$$\tilde{s}_i = \frac{e^{(\gamma_i + g_i)/\tau}}{\sum_j e^{(\gamma_j + g_j)/\tau}}, i = 1, \ldots, M$$
where $\tau > 0$ is called the **temperature**

- $\tilde{\boldsymbol{s}}$ is a probability vector such that $s = \operatorname*{argmax}_{i \in \mathcal{M}}(\tilde{s}_i)$

- As $\tau \to 0$, $\tilde{\boldsymbol{s}}$ becomes close to a one-hot vector

- $\tilde{\boldsymbol{s}}$ is differentiable w.r.t. γ_i

The straight-through estimator

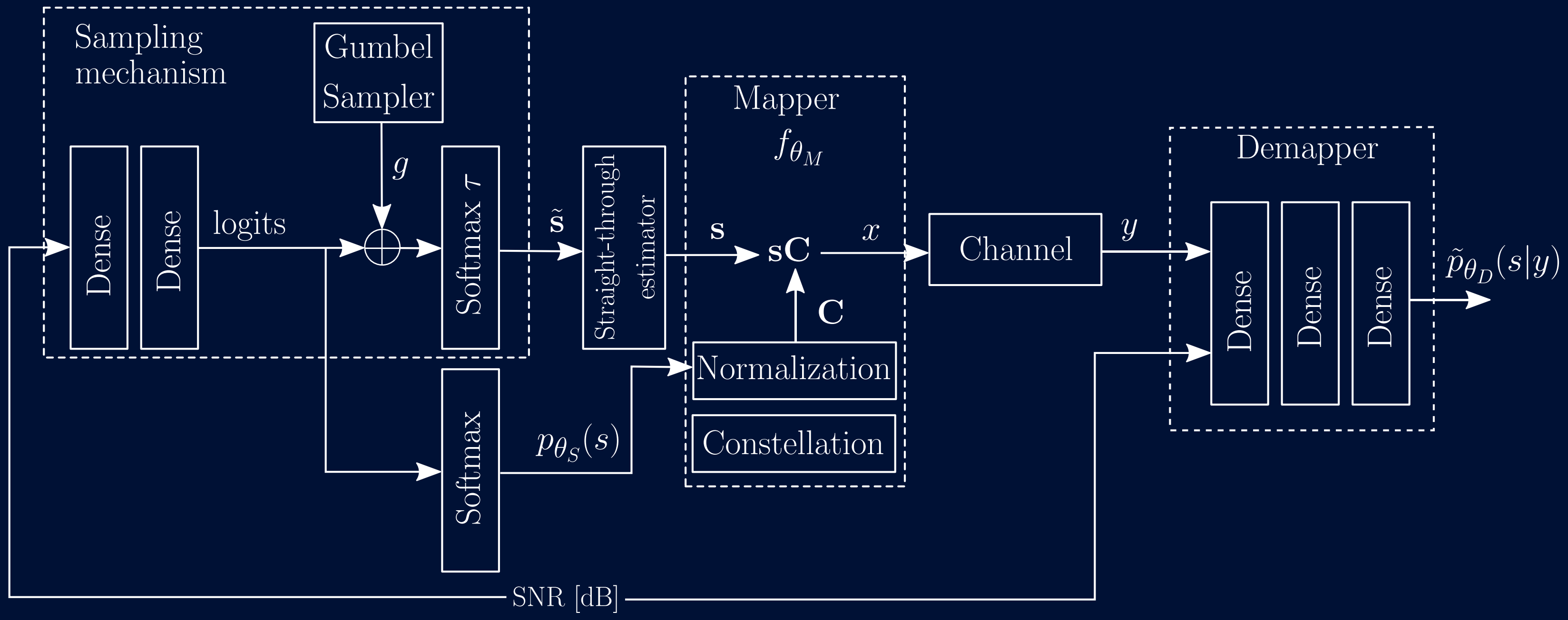
- Since $\tilde{\boldsymbol{s}}$ only approximately one-hot, $\tilde{\boldsymbol{s}}\boldsymbol{C}$ would result in the transmission of a convex combination of constellation points

- Key idea:
    1. Use true one-hot vector $\boldsymbol{s}$ for the forward pass
    2. Use approximate one-hot vector $\tilde{\boldsymbol{s}}$ in the backward pass

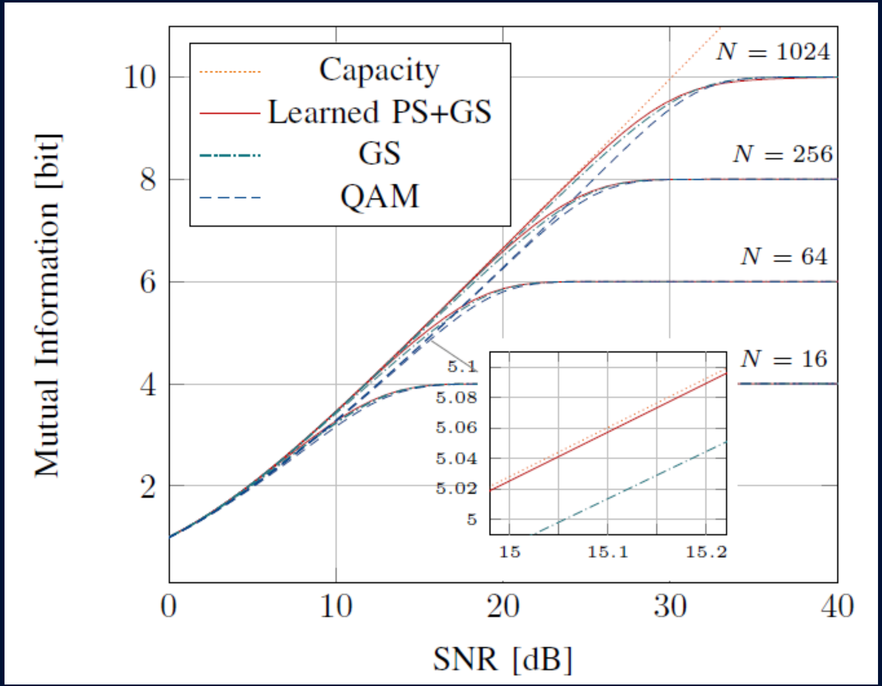
- Pseudo-code:

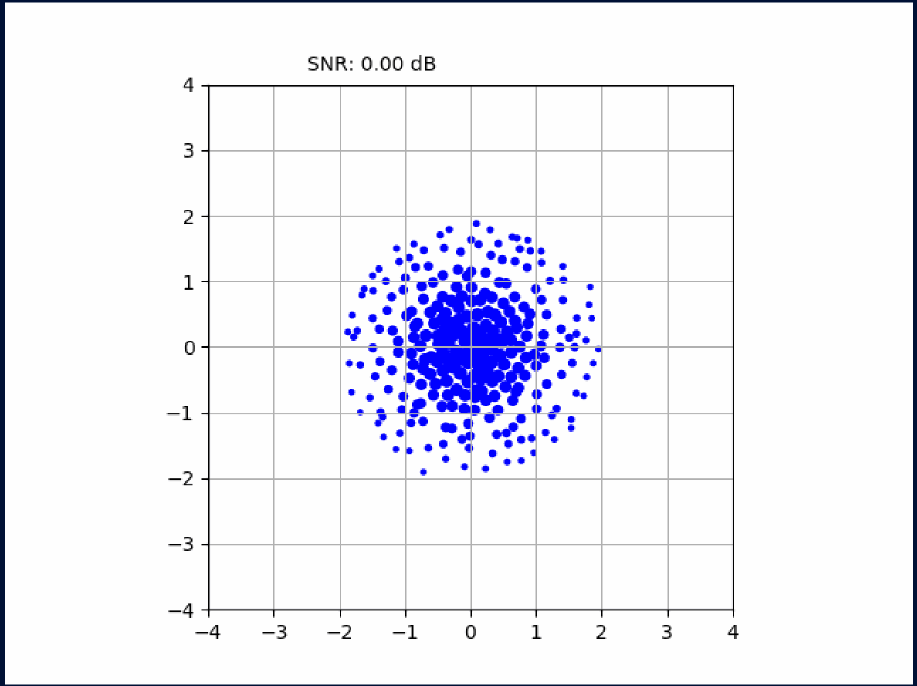
$$\boldsymbol{s} = \text{tf.stop\_gradient}(\boldsymbol{s} - \tilde{\boldsymbol{s}}) + \tilde{\boldsymbol{s}}$$

Neural Network Architecture



Results over AWGN Channel

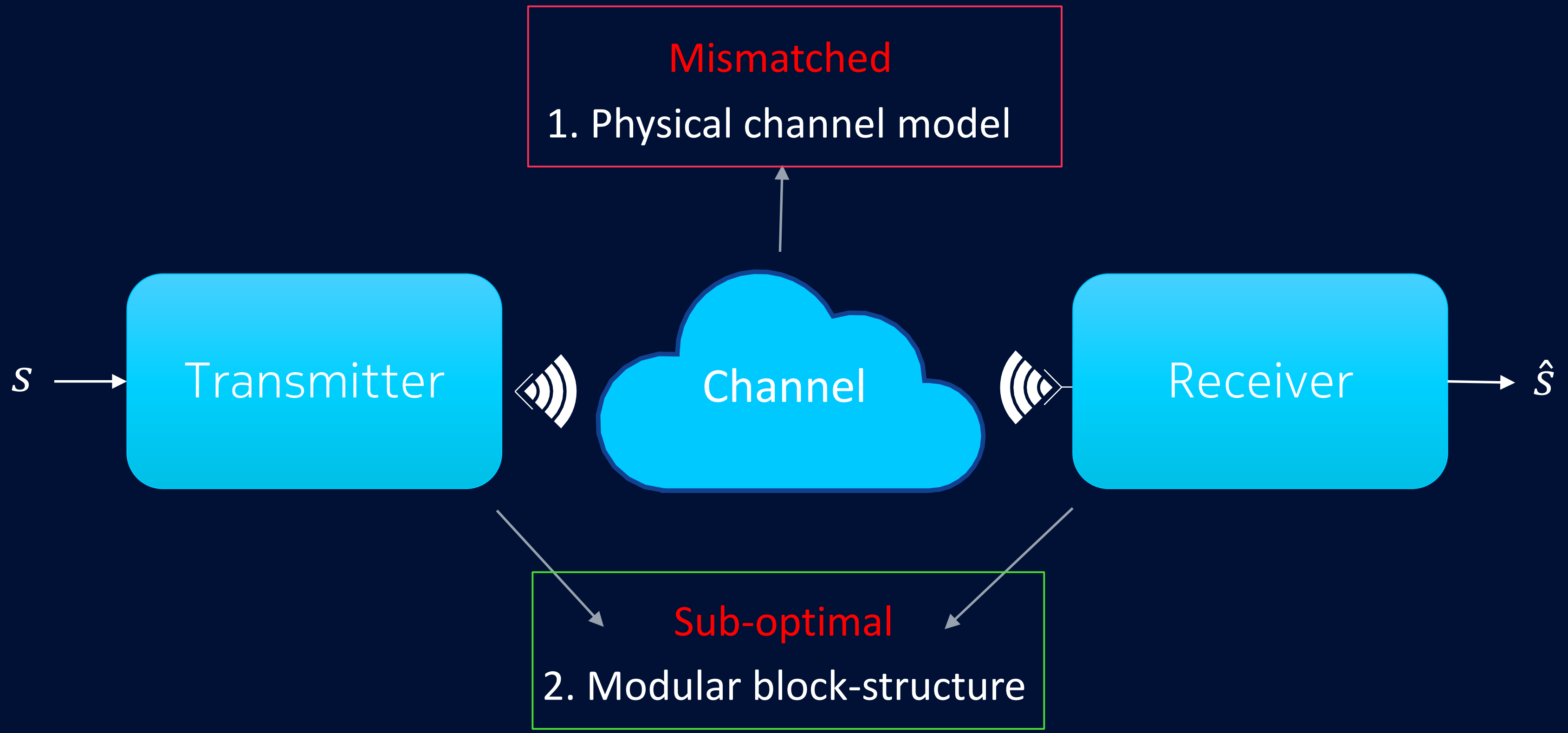




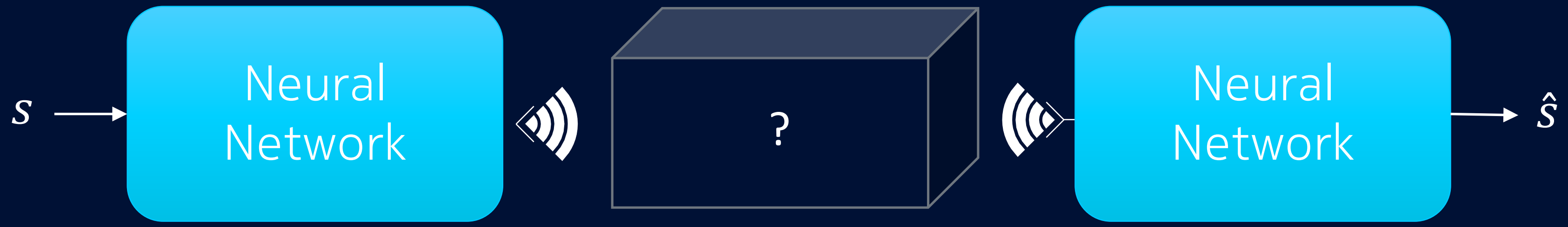
NOKIA Bell Labs

Learning over the actual channel

How we have solved the problem until now



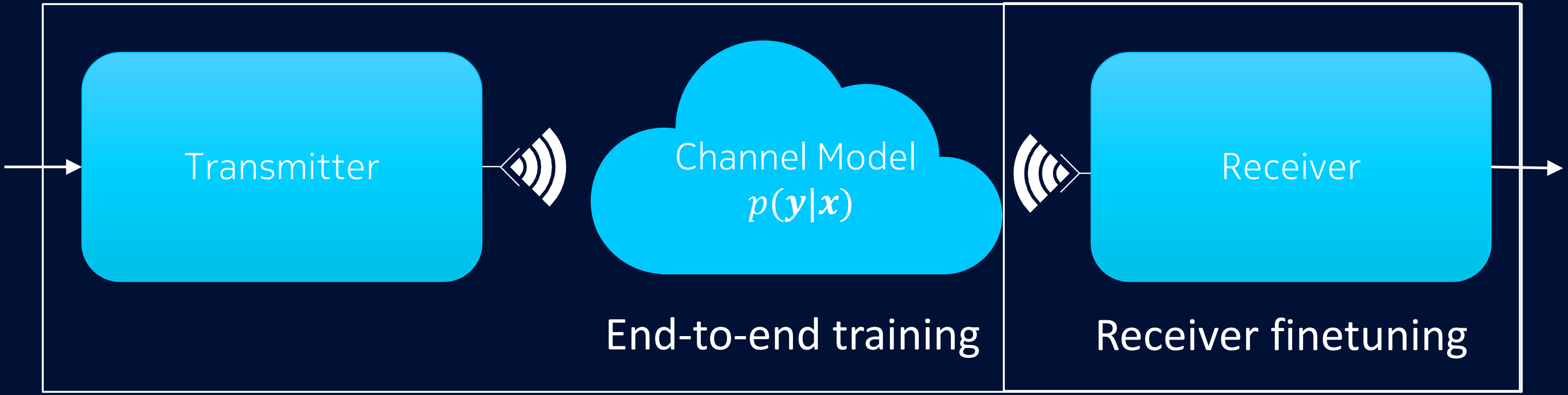
Practical challenge: Channel is a black box



How can we learn to communicate through an unknown channel?

1. Analytic channel model + Receiver finetuning (arxiv:1707.03384)
2. Learned channel model (Conditional GAN) → Supervised learning (arxiv:1807.00447)
3. Avoid channel modeling → Reinforcement learning (arxiv:1804.02276)

Option 1: Analytical channel model & receiver finetuning



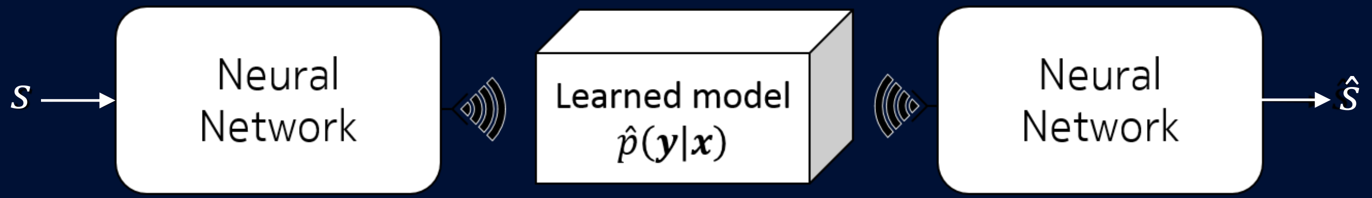
Phase 1: End-to-end training on analytical channel model

- Training based on a differentiable generative model $p(\boldsymbol{y}|\boldsymbol{x})$
- Does (probably) not cover all hardware impairments and channel characteristics
- Performance is limited by model accuracy

Phase 2: Receiver finetuning on physical channel

- Deployment of pre-trained AE on physical devices
- Freezing TX weights and transmission of training sequences
- Supervised SGD-based training of RX based on recorded training sequences

Option 2: Learn a generative channel model



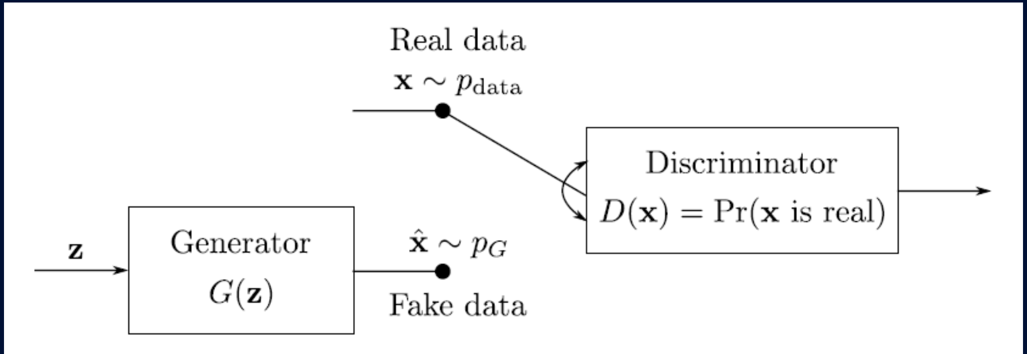
Key idea:

1. Learn a generative channel model from data
2. Train the autoencoder over the learned channel model

Challenges:

- How to build the model $\hat{p}(\boldsymbol{y}|\boldsymbol{x})$?
- How to draw sample from this model?
- How to compute gradients of $\boldsymbol{y}$ w.r.t. $\boldsymbol{x}$?

Generative adversarial networks (GANs)



- GANs can be thought of as two adversaries with opposing goals
- Generator $G(\boldsymbol{z})$:
  - Generates new data samples & tries to fool the discriminator
  - $\boldsymbol{z}$ is a latent (unobserved) variable, typically Gaussian noise
- Discriminator $D(\boldsymbol{x})$:
  - Tries to distinguish fake from real data samples

https://www.thispersondoesnotexist.com/

GAN details

Two-player min-max game:

$$\min_G \max_D E_{\boldsymbol{x} \sim p_{data}} [\log D(\boldsymbol{x})] + E_{\boldsymbol{z} \sim p_{\boldsymbol{z}}} [\log(1 - D(G(\boldsymbol{z})))]$$

Generator:

- $G = G_{\theta_g}: \mathbb{R}^L \mapsto \mathbb{R}^n$ is a neural network
- $\hat{\boldsymbol{x}} = G_{\theta_g}(\boldsymbol{z}) \sim p_g$, for some pior distribution on the noise $\boldsymbol{z} \sim p_{\boldsymbol{z}}$
- Typical choice $p_{\boldsymbol{z}}(\boldsymbol{z}) = N(\boldsymbol{z}; \sigma^2 \boldsymbol{I})$

Discriminator:

- $D = D_{\theta_d}: \mathbb{R}^n \mapsto [0,1]$ is another neural network
- D_{θ_d} is a binary classifier with sigmoid output activation
- $D_{\theta_d} = \Pr(\boldsymbol{x} \text{ is generated by } p_{data})$

GAN vanilla training algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

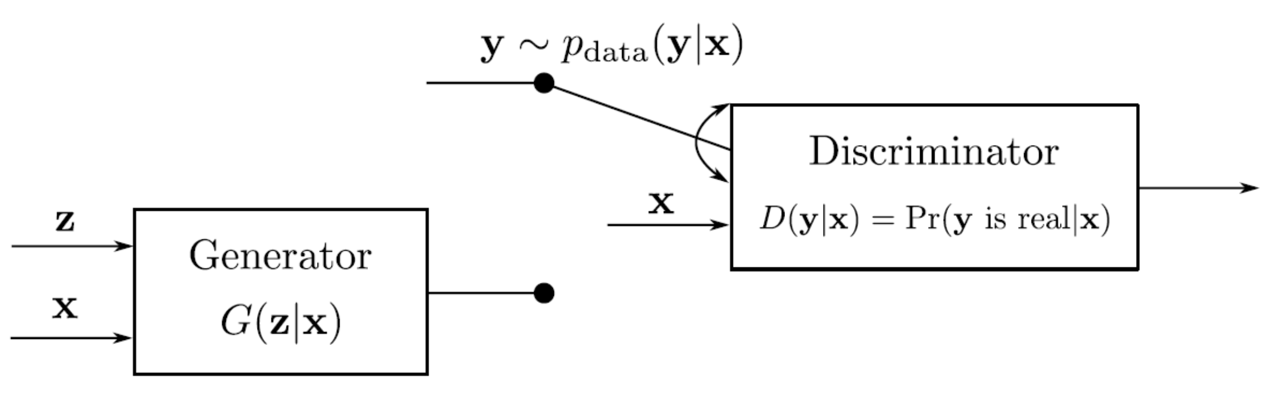
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---

http://papers.nips.cc/paper/5423-generative-adversarial-nets

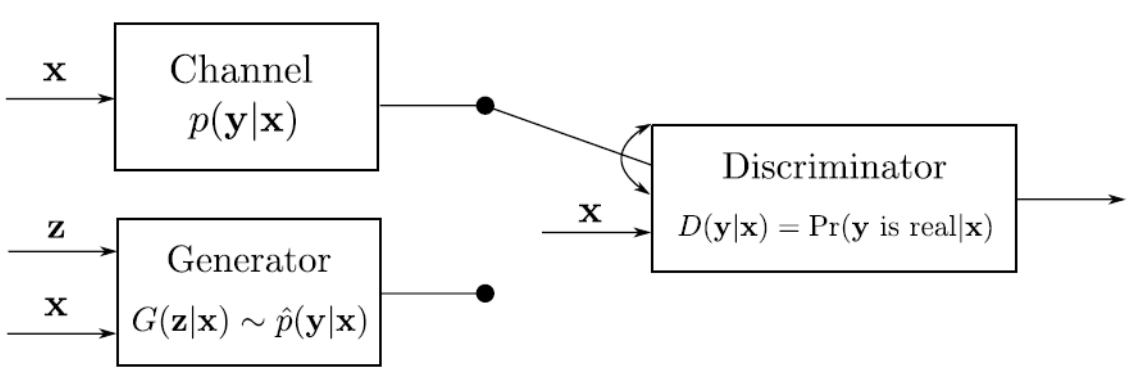
Conditional GANs



- Joint distribution $p_{data}(\boldsymbol{x}, \boldsymbol{y}) = p_{data}(\boldsymbol{y}|\boldsymbol{x})p_x(\boldsymbol{x})$ for data $\boldsymbol{y}$ and some auxilliary information $\boldsymbol{x}$, e.g., class label
- New min-max game (https://arxiv.org/abs/1411.1784):

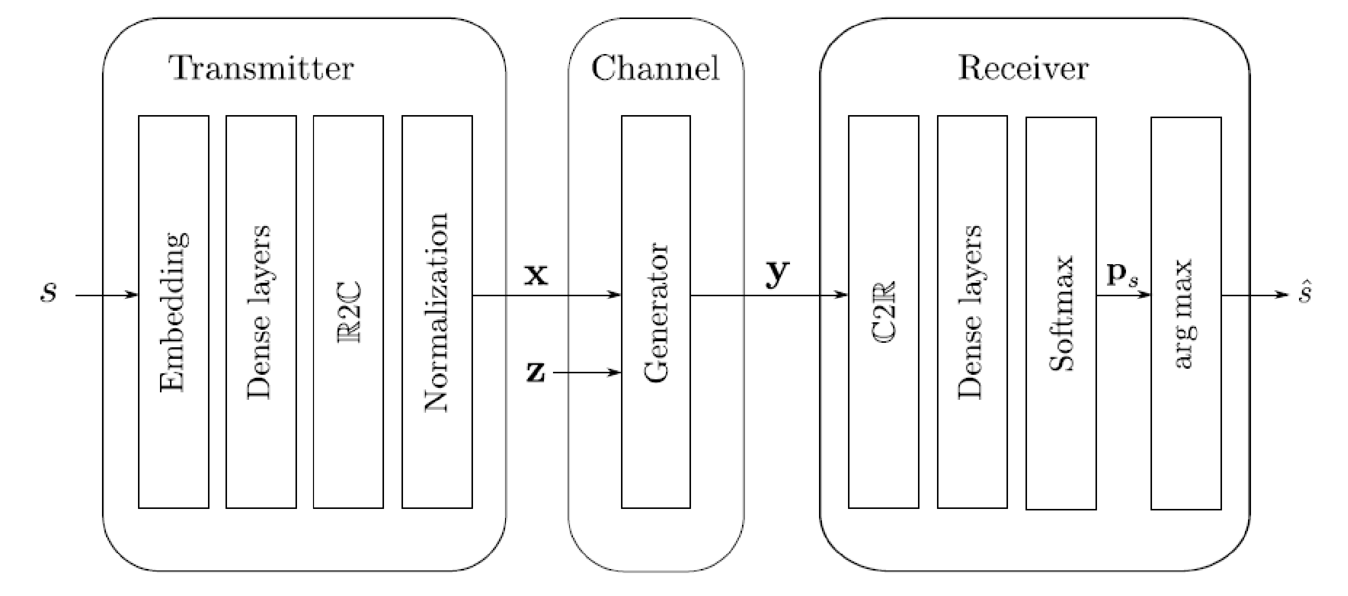
$$\min_G \max_D E_{x \sim p_{data}} [\log D(\boldsymbol{y}|\boldsymbol{x})] + E_{z \sim p_z, x \sim p_x}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{x}))|\boldsymbol{x})]$$

Conditional GANs for channel modeling



- Goal: Learn to sample channel outputs $\boldsymbol{y}$ for a given input $\boldsymbol{x}$
- $\boldsymbol{x}$ is the transmitted message representation
- $p_{\boldsymbol{x}}$ depends on the transmitter (e.g., QPSK modulation)
- For the autoencoder, it is not known before training

Autoencoder trained on a conditional GAN

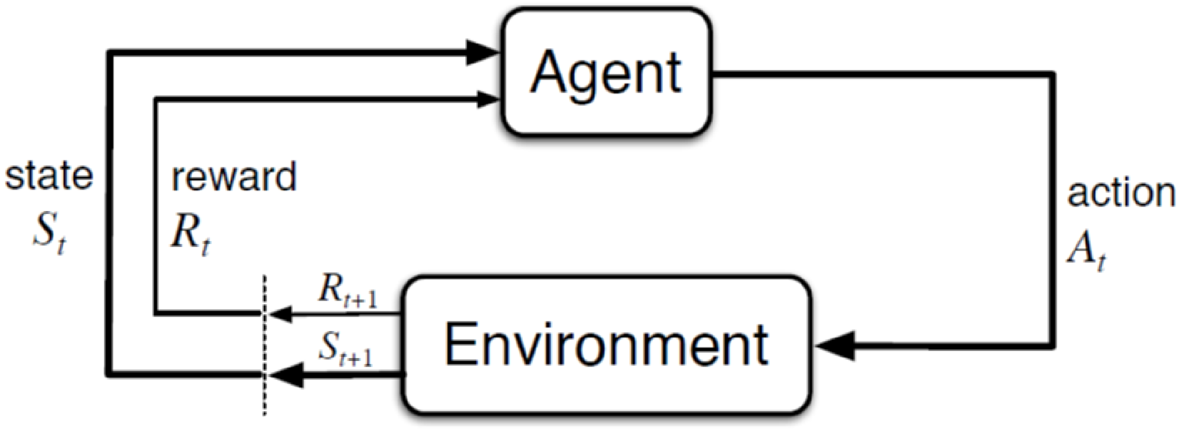


1. Train the Generator for some distribution p_x
2. Train the autoencoder over a fixed Generator

Try it yourself

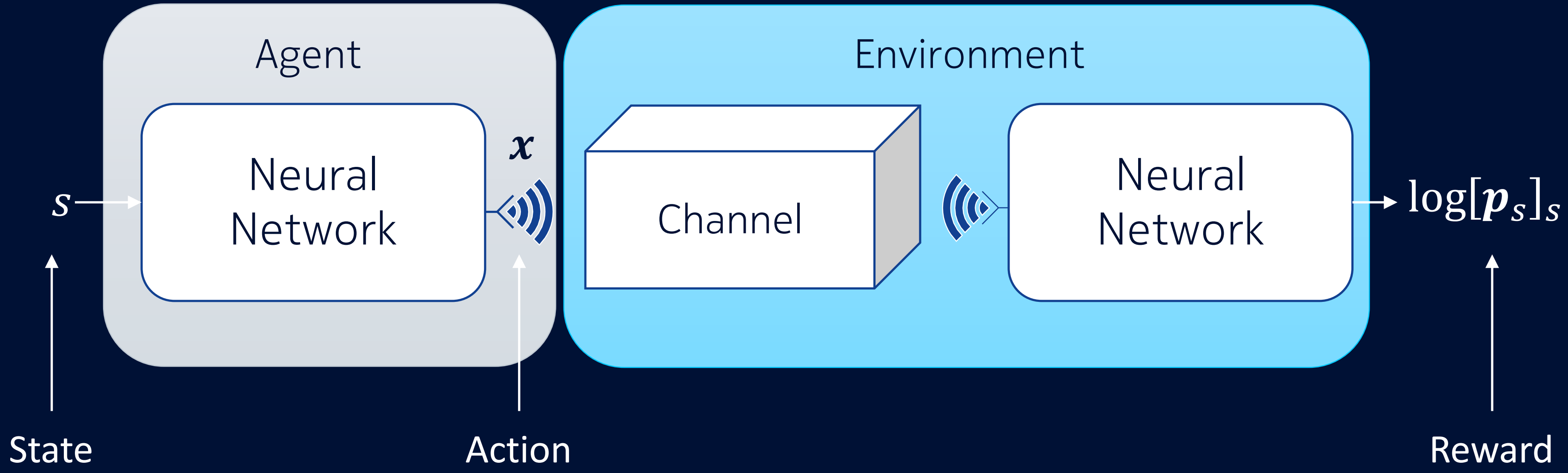
NOKIA Bell Labs

Option 3: Reinforcement learning



- An Agent interacts with an Evironment by taking Actions and observing a State and Reward
- The goal of the Agent is to take Actions such that a funtion of the intermediate Rewards is maximized

The transmitter as an agent



- The transmitter observes the state $s \in \mathcal{M}$,
- ...takes the action $\boldsymbol{x} = f_{\theta_t}(s)$,
- ...and observes the reward $\log[\boldsymbol{p}_s]_s \triangleq -l$
- Problem:

$$\text{argmax}_{\theta_t} E[\log[\boldsymbol{p}_s]_s] = \text{argmin}_{\theta_t} E[\text{l}]$$

From fixed actions to a stochastic policy

- To enable exploration (learning), the transmitter applies a **Gaussian Policy** $\pi_{\theta_t}(\boldsymbol{x}|s) = N(\boldsymbol{x}; f_{\theta_t}(s), \sigma_\pi^2)$, i.e.,

$$\boldsymbol{x} = f_{\theta_t}(s) + \boldsymbol{\varepsilon}, \ \boldsymbol{\varepsilon} \sim N(\boldsymbol{\varepsilon}; 0, \sigma_\pi^2 \boldsymbol{I})$$

- Expected loss when message s is transmitted as $\boldsymbol{x}$

$$\mathcal{L}(s, \boldsymbol{x}) = E[l|s, \boldsymbol{x}]$$

- New problem:

« Find the policy which minimizes the expected loss »

$$\underset{\theta_t}{\operatorname{argmin}} \, E_s[\int_{\boldsymbol{x} \in \mathbb{R}^n} \pi_{\theta_t}(\boldsymbol{x}|s) \, \mathcal{L}(s, \boldsymbol{x}) d\boldsymbol{x}]$$

Policy gradient

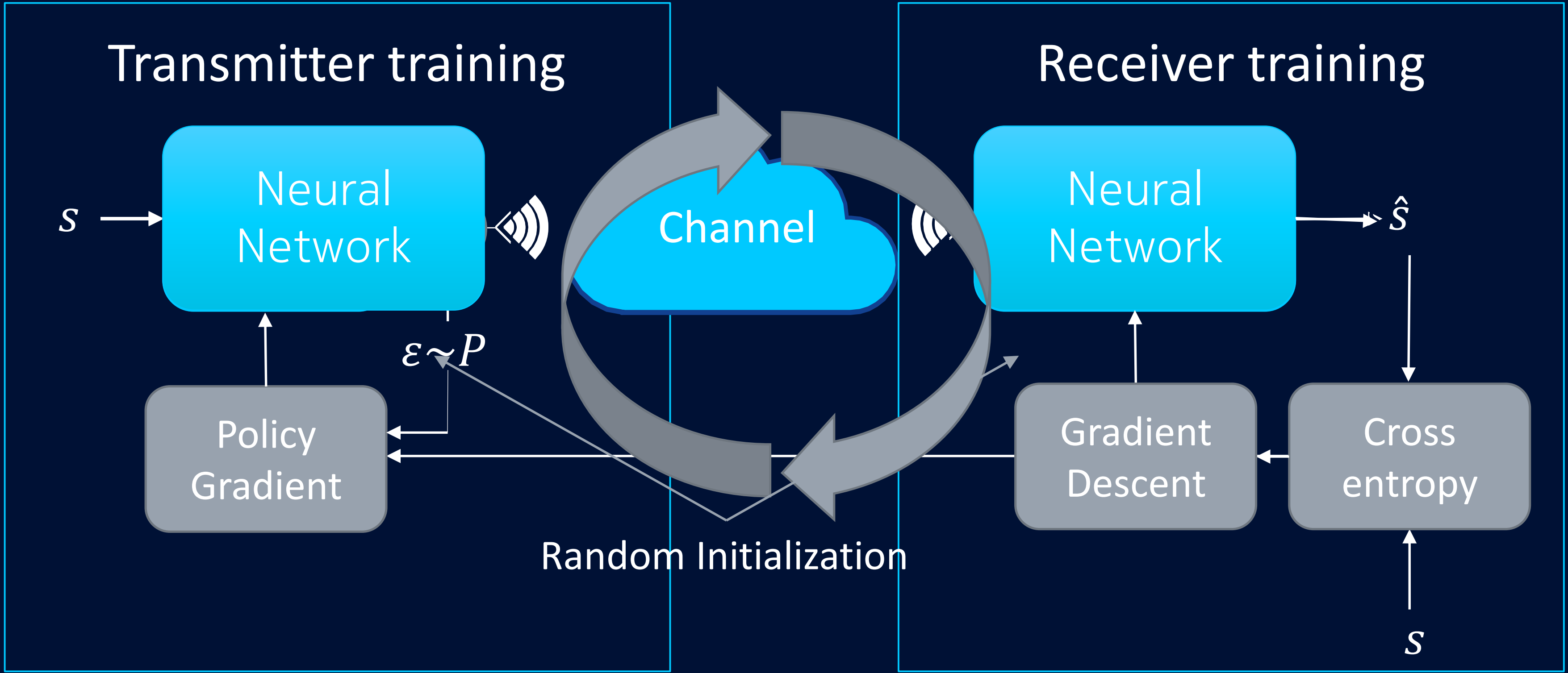
- Update weights according to

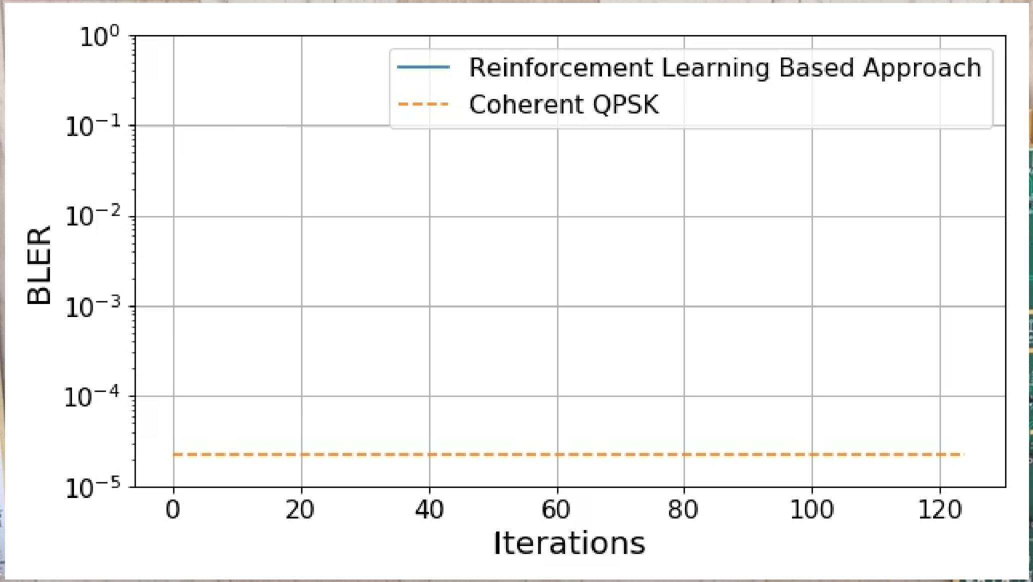
$$\theta_t = \theta_t - \eta \underbrace{\nabla_{\theta_t} E_s \left[\int_{\boldsymbol{x} \in \mathbb{R}^n} \pi_{\theta_t}(\boldsymbol{x}|s)\, \mathcal{L}(s, \boldsymbol{x}) d\boldsymbol{x} \right]}_{\text{Policy gradient}}$$

with the following approximation

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}_t} \mathbb{E}_s \left[\int_{\mathbf{x} \in \mathbb{R}^n} \pi_{\boldsymbol{\theta}_t}(\mathbf{x}|s) \mathcal{L}(s, \mathbf{x}) dx \right] \\
&= \mathbb{E}_s \left[\int_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}(s, \mathbf{x}) \nabla_{\boldsymbol{\theta}_t} \pi_{\boldsymbol{\theta}_t}(\mathbf{x}|s) dx \right] \\
\left(\nabla \log f(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{f(\mathbf{x})} \right) &= \mathbb{E}_s \left[\int_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}(s, \mathbf{x}) \pi_{\boldsymbol{\theta}_t}(\mathbf{x}|s) \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\mathbf{x}|s) dx \right] \\
(\text{Sample mean}) &\approx \frac{1}{N} \sum_{i=1}^{N} l_i \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\mathbf{x}_i|s_i) \\
&\approx \frac{1}{N} \sum_{i=1}^{N} l_i \frac{2}{\sigma_\pi^2} \left(\nabla_{\boldsymbol{\theta}_t} f_{\boldsymbol{\theta}_t}(s_i) \right)^{\mathsf{T}} \left(\mathbf{x}_i - f_{\boldsymbol{\theta}_t}(s_i) \right)
\end{aligned}$$

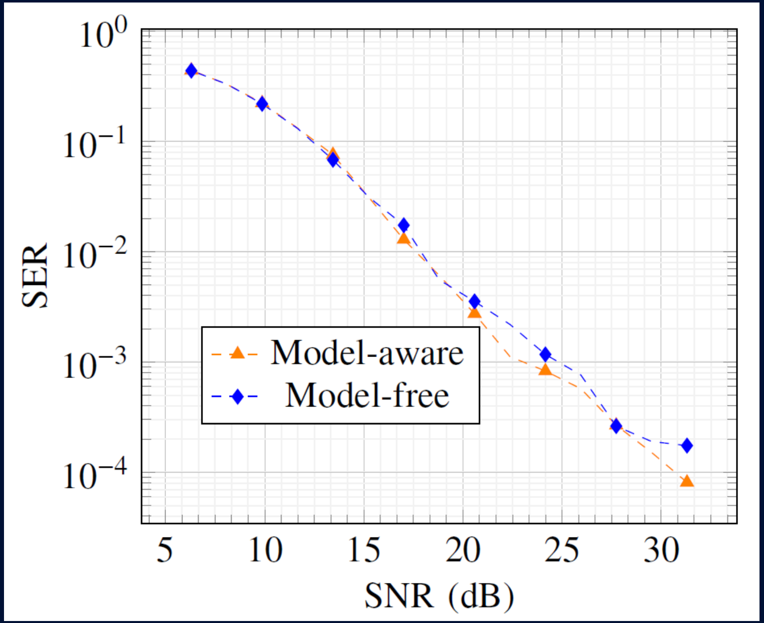
The deep reinforcement learning-based solution





- 5 MHz Bandwidth
- Unsynchronized Software-Defined Radios
- Implemented in Tensorflow/Python
- Trains in less than 1 hour (NVIDIA GTX1080)

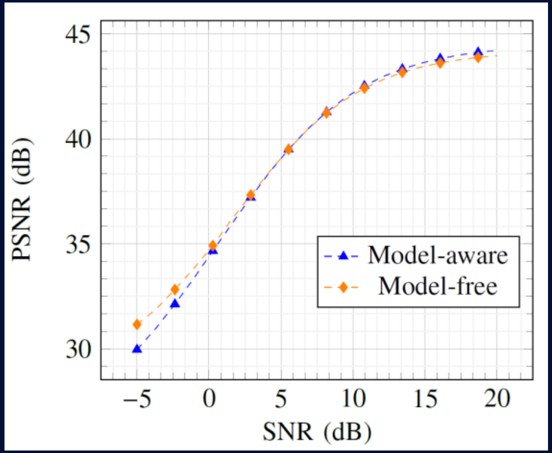
It works on fiber-optical channels (https://arxiv.org/abs/1812.05929)



$$\mathbf{x}_k = \begin{cases} \mathbf{x}_{k-1} \exp j \frac{L\gamma|\mathbf{x}_{k-1}|^2}{K} + \mathbf{n}_k & \text{for } 1 \leq k < K \\ \mathbf{x} & \text{for } k = 0 \end{cases}$$

NOKIA Bell Labs

…for joint source-channel coding (https://arxiv.org/abs/1812.05929)



Future directions

- End-to-end learning has been applied to various areas, e.g., optical communications, MIMO, VLC, in-body communications, joint source-channel coding,etc,

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

- Big potential for multiuser communications: MAC, BC (NOMA)

- Should always be considered with channel coding in mind