

Introduction to Deep Learning for the Physical Layer

****Updated****

**Instructors: Fayçal Aït Aoudia and
Sebastian Cammerer**

10-11 May 2023, 9:00am-12:30pm EDT each day

Presented by the IEEE Communications Society

Copyright 2023 © IEEE. All rights reserved.

Published May 2023; United States of America.

No part of this publication may be reproduced in any form, in an electronic retrieval system, or otherwise, without the prior written permission of the publisher.

Instructors



Fayçal Aït Aoudia

Research Scientist
NVIDIA

Fayçal Aït Aoudia is a Senior Research Scientist at NVIDIA working on the convergence of wireless communications and machine learning. Before joining NVIDIA, he was a research scientist at Nokia Bell Labs, France. He is one of the maintainers and core developers of the Sionna open-source link-level simulator. He obtained the diploma degree in computer science from the Institut National des Sciences Appliquées de Lyon, France, in 2014, and the PhD in signal processing from the University of Rennes 1, France, in 2017.



Sebastian Cammerer

Research Scientist
NVIDIA

Sebastian Cammerer is a Research Scientist at NVIDIA working on the intersection of wireless communications and machine learning. Before joining NVIDIA, he received his PhD in electrical engineering and information technology from the University of Stuttgart, Germany, in 2021. He is one of the maintainers and core developers of the Sionna open-source link-level simulator.

His main research topics are machine learning for wireless communications and channel coding. Further research interests include parallel computing for wireless signal processing and information theory. He is recipient of the VDE ITG Dissertationspreis 2022, the IEEE SPS Young Author Best Paper Award 2019, the Best Paper Award of the University of Stuttgart 2018, and third prize winner of the Nokia Bell Labs Prize 2019.

Introduction to Deep Learning for the Physical Layer *(updated)*

Sebastian Cammerer & Fayçal AïtAoudia

Goals of this course

- Identify when it makes sense to use machine learning
- Learn how to use deep learning to solve physical layer problems
- Know most important neural network components & architectures
- Setup simple experiments by yourself
- Be aware of current research trends and 3GPP activities
- Describe the idea of graph neural networks and self-attention
- Understand how deep learning is used for OFDM MIMO detection
- Understand the idea of end-to-end learning
- Understand how deep learning is used for channel decoding
- Understand the idea of differentiable ray tracing

Course outline

- Part I: Primer on Machine Learning and Deep Learning
 - Introduction & Machine Learning Basics (45min)
 - Advanced Neural Network Architectures (40mins)
 - Introduction to Sionna and Differentiable Ray Tracing (45min)
- Part II: Applications of Deep Learning for Communications
 - DL at the Receiver (1:15h)
 - End-to-end Learning (1h)
 - Deep Learning for Channel Coding (1:15h)

Part I: Primer on Machine Learning and Deep Learning

Resources

Online learning material:

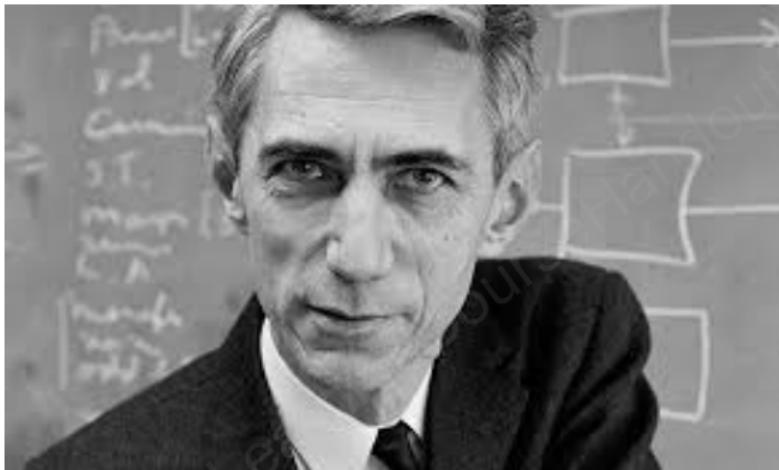
- <http://www.deeplearningbook.org/>
- <https://work.caltech.edu/telecourse.html>
- <https://www.coursera.org/specializations/deep-learning>
- <http://course.fast.ai/>
- <https://distill.pub/>
- <https://www.youtube.com/c/AndrejKarpathy>

Course material:

- Jupyter notebooks (linked in the slides)
- Sionna library: <https://nvlabs.github.io/sionna/>

IEEE COMSOC ETI “Machine Learning for Communications”
<https://mlc.committees.comsoc.org>

Why is deep learning for communications a bad idea?



Credit: Alfred Eisenstaedt/The Life Picture Collection/Getty Images

- Fundamental limits known
- We are already close
- Little room for improvements
- Man-made signals

Why is deep learning for communications a good idea?

- **Inadequate system models**
Gaussian, stationary, linear
- **Limiting functional block-structure**
Chain of blocks vs. end-to-end optimized system?
- **Parallelization gains of neural networks**
“Learned” vs. “programmed” algorithms?
- **Specialized hardware for ML applications**
Unprecedented energy efficiency & throughput

Snapdragon X70 powered by world's first 5G AI Processor pushes smarter, faster 5G performance [video]

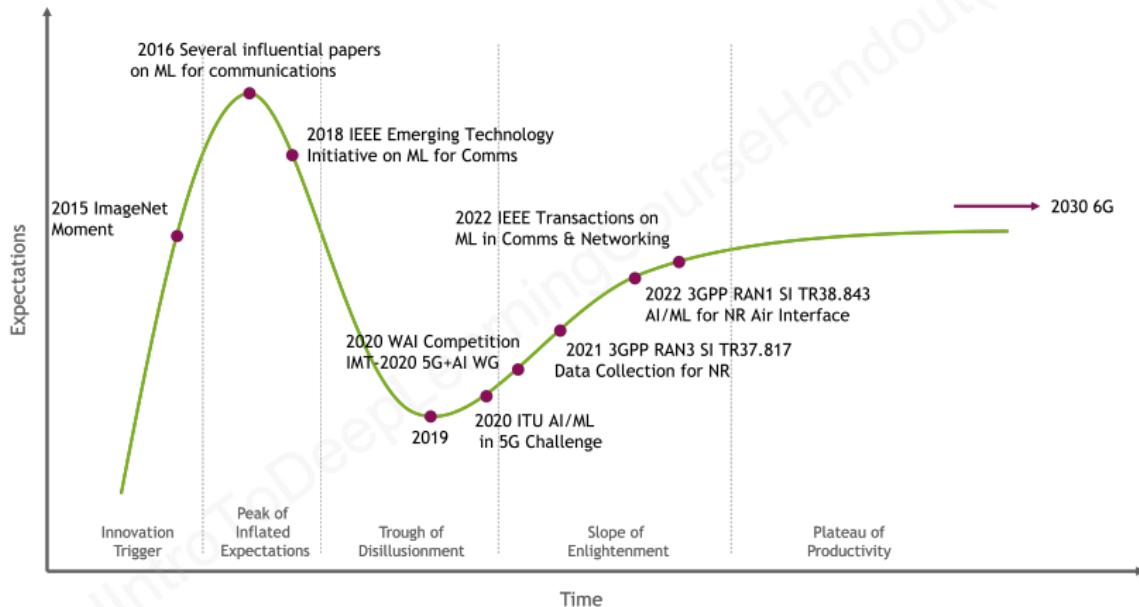
Qualcomm Technologies unveils Snapdragon X70 at MWC 2022.

FEB 28, 2022

Qualcomm products mentioned within this post are offered by Qualcomm Technologies, Inc. and/or its subsidiaries.

- Inadequate system models
Gaussian, stationary, linear
- Limiting functional block-structure
Chain of blocks vs. end-to-end optimized system?
- Parallelization gains of neural networks
“Learned” vs. “programmed” algorithms?
- Specialized hardware for ML applications
Unprecedented energy efficiency & throughput

Machine learning (ML) for Communications: Where are we?



3GPP Workplan for 5G Advanced (Release 18)



Release 18



TSG SA priorities*

SA2 led - System Architecture and Services

- XR (Extended Reality) & media services
- Edge Computing Phase 2
- System Support for AI/ML-based Services
- Enablers for Network Automation for 5G Phase 3
- Enh. support of Non-Public Networks Phase 2
- Network Slicing Phase 3
- 5GC LoCallion Services Phase 3
- 5G multicast-broadcast services Phase 2
- Satellite access Phase 2
- 5G System with Satellite Backhaul
- 5G Timing Resiliency and TSC / ULLC enh.
- Evolution of IMS multimedia telephony service
- Personal IoT Networks
- Vehicle Mounted Relays

SA3 led - Security and Privacy

- Privacy of identifiers over radio access
- SECAM and SCAS for 3GPP virtualized network products and Management Function (MnF)
- Mission critical security enhancements Phase 3
- Security and privacy aspects of RAN & SA features

SA4 led - Multimedia Codecs, Systems and Services

Systems & Media Architecture:

- 5G Media, Service Enablers
- Split-Rendering
- 5G AR Experiences Architecture

Media:

- Video codec for 5G
- Media Capabilities for Augmented Reality Glasses
- AI / ML Study

Real-Time Communications:

- XR conversational services
- WebRTC-based services and collaboration models

Immersive Voice & Audio:

- EVS Codec Extension
for Immersive Voice and Audio Services [IVAS_Codec]
- Terminal Audio quality performance and Test methods
for Immersive Audio Services [ATIAS]

Streaming & Broadcast services:

- 5GMS Enh. [Network slicing, Low latency, Background traffic, MBS Uplink]
- Further MBS Enh. [Free to air, Hybrid unicast/broadcast]

- Access Traffic Steering, Switching & Splitting support in the 5G system architecture Phase 3
- Proximity-based Services in 5GS Phase 2
- UPF enh. for Exposure & SBA
- Ranging based services & sidelink positioning
- Generic group management, exposure & communication enh.
- 5G UE Policy Phase 2
- UAS, UAV & UAM Phase 2
- 5G AM Policy Phase 2
- RedCap Phase 2
- Support for SWWC Phase 2
- System Enabler for Service Function Chaining
- Extensions to TSC Framework to support DfNet
- Seamless UE context recovery
- MPS when access to EPC/5GC is WLAN

SA5 led - Management, Orchestration and Charging

Operations, Administration, Maintenance and Provisioning (OAM&P):

- Intelligence and Automation: Self-Configuration of RAN NEs, Enh. autonomous network levels, Evaluation of autonomous network levels, Enh. intent driven management services for mobile networks, AI / ML management, Enh. of the management aspects related to NWDAF
- Management Architecture and Mechanisms: Network slicing provisioning rules, Enh. service based management architecture
- Support of New Services: Enh. Energy Efficiency for 5G Phase 2, New aspects of Energy Efficiency for 5G networks Phase 2, Enh. management of Non-Public Networks, Network and Service Operations for Energy Utilities, Key Quality Indicators (KQIs) for 5G service experience, Deterministic Communication Service Assurance
- Charging: Charging Aspects for Enh. Support of Non-Public Networks

SA6 led - Application Enablement & Critical Communication Applications

Critical Communications:

- MCX Enhancements – MC over 5GS (5MBS, ProSe) Adhoc group comm., MCPTI Enh.
- Railways – Gateway UE, Interworking
- Service Frameworks:
- Edge App Architecture Enh., SEAL Enh., Subscriber-Aware API (CAPIF Enh.)
- Fused location, Application Data Analytics, App Layer NW Slicing
- Enablers for Vertical Applications:
- Enhancements to V2X, UAS application-enablement
- Future Factories, Personal IoT networks, Capability exposure for IoT platforms

TSG RAN priorities*

RAN1 led - Radio Layer 1 (Physical layer)

- NR-MIMO Evolution
- AI/ML - Air Interface
- Evolution of duplex operation
- NR Sidelink Evolution
- Positioning Evolution
- RedCap Evolution
- Network energy savings
- Further UL coverage enhancement
- Smart Repeater
- DSS
- Low power WUS
- CA enhancements

RAN2 led - Radio layer 2 & layer 3 Radio Resource Control

- Mobility Enhancements
- Enhancements for XR
- Sidelink Relay Enhancements
- NTN (Non-Terrestrial Networks) evolution - NR
- NTN (Non-Terrestrial Networks) evolution - IoT
- UAV (Uncrewed Aerial Vehicle)
- Multiple SIM (MUSIM) Enhancements
- In-Device Co-existence (IDC) Enhancements
- Small data
- MBS

RAN3 led - UTRAN/E-UTRAN/NG-RAN architecture & related network interfaces

- Additional topological improvements – IAB/VMR
- AI/ML for NG-RAN WI
- AI/ML for NG-RAN SI
- SON/MDT Enhancements
- QoE Enhancements
- Resiliency of gNB-CU-CP

RAN4 led - Radio Performance and Protocol Aspects

- RAN4-led spectrum items
- <5MHz in dedicated spectrum

Rel-18 Workplan for TSG CI

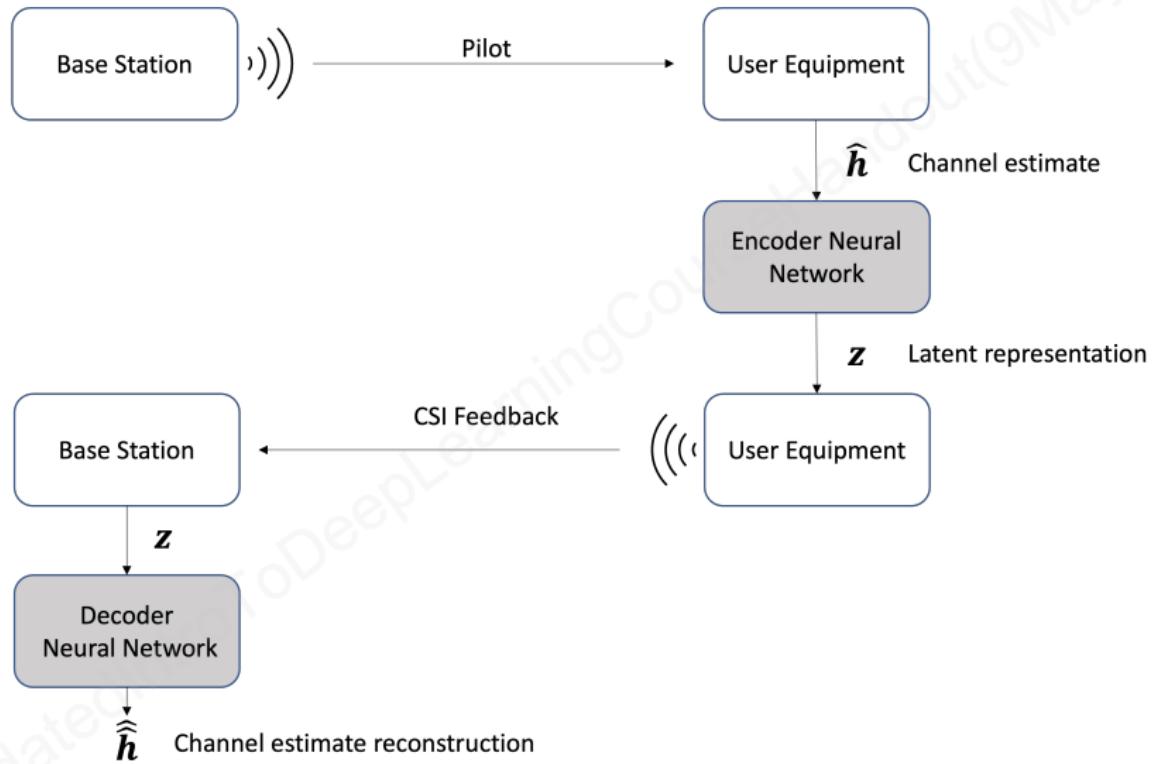
CI will work on Stage 3 completion and ASN.1 code and OpenAPI freeze of Rel-17 until June 2022 (TSG#99).

Work item discussion on Rel-18 Stage 2 / Stage 3 (under CI) from June 2022.

“[...] explore the benefits of augmenting the air-interface with features enabling improved support of AI/ML based algorithms for enhanced performance and/or reduced complexity/overhead. [...] this SI will lay the foundation for future air-interface use cases leveraging AI/ML”

- Led by RAN1 (physical layer), involving RAN2 and RAN4
- ~ 50 supporting members (Ericsson, Nokia, Huawei,...)
- Objectives:
 - Identify areas of potential performance improvements
 - Understand attainable gains and complexity requirements
 - Assess specification impact
- Study a few carefully selected use-cases to enhance:
 - Channel state information (CSI) (feedback) compression & prediction
 - Beam management (spatial & time-domain prediction)
 - Positioning accuracy

Example: Autoencoder for CSI feedback compression



Wireless Communication Artificial Intelligence Competition

Table 2. DL-based F-CSI feedback solutions with different schemes.

Solution ID	Bits ¹	Scheme											
		Data Preprocessing				Backbone			Quantization Enhancement				
Path Cutting	Domain Transforming	Additional Info. Exploiting	Data Augmentation	FCN	CNN	Transformer	Adaptive Scalar Quan.	Vector Quan.	Path-level Non-uniform Quan.	Learnable Quan. Error Offset	Derivable Quantized Step Function		
1	286	✓					✓		✓	✓			
2	354	✓			✓				✓	✓			
3	363	✓	✓				✓						
4	369			✓			✓						
5	372						✓		✓				
6	375				✓		✓				✓		
7	378				✓		✓				✓		
8	381						✓						
9	384							✓					
10	384						✓					✓	

¹ Note that the number of feedback bits satisfies that $\text{NMSE} = \mathbb{E} \left\{ \frac{\|\mathbf{H}' - \mathbf{H}\|_2^2}{\|\mathbf{H}\|_2^2} \right\} \leq 0.1$.

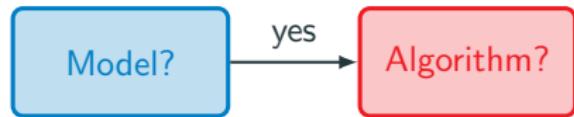
- Organized by IMT-2020(5G) Promotion Group 5G+AI Work Group
- 900+ competing teams (210 companies, 160 universities)
- Best solution achieves more than 98% compression! [XWT⁺²¹]

Machine Learning Basics

When (not) to use ML?

Model?

When (not) to use ML?



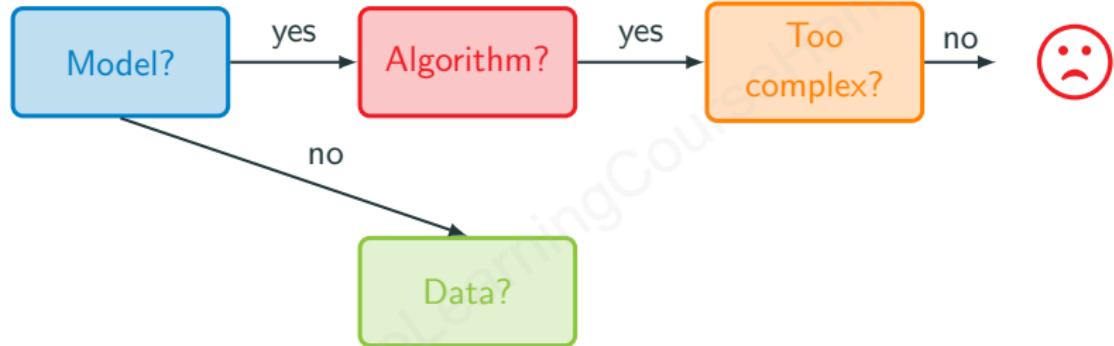
When (not) to use ML?



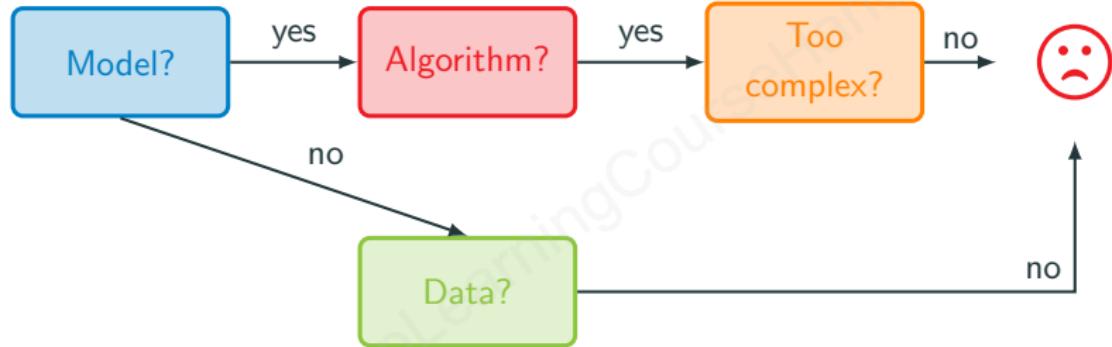
When (not) to use ML?



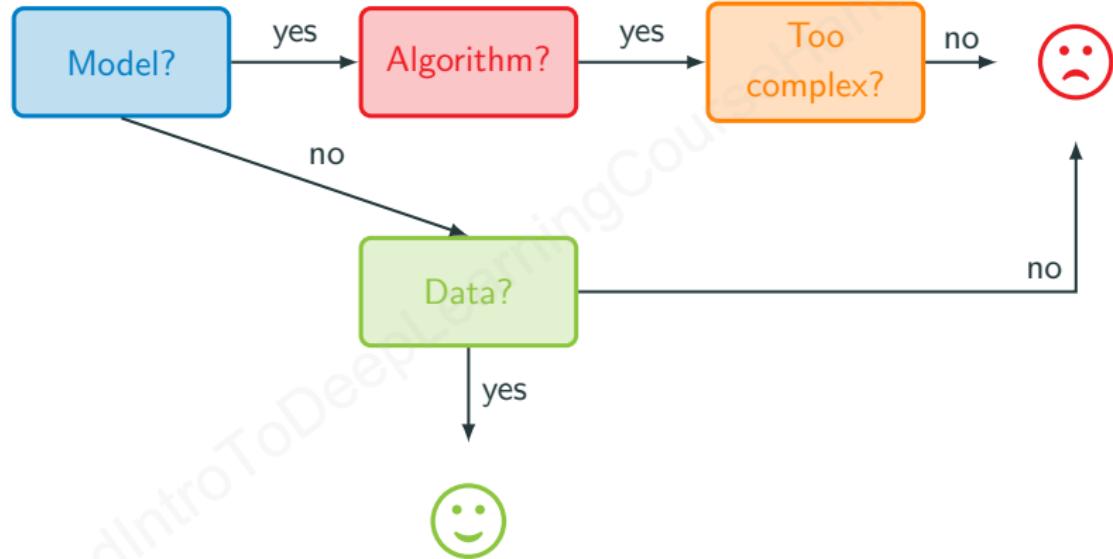
When (not) to use ML?



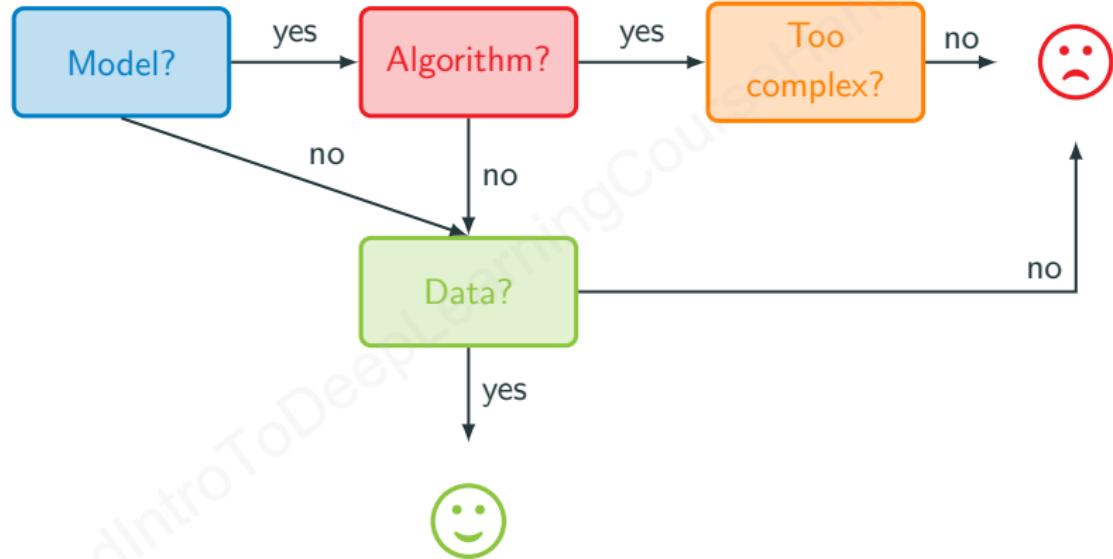
When (not) to use ML?



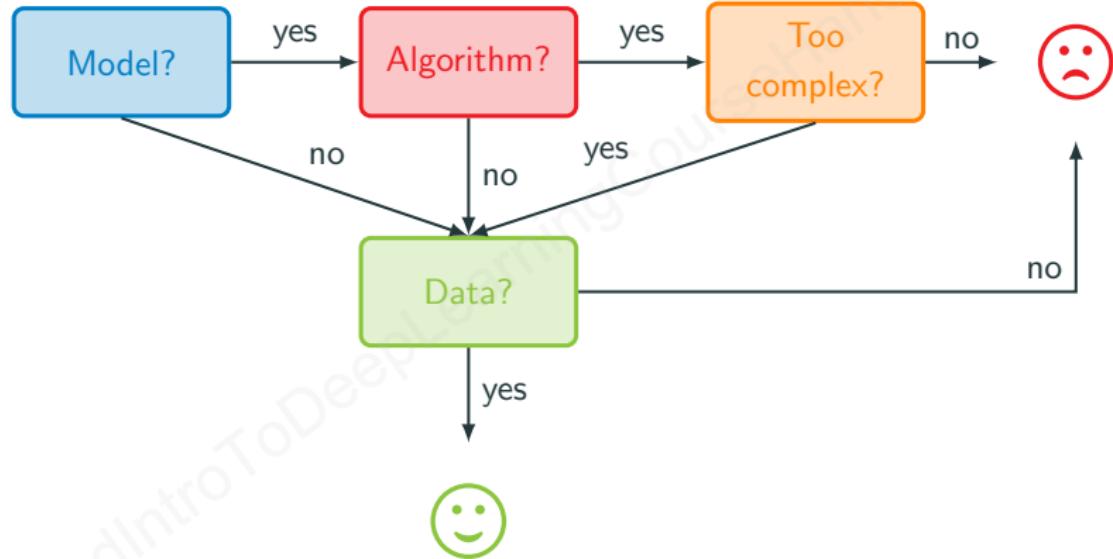
When (not) to use ML?



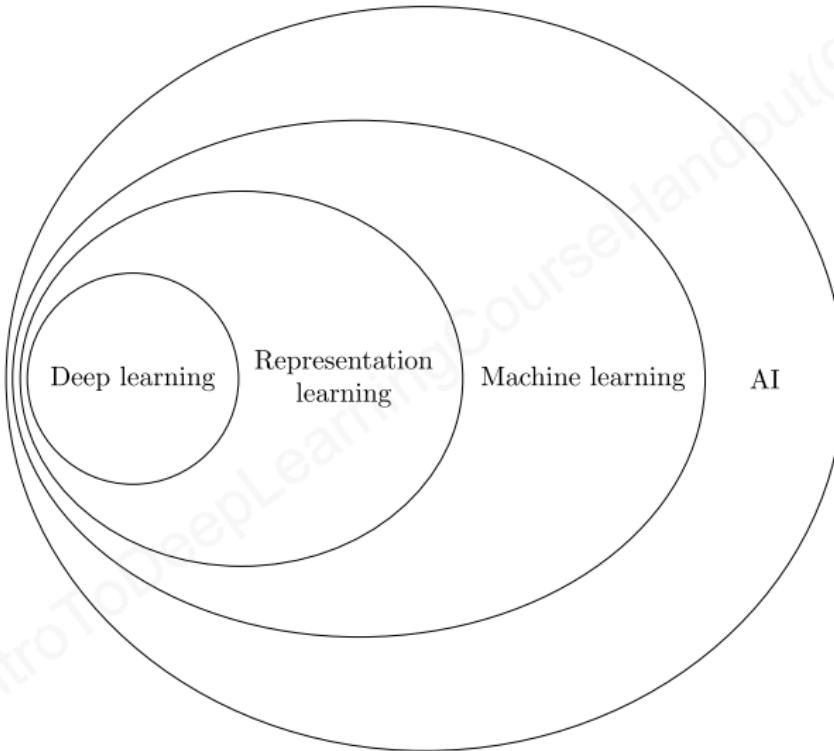
When (not) to use ML?



When (not) to use ML?

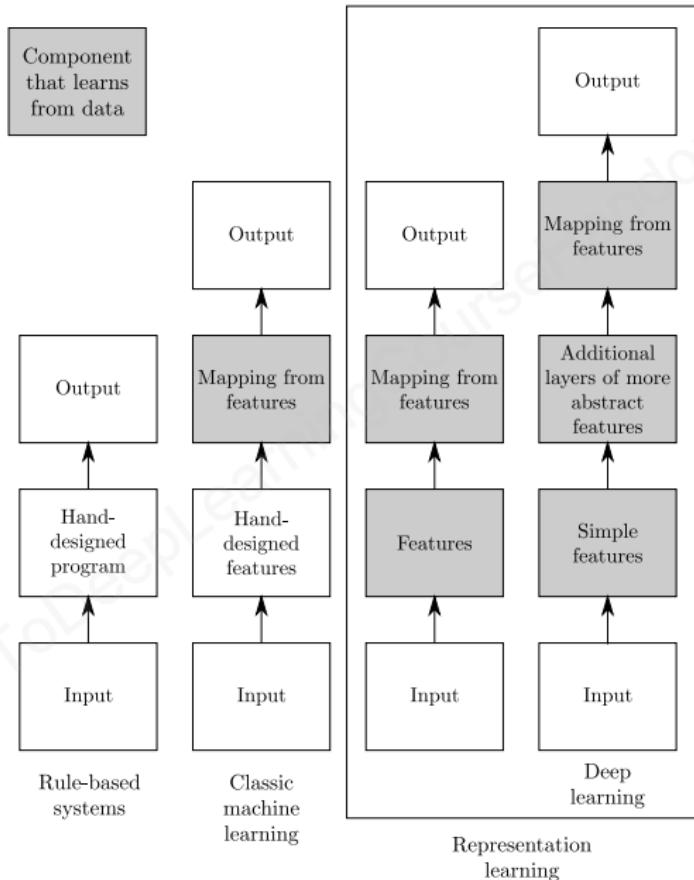


Machine learning and AI



[GBC16][Fig. 1.4]

Relation between AI disciplines



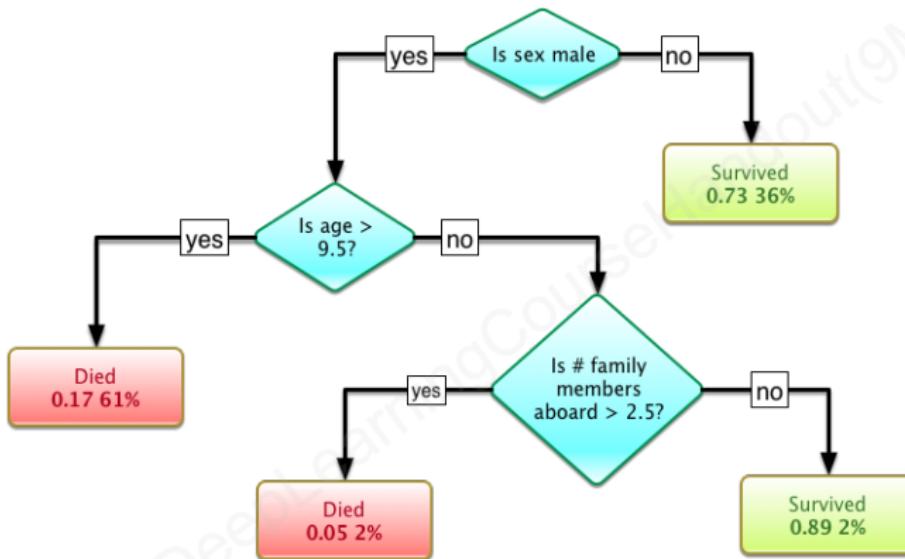
[GBC16][Fig. 1.5]

Rule-based system

```
if temperature < 20 deg and time < 10pm and time > 6am:  
    increase temperature by 1 degree
```

- Also called **expert systems**
- Collection of if–then statements that are used to make decisions

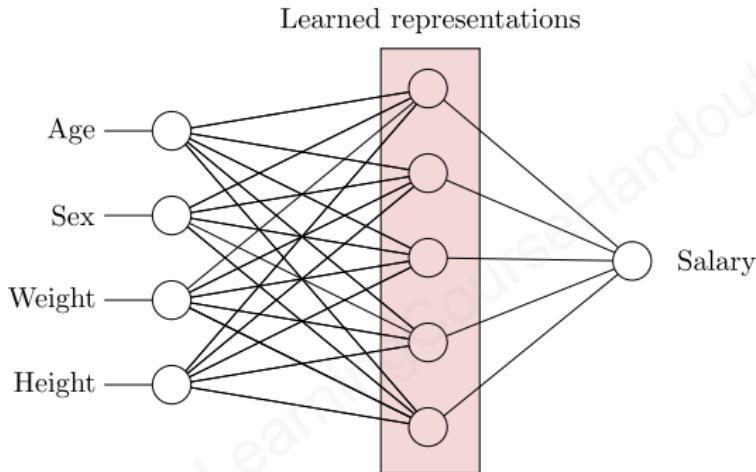
Classic machine learning



- Hand-selected or designed features are combined to make a decision
- The way they features are combined is learned
- Example: Decision trees

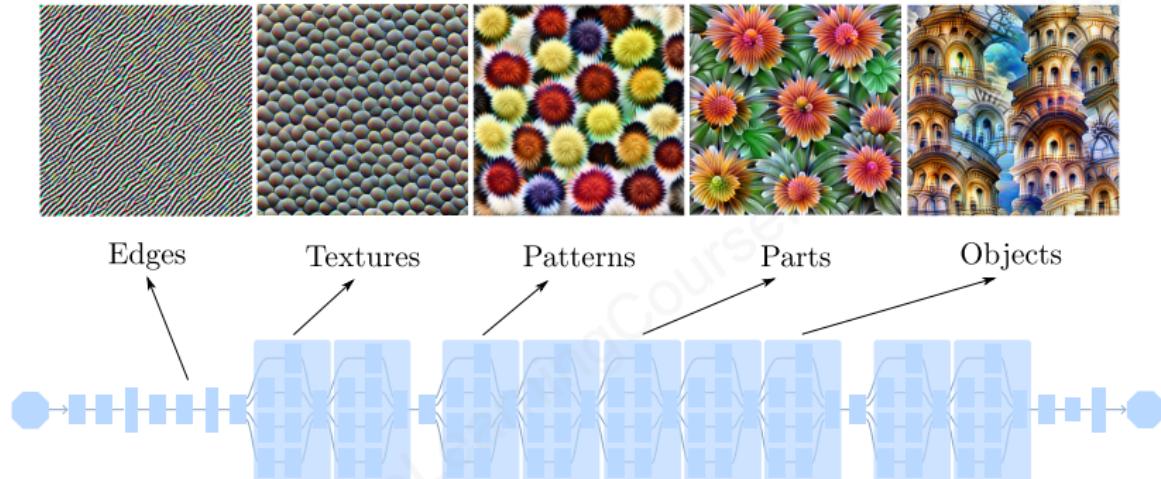
<https://commons.wikimedia.org/w/index.php?curid=62891428>

Representation learning



- Raw features are mapped to representations
- Representations are mapped to the output
- All mappings are learned
- Example: Shallow neural network

Deep learning: From simple to abstract features



- Input-output mapping is broken into nested simpler mappings
- Each mapping generates more abstract feature representations
- All mappings are learned

<https://distill.pub/2017/feature-visualization/>

What is machine learning?

Definition ([Sam59])

ML is a field of computer science that gives computers the ability to learn without being explicitly programmed.

Definition ([Mit97])

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

The task T

- Task T : How should the ML system process an **example**?
- An example is a collection of **features** that are somehow measured
- We represent an example as $\mathbf{x} \in \mathbb{R}^n$, each entry x_i is one feature
- **Classification**

- Specify to which out of k classes \mathbf{x} belongs
- $f : \mathbb{R}^n \mapsto \{1, \dots, k\}$ (or probability distribution over classes)
- Modulation classification, symbol detection, decoding, etc.

- **Regression**

- Predict a numerical value from \mathbf{x}
- $f : \mathbb{R}^n \mapsto \mathbb{R}^k$
- Localization, speed estimation, channel prediction, etc.
- Translation, denoising, density estimation, and many more

Target function

The target function f the ML system should implement is unknown.
The goal of ML is to learn an approximation $\hat{f} \approx f$ from experience.

The experience E

- ML algorithms experience a **dataset**, i.e., a collection of examples
- Unsupervised learning:
 - Dataset $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, \mathbf{x}_i drawn independently from $p_{\text{data}}(\mathbf{x})$
 - Learn useful properties of the data
 - Clustering, density estimation, generative model, etc.
- Supervised learning:
 - Dataset $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)\}$, $(\mathbf{x}_i, \mathbf{y}_i)$ drawn independently from $p_{\text{data}}(\mathbf{x}, \mathbf{y})$
 - Each example \mathbf{x}_i has an associated **label** or **target** \mathbf{y}_i
 - Predict \mathbf{y} from \mathbf{x}
 - Classification, regression, etc.
- Reinforcement learning:
 - Interacts with an environment
 - Feedback loop between the ML algorithm and its experiences
 - Not the focus of this course

The performance measure P

- P measures quantitatively the performance of an ML algorithm
- Depends on the task:
 - Classification — Accuracy or error rate
 - Regression — Mean squared error (MSE)
- Evaluated on examples not used for training, i.e., the **test set**
- Defining P to obtain the right system behavior is often difficult
- If we cannot measure the quantity we are interested in, an alternative criterion that still achieves the design objectives must be found.

Generalization

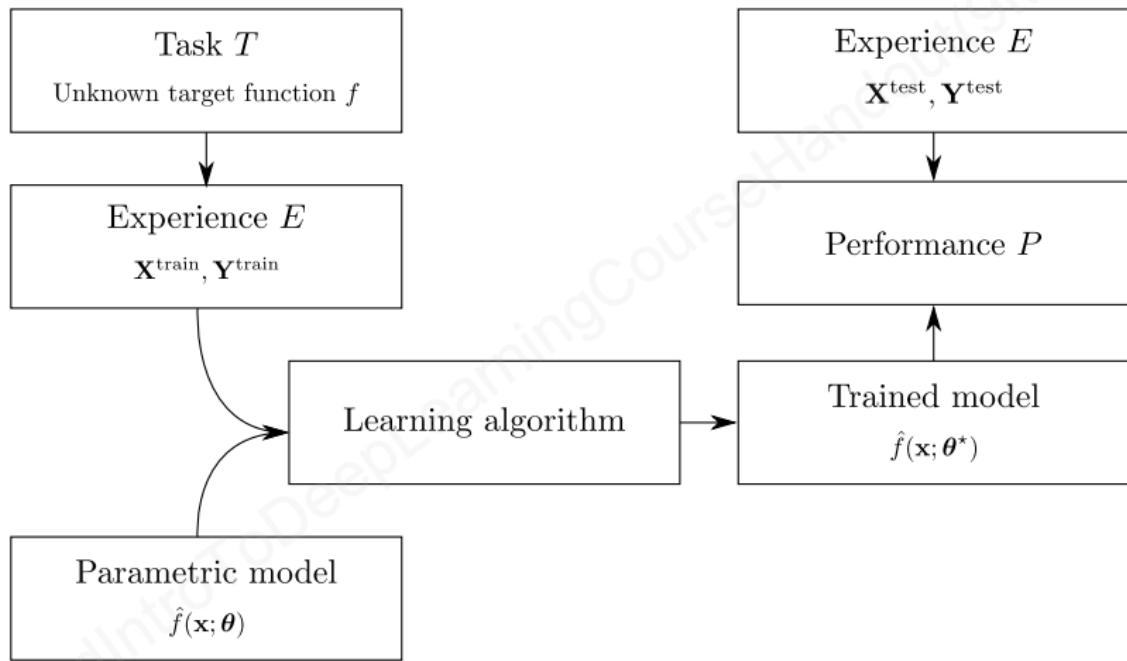
Training set	$\mathbf{X}^{\text{train}} = \{\mathbf{x}_1^{\text{train}}, \mathbf{x}_2^{\text{train}}, \dots\}$	Test set	$\mathbf{X}^{\text{test}} = \{\mathbf{x}_1^{\text{test}}, \mathbf{x}_2^{\text{test}}, \dots\}$
	$\mathbf{Y}^{\text{train}} = \{\mathbf{y}_1^{\text{train}}, \mathbf{y}_2^{\text{train}}, \dots\}$		$\mathbf{Y}^{\text{test}} = \{\mathbf{y}_1^{\text{test}}, \mathbf{y}_2^{\text{test}}, \dots\}$

- **Training set:** Used to train the ML algorithm
- **Training error:** Performance P on training set
- **Test set:** Only(!) used to evaluated its performance
- **Test (generalization) error:** Performance P on test set

Definition (Generalization)

The ML algorithm's ability to perform well on previously unseen inputs
Goal of reducing the test error separates ML from optimization

Components of an ML system



The need for parametric models

- ML requires a **model**, i.e., a mathematical formula with parameters

$$\hat{f}(\mathbf{x}; \boldsymbol{\theta})$$

- The parameters $\boldsymbol{\theta}$ determine the behavior of the chosen model
- Examples:
 - Linear model: $\boldsymbol{\theta}^T \mathbf{x}$
 - Gaussian model: $\mathcal{N}(\mathbf{y}; \boldsymbol{\theta}^T \mathbf{x}, \mathbf{I}_k)$
 - Neural networks
 - Support vector machines

Definition (Multivariate Gaussian distribution)

$$\mathcal{N}(\mathbf{y}; \mathbf{m}, \boldsymbol{\Theta}) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Theta}|}} e^{-\frac{1}{2}(\mathbf{y}-\mathbf{m})^T \boldsymbol{\Theta}^{-1} (\mathbf{y}-\mathbf{m})}$$

Goal of the learning algorithm

- Performance P measured through the **objective function**

$$J(\theta, \mathbf{X}^{\text{test}}, \mathbf{Y}^{\text{test}})$$

- J is called **loss (reward)** if it should be minimized (maximized)
- Goal of learning:

$$\theta^* = \operatorname{argmin}_{\theta} J(\theta, \mathbf{X}^{\text{test}}, \mathbf{Y}^{\text{test}})$$

without using $\mathbf{X}^{\text{test}}, \mathbf{Y}^{\text{test}}$

- What we do in practice:

$$\theta^* = \operatorname{argmin}_{\theta} J(\theta, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}})$$

and hope that the solution generalizes well!

Lessons learned

Introduction & Basics of Machine Learning



- ML for communications is a rather novel field. Many encouraging results, but practical viability to be shown.
 - Pro: Compensate for mismatch between model and reality
 - Con: We are close to Shannon in many cases
- ML shines for problems with a model or algorithm deficit.
- Deep learning (DL) is a sub-field of representation learning: more and more abstract features learned from raw inputs.
- An ML system solves a task T , described by the target function $f(\mathbf{x})$, and improves its performance P with experience E .
- The learning algorithm seeks to find good parameters θ^* of a parametric model $\hat{f}(\mathbf{x}; \theta)$.

Primer on Deep Learning

Neural networks and deep learning

- A neural network (NN) defines a mapping

$$\hat{f}(\mathbf{x}; \boldsymbol{\theta}) : \mathbb{R}^n \rightarrow \mathbb{R}^k$$

of an input vector $\mathbf{x} \in \mathbb{R}^n$ to an output vector $\hat{\mathbf{y}} \in \mathbb{R}^k$

- $\boldsymbol{\theta}$ are the parameters that determine the behavior of the NN
- DL describes the process of finding good values for $\boldsymbol{\theta}$ from data to achieve a desired behavior
- Examples:
 - Classify handwritten digits
 - Predict stock prices
 - Detect modulated symbols from noisy observations

Feedforward neural networks

- A feed-forward NN is a mapping

$$\hat{f}(\mathbf{x}_0; \boldsymbol{\theta}) : \mathbb{R}^{N_0} \mapsto \mathbb{R}^{N_L}$$

from an input vector $\mathbf{x}_0 \in \mathbb{R}^{N_0}$ to an output vector $\mathbf{x}_L \in \mathbb{R}^{N_L}$

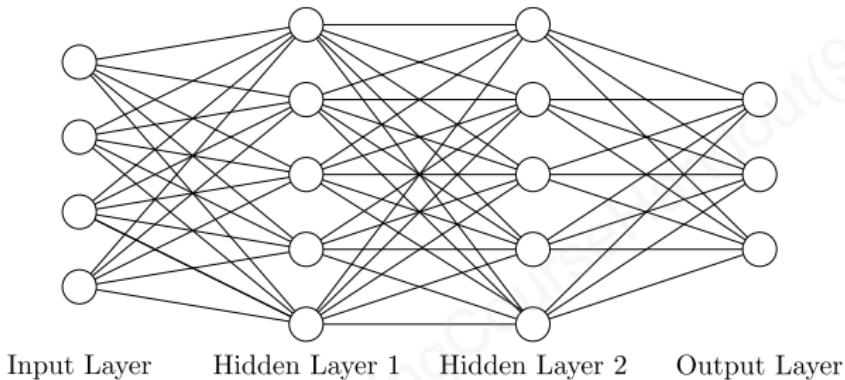
- This mapping is implemented through L iterative processing steps

$$\mathbf{x}_\ell = \hat{f}_\ell(\mathbf{x}_{\ell-1}; \boldsymbol{\theta}_\ell), \quad \ell = 1, \dots, L$$

where $\hat{f}_\ell(\mathbf{x}_{\ell-1}; \boldsymbol{\theta}_\ell) : \mathbb{R}^{N_{\ell-1}} \mapsto \mathbb{R}^{N_\ell}$ is the mapping of the ℓ th layer

- It is called **feedforward** because information flows from layer to layer without feedback connections
- $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_L\}$ is the set of all parameters of the NN; $\boldsymbol{\theta}_\ell$ are the parameters of the ℓ th layer

Dense or fully connected layers



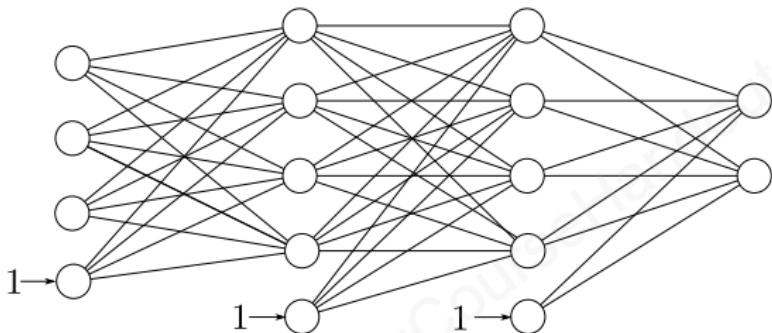
- A **dense** or **fully-connected** layer has the form

$$\hat{f}_\ell(\mathbf{x}_{\ell-1}; \boldsymbol{\theta}_\ell) = \sigma(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell)$$

where $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ are the **weights**, $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$ is the **bias**, and $\sigma(\cdot)$ is a non-linear **activation** function (more about this later)

- The set of parameters for this layer is $\boldsymbol{\theta}_\ell = \{\mathbf{W}_\ell, \mathbf{b}_\ell\}$
- $\theta_{\ell,i}$ is the i th parameter of the ℓ th layer (ordering unimportant)

Bias-trick and multilayer perceptrons



- An NN composed of only fully connected layers is called a multilayer perceptron (MLP)
- Useful trick to avoid writing the bias \mathbf{b}_ℓ :

$$\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell = \underbrace{\begin{bmatrix} \mathbf{W}_\ell & \mathbf{b}_\ell \end{bmatrix}}_{\tilde{\mathbf{w}}_\ell} \begin{bmatrix} \mathbf{x}_{\ell-1} \\ 1 \end{bmatrix} = \tilde{\mathbf{w}}_\ell \tilde{\mathbf{x}}_{\ell-1}$$

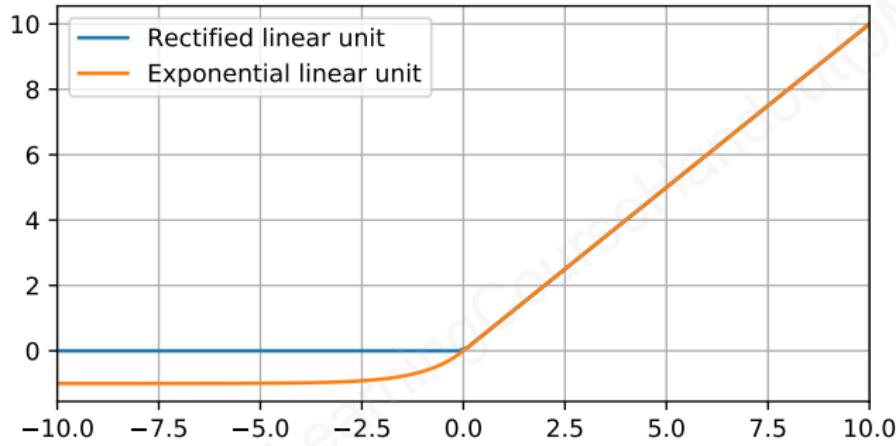
Activation functions for hidden layers

- Generally, the activation function is applied element-wise, i.e.,

$$[\sigma(\mathbf{x})]_i = \sigma(x_i)$$

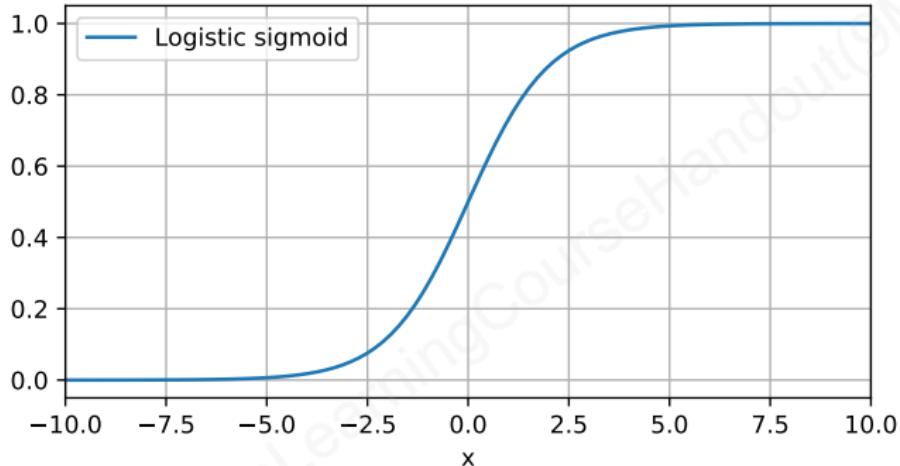
- $\sigma(\cdot)$ introduces a **non-linearity** which is very important for the expressive power of the NN
- Without this non-linearity there would be no advantage of stacking multiple layers on top of each other (Why?)
- The choice of activation function is an active area of research. There are not many theoretical guidelines of how to choose them.
- If you are not scared of maths:
<https://arxiv.org/abs/1706.02515>

Rectified linear unit (ReLU)



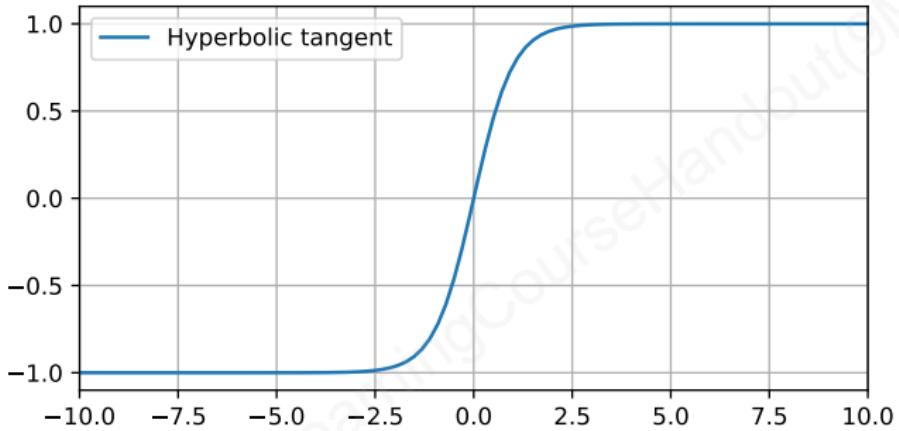
- $\sigma(x) = \text{ReLU}(x) \triangleq \max(0, x) \in [0, \infty)$
- Rectified linear units (ReLUs) are a good default choice
 - Simple gradient
 - No saturation
 - Nondifferentiability at $x = 0$ practically irrelevant
- Generalizations: leaky ReLU, exponential ReLU (ELU),...

Logistic sigmoid



- $\sigma(x) = \text{sigmoid}(x) \triangleq \frac{1}{1+\exp(-x)} \in (0, 1)$
- Widely used before ReLUs were introduced
- Only sensitive when input near to 0
- Saturation makes gradient based learning difficult
- Not recommended for use in hidden layers
- Nice property: $1 - \text{sigmoid}(x) = \text{sigmoid}(-x)$

Hyperbolic tangent



- $\sigma(x) = \tanh(x) \triangleq \frac{e^x - e^{-x}}{e^x + e^{-x}} \in (-1, 1)$
- Related to sigmoid(x) as

$$\tanh(x) = 2 \text{sigmoid}(2x) - 1$$

- Performance generally better than sigmoid due to its similarity with the identity function near 0

Gradient descent

$$\text{Goal: } \mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} g(\mathbf{x}), \quad g : \mathbb{R}^n \mapsto \mathbb{R}$$

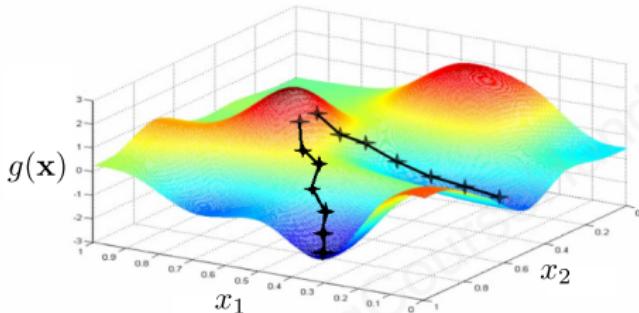
Definition (Gradient descent)

Find a local minimum of the objective function g by iterating:

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \nabla_{\mathbf{x}} g(\mathbf{x}^t), \quad t = 1, 2, \dots$$

- Given initial value \mathbf{x}^0
- Learning rate $\eta > 0$

Some comments on gradient descent



- In general, the objective is non-convex and converging to a local minimum is the best that we can hope for
- Choosing a suitable learning rate η is difficult:
 - too small \rightarrow slow convergence
 - too large \rightarrow fluctuations or divergence

Figure: http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html.

Stochastic gradient descent (SGD)

- Training an NN using gradient descent is costly

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}})$$

because each iteration requires the computation of N gradients

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}}) = \frac{1}{N} \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} L \underbrace{\left(\mathbf{y}_i^{\text{train}}, \hat{f}(\boldsymbol{\theta}^t, \mathbf{x}_i^{\text{train}}) \right)}_{\text{per-example loss}}$$

Definition (Stochastic gradient descent (SGD))

SGD approximates the gradient of the objective function by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}}) \approx \frac{1}{|\mathbb{B}^t|} \sum_{i \in \mathbb{B}^t} \nabla_{\boldsymbol{\theta}} L \left(\mathbf{y}_i^{\text{train}}, \hat{f}(\boldsymbol{\theta}^t, \mathbf{x}_i^{\text{train}}) \right)$$

based on a randomly drawn minibatch $\mathbb{B}^t \subset \{1, \dots, N\}$ for each t

Activation functions for the output layer

- Let $\hat{y} = \sigma(\mathbf{h})$, i.e., \mathbf{h} is the output before the activation
- $\sigma(\mathbf{h})$ determines the domain of \hat{y} and how it must be interpreted
- Very related to the objective (or loss/reward) function
- Should be smooth and differentiable, e.g., not a step function

Linear activation

- $\sigma(\mathbf{h}) = \mathbf{h}$
- Used for regression tasks, i.e., $\hat{\mathbf{y}} \in \mathbb{R}^k$
- Examples:
 - Estimate position of an object
 - Predict stock prize
 - Wireless channel prediction
- Typically used with the MSE loss function:

$$J(\boldsymbol{\theta}, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}}) = \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} \|\mathbf{y} - \hat{f}(\mathbf{x}; \boldsymbol{\theta})\|_2^2 \right]$$

Sigmoid activation

- $\sigma(\mathbf{h}) = \text{sigmoid}(\mathbf{h})$
- Used to predict the value of a binary random variable, i.e., $\hat{y} \in (0, 1)$:

$$\hat{y} = \Pr(y = 1 | \mathbf{x}) = \text{sigmoid}(h) = \hat{f}(\mathbf{x}; \theta)$$

- h is called a **logit**
- Widely used with the cross-entropy loss:

$$J(\theta, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}}) \\ = - \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \left[\underbrace{y \log(\hat{f}(\mathbf{x}; \theta)) + (1 - y) \log(1 - \hat{f}(\mathbf{x}; \theta))}_{p_{\text{model}}(y | \mathbf{x}; \theta)} \right]$$

Softmax activation

- $\sigma(\mathbf{h}) = \text{softmax}(\mathbf{h}) \triangleq \left[\frac{e^{h_i}}{\sum_{j=1}^k e^{h_j}} \right]_{i=1}^k$
- $\hat{\mathbf{y}} \in \mathbb{R}^k$ is a probability vector, i.e., $\sum_{i=1}^k \hat{y}_i = 1$
- Used to represent a random variable with k possible values
- Examples
 - Classification of handwritten digits
 - Classification of modulation types
 - Classification of modulated signals/messages
- Typically used with the categorical (or softmax) cross entropy cost:

$$J(\theta, \mathbf{X}^{\text{train}}, \mathbf{Y}^{\text{train}}) = \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \left[- \underbrace{\sum_{i=1}^k y_i \log \left(\left[\hat{f}(\mathbf{x}; \theta) \right]_i \right)}_{-\log(p_{\text{model}}(\mathbf{y}|\mathbf{x}; \theta))} \right]$$

where \mathbf{y} is a **one-hot** vector encoding the class c , i.e., an all-zero vector with the c th element equal to one

Universal approximation theorem

Theorem ([Cyb89, HSW89])

Let $\sigma()$ be a bounded, non-constant continuous function. Let I_m denote the m -dimensional hypercube¹, and $C(I_m)$ denote the space of continuous functions on I_m . Given any $f \in C(I_m)$ and $\varepsilon > 0$, there exists $N > 0$ and $\mathbf{W} \in \mathbb{R}^{N \times m}$, $\mathbf{b} \in \mathbb{R}^N$, and $\mathbf{v} \in \mathbb{R}^N$, such that

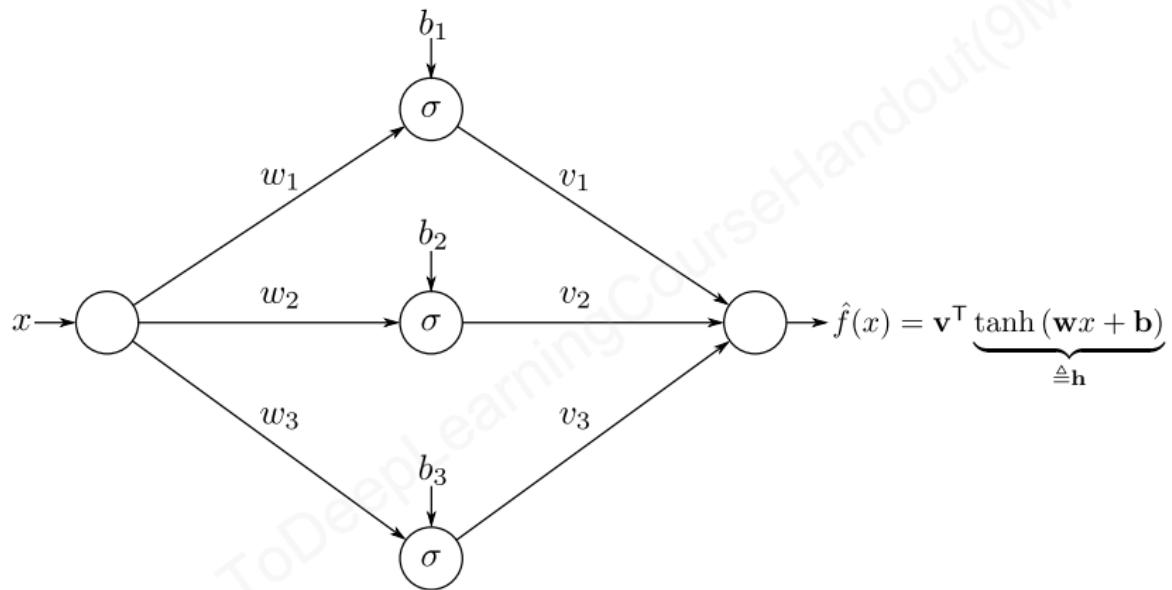
$$\hat{f}(\mathbf{x}) = \mathbf{v}^\top \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\sup_{\mathbf{x} \in I_m} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \varepsilon.$$

- A single hidden-layer NN can approximate any continuous function on a bounded set arbitrary closely
- The theorem does not tell us anything about the required size N or how to find the right parameters $\mathbf{W}, \mathbf{b}, \mathbf{v}$

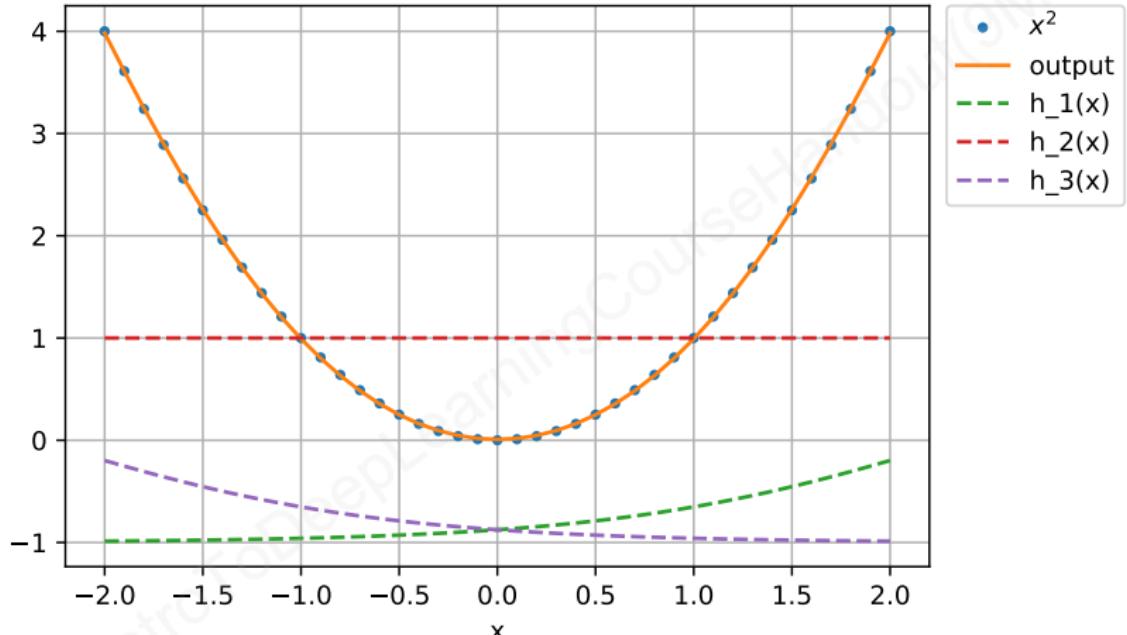
¹The m -dimensional hypercube is the convex hull of all the points $\{\pm\frac{1}{2}, \pm\frac{1}{2}, \dots, \pm\frac{1}{2}\}$ (over all sign-permutations). It has unit side-length and volume.

Example: MLP approximation of functions on \mathbb{R}



Jupyter Notebook

Example: MLP approximation of $f(x) = x^2$



$$\mathbf{v} = [7.02, 12.3, 7.02]^T$$

Approximation and estimation bound

Theorem ([Bar94])

If the parameters of $\hat{f}()$ are estimated from K training examples and $f()$ satisfies a certain smoothness condition, then

$$\int_{I_m} \left| f(\mathbf{x}) - \hat{f}(\mathbf{x}) \right|^2 d\mathbf{x} = \mathcal{O}\left(\frac{1}{N}\right) + \mathcal{O}\left(\frac{Nm}{K} \log(K)\right).$$

- The universal approximation theorem tells us that the approximation error can be made as small as desired by increasing the number of neurons N
- On the other hand, a large number of nodes makes it difficult to estimate the optimal parameters $(\mathbf{W}, \mathbf{b}, \mathbf{v})$ of the NN
- The above theorem addresses the combined effect of approximation and estimation error, i.e., the more accurate we want to be, the larger the network must be, and the more training data we need

Gradient descent is asymptotically globally optimal

Theorem ([DZPS18])

Let $\sigma = \text{ReLU} = \max(0, \cdot)$ and assume that the parameters $\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{v}\}$ are randomly initialized and then updated according to gradient descent on the MSE loss function $J(\theta)$ computed on K training examples. Then, for every $\varepsilon > 0$, there exist an N and a number of iterations t , such that

$$J(\theta_t) \leq \varepsilon, \quad t = \mathcal{O}(\log(1/\varepsilon)).$$

- For a large enough NN, gradient descent converges to a globally optimal solution at linear rate
- Proofs was extended to a general class of activation functions, MLPs, and residual convolutional neural networks (ConvNets) [DLL⁺19]
- Result holds for the training loss; nothing about the test loss is said
- Probably holds for SGD and other loss functions, too

The backpropagation algorithm

- **Goal:** Compute $\frac{\partial}{\partial \theta_i} L \left(\mathbf{y}_i^{\text{train}}, \hat{f}(\boldsymbol{\theta}, \mathbf{x}_i^{\text{train}}) \right) \quad \forall i$
- **Chain rule of calculus:** Let $y = g(x)$ and $z = h(y)$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

- The backpropagation (backprop) algorithm recursively computes efficiently the derivatives of the loss function with respect to (w.r.t.) all parameters using the chain rule, starting from the last layer
- All modern machine learning libraries provide efficient implementations of this algorithm
- Not a ML algorithm, only computes derivatives
- For a great explanation, see this video by A. Karpathy

Gradient descent optimization algorithms (1/2)

- **Momentum [Qia99]:**

Avoid strong gradient fluctuations in a single dimension

$$\theta^{t+1} = \theta^t - \mathbf{m}_t$$

$$\mathbf{m}_t = \gamma \mathbf{m}_{t-1} + \eta \nabla_{\theta} J(\theta_t)$$

- **Adagrad [DHS11]:**

Adapt η to individual parameters θ_i and decrease it

$$\theta_i^{t+1} = \theta_i^t - \frac{\eta}{\sqrt{v_t^i + \varepsilon}} [\nabla_{\theta} J(\theta_t)]_i$$

$$v_t^i = \sum_{t'=0}^t [\nabla_{\theta} J(\theta_{t'})]_i^2$$

Nice visualization and summary:

<http://ruder.io/optimizing-gradient-descent/>

Gradient descent optimization algorithms (2/2)

- **RMSprop [Hin16]:**

Exponentially decaying average over squared gradients

$$\theta_i^{t+1} = \theta_i^t - \frac{\eta}{\sqrt{v_t^i + \varepsilon}} [\nabla_{\theta} L(\theta_t)]_i$$
$$v_t^i = \gamma v_{t-1}^i + (1 - \gamma) [\nabla_{\theta} L(\theta_t)]_i^2$$

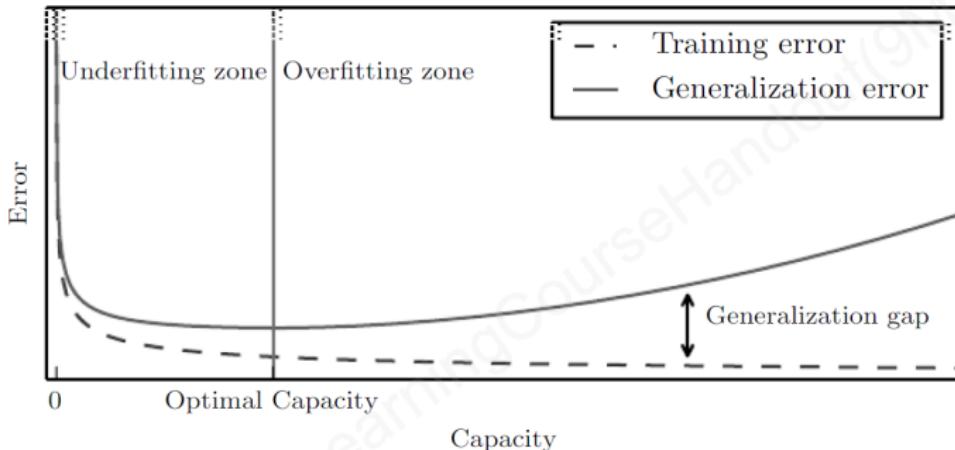
- **Adam [KB14]:**

Exponentially decaying average over gradients and squared gradients

$$\theta_i^{t+1} = \theta_i^t - \frac{\eta \hat{m}_t^i}{\sqrt{\hat{v}_t^i + \varepsilon}}, \quad \hat{m}_t^i = \frac{m_t^i}{1 - \beta_1^t}, \quad \hat{v}_t^i = \frac{v_t^i}{1 - \beta_2^t}$$
$$m_t^i = \beta_1 m_{t-1}^i + (1 - \beta_1) [\nabla_{\theta} L(\theta_t)]_i$$
$$v_t^i = \beta_2 v_{t-1}^i + (1 - \beta_2) [\nabla_{\theta} L(\theta_t)]_i^2$$

- Adam is a good default choice

Capacity, over- and underfitting



- Challenge **generalization**: Perform well on unseen inputs
- **Underfitting**: Test error does not decrease sufficiently
- **Overfitting**: Gap between training and test error too large
- The **capacity²** of a NN determines if it under- or overfits

²Many definitions, e.g., Vapnik-Chervonenkis (VC) dimensions [Vap13]. Often relates to the number of free parameters (trainables weights) of a NN.

Figure taken from [GBC16, Fig. 5.3]

Lessons learned

A Primer on Deep Learning

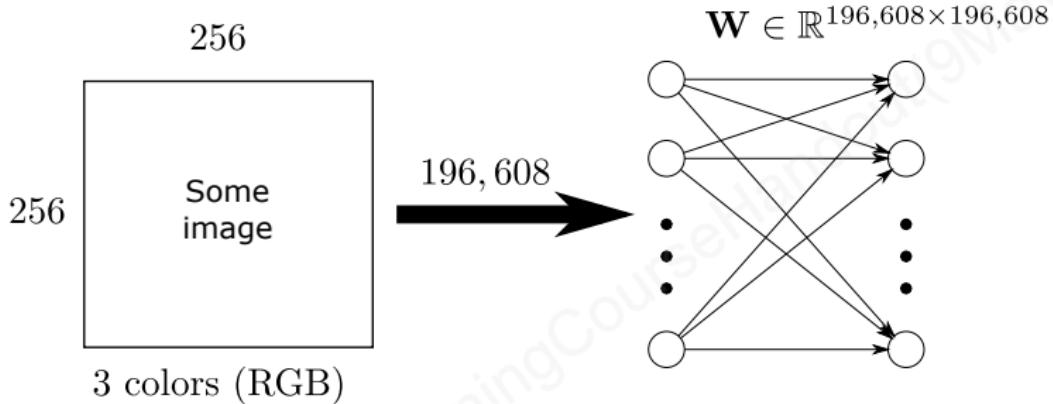


- NNs are good parametric functions for ML because they are universal function approximators.
- SGD is the most widely used learning algorithm for NNs. Many variants exist which ensure faster convergence.
- Backpropagation is used to efficiently compute gradients.
- Activation functions of hidden layers introduce non-linearities which augment the expressive power of a NN. The output activation function depends on the loss function.
- Regularization is any method that reduces test error at the expense of increased training error.

Advanced Neural Network Architectures

Convolutional Neural Networks & ResNets

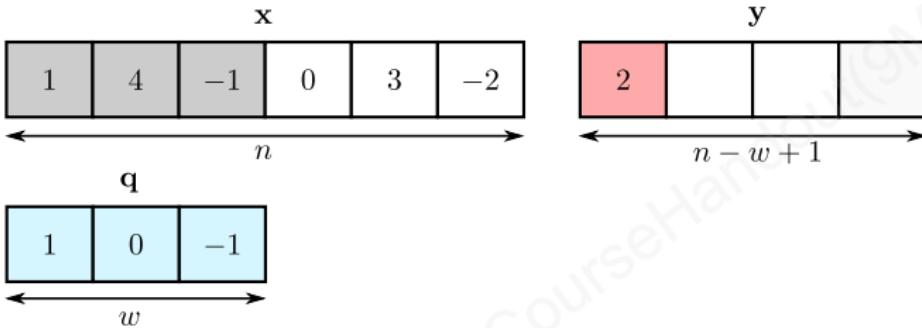
Motivation



- Treating large-dimensional signals (sound samples/images) as **unstructured** vectors requires models of intractable size
- Such large signals have some “invariance in translation”
- Representations meaningful at a certain location can & should be used everywhere
- ConvNets embody this idea
- Good intro:

<https://cs231n.github.io/convolutional-networks/>

Convolutional layer in one dimension

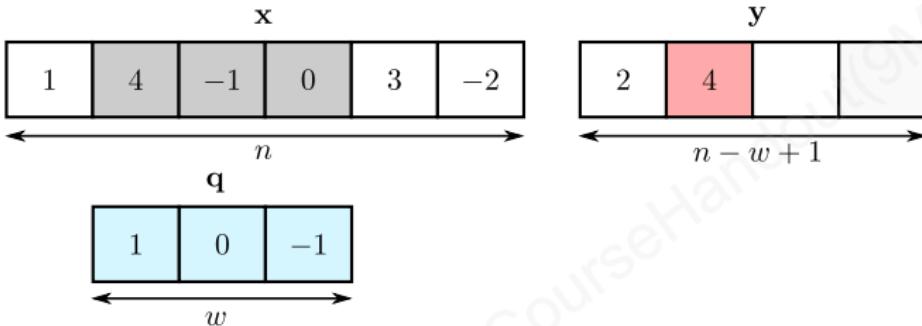


- For input $\mathbf{x} \in \mathbb{R}^n$ and **filter** (kernel) $\mathbf{q} \in \mathbb{R}^w$ of **width** w :

$$y_i = \sum_{j=1}^w x_{i+j-1} q_j, \quad i = 1, \dots, n - w + 1$$

- Output $\mathbf{y} \in \mathbb{R}^{n-w+1}$ is called **feature map**
- Similar to usual convolution, but increasing index order for \mathbf{x}, \mathbf{q}
- A filter looks for presence of a certain pattern/feature in the input:
Like the scalar product between \mathbf{q} and slices of \mathbf{x} , $|y_i|$ large when both are “parallel”

Convolutional layer in one dimension

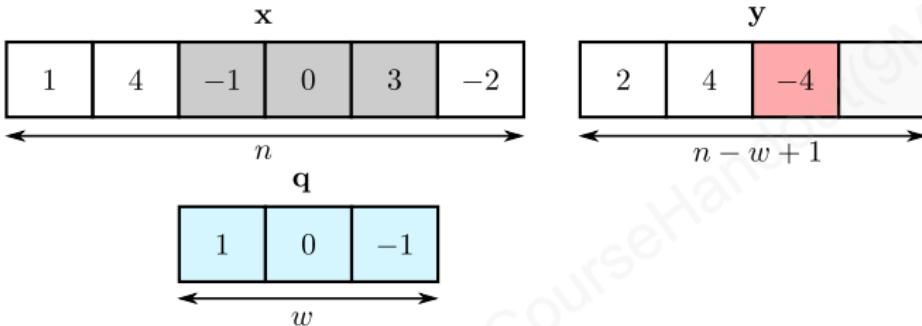


- For input $\mathbf{x} \in \mathbb{R}^n$ and **filter** (kernel) $\mathbf{q} \in \mathbb{R}^w$ of **width** w :

$$y_i = \sum_{j=1}^w x_{i+j-1} q_j, \quad i = 1, \dots, n - w + 1$$

- Output $\mathbf{y} \in \mathbb{R}^{n-w+1}$ is called **feature map**
- Similar to usual convolution, but increasing index order for \mathbf{x}, \mathbf{q}
- A filter looks for presence of a certain pattern/feature in the input:
Like the scalar product between \mathbf{q} and slices of \mathbf{x} , $|y_i|$ large when both are “parallel”

Convolutional layer in one dimension

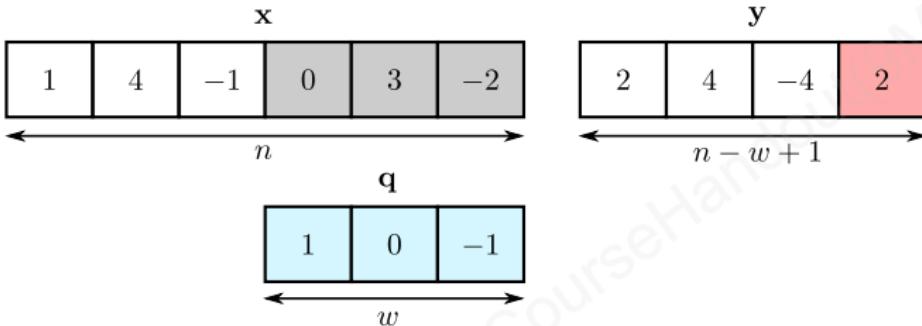


- For input $\mathbf{x} \in \mathbb{R}^n$ and **filter** (kernel) $\mathbf{q} \in \mathbb{R}^w$ of **width** w :

$$y_i = \sum_{j=1}^w x_{i+j-1} q_j, \quad i = 1, \dots, n - w + 1$$

- Output $\mathbf{y} \in \mathbb{R}^{n-w+1}$ is called **feature map**
- Similar to usual convolution, but increasing index order for \mathbf{x}, \mathbf{q}
- A filter looks for presence of a certain pattern/feature in the input:
Like the scalar product between \mathbf{q} and slices of \mathbf{x} , $|y_i|$ large when both are “parallel”

Convolutional layer in one dimension



- For input $\mathbf{x} \in \mathbb{R}^n$ and **filter** (kernel) $\mathbf{q} \in \mathbb{R}^w$ of **width** w :

$$y_i = \sum_{j=1}^w x_{i+j-1} q_j, \quad i = 1, \dots, n-w+1$$

- Output $\mathbf{y} \in \mathbb{R}^{n-w+1}$ is called **feature map**
- Similar to usual convolution, but increasing index order for \mathbf{x}, \mathbf{q}
- A filter looks for presence of a certain pattern/feature in the input:
Like the scalar product between \mathbf{q} and slices of \mathbf{x} , $|y_i|$ large when both are “parallel”

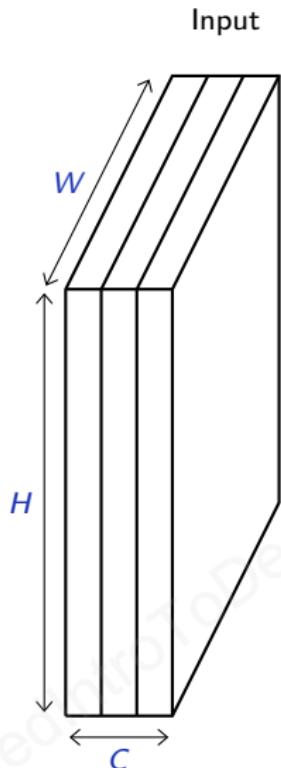
Convolutional layers with arbitrary dimensions

- Concept generalizes to input tensors with arbitrary dimensions
- In computer vision, we have typically three dimensions:

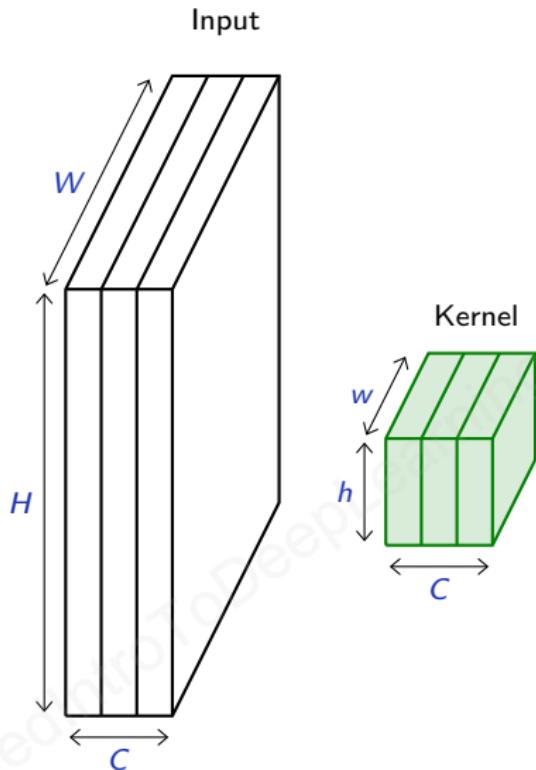
width \times height \times number of color channels

- Each filter $\mathbf{Q} \in \mathbb{R}^{w \times h \times c}$ produces a two-dimensional feature map
- Multiple filters are combined to produce a three-dimensional output (by stacking the feature maps)
- The number of different filters F is called the **depth**

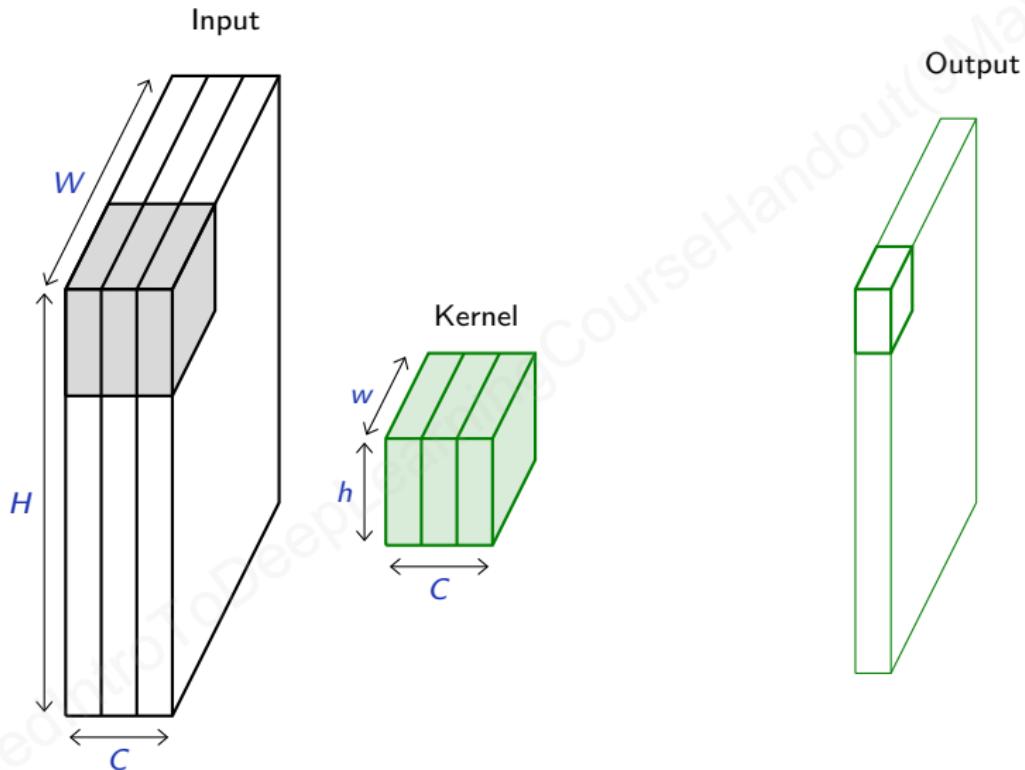
Convolutional layers visualization: 3D inputs



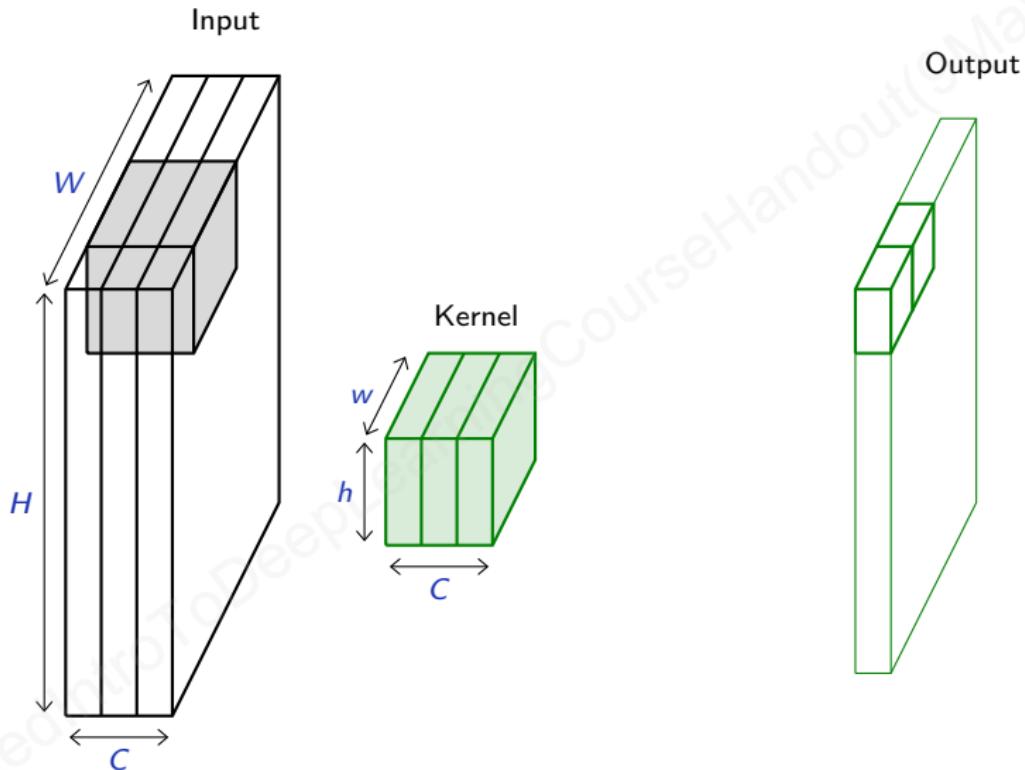
Convolutional layers visualization: 3D inputs



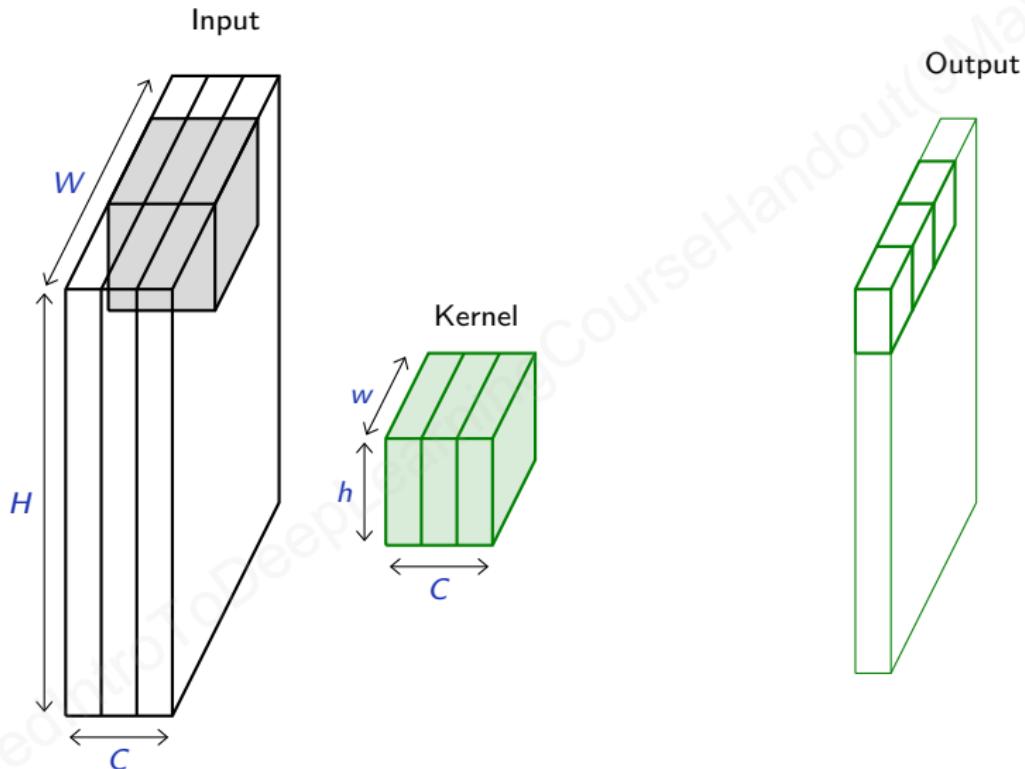
Convolutional layers visualization: 3D inputs



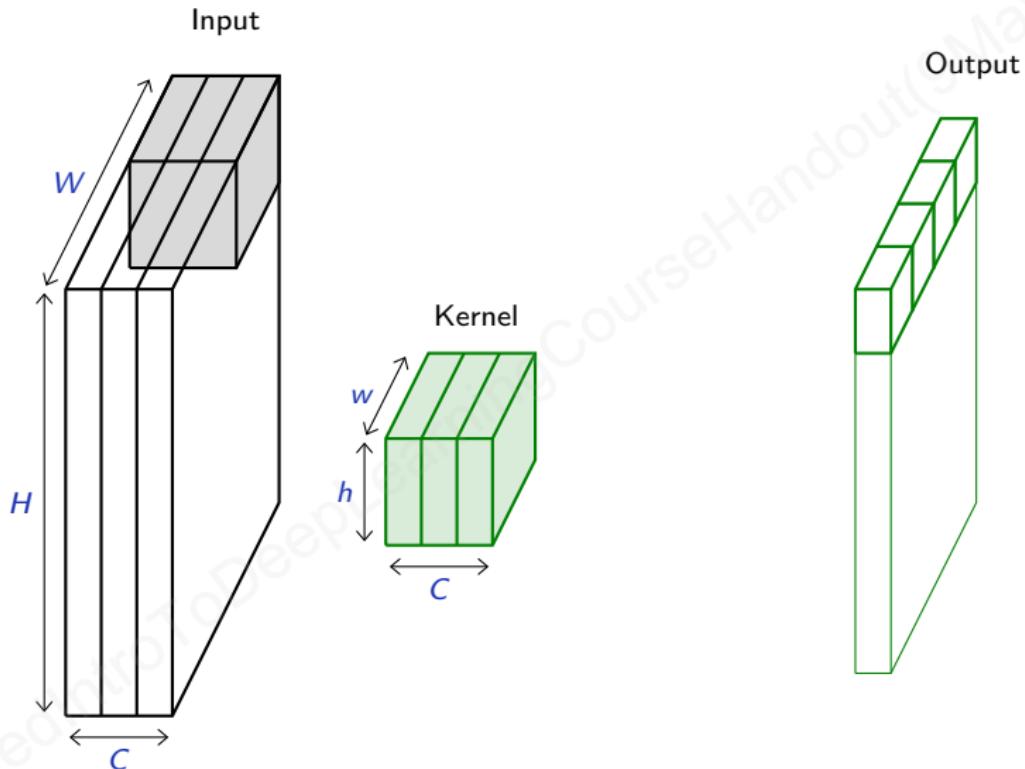
Convolutional layers visualization: 3D inputs



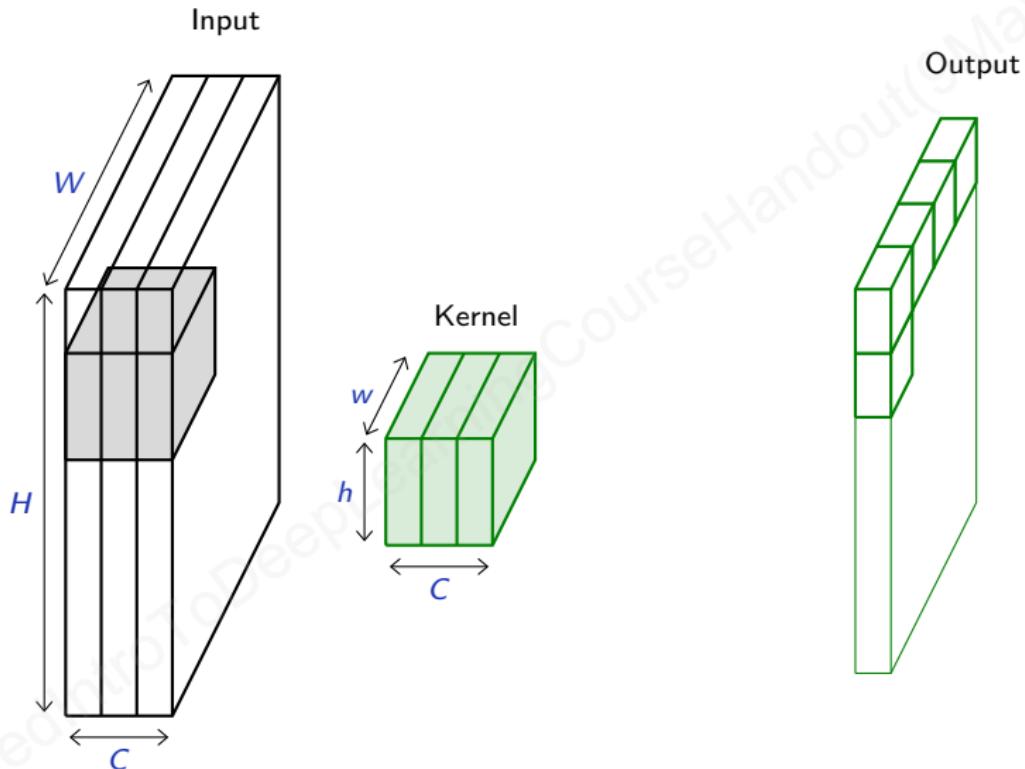
Convolutional layers visualization: 3D inputs



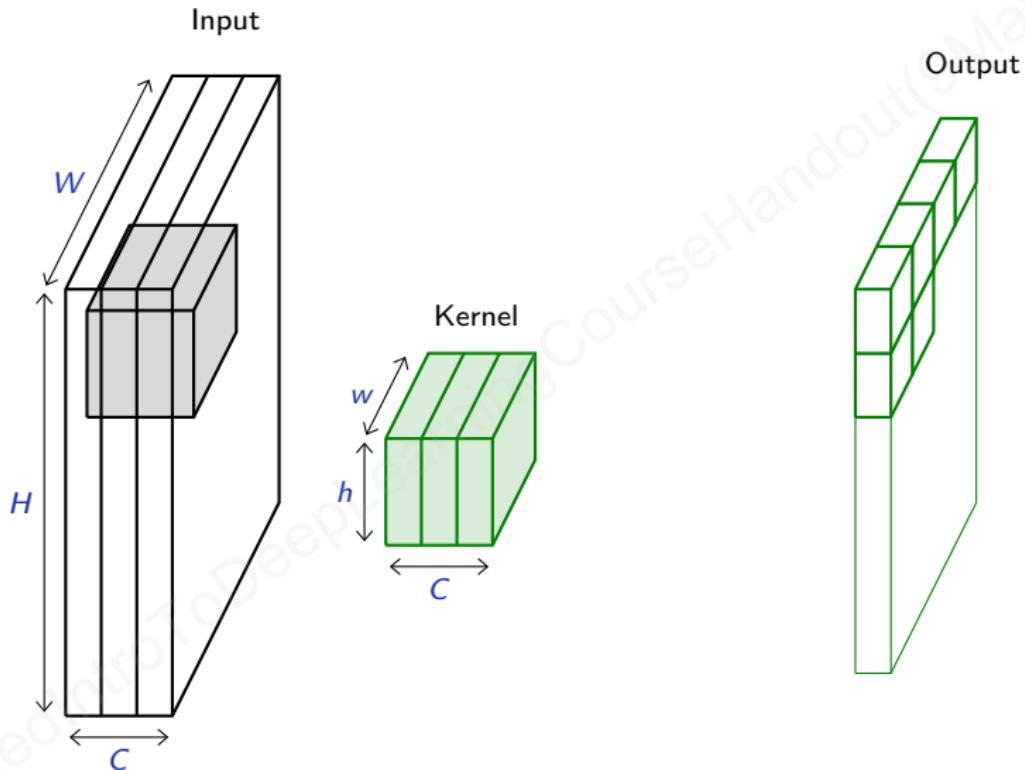
Convolutional layers visualization: 3D inputs



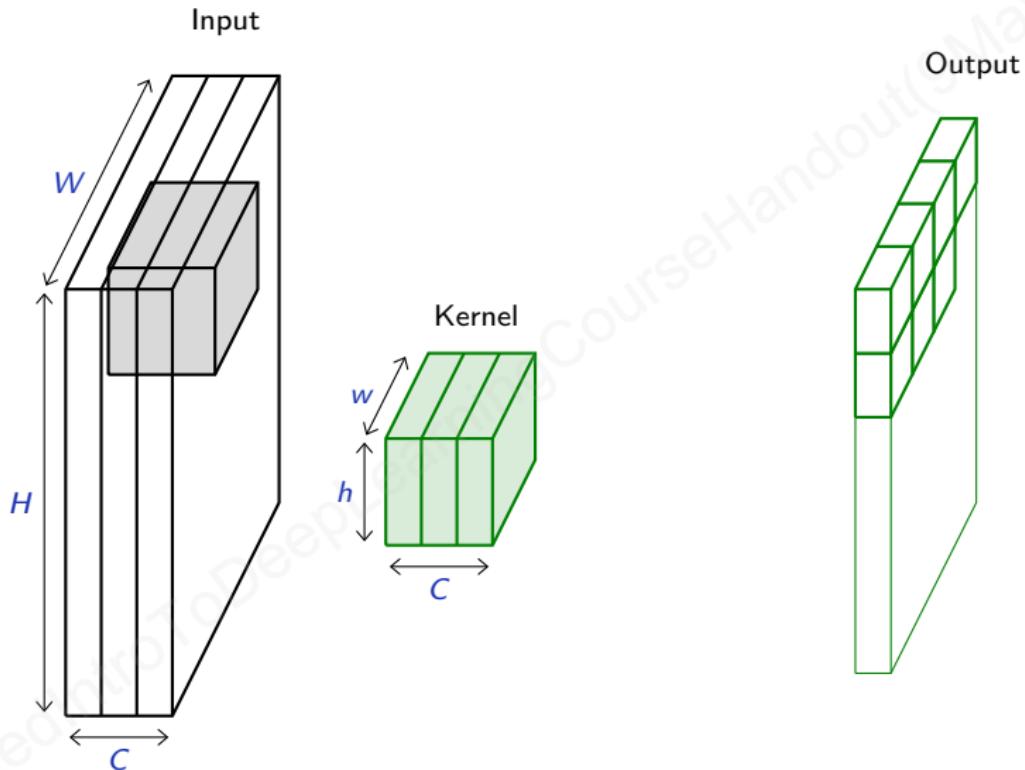
Convolutional layers visualization: 3D inputs



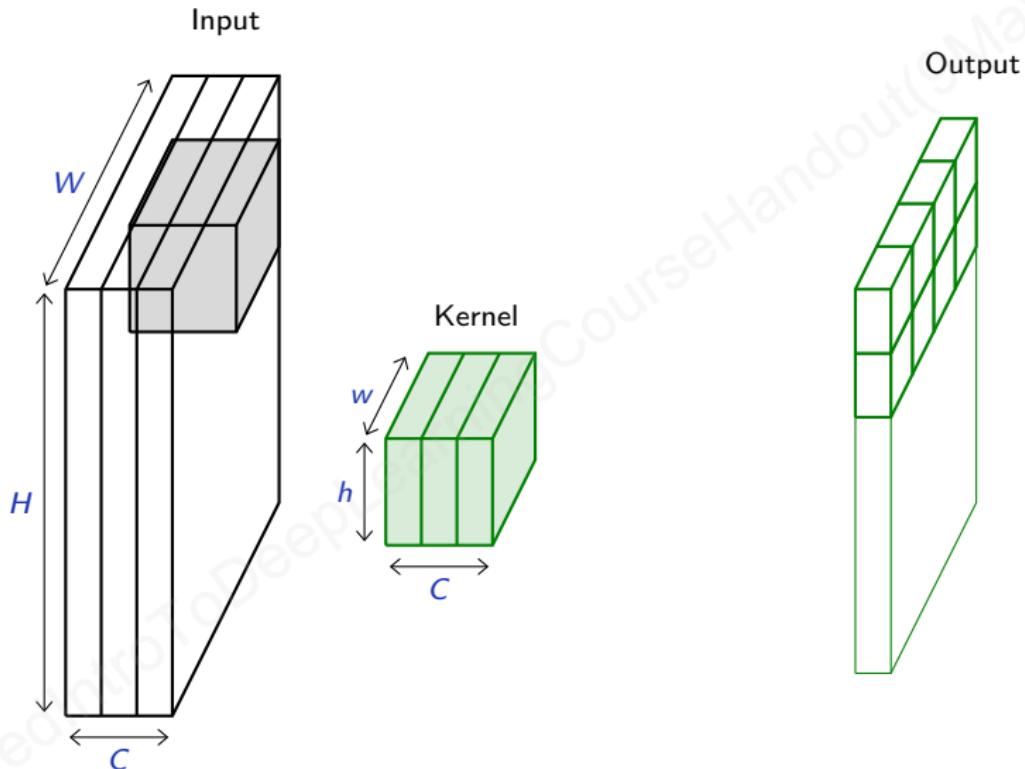
Convolutional layers visualization: 3D inputs



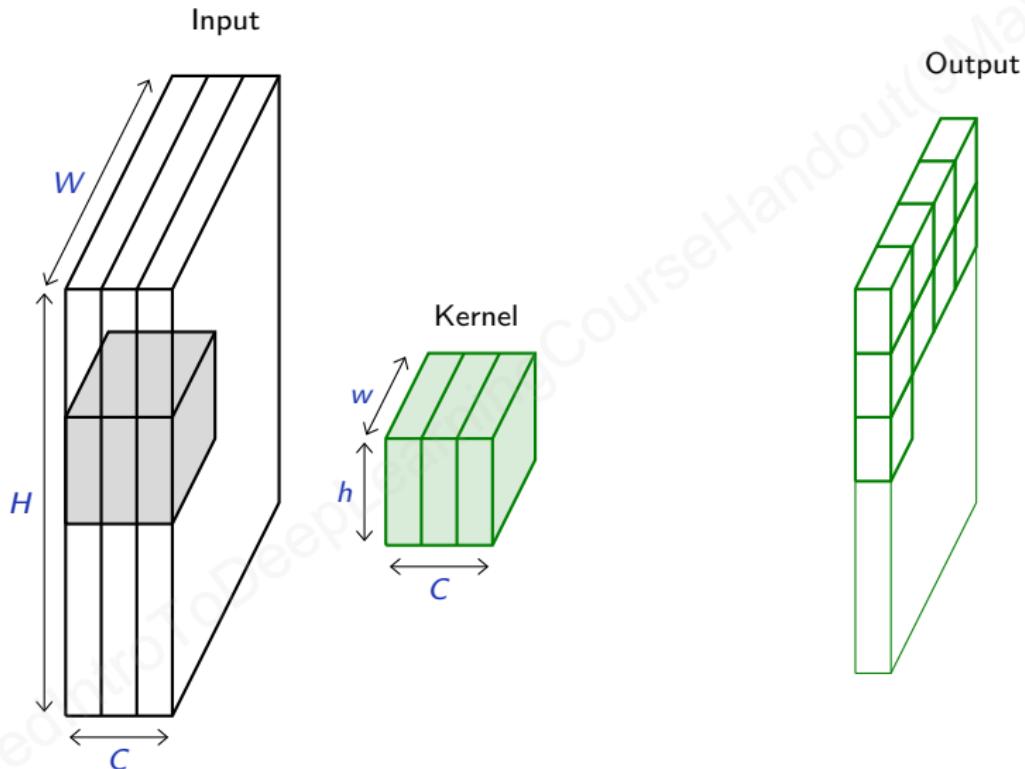
Convolutional layers visualization: 3D inputs



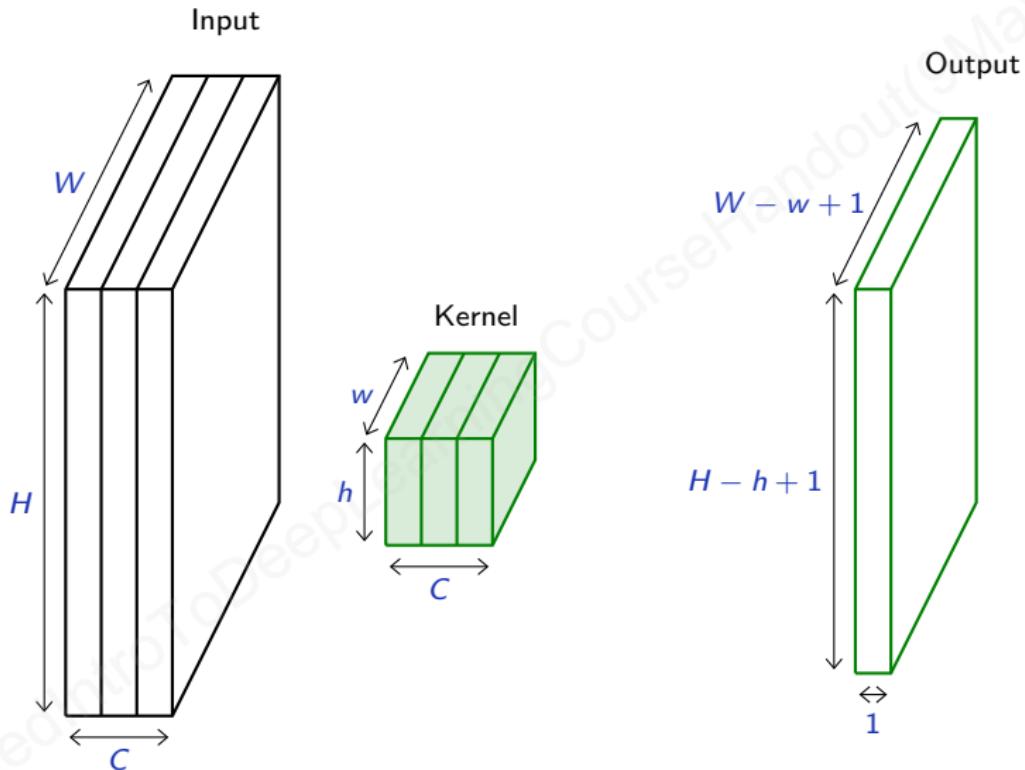
Convolutional layers visualization: 3D inputs



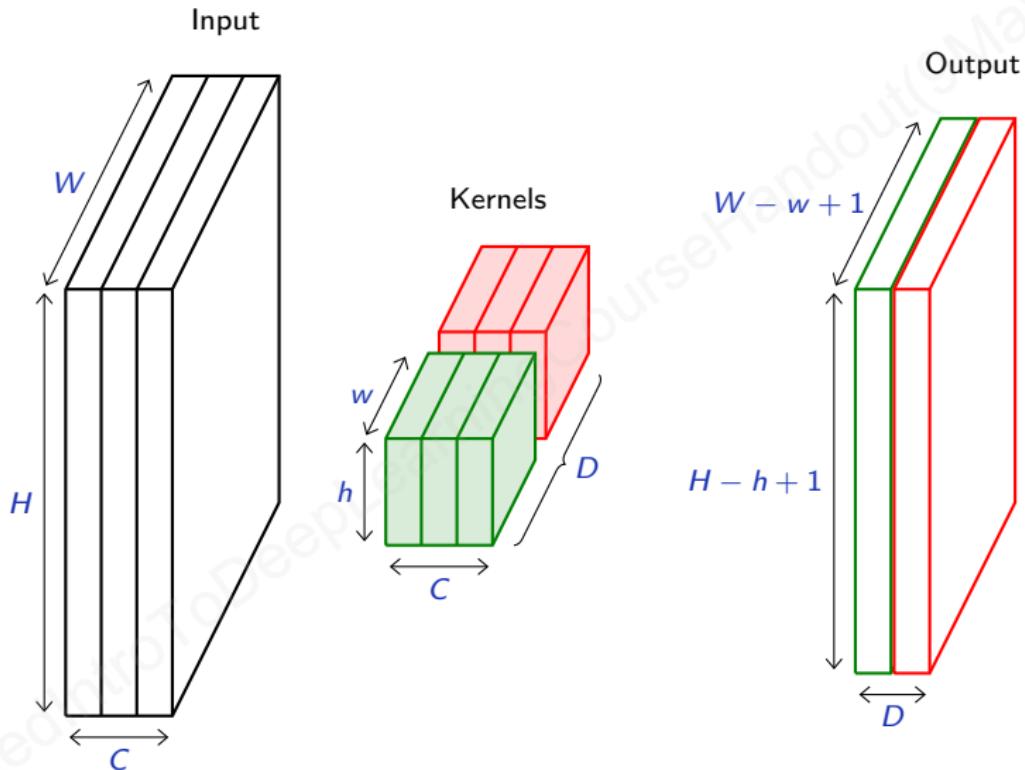
Convolutional layers visualization: 3D inputs



Convolutional layers visualization: 3D inputs

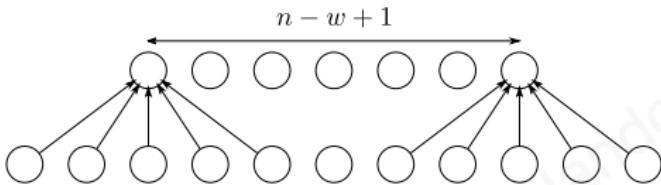


Convolutional layers visualization: 3D inputs

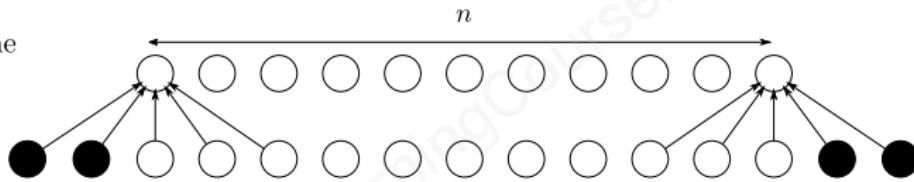


Zero padding

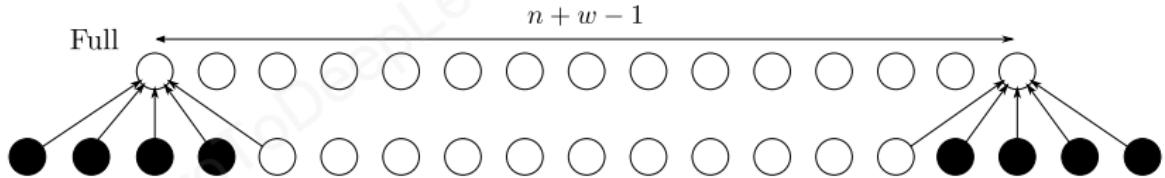
Valid



Same



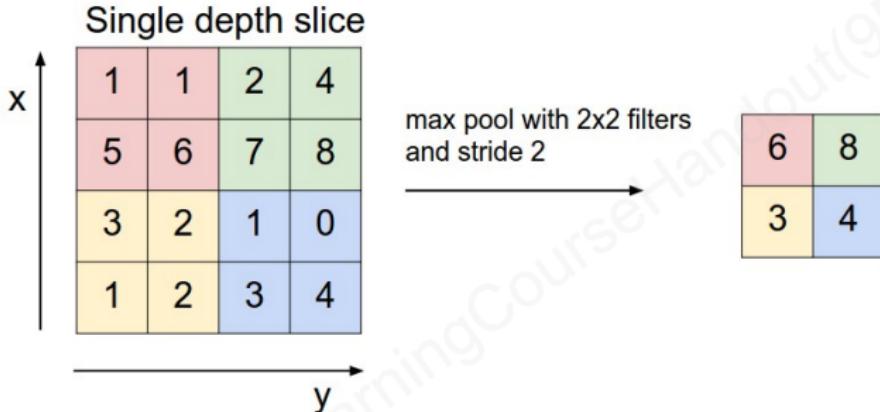
Full



Pooling

- Dense layers often reduce the dimensionality of a signal, i.e., find a low-dimensional representation
- For convolutional layers, it makes sense to reduce the signal's size in a structure-preserving way, i.e., downscaling
- **Pooling** cuts a feature map into $P \times P$ regions for each of which only a single value is retained, e.g., max, average, $L2$ -norm
- Using **stride** during convolutions/pooling operations can further reduce the representation size (next slides)

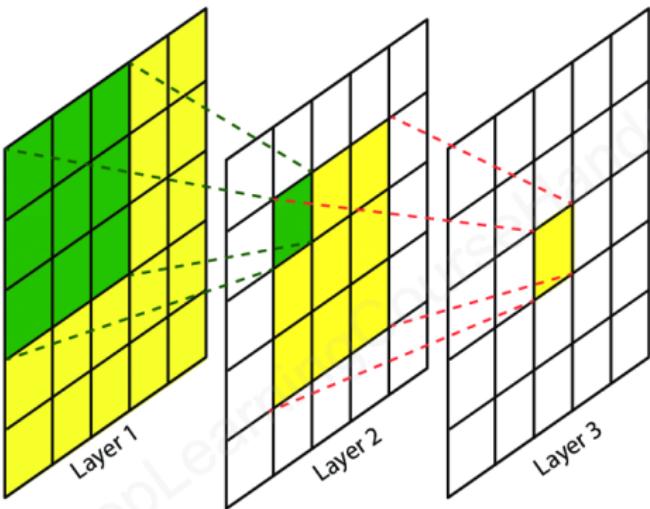
Convolutional layers visualization: Pooling



- Pooling can compensate small input translations
- Important for the detection of features where absolute position within the signal shall not matter

Image taken from <http://cs231n.github.io/convolutional-networks/#pool>

Receptive field



Definition (Receptive field (RF))

Size of the region in the input that produces a single feature value.

<https://distill.pub/2019/computing-receptive-fields/>

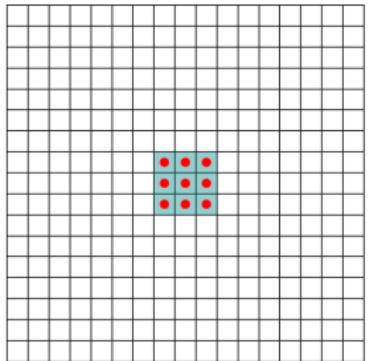
Image taken from <https://www.mdpi.com/197252>

Dilation

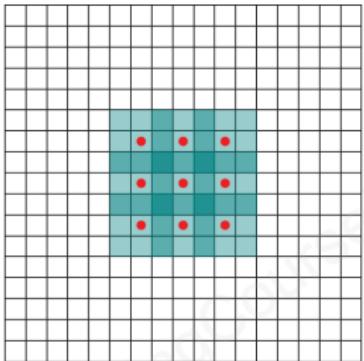
- Increases the kernel size without additional memory or complexity
- Dilation (i, j) inserts $i - 1$ ($j - 1$) 0's along the first (second) dim.

Animation taken from https://github.com/vdumoulin/conv_arithmetic

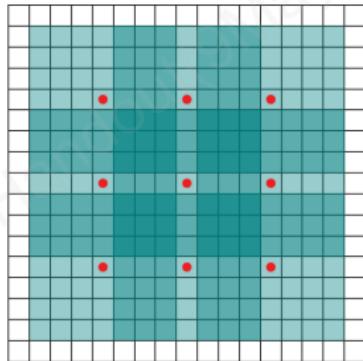
Receptive field of dilated convolutional layers



(1, 1)



(2, 2)



(4, 4)

- By appropriately increasing the dilation across subsequent layers, one can obtain an exponentially increasing receptive field:

$$3 \times 3 \mapsto 7 \times 7 \mapsto 15 \times 15$$

$$3 \times 3 \mapsto 5 \times 5 \mapsto 7 \times 7 \quad (\text{without dilation})$$

- Useful to get large RF with small number of layers and parameters
- Related to the *algorithme à trous* for wavelet decomposition [HKMMT90]

Separable convolutions

- Key idea:
 1. Split kernel into multiple smaller kernels
 2. Apply them sequentially
- Two types:
 - Spatially separable convolutions
 - Depthwise separable convolutions

Spatially separable convolutions

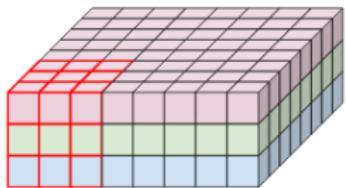
- Many kernels can be written as products of smaller kernels, e.g.,

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

- Consider a 3×3 kernel sliding over an 8×8 input:
 - Full conv.: 324 multiplications, 288 additions, 9 parameters
 - Sep. conv.: 252 multiplications, 120 additions, 6 parameters
 - Pros: Less memory and complexity
 - Cons: Not all kernels can be decomposed
- Spatially separable convolutions hardly used in practice

The example is the Prewitt operator used for edge detection.

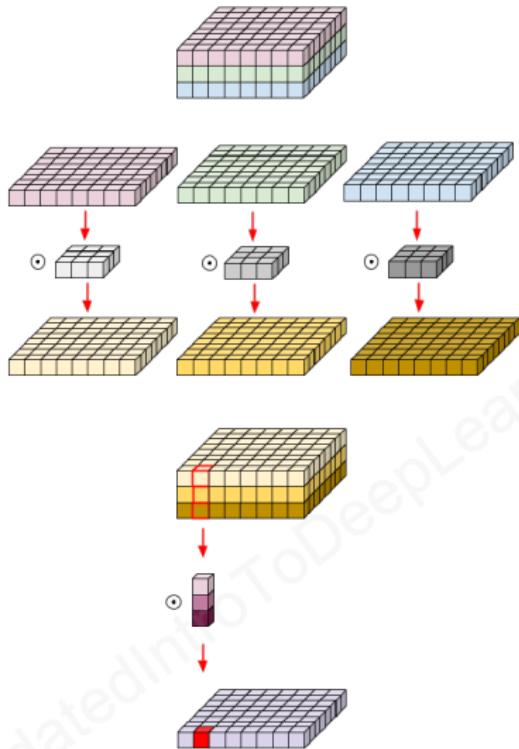
Depthwise separable convolutions (1/3)



Standard convolution:

- Dedicated 3D kernel for each output channel

Depthwise separable convolutions (2/3)



Depthwise separable convolution:

1. Dedicated (spatial) 2D kernel for each input channel
2. Weighted sum (1×1 or pointwise convolution) of the resulting feature maps for each output channel. (i.e., a fully connected layer)

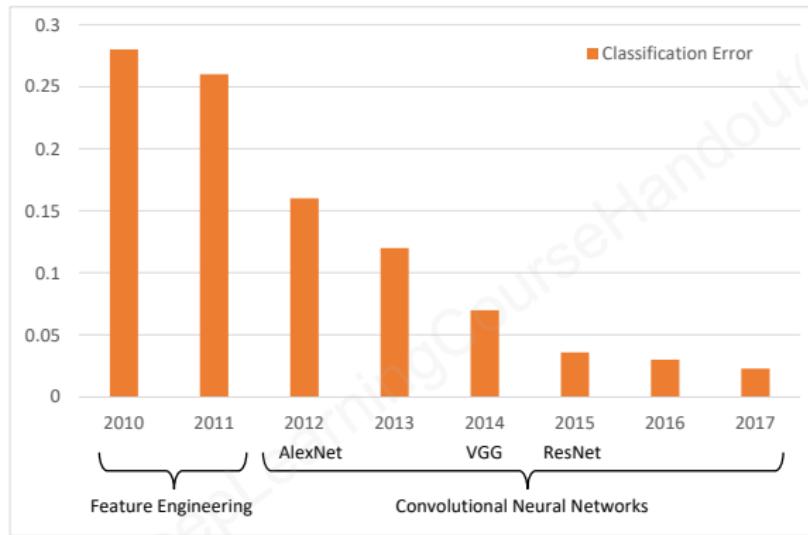
Depthwise separable convolutions (3/3)

Example:

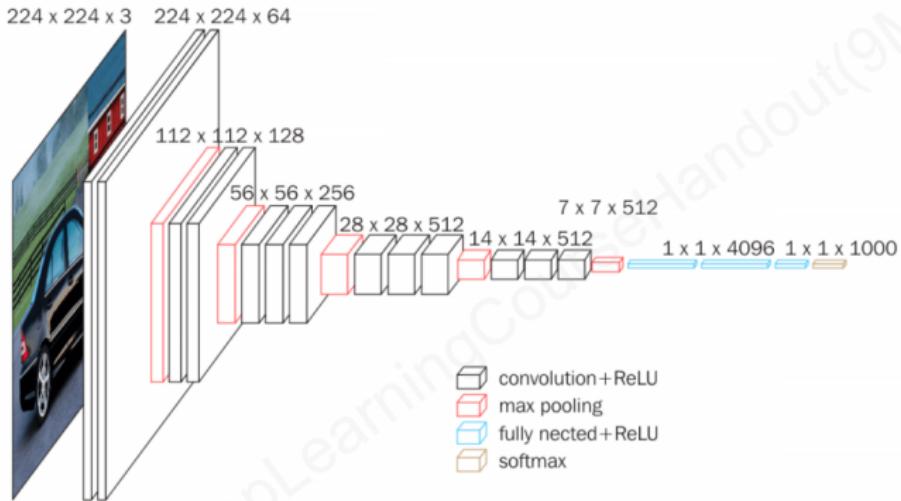
- Input: $8 \times 8 \times 3 \mapsto$ Output: $4 \times 4 \times 32$
- Standard 5×5 convolution:
 - 32 kernels of size $5 \times 5 \times 3$
 - 2400 parameters
 - 38400 MUL & 37888 ADD
- Depthwise separable convolution:
 - 3 kernels of size $5 \times 5 \times 1$ & 32 kernels of size $1 \times 1 \times 3$
 - 171 parameters
 - 1200 MUL & 1152 ADD (Spatial convolution)
 - 1536 MUL & 1024 ADD (Pointwise convolution)
 - $\approx 7\%$ of parameters & complexity of standard convolution

Depthwise separable convolutions drastically reduce the memory footprint and complexity. But: Memory access pattern change which does not always pay off.

How ConvNets revolutionized computer vision

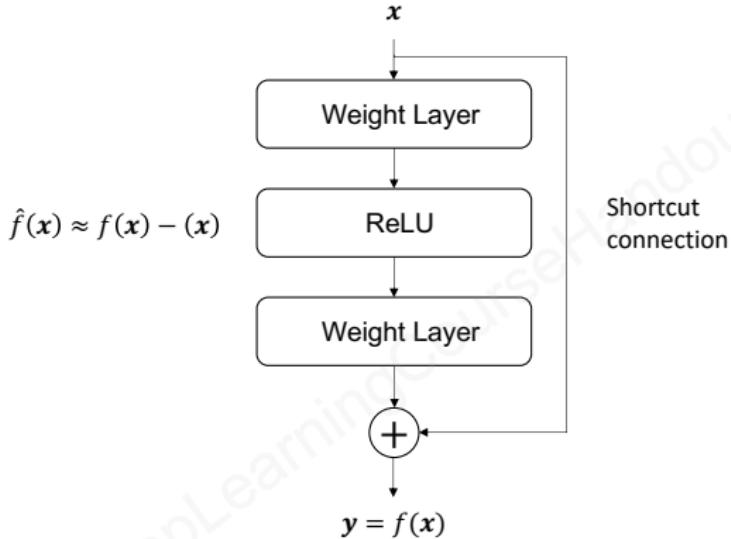


- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)
- 1.2M images taken from the ImageNet dataset
- $10\times$ reduction of classification error between 2010-2017
- Competition dominated by ConvNets since 2012



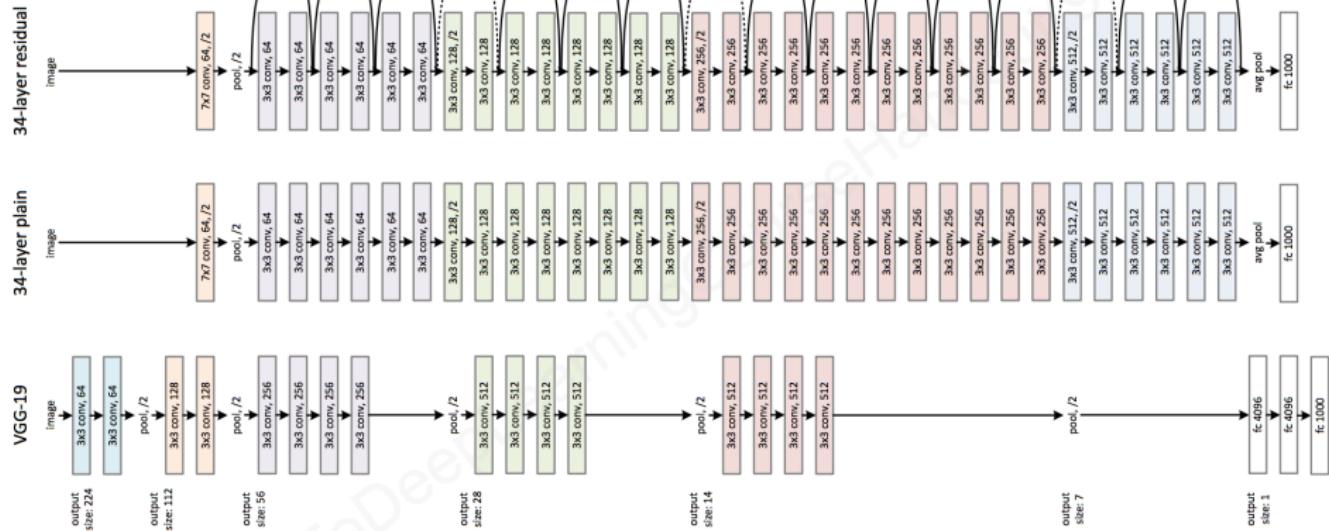
- 2014 entry from Oxford Vision Geometry Group (VGG) [SZ14]
- Demonstrated the power of deep ConvNets
- Widely used due to its simple architecture

Residual connections



- Intuition: Easier to learn residual $f(\mathbf{x}) - \mathbf{x}$ than $f(\mathbf{x})$ directly
- If the identity mapping was optimal, it would be easier to learn it through the shortcut connection than through the weight layers
- *Shortcut, residual, or skip* connections allow for training of very deep architectures as they avoid the *vanishing gradient problem*
- Idea proposed in [HZRS16]

Residual neural networks (ResNets)



Thanks to the ResNet architecture, researchers from Microsoft won five different computer vision competitions in 2015 [HZRS16]

Attention

Attention and Transformers [VSP⁺17]

- Often, input data has sequential structure
 - Multi-path channels
 - Convolutional codes
 - Natural language processing
- How to learn relations within sequences?
- See *nanoGPT* and *makemore* for step-by-step introduction
- See "*Formal Algorithms for Transformers*" [PH22] for extensive summary

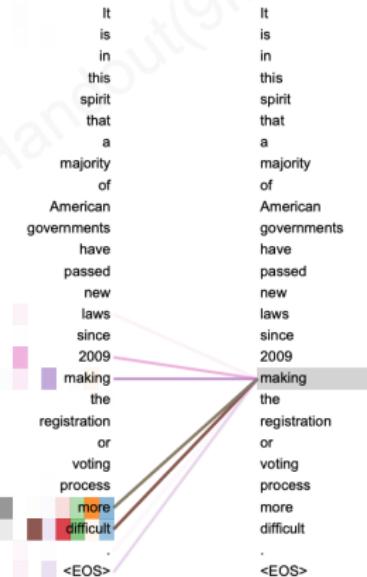


Figure taken from <https://arxiv.org/pdf/1706.03762.pdf>

Attention is all you need [VSP⁺17]

- Dot-products to measure similarity
- Compare query \mathbf{Q} and key \mathbf{K} to obtain scores for value \mathbf{V}

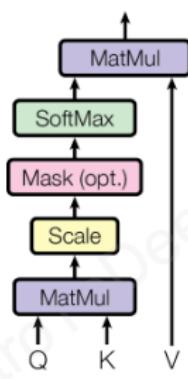
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right) \cdot \mathbf{V}$$

- $\mathbf{Q} \in \mathbb{R}^{N \times d}$, $\mathbf{K} \in \mathbb{R}^{N \times d}$, and $\mathbf{V} \in \mathbb{R}^{N \times k}$
- Scalar product $\mathbf{Q}\mathbf{K}^T$ to measure similarity between \mathbf{Q} and \mathbf{K}
- Softmax yields convex combination of features, i.e. the non-negative weights sum up to one
- Scaling with \sqrt{d} to avoid that dot products grow large and softmax gradient vanishes
- Different kinds:
 - Self-attention: query, key and value are linear transformations of the same input
 - Encoder-decoder attention: query comes from decoder, key and value from encoder

Multi-head attention

- Core idea: multiple layers in parallel, concatenate outputs
- N heads of size D/N are often better than one big head of size D
 - They can learn independent functions [MLN19]

Scaled Dot-Product Attention



Multi-Head Attention

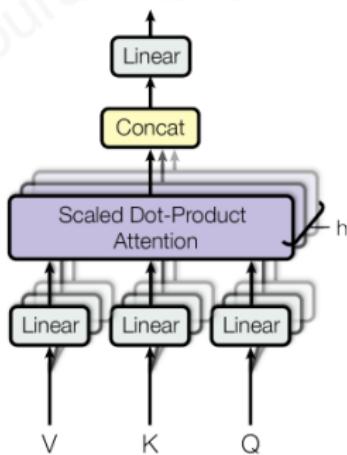


Figure taken from <https://arxiv.org/pdf/1706.03762.pdf>

Attention & non-linearity

- Dot-product attention is a linear operation
- Transformer adds non-linearity by a feedforward NN (MLP)
- Often, realized with additional skip connection & normalization

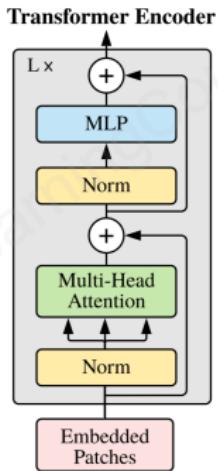


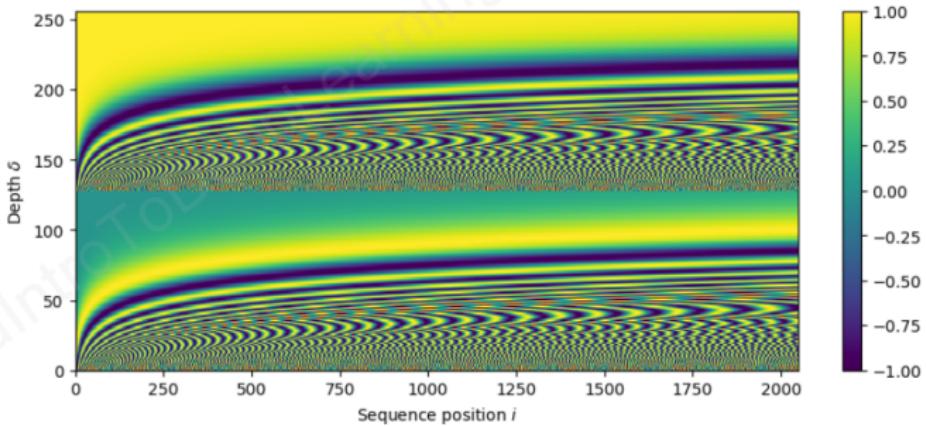
Figure taken from <https://arxiv.org/pdf/2010.11929.pdf>

Positional encoding

- Traditional RNNs process input sequentially (implicit temporal info.)
- Attention is permutation-invariant (no positional information)
- Positional encoding: offset based on position i within sequence

$$PE(i, \delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

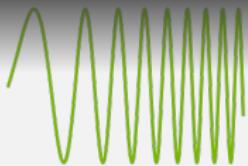
where δ denotes the dimension and d the total number of features



Introduction to Sionna

WIDE RANGE OF 6G RESEARCH TOPICS

TERAHERTZ



RECONFIGURABLE INTELLIGENT SURFACES



JOINT SENSING & COMMUNICATIONS



AI-NATIVE AIR INTERFACE



SEMANTIC COMMUNICATIONS



CELL-FREE AND HOLOGRAPHIC MIMO



6G RESEARCH NEEDS NEXT-GENERATION TOOLS

ENVIRONMENT SPECIFIC DATA

- Need for simulation of specific environments
- Physically consistent multimodal data



NATIVE INTEGRATION OF AI/ML

- Integration of link-level simulation and ML capabilities
- Automatic gradient computation from end-to-end



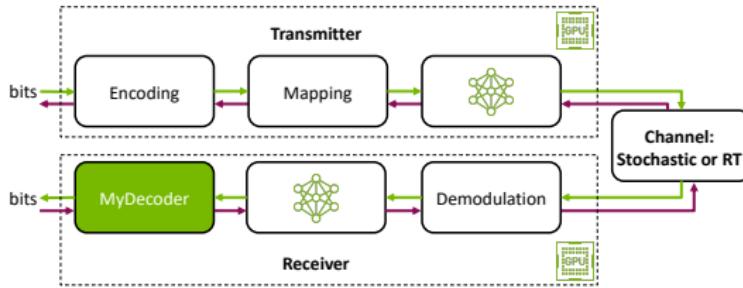
VERY HIGH DETAIL AND SCALE

- Unprecedented modelling accuracy and scale
- Very large bandwidth, number of devices and antennas



SIONNA

A GPU-Accelerated Framework for Fully-Differentiable Link-Level Simulations



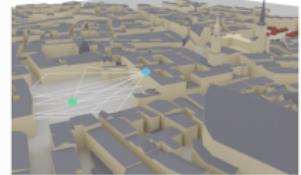
- Rapid prototyping of complex communication systems
- Native support of neural networks
- Differentiable from end-to-end
- Superfast and scalable to multi-GPU setups
- Free to use and open source
- Extensively documented



```
(14): # Compute propagation paths
paths = sionna.compute_paths(max_nbs,
                           max_nb_angles=1000, # For each corner the method can be able to compute up to 1000 paths
                           max_nb_angles_per_nb=100, # Number of rays over they receive direct
                           max_nb_angles_per_nb=400) # By fixing the seed, reproducible result
```

Visualize paths in the 3D preview

```
# If you want to see the paths in a 3D viewer
sionna.render("my_path", paths=paths, show_devices=True, show_paths=True, result_device="value", toolkits="pyvista", show_devices=True, show_paths=True)
```



Remark: Only one preview instance can be opened at the same time. Please check the previous preview if no output appears.

The `Paths` object contains all paths that have been found between transmitters and receivers. In principle, the existence of each path is determinable for a given position and environment. Please note that due to the stochastic nature of the sheet-and-house algorithm, different runs of the `compute_paths` function lead to different paths that are found. For reproducible results, it is recommended to fix the `seed`:

```
(1): # We can access for every pair the resulting transfer matrices, the propagation delay,
# as well as the angle of departure and arrival, respectively (points and angles):
ext_x, int_x, (theta_x, phi_x), (theta_x, phi_x) = paths[0].tuple()

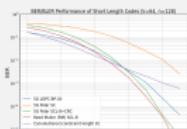
print("Shape of net_t:", net_t.shape)
print("Shape of net_i:", net_i.shape)

# The dimensions are batch_size, num_rx, num_tx, max_nb_paths, 2, 2 where the transfer is
# printed. --- Detailed results for pair (path_idx, ---) ---:
print("Path delay: ", paths[0].delay)
print("Path propagation delay: ", paths[0].path_delay[0])
print("Path angle of departure: ", paths[0].angle_of_departure[0])
print("Path angle of arrival: ", paths[0].angle_of_arrival[0])
print("Path angle of arrival: ", paths[0].angle_of_arrival[0])
```

KEY FEATURES

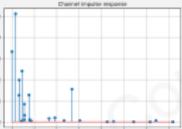
FORWARD ERROR CORRECTION

- All 2G-5G coding schemes
- Many baseline decoders



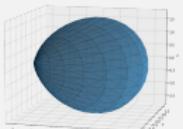
CHANNEL MODELS

- 3GPP 38.901
- Optical Fibers



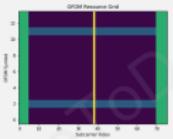
MULTIUSER MIMO

- Precoders & Detectors
- Multicell support



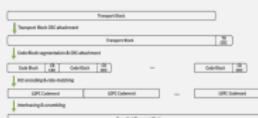
OFDM

- Flexible 5G-like frame structure
- Various channel estimators



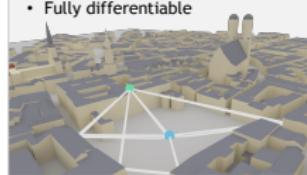
5G NR

- Full PUSCH support
- Transport block segmentation



RAY TRACING

- Runs in a Jupyter notebook
- Fully differentiable



DESIGN PRINCIPLES

Custom TF ops in C++, CUDA, or XLA

- If composition of existing ops is not enough

TF Eager and Graph Execution

- XLA (experimental)

Default data type `tf.complex64`

- `tf.complex128` available for high precision

All algorithms use high-dimensional tensors:

- 1st dimension is batch_size, e.g.:
`[batch_size, num_tx, ..., fft_size]`
- Efficient parallelization of workloads

Every component is a TensorFlow (TF) Keras Layer:

- Simply connect desired layers
- Easily replace components by neural networks
- Automatic gradient computation

HELLO, SIONNA!

- Instantiate and connect layers with the Keras functional API
- Replace components by neural networks or make them trainable
- Automatic gradient computation from end-to-end for a given loss

```
# Coded BER over AWGN Channel Simulation

batch_size = 1024
n = 1000 # codeword length
k = 500 # information bits per codeword
m = 4 # bits per symbol
snr = 10

c = Constellation("qam", m, trainable=True)
b = BinarySource()([batch_size, k])
u = LDPC5GEncoder(k, n)(b)
x = Mapper(constellation=c)(u)
y = AWGN()([x, 1/snr])
llr = NeuralDemapper()(y, 1/snr)
b_hat = LDPC5GDecoder(LDPC5GEncoder(k, n))(llr)
```

CREATING DIGITAL TWINS HAS NEVER BEEN EASIER

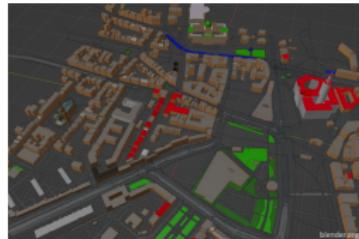
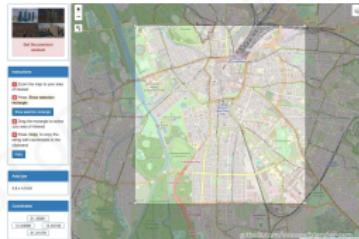
Lidar



Images (NeRFs)



OpenStreetMap/Blender



Digital Twins for Radio Propagation

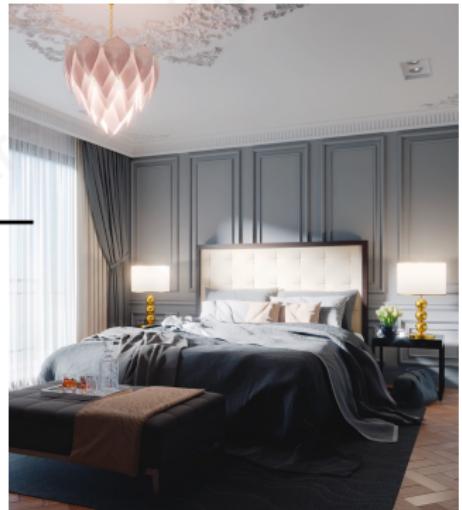


- Scene geometries with cm-accuracy
- What about material properties?

Inverse Rendering



Geometry, materials, light sources, cameras, ...

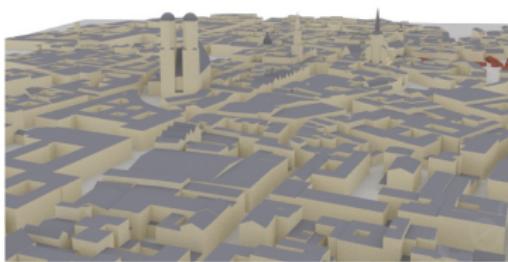


Slide by Merlin Nimier-David, Scene: "bed classic" from jiraniano

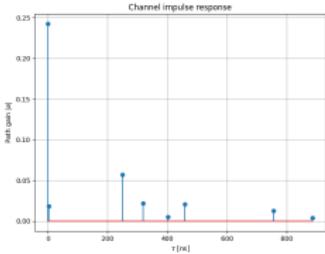


Can this be done for radio propagation modeling?

Inverse Radio Wave Propagation

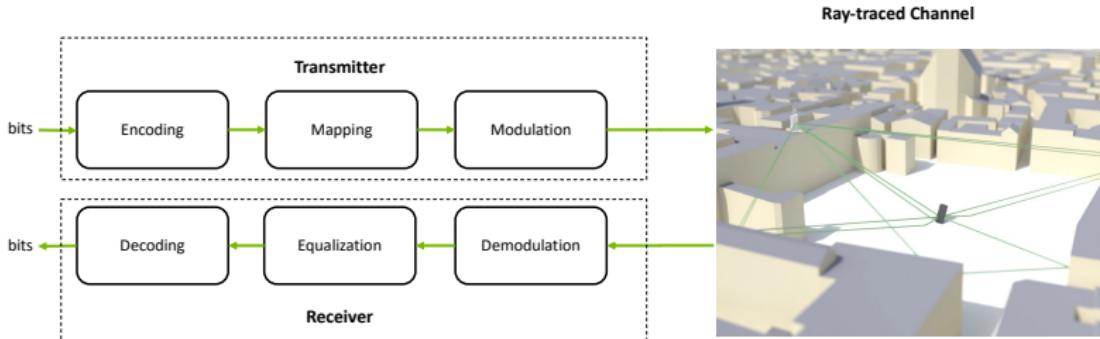


$$x = f^{-1}(y)?$$



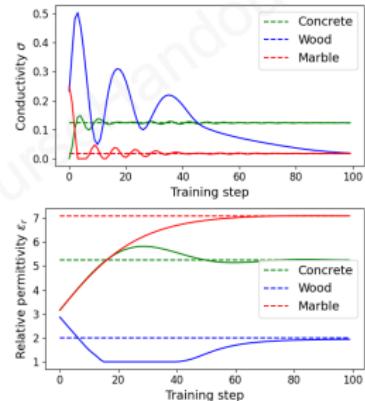
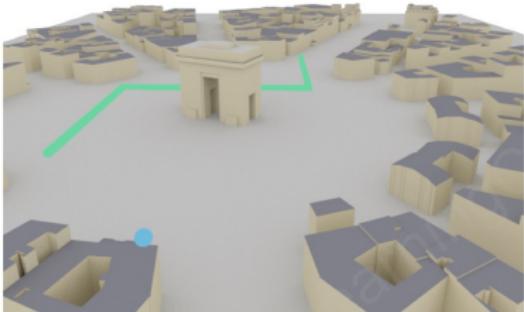
Materials, antenna arrays/patterns, RX/TX locations, ...

SIONNA RT: DIFFERENTIABLE RAY TRACING FOR PROPAGATION MODELING



- Ray-traced channels in-lieu of stochastic channel models
- Differentiability enables entirely new research directions:
 - Learning of material properties
 - Calibration to measurement results for digital twins
 - Gradient-based optimization of system parameters (antenna patterns, array geometries, RIS configuration, etc.)
 - Straight-forward integration of neural networks (e.g., learned scattering functions)

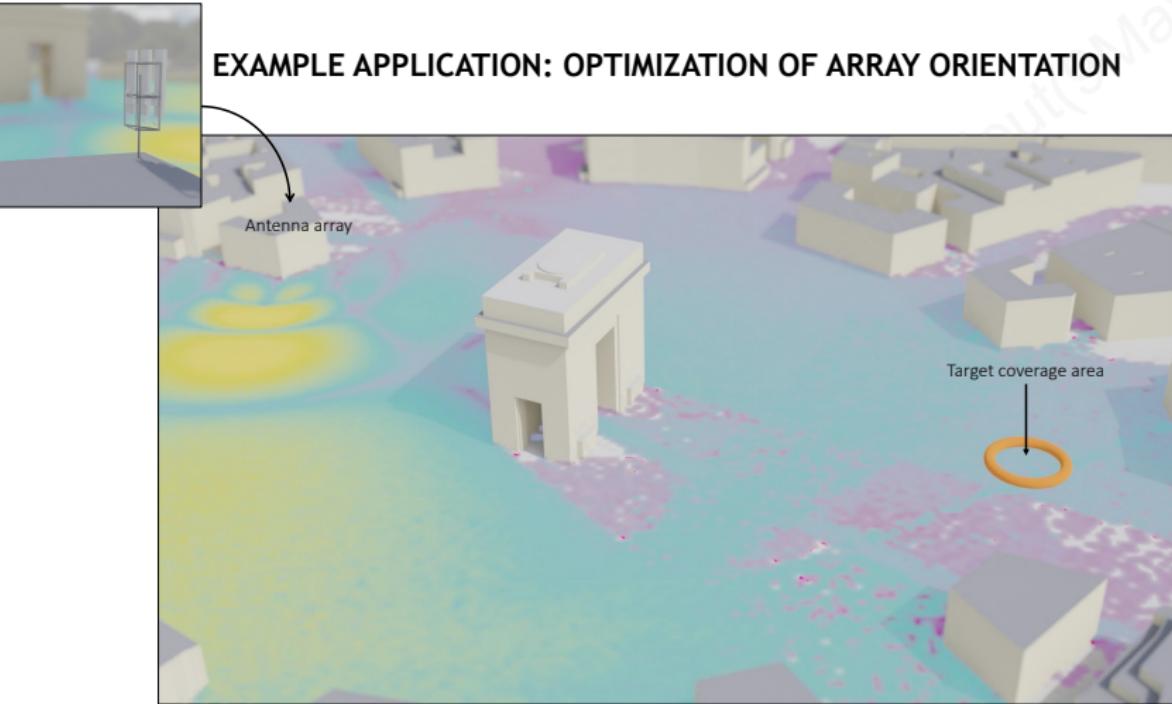
Example Application: Learning of Radio Materials

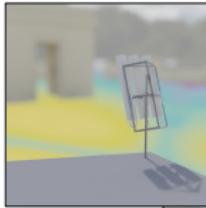


- Learn relative permittivity and conductivity by gradient descent
- MSE loss between channel frequency responses of “true” and “trainable” materials

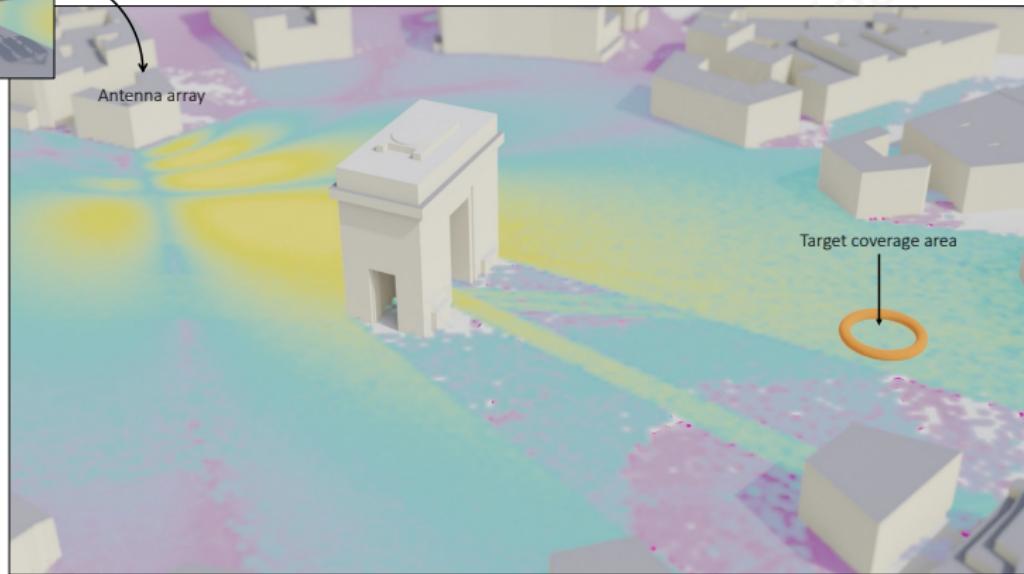
What to do with a digital twin?

EXAMPLE APPLICATION: OPTIMIZATION OF ARRAY ORIENTATION





EXAMPLE APPLICATION: OPTIMIZATION OF ARRAY ORIENTATION



Sionna Live Demo



Live Demo

Part II: Applications of Deep Learning for Communications

DL at the Receiver

Multi-tap channel

Consider an L -tap channel

$$y_n = \sum_{\ell=0}^{L-1} h_{n,\ell} x_{n-\ell} + w_n$$

where the channel input is

$$\mathbf{x} = [x_0, \dots, x_{N-1}]^T \in \mathbb{C}^N$$

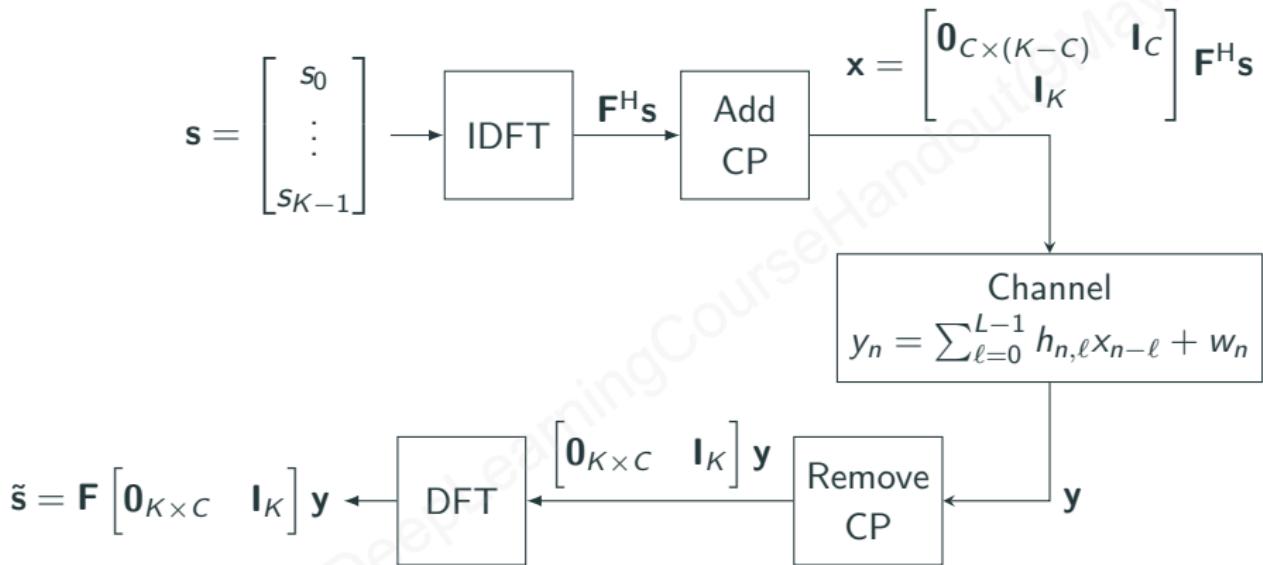
the channel output is

$$\mathbf{y} = [y_0, \dots, y_{N-1}]^T \in \mathbb{C}^N$$

and the additive noise is assumed to be white Gaussian

$$\mathbf{w} = [w_0, \dots, w_{N-1}]^T \sim \mathcal{CN}(0, \sigma^2 \mathbf{I}_N).$$

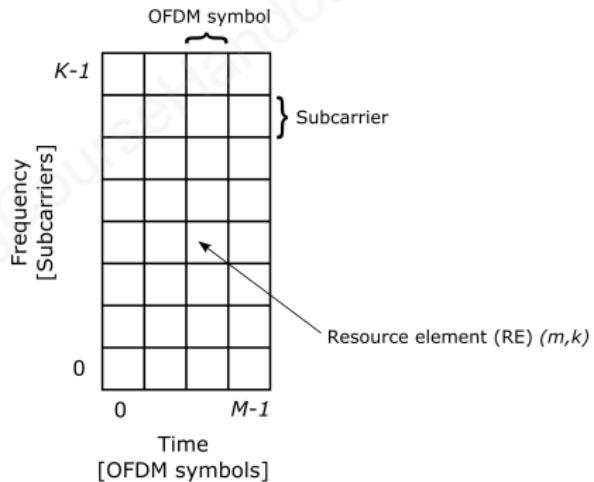
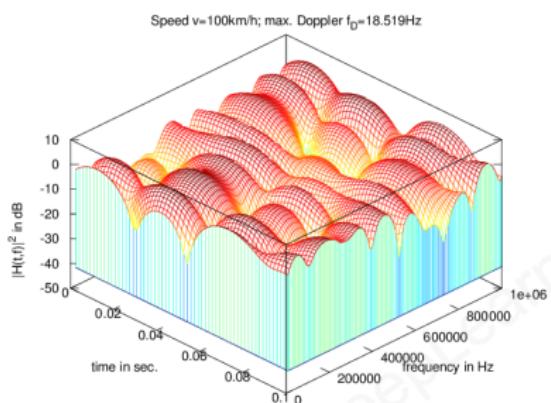
OFDM architecture



- \mathbf{F} is the discrete Fourier transform (DFT) matrix
- C is the cyclic prefix (CP) length
- $N = K + C$ is the length of an OFDM symbol

OFDM resource grid

In practice, multiple OFDM symbols are arranged and transmitted as a *resource grid*



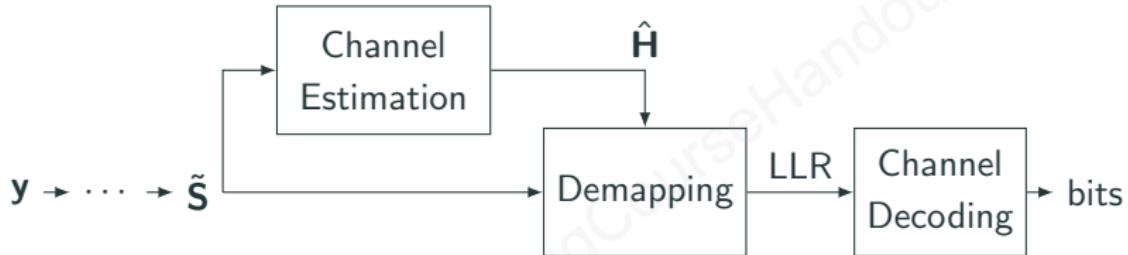
Left figure is from University of Stuttgart webdemo

OFDM detection

OFDM *detection* consists in reconstructing the transmitted bits from the received post-DFT signal $\tilde{\mathbf{S}} \in \mathbb{C}^{K \times M}$

OFDM detection

OFDM *detection* consists in reconstructing the transmitted bits from the received post-DFT signal $\tilde{\mathbf{S}} \in \mathbb{C}^{K \times M}$

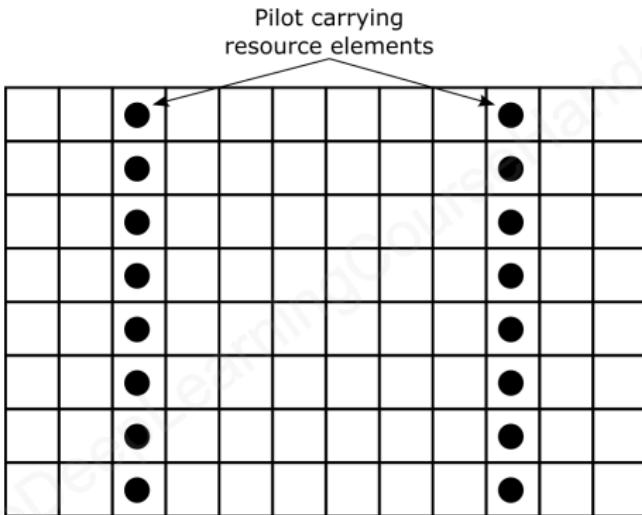


where

- $\hat{\mathbf{H}} \in \mathbb{C}^{K \times M}$ is the channel estimate
- $\text{LLR} \in \mathbb{R}^B$ is the tensor of log-likelihood ratios (LLRs)
- B is the number of bits per resource grid

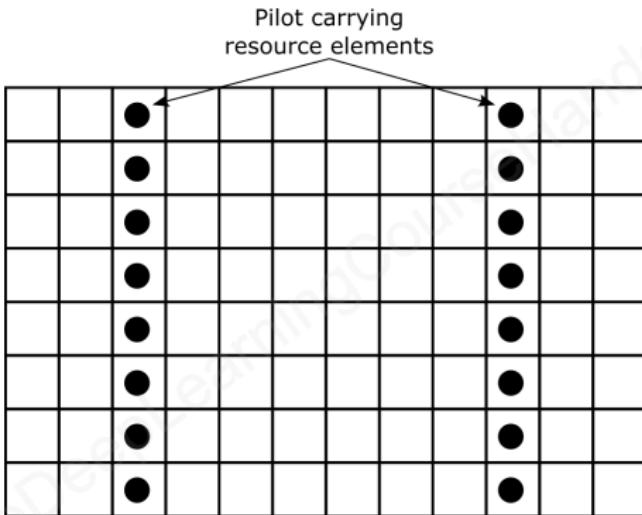
OFDM detection – Channel estimation

Channel estimation is achieved through the transmission of *pilot signals*



OFDM detection – Channel estimation

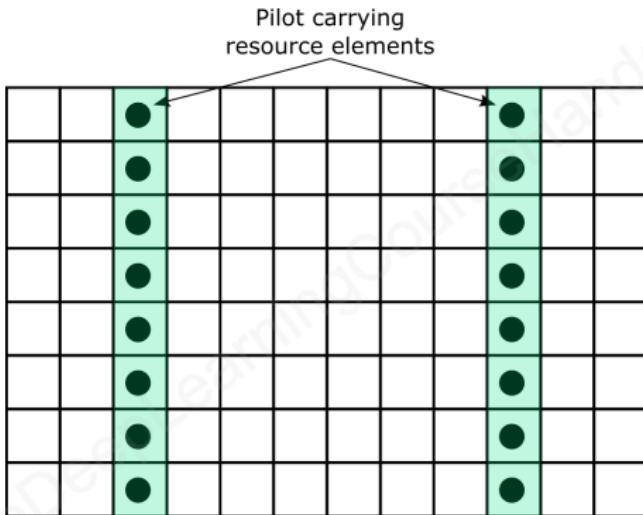
Channel estimation is achieved through the transmission of *pilot signals*



Channel estimation typically consists in two steps:

OFDM detection – Channel estimation

Channel estimation is achieved through the transmission of *pilot signals*

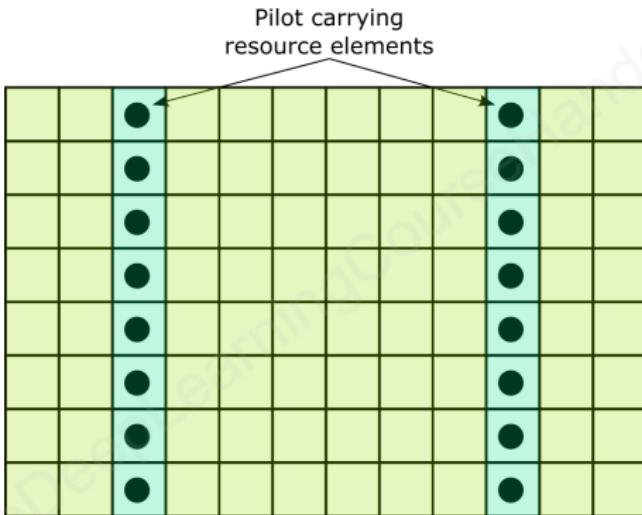


Channel estimation typically consists in two steps:

1. Estimate the channel for pilot-carrying REs

OFDM detection – Channel estimation

Channel estimation is achieved through the transmission of *pilot signals*



Channel estimation typically consists in two steps:

1. Estimate the channel for pilot-carrying REs
2. Extrapolate to estimate the channel for data-carrying REs

OFDM – Practical issues

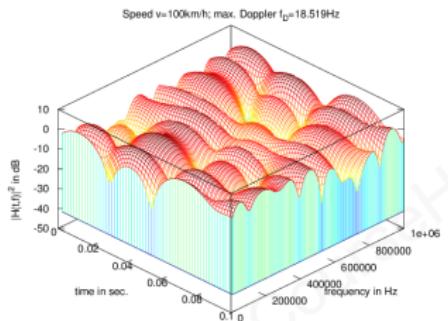
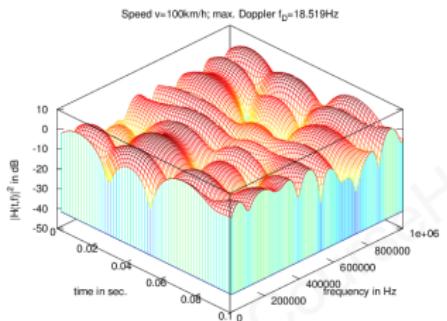


Figure is from University of Stuttgart webdemo

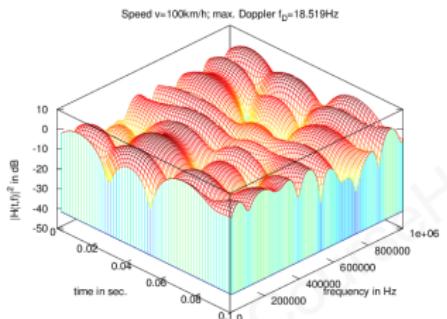
OFDM – Practical issues



- Channel statistics are not known in practice

Figure is from University of Stuttgart webdemo

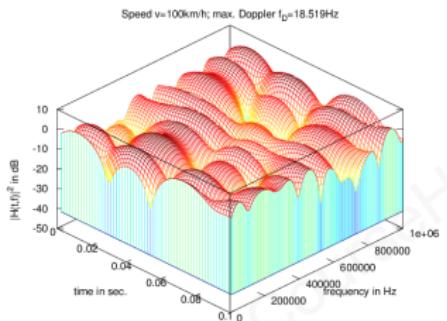
OFDM – Practical issues



- Channel statistics are not known in practice
- Channel aging → simple estimation methods perform poorly

Figure is from University of Stuttgart webdemo

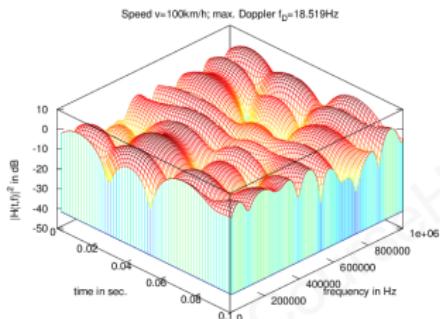
OFDM – Practical issues



- Channel statistics are not known in practice
- Channel aging → simple estimation methods perform poorly
- Channel *does* age over the duration of OFDM symbols → intercarrier interference

Figure is from University of Stuttgart webdemo

OFDM – Practical issues



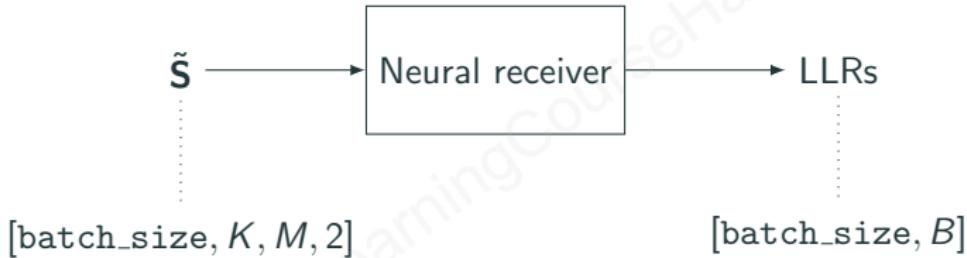
- Channel statistics are not known in practice
- Channel aging → simple estimation methods perform poorly
- Channel *does* age over the duration of OFDM symbols → intercarrier interference
- Hardware impairments: carrier frequency offset, nonlinearities, . . .

Figure is from University of Stuttgart webdemo

Neural receivers – Overview

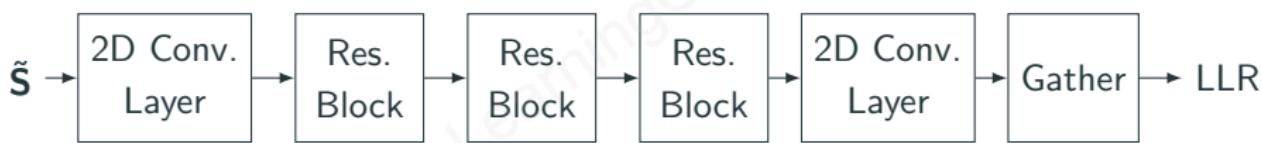


Neural receivers – Overview



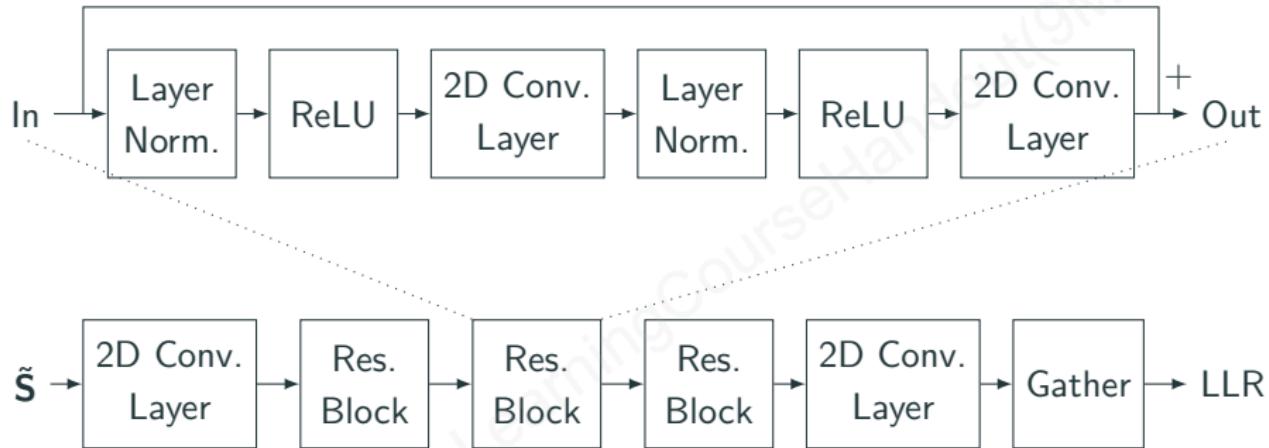
- K : Number of subcarriers
- M : Number of OFDM symbols
- B : Number of bits per resource grid

Neural receivers – Residual networks



Residual networks are typically built from *residual blocks*

Neural receivers – Residual networks



Residual networks are typically built from *residual blocks*

Layer normalization [BHM⁺16]

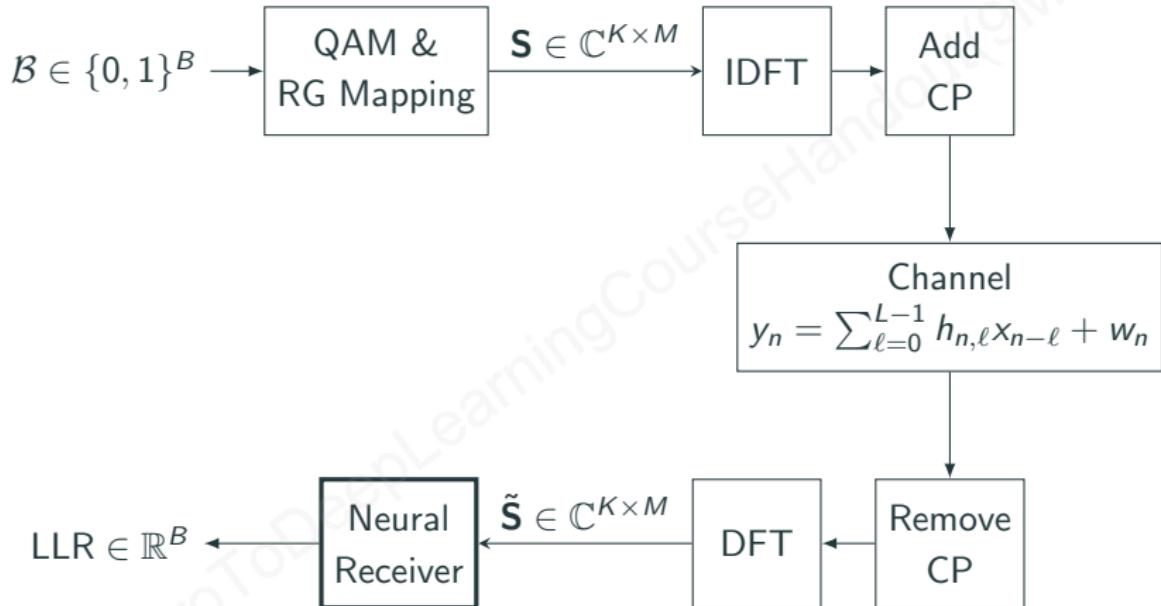
- Key idea: Normalize such that the output mean is close to zero and the output standard deviation is close to one
- The activations are normalized for each example *independently*, and *not* across a batch like with batch normalization
- For each example \mathbf{x}_n of a batch of inputs $[\mathbf{x}_1 \dots \mathbf{x}_N]$:

$$\mu_n = \text{mean}(\mathbf{x}_n), \quad \hat{\sigma}_n^2 = \text{var}(\mathbf{x}_n)$$

$$\mathbf{y}_n = \gamma_n \frac{\mathbf{x}_n - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}} + \beta_n$$

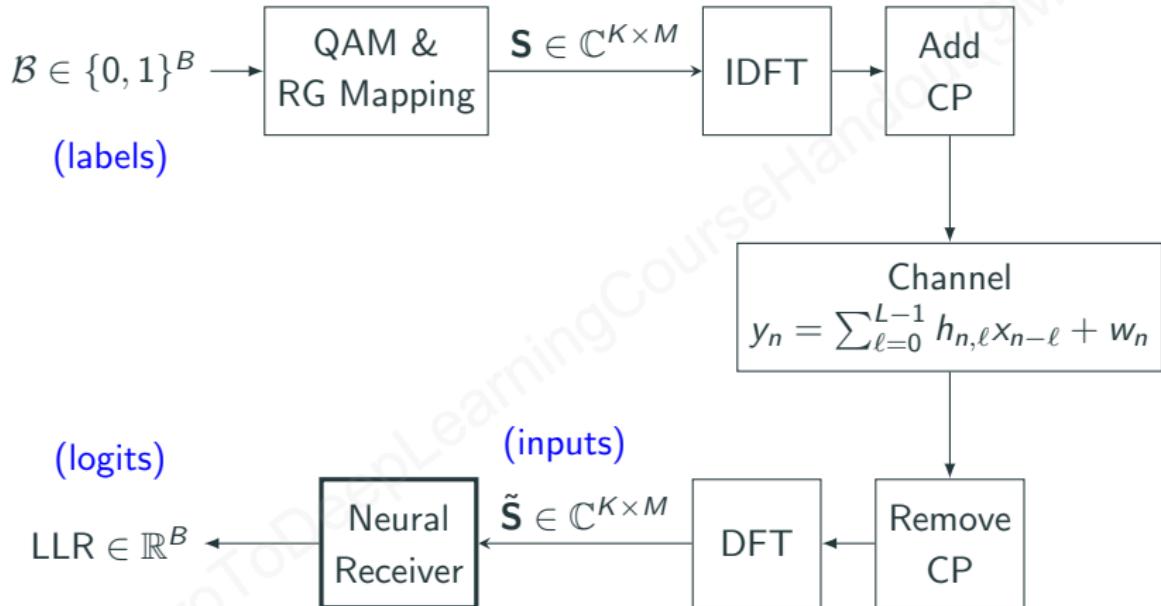
- \mathbf{y}_n is the n th normalized output of the batch
- γ_n and β_n are trainable variables and ϵ is a small constant

OFDM architecture with neural receiver



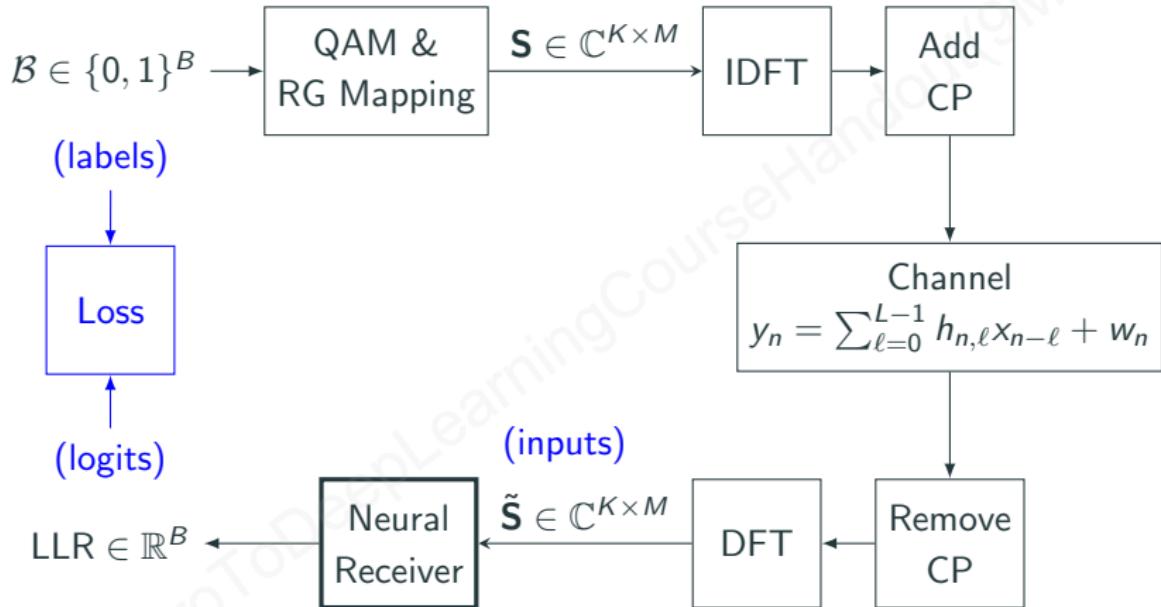
- K : Number of subcarriers
- M : Number of OFDM symbols
- B : Number of bits per symbol resource grid

OFDM architecture with neural receiver



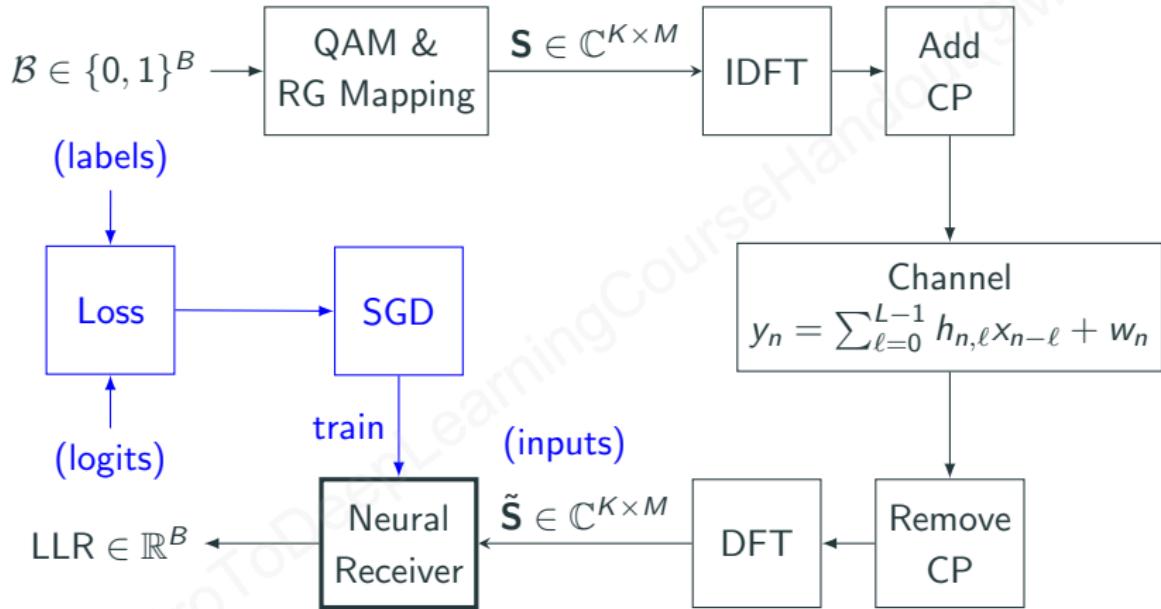
- K : Number of subcarriers
- M : Number of OFDM symbols
- B : Number of bits per symbol resource grid

OFDM architecture with neural receiver



- K : Number of subcarriers
- M : Number of OFDM symbols
- B : Number of bits per symbol resource grid

OFDM architecture with neural receiver



- K : Number of subcarriers
- M : Number of OFDM symbols
- B : Number of bits per symbol resource grid

Loss function

The LLRs computed by the neural receiver are such that

$$\text{LLR}_b = \log \left(\frac{Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})}{1 - Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})} \right)$$

where

$$Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})$$

is the probability computed by the neural receiver with trainable parameters θ of the b^{th} bit of the resource grid to equal 1 given $\tilde{\mathbf{S}}$

Loss function

The LLRs computed by the neural receiver are such that

$$\text{LLR}_b = \log \left(\frac{Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})}{1 - Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})} \right)$$

where

$$Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}})$$

is the probability computed by the neural receiver with trainable parameters θ of the b^{th} bit of the resource grid to equal 1 given $\tilde{\mathbf{S}}$

One has

$$\begin{aligned} Q_{\theta}(\mathcal{B}_b = 1 | \tilde{\mathbf{S}}) &= \frac{e^{\text{LLR}_b}}{1 + e^{\text{LLR}_b}} \\ &= \text{sigmoid}(\text{LLR}_b) \end{aligned}$$

Loss function

The receiver aims to jointly solve B binary classification problems

Loss function

The receiver aims to jointly solve B binary classification problems

→ Binary cross-entropy (BCE) is a good fit

$$\ell(\mathcal{B}_b, \text{LLR}_b) = -\mathcal{B}_b \log \left(Q_{\theta} \left(\mathcal{B}_b = 1 \mid \tilde{\mathbf{S}}^{(u)} \right) \right) \\ - (1 - \mathcal{B}_b) \log \left(1 - Q_{\theta} \left(\mathcal{B}_b = 1 \mid \tilde{\mathbf{S}}^{(u)} \right) \right)$$

Loss function

The receiver aims to jointly solve B binary classification problems

→ Binary cross-entropy (BCE) is a good fit

$$\ell(\mathcal{B}_b, \text{LLR}_b) = -\mathcal{B}_b \log \left(Q_{\theta} \left(\mathcal{B}_b = 1 \mid \tilde{\mathbf{S}}^{(u)} \right) \right) \\ - (1 - \mathcal{B}_b) \log \left(1 - Q_{\theta} \left(\mathcal{B}_b = 1 \mid \tilde{\mathbf{S}}^{(u)} \right) \right)$$

The neural receiver is trained on the average BCE

$$\mathcal{L} = \frac{1}{UB} \sum_{u=0}^{U-1} \sum_{b=0}^{B-1} \ell \left(\mathcal{B}_b^{(u)}, \text{LLR}_b^{(u)} \right)$$

where U is the batch size

Loss function

Assuming that the examples in a batch are i.i.d.,

$$\lim_{U \rightarrow \infty} \mathcal{L} = \bar{\mathcal{L}}$$

where

$$\bar{\mathcal{L}} = \frac{1}{B} \sum_{b=0}^{B-1} \mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]$$

from the law of large numbers, and where the expectation is over the coded bits \mathcal{B} and channel output $\tilde{\mathbf{S}}$

Loss function

Assuming that the examples in a batch are i.i.d.,

$$\lim_{U \rightarrow \infty} \mathcal{L} = \bar{\mathcal{L}}$$

where

$$\bar{\mathcal{L}} = \frac{1}{B} \sum_{b=0}^{B-1} \mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]$$

from the law of large numbers, and where the expectation is over the coded bits \mathcal{B} and channel output $\tilde{\mathbf{S}}$

When computing \mathcal{L} , one computes a Monte Carlo estimate of $\bar{\mathcal{L}}$ by sampling the coded bits \mathcal{B} and corresponding received symbols $\tilde{\mathbf{S}}$

Loss function

$$\begin{aligned}\mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right] &= H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \\ &\quad + \mathbb{E}_{\tilde{\mathbf{S}}} \left[D_{\text{KL}} \left(\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \parallel Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]\end{aligned}$$

where

- $H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ is the entropy of \mathcal{B}_b conditioned on $\tilde{\mathbf{S}}$
- D_{KL} is the Kullback–Leibler (KL) divergence

$$D_{\text{KL}}(P(X) \parallel Q(X)) := \mathbb{E} \left[\log \frac{P(X)}{Q(X)} \right]$$

- $\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ the *true* posterior probability on \mathcal{B}_b given $\tilde{\mathbf{S}}$

Loss function

$$\begin{aligned}\mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{s}} \right) \right) \right] &= H \left(\mathcal{B}_b \mid \tilde{\mathbf{s}} \right) \\ &+ \mathbb{E}_{\tilde{\mathbf{s}}} \left[D_{KL} \left(\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{s}} \right) \parallel Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{s}} \right) \right) \right]\end{aligned}$$

Loss function

$$\begin{aligned}\mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right] &= H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \\ &\quad + \mathbb{E}_{\tilde{\mathbf{S}}} \left[D_{KL} \left(\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \parallel Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]\end{aligned}$$

One can observe that

- $H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ does not depend on the receiver (only on the transmitter and channel)

Loss function

$$\begin{aligned}\mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right] &= H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \\ &\quad + \mathbb{E}_{\tilde{\mathbf{S}}} \left[D_{KL} \left(\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \parallel Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]\end{aligned}$$

One can observe that

- $H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ does not depend on the receiver (only on the transmitter and channel)
- $\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ is the probability that an *optimal* receiver would provide

Loss function

$$\begin{aligned}\mathbb{E} \left[-\log \left(Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right] &= H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \\ &\quad + \mathbb{E}_{\tilde{\mathbf{S}}} \left[D_{KL} \left(\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \parallel Q_{\theta} \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right) \right) \right]\end{aligned}$$

One can observe that

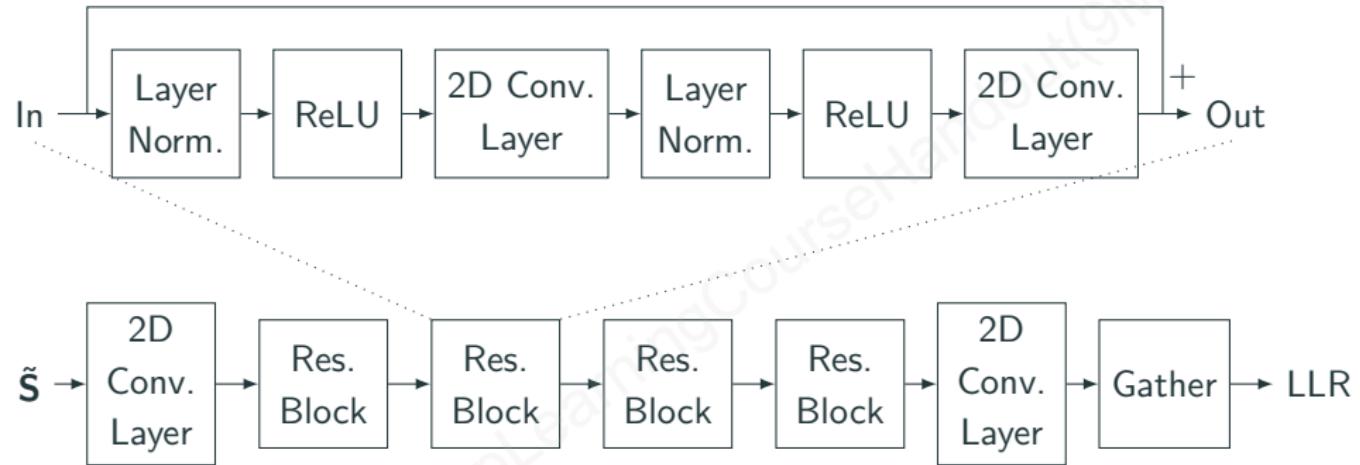
- $H \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ does not depend on the receiver (only on the transmitter and channel)
- $\Pr \left(\mathcal{B}_b \mid \tilde{\mathbf{S}} \right)$ is the probability that an *optimal* receiver would provide

By minimizing the loss \mathcal{L} , one aims to minimize the KL divergence, which can be viewed as a distance, between the neural receiver and an optimal receiver that would compute the true posterior distribution on the coded bits given the channel output

Simulation setup

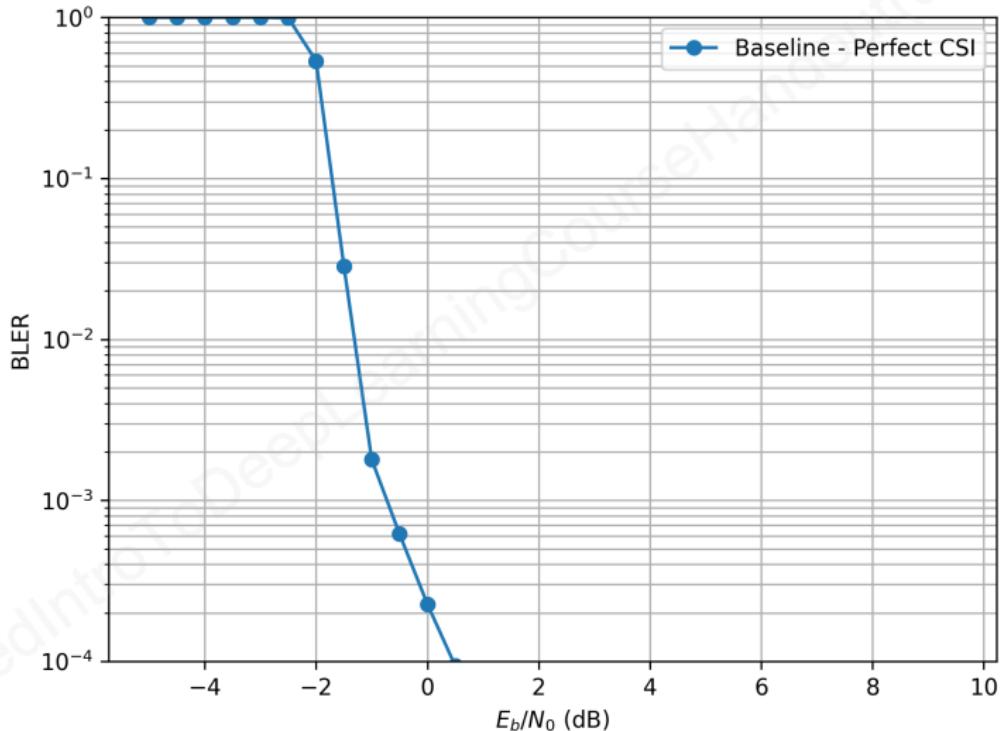
- 1x2 SIMO link
- Carrier frequency: 3.5 GHz
- RMS delay spread: 300 ns
- Speed: 10 m/s
- Channel model: 3GPP CDL-C (NLoS)
- Subcarrier spacing: 30 kHz
- DFT size: $K = 128$
- Number of OFDM symbols: $M = 14$
- Modulation: QPSK (2 bits per data carrying RE)
- Pilot structure: 3rd and 12th symbols full of pilots (Kronecker)
- FEC: 5G NR LDPC with rate 1/2 and length 2784 bit

Neural receiver architecture

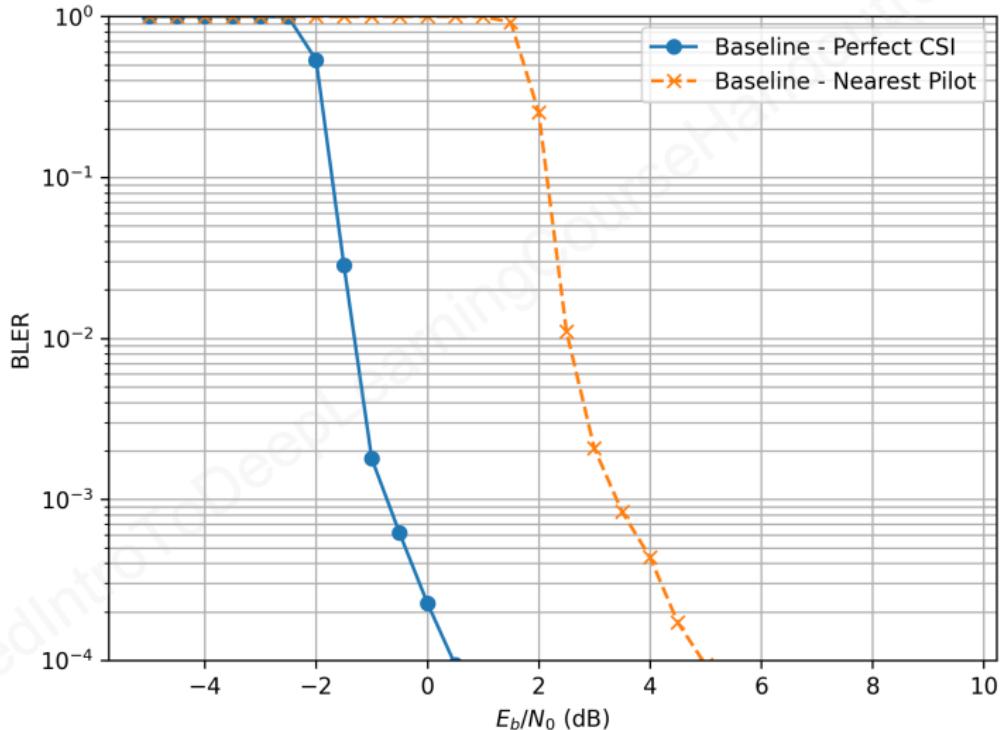


- 4 residual blocks
- One input convolution and one output convolution
- All convolution layers have kernel size set to (3, 3)
- All convolution layers have 128 channels

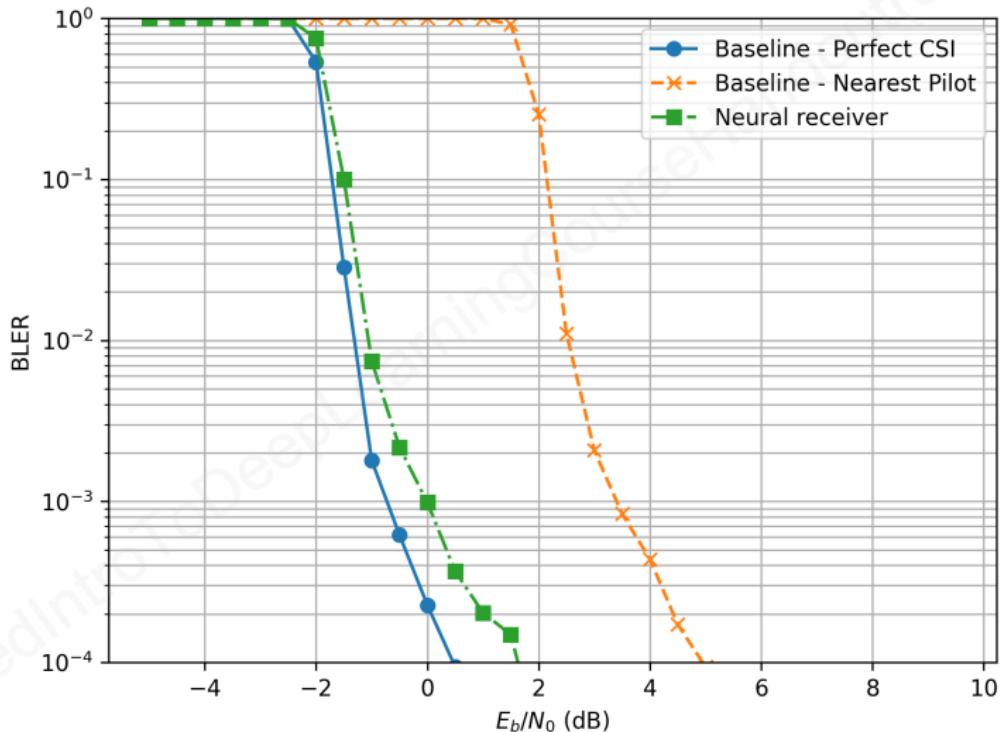
Bit error rate results



Bit error rate results



Bit error rate results

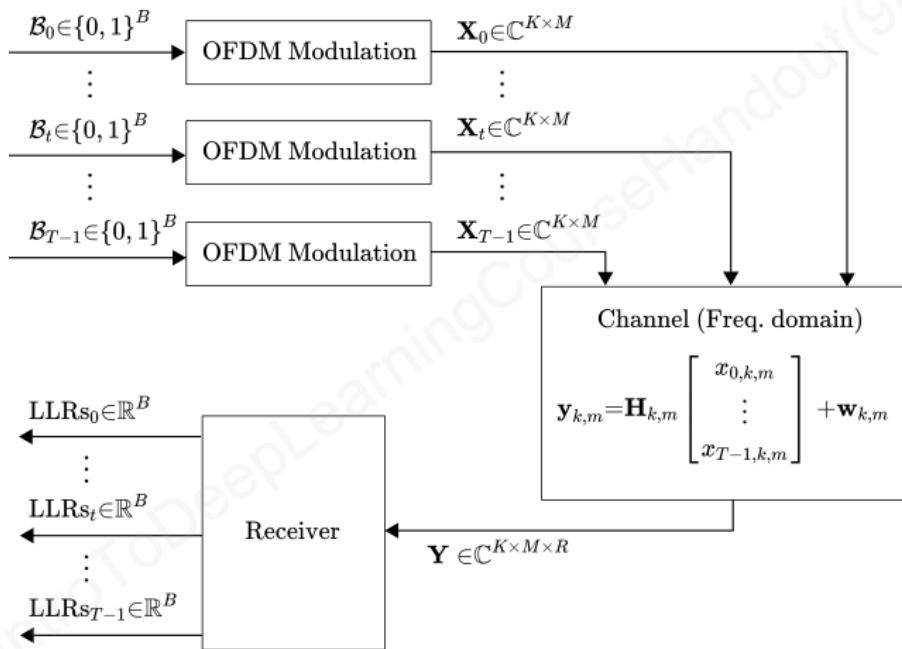


Neural receiver example notebook



Neural Receiver

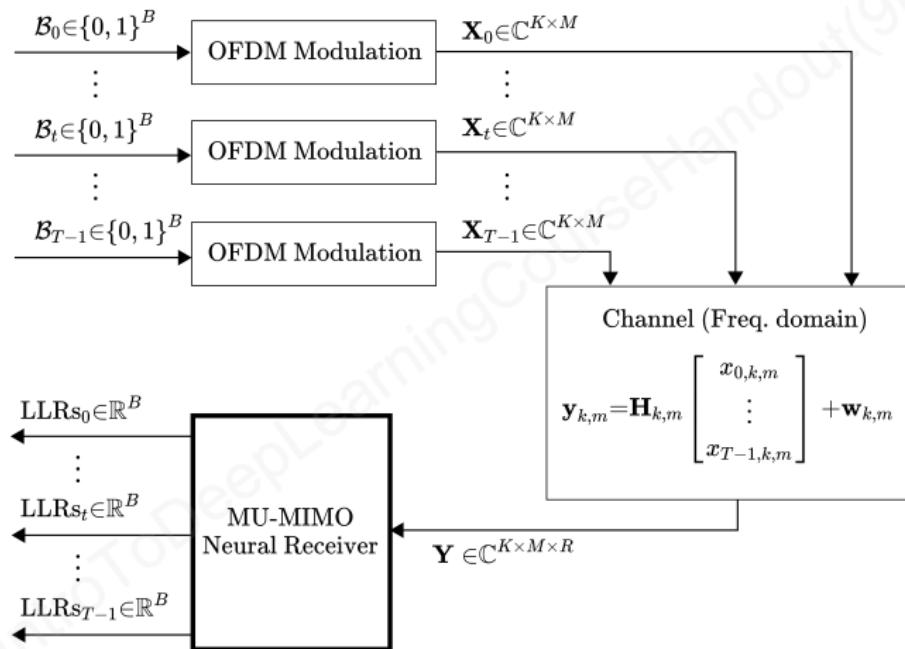
MU-MIMO OFDM architecture



- T : Number of transmitters

- R : Number of receive antennas

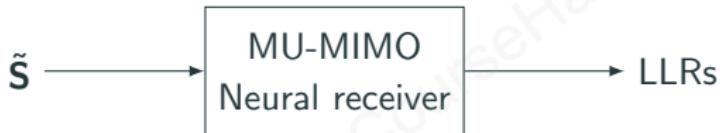
MU-MIMO OFDM architecture with neural receiver



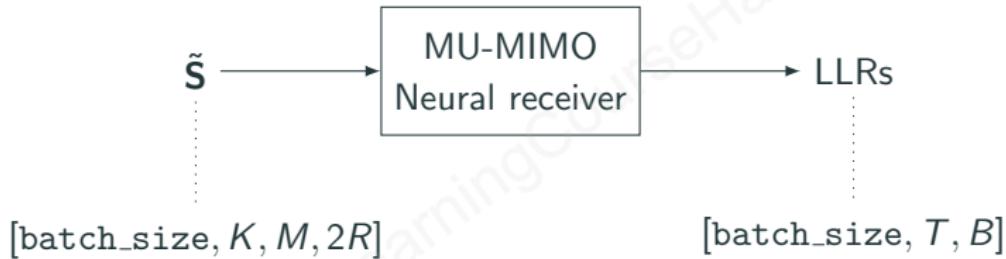
- T : Number of transmitters

- R : Number of receive antennas

MU-MIMO neural receivers – Overview

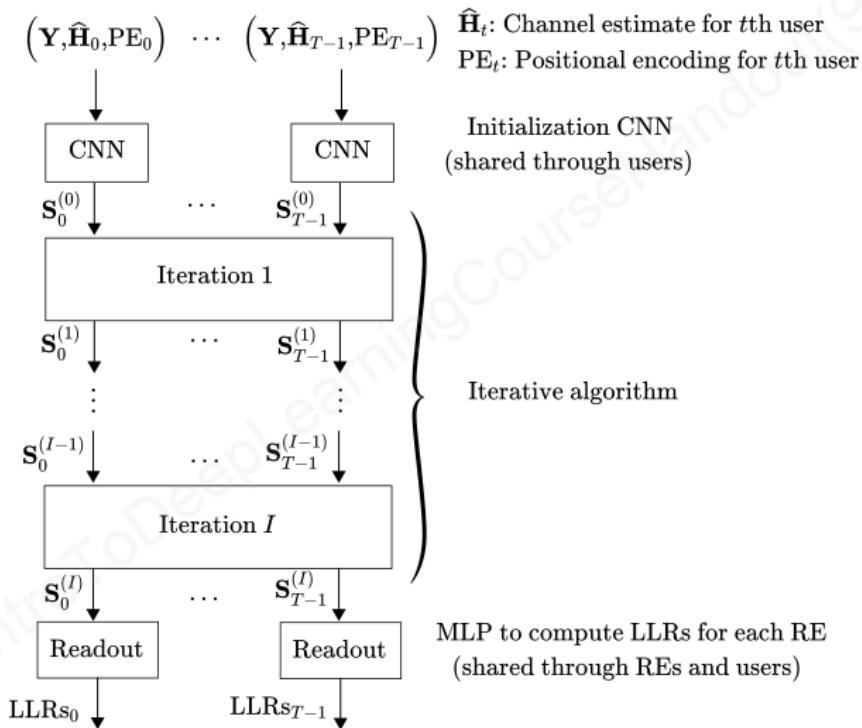


MU-MIMO neural receivers – Overview



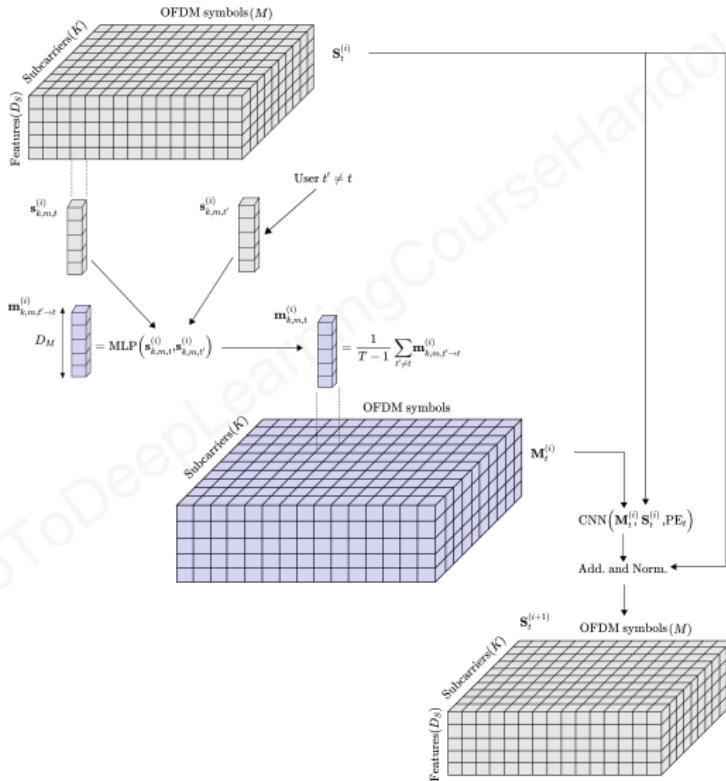
- K : # subcarriers
- M : # OFDM symbols
- B : # bits per resource grid
- T : # transmitters
- R : # receive antennas

MU-MIMO OFDM neural receiver architecture



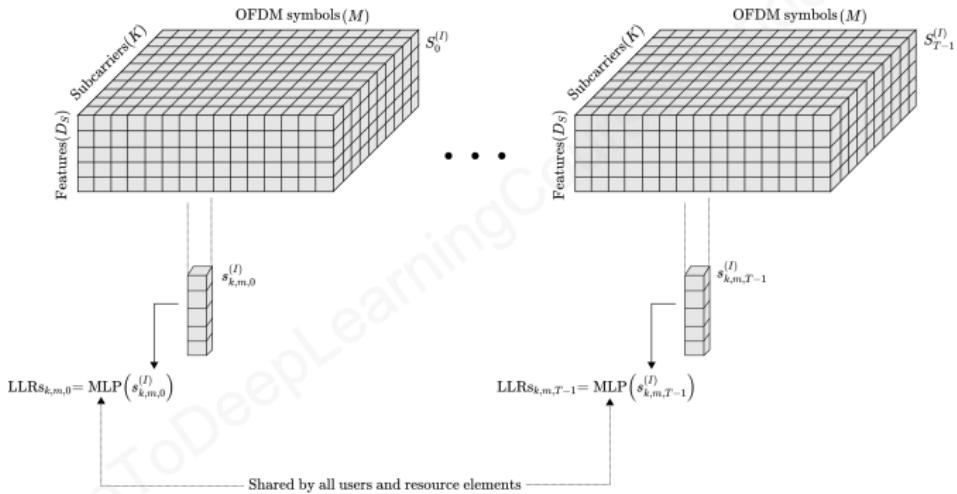
MU-MIMO OFDM neural receiver architecture

Details of an iteration



MU-MIMO OFDM neural receiver architecture

The readout network computes LLRs for every user and resource element independently

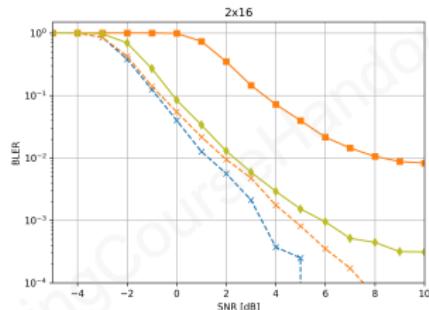
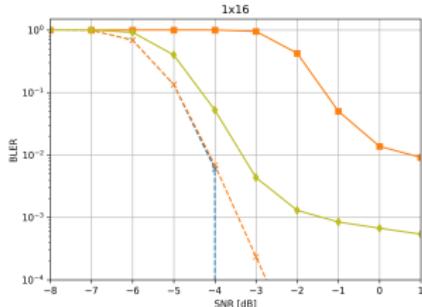


Simulation setup

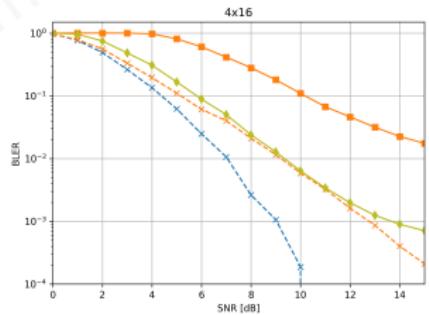
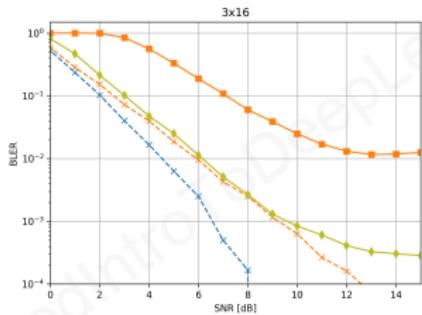
- 16 receive antennas
- 1 to 4 single-antenna transmitters
- Carrier frequency: 3.5 GHz
- Training range of speed: 0 m/s to 20 m/s
- Channel model: 3GPP urban macro-cell (UMa)
- Subcarrier spacing: 30 kHz
- DFT size: $K = 12$
- Number of OFDM symbols: $M = 14$
- Modulation: 16-QAM (4 bits per data carrying RE)
- Pilot structure: Kronecker
- FEC: 5G NR LDPC with rate 1/2 and length 576 bit

Block error rate results

Results for a speed of 10 m/s



Max. Lik. w. Perf. CSI
LMMSE w. Perf. CSI
LMMSE w/o. Perf. CSI
Neural Receiver w/o. Perf. CSI

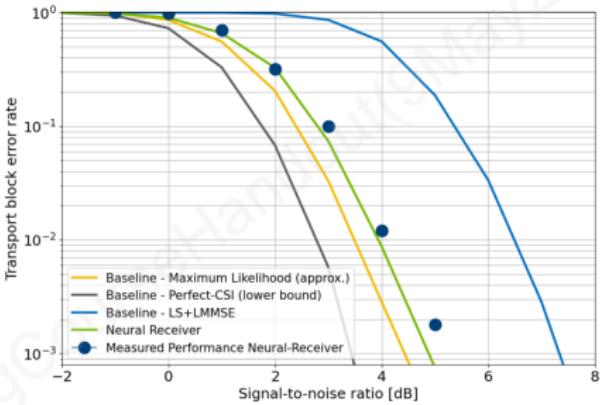


Mobile World Congress 2023 Demonstrator⁵



Hardware-in-the-loop experiment

- Multi-user MIMO configuration with 2 users
- 2 ants per user & 4 receive ants
- 217 PRBs (=80 MHz bandwidth)



Verified by Rohde & Schwarz test equipment

- Fully 5G NR compliant PUSCH signals
- Fading profiles following the 3GPP conformance tests
 - TDL-B 100ns/400Hz
 - TDL-C 300ns/100Hz

5

<https://developer.nvidia.com/blog/>

Benefits and drawbacks of this architecture

Benefits:

- Works for a variable number of users
- Works for a variable number of OFDM symbols and subcarriers
- Works for any code-rate
- Could in principle handle multiple pilot patterns

Drawbacks:

- Number of receive antennas is fixed
- Constellation is fixed
- Risk of overfitting on the channel model used for training
- Lowering the complexity is still under investigation

Useful material on DL Receivers

- M. Honkala, D. Korpi and J. M. J. Huttunen, "DeepRx: Fully Convolutional Deep Learning Receiver," in IEEE Transactions on Wireless Communications, June 2021.
- K. Pratik, B. D. Rao and M. Welling, "RE-MIMO: Recurrent and Permutation Equivariant Neural MIMO Detection," in IEEE Transactions on Signal Processing, 2021.
- D. Korpi, M. Honkala, J. M. J. Huttunen and V. Starck, "DeepRx MIMO: Convolutional MIMO Detection with Learned Multiplicative Transformations," IEEE International Conference on Communications, 2021.
- A. Scotti, N. N. Moghadam, D. Liu, K. Gafvert, J. Huang, "Graph Neural Networks for Massive MIMO Detection", arXiv preprint, 2021.
- M. Khani, M. Alizadeh, J. Hoydis and P. Fleming, "Adaptive Neural Signal Detection for Massive MIMO," in IEEE Transactions on Wireless Communications, Aug. 2020.

Ongoing research

- Generalization to MU-MIMO still presents challenges
- Reducing pilot and CP overhead [YLJ18]
- End-to-end learning [AAH22, YLJ21]

Lessons learned

Deep Neural Receivers

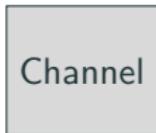
- Conventional OFDM receivers rely on assumptions on the channel that typically don't hold in practice
- This leads to suboptimal performance, especially under mobility or long delay spread
- Convolutional neural receivers leverage convolutional layers to process the resource grid of received symbols
- Different approaches were proposed to handle multiple streams: use of attention, GNN, ad hoc architectures...
- Neural receivers are trained using backpropagation and SGD on the binary cross-entropy
- Neural receivers show promising performance compared to conventional detection algorithms



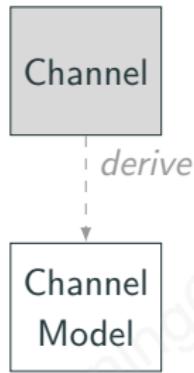
End-to-end learning

**“The fundamental problem of communication
is that of reproducing at one point either
exactly or approximately a message selected
at another point.” (Shannon, 1948)**

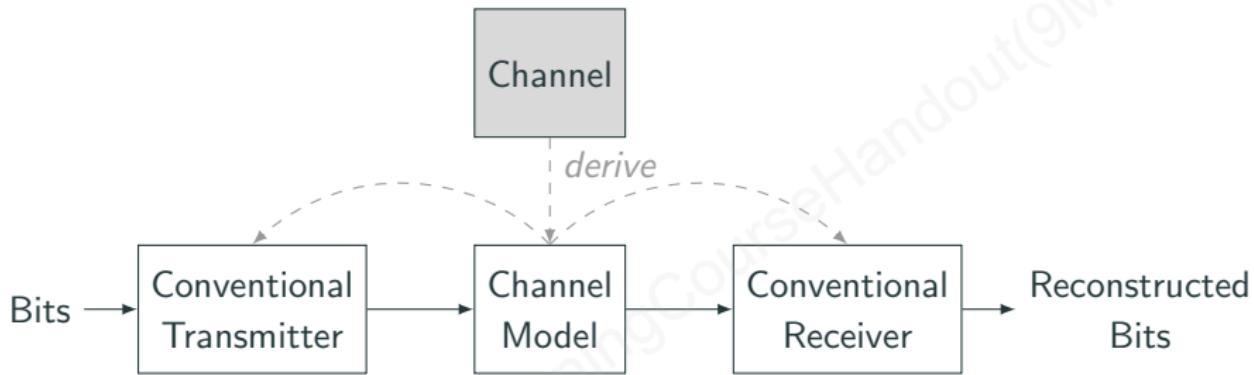
How communications systems are designed today



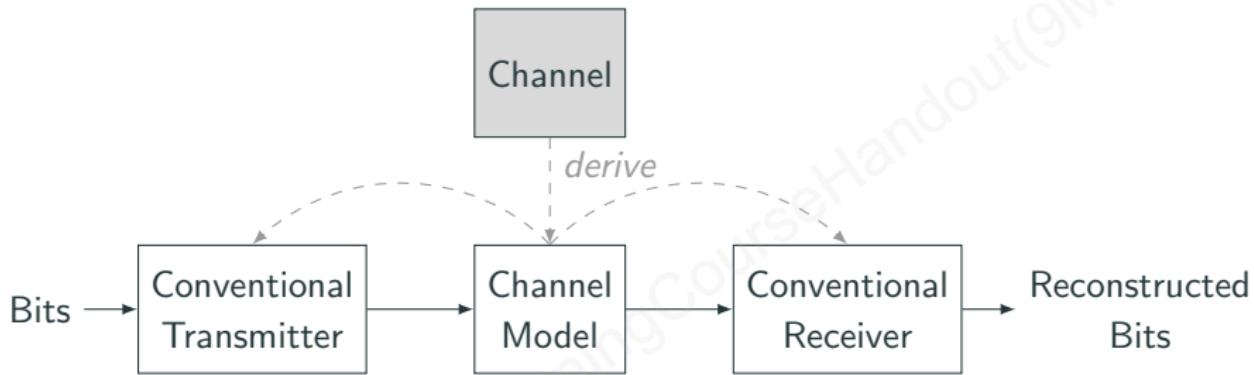
How communications systems are designed today



How communications systems are designed today

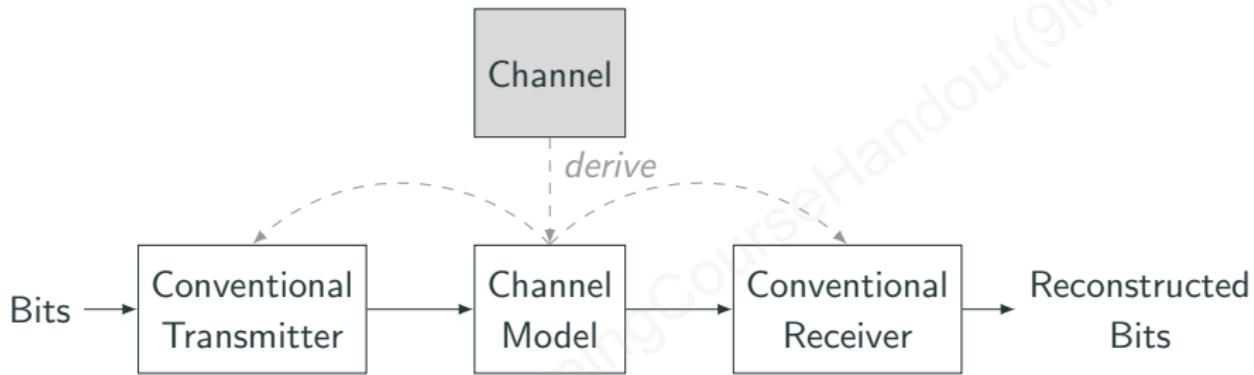


How communications systems are designed today



Drawbacks:

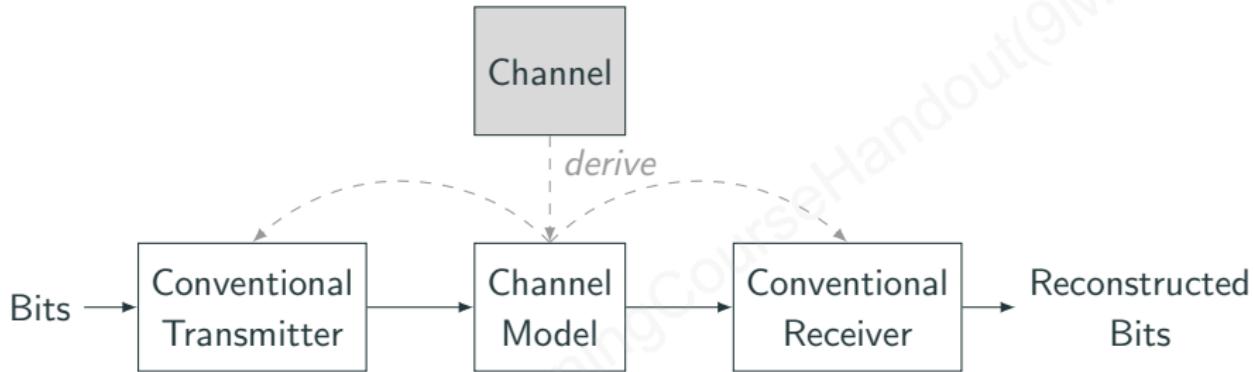
How communications systems are designed today



Drawbacks:

- Channel model intractable → simplifications are made
e.g., ignore hardware impairments

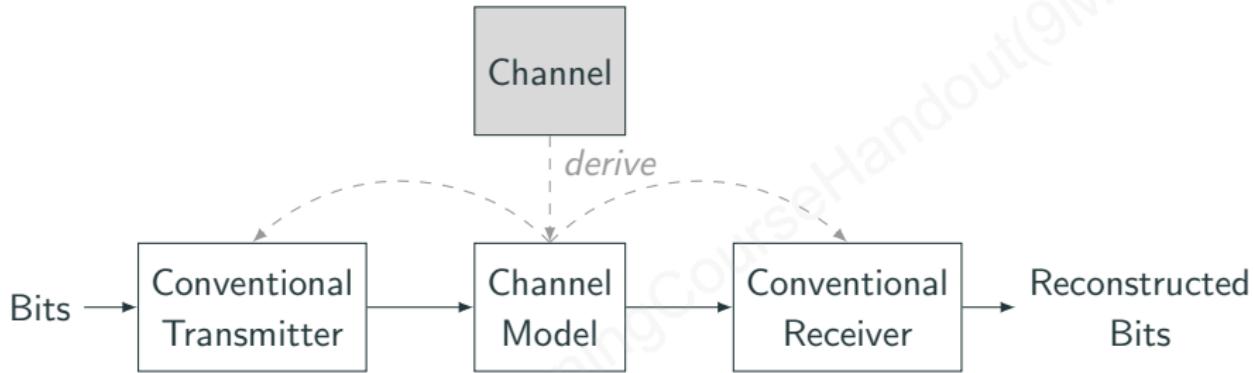
How communications systems are designed today



Drawbacks:

- Channel model intractable → simplifications are made
e.g., ignore hardware impairments
- Mismatch between the channel model and the actual channel

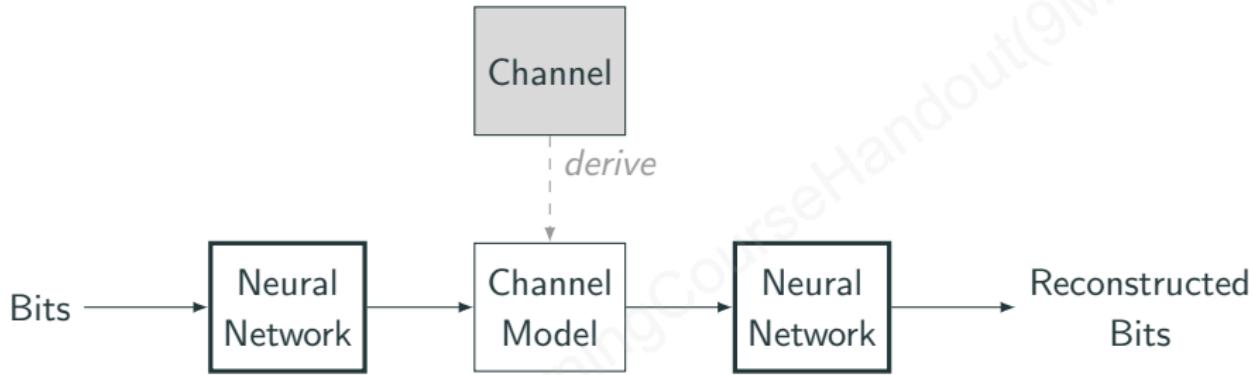
How communications systems are designed today



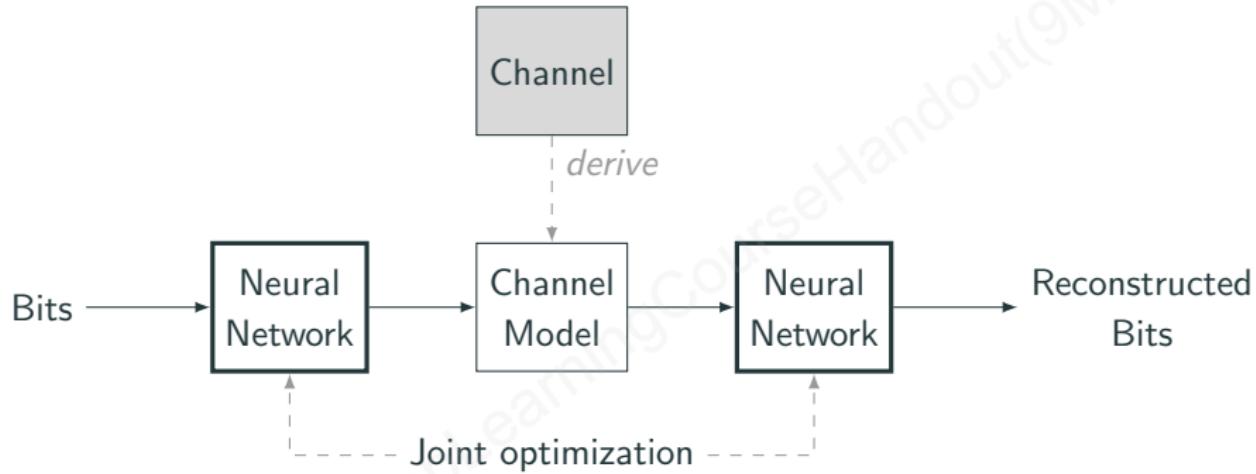
Drawbacks:

- Channel model intractable → simplifications are made
e.g., ignore hardware impairments
- Mismatch between the channel model and the actual channel
- Conventional algorithms may be too complex for practical use

Communications systems as autoencoder

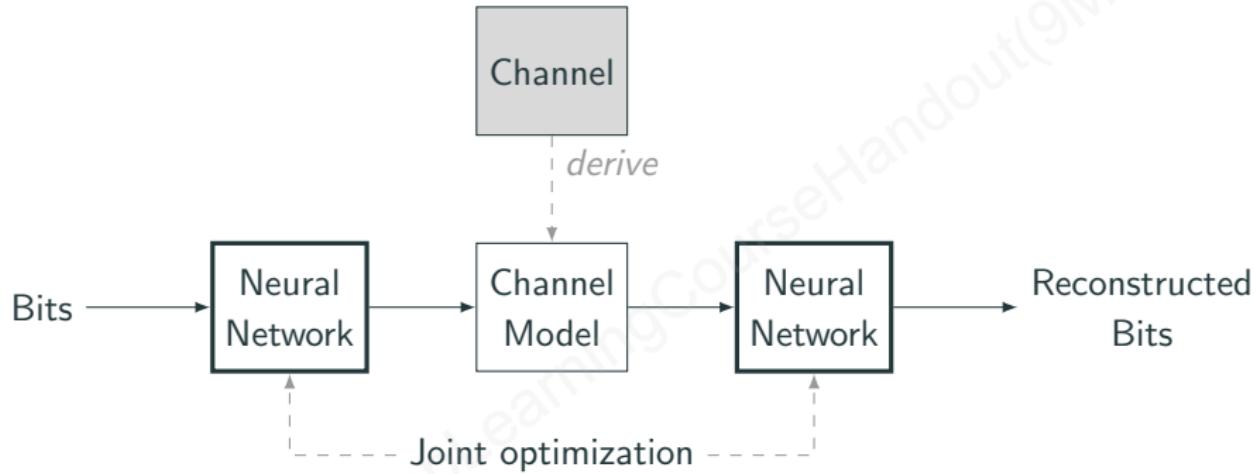


Communications systems as autoencoder



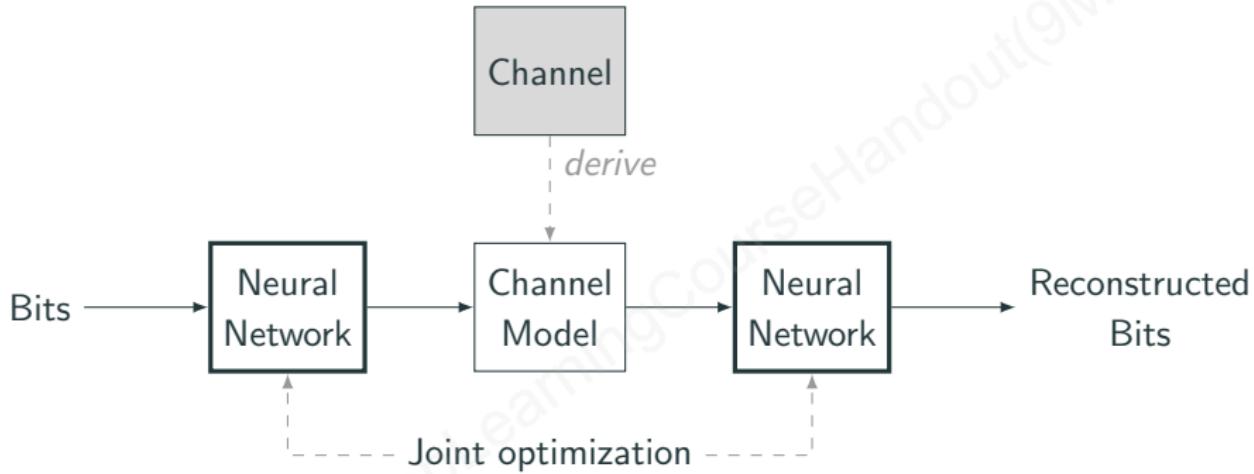
- Joint optimization to maximize an information rate R [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] by *backpropagation* and *stochastic gradient descent* (SGD) [OH17]

Communications systems as autoencoder



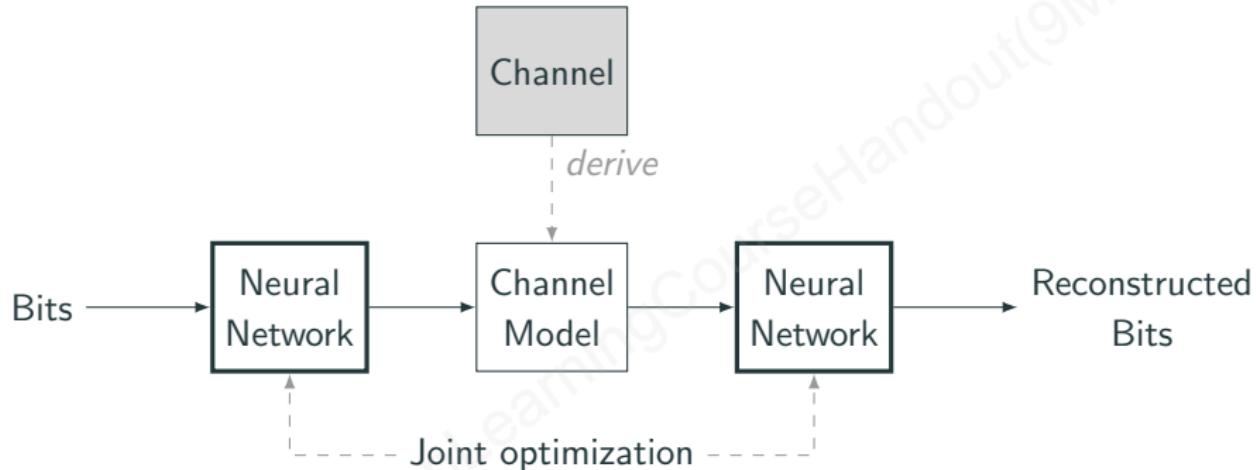
- Joint optimization to maximize an information rate R [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] by *backpropagation* and *stochastic gradient descent* (SGD) [OH17]
- Universal approach:

Communications systems as autoencoder



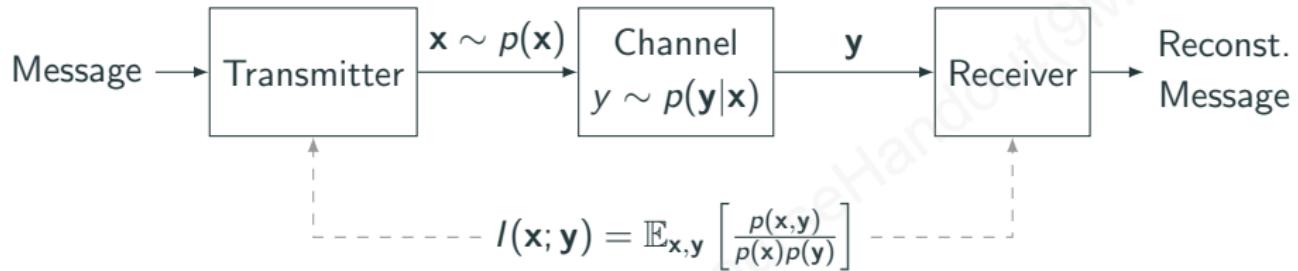
- Joint optimization to maximize an information rate R [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] by *backpropagation* and *stochastic gradient descent* (SGD) [OH17]
- Universal approach:
 - Can be applied to any *differentiable* channel model...

Communications systems as autoencoder

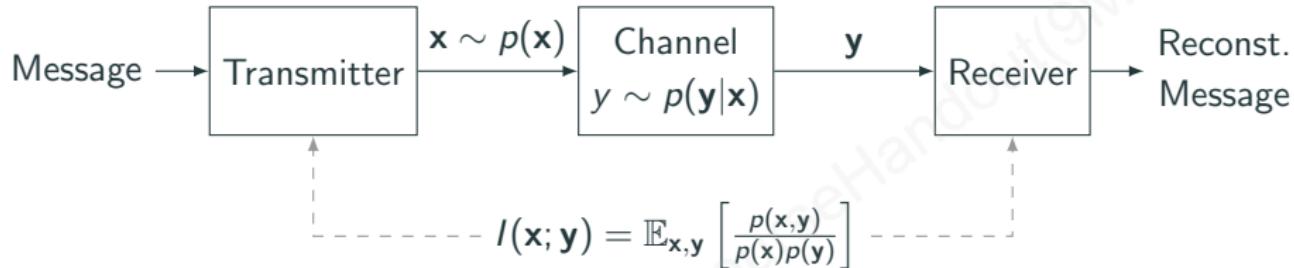


- Joint optimization to maximize an information rate R [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] by *backpropagation* and *stochastic gradient descent* (SGD) [OH17]
- Universal approach:
 - Can be applied to any *differentiable* channel model...
 - ... to train any part of the transmitter and/or receiver

Information theory perspective

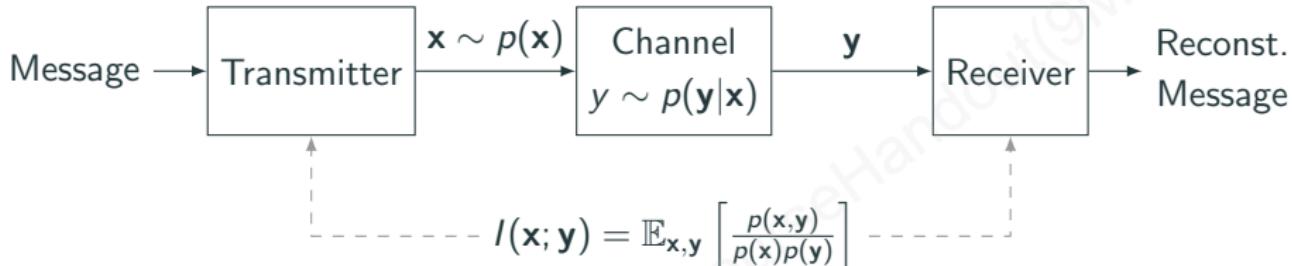


Information theory perspective



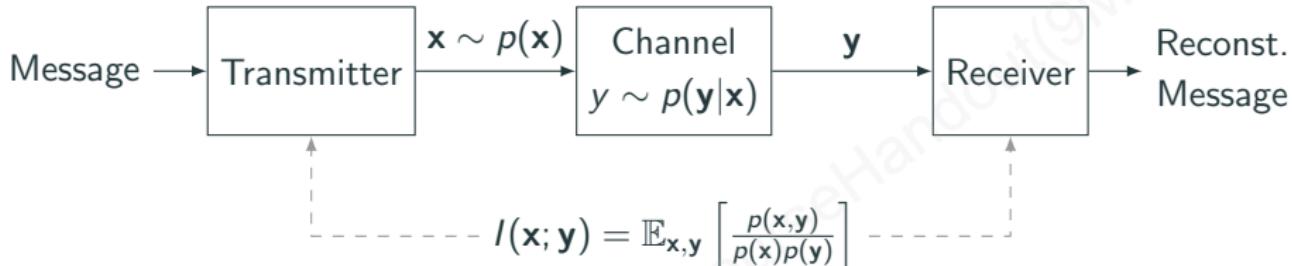
- An information rate is said to be *achievable* if there exists a sequence of codes indexed by the codelength such that the maximum probability of error vanishes as the codelength $\rightarrow \infty$

Information theory perspective



- An information rate is said to be *achievable* if there exists a sequence of codes indexed by the codelength such that the maximum probability of error vanishes as the codelength $\rightarrow \infty$
- $I(x; y)$ is the *highest achievable information rate* [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] for the channel distribution $p(y|x)$ and the input distribution $p(x)$

Information theory perspective

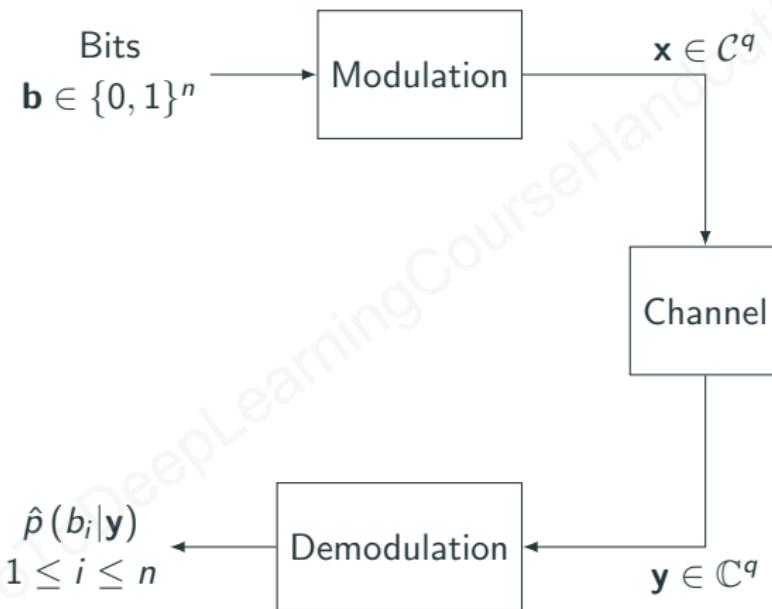


- An information rate is said to be *achievable* if there exists a sequence of codes indexed by the codelength such that the maximum probability of error vanishes as the codelength $\rightarrow \infty$
- $I(x; y)$ is the *highest achievable information rate* [$\text{bit} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$] for the channel distribution $p(y|x)$ and the input distribution $p(x)$
- *Capacity:* Highest achievable rate over all channel input distributions:

$$C := \max_{p(x)} I(x; y)$$

Information theory perspective

Practical systems rely on *bit-metric decoding* (BMD) [CTB98]



Information theory perspective

Consider the following information rate [Bö17]

$$R := \sum_{i=1}^n I(b_i; \mathbf{y}) - \sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))$$

where

Information theory perspective

Consider the following information rate [Bö17]

$$R := \sum_{i=1}^n I(b_i; \mathbf{y}) - \sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))$$

where

- $I(b_i; \mathbf{y})$: Bit-wise mutual information

Information theory perspective

Consider the following information rate [Bö17]

$$R := \sum_{i=1}^n I(b_i; \mathbf{y}) - \sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))$$

where

- $I(b_i; \mathbf{y})$: Bit-wise mutual information
- $p(b_i|\mathbf{y})$: *True (optimal)* bit-wise posterior distribution

Information theory perspective

Consider the following information rate [Bö17]

$$R := \sum_{i=1}^n I(b_i; \mathbf{y}) - \sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))$$

where

- $I(b_i; \mathbf{y})$: Bit-wise mutual information
- $p(b_i|\mathbf{y})$: *True (optimal) bit-wise posterior distribution*
- $\hat{p}(b_i|\mathbf{y})$: Bit-wise posterior distribution *approximated by the receiver*

Information theory perspective

Consider the following information rate [Bö17]

$$R := \sum_{i=1}^n I(b_i; \mathbf{y}) - \sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))$$

where

- $I(b_i; \mathbf{y})$: Bit-wise mutual information
- $p(b_i|\mathbf{y})$: *True (optimal) bit-wise posterior distribution*
- $\hat{p}(b_i|\mathbf{y})$: Bit-wise posterior distribution *approximated by the receiver*
- $D_{\text{KL}}(\cdot \parallel \cdot)$: Kullback-Leibler (KL) divergence

Information theory perspective

Consider the following information rate [Bö17]

$$R := \underbrace{\sum_{i=1}^n I(b_i; \mathbf{y})}_{\text{Achievable rate assuming an optimal receiver}} - \underbrace{\sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))}_{\text{Rate-loss due to receiver mismatch}}$$

Information theory perspective

Consider the following information rate [Bö17]

$$R := \underbrace{\sum_{i=1}^n I(b_i; \mathbf{y})}_{\text{Achievable rate assuming an optimal receiver}} - \underbrace{\sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))}_{\text{Rate-loss due to receiver mismatch}}$$

- R is an achievable rate for BMD with receiver mismatch

Information theory perspective

Consider the following information rate [Bö17]

$$R := \underbrace{\sum_{i=1}^n I(b_i; \mathbf{y})}_{\text{Achievable rate assuming an optimal receiver}} - \underbrace{\sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))}_{\text{Rate-loss due to receiver mismatch}}$$

- R is an achievable rate for BMD with receiver mismatch
- In general, $R \leq I(x; y) \dots$

Information theory perspective

Consider the following information rate [Bö17]

$$R := \underbrace{\sum_{i=1}^n I(b_i; \mathbf{y})}_{\text{Achievable rate assuming an optimal receiver}} - \underbrace{\sum_{i=1}^n D_{\text{KL}}(p(b_i|\mathbf{y}) \parallel \hat{p}(b_i|\mathbf{y}))}_{\text{Rate-loss due to receiver mismatch}}$$

- R is an achievable rate for BMD with receiver mismatch
- In general, $R \leq I(x; y) \dots$
- ... but the gap is small in practice

Information theory perspective

R is closely related to the *binary cross-entropy (BCE)*:

$$\text{BCE} = - \sum_{i=1}^n \mathbb{E}_{b_i, \mathbf{y}} [\log \hat{p}(b_i | \mathbf{y})] = n - R$$

Information theory perspective

R is closely related to the *binary cross-entropy (BCE)*:

$$\text{BCE} = - \sum_{i=1}^n \mathbb{E}_{b_i, \mathbf{y}} [\log \hat{p}(b_i | \mathbf{y})] = n - R$$

Minimize BCE \leftarrow Equivalent \rightarrow Maximize R

Information theory perspective

R is closely related to the *binary cross-entropy (BCE)*:

$$\text{BCE} = - \sum_{i=1}^n \mathbb{E}_{b_i, \mathbf{y}} [\log \hat{p}(b_i | \mathbf{y})] = n - R$$

Minimize BCE \leftarrow Equivalent \rightarrow Maximize R

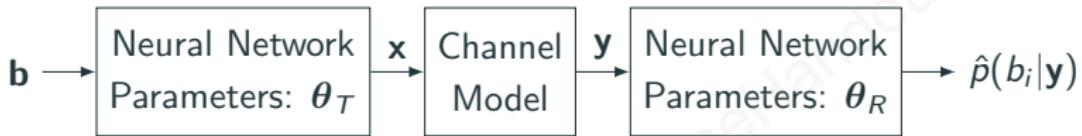
BCE can be estimated by Monte Carlo integration

$$\text{BCE} \approx \frac{1}{S} \sum_{s=1}^S \sum_{i=1}^n \log \hat{p}(b_i^{(s)} | \mathbf{y}^{(s)})$$

where S is the batch size.

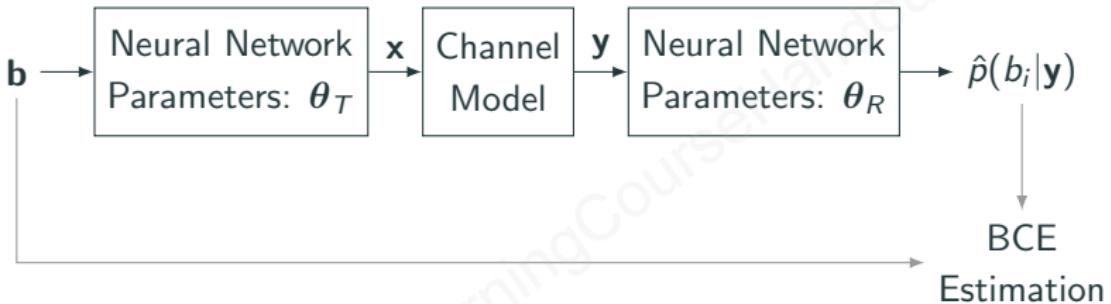
Training with backpropagation

Training with backpropagation is straightforward:



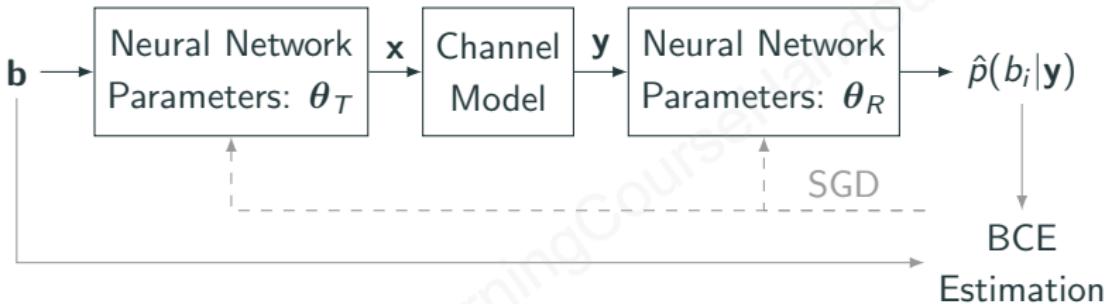
Training with backpropagation

Training with backpropagation is straightforward:



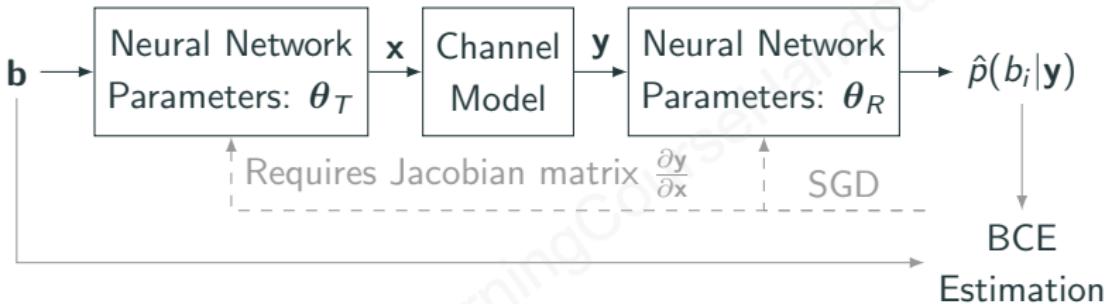
Training with backpropagation

Training with backpropagation is straightforward:

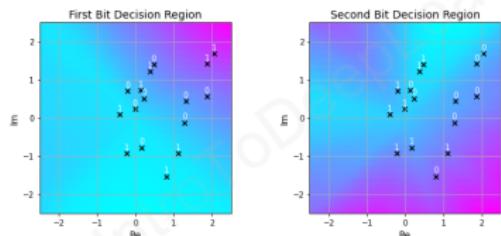
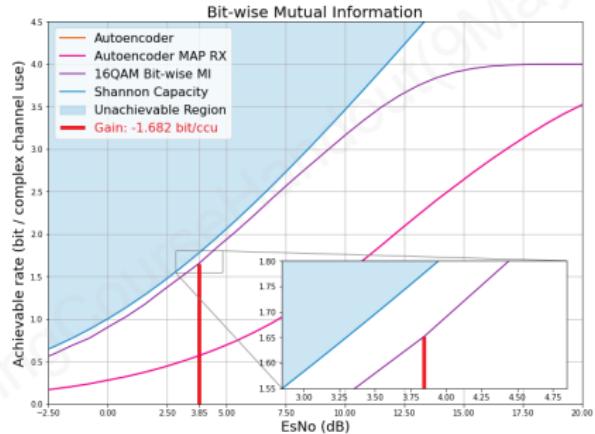
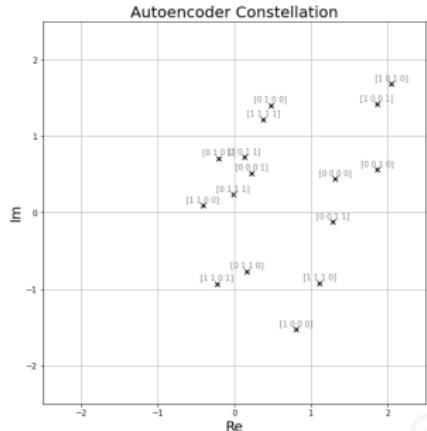


Training with backpropagation

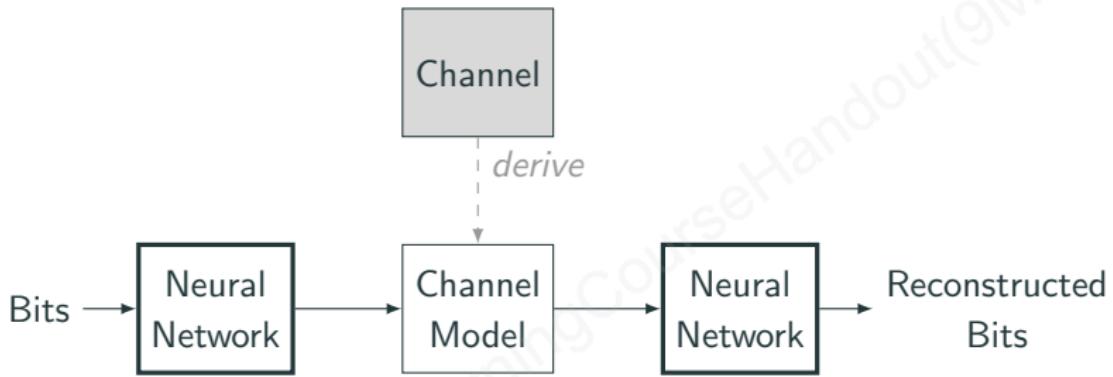
Training with backpropagation is straightforward:



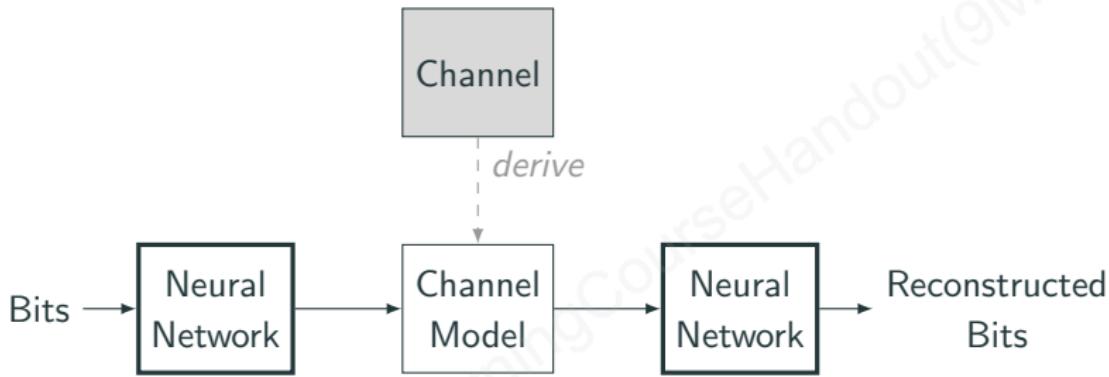
Video: Bit-autoencoder training in action



Motivation for training without a channel model

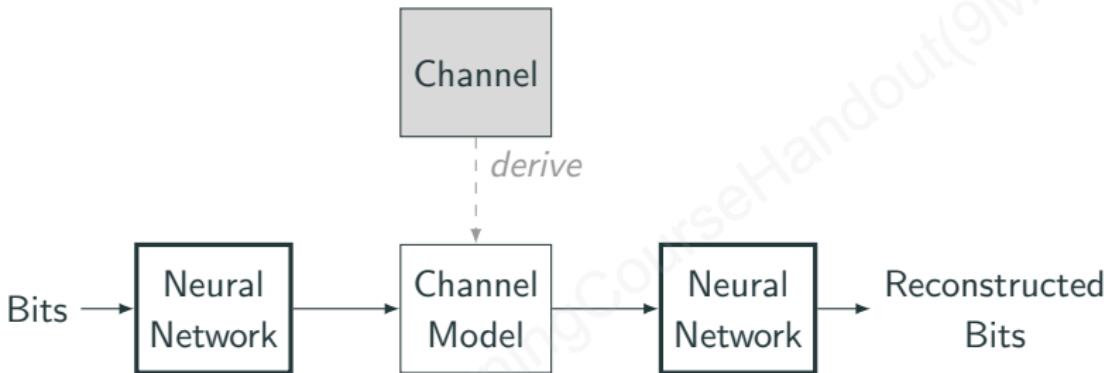


Motivation for training without a channel model



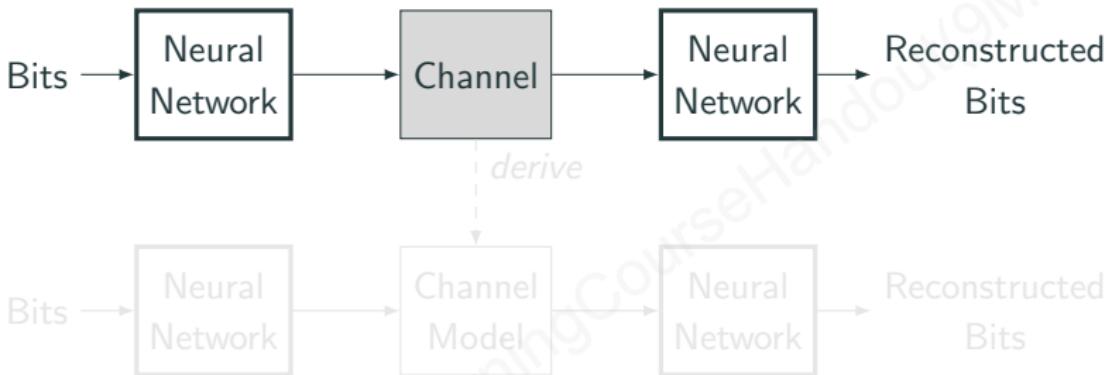
- Mismatch between the channel model and the actual channel might lead to disappointing performance

Motivation for training without a channel model



- Mismatch between the channel model and the actual channel might lead to disappointing performance
- The channel model might not be differentiable (e.g., quantization)
→ Backpropagation of the gradients to the transmitter is not possible

Motivation for training without a channel model



- Mismatch between the channel model and the actual channel might lead to disappointing performance
- The channel model might not be differentiable (e.g., quantization)
→ Backpropagation of the gradients to the transmitter is not possible
- *Solution: Train over the actual channel*

How to train over the actual channel?

Three known methods:

How to train over the actual channel?

Three known methods:

1. Pre-train the system on a channel model, then fine-tune the receiver on the actual channel [DCHt18]

How to train over the actual channel?

Three known methods:

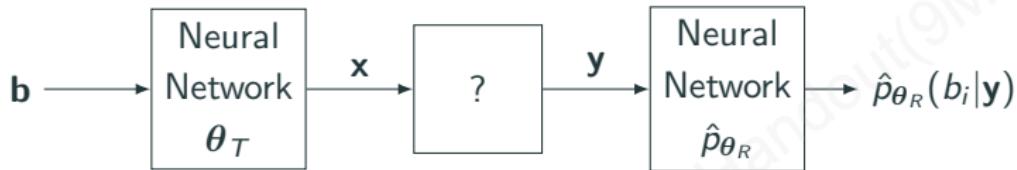
1. Pre-train the system on a channel model, then fine-tune the receiver on the actual channel [DCHt18]
2. Learn a model of the channel using a generative adversarial network (GAN), then train over the learned model [ORWH18]

How to train over the actual channel?

Three known methods:

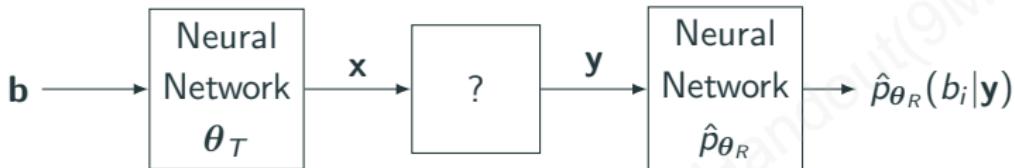
1. Pre-train the system on a channel model, then fine-tune the receiver on the actual channel [DCHt18]
2. Learn a model of the channel using a generative adversarial network (GAN), then train over the learned model [ORWH18]
3. Use reinforcement learning (RL) to train the transmitter, and conventional training to train the receiver [AH19]

Training over the channel with reinforcement learning



Key observations:

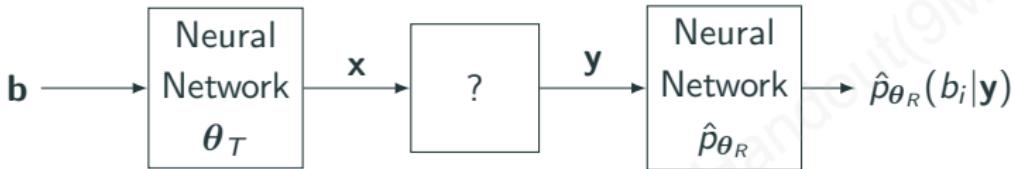
Training over the channel with reinforcement learning



Key observations:

- Training of the receiver on the actual channel is easy, as it does not require backpropagating through the channel

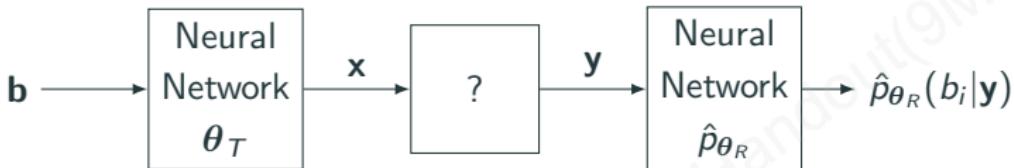
Training over the channel with reinforcement learning



Key observations:

- Training of the receiver on the actual channel is easy, as it does not require backpropagating through the channel
- Training of the transmitter on the actual channel is challenging, as one cannot compute the channel gradients

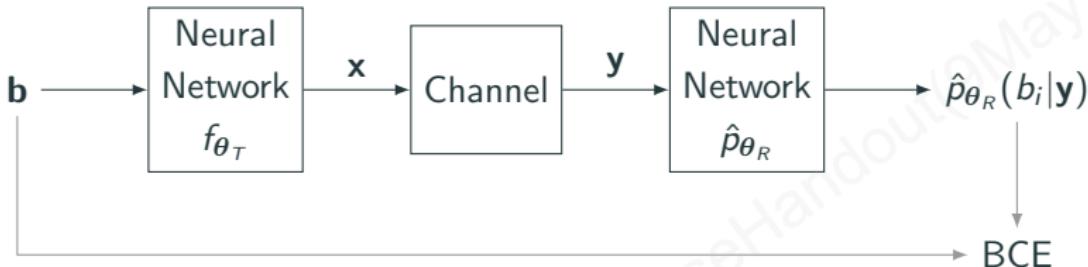
Training over the channel with reinforcement learning



Key observations:

- Training of the receiver on the actual channel is easy, as it does not require backpropagating through the channel
- Training of the transmitter on the actual channel is challenging, as one cannot compute the channel gradients
- → *Key idea: Use reinforcement learning to estimate the transmitter gradients*

Training over the channel with reinforcement learning



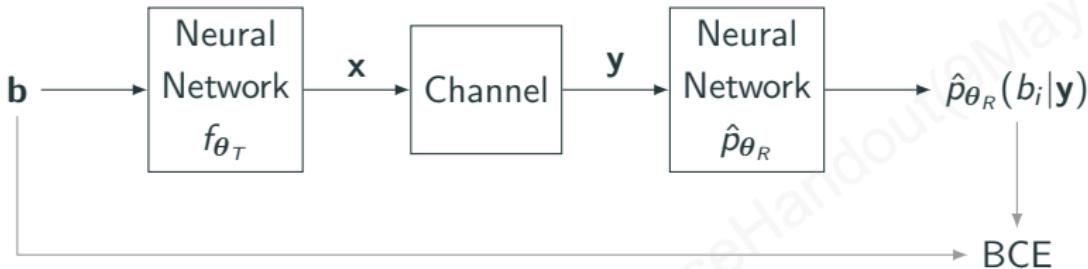
$$\begin{aligned}\text{BCE}(\theta_T, \theta_R) &= \mathbb{E}_{\mathbf{b}, \mathbf{y}} \left[- \sum_{i=1}^n \log \hat{p}_{\theta_R}(b_i | \mathbf{y}) \mid \theta_T \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int p(\mathbf{y} \mid \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right]\end{aligned}$$

where

$$\ell(\mathbf{b}, \mathbf{y}, \theta_R) = - \sum_{i=1}^n \log \hat{p}_{\theta_R}(b_i | \mathbf{y})$$

and $\mathbf{b} = [b_1, \dots, b_n]^T$

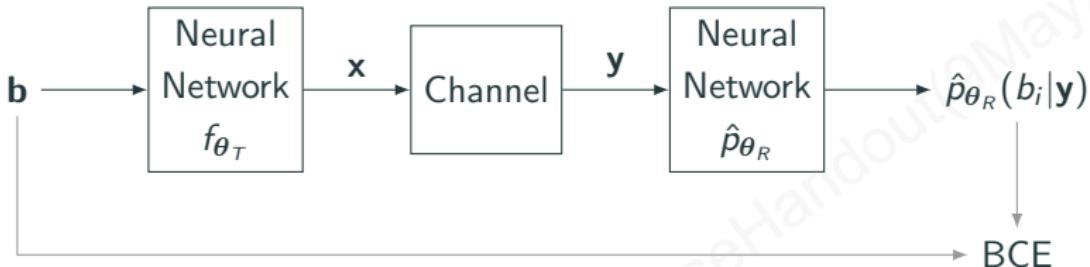
Training over the channel with reinforcement learning



Let's now compute the gradient of the BCE w.r.t. θ_T :

$$\nabla_{\theta_T} \text{BCE}(\theta_T, \theta_R) = \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int p(\mathbf{y} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right]$$

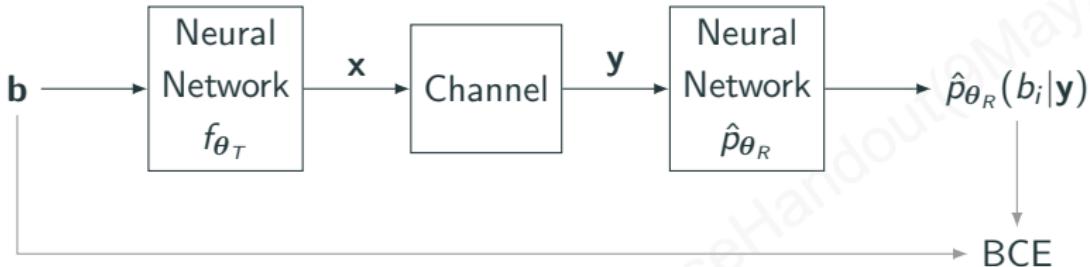
Training over the channel with reinforcement learning



Let's now compute the gradient of the BCE w.r.t. θ_T :

$$\begin{aligned}\nabla_{\theta_T} \text{BCE}(\theta_T, \theta_R) &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int p(\mathbf{y} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \nabla_{\theta_T} (p(\mathbf{y} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right]\end{aligned}$$

Training over the channel with reinforcement learning

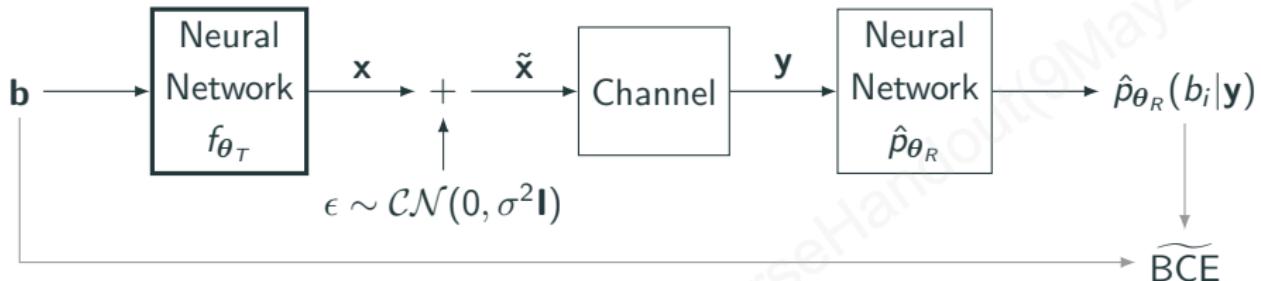


Let's now compute the gradient of the BCE w.r.t. θ_T :

$$\begin{aligned}\nabla_{\theta_T} \text{BCE}(\theta_T, \theta_R) &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int p(\mathbf{y} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \nabla_{\theta_T} (p(\mathbf{y} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \nabla_{\theta_T} f_{\theta_T}(\mathbf{b}) \nabla_{\mathbf{x}} (p(\mathbf{y} | \mathbf{x})) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\mathbf{y} \right]\end{aligned}$$

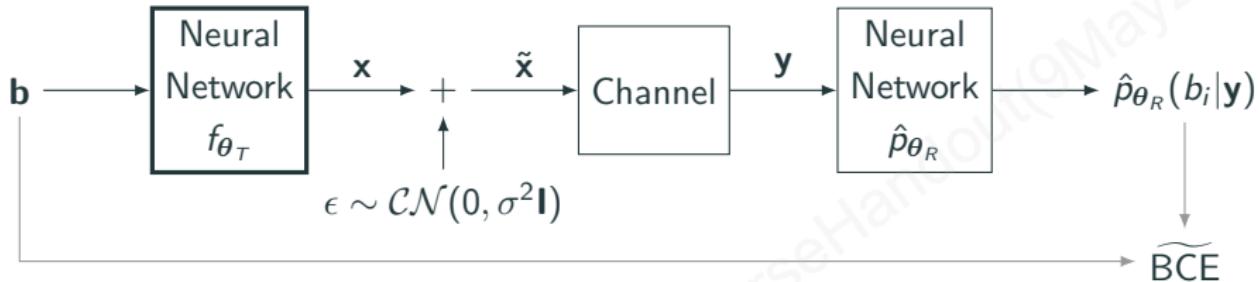
$p(\mathbf{y} | \mathbf{x})$ not known or not differentiable!

Training over the channel with reinforcement learning



Key idea: Relax the channel input to a Gaussian distribution centered at \mathbf{x} and with covariance matrix $\sigma^2 \mathbf{I}$

Training over the channel with reinforcement learning



Key idea: Relax the channel input to a Gaussian distribution centered at \mathbf{x} and with covariance matrix $\sigma^2 \mathbf{I}$

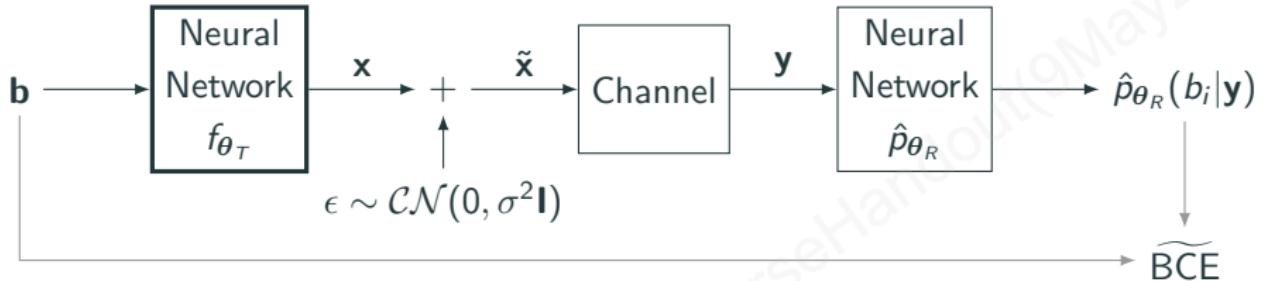
New loss:

$$\widetilde{\text{BCE}}(\theta_T, \theta_R) = \mathbb{E}_{\mathbf{b}} \left[\int \int p(\tilde{\mathbf{x}} | f_{\theta_T}(\mathbf{b})) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right]$$

where

$$p(\tilde{\mathbf{x}} | \mathbf{x}) = \frac{1}{(2\pi)^n \sigma^n} \exp \left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|^2}{2\sigma^2} \right)$$

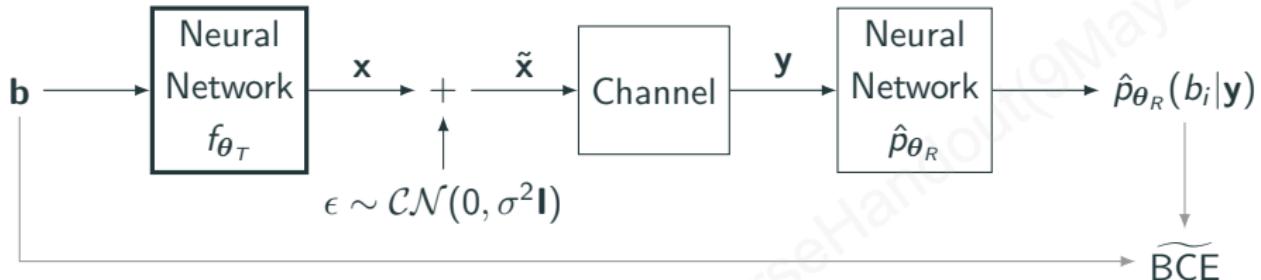
Training over the channel with reinforcement learning



Gradient of $\widetilde{\text{BCE}}$ w.r.t. θ_T :

$$\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) = \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int \int p(\tilde{\mathbf{x}} | f_{\theta_T}(\mathbf{b})) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right]$$

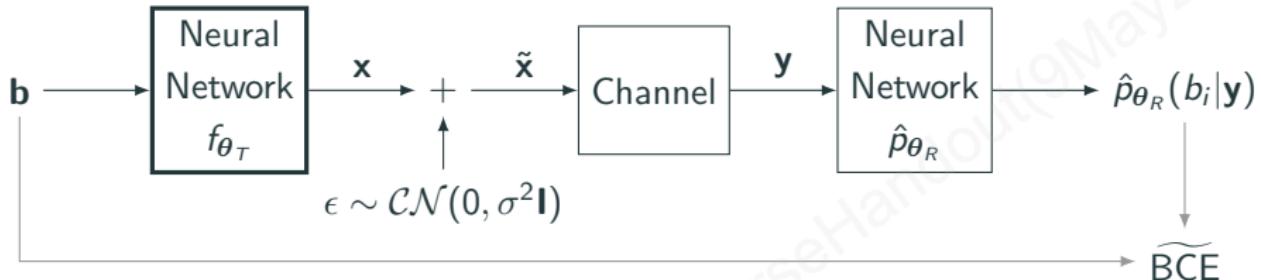
Training over the channel with reinforcement learning



Gradient of $\widetilde{\text{BCE}}$ w.r.t. θ_T :

$$\begin{aligned}\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int \int p(\tilde{\mathbf{x}} | f_{\theta_T}(\mathbf{b})) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \int \nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right]\end{aligned}$$

Training over the channel with reinforcement learning

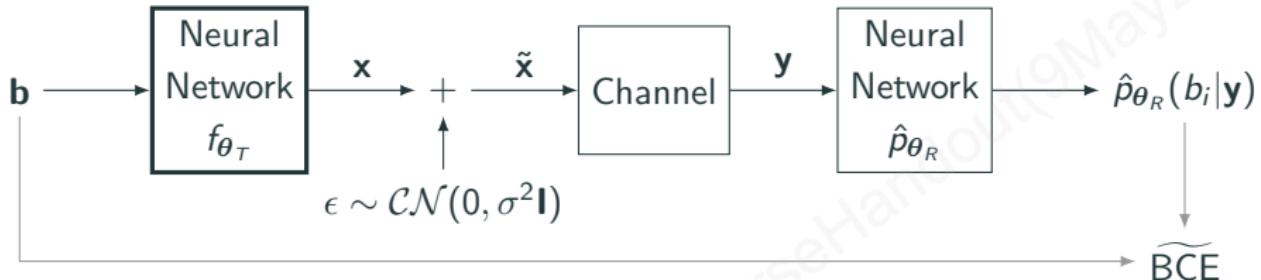


Gradient of $\widetilde{\text{BCE}}$ w.r.t. θ_T :

$$\begin{aligned}\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}} \left[\int \int p(\tilde{\mathbf{x}} | f_{\theta_T}(\mathbf{b})) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \int \nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right] \\ &= \mathbb{E}_{\mathbf{b}} \left[\int \int \nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \right. \\ &\quad \left. \cdot p(\tilde{\mathbf{x}} | f_{\theta_T}(\mathbf{b})) p(\mathbf{y} | \tilde{\mathbf{x}}) \ell(\mathbf{b}, \mathbf{y}, \theta_R) d\tilde{\mathbf{x}} d\mathbf{y} \right]\end{aligned}$$

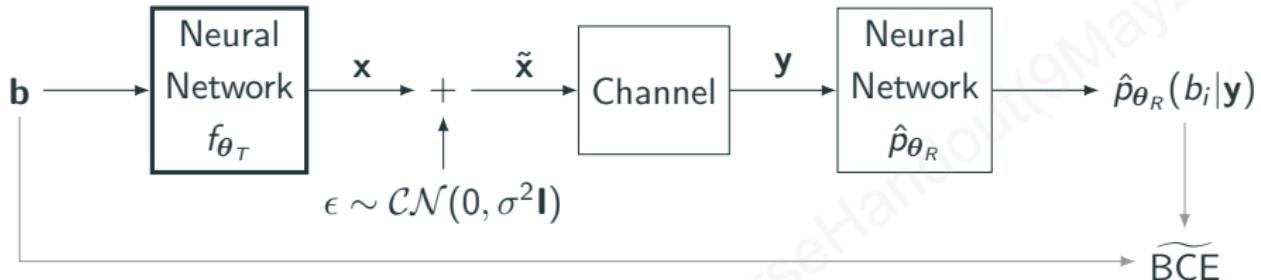
where we used $\nabla_{\mathbf{x}} p(\tilde{\mathbf{x}} | \mathbf{x}) = p(\tilde{\mathbf{x}} | \mathbf{x}) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x}))$

Training over the channel with reinforcement learning



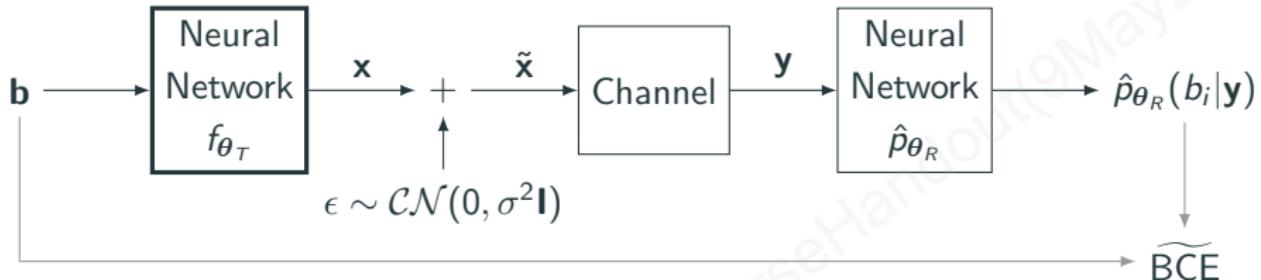
$$\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) = \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R)]$$

Training over the channel with reinforcement learning



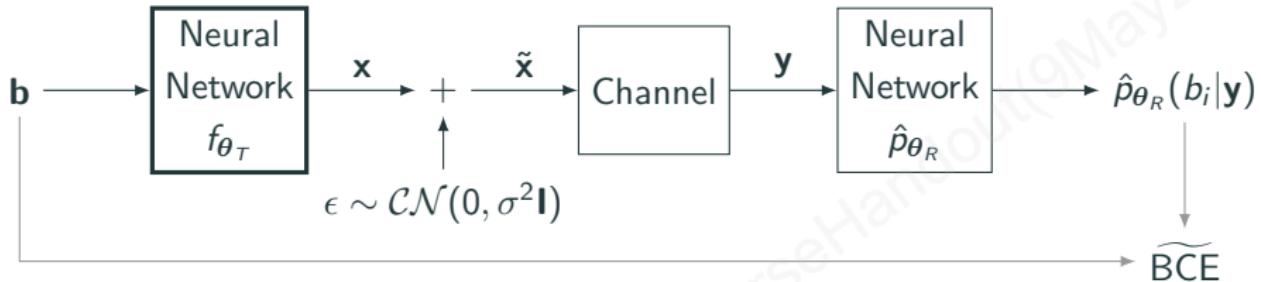
$$\begin{aligned}\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) &= \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R)] \\ &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R)]\end{aligned}$$

Training over the channel with reinforcement learning



$$\begin{aligned}\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) &= \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R)] \\ &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R)] \\ &\approx \frac{1}{S} \sum_{s=1}^S \nabla_{\theta_T} \left(\log p(\tilde{\mathbf{x}}^{(s)} | \mathbf{x} = f_{\theta_T}(\mathbf{b}^{(s)})) \right) \ell(\mathbf{b}^{(s)}, \mathbf{y}^{(s)}, \theta_R)\end{aligned}$$

Training over the channel with reinforcement learning



$$\begin{aligned}\nabla_{\theta_T} \widetilde{\text{BCE}}(\theta_T, \theta_R) &= \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\nabla_{\theta_T} (f_{\theta_T}(\mathbf{b})) \nabla_{\mathbf{x}} (\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b}))) \ell(\mathbf{b}, \mathbf{y}, \theta_R)] \\ &= \nabla_{\theta_T} \mathbb{E}_{\mathbf{b}, \tilde{\mathbf{x}}, \mathbf{y}} [\log p(\tilde{\mathbf{x}} | \mathbf{x} = f_{\theta_T}(\mathbf{b})) \ell(\mathbf{b}, \mathbf{y}, \theta_R)] \\ &\approx \frac{1}{S} \sum_{s=1}^S \nabla_{\theta_T} \left(\log p(\tilde{\mathbf{x}}^{(s)} | \mathbf{x} = f_{\theta_T}(\mathbf{b}^{(s)})) \right) \ell(\mathbf{b}^{(s)}, \mathbf{y}^{(s)}, \theta_R)\end{aligned}$$

No need to differentiate the channel! \rightarrow We can train over a “black box”!

Training over the channel with reinforcement learning

Theorem: Under some mild conditions given in [AH19], one has

$$\lim_{\sigma \rightarrow 0} \nabla_{\theta_T} \widetilde{BCE}(\theta_T, \theta_R) = \nabla_{\theta_T} BCE(\theta_T, \theta_R)$$

Training over the channel with reinforcement learning

Theorem: Under some mild conditions given in [AH19], one has

$$\lim_{\sigma \rightarrow 0} \nabla_{\theta_T} \widetilde{BCE}(\theta_T, \theta_R) = \nabla_{\theta_T} BCE(\theta_T, \theta_R)$$

- This means that for σ sufficiently small, $\nabla_{\theta_T} \widetilde{BCE}(\theta_T, \theta_R)$ is a good estimate of $\nabla_{\theta_T} BCE(\theta_T, \theta_R)$

Training over the channel with reinforcement learning

Theorem: Under some mild conditions given in [AH19], one has

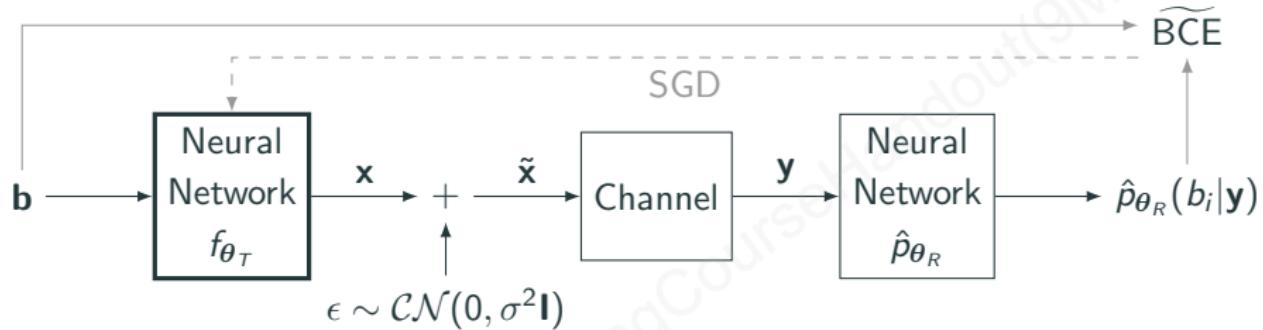
$$\lim_{\sigma \rightarrow 0} \nabla_{\theta_T} \widetilde{BCE}(\theta_T, \theta_R) = \nabla_{\theta_T} BCE(\theta_T, \theta_R)$$

- This means that for σ sufficiently small, $\nabla_{\theta_T} \widetilde{BCE}(\theta_T, \theta_R)$ is a good estimate of $\nabla_{\theta_T} BCE(\theta_T, \theta_R)$
- In practice, σ controls the tradeoff between the accuracy of the gradient estimate and its variance

Training over the channel with reinforcement learning

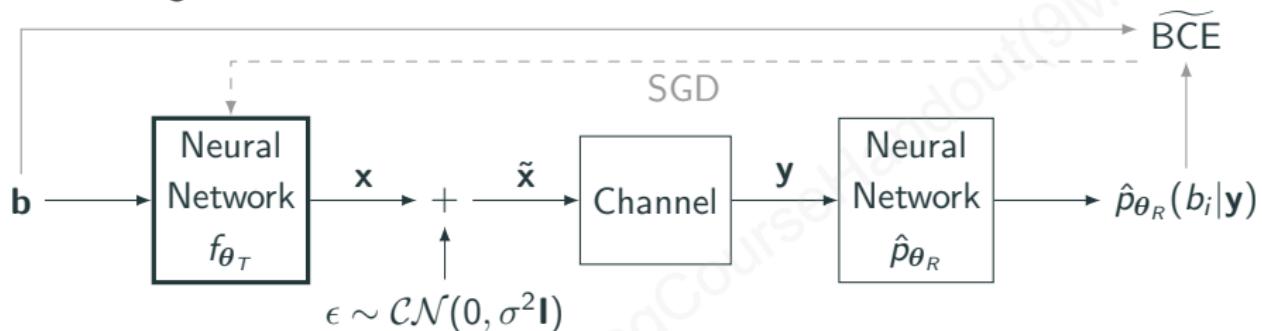
Training over the channel with reinforcement learning

Training of the transmitter:

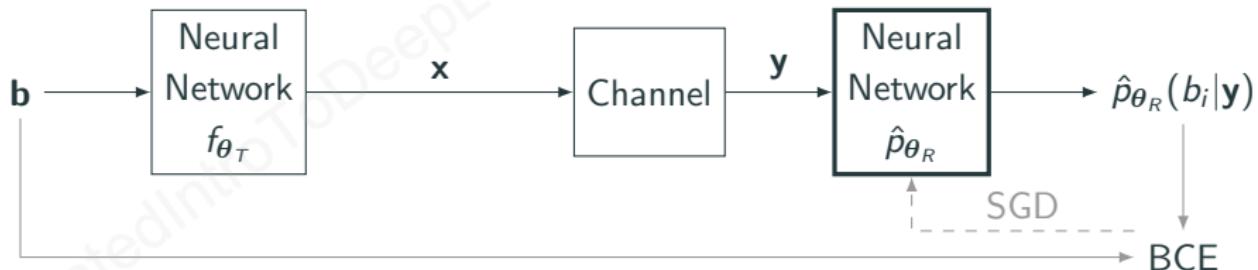


Training over the channel with reinforcement learning

Training of the transmitter:

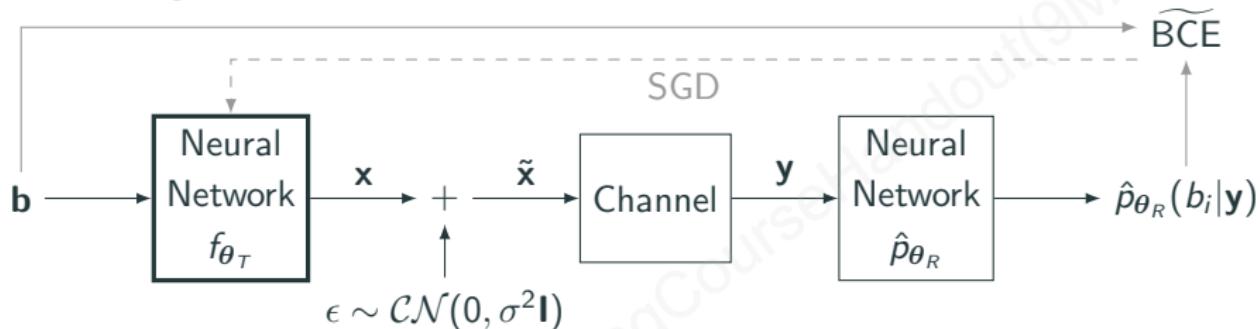


Training of the receiver:

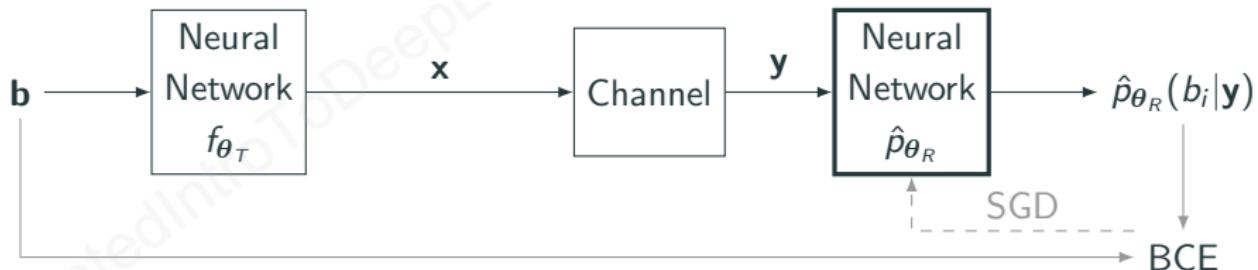


Training over the channel with reinforcement learning

Training of the transmitter:



Training of the receiver:



Alternate between training of the transmitter and of the receiver

Training over the channel with reinforcement learning

Benefits:

Training over the channel with reinforcement learning

Benefits:

- Enables training over the actual channel without any need for a model

Training over the channel with reinforcement learning

Benefits:

- Enables training over the actual channel without any need for a model
- Enables joint optimization of the transmitter and receiver

Training over the channel with reinforcement learning

Benefits:

- Enables training over the actual channel without any need for a model
- Enables joint optimization of the transmitter and receiver

Drawbacks:

Training over the channel with reinforcement learning

Benefits:

- Enables training over the actual channel without any need for a model
- Enables joint optimization of the transmitter and receiver

Drawbacks:

- Training of the transmitter can require a lot of samples

Training over the channel with reinforcement learning

Benefits:

- Enables training over the actual channel without any need for a model
- Enables joint optimization of the transmitter and receiver

Drawbacks:

- Training of the transmitter can require a lot of samples
- Scalability to a high number of channel inputs is an open problem

Autoencoder example notebook



Autoencoder

Lessons learned

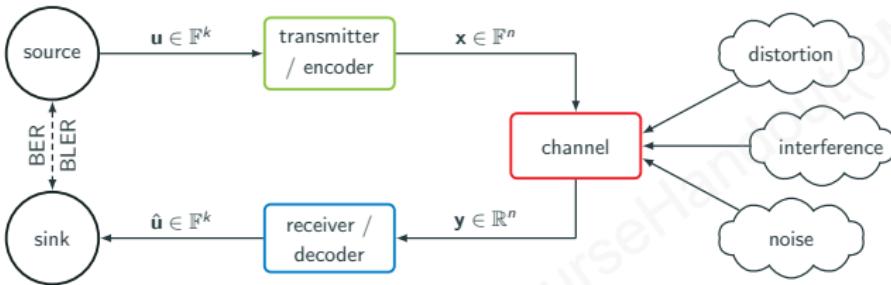
Overcoming the missing channel gradient problem



- End-to-end learning with conventional backpropagation is challenging when no channel model is available or when the channel model is not differentiable
- Three methods are known for overcoming this issue:
 1. Receiver fine-tuning: Train the end-to-end system over a channel model, then fine-tune the receiver over the actual channel
 2. GAN: Learn a channel model as a GAN, and then use it for end-to-end learning
 3. RL: Jointly train the transmitter using reinforcement learning and the receiver using conventional backpropagation

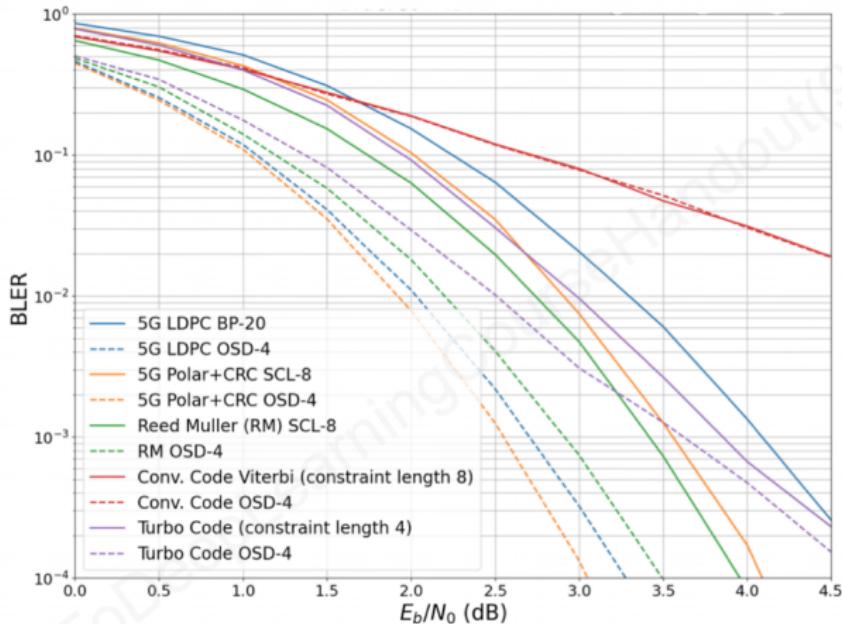
Deep Learning for Channel Coding

Introduction: Channel coding



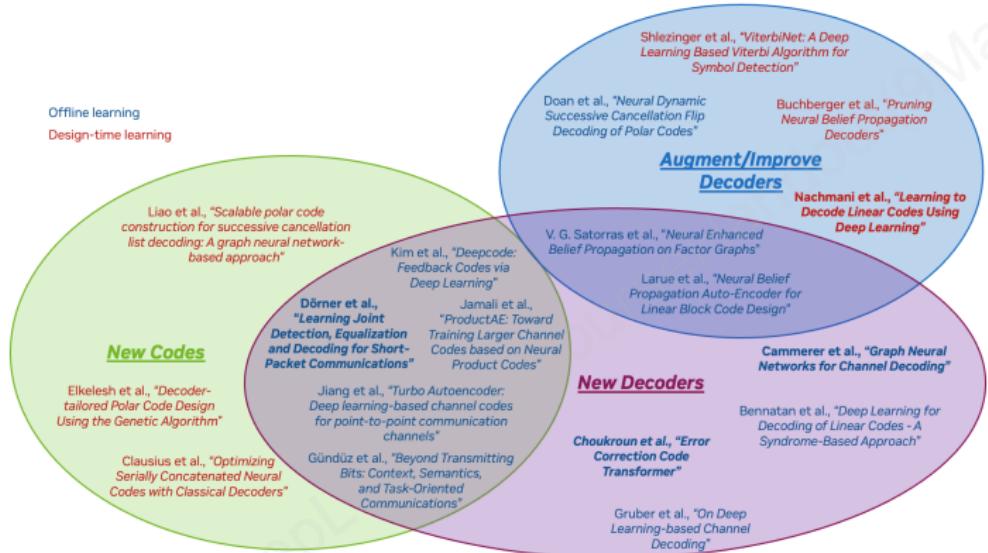
- Goal: reliable communication over a noisy channel
 - Bit error probability $P_e \xrightarrow{n \rightarrow \infty} 0$
- Transmit k information bits within a codeword of length $n > k$
 - Add redundancy, i.e., transmit $n - k$ redundant bits (rate $r = \frac{k}{n}$)
- Requirements today:
 - Reliability (optics: $P_e < 10^{-15}$; wireless $P_e < 10^{-6}$)
 - Low-complexity / high throughput (optics: $> 100\text{Gbit/s}$)
 - Energy-efficiency (significant amount of power consumption in mobile chips are for forward error correction (FEC))

Code vs. decoding performance



- Results for $n = 128, r = 0.5$
- Ordered statistics decoding (OSD) approximates ML performance
- Current (practical) decoders are sub-optimal

Research Landscape



Offline learning

- Replacing *classical* signal processing by pre-trained NNs
- Limits: inference complexity

Online learning

- Adaptivity through weight updates
- Limits: training complexity (& training data)

Design-time learning

- Use ML tools to find new codes/decoders
- Limits: training complexity

Learning with rules

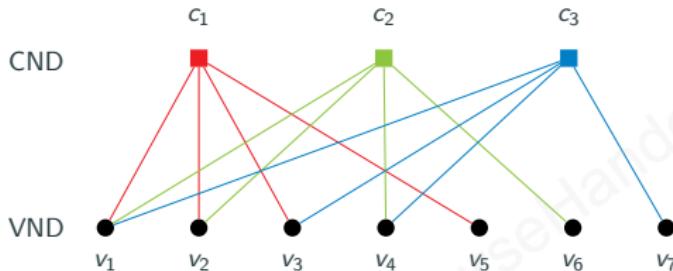


Man-made signals

- We often know the signal structure (i.e., the rules)
- A lot of data is required to learn what does *not* depend on each other

Figure taken from <https://www.chess.com/article/view/the-man-who-built-the-chess-machine>

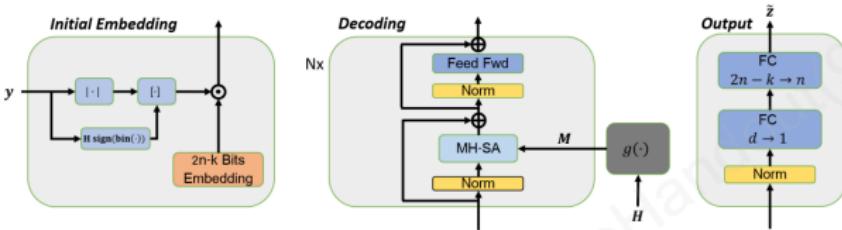
Curse of dimensionality



- Why don't we just *learn to decode* from the scratch?
- Big problem of deep learning-based decoding:
 - For a short code with $n = 100$ and
 $r = 0.5 \Rightarrow 2^{50} = 1125899906842624$ codewords exist
- Conclusion: training needs to infer the full codebook from a couple of codewords [GCHt17]
- The limiting factor is the training, not (necessarily) the inference

We know a suitable decoding graph since many years [Gal62, Tan81]

Error Correction Code Transformer [CW22]



- Pre-processing: $\tilde{\mathbf{y}} = [|\mathbf{y}|, s(\mathbf{y})]$ where $s(\mathbf{y})$ is the (binary) syndrome
- Input dimensionality $n + (n - k) = 2n - k$
- Trainable input embedding (projecting each value to d -dimensional space)

$$\phi_i = \begin{cases} |\mathbf{y}_i| \mathbf{w}_i & \text{if } i < n \\ [1 - 2(s(\mathbf{y}))]_{i-n} \mathbf{w}_i & \text{otherwise} \end{cases}$$

where $\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=0}^{2n-k-1}$ is a (trainable) embedding

- h multi-headed attention layers

Figure taken from <https://aps.arxiv.org/pdf/2203.14966.pdf>

Error Correction Code Transformer [CW22]

- *Code aware self-attention* [CW22]

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T + g(\mathbf{H})}{\sqrt{d}} \right) \cdot \mathbf{V}$$

where $g(\mathbf{H}) : \{0, 1\}^{(n-k) \times k} \rightarrow \{-\infty, 0\}^{(2n-k) \times (2n-k)}$ is a code depending mask

$$g(\mathbf{H}) = \begin{cases} -\infty & \text{if no direct connection between nodes exists} \\ o & \text{otherwise} \end{cases}$$

- Example for the (7,4) Hamming code

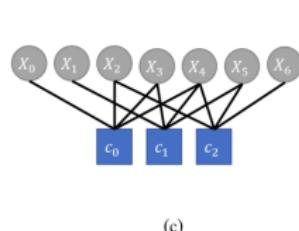
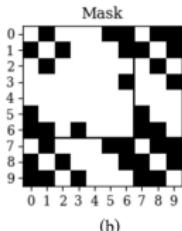
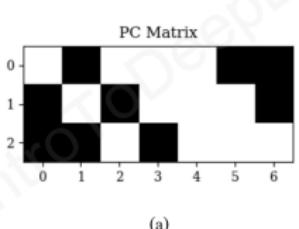
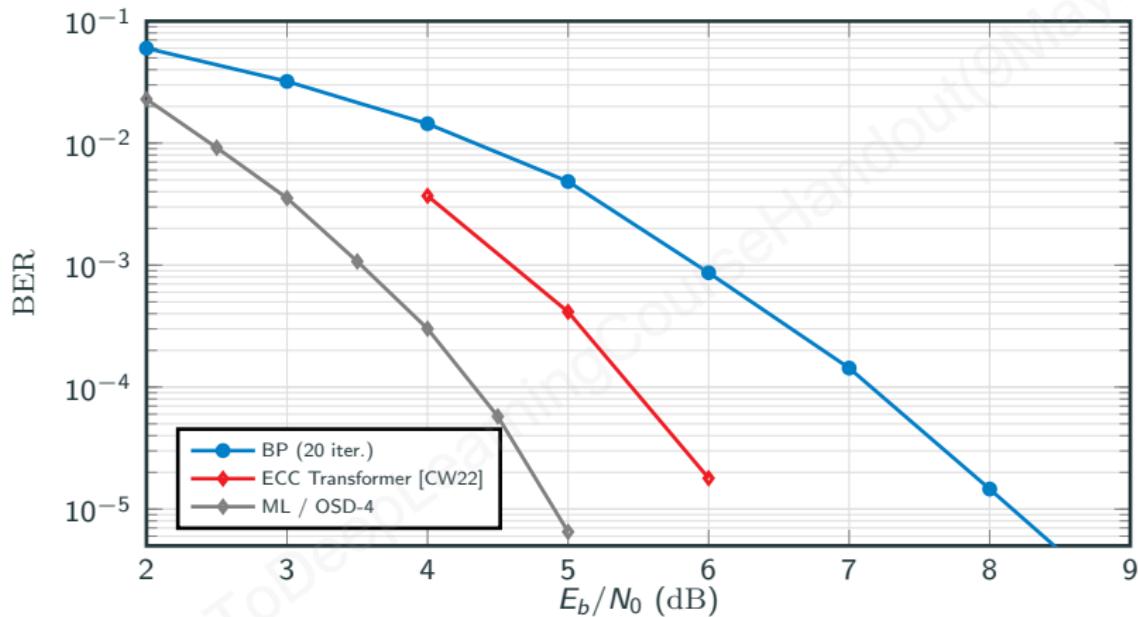


Figure taken from <https://arxiv.org/pdf/2203.14966.pdf>

ECC Transformer - Results BCH (63,45)

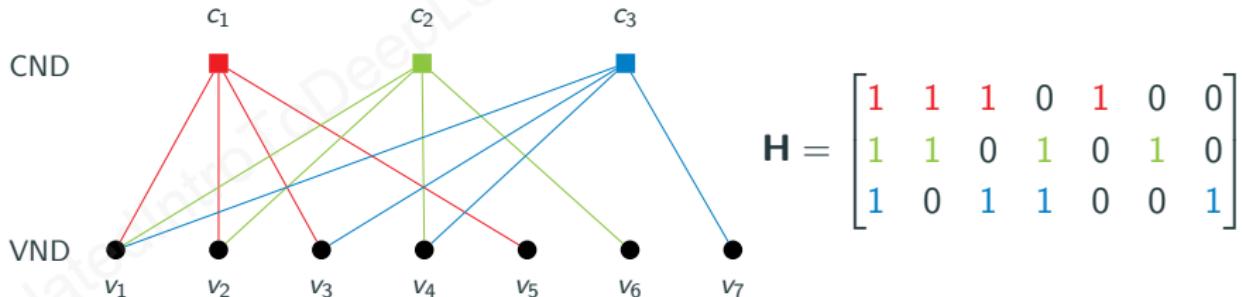


- Architecture scales to medium code lengths
- Complexity [CW22]: $\mathcal{O}(N(d^2(2n-k) + hd))$ where h accounts for the fixed number of computations in the self-attention

Neural Belief Propagation

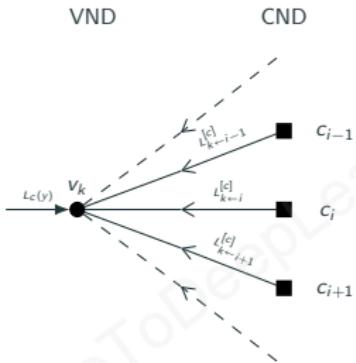
Background: Graph-based codes

- Any linear code can be described by a *bipartite* graph, called *Tanner graph* [Tan81]
- Two sets of nodes for belief propagation (BP) decoding:
 - Variable node (VN): each of the n bit positions of a codeword are represented by a VN
 - Check node (CN): each of the $n - k$ parity checks are represented by a *constraint* node, called CN
- Each row in \mathbf{H} defines a single CN; each 1 defines an *edge* to an VN
- Example: Tanner graph of the (7,4) Hamming code



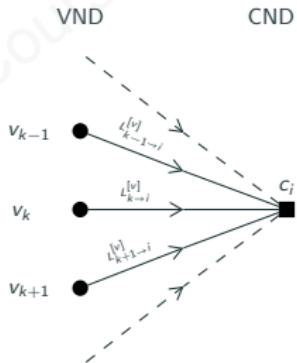
Node update functions

- Initialize all messages with 0
- Iteratively update of messages:
 - VN k to CN i message $L_{k \rightarrow i}^{[v]}$
 - CN i to VN k message $L_{k \leftarrow i}^{[c]}$
- Stop after fixed number of iterations N_{iter}



VND-Update

$$L_{k \rightarrow i}^{[v]} = L_c(y) + \sum_{i' \in \mathcal{N}(k) \setminus \{i\}} L_{k \leftarrow i'}^{[c]}$$

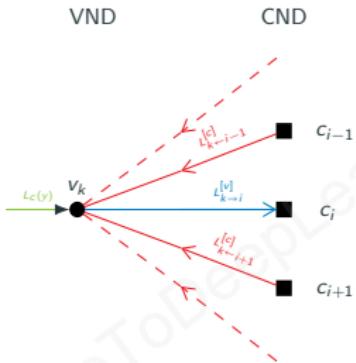


CND-Update

$$L_{k \leftarrow i}^{[c]} = 2 \tanh^{-1} \left(\prod_{k' \in \mathcal{M}(i) \setminus \{k\}} \tanh \left(\frac{L_{k' \rightarrow i}^{[v]}}{2} \right) \right)$$

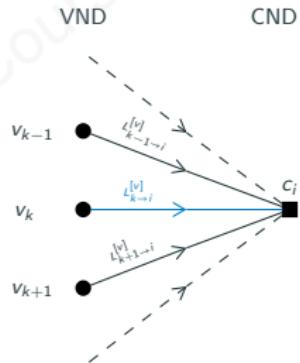
Node update functions

- Initialize all messages with 0
- Iteratively update of messages:
 - VN k to CN i message $L_{k \rightarrow i}^{[v]}$
 - CN i to VN k message $L_{k \leftarrow i}^{[c]}$
- Stop after fixed number of iterations N_{iter}



VND-Update

$$L_{k \rightarrow i}^{[v]} = L_c(y) + \sum_{i' \in \mathcal{N}(k) \setminus \{i\}} L_{k \leftarrow i'}^{[c]}$$

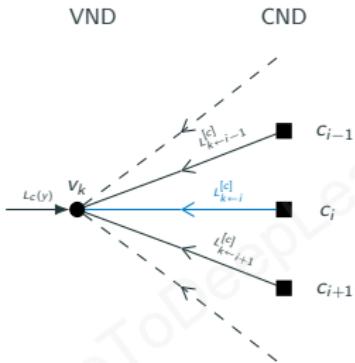


CND-Update

$$L_{k \leftarrow i}^{[c]} = 2 \tanh^{-1} \left(\prod_{k' \in \mathcal{M}(i) \setminus \{k\}} \tanh \left(\frac{L_{k' \rightarrow i}^{[v]}}{2} \right) \right)$$

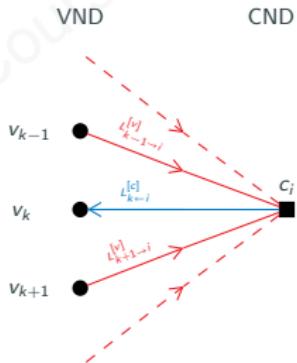
Node update functions

- Initialize all messages with 0
- Iteratively update of messages:
 - VN k to CN i message $L_{k \rightarrow i}^{[v]}$
 - CN i to VN k message $L_{k \leftarrow i}^{[c]}$
- Stop after fixed number of iterations N_{iter}



VND-Update

$$L_{k \rightarrow i}^{[v]} = L_c(y) + \sum_{i' \in \mathcal{N}(k) \setminus \{i\}} L_{k \leftarrow i'}^{[c]}$$

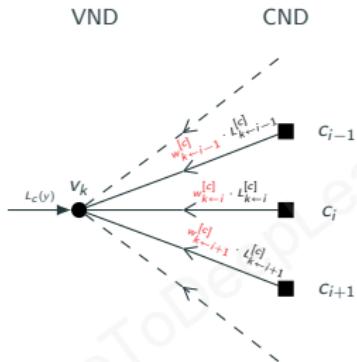


CND-Update

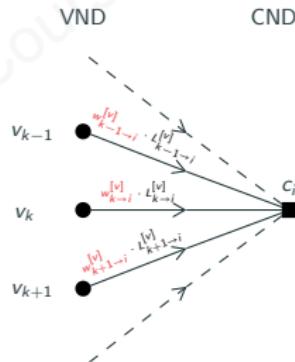
$$L_{k \leftarrow i}^{[c]} = 2 \tanh^{-1} \left(\prod_{k' \in \mathcal{M}(i) \setminus \{k\}} \tanh \left(\frac{L_{k' \rightarrow i}^{[v]}}{2} \right) \right)$$

Neural belief propagation

- The BP-algorithm has no degree of freedom, i.e., nothing to train
→ We need to add additional degrees of freedom
- Idea: *soft Tanner graph* [NBB16], i.e., assign trainable weights to each message
- Each message on the graph is scaled by a trainable weight $w_{i \rightarrow j}$



VND-Update



CND-Update

$$L_{k \leftarrow i}^{[v]} = L_c(y) + \sum_{i' \in \mathcal{N}(k) \setminus \{i\}} w_{k \leftarrow i'}^{[c]} \cdot L_{k \leftarrow i'}^{[c]}$$

$$L_{k \leftarrow i}^{[c]} = 2 \tanh^{-1} \left(\prod_{k' \in \mathcal{M}(i) \setminus \{k\}} \tanh \left(\frac{w_{k' \leftarrow i}^{[v]} \cdot L_{k' \leftarrow i}^{[v]}}{2} \right) \right)$$

Iterative loop unrolling

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]
 - Loop unrolling
 - Interpret each VND/CND update as one layer (stacked in time)
 - Use shared weights for all VND/CND layers

Decoding Loop

```
for it in range(0, Niter):  
    x=VNupdate(y, Lch)  
    y=CNupdate(x)
```

Unrolled Loop

$it = 0$	
$x=VNupdate(y, L_{ch})$	\leftarrow Layer 0
$y=CNupdate(x)$	\leftarrow Layer 1
$it = 1$	
$x=VNupdate(y, L_{ch})$	\leftarrow Layer 2
$y=CNupdate(x)$	\leftarrow Layer 3
\dots	
$it = N_{iter} - 1$	
$x=VNupdate(y, L_{ch})$	\leftarrow Layer 2it
$y=CNupdate(x)$	\leftarrow Layer 2it + 1

Iterative loop unrolling (2)

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]

Iteration 1

v_1



v_2



v_3



v_4



v_5



v_6



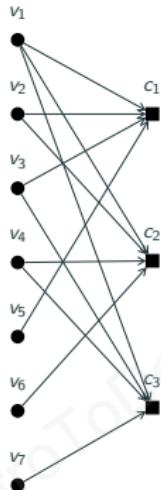
v_7



Iterative loop unrolling (2)

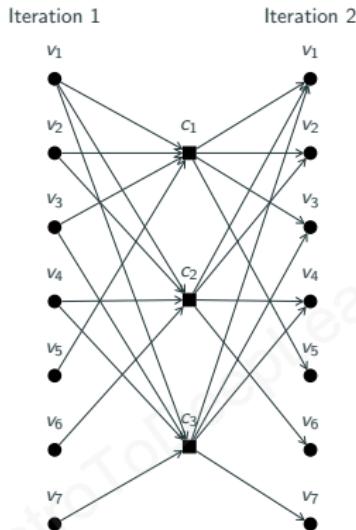
- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]

Iteration 1



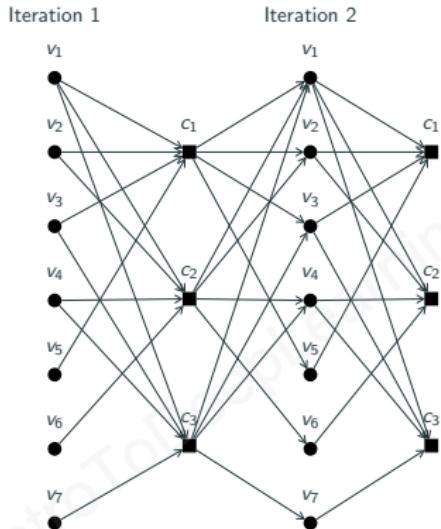
Iterative loop unrolling (2)

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]



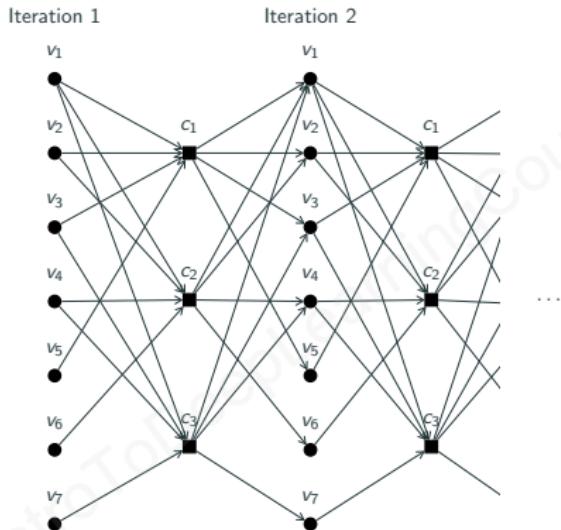
Iterative loop unrolling (2)

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]



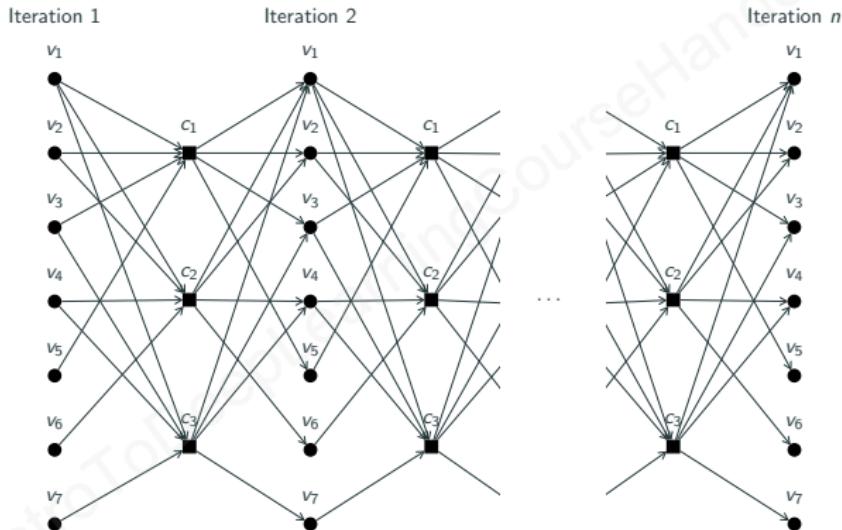
Iterative loop unrolling (2)

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]



Iterative loop unrolling (2)

- Problem: *How to describe the NN-BP by a feedforward NN?*
- Solution: *Deep unfolding* [HRW14]

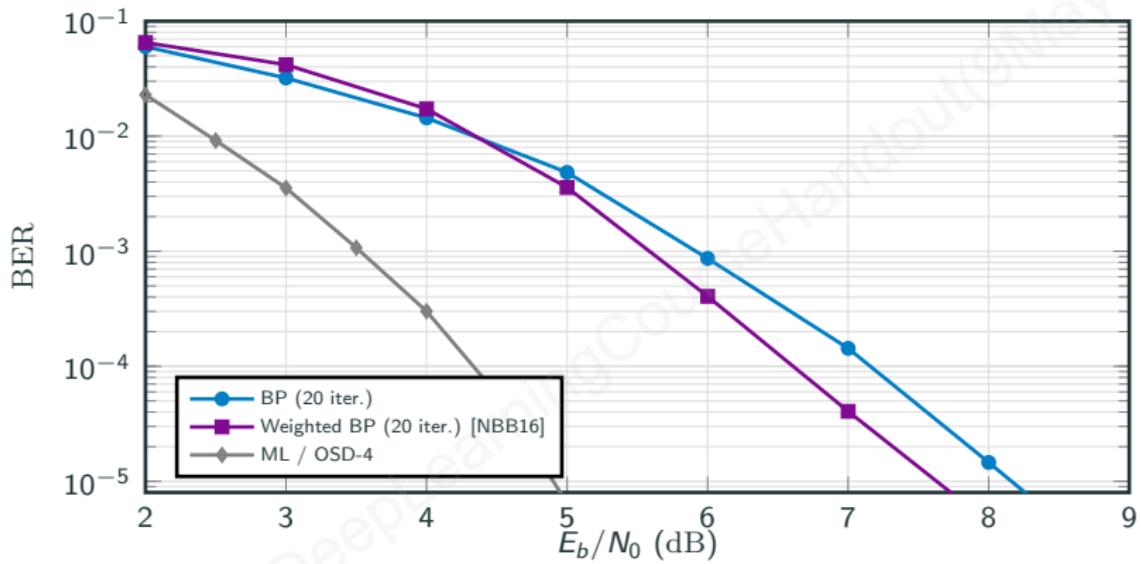


→ Provides explicit (feedforward) NN structure

Implementation details and suitable loss

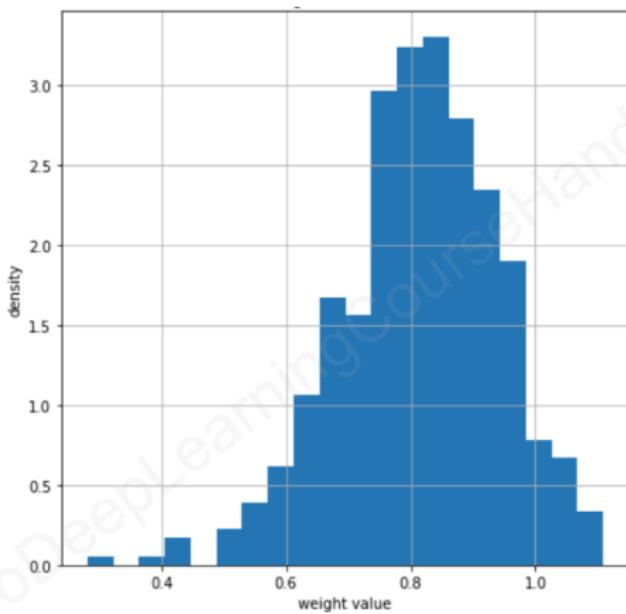
- Feedforward structure with two kinds of layers:
 - CN-layer: Only incoming edges from previous VN-layer
 - VN-layer: Channel observation L_c as additional input
- Final marginalization $L(\hat{x}_k) = L_c(y) + \sum_{i \in \mathcal{N}(k)} L_{k \leftarrow i}^{[c]}$
- Classification problem: each bit x_i is either ‘0’ or ‘1’
 - Binary cross-entropy loss
 - *Multi-loss* [NBB16]: calculate and average cross-entropy after each iteration to avoid vanishing gradient problems
- Gradient clipping required for numerical stability
- VN update is linear → apply weights only to outgoing VN messages

BER performance for $(n, k) = (63, 45)$ BCH



- Remark: BP decoding for BCH codes is highly suboptimal (cycles!)
- Improved code constructions lead to better code constructions (e.g., LDPC with progressive edge growth (PEG)-algorithm)
- Typically, the gain vanishes with a larger number of BP iterations

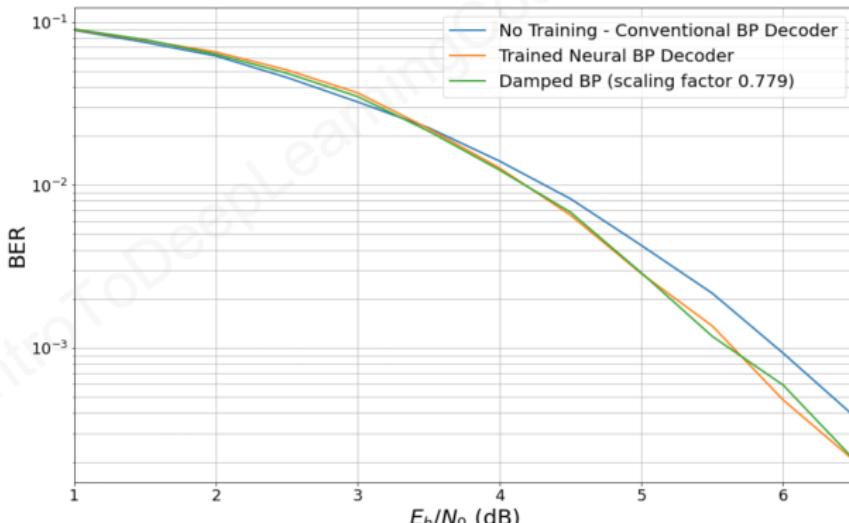
Weight distribution for $(n, k) = (63, 45)$ BCH



- We observe that the weights tend to be smaller after training
→ Edges are (partly) removed from the Tanner graph

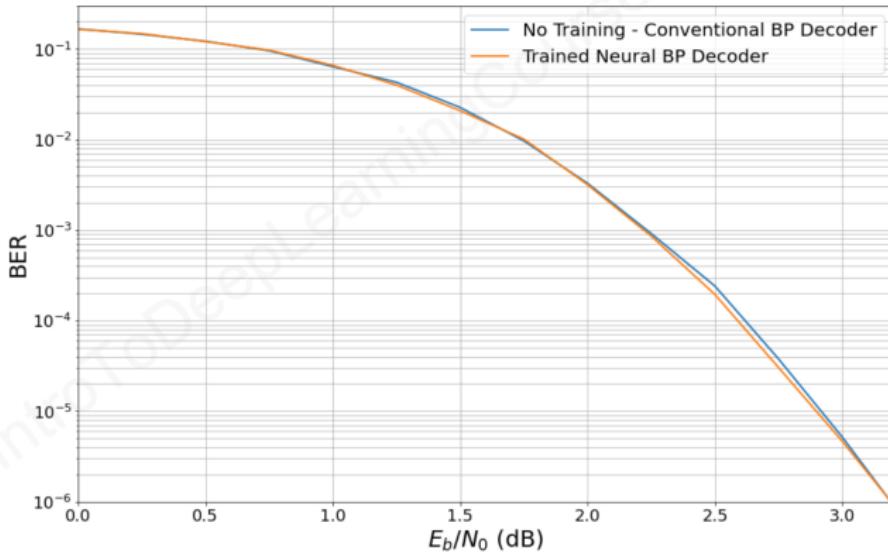
Damped BP for $(n, k) = (63, 45)$ BCH

- Scaling messages can improve performance of BP [Pre05, YFW05]
 - Mostly relevant for short codes
→ Graphs with short cycles
- Basic idea: scale all messages with the same scalar α
- α can be *learned* via SGD [LHP18]



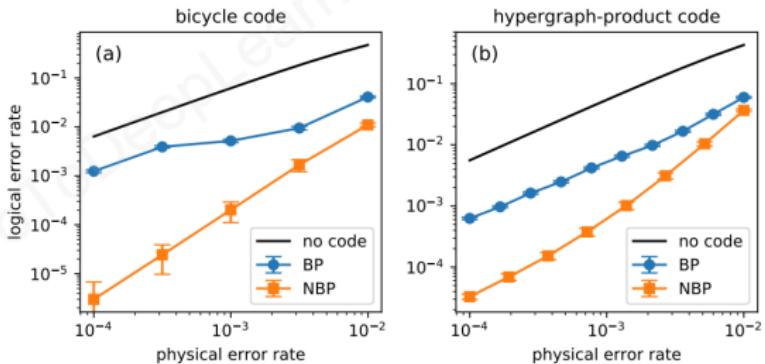
Learning the 5G LDPC code

- BER results for the 5G NR LDPC code with $n = 800$ and $r = 0.5$
- Number of decoding iterations $N_{\text{it}} = 10$
- Reminder: code is well-designed, short cycles are optimized



Further Applications

- Neural offset corrected min-sum decoding [LG17]
<https://arxiv.org/abs/1701.05931>
 - Theoretical investigation <https://arxiv.org/abs/2205.10684>
- Use neural BP to prune least important edges [BHP⁺20]
<https://arxiv.org/abs/2001.07464>
- Apply neural BP to quantum error correction code (QECC) [LP19]⁶
<https://arxiv.org/pdf/1811.07835.pdf>



⁶Fig. taken from [LP19]



Weighted BP Decoding

Lessons learned – Deep Unfolding

Neural Belief Propagation



- *Loop unrolling*: iterative (graph-based) algorithms can be interpreted as *explicit* neural network
- Additional weights / degrees of freedom
 - Can be optimized with SGD
 - Avoid learning the already known graph (man-made signal)
- Outlook:
 - Apply idea to other iterative algorithms: MIMO-detectors,...
 - Decoding of Polar codes (high density → many loops)
 - Theoretical analysis: under which condition does it improve?

Graph Neural Networks for Channel Decoding

Resources

- CS224W: Machine Learning with Graphs, Stanford
<https://web.stanford.edu/class/cs224w/>
- Introduction to Graph Neural Networks (GNN)
<https://www.youtube.com/watch?v=8owQBFAHw7E>
- A Gentle Introduction to Graph Neural Networks
<https://distill.pub/2021/gnn-intro>
- Understanding Convolutions on Graphs
<https://distill.pub/2021/understanding-gnns>
- Geometric Deep Learning: The Erlangen Programme of ML
<https://www.youtube.com/watch?v=w6Pw4M0zMuo>
- Must-read papers on GNN
<https://github.com/thunlp/GNNPapers>

Graphs are everywhere



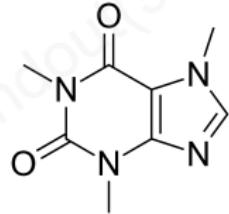
Credit: RATP

Transportation networks



Credit: Wikipedia

Social networks



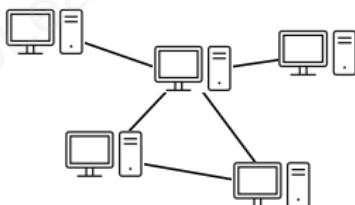
Credit: Wikipedia

Molecular networks

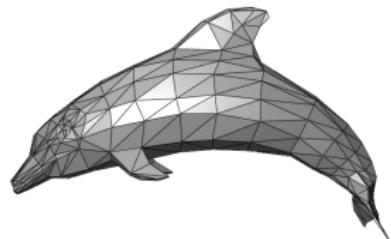


Credit: Connected Papers

Citation networks



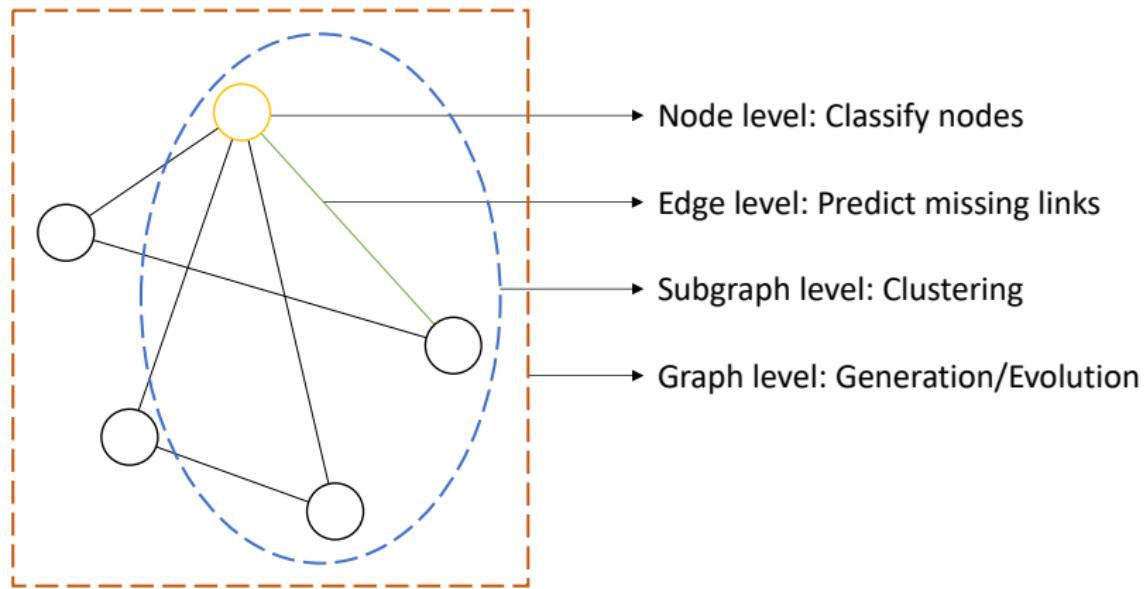
Computer networks



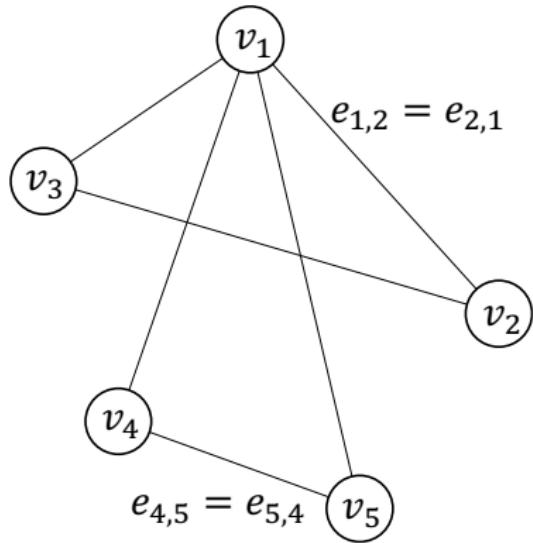
Credit: Wikipedia

Polygon meshes

Types of tasks on graphs

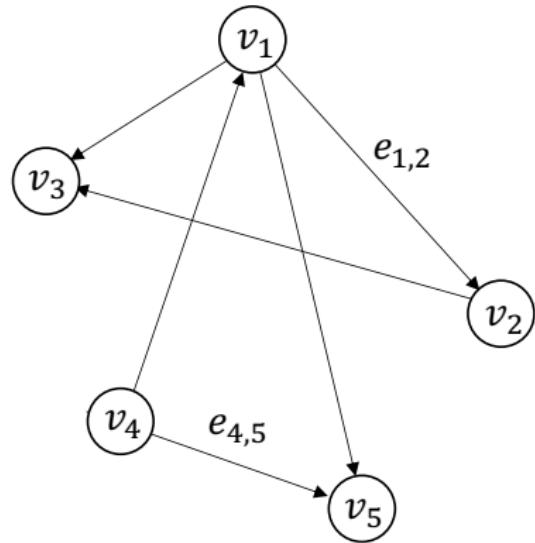


Undirected vs directed graphs



Undirected graph:

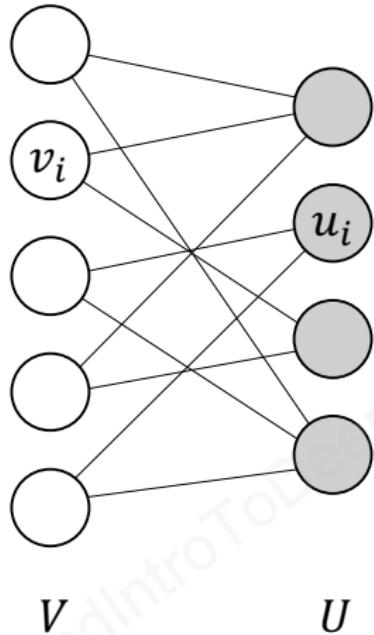
- Links are reciprocal/symmetric
- Friendships/Collaborations



Directed graph:

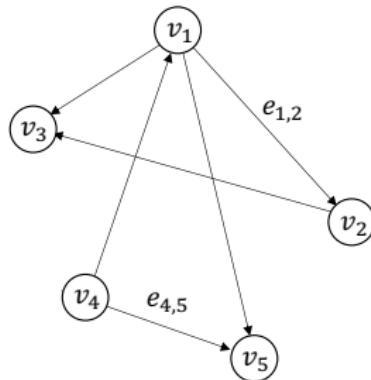
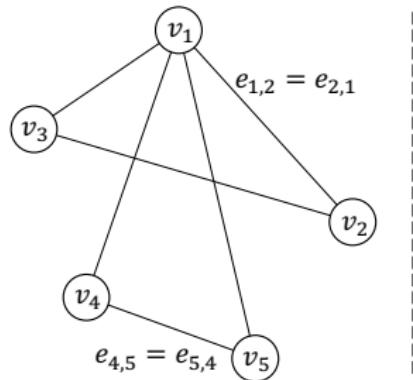
- Links are directed arcs
- Followers/Phone calls

Bipartite graphs



- Two disjoint sets of nodes V and U
- Every edge connects a node from V to a node from U
- Examples:
 - Authors to papers
 - Actors to movies
 - Users to movies
 - Variable to check nodes

Adjacency matrix

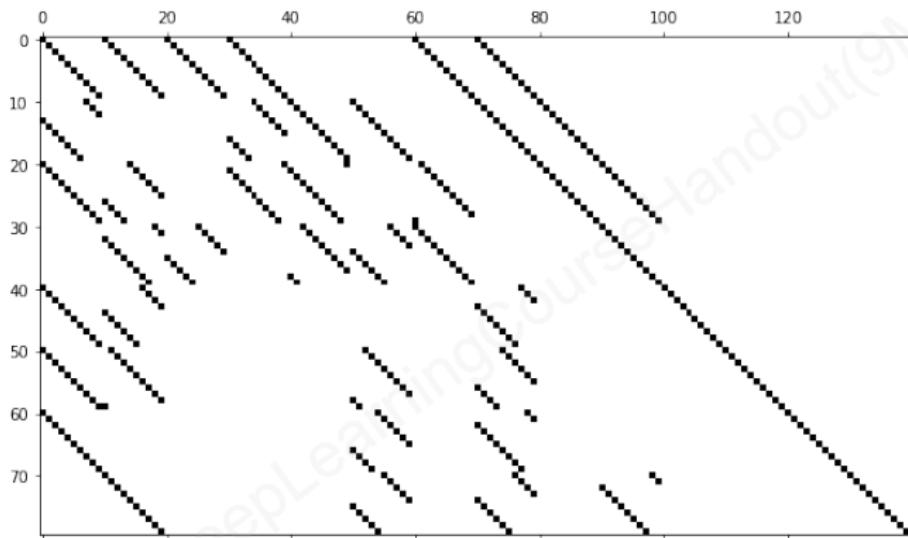


$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

For a directed graph, the adjacency matrix is may not be symmetric

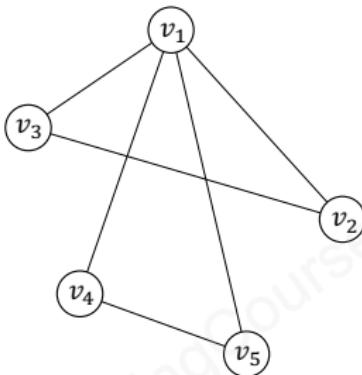
Most adjacency matrices are sparse



Most graphs we find in nature have sparse adjacency matrices, e.g.:

- Social networks: #friends \ll #members
- Citation networks: #citations \ll #papers

Other graph representations



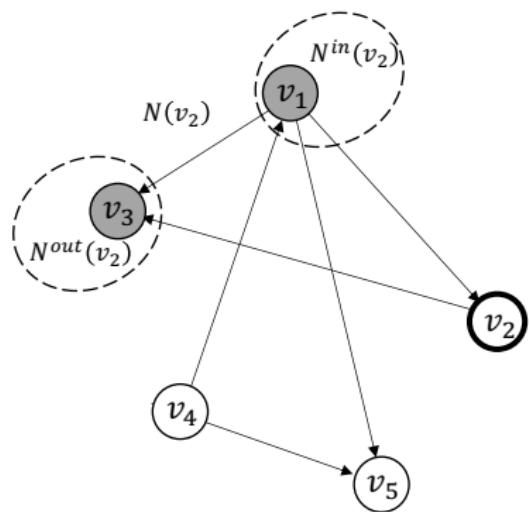
List of edges:

- $[(1, 2), (1, 3), (1, 4), \dots]$
- Widely used, but not practical

Adjacency list:

- 1: [2, 3, 4, 5]
2: [1, 3]
3: ...
- Easily retrieve all neighbors of a given node
- Great for large & sparse graphs

Neighborhood and degree of a node



Undirected graph

$$N(v_i) = \{v_j | e_{i,j} \in E\} = \{v_j | e_{j,i} \in E\}$$

$$d(v_i) = |N(v_i)| = \sum_{j=1}^N \mathbf{A}_{i,j} = \sum_{j=1}^N \mathbf{A}_{j,i}$$

Directed graph

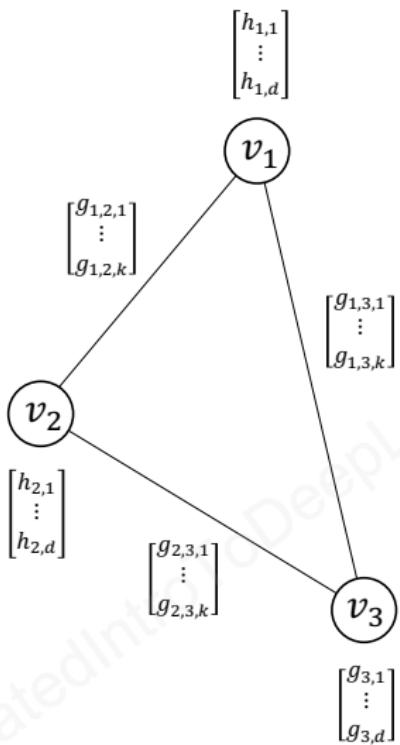
$$N^{\text{out}}(v_i) = \{v_j | e_{i,j} \in E\}$$

$$d^{\text{out}}(v_i) = |N(v_i)^{\text{out}}| = \sum_{j=1}^N \mathbf{A}_{i,j}$$

$$N^{\text{in}}(v_i) = \{v_j | e_{j,i} \in E\}$$

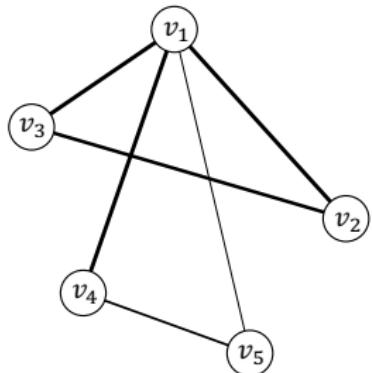
$$d^{\text{in}}(v_i) = |N(v_i)^{\text{in}}| = \sum_{j=1}^N \mathbf{A}_{j,i}$$

Node and edge attributes



- Nodes and edges can have *attributes* or *features*
- Attributes can be given or learned
- Node attribute examples:
 - Profile information/photo
 - Type
 - Weight or ranking
- Edge attribute examples:
 - Speed limit
 - Amount of traffic
 - Delay/length
 - Weight

Weighted graphs

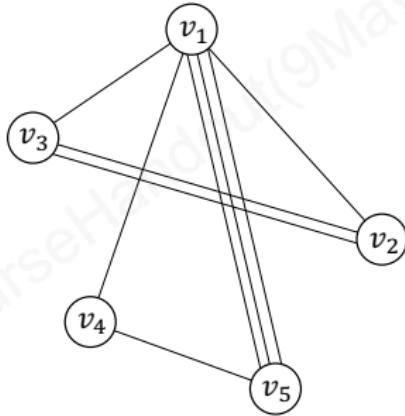
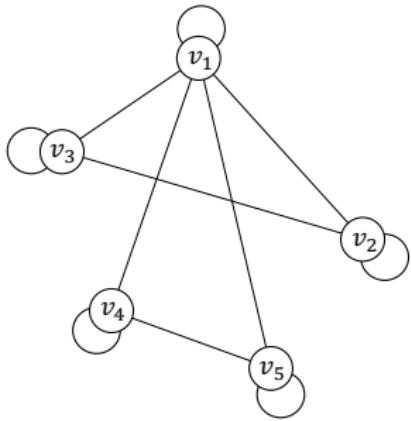


$$\mathbf{A} = \begin{pmatrix} 0 & 6 & 4 & 3 & 0.5 \\ 6 & 0 & 2 & 0 & 0 \\ 4 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- Adjacency matrix gives different weights to edges
- Special case of edge attributes
- Examples:
 - Fuel costs of certain routes
 - Traffic on a road

All combinations of (un)directed and (un)weighted graphs possible

Self-edges & Multi-graphs



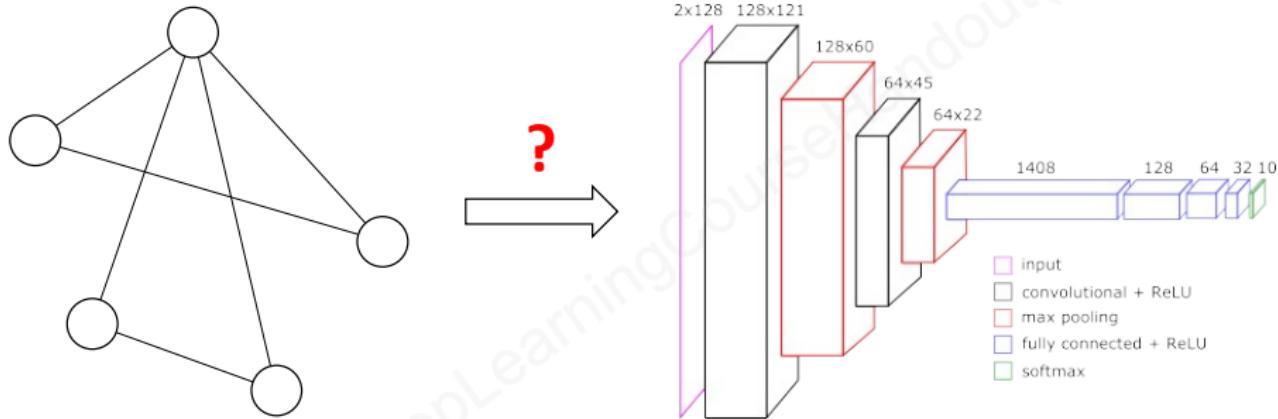
$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

E.g., websites

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 1 & 3 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 3 & 0 & 0 & 1 & 0 \end{pmatrix}$$

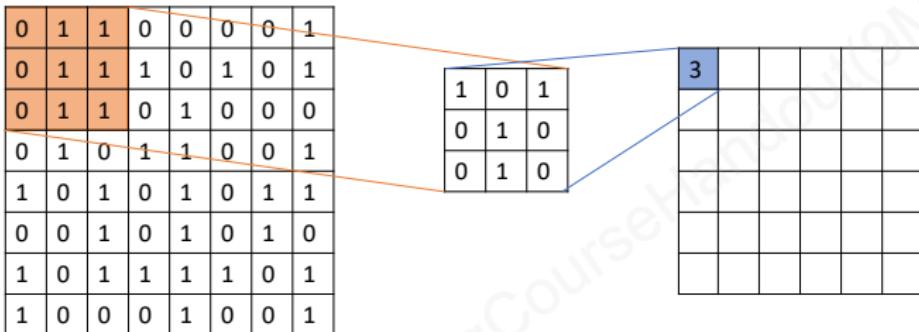
E.g., Different roads between points

How can we apply NNs to graphs?



- Arbitrary size & structure (no spatial locality like grids or sequences)
- No fixed ordering or reference point
- Possibly multimodal features and time-varying

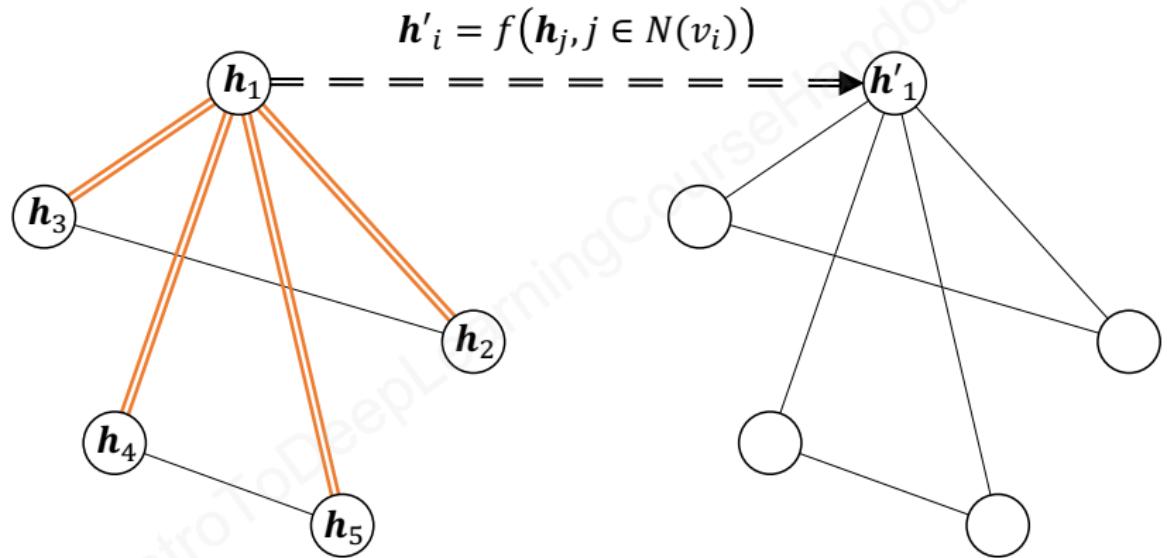
Reminder: Convolution on images



- Kernel defines “edge weights” from a pixel to its neighbors
- Pixel attributes are multiplied by weights and summed (aggregated)

How can we apply this idea to graphs with complex topological structure?

Convolution on graphs



Graph convolutional networks

- Undirected graph with self-loops, i.e., $\mathbf{A} = \mathbf{A}^T$ and $A_{ii} = 1$, for all i
- GCN update rule (or layer) [KW17]:

$$\mathbf{h}'_i = \sigma \left(\sum_{j \in N(i)} \frac{1}{\sqrt{|N(i)||N(j)|}} \mathbf{W} \mathbf{h}_j \right), \quad i = 1, \dots, N$$

where $\sigma()$ is a non-linearity that is applied element-wise and $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is a shared trainable linear transformation for the attribute of each node

- Can be conveniently written in matrix form:

$$\mathbf{H}' = \sigma \left(\mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{H} \right)$$

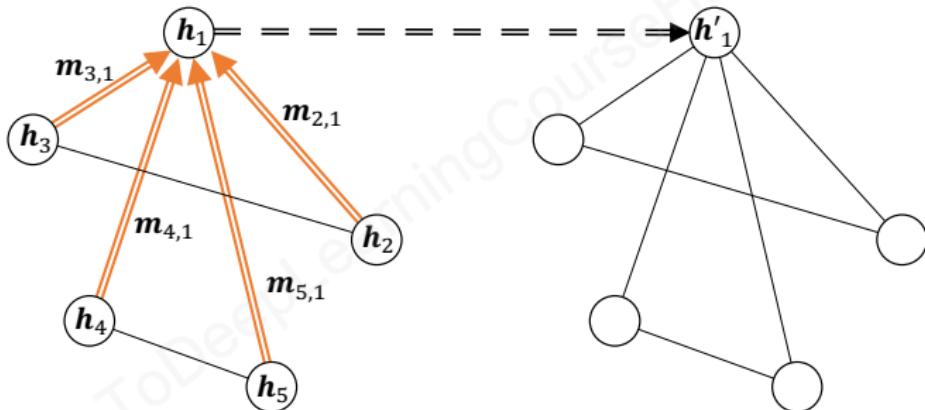
$$\mathbf{D} = \text{diag}(d(v_i), i = 1, \dots, N)$$

- Only scalar edge features possible (i.e., weighted adjacency matrix)

Message passing neural networks (MPNNs) [GSR⁺17]

$$\mathbf{m}_{i,j} = f_e(\mathbf{h}_i, \mathbf{h}_j, \mathbf{g}_{i,j})$$

$$\mathbf{h}'_i = f_v\left(\mathbf{h}_i, \sum_{j \in N(v_i)} \mathbf{m}_{i,j}, \mathbf{g}_i\right)$$



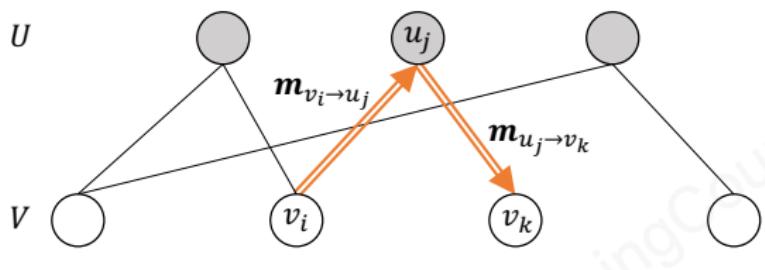
- Messages $\mathbf{m}_{i,j} \in \mathbb{R}^k$ are sent along edges
- Nodes aggregate incoming messages and update their embeddings \mathbf{h}_i
- Trainable node (\mathbf{g}_i) and edge ($\mathbf{g}_{i,j}$) attributes
- Message & readout functions f_e and f_v shared between nodes/edges

General GNN structure

$$\mathbf{h}'_i = f_v \left(\mathbf{h}_i, \bigoplus_{j \in N(v_i)} f_e(\mathbf{h}_j, \mathbf{h}_i, \mathbf{g}_{j,i}), \mathbf{g}_i \right)$$

- $\bigoplus_{j \in N(v_i)}$ is a permutation-invariant *aggregation function*:
 - E.g., sum, average, min, max, variance, standard deviation, etc.
 - Results of different aggregations can be concatenated
- f_e, f_v are trainable
 - Low complexity, e.g., small MLPs
 - Often operate on concatenated inputs
- MPNN is the most powerful graph neural network (GNN) layer
 - High complexity & memory consumption (due to messages)
 - Only applicable to small graphs
 - A lot of research on lower complexity variants (see next slide)

Message passing for bipartite graphs [SW21]



$$\mathbf{m}_{v_i \rightarrow u_j} = f_{v \rightarrow u}(\mathbf{h}_{v_i}, \mathbf{h}_{u_j}, \mathbf{g}_{v_i, u_j})$$

$$\mathbf{m}_{u_j \rightarrow v_i} = f_{u \rightarrow v}(\mathbf{h}_{u_j}, \mathbf{h}_{v_i}, \mathbf{g}_{u_j, v_i})$$

$$\mathbf{h}'_{u_j} = f_u\left(\mathbf{h}_{u_j}, \sum_{i \in N(u_j)} \mathbf{m}_{v_i \rightarrow u_j}, \mathbf{g}_{u_j}\right)$$

$$\mathbf{h}'_{v_i} = f_v\left(\mathbf{h}_{v_i}, \sum_{j \in N(v_i)} \mathbf{m}_{u_j \rightarrow v_i}, \mathbf{g}_{v_i}\right)$$

- Two types of shared message functions $f_{v \rightarrow u}$ and $f_{u \rightarrow v}$
- Two types of shared readout functions f_v and f_u
- Trainable node and (directed) edge attributes

GNN-based BP decoding

1. Initialize node attributes: $\mathbf{h}_v = \text{LLR} \times \mathbf{w}$, $\mathbf{h}_u = \mathbf{0}$
2. For $\ell = 0, \dots, L - 1$:

$$\mathbf{m}_{v \rightarrow u} = \text{MLP}_1([\mathbf{h}_v \| \mathbf{h}_u])$$

$$\mathbf{h}_u = \text{MLP}_2\left(\left[\mathbf{h}_u \middle\| \frac{1}{|N(u)|} \sum_{i \in N(u)} \mathbf{m}_{v_i \rightarrow u}\right]\right)$$

$$\mathbf{m}_{u \rightarrow v} = \text{MLP}_3([\mathbf{h}_u \| \mathbf{h}_v])$$

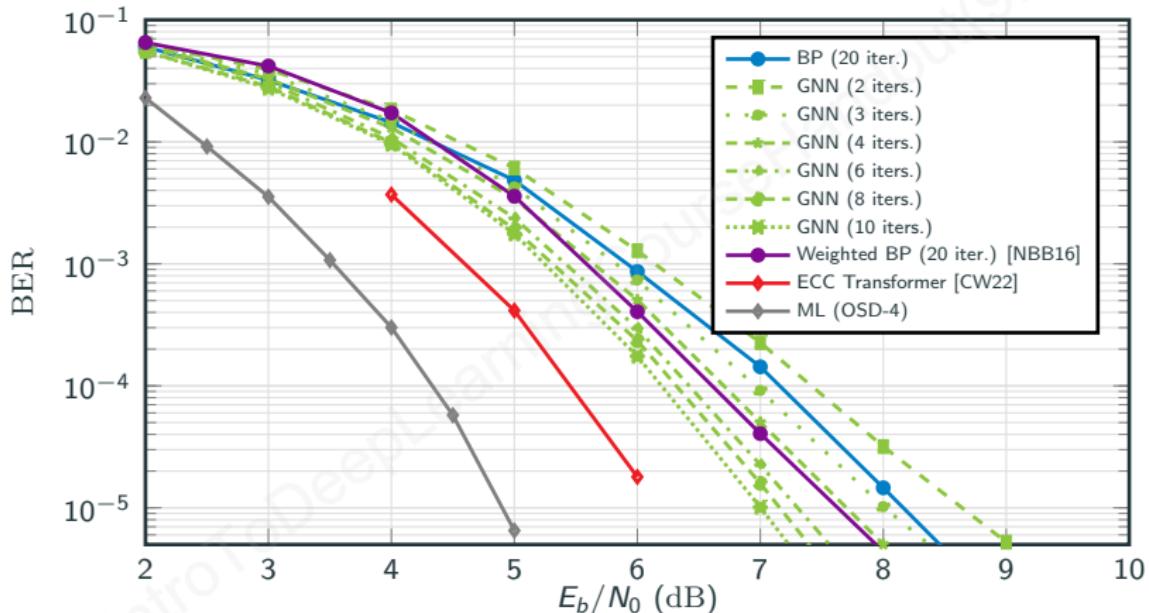
$$\mathbf{h}_v = \text{MLP}_4\left(\left[\mathbf{h}_v \middle\| \frac{1}{|N(v)|} \sum_{j \in N(v)} \mathbf{m}_{u_j \rightarrow v}\right]\right)$$

$$\text{LLR}^{(\ell)} = \mathbf{q}^T \mathbf{h}_v$$

3. Loss: Binary cross-entropy (from logits) summed over all L iterations

The GNN consists of four small MLPs and two trainable vectors \mathbf{w} and \mathbf{q}

Simulation results for (63,45) BCH code



- $L = 8$ iterations
- Messages and node attributes have 20 dimensions
- MLPs have one hidden layer with 40 units and tanh activation

GNN-based decoding notebook



GNN-based decoding

Other applications of GNNs & self-attention

- MIMO Detection
 - MIMO-GNN-MMSE [SML⁺20]
 - RE-MIMO [PRW21]
 - AMP-GNN [HKY⁺22]
 - GEPNet [KOH⁺22]
 - Self-attention-based Multi-User MIMO Demapper [MAS22]
- Channel estimation [LT22]
- Channel coding [SW21]
- Radio Resource Management [SSZL21, HXO⁺21]

A lot of potential for self-attention and GNNs in our field

Lessons learned

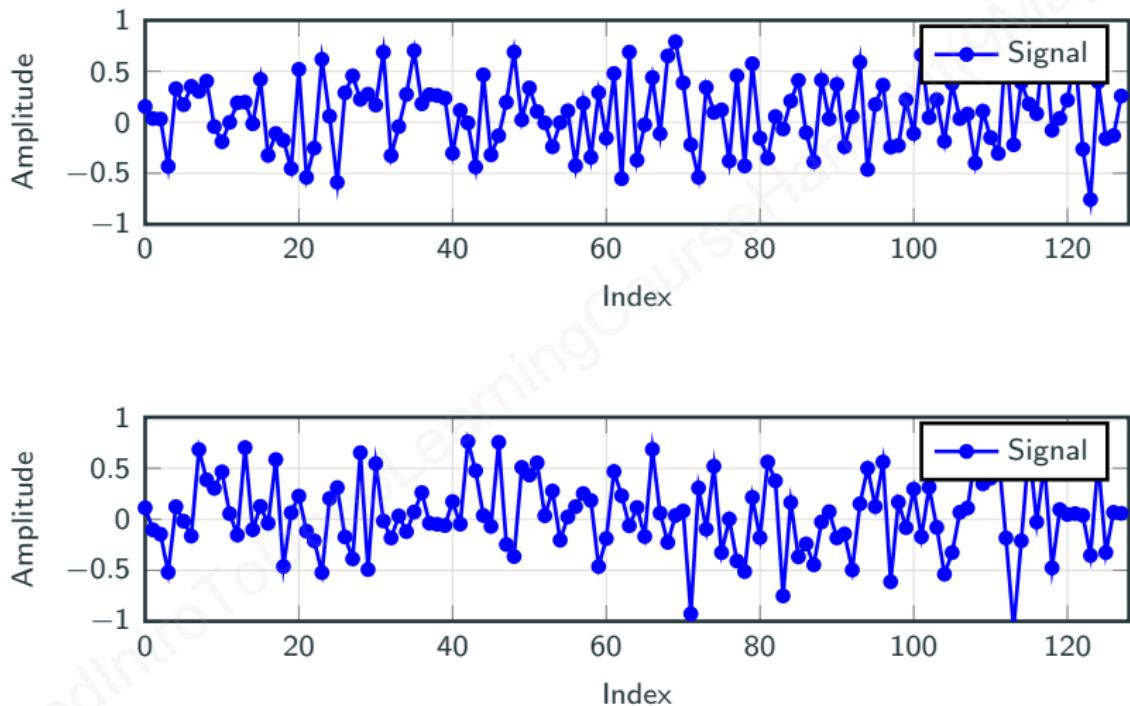
Neural Networks for Codes and Graphs

- GNNs extend the use of NNs to data represented as graph
- GNNs typically consist of
 - Trainable message & readout functions that are shared across nodes
 - Permutation-invariant aggregation function
- Most powerful GNN is MPNN, but has high complexity & memory consumption

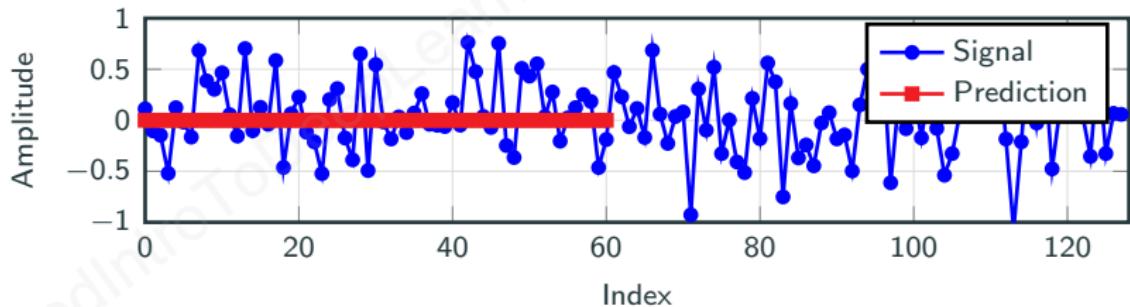
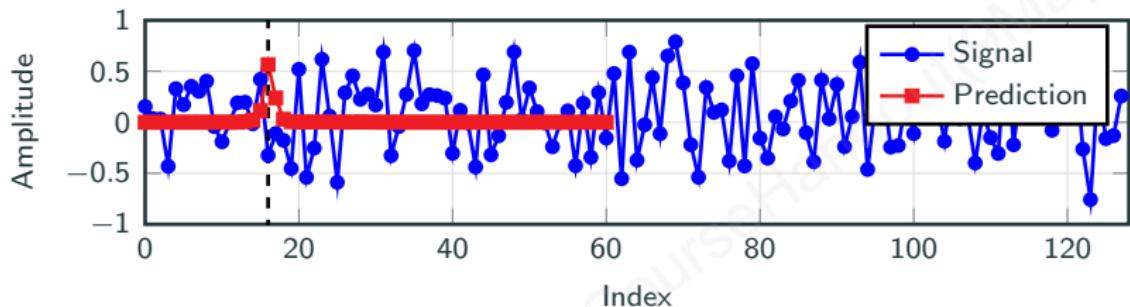


Short-message Communications

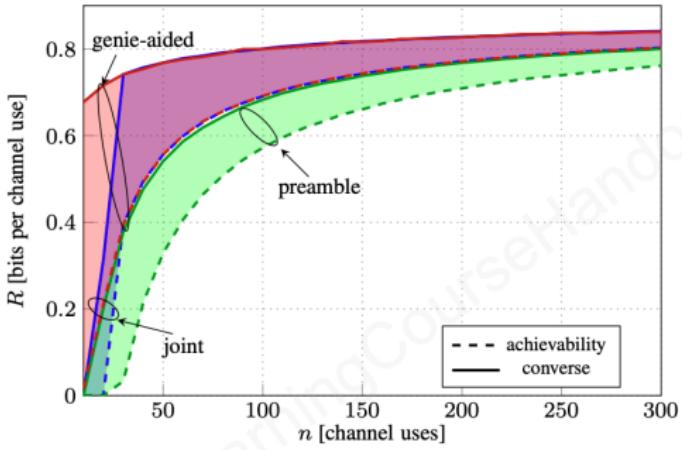
Where is the payload? [DCCB22]



Where is the payload? [DCCB22]



Detect the presence of a message [LÖD21]



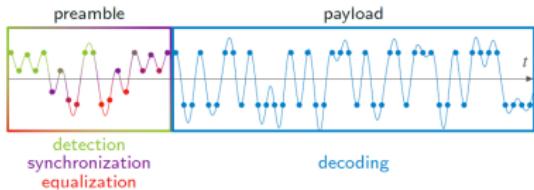
Terminology

(c) $\rho = 6 \text{ dB}$.

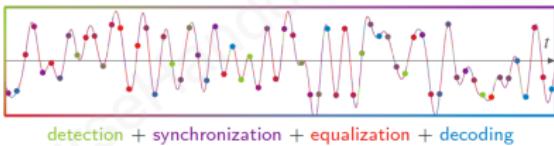
- Detection: presence of message within time-frame (yes/no)
- Synchronization: find exact timing offset (per sample-basis)

For short length messages, the separation into preamble for detection and data payload has been proven to be sub-optimal [LÖD21].

Detect the presence of a message [LÖD21]



(a) classic scheme



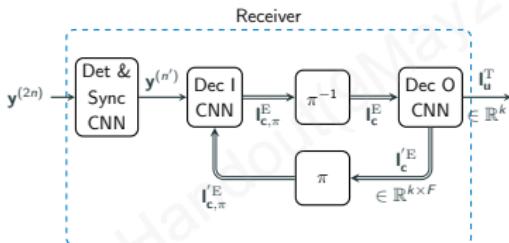
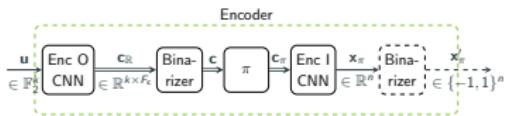
(b) Holistic scheme

Terminology

- Detection: presence of message within time-frame (yes/no)
- Synchronization: find exact timing offset (per sample-basis)

For short length messages, the separation into preamble for detection and data payload has been proven to be sub-optimal [LÖD21].

Turbo-Autoencoders [JKA⁺19] for synchronization



Detection as classification problem:

- Sequentially process snippets of length $2n$
- Det. & sync NN has softmax activated output with $n + 1$ neurons⁷
- Training with random time-offsets $\tau = 0, \dots, n - 1$ and zero messages (noise only)

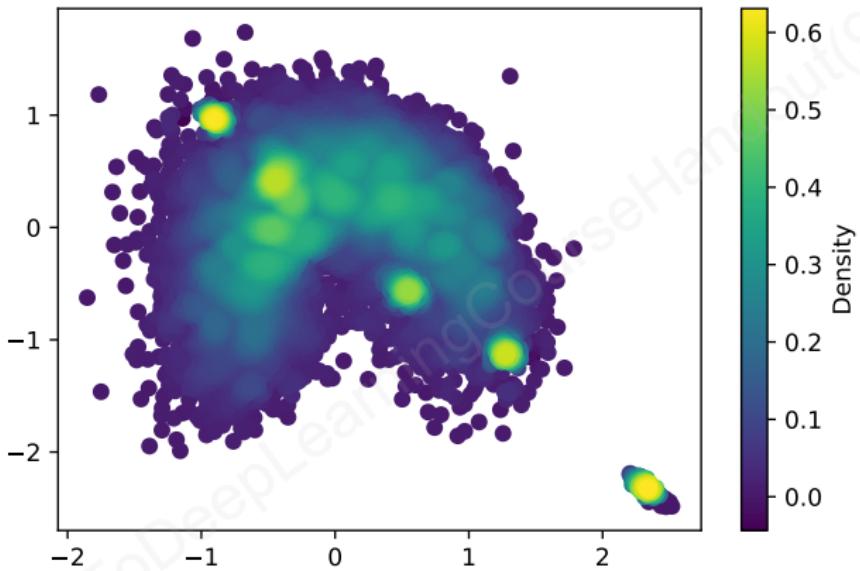
$$L_{\text{tot}} = L_{\text{BCE,AE}} + \alpha \cdot L_{\text{CCE,det}}$$

where

- $L_{\text{BCE,AE}}$ is the autoencoder end-to-end metric (BCE)
- $L_{\text{CCE,det.}}$ is the categorical cross-entropy between τ and $\hat{\tau}$
- α is a hyperparameter to control false positive rate

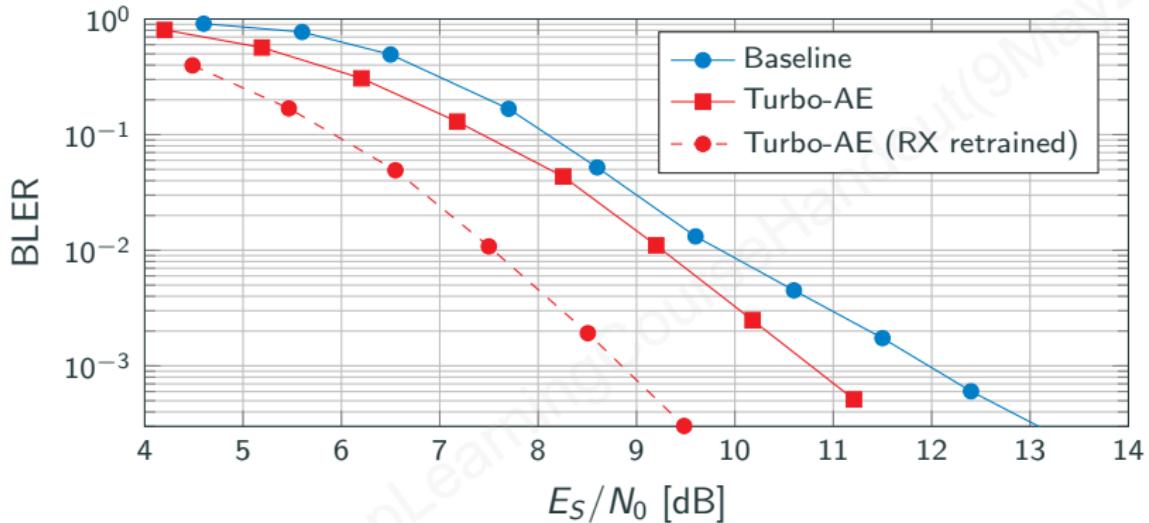
⁷Last neuron indicates *no message* state.

Learned constellations $(k, n) = (64, 64)$



- Learned n -dimensional constellation visualized via heatmap
- Superimposed pilot positions are visible
- Further constraints are possible: PAPR, ACLR, ...

Over-the-air results for $(k, n) = (64, 64)$ ⁸



- Two universal software radio peripherals (USRP)s B210
- 40MHz bandwidth @ $f_c = 2.35\text{GHz}$
- Static indoor environment (non-line of sight)

⁸S. Dörner et al. "Learning Joint Detection, Equalization and Decoding for Short-Packet Communications", <https://arxiv.org/pdf/2207.05699.pdf>

Lessons learned

End-to-end Learning

- Learning of the entire physical layer is possible
- We *can* achieve the theoretically optimal performance
- Learning of holistic messages without the need of a dedicated preamble



Outlook & Conclusion

Many topics that we cannot cover in this course

- Self-supervised learning
- Generative adversarial networks
- Diffusion models
- Variational autoencoders
- Compression & Quantization
- Architecture search
- Reinforcement learning
- Bayesian optimization
- ...

Final thoughts

- ML is now a well established tool in our field
- Hardly any problem that cannot be solved with \geq SOTA performance
- ML-based solutions often simpler to implement than algorithms
- Practical problems:
 - Complexity (often too high)
 - Parametrization (one model per set of parameters is prohibitive)
 - Adaptation & Online training (where would this be done?)
 - Data acquisition

References i

- [AAH22] F. Aït Aoudia and J. Hoydis, "End-to-End Learning for OFDM: From Neural Receivers to Pilotless Communication," *IEEE Transactions on Wireless Communications*, vol. 21, no. 2, pp. 1049–1063, 2022.
- [AH19] F. Aït Aoudia and J. Hoydis, "Model-Free Training of End-to-End Communication Systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2503–2516, 2019.
- [Bar94] A. R. Barron, "Approximation and estimation bounds for artificial neural networks," *Machine Learning*, vol. 14, no. 1, pp. 115–133, 1994.

References ii

- [BHM⁺16] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, “Using fast weights to attend to the recent past,” in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/9f44e956e3a2b7b5598c625fcc802c36-Paper.pdf
- [BHP⁺20] A. Buchberger, C. Häger, H. D. Pfister, L. Schmalen, and A. Graell i Amat, “Pruning and quantizing neural belief propagation decoders,” *IEEE Journal on Selected Areas in Communications*, 2020.
- [Bö17] G. Böcherer, *Achievable Rates for Probabilistic Shaping*, 2017.

References iii

- [CTB98] G. Caire, G. Taricco, and E. Biglieri, “Bit-interleaved Coded Modulation,” *IEEE Transactions on Information Theory*, vol. 44, no. 3, pp. 927–946, 1998.
- [CW22] Y. Choukroun and L. Wolf, “Error correction code transformer,” *arXiv preprint arXiv:2203.14966*, 2022.
- [Cyb89] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [DCCB22] S. Dörner, J. Clausius, S. Cammerer, and S. t. Brink, “Learning joint detection, equalization and decoding for short-packet communications,” 2022.

- [DCHt18] S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, “Deep Learning Based Communication Over the Air,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2018.
- [DHS11] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

References v

- [DLL⁺19] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, “Gradient descent finds global minima of deep neural networks,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1675–1685. [Online]. Available: <https://proceedings.mlr.press/v97/du19c.html>
- [DZPS18] S. S. Du, X. Zhai, B. Poczos, and A. Singh, “Gradient descent provably optimizes over-parameterized neural networks,” *arXiv preprint arXiv:1810.02054*, 2018.
- [Gal62] R. Gallager, “Low-density parity-check codes,” *IRE Trans. Inform. Theory*, pp. 21–28, 1962.

References vi

- [GBC16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [GCHt17] T. Gruber, S. Cammerer, J. Hoydis, and S. ten Brink, “On deep learning-based channel decoding,” in *Proc. of CISS*, 2017.
- [GSR⁺17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

- [Hin16] G. Hinton, “Coursera Course: Neural Networks for Machine Learning - Lecture 6e,” 2016. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- [HKMMT90] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, “A real-time algorithm for signal analysis with the help of the wavelet transform,” in *Wavelets*. Springer, 1990, pp. 286–297.
- [HKY⁺22] H. He, A. Kosasihy, X. Yu, J. Zhang, S. . H. Song, W. Hardjawanay, and K. B. Letaief, “Graph Neural Network Enhanced Approximate Message Passing for MIMO Detection,” *arXiv preprint arXiv:2205.10620*, 2022.

References viii

- [HRW14] J. Hershey, J. Roux, and F. Weninger, “Deep unfolding: Model-based inspiration of novel deep architectures,” *arXiv preprint arXiv:1409.2574*, 2014.
- [HSW89] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [HXO⁺21] S. He, S. Xiong, Y. Ou, J. Zhang, J. Wang, Y. Huang, and Y. Zhang, “An overview on the application of graph neural networks in wireless networks,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2547–2565, 2021.

- [HZRS16] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [JKA⁺19] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, “Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [KB14] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

References x

- [KOH⁺22] A. Kosasih, V. Onasis, W. Hardjawana, V. Miloslavskaya, V. Andrean, J.-S. Leuy, and B. Vucetic, "Graph neural network aided expectation propagation detector for mu-mimo systems," *arXiv preprint arXiv:2201.03731*, 2022.
- [KW17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.
- [LG17] L. Lugosch and W. Gross, "Neural offset min-sum decoding," *IEEE Int. Symp. on Inform. Theory (ISIT)*, pp. 1361–1365, June 2017.
- [LHP18] M. Lian, C. Häger, and H. Pfister, "What can machine learning teach us about communications?" in *2018 IEEE Information Theory Workshop (ITW)*, 2018, pp. 1–5.

References xi

- [LÖD21] A. Lancho, J. Östman, and G. Durisi, “On joint detection and decoding in short-packet communications,” in *IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.
- [LP19] Y.-H. Liu and D. Poulin, “Neural belief-propagation decoders for quantum error-correcting codes,” *Physical review letters*, vol. 122, no. 20, p. 200501, 2019.
- [LT22] D. Luan and J. Thompson, “Attention based neural networks for wireless channel estimation,” *arXiv preprint arXiv:2204.13465*, 2022.
- [MAS22] A. Michon, F. A. Aoudia, and K. P. Srinath, “Convolutional self-attention-based multi-user mimo demapper,” *arXiv preprint arXiv:2201.11779*, 2022.

- [Mit97] T. M. Mitchell, *Machine learning*. McGraw-Hill, 1997.
- [MLN19] P. Michel, O. Levy, and G. Neubig, “Are sixteen heads really better than one?” *Advances in neural information processing systems*, vol. 32, 2019. [Online]. Available: <https://arxiv.org/pdf/1905.10650.pdf>
- [NBB16] E. Nachmani, Y. Be’ery, and D. Burshtein, “Learning to decode linear codes using deep learning,” *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, Sept 2016.
- [OH17] T. O’Shea and J. Hoydis, “An Introduction to Deep Learning for the Physical Layer,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

References xiii

- [ORWH18] T. O'Shea, T. Roy, N. West, and B. C. Hilburn, "Physical Layer Communications System Design Over-the-Air Using Adversarial Networks," in *26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 529–532.
- [PH22] M. Phuong and M. Hutter, "Formal algorithms for transformers," *arXiv preprint arXiv:2207.09238*, 2022.
- [Pre05] M. Petti, "A message-passing algorithm with damping," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 11, p. P11008, 2005.
- [PRW21] K. Pratik, B. D. Rao, and M. Welling, "RE-MIMO: Recurrent and Permutation Equivariant Neural MIMO Detection," *IEEE Transactions on Signal Processing*, vol. 69, pp. 459–473, 2021.

References xiv

- [Qia99] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.
- [Sam59] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.
- [SML⁺20] A. Scotti, N. N. Moghadam, D. Liu, K. Gafvert, and J. Huang, "Graph neural networks for massive MIMO detection," *arXiv preprint arXiv:2007.05703*, 2020.

- [SSZL21] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, “Graph neural networks for scalable radio resource management: Architecture design and theoretical analysis,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 101–115, 2021.
- [SW21] V. G. Satorras and M. Welling, “Neural enhanced belief propagation on factor graphs,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 685–693.
- [SZ14] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [Tan81] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inform. Theory*, pp. 533–547, 1981.

References xvi

- [Vap13] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2013.
- [VSP⁺17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [XWT⁺21] H. Xiao, Z. Wang, W. Tian, X. Liu, W. Liu, S. Jin, J. Shen, Z. Zhang, and N. Yang, “AI enlightens wireless communication: Analyses, solutions and opportunities on CSI feedback,” *China Communications*, vol. 18, no. 11, pp. 104–116, 2021.

References xvii

- [YFW05] J. Yedidia, W. Freeman, and Y. Weiss, “Constructing free-energy approximations and generalized belief propagation algorithms,” *IEEE Trans. Inform. Theory*, vol. 51, no. 7, pp. 2282–2312, 2005.
- [YK15] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” *arXiv preprint arXiv:1511.07122*, 2015.
- [YLJ18] H. Ye, G. Y. Li, and B.-H. Juang, “Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems,” *IEEE Wireless Communications Letters*, vol. 7, no. 1, pp. 114–117, 2018.

References xviii

- [YLJ21] ——, “Deep Learning Based End-to-End Wireless Communication Systems Without Pilots,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 702–714, 2021.

Abbreviations and Acronyms

Term = Description

3G = third generation

3GPP = third generation partnership project

3GPP2 = 3rd Generation Partnership Project 2

5G = 5th generation wireless

5GC = 5G Core Network

5G-EIR = 5G Equipment Identity Register

5G NR = 5th Generation New Radio

5G NSA = 5G Non-Stand-Alone network

5G RG = 5G Residential Gateway

5GS = 5G System

5G SA = 5G Stand-Alone network

5QI = 5G QoS Class Index

A = availability

A5 = encryption algorithm

AAA= authentication authorization accounting

AAD = additional authentication data

AAH = asterisk at home (server)

AAL = Accelerator Abstraction Layer

AAnF = AKMA Anchor Function

AAS = Active Antenna Systems

ABBA = Anti-Bidding down Between Architectures

ABS = almost blank subframes

ABW = Aggregated Bandwidth

ACK = Positive acknowledgement

ACM = address complete message

ADC = analog to digital converter

AES = advanced encryption standard

AF = Application Function

AF = Diffserv Assured Forwarding

AFD = average fade distortion

AGC = automatic gain control

AI = Artificial Intelligence

AIFS = Arbitration Inter Frame Spacing
AKMA = Authentication and Key Management for Applications
AM = amplitude modulation
AM = Acknowledged Mode
AMC = adaptive modulation and coding
AMF = Access and Mobility Management Function
AMPS = advanced mobile phone service (system)
AN = Access Node/Network
ANM = answer message
ANR = Automatic Neighbor Relations
ANSI = American National Standards Institute
AODR = Ad hoc on demand routing
AP = access point
APD = Avalanche photodiode
APCO = Association of Public safety Communication Officials
API = Application programming interface
AR = axial ratio for elliptical polarization
ARIB = Association of Radio Industries and Business
ARFCN = Absolute Radio-frequency Channel Number
ARP = Absolute Radio-frequency Channel Number
ARPF = Authentication credential Repository and Processing Function
ARQ = automatic repeat-request
AS = application server
ASCII = American Standard code for information interchange
ASIC = Application-specific integrated circuit
ASK = amplitude shift keying
ASN = access service network
ASN.1 = abstract syntax notion one
ASP = application service provider
ATIS = Association Telecommunications Industries Standards
ATM = asynchronous transfer mode
ATPC = automatic transmit power control
AS = autonomous system
AuC = authentication center
AUSF = Authentication Service Function

AUT = antenna under test
AUTN = network authentication token
AUTS = Token used in resynchronization
AWGN = additive white Gaussian noise
AWS = advanced wireless services
AxC = Antenna-carrier
BBU = baseband unit
BCMCS = broadcast and multicast services
BE = best effort
BER = bit error rate
BF = Beamforming
BGP = border gateway protocol
BoD = Bandwidth on demand
BPSK= binary phase shift
BRAS = broadband remote access server
BS = base station
BSC = base station controller
BSF = Binding Support Function
BSR = Buffer Status Report
BSS = basic service set
BTS = base transceiver station
BWP = Bandwidth Part
CA = Carrier Aggregation
CB = certification body
CB = Coordinated Beamforming
CBC = cipher block chaining message
CBC-MAC = cipher block chaining message authentication code
CBG = Code Block Group
CBQ = class based queuing
CBW = Channel Bandwidth
CC = call control
CCCH/BCCH = common control channel broadcast control channel
CCI = co-channel interface
CCK = Complimentary Code Keying
CCM = CTR mode with CBC-MAC

CCMP = counter mode with cipher block chaining message authentication code protocol
CCSA = China Communications Standard Association
CDMA = code division multiple access
CDN = content delivery network
C-DRX = Connected Mode Discontinuous Reception
CE = Coverage Enhancement
CEL = Coverage Enhancement Level
CGM = conjugate gradient method
CH = Corresponding Host
CHF = Charging Function
CID = connection ID
CIoT = Cellular IoT
CIR = carrier to interference ratio
C&M = Control and Management
CM = connection management
CM = Configuration Management
CMA = constant modulus algorithm
cMTC = Critical Machine Type Communications
CMTS = Cable Modem Termination System
CN = Core Network
CNR = carrier-to-noise ratio
COFDM = coded orthogonal frequency division multiplexing
CoMP = Coordinated MultiPoint
COMP128 = algorithm
CP = circular polarization
CP = cyclic prefix
CP = Control Plane
CPC = cyclic prefix code
CP-OFDM = Cyclic Prefix OFDM
CPRI = common public radio interface
CPU = Central processing unit
CQI = channel quality indicator
C-RAN = Cloud radio access network
C-RAN = Centralized RAN
CRC = cyclic redundancy check (cyclic redundancy code)

CRC-32 = cyclic redundancy check 32 bits
CRS = Cell-specific Reference Signal
CS = circuit switched
CS = coding scheme
CS = Coordinated Scheduling
CSA = Canadian Standard Association
CSCF = call session control function
CSFB = Circuit Switched FallBack
CSG = closed subscriber group
CSI = Channel State Information
CSI-IM = Channel State Information Interference Measurement
CSI-RS = Channel State Information-Reference Signals
CSMA/CA = carrier-sense multiple access with collision avoidance
CSMA/CD = carrier sense multiple access with collision detection
CSN = connectivity service network
CSP = Communications Service Provider
CST = computer simulation technology
CTIA = International Association for the Wireless Telecommunications Industry
CTS = clear to send
CU = centralized unit
CW = Control word
D2D = Device to Devices
DAC = Digital-to-analog conversion
DARPA = Defense Advanced Research Projects Agency
DBA = Dynamic bandwidth allocation
dBi = decibel isotropic
dBm = decibel milliwatt
dB_r = decibel relative
DC = Dual Connectivity
DCF = distributed coordination function
DCH = dedicated channel
DDoS = distributed denial-of-service
DECT = digital enhanced cordless telephony
DeNB = Donor eNodeB
DES= data encryption standard

DFS = Dynamic Frequency Selection
DFT = Discrete Fourier Transform
DHCP = Dynamic Host Configuration Protocol
DI = Delay interferometer
DIFS = Distributed Inter-Frame Space
DL = down link
DL-SCH = Downlink Shared CHannel
DMB = digital multimedia broadcasting
DML = Directly modulated laser
DM-RS = DeModulation Reference Signal
DMT = Discrete multi-tone
DNS = domain name system
DoS = denial-of-service
DPCCH = dedicated physical control channel
DPI = Deep Packet Inspection
DPS = Dynamic Point Selection
DPSK = differential phase shift keying
DQPSK = differential quadrature (or quaternary) phase shift keying
DRA = dielectric resonator antenna
D-RAN= Distributed RAN
DRB = Data Radio Bearer
DRC = data rate control
DRX = Discontinuous Reception
D-TLS = Datagram Transport Layer Security
DS = Distribution System
DS0 = Digital Signal 0 (zero)
DS-CDMA = direct sequence code division multiple access
DSL = digital subscriber line
DSP = Digital signal processor
DSR = dynamic source routing
DSS1 = digital subscriber signaling system 1
DSSS = direct sequence spread spectrum
DU = distributed unit
DVB-H = digital video broadcast handheld
DWDM = dense wavelength division multiplexing

DwPTS = Downlink Pilot Time Slot

E_b / N_0 = energy per bit to white noise power spectral density ratio

EAP = extensible authentication protocol

EAP-FAST = EAP flexible authentication via secure tunneling

EAPoL = EAP over LAN

EAP-TLS = EAP transport layer security

EAP-TTLS = EAP tunneled TLS

EC = Edge Computing

ECGI = EUTRAN Cell Global ID (EUTRAN is Universal Terrestrial Radio Access Network – same as eNB)

ECIES = Elliptic Curve Integrated Encryption Scheme

ECP = Extended Cyclic Prefix

eCPRI = Evolved CPRI

EC-GSM = Extended Coverage GSM

E-DCH = enhanced dedicated channel

EDFA = Erbium-doped fiber amplifier

EDGE = enhanced data rates for GSM evolution

eDRX = Extended Discontinuous Reception

EDT = Early data transmission

EF = diffserv expedited forwarding

EGC = equal gain combining

EGPRS = enhanced GPRS

EIA = Electronics Industries Alliance

EIFS = Extended Inter Frame Spacing

EIR = equipment identity register

EIRP = effective isotropic radiated power

EM = electromagnetic

eMBB = enhanced Mobile BroadBand

EMF = Efficient mobile fronthaul

EMS/NMS = Element/network management system

eNB = Evolved Node B – The LTE Radio Access Network

EN-DC = E-UTRA NR Dual Connectivity

EP = elliptical polarization

EPC = Evolved Packet Core

EPON = Ethernet PON
EQ = Equalizer
ERP = effective radiated power
eSIM = embedded SIM
ESS = extended service set
eTOM = enhanced telecom operations map
ET = error tracking
ETSI - European Telecommunication Standards Institute
eUICC = embedded universal integrated circuit card
E-UTRA = Evolved universal terrestrial radio access
E-UTRAN = Evolved Universal Terrestrial Radio Access Network
EVM = Error vector magnitude
FA = foreign agent
FACA = US Federal Advisory Committee Act
FBA = Fixed bandwidth allocation
FBSS = fast base station switching
FCAPS = Fault Configuration Accounting Performance and Security
FCC = Federal Communications Commission
FDD = frequency division duplexing
FDM = Frequency-division multiplexing
FDMA = frequency division multiple access
FDTD = finite difference time domain
FEC = Forward error correction
FEM = finite element method
FFE = Feed-forward equalizer
FFS = For Future Study
FFT = fast fourier transform
FH-CUS = Fronthaul Control, User and Synchronization Planes
FH-M = Fronthaul Management Plane
FHSS = frequency hop spread spectrum
FM = Fault Management
FN-RG = Fixed Network Residential Gateway
FPGA = Field-programmable gate array
FR1 - Frequency Range 1
FR2 = Frequency Range 2

FSK = frequency shift keying
FSO = free space optics
FSS = frequency selective surfaces
FSTD = Frequency Switched Transmit Diversity
FTTB = Fiber to the business
FTTH = Fiber to the home
G.711 = encoder
GAA = General Authorized Access
GBA = Generic Bootstrapping Architecture
GEO = geostationary earth orbit
GGSN = gateway GPRS support node
GKH = group key hierarchy
GMLC = Gateway Mobile Location Centre
GMPLS = Generalized multiprotocol label switching
GMSC = gateway mobile switching center
GMSK = Gaussian minimum shift keying
gNB - gNODE B
gNB = next generation Node B
gNB-CU = gNB Central Unit
gNB-DU = gNB Distributed Unit
GPON = Gigabit-capable PON
GPRS = general packet radio service
GPS = global positioning system
GSA = GSM Suppliers Association
GSM = Global System for Mobile-communications
GSMA = GSM Association
GTC = generic token card
GTP = GPRS tunneling protocol
GUTI = Globally Unique Temporary UE Identity
GW = gateway
H.263 = video codec low-bit rate
H.264 = video codec MPEG-4 advanced video codec
HA = home agent
HARQ = hybrid automatic repeat request
HD-FDD = Half-duplex frequency-division duplexing

HD-FEC = Hard-decision FEC
HDLC = high-level data link control
HE = home environment
HeNB = Home eNodeB
HetNet = Heterogeneous Network
HF = Hyper-frame
HFSS = high frequency structure simulator
HHO = hard handoff
Hi-Cap = high capacity
HLR = home location register
HLS = Higher Layer Split
HLR/AUC = home location register / authentication center
HN = home network
H/O = Handover
HO = handoff
HPLMN = home public land mobile network
HSDPA = high speed downlink packet access
HS-DSCH = high speed downlink shared channel
H-SFN = H-System frame number
HSPA = high speed packet access
HSS = home subscriber server
HSUPA = high-speed uplink packet access
HTTP = hypertext transfer protocol
HW = Hardware
IBSS = independent basic service set
ICIC = Inter-Cell Interference Coordination
ICMP = internet control message protocol
I-CSCF = interrogating CSCF
ICV = integrity check value
ID= Identification
IDEN = integrated digital enhanced network
I-DRX = Idle Mode Discontinuous Reception
IDU = indoor unit
IEC = International Electro Technical Commission

IECEE = International Electrotechnical Committee for Conformity Testing to Standards for Electrical Equipment

IETF = Internet Engineering Task Force

IF = intermediate frequency

IFFT = inverse fast fourier transform

IK = integrity key

IKE = internet key exchange

IM = Information Model

IMD = Inter-Modulation Distortion

IMS = IP multimedia subsystem

IMSI = international mobile subscriber identity

IMP-2000 = International Mobile Telecommunications 2000 ITU Standard

IMT = International Mobile Telecommunications

IOT = Internet of Things

IP = internet protocol

IPv4 = internet protocol version 4

IPv6 = internet protocol version 6

IP-CAN = IP connectivity access networks

IPSec = internet protocol security

IPUPS = Inter-PLMN User Plane Security

I/Q = In-phase / Quadrature

IRP = Integration Reference Point

IS-136 = Interim Standard 136

IS-95 = Interim Standard 95

ISAKMP = Internet Security Association and Key Management Protocol

ISI = inter-symbol interference

iSIM = Integrated SIM

IS-IS = intermediate system to intermediate system

ISM = industrial, scientific, and medical (band)

ISO = International Standard Organization

ISUP = ISDN user part

ISUP IAM = ISUP initial address message

I-TCP = indirect transmission control protocol

Itf-N = “Northbound Interface” – Interface between Network Management System and Element Management System

Itf-P2P = “Peer-To-Peer Interface” – Interface between two element management systems
Itf-S = “Southbound Interface” – Interface between Element Management System and the Elements

ITIL = Information Technology Infrastructure Library

ITU = International Telecommunication Union

ITU-R = ITU-Radiocommunication (radio communication sector)

ITU-T = ITU-Telecommunication (standards sector)

KA = knowledge area

KC = ciphering key

KCI = EAPoL key communication key

KDF = Key Derivation Function

KEK = EAPoL key encryption key

KPI = key performance indicators

KQI = Key Quality Indicators

L1 = Layer 1

L2 = Layer 2

L3 = Layer 3

LAA = License Assisted Access

LAN = local area network

LEO = low earth orbit

LH = left hand circular polarization

LLS = Lower Layer Split

LINP = Logically Isolated Network Partitions

LMF = Location Management Function

LMS = least mean square

LO = local oscillator

Lo-Cap = low capacity

LOS = line of sight

LP = linear polarization

LPWAN = Low-power wide-area networks

LR-WPAN = low rate wireless personal area network

LS-CMA = least squares constant modulus algorithm

LTE = long term evolution

LTE-A = LTE-Advanced

LTE-M = LTE Cat-M1

MA = Managed Application
ME = Managed Element
MF = Managed Function
ML = Machine Learning
MAC = Media Access Control
MAC = media access protocol
MAC = message authentication code
MAC-S = authentication token used in resynchronization
MAN = metropolitan area network
MAP = mobile application part
MBB = Mobile broadband
MBMS = multimedia broadcast / multicast service
MBSFN = Multicast-broadcast single-frequency network
MCG = Master Cell Group
MCL = Maximum Coupling Loss
MCPTT = Mission critical push-to-talk over LTE
MCS = Modulation and Coding Scheme
MCS = Mission Critical Service
MCW = multi codeword
MD5 = message digest (algorithm) 5
MDHO = macro diversity handover
MDS = minimum discernible signal
MDT = Minimization of Drive Tests
MEC = Mobile Edge Computing
MEdiaFLO = forward link only technology
MeNB = Master eNB
MEO = medium earth orbit
METIS = Mobile and Wireless Communications Enablers for the 2020 Information Society
MFH = Mobile fronthaul
MGCF = media gateway control function
MGW = media gateway
MH = Mobile Host
MIB = management information base
MIC = message integrity check
MIMO = multiple-input multiple-output

MIP = mobile IP
MISO = multiple input single output
MLSE = Maximum likelihood sequence estimation
MM = mobility management
MME = mobility management entity
mMIMO = Massive MIMO
mMTC = Massive MTC
mMTC = Massive Machine Type Communication
MMUSIC = multiparty multimedia session control
mMW = Millimeter Wave
MN = Master Node
MNO = Mobile network operator
MoM = method of moments
MOS = mean opinion score
MPDCCH = MTC physical downlink control channel
MPDU = MAC protocol data unit
MPEG = moving picture expert group
MPLS = multiprotocol label switching
MR = mesh router
MRC = maximum ratio combining
MRF = media resource function
MS = mobile station
MSC / VLR = mobile switching center / visitor location register
MSC = mobile switching center
MSISDN = mobile station integrated services digital network
MSK = minimum-shift keying
MSRN= mobile station roaming number
MSS = maximum segment size
MTBF = mean time between failures
MTC = Machine Type Communication(s)
MTTR = mean time to repair
MU-MIMO = multiple user MIMO
MU = Multi User
MZM = Mach-Zehnder modulator
N3IWF = Non-3GPP Interworking Function

NACK = negative acknowledgement
NAC = Network Access Control
NAS = network access server
NAV = network allocation vector
NB = Narrow Band
NBAP= node B application part
NB-IOT = NarrowBand Internet of Things
NCRP = National Council on Radiation Protection
Near-RT RIC = Near-Real-Time RAN Intelligent Controller
NEBS = Network Equipment Building Systems Standard
NEC = numerical electromagnetics code
NE-DC = NR-EUTRA Dual Connectivity
NEF = Network Exposure Function
NESAS = Network Equipment Security Assurance Scheme
NETCONF = Network Configuration Protocol
NG = Next Generation
NGC/N = Next Generation Core/ Network
NGEN-DC = Next Generation EUTRA-NR Dual Connectivity
NGMN = Next Generation Mobile Network
NG-RAN = Next Generation RAN
NFV = Network Functions Virtualization
NF = noise figure
NFC = near field communication
NGEN-DC = Next Generation EUTRA-NR Dual Connectivity
NGFI = Next-generation fronthaul interface
NGMC = next generation mobile committee
NGMN = next generation mobile networks
NGN = next generation network(s)
NIC= network interface card
NIST = National Institute of Standards and Technology
NLOS = non-line-of-sight
NMHA = normal mode helical antenna
NMS = network management system
Node B = base station designation in UMTS
Non-RT RIC = Non-Real-Time RAN Intelligent Controller

NPA = nonlinear power amplifier
NPBCH = Narrowband physical broadcast channel
NPDCCH = Narrowband physical downlink control channel
NPDSCH = Narrowband physical downlink shared channel
NPN = Non-public network
NPRACH = Narrowband physical random access channel
NPRS = Narrowband positioning reference signal
NPSS = Narrowband primary synchronization signal
NPUSCH = Narrowband physical uplink shared channel
NR = 5G New Radio
NR = New Radio
NRF = Network Repository Function
NR-PSS = New Radio-Primary Synchronization Signal
NR-SSS = New Radio-Secondary Synchronization Signal
NRSC = network reliability steering committee
NRZ = non-return to zero
NSA = Non Stand Alone
NSP = network service provider
NSS = network subsystem
NSSAI = Network Slice Selection Assistance Information
 NSSF = Network Slice Selection Function
NSTAC = National Security Telecommunications Advisory Committee (US)
NW = Network
NWDAF = Network Data Analytics Function
NWUS = Narrowband Wake-up Signal
NZDF = Non-zero dispersion fiber
OAM = Operation, Administration and Maintenance – Management System
OATS = open area test site
OCC = Orthogonal Cover Code
O-Cloud = O-RAN Cloud
O-CU-CP = O-RAN Central Unit – Control Plane
O-CU-UP = O-RAN Central Unit – User Plane
ODN = Optical distribution network
ODSP = DSP for optical transmission
ODU = outdoor unit

O-DU = O-RAN Distributed Unit
O-eNB = O-RAN eNB
OFDM = orthogonal frequency division multiplexing
O-FH = Open Fronthaul
OFDMA = orthogonal frequency division multiple access
OGC = Office of Government Commerce
OLO = Optical Local oscillator
OLSR = optimized link state routing
OLT = Optical line terminal
ONU = Optical network unit
OOK = On-off keying
O-RAN = Open RAN
O-RU = O-RAN Radio Unit
OSA = opportunistic spectrum address
OSG = Open Subscriber Group
OSNR = Optical signal-to-noise ratio
OSS/BSS = operational and business support systems
OSPF = open shortest path first
OSI = open systems interconnection
OTA = over the air programming
OTDOA = Observed time difference of arrival
OTN = Optical transport network
OTP = one time password
OVS = open virtual switch
OXC = Optical cross-connect
PA = power amplifier
PAaM = Phased Array Antenna Module
PAM = Pulse-amplitude modulation
PAN = personal area network
PAPR = high peak to average power ratio
PBCH = Physical Broadcast Channel
PBCCH = Packet Broadcast Control Channel
PCCH = Paging Control Channel
pCELL = Primary Cell
PCF = Policy Control Function

PCFR = policy and charging rules function
PCI = Physical Cell ID
PCM = pulse code modulation
P-CSCF = proxy CSCF
PD = Photo-detector
PDC = personal digital cellular
PDCCH = Physical Downlink Control Channel
PDCP = Packet data convergence protocol
PDM = Polarization-division multiplexing
PDSCH = Physical Downlink Shared Channel
PDSN = packet data serving node
PDU = protocol data unit
PDU = Packet data Unit
PEAP = protected EAP
PFDM = orthogonal frequency division multiplex
P-GW = variant of PDN-GW, packet data network gateway
PHICH = Physical Hybrid ARQ Indicator Channel
PHR = Power Head-room Reporting
PHY = physical (layer)
PIFA = planar inverted F antenna
PIFS = PCF Inter Frame Spacing
PIN = personal identification number
PKH = pairwise key hierarchy
PKI = Public Key Infrastructure
PKIX = Public Key Infrastructure (X.509)
PL = path loss
PM = Performance Management
PLMN = Public Land-Mobile Network
PMD = Polarization-mode dispersion
PMI = Precoder Matrix Indicator
PN = pseudo-noise
PO = physical optics
PON = Passive optical network
PPP = point to point protocol
PRACH = Physical Random Access Channel

PRB = Physical Resource Block
PRS = Positioning Reference Signals
PS = packet switched
PSCELL = Primary SCell
PSD = Power Spectral Density
PSK = phase shift keying
PSS = Primary Synchronization Sequence/Signal
PSTN = public switched telephone network
PTRS = Phase and frequency tracking reference signal
PT-RS = Phase Tracking Reference Signal
PUCCH = Physical Uplink Control Channel
PUSCH = Physical Uplink Shared Channel
QAM = quadrature amplitude modulation
QCI = Quality of Service Class Index
QFI = QoS Flow Identifier
QoS = quality of service
QPSK = quadrature (quaternary) phase shift keying
RA = Random Access
RAB = radio access bearer
RACH = random access channel
RADIUS = remote access dial in user server
RAI = Release assistance indication
RAN = radio access network
RAND = random
rApp = Non-RT RIC Application
RAR = Random Access Response
RAT - Radio Access Technology
RB = Resource Block
RBS = Radio Base Station
RC4 = RC4 cipher algorithm
RE = Radio equipment
RE = Resource Element
REC = Radio equipment control
RET = remote electrical tilt
RF = radio frequency

RFC = request for change
RFC = request for comment
RFID = radio frequency identification
RHCP = right hand circular polarization
RI = Rank Indicator
RIP = routing information protocol
RLC = radio link control
RLF = Radio Link Failure
RLS = recursive least squares
RMON = Remote network Monitoring
RN = Relay Node
RNC = radio network control
RNTI = Radio Network Temporary Identifier
ROADM = Reconfigurable optical add/drop multiplexer
ROAMOPS = IETF roaming operations
RoF = Radio-over-Fiber
ROHC = robust header compression
R-PDCCG = Relay Physical Downlink Control Channel
RR = radio resource
RRC = radio resource control
RRM = Radio Resource Management
RRH = Remote Radio Head
RRU = Remote radio unit
RS = Reference Signal
RSRP = Reference Signal Received Power
RSRQ = Reference Signal Received Quality
RSA = Rivest, Shamir, Alderman
RSN = robust security networks
RSNA = robust security network associations
RSRP = Reference Signal Received Power
RSSI = received signal strength indicator
RT = Real Time
RTP = real time protocol
RTS = request to send
RTT = round trip time

RU = Radio Unit
RX = Receiver
RZ = return to zero
S1 = Interface between RAN and CN in LTE
S1AP = S1 Application Protocol
S1-C(P) = S1 Control (Plane)
S1-U(P) = S1 User (Plane)
S/N = signal to noise ratio
SA = Security Association
SA = System Architecture
SAP = Service access point
SAR = specific absorption rate
SCF = Small Cell Forum
SDL = Shared Data Layer
SCAS = SeCurity Assurance Specifications
SCCP = signaling connection control protocol
SC-FDMA = Single Carrier FDMA
SCG = Secondary Cell Group
SCH = Shared Channel
SC-MCCH = Single cell multicast control channel
SC-MTCH = Single cell multicast traffic channel
SCP = ETSI smart card platform
SC-PTM = Single cell point to multipoint
SCS = Sub-carrier Spacing
S-CSCF = serving CSCF
SCTP = stream control transmission protocol
SCW = single codeword
SDAP = Service Data Adaptation Protocol
SDCCH = stand alone dedicated channel
SD-FEC = Soft-decision FEC
SDH = synchronous digital hierarchy
SDMA = space division multiple access
SDN = Software-defined network
SDP = session description protocol
SDR = software defined radio

SEAF = Security Anchor Function
SEGf = security gateway function
SEPP = Security Edge Protection Proxy
SERDES = Serializer & Deserializer
SET = secure electronic transaction
SF = spreading factor
SFBC = Space-Frequency Block Codes
SFDR = spurious free dynamic range
SFID = service flow ID
SFN = System frame number
SFP = Small Form-factor Pluggable
SG/MGC = signaling gateway/media gateway controller
SgNB = Secondary gNB
SGSN = serving GPRS support node
S-GW = serving gateway
SGW = signaling gateway
SHA = secure hash algorithm
SI = System Information
SIB = System Information Block
SIB-BR = SIB for Bandwidth-reduced
SID = system identification number
SIDF = Subscription Identifier De-concealment Function
SIFS = short inter-frame space
SIG = special interest group of WWRF
SIGTRAN = signal transport
SIM= subscriber identity module
SIMO = single input multiple output
SINR = signal-to-interference-plus-noise ratio
SiP = Silicon photonics
SIP = session initiation protocol
SIR = signal to interference ratio
SISO = single input single output
SLF = subscriber location function
SMF = Session Management Function
SMI = structure of management information

SMO = Service Management and Orchestration
SMS = short message service
SM-SC = short message service center
SMSF = Short Message Service Function
SN = Secondary Node
S-NSSAI = Single Slice NSSAI
SMTP = simple message transfer protocol
SNMP = simple network management protocol
SNN = Serving Network Name
SNR = signal-to-noise ratio
SON = Self Organizing Network
SPC = single parity check
SQN = sequence number
SR = Scheduling Request
SRB = Signaling Radio Bearer
SRES = signed response
SRS = Sounding Reference Signal
SRTP = secure RTP
SS - Synchronizing Signal
SS7 = signaling system number 7
SSB = single sideband
SSB = SS Block
SSHv2 = Secure Shell 2.0
SSID = service site identifier
SSMF = Standard single-mode fiber
SSPA = solid state power amplifier
SSS = Secondary Synchronization Sequence/Signal
SST = Slice Service Type
STA = stations
STM = synchronous transfer mode
SU = Single-User
SUL = Supplementary Uplink
SU-MIMO = Single User MIMO
SUPI = SUbscription Permanent Identifier
SUCI = SUbscription Concealed Identifier

SVLTE = simultaneous voice and LTE
SW = Software
SYN = synchronization
T2P = traffic to pilot
TA = Timing Advance
TA = Tracking Area
TB = Transport Block
TBD = To Be Defined
TCAP = transaction capabilities application part
TCH / FS = traffic channel full rate speech
TCH / HS = traffic channel half rate speech
TCH = traffic channel
TCP / IP = suite of protocols
TCP = transmission control protocol
TD-CDMA = time division CDMA
TDD = time division duplex
TDM = time-division multiplexing
TDMA = time division multiple access
TDOA = time difference of arrival
TD-SCDMA = time division synchronous CDMA
TE = Timing error
TIA = Telecommunications Industry Association
TIMSI = temporary international mobile subscriber identity
TK = temporal key
TKIP = temporal key integrity protocol
TLS = Transport Layer Security
TM = transmission Mode
TMF = TM Forum
TN = Transport Network
TPC = Transmit Power Control
TR = Technical Report
TRAP = TDMA-based randomly addressed polling
Triple DES = encryption standard
TRP = Total Radiated Power
TRxP = Transmission Reception Point

TS = time slot
TS = Technical Specification
TSC = TKIP sequence counter
T-SDN = Transport SDN
TSG CT = TSG core network & terminals
TSG GERAN = TSG GSM EDGE radio access network
TSG RA = ETG services and system aspects
TSG RAN = TSG radio access network
TTA = Telecommunications Technology Association of Korea
TTAK = TKIP-mixed transmit address and key
TTC = Telecommunications Technology Committee
TTI = Transmit Time Interval
TX = Transmitter
TXOP = Transmit Opportunity
UAS = Unmanned Aerial System
UAV = Unmanned Aerial Vehicle
UCI = Uplink Control Information
UDM = Unified Data Management
UDN = Ultra-dense networks
UDP = user datagram protocol
UDR = Unified Data Repository
UDSF = Unstructured Data Storage Function
UE = user equipment
UICC = Universal Integrated Circuit Card
UL = Underwriters Laboratories
UL = Uplink
ULA = Uniform Linear Array
UL-SCH = Uplink Shared CHannel
UMa = Urban Macro
UMi = Urban Micro
UMB = ultra mobile broadband
UMTS = universal mobile telecommunications system
UMTS AKA = protocol used in 3G
UP = User Plane
UPF = User plane Function

UpPTS = Uplink Pilot Time Slot
UPS = uninterruptible power supply
URLLC = Ultra-Reliable and Low Latency Communications
USGS = United States Geological Survey
USIM= universal subscriber identity module
UTRA = universal terrestrial radio access
UTRA TDD-HCR = TD-CDMA UTRA mode
UTRA TDD-LCR = TD-SCDMA UTRA mode
UTRAN = UMTS terrestrial radio access network
UWB = ultra-wideband
V2I = Vehicle to Infrastructure
V2P = Vehicle to Pedestrian
V2V = Vehicle to vehicle
V2X = Vehicle to Anything
V2X = Vehicle to Everything
VLR = visitor location register
VN = visited network
VNF = Virtualized Network Function
VoIP = Voice over Internet Protocol
VOLTE = voice over LTE
VOLGA = voice over LTE via Generic Access
VPLMN = visited public land mobile network
vRAN = Virtualized RAN
VSAT = very small aperture terminal
VSWR = voltage standing-wave ratio
WAN = wide area network
W-CDMA = wideband code division multiple access
WCET = wireless communications engineering technologies
WCP = wireless communication professional
WDM = Wavelength division multiplexing
WEP = wired equivalent privacy
WERT = wireless emergency response team
WFQ = weighted fair queuing
WG = working group of WWRF
WG = Working Group

Wi-Fi = Wireless Fidelity

WiMAX = worldwide interoperability for microwave access

WINNER = wireless world initiative new radio

WLAN = wireless local area network

WMAN = wireless metropolitan area network

WMN = wireless mesh network

WPA = Wi-Fi Protected Access

WPAN = wireless personal area network

WPS = Wi-Fi Protected Setup

WRC = World Radiocommunication Conference

WSS = Wavelength selective switch

WUS = Wake-up signal

WWRF = wireless world research forum

x2 = Interface between eNBs in LTE

X2-AP = X2 Application Protocol

X2 - UP = X2 User Plane

xApp = Near-RT RIC Application

XG = next generation

XKMS = XML Key Management Services

XMAC (PG 26) = cryptographic primitive in the 3GSM key generation process

xNF = Any Network Function

XOR = exclusive or

ZRP = zone routing protocol

ZTA = Zero Trust Architecture