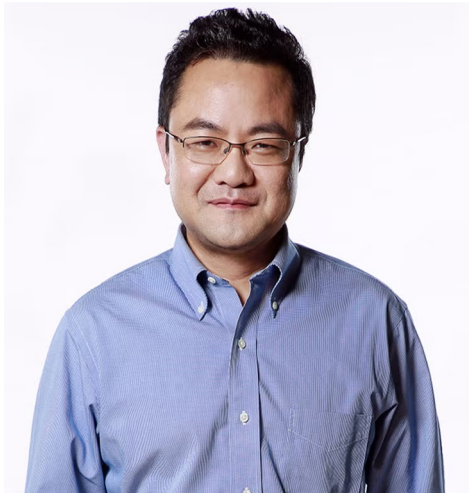


Deep Learning Methods in Advanced Statistical Problems

JSM 2025 Short Course

Agenda

- ❑ **Foundations of Deep Learning Methods (Hongtu)**
- ❑ **Computational Resources and Examples (Runpeng)**
- ❑ **Deep Generative Models (Xiao)**
- ❑ **Attention and Transformer (Xiao)**
- ❑ **Deep Sequence Modeling and Spatio-temporal Modeling (Hongtu)**
- ❑ **Large Language Models (Runpeng)**
- ❑ **Deep Learning in Advanced Statistical Problems (Hongtu)**




Course Websites

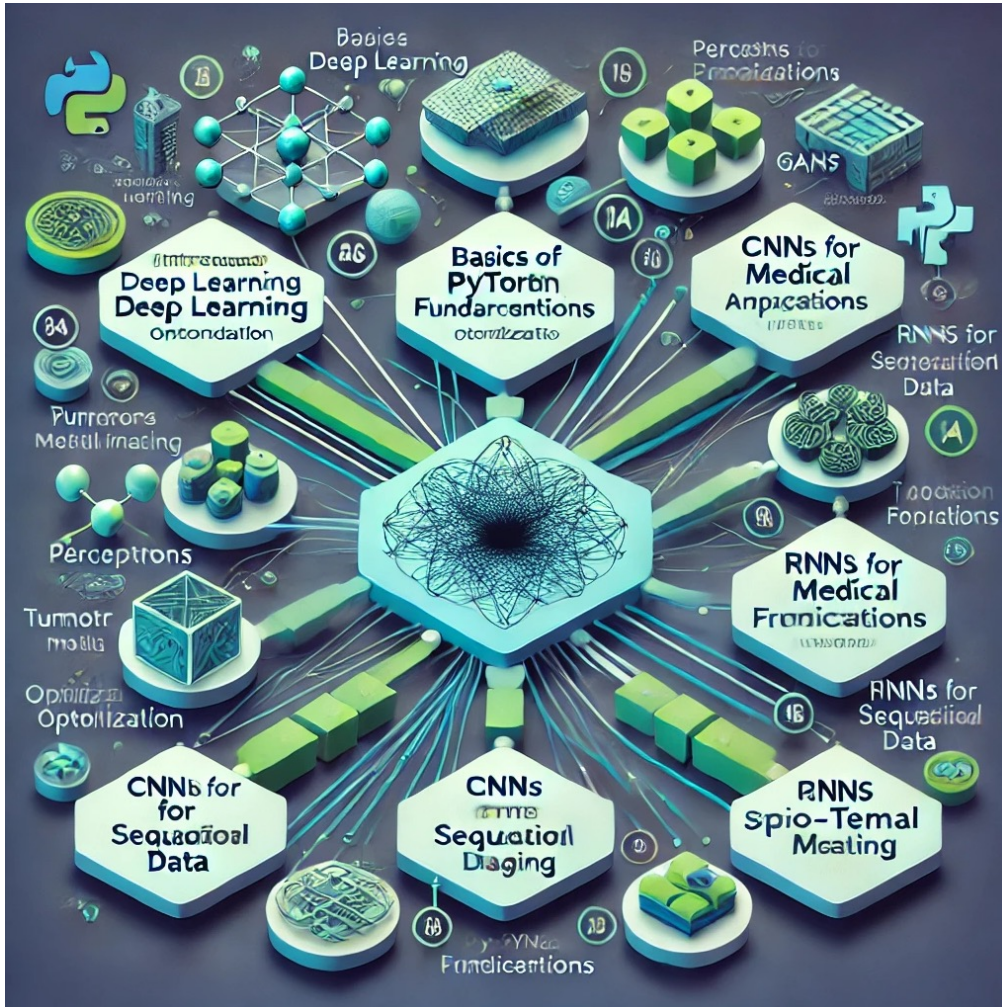
<https://bios740.github.io/>

BIOS740	
Syllabus	
Fundamentals of Deep Learning	
Lec1:	Introduction to Deep Learning, PyTorch & Basic Algorithms
Lec2:	Neural Networks Fundamentals
Basic Network Structures	
Lec3:	Convolutional Neural Networks (CNN) HW 1
Lec4:	Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) HW 2 HW 1 DUE
Lec5:	Graph Neural Networks: GNN, GCN HW 3 HW 2 DUE
Lec6:	Generative Adversarial Networks (GAN) HW 4 HW 3 DUE

<https://bios740.github.io/short/jsm2025>

BIOS740	
Short Courses / JSM2025	
	Deep Learning Methods in Advanced Statistical Problems -- JSM 2025 Short Course Nashville, Tennessee August 3, 2025
Introduction This short course is designed for researchers in statistics and data analysis who are eager to explore the latest trends in deep learning and apply these methods to solve complex statistical problems. The course delves into the intersection of deep learning and statistical analysis, covering topics familiar to statisticians such as time series analysis, survival analysis, and quantile regression. Additionally, it addresses cutting-edge topics in the deep learning community, including transformers, diffusion models, and large language models. In this one-day short	

Key Modules



Introduction: Basics of deep learning, supervised/unsupervised learning, and PyTorch fundamentals.

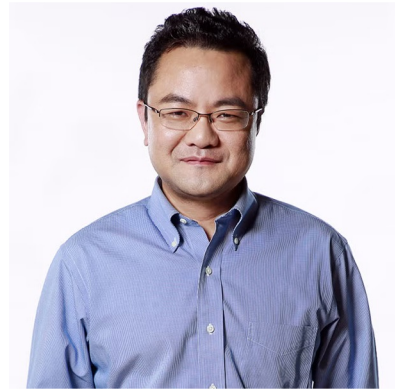
1. Neural Networks: Perceptrons, optimization techniques, and activation functions.

2. Advanced Topics:

- CNNs.
- GNNs/GCNs
- RNNs and LSTMs
- GANs/ Diffusion Models
- Transformers
- BioBERT.

3. Applications: Segmentation, Registration, Tumor localization, Disease spread prediction, Biomedical text mining, and Drug discovery.

Foundations of Deep Learning Methods



Dr. Hongtu Zhu
Kenan Distinguished Professor
University of North Carolina at Chapel Hill
URL: www.med.unc.edu/bigs2/

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

5 Optimization Techniques

6 Convolutional Neural Networks (CNN)

7 Graph Neural Networks (GNNs/GCNs)

8 Theoretical Properties

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

5 Optimization Techniques

6 Convolutional Neural Networks (CNN)

7 Graph Neural Networks (GNNs/GCNs)

8 Theoretical Properties

Deep Learning

Deep

Many hidden layers

Learning

Supervised, semi-supervised, unsupervised
Learn adaptive parameters

- Use a cascade of multiple layers of nonlinear processing units for feature extract and transformation
- Learn in supervised and/or unsupervised manner
- Learn representations in different level of abstraction

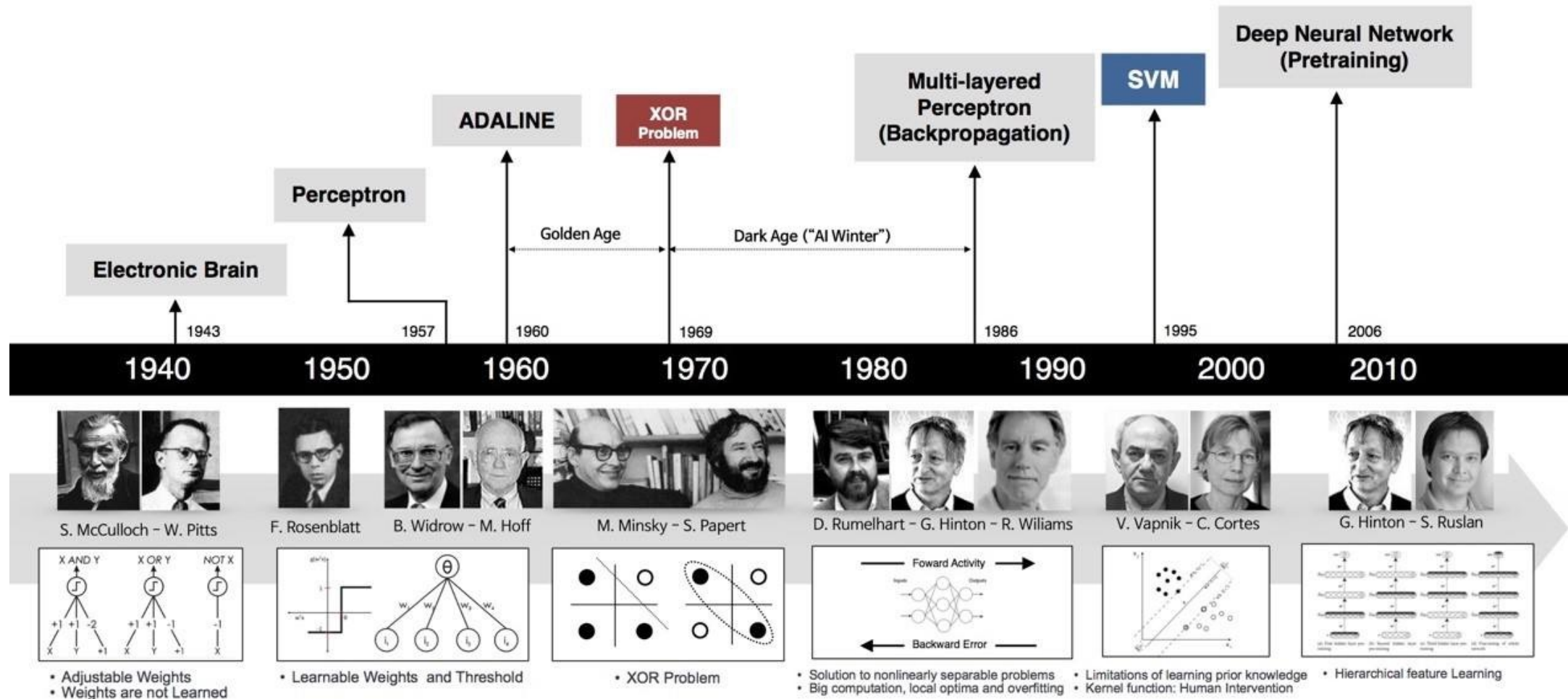
Why popular?

- Chip processing ability
- Increased size of data for training
- Advances in machine learning and signal/information researches

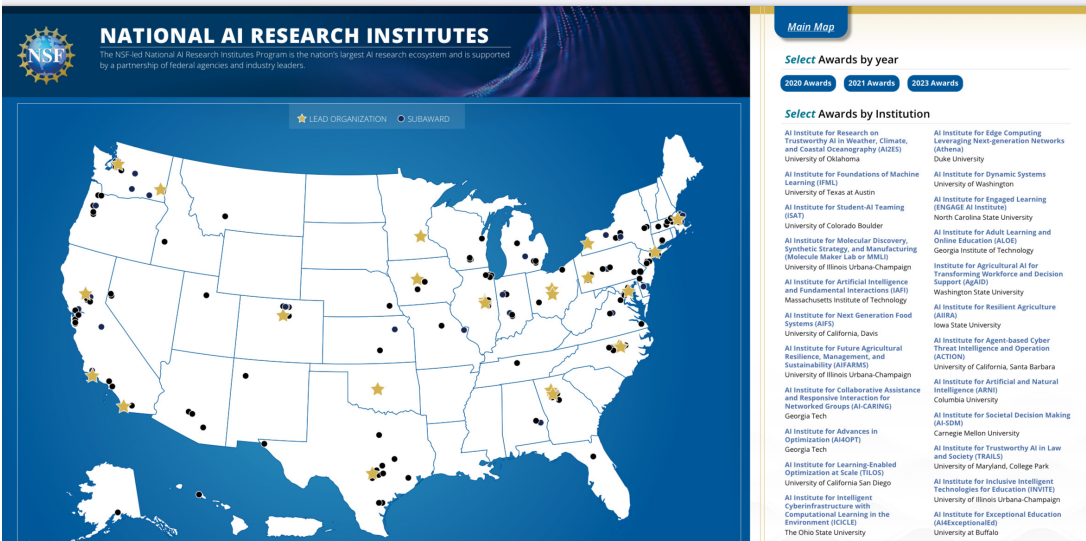


Deep models to efficiently exploit **complex, compositional nonlinear** functions to learn **distributed and hierarchical feature representations**, to make best use

Historical Summary



Deep Learning Explosion



Feature

THE MOST-CITED PAPERS OF THE TWENTY-FIRST CENTURY

An exclusive *Nature* analysis reveals the 25 highest-cited papers published this century and explores why they are breaking records. By Helen Pearson, Heidi Ledford, Matthew Hutson and Richard Van Noorden



1	Title
2	
3	Deep Residual Learning for Image Recognition
4	Analysis of Relative Gene Expression Data Using Real-Time Quantitative PCR and the 2-ΔΔCT Method
5	Using thematic analysis in psychology
6	Diagnostic and Statistical Manual of Mental Disorders, DSM-5
7	A short history of SHELX
8	Random Forests
9	Attention is all you need
10	ImageNet classification with deep convolutional neural networks
11	Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for Selected Cancers
12	Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for selected sites and causes
13	Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement
14	U-Net: Convolutional Networks for Biomedical Image Segmentation
15	Electric Field Effect in Atomically Thin Carbon Films
16	Fitting Linear Mixed-Effects Models Using lme4
17	Scikit-learn: Machine learning in Python
18	Deep learning
19	Common Method Biases in Behavioral Research: A Critical Review of the Literature and Recommendations for Avoidance
20	Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2

Deep Learning Platforms

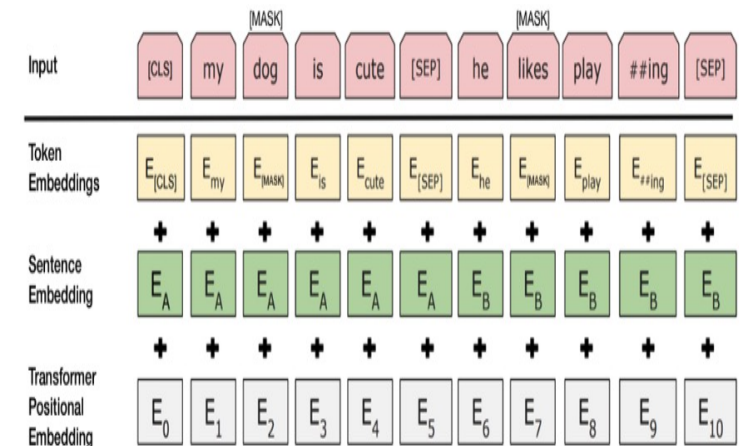
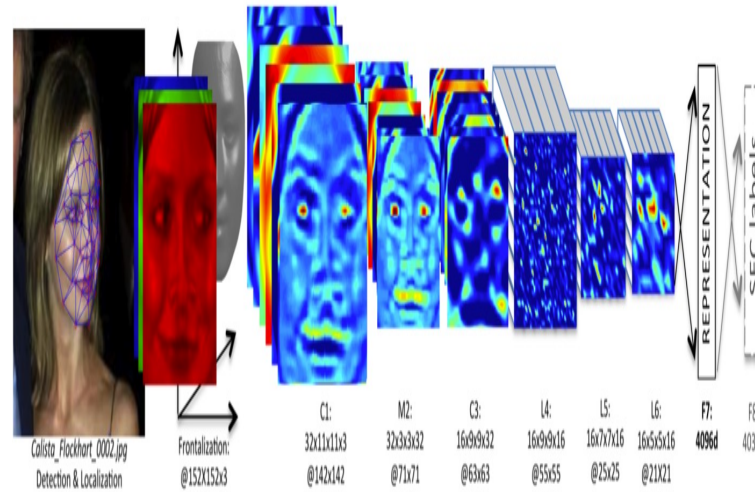


theano



Applications - Vision

IMAGENET



Applications - Vision



Disease Detection in Healthcare and Medicine

Deep learning can be utilized for early and more accurate detection of diseases like cancer, Alzheimer's, and heart diseases through image analysis.

High quality image generalization

DALL·E 2

“a teddy bear on a skateboard in times square”

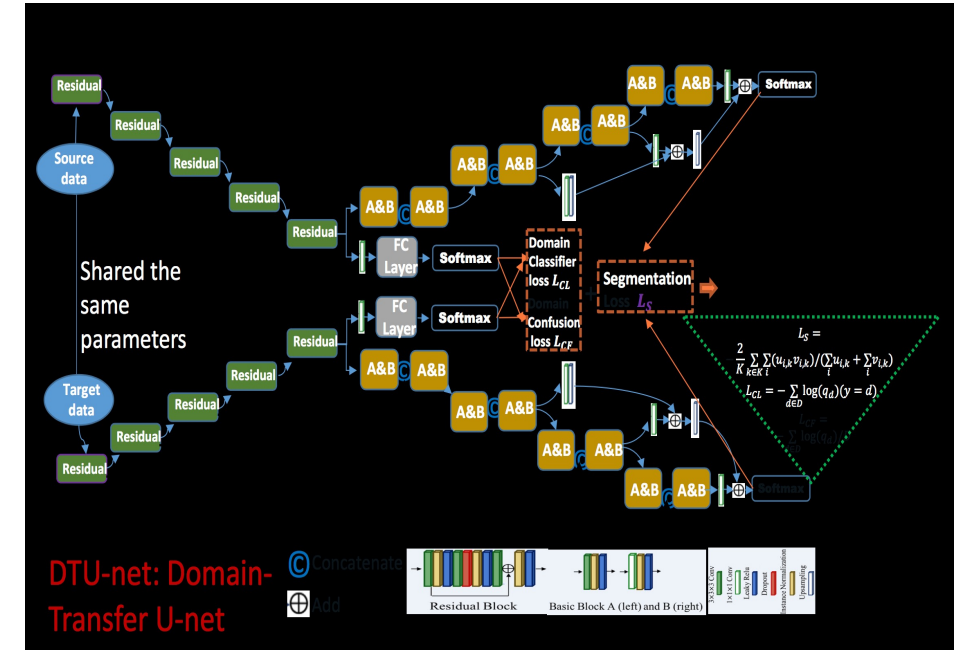


“Hierarchical Text-Conditional Image Generation with CLIP Latents”
Ramesh et al., 2022

Applications – Medical Imaging

Segmentation Annotation

U-Nets



Liu, Q., Xu, Z., Bertasius, G., & Niethammer, M. (2023). SimpleClick: Interactive Image Segmentation with Simple Vision Transformers. ICCV, 22290-22300. 2023.

Azad *et al.*, “Medical Image Segmentation Review: The success of U-Net.” arXiv, Nov. 27, 2022.
Minaee, Shervin, et al. "Image segmentation using deep learning: A survey." *IEEE PAMI* 44.7 (2021): 3523-3542

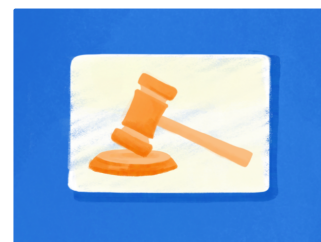
Application - Language

Language Translation in Natural Language Processing

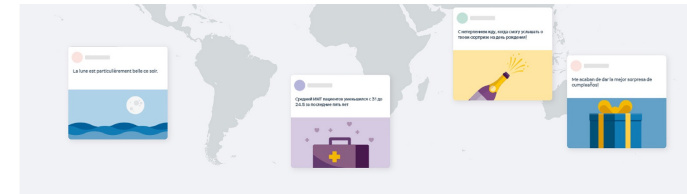
Deep learning enhances real-time, accurate translation of languages, as seen in tools like Google Translate. The following picture shows the translation of a webpage from English to Chinese.



- Facebook AI is introducing M2M-100, the first multilingual machine translation (MMT) model that can translate between any pair of 100 languages without relying on English data. It's open sourced [here](#).
- When translating, say, Chinese to French, most English-centric multilingual models train on Chinese to English and English to French, because English training data is the most widely available. Our model directly trains on Chinese to French data to better preserve meaning. It outperforms English-centric systems by 10 points on the widely used BLEU metric for evaluating machine translations.
- M2M-100 is trained on a total of 2,200 language directions — or 10x more than previous best, English-centric multilingual models. Deploying M2M-100 will improve the quality of translations for billions of people, especially those that speak low-resource languages.
- This milestone is a culmination of years of Facebook AI's foundational work in machine translation. Today, we're sharing details on how we built a more diverse MMT training data set and model for 100 languages. We're also [releasing the model, training, and evaluation setup](#) to help other researchers reproduce and further advance multilingual models.



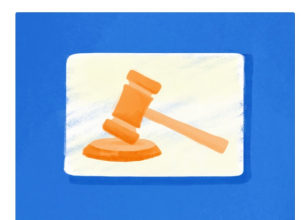
Meta
Meta and Christian Louboutin File
Joint Lawsuit Against Counterfeiter
November 16, 2023



- Facebook AI 正在推出 M2M-100，这是第一个多语言机器翻译 (MMT) 模型，可以在 100 种语言中的任意对之间进行翻译，而无需依赖英语数据。[这里是开源的](#)。
- 例如，在将中文翻译成法语时，大多数以英语为中心的多语言模型都会在中文到英语和英语到法语上进行训练，因为英语训练数据是最广泛可用的。我们的模型直接对中文到法语的数据进行训练，以更好地保留含义。在广泛使用的用于评估机器翻译的 BLEU 指标上，它比以英语为中心的系统高出 10 个点。
- M2M-100 接受了总共 2,200 种语言方向的训练，比以前最好的、以英语为中心的多语言模型多了 10 倍。部署 M2M-100 将为数十亿人提高翻译质量，尤其是那些使用资源匮乏语言的人。
- 这一里程碑是 Facebook AI 多年来在机器翻译领域基础工作的结晶。今天，我们将分享如何为 100 种语言构建更加多样化的 MMT 训练数据集和模型的详细信息。我们还发布了[模型、训练和评估设置](#)，以帮助其他研究人员重现和进一步推进多语言模型。



元
支持独立研究的新工具
2023 年 11 月 21 日



元
Meta 和 Christian Louboutin 对仿冒者提

Application - Language

ChatGPT		
Examples	Capabilities	Limitations
"Explain quantum computing in simple terms" →	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Get any creative ideas for a 10 year old's birthday?" →	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?" →	Trained to decline inappropriate requests	Limited knowledge - all models are trained after 2021

Large language models

Large language models can perform various tasks such as answering questions, generating creative content, summarizing text, translating languages, and engaging in conversations. It's designed to understand and generate text in a coherent and contextually relevant manner.

Application - Decision



March 2016, AlphaGo made headlines by defeating Lee Sedol.

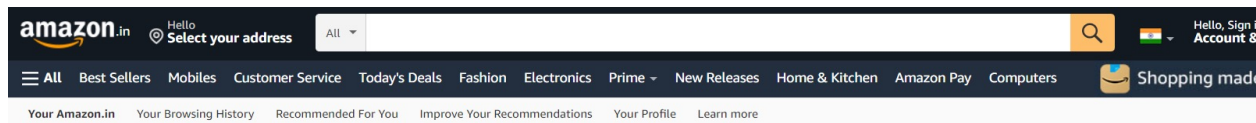


Reinforcement learning methods have shown priority in video games.













Applications - more

Personalized Shopping Experience in Retail and E-Commerce

Deep learning is leveraged to provide personalized recommendations and targeted advertising to customers based on their shopping behavior.

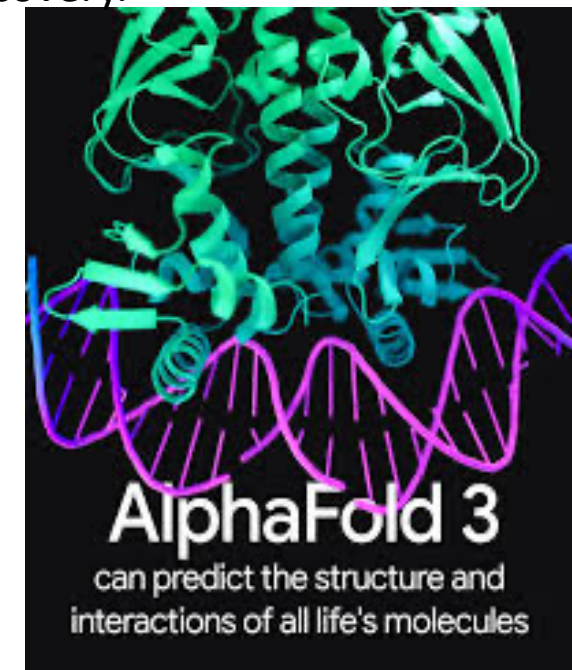


Top picks for you

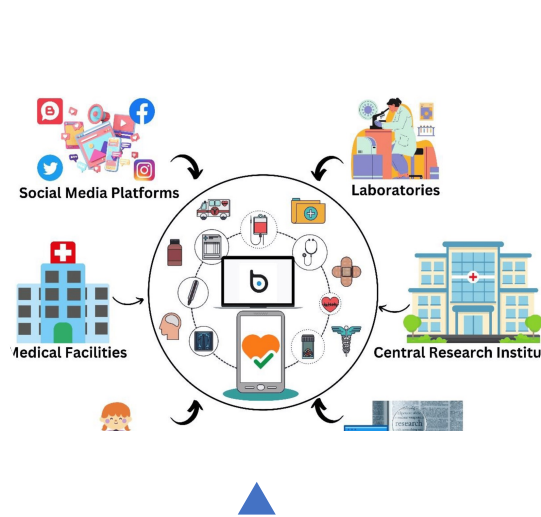
 boAt Airdopes 141 True Wireless Earbuds with 42H Playtime... ★★★★☆ 58,038 ₹1,499.00 Get it by Saturday, April 23	 Redmi 9A Sport (Coral Green, 2GB RAM, 32GB Storage) 2GHz... ★★★★☆ 159,691 ₹6,999.00 Get it by Saturday, April 23	 OnePlus Nord CE 2 5G (Bahamas Blue, 8GB RAM, 128GB Storage) ★★★★☆ 12,639 ₹24,999.00 Get it by Saturday, April 23	 boAt Airdopes 121v2 True Wireless Earbuds with Upto 14... ★★★★☆ 116,391 ₹1,299.00 Get it by Saturday, April 23	 APLUS Super Saver Wheat Flour (Maida) 500 g ★★★★☆ 489 ₹30.00	 Mattel Uno Playing Card Game ★★★★☆ 32,216 ₹104.00 Get it by Saturday, April 23
 Tata Sampann Unpolished Toor Dal/Arhar Dal, 1kg ★★★★☆ 12,726 ₹135.00 - ₹189.00	 Maggi 2-Minute Noodles Masala, 70g (Pack of 12) ★★★★☆ 47,101 ₹136.00 - ₹330.00	 Fortune Premium Kachi Ghani Pure Mustard Oil, 1tr PET Bottle ★★★★☆ 13,858 ₹205.00 - ₹214.00	 Happilo 100% Natural Premium California Almonds 1kg Value Pack Pouch High Quality... ★★★★☆ 19,151	 Glucon-D Glucon D Instant Energy Health Drink Nimbu Pani - 1kg... ★★★★☆ 4,335 ₹309.00	 boAt Bassheads 100 in Ear Wired Earphones with Mic(Black) ★★★★☆ 317,716 ₹399.00

Predict the folding and 3D structure of protein

AlphaFold aims to solve the protein folding problem, which involves predicting a protein's three-dimensional structure based solely on its amino acid sequence. Understanding protein structures is crucial for biological research and drug discovery.

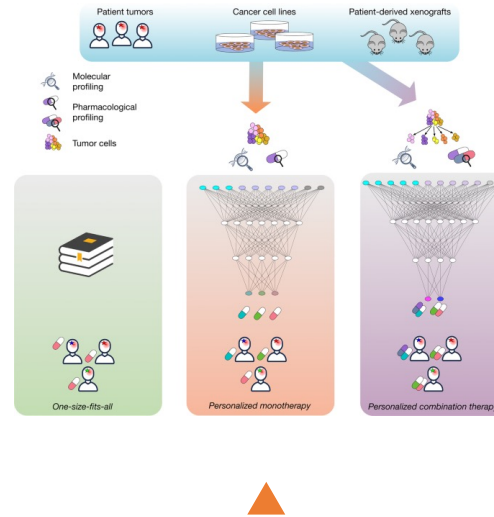


Some Future Directions in DL for Biostatistics



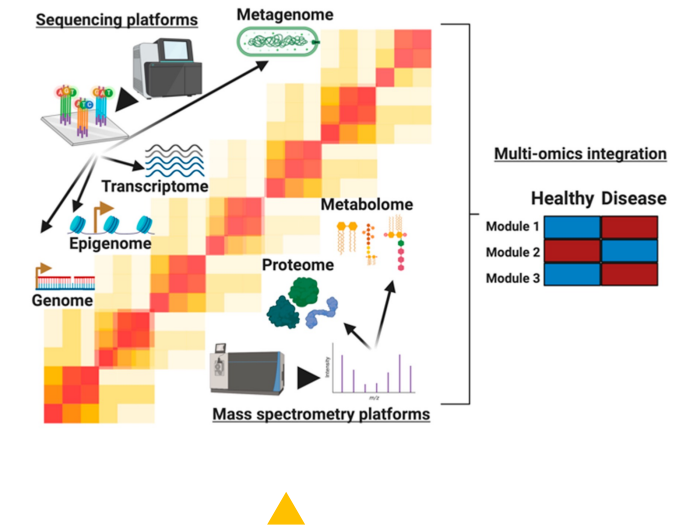
AI-driven Public Health Interventions

Utilizing deep learning models to analyze large-scale public health data for informed decision-making and policy development. Allowing better resource allocation, and more effective epidemic control strategies.



Advanced Drug Response Modeling

Using deep learning to model and predict individual responses to drugs, considering genetic, environmental, and lifestyle factors. Developing more effective personalized treatments.



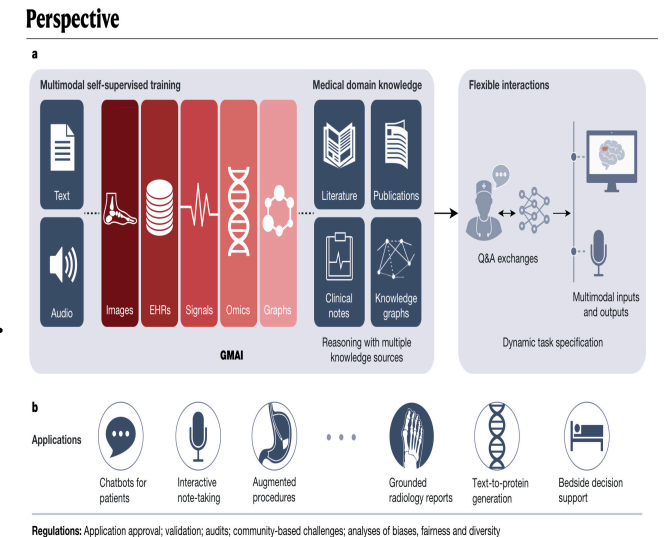
Integrative Analysis of Multi-omic Data

Leveraging deep learning to integrate and analyze data from genomics, proteomics, metabolomics, and other omics fields for a comprehensive understanding of biological processes and disease mechanisms.

Generalist Medical Artificial Intelligence

- **Foundation Models in Medicine:** These models leverage large-scale datasets and generalizable architectures to address diverse medical tasks, moving beyond task-specific AI systems.
- **Generalist AI:** Unlike traditional models, foundation models aim to function across multiple domains, such as imaging, text, and genomics, enabling integration of multimodal data for holistic medical insights.
- **Challenges:**
 - Data heterogeneity: Medical data comes in varied formats, requiring harmonization.
 - Privacy and ethics: Ensuring secure, unbiased AI while maintaining patient confidentiality.
 - Interpretability: Providing clinicians with actionable insights from AI outputs.
- **Applications:**
 - Diagnostics: Detecting diseases across imaging modalities (e.g., radiology).
 - Prognostics: Predicting patient outcomes using integrated data.
 - Personalized medicine: Tailoring treatments based on multimodal patient profiles.
- **Future Directions:**
 - Collaboration between AI experts and clinicians to co-design models.
 - Development of robust validation frameworks for clinical adoption.
 - Advancing explainability and trust in AI-driven medical decisions.

Moor, M.,, Rajpurkar, P. (2023) *Nature*.



Regulations: Application approval; validation; audits; community-based challenges; analyses of biases, fairness and diversity

Fig. 1 | Overview of a GMAI model pipeline. **a.** A GMAI model is trained on multiple medical data modalities, through techniques such as self-supervised learning. To enable flexible interactions, data modalities such as images or data from EHRs can be paired with language, either in the form of text or speech data. Next, the GMAI model needs to access various sources of medical knowledge to carry out medical reasoning tasks, unlocking a wealth of capabilities that can be used in downstream applications. The resulting GMAI model then carries out tasks that the user can specify in real time. For this, the GMAI model can retrieve contextual information from sources such as knowledge graphs or databases, leveraging formal medical knowledge to reason about previously unseen tasks. **b.** The GMAI model builds the foundation for numerous applications across clinical disciplines, each requiring careful validation and regulatory assessment.

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

5 Optimization Techniques

6 CNNs

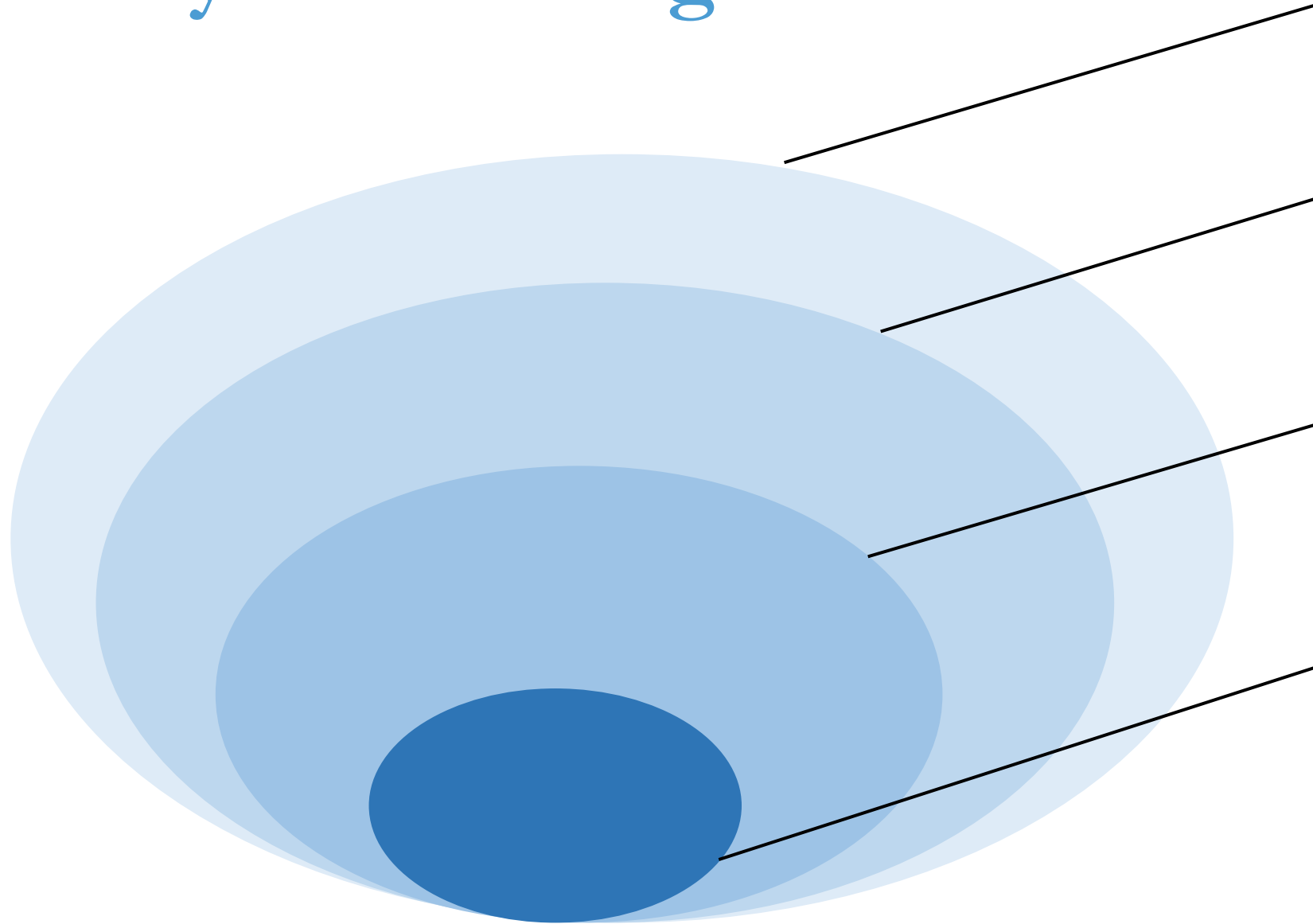
7 GNNs/GCNs

8 Theoretical Properties



What Exactly is Deep Learning?

Key Terminologies



Artificial Intelligence

Simulates human intelligence in machines for tasks like decision-making and language translation.

Machine Learning (ML)

A subset of AI where algorithms learn from data to make predictions or decisions without being explicitly programmed for each scenario.

Deep Learning (DL)

A branch of machine learning using multi-layered neural networks, effective in processing large amounts of unstructured data like images and speech.

Generative AI

AI algorithms that generate new, original content (like text or images) based on existing data, using techniques like Generative Adversarial Networks (GANs).

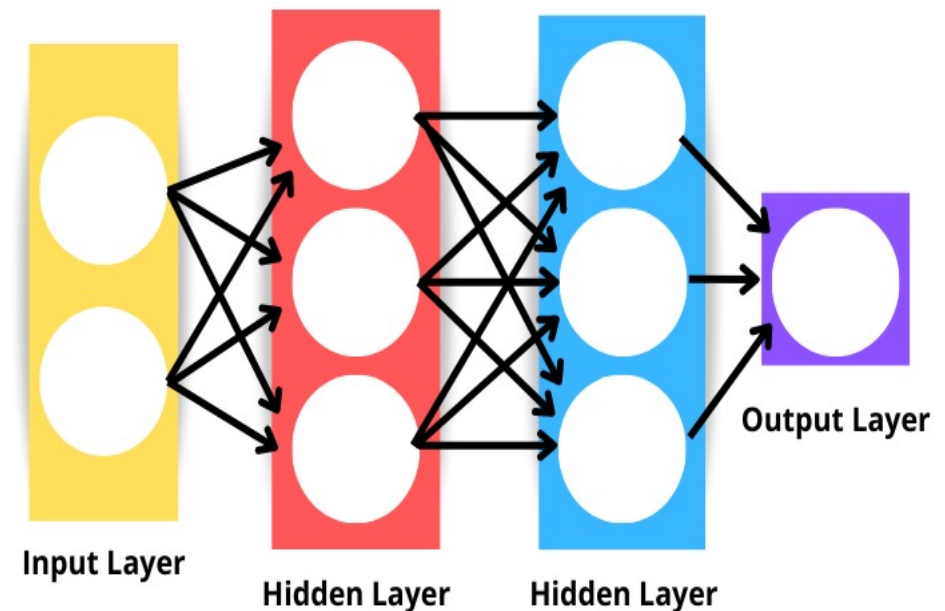
Deep Learning

- Deep learning is a subset of machine learning that focuses on training **algorithmic neural networks** to perform tasks. Its algorithms were inspired by the working of the human brain.
- It's characterized by the use of **multiple layers (deep architectures)** that allow networks to learn hierarchical representations of data and to learn to complete specific tasks.
- In contrast to traditional machine learning/data models, which often requires **manual feature extraction**, deep learning can **automatically learn features from raw data**, which you can think of as patterns that occur within the data.
- Deep learning can be used for supervised, unsupervised, self-supervised, semi-supervised, generative, contrastive, few-shot, as well as reinforcement learning.

Objective: teaching computer how to learn a task directly from raw data

Backbone of DL - Neural Networks

- Neural networks, also called artificial neural networks (ANNs) or simulated neural networks (SNNs), are a **subset of machine learning** and are the **backbone of deep learning algorithms**.
- The neural network is inspired by the human brain's interconnected neurons. They are called “neural” because they mimic how neurons in the brain signal one another.
- It consists of layers: an **input layer**, one or more **hidden layers**, and an **output layer**.
- The “**deep**” in deep learning refers to the depth of layers in a neural network.
- Usually, a neural network of more than three layers, including the inputs and the output, can be considered a deep-learning algorithm.



Further details on neural networks will be in upcoming courses.

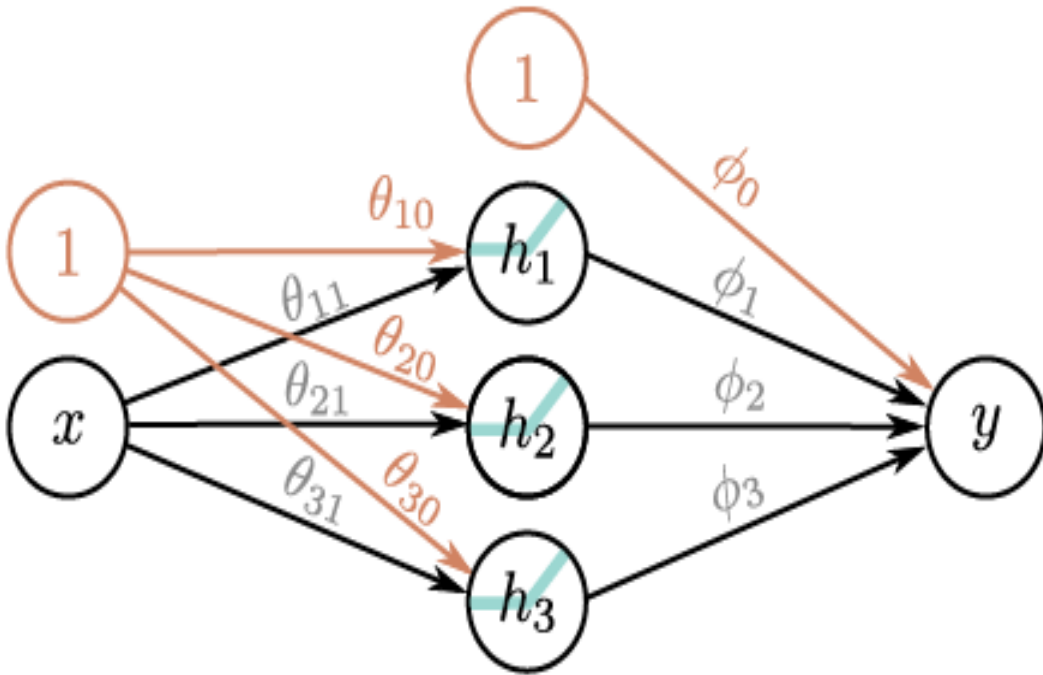
Deep Learning Basics

Neurons (Nodes) receive input signals and perform computations and produce an output.

Channels (connections) are associated with a **weight** value that determines the strength of the connection.

Bias is conceptually similar to the intercept in linear regression, accounting for potential deviations from the ideal relationship between inputs and outputs.

Activation function are threshold values that introduce non-linearities into the neural network, determining if the particular neuron will get activated or not.

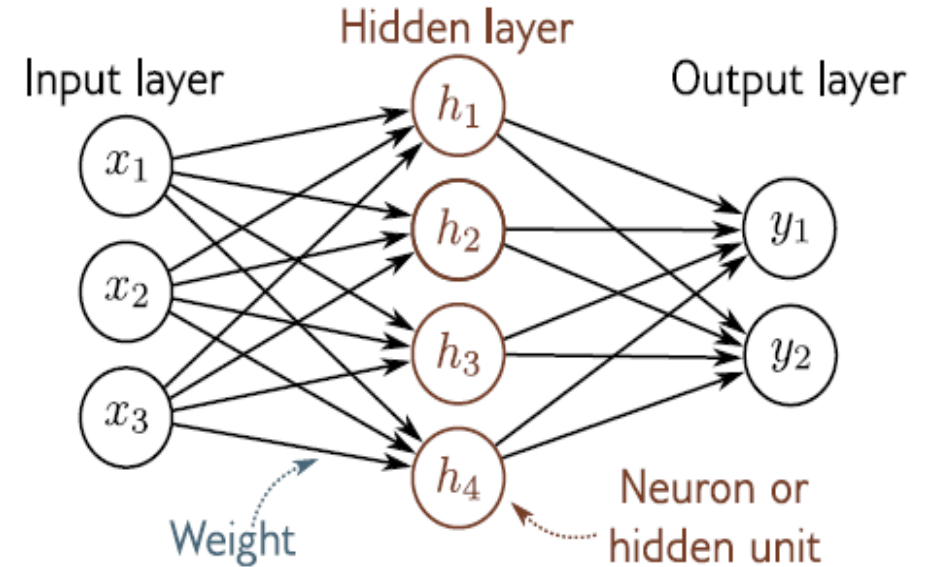


Shallow Neural Network

Universal Approximation Theorem

Cybenko (1989) and Hornik (1991)

A feed-forward network with a **single hidden layer** containing a finite number of neurons can approximate continuous functions on compact subsets of \mathbb{R}^n , under mild assumptions on the activation function.



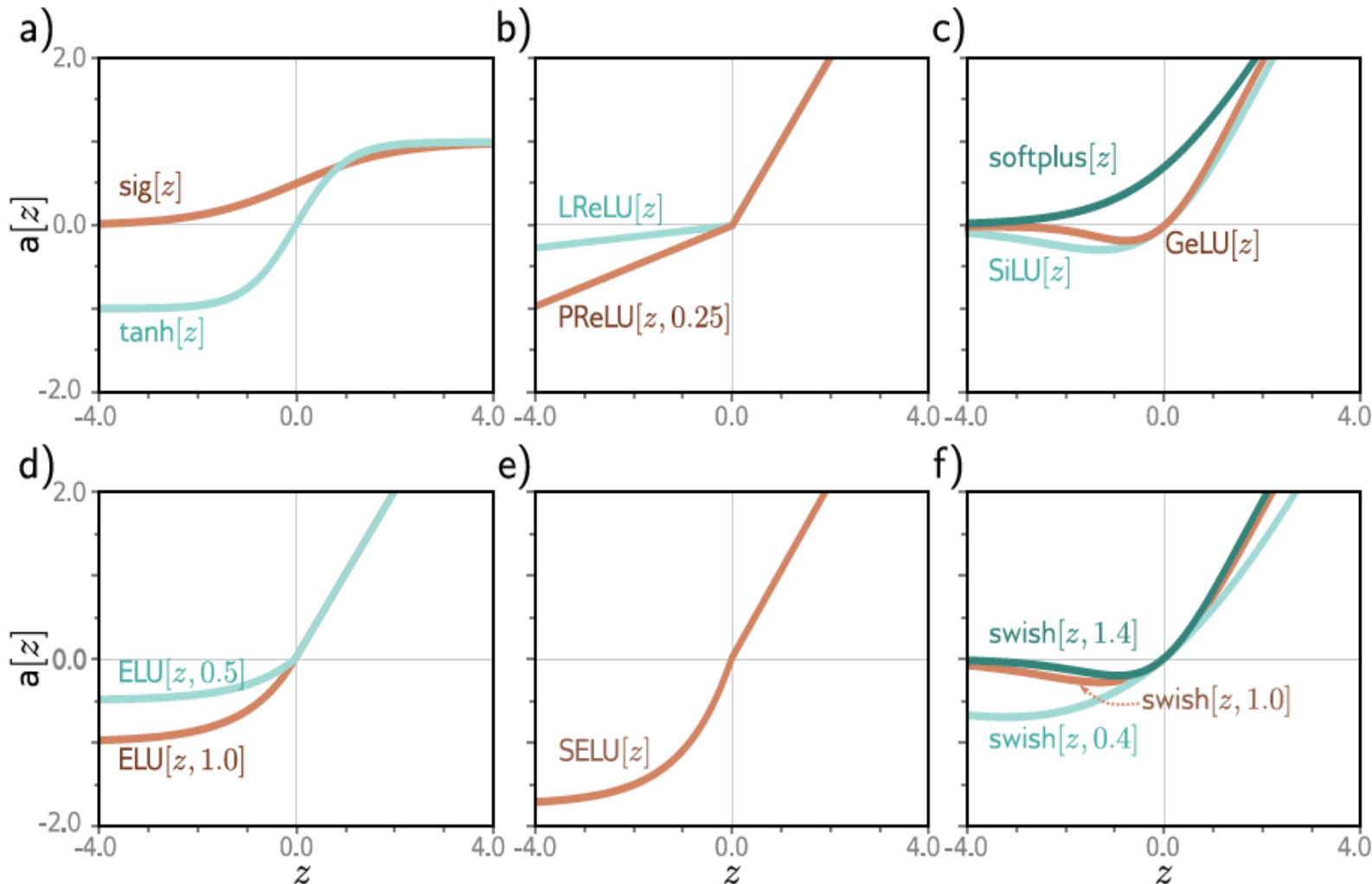
$$h_d = a \left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i \right],$$

$$y_j = \phi_{j0} + \sum_{d=1}^D \phi_{jd} h_d,$$

Activation Function - The Gateway to Non-Linearity

- **Introducing Non-Linearity:** Activation functions **introduce non-linear properties to the network**, enabling it to learn complex data patterns beyond the capability of linear models.
- **Transforming Inputs to Outputs:** It takes input from previous layers and converts it to some form of input for the next layers.
- **Essential Building Blocks:** It decides what is to be fired to the next neuron.
- **Beyond Linear Modeling:** Without non-linearity, neural networks would be limited to linear decision boundaries, similar to linear regression.
- **Crucial for Performance:** Non-linear functions allow neural networks to solve advanced problems like image and speech recognition, and natural language processing.

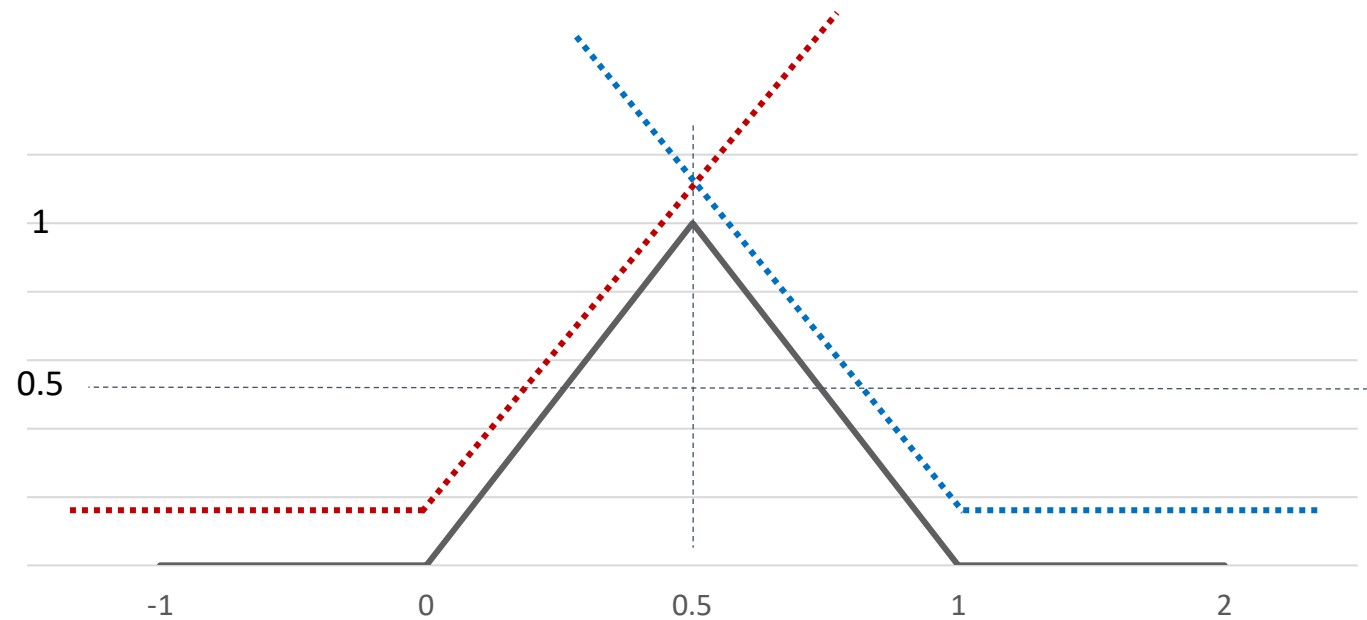
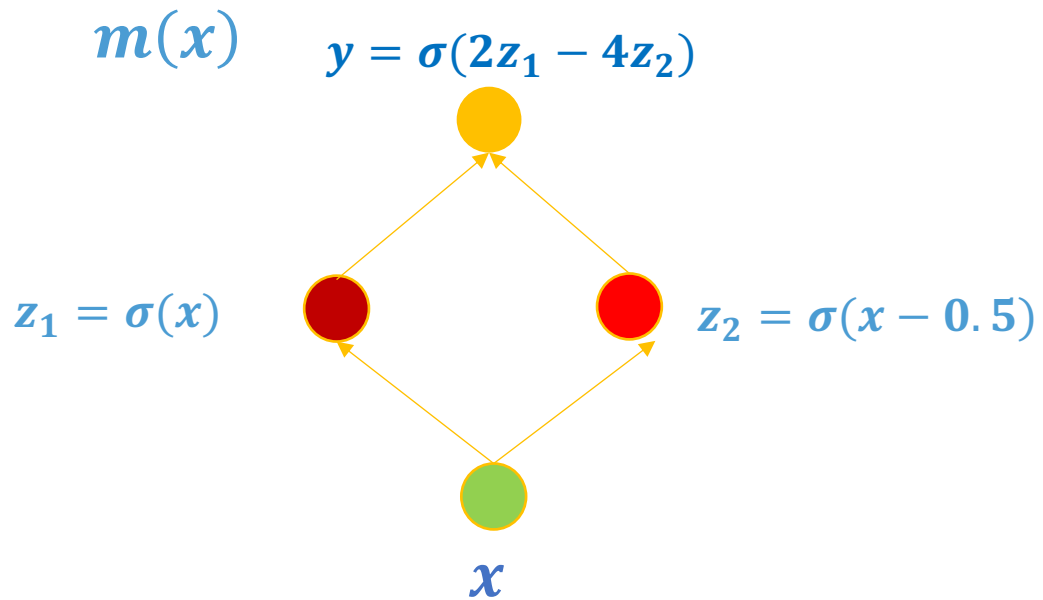
Activation Functions



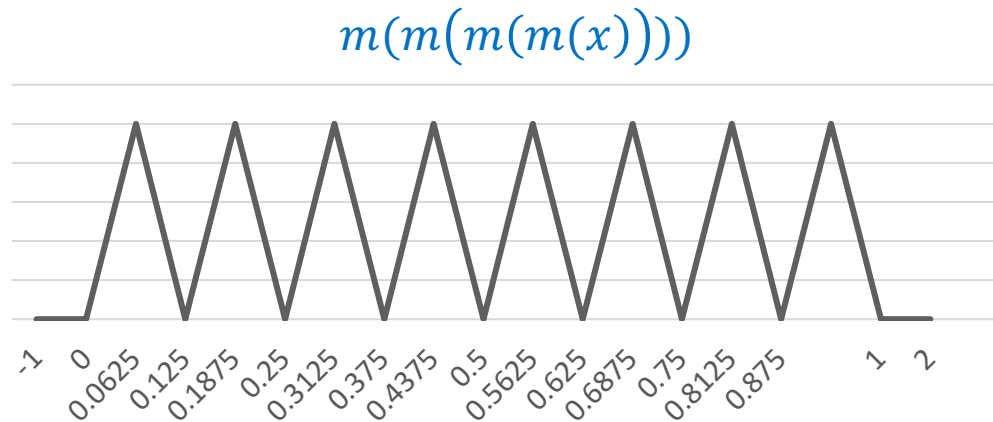
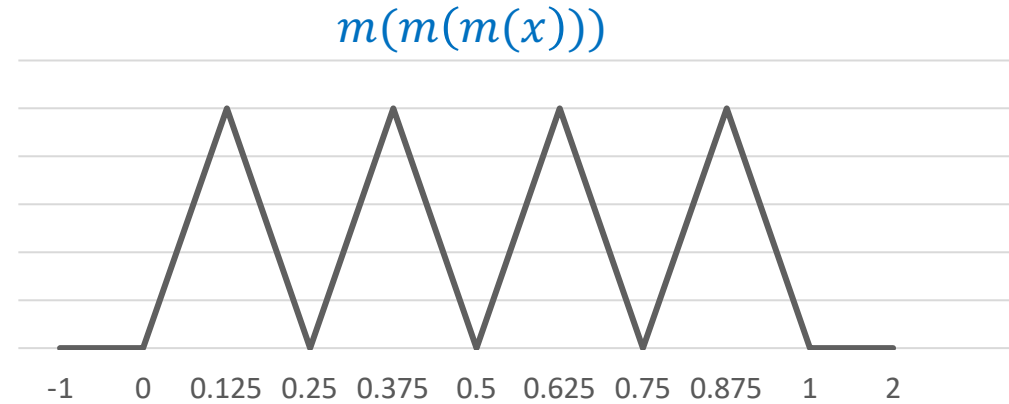
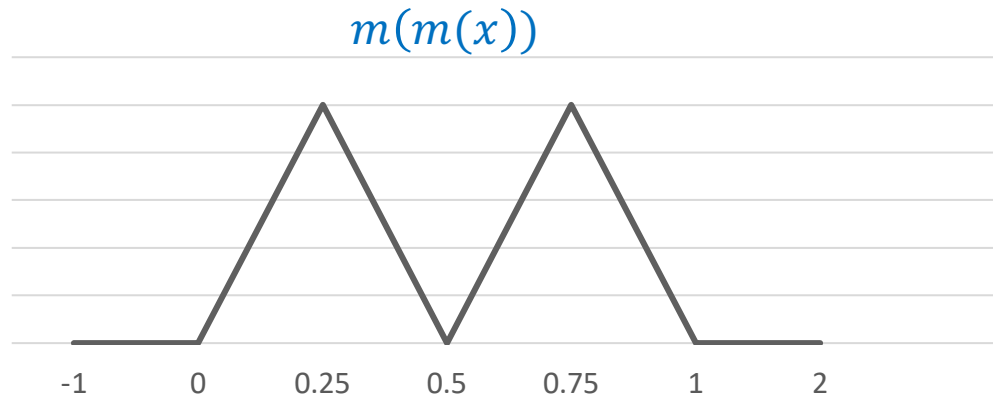
- a) Logistic sigmoid and tanh functions.
- b) Leaky ReLU and parametric ReLU with parameter 0.25.
- c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit.
- d) Exponential linear unit with parameters 0.5 and 1.0.
- e) Scaled exponential linear unit.
- f) Swish with parameters 0.4, 1.0, and 1.4.

Motivation for Deep Learning

Consider a piecewise linear function $m(x) = \begin{cases} 2x, & x \in [0, 0.5] \\ 2 - 2x, & x \in [0.5, 1] \\ 0 & \text{otherwise} \end{cases}$ Define $\sigma(x) = \max(0, x)$

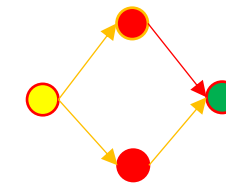


Motivation for Deep Learning



Generate a deep network:

$3n+1$ nodes to represent the $m^{(n)}(x)$
with width of each layer ≤ 2

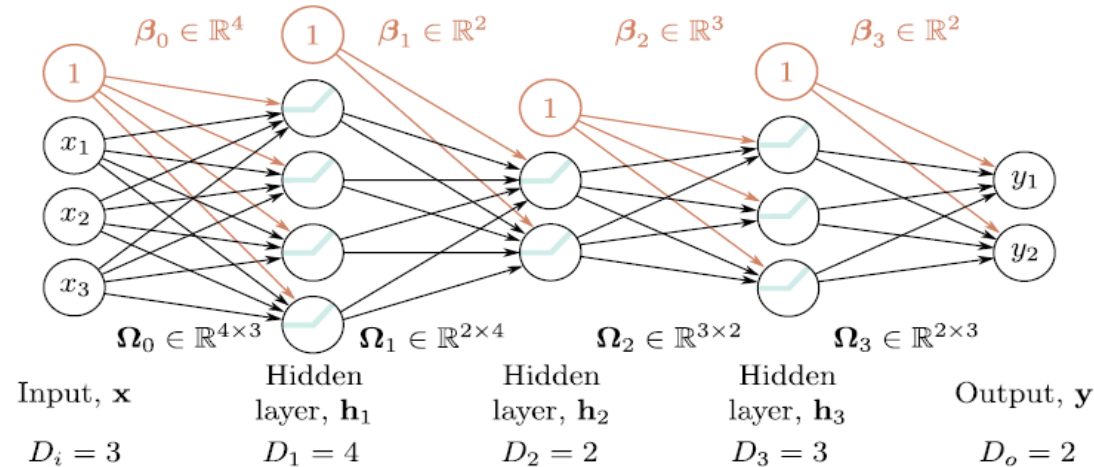


Depth
- Multiplicatively

If we generate a shallow one, we need 2^n nodes

Width
- additively

Deep Neural Network



Shallow vs deep networks

- ❖ both networks can approximate any function given enough capacity,
- ❖ deep networks produce many more linear regions per parameter,
- ❖ some functions can be approximated much more efficiently by deep networks,
- ❖ in practice, the best results for most tasks are achieved using deep networks with many layers.

$$\mathbf{y} = \beta_K + \Omega_K \mathbf{a} [\beta_{K-1} + \Omega_{K-1} \mathbf{a} [\dots \beta_2 + \Omega_2 \mathbf{a} [\beta_1 + \Omega_1 \mathbf{a} [\beta_0 + \Omega_0 \mathbf{x}]] \dots]] .$$

- ❖ The number of hidden units in each layer is referred to as the **width** of the network, and the number of hidden layers as the **depth**. The total number of hidden units is a measure of the network's **capacity**.
- ❖ The **depth version** of the universal approximation theorem (Lu et al., 2017): There exists a network with ReLU activation functions and at least D_i+4 hidden units in each layer can approximate any specified D_i -dimensional Lebesgue integrable function to arbitrary accuracy given enough layers.

$$\mathbf{h}_1 = \mathbf{a}[\beta_0 + \Omega_0 \mathbf{x}]$$

$$\mathbf{h}_2 = \mathbf{a}[\beta_1 + \Omega_1 \mathbf{h}_1]$$

$$\mathbf{h}_3 = \mathbf{a}[\beta_2 + \Omega_2 \mathbf{h}_2]$$

$$\vdots$$

$$\mathbf{h}_K = \mathbf{a}[\beta_{K-1} + \Omega_{K-1} \mathbf{h}_{K-1}]$$

$$\mathbf{y} = \beta_K + \Omega_K \mathbf{h}_K.$$

Fitting DL Models

AS=Applied Statistics

Define a set of functions/models



Find a criterion/measurement of goodness –
loss(+ regularization)



Get the best model for the problem

AI=Artificial Intelligence

Design the neural network

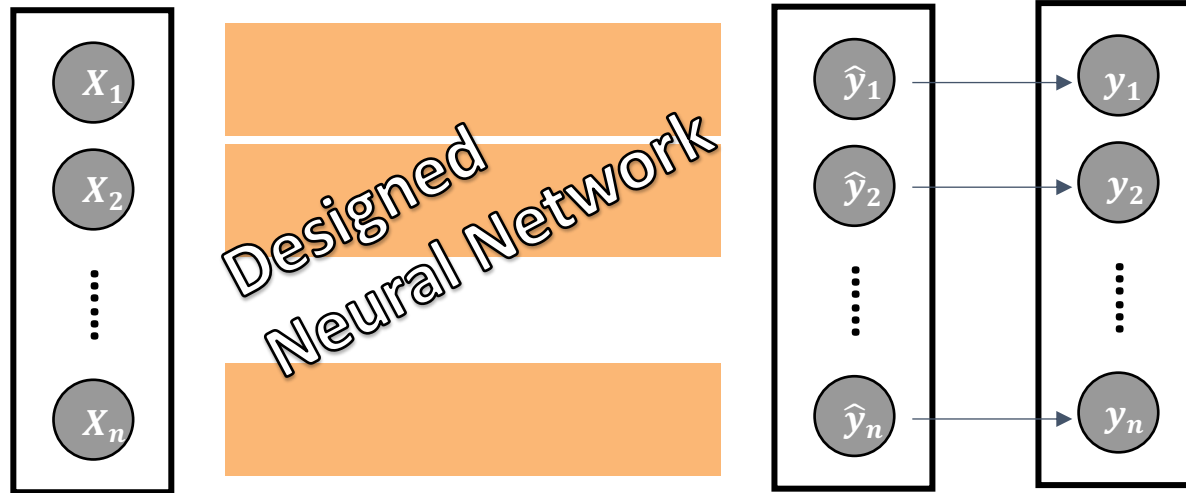


Find a criterion/measurement of goodness –
loss(+ regularization)



Get the best model for the problem

Fitting DL Models



$$\begin{aligned}
 \mathbf{h}_1 &= \mathbf{a}[\beta_0 + \Omega_0 \mathbf{x}] \\
 \mathbf{h}_2 &= \mathbf{a}[\beta_1 + \Omega_1 \mathbf{h}_1] \\
 \mathbf{h}_3 &= \mathbf{a}[\beta_2 + \Omega_2 \mathbf{h}_2] \\
 &\vdots \\
 \mathbf{h}_K &= \mathbf{a}[\beta_{K-1} + \Omega_{K-1} \mathbf{h}_{K-1}] \\
 \mathbf{y} &= \beta_K + \Omega_K \mathbf{h}_K.
 \end{aligned}$$

A **loss function** is needed here,
to measure the difference between the output and truth

Total loss:
$$L = \sum \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$$

$$\hat{\mathbf{y}}_i = \beta_K + \Omega_K \mathbf{h}_K(\mathbf{x}_i; [(\beta_0, \Omega_0), \dots, (\beta_{K-1}, \Omega_{K-1})])$$

Find the network parameters to minimize the loss

Design the neural network

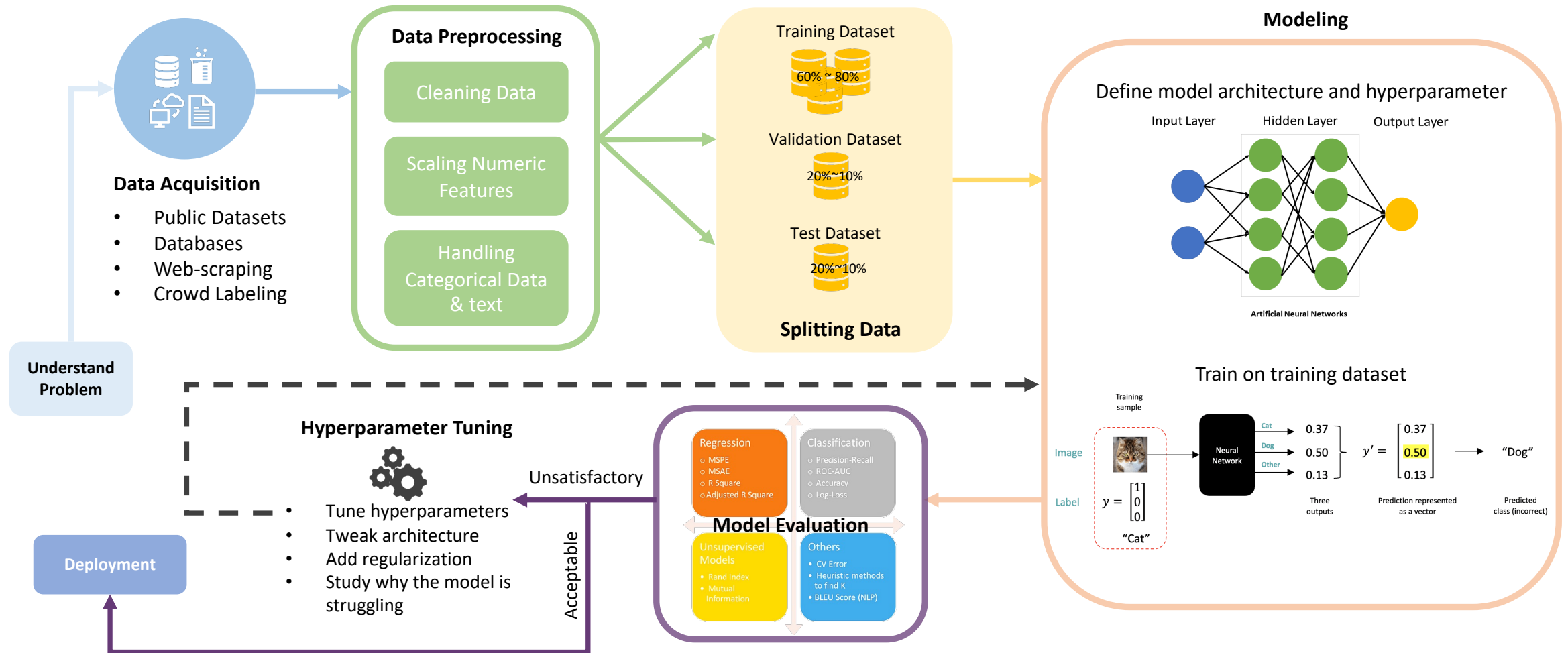


Find a criterion/measurement of goodness



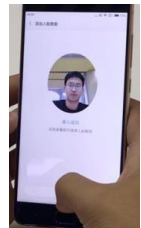
Get the best model for the problem

Workflow of a Typical DL Project



Face Recognition System

Image Acquisition



Face Recognition

Face Detection



Feature Extraction



Face Matching



Recognition Results

Model Training



Gabor, LBP etc.



Deep Learning



Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

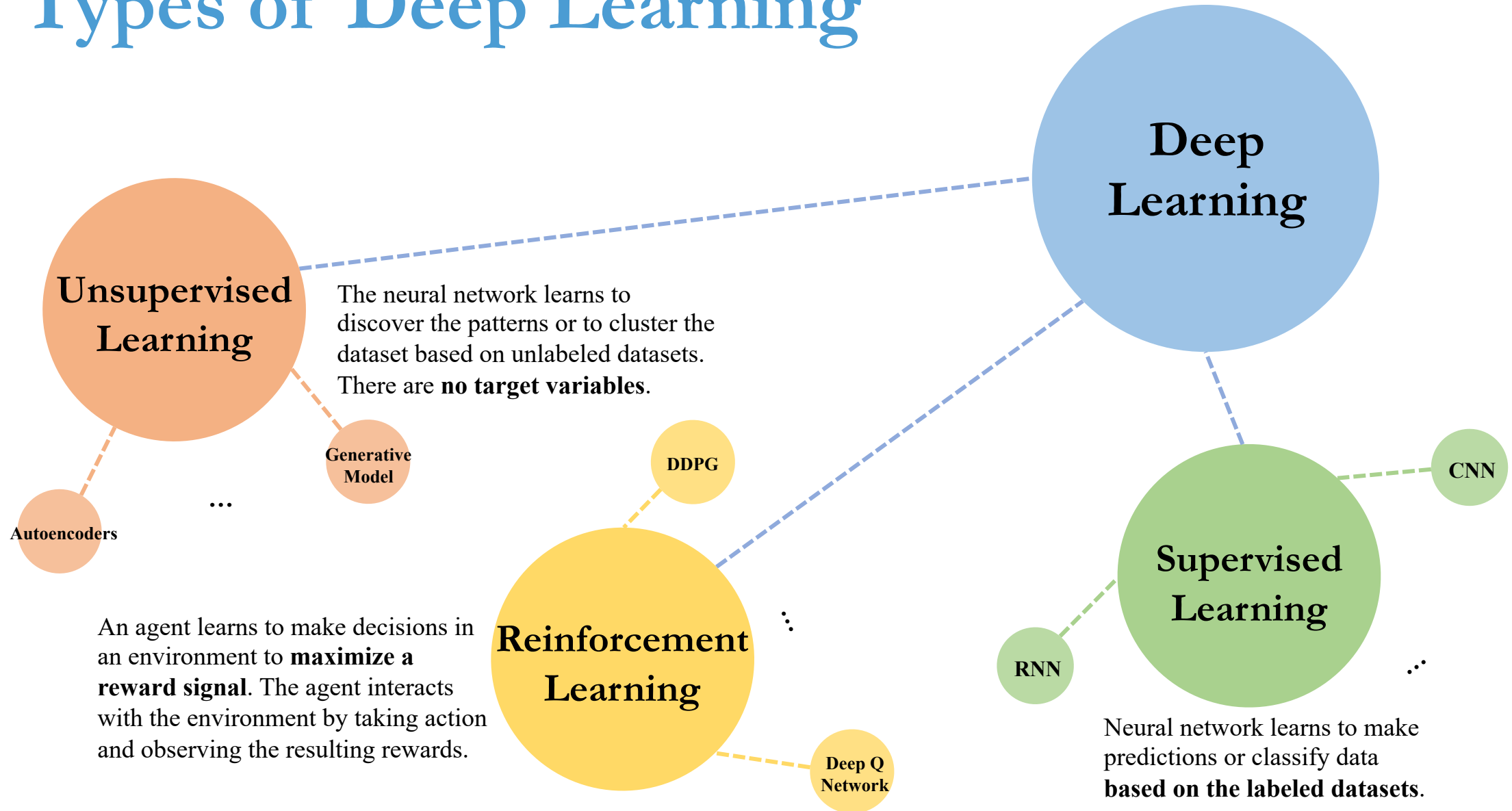
5 Optimization Techniques

6 CNNs

7 GNNs/GCNs

8 Theoretical Properties

Types of Deep Learning



Modern DL Model Architectures

1 Convolutional Neural Networks (CNNs)

- **Key Features:** Utilizes convolutional layers to process data in a grid pattern (like images).
- **Key Components:**
 - Convolutional Layers: Extract features from input images using filters.
 - Pooling Layers: Reduce dimensions and computational load, retaining key information.
 - Fully Connected Layers: Classify images based on extracted features.
- **Example Models:** LeNet-5, AlexNet, VGGNet.

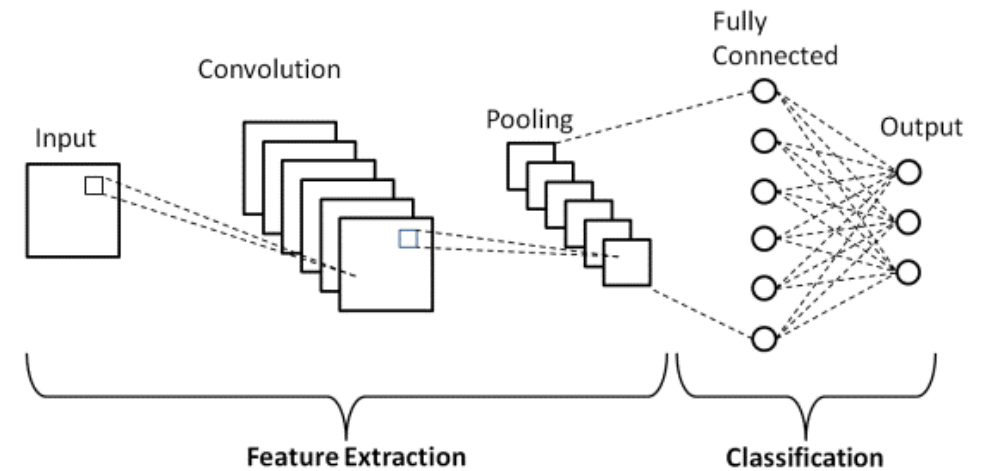


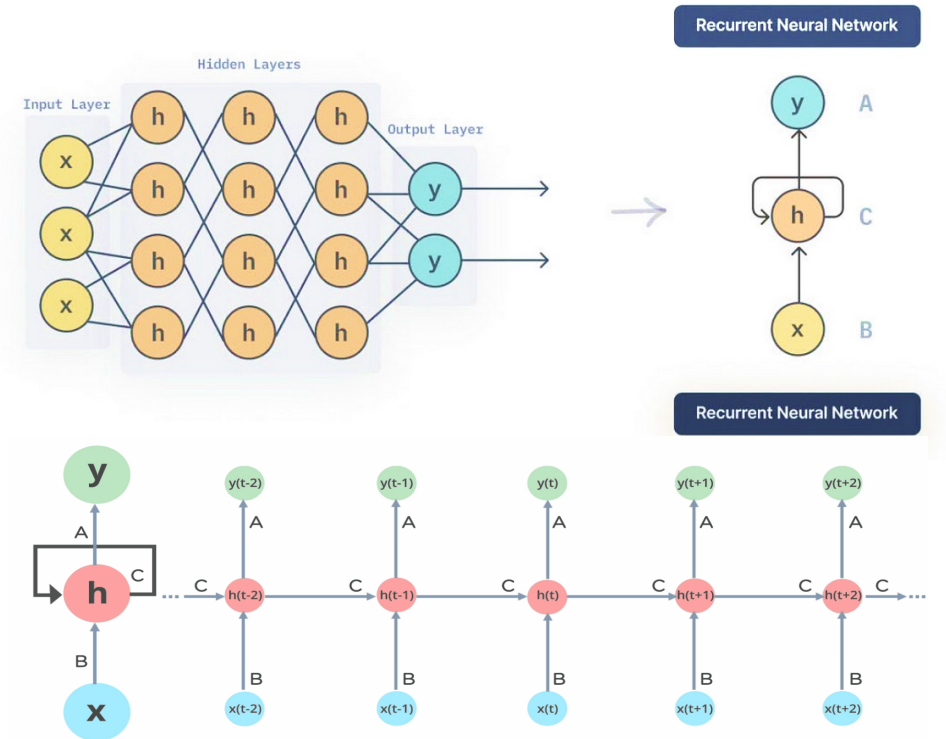
Figure. Basic CNN structure.

- **Applications in Biomedicine:**
 - Image classification in diagnostics (e.g., cancer detection from scans).
 - Image segmentation for identifying regions of interest in medical images.

Modern DL Model Architectures

2 Recurrent Neural Networks (RNNs)

- **Key Features:** Processes sequences of data (time-series data), with memory of previous inputs, capturing temporal dynamics.
- **Unique Feature:** Loop-like architecture allowing previous outputs to be used as inputs while having hidden states, enabling information persistence.
- **Challenges & Solutions:** Problem of vanishing gradients; solved by advanced RNNs, e.g. LSTM and GRU.
- **Example Models:** LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit).

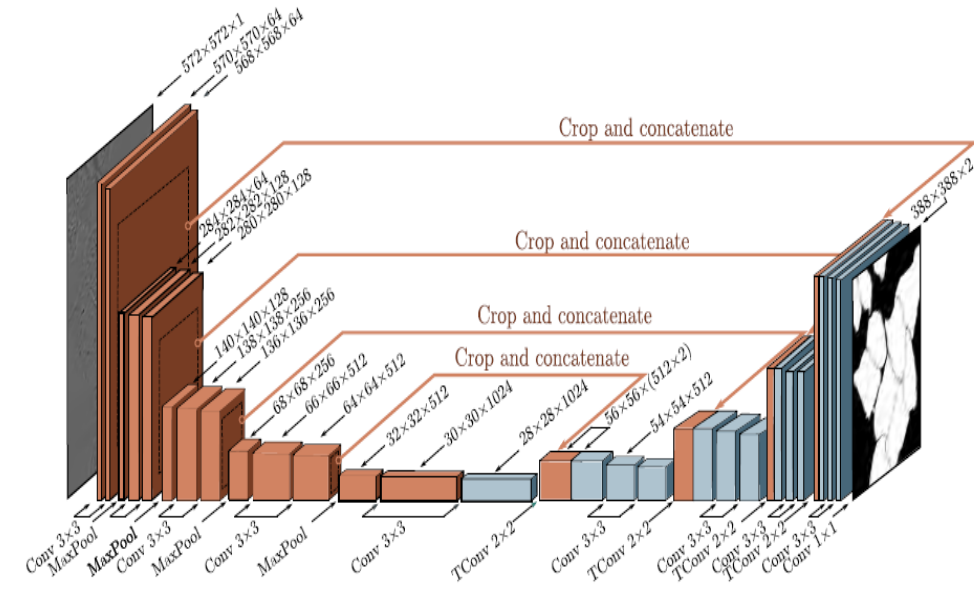


- **Applications in Biomedicine:**
 - Analysis of sequential patient data in EHRs.
 - Time-series analysis in physiological signal processing.

Modern DL Model Architectures

3 U-Net

- **Key Features:** U-shaped architecture with symmetric encoder and decoder paths. Skip connections that concatenate feature maps from encoder to decoder
- **Structure:** Encoder: Series of convolutional and max-pooling layers that capture context. Bottleneck: Intermediate layer connecting encoder and decoder. Decoder: Series of up-convolution and concatenation layers that restore resolution. Final Layer: Convolutional layer that maps features to the desired output.
- **Types:** 2D/3D U-Net, Attention U-Net.
- **Applications in Biomedicine:** Medical image segmentation. Satellite image segmentation. Biomedical image analysis. Autonomous driving. General image segmentation tasks.



U-Net for segmenting HeLa cells. The U-Net has an encoder-decoder structure, in which the representation is downsampled (orange blocks) and then re-upsampled (blue blocks). The encoder uses regular convolutions, and the decoder uses transposed convolutions. Residual connections append the last representation at each scale in the encoder to the first representation at the same scale in the decoder (orange arrows).

Modern DL Model Architectures

4 Autoencoders

- **Key Features:** Unsupervised learning models for dimensionality reduction and feature learning.
- **Structure:** Composed of an encoder (compressing input) and a decoder (reconstructing input).
- **Types:** Standard Autoencoders, Variational Autoencoders (VAEs).
- **Applications in Biomedicine:**
 - Data denoising (e.g., removing noise from images).
 - Anomaly detection in medical imaging (e.g., identifying unusual patterns).

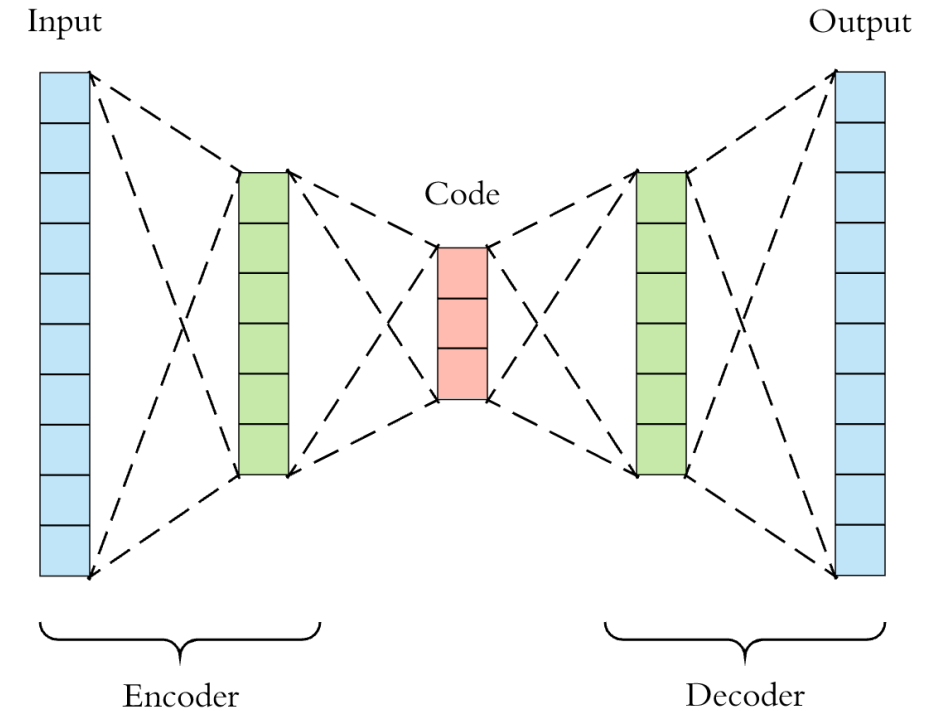


Figure 1. Visualization of an autoencoder

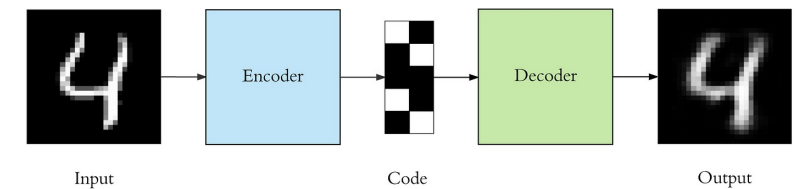
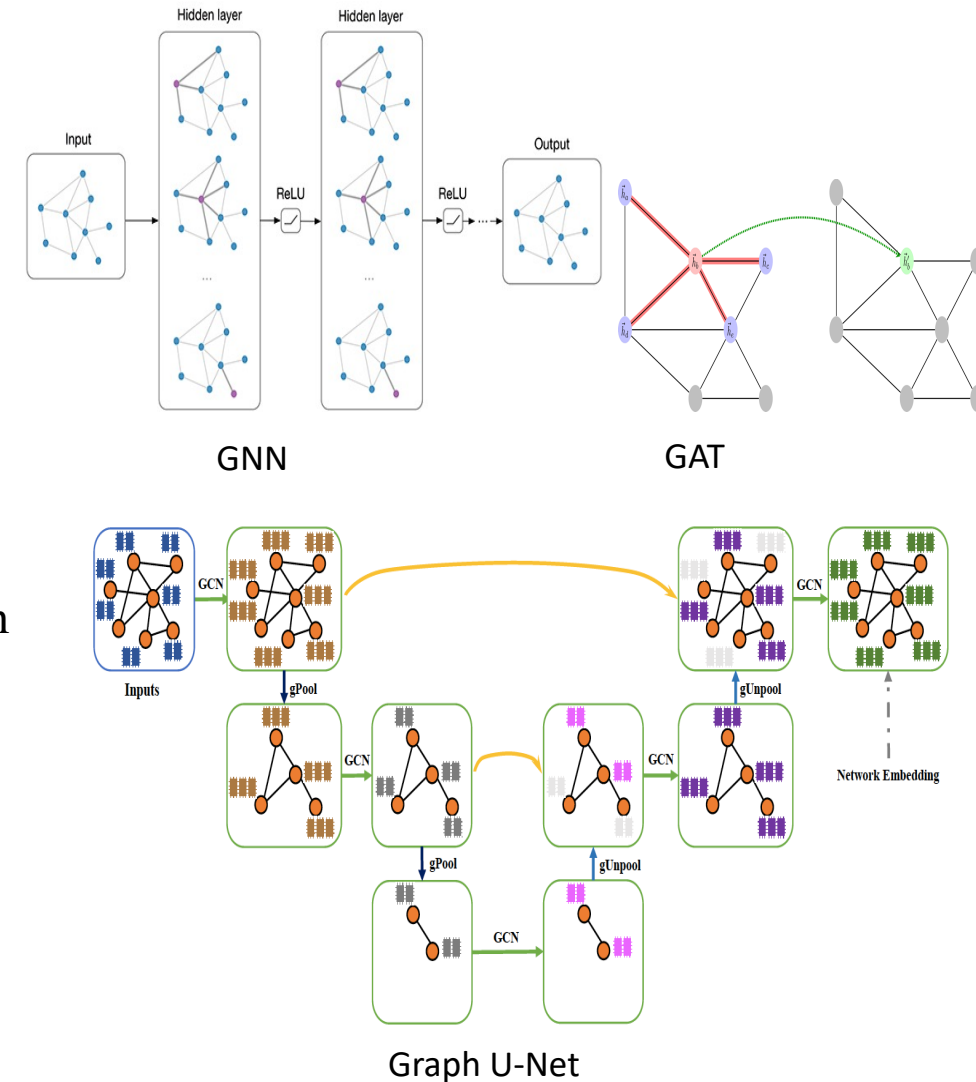


Figure 2. Autoencoders are a specific type of feedforward neural networks where the input is the same as the output.

Modern DL Model Architectures

5 Graph Neural Network

- **Key Features:** Ability to process graph-structured data. Utilizes node features and graph topology for learning. Effective in capturing dependencies between nodes. Supports inductive and transductive learning.
- **Structure:** Nodes, Edges, Node Features, Graph Convolution, and Readout Layer.
- **Types:** Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), Graph Recurrent Networks (GRNs), Graph Autoencoders, Graph U-Net
- **Applications in Biomedicine:**
Social Network Analysis, Knowledge Graphs, Drug Discovery, Recommender Systems, Network Security



Modern DL Model Architectures

6 Generative Adversarial Networks (GANs)

- **Key Features:** Comprises two neural networks, a generator and a discriminator, competing against each other.
- **Mechanism:**
 - Generator creates images, trying to fool the discriminator by generating data similar to those in the training set.
 - Discriminator evaluates them, trying to distinguish between fake data and real data
- **Example Models:** DCGAN, Pix2Pix, CycleGAN.

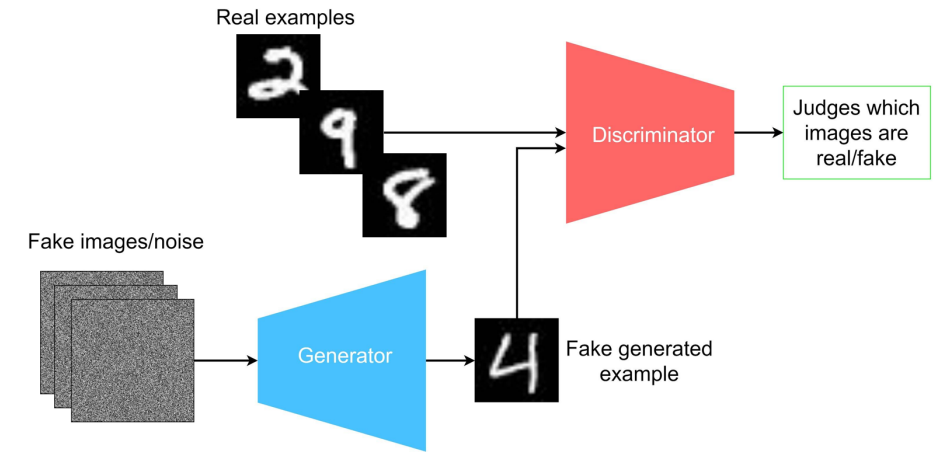


Figure. Visualization of the flow of GAN

- **Applications in Biomedicine:**
 - Generate high-resolution images from low-resolution inputs, enabling improved image quality.
 - Data augmentation in medical imaging for robust model training.

Modern DL Model Architectures

7 Transformer Models

- **Key Features:** Utilizes self-attention mechanisms, excellent for handling sequences of data.
- **Key Innovation:** Following an encoder-decoder structure, eliminating recurrence and convolutions.
- **Example Models:** BERT (adapted for biomedical applications), AlphaFold.
- **Applications in Biomedicine:**
 - Genomic sequence analysis for personalized medicine.
 - Protein structure prediction (e.g., AlphaFold's breakthroughs).

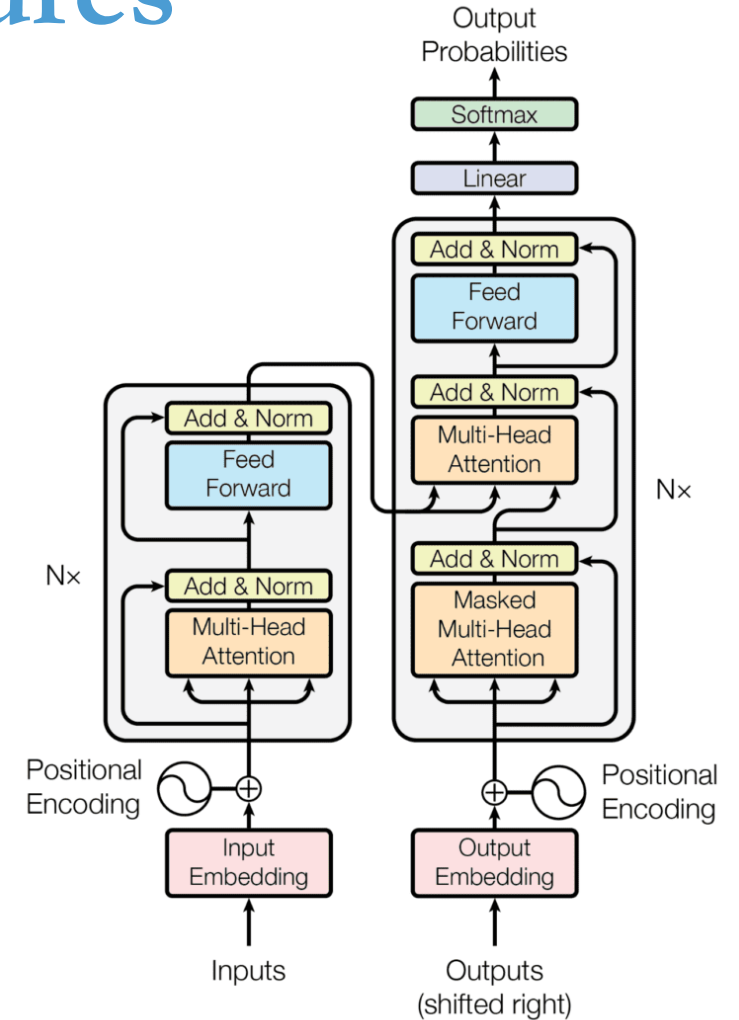
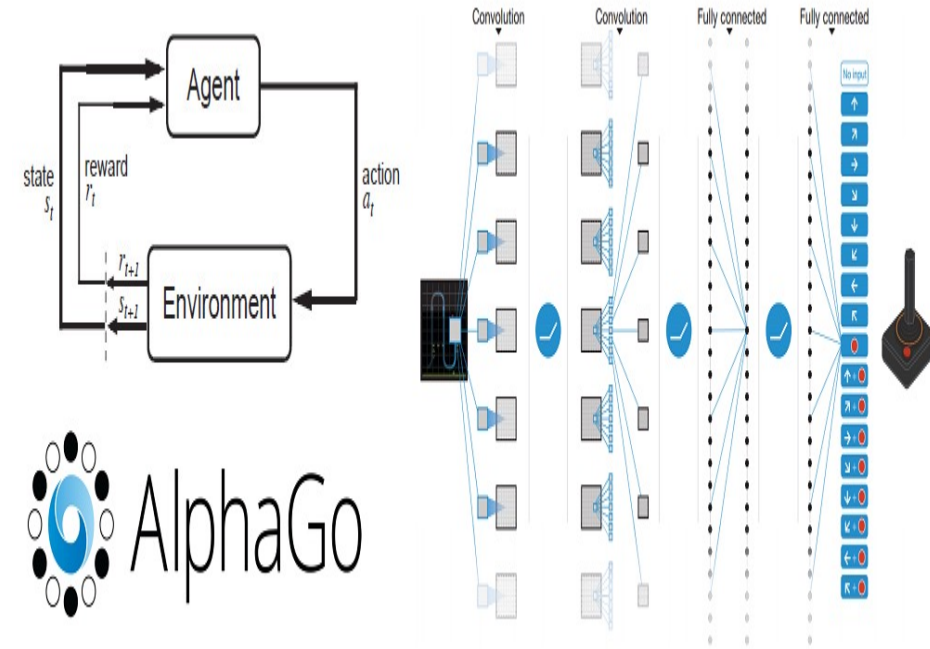


Figure. Transformer architecture

Modern DL Model Architectures

8 Deep Reinforcement Learning

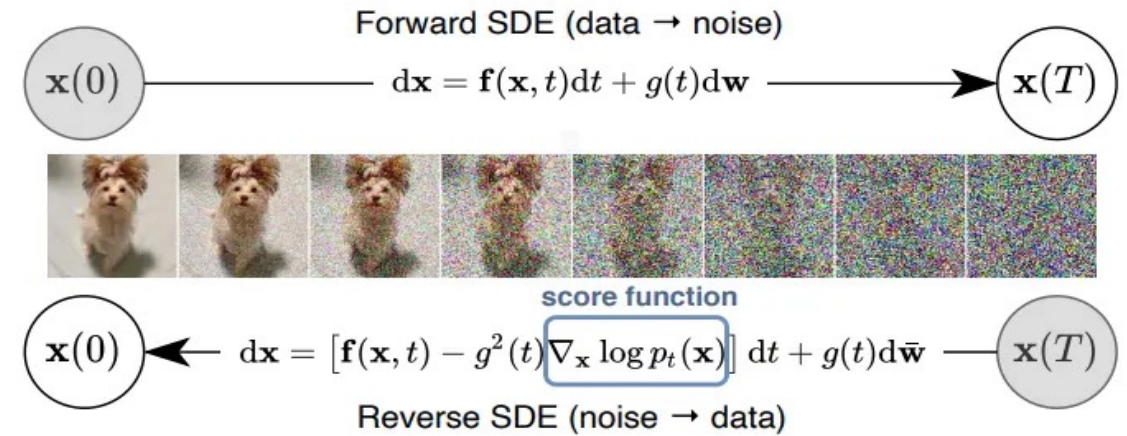
- **Key Features:** DRL leverages neural networks to approximate value functions and policies, enabling agents to learn complex tasks from high-dimensional sensory inputs.
- **Key Components:** Agent, Environment, Reward, Policy, and Value Function.
- **Example Models:** DQN (Deep Q-Network), A3C (Asynchronous Advantage Actor-Critic), PPO (Proximal Policy Optimization), SAC (Soft Actor-Critic)
- **Applications:**
 - Game Playing; Robotics
 - Autonomous Vehicles; Healthcare



Modern DL Model Architectures

9 Diffusion Models

- **Key Features:** Iterative forward noise \leftrightarrow learned reverse denoising, Simple MSE loss via denoising score matching, Exact or tractable likelihood bounds, Flexible conditioning &
- **Key Innovations:** Score-based learning, Optimized noise schedules, Fast samplers (DDIM) & continuous-time SDE/ODE formulations, Latent-space diffusion, Classifier-free guidance
- **Example Models:** DDPM (discrete diffusion), DDIM (implicit sampler), Score SDE / ODE (continuous), Stable Diffusion (latent), Imagen /DALL·E2 (text-to-image), WaveGrad / DiffWave (audio).
- **Applications:** Unconditional & conditional image generation, Text-to-image & multimodal AIGC, Inpainting, super-resolution, style transfer, Audio synthesis & denoising, Molecular structure generation, Video frame interpolation & generation



Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

Loss Function

Definition: a measure of error between what your model predicts and what the actual value is.

Purpose: quantifies how well the neural network matches what we want to output and thus guides the optimization process.

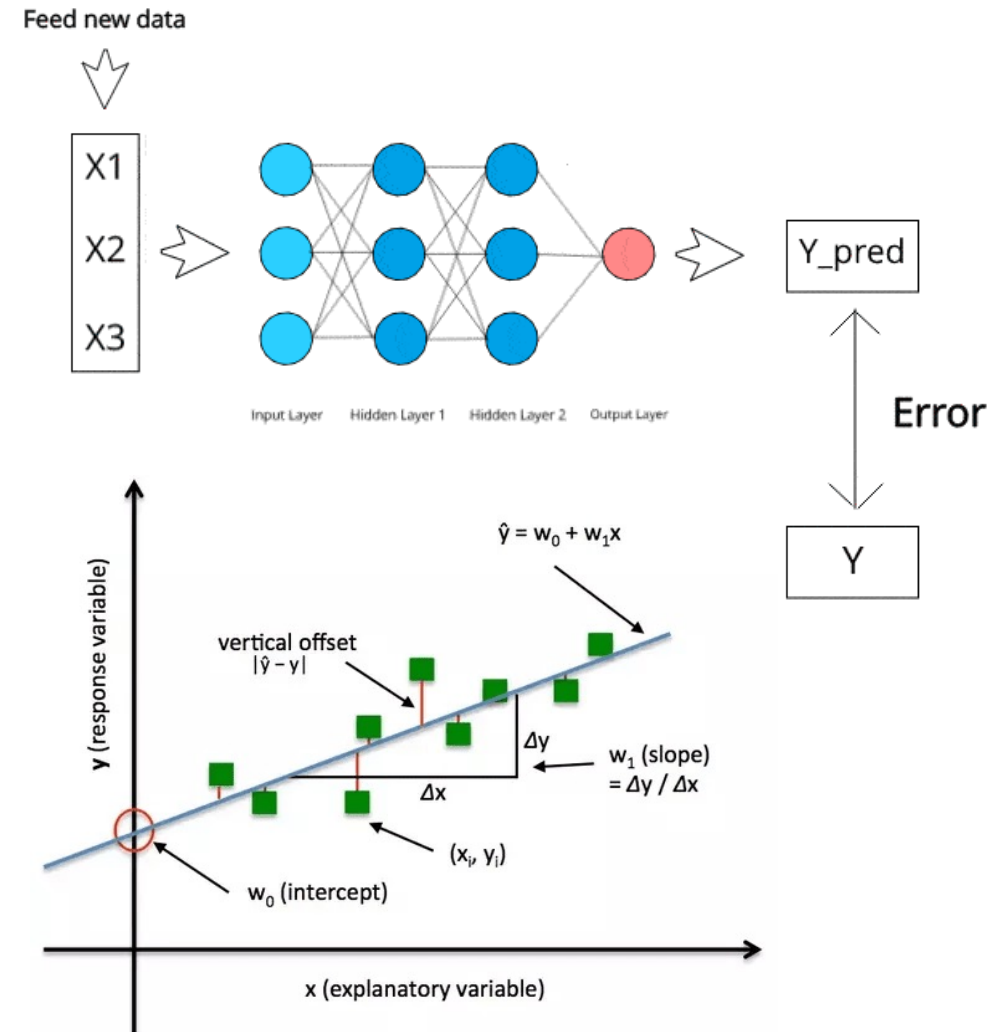
Importance: The choice of loss function directly impacts how the weights of the model are adjusted.

Examples: Mean Squared Error (Regression), Cross-Entropy (Classification).

Notation:

$$\mathcal{L}(f(X; W) \boxed{y})$$

Prediction True



Recipe for Constructing Loss Functions

Recipe for constructing loss functions

The recipe for constructing loss functions for training data $\{\mathbf{x}_i, \mathbf{y}_i\}$ using the maximum likelihood approach is hence:

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ defined over the domain of the predictions \mathbf{y} with distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \phi]$ to predict one or more of these parameters, so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \phi]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \phi])$.
3. To train the model, find the network parameters $\hat{\phi}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} [L[\phi]] = \underset{\phi}{\operatorname{argmin}} \left[- \sum_{i=1}^I \log [Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \phi])] \right]. \quad (5.6)$$

4. To perform inference for a new test example \mathbf{x} , return either the full distribution $Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \hat{\phi}])$ or the value where this distribution is maximized.

Data Type	Domain	Distribution	Use
univariate, continuous, unbounded	$y \in \mathbb{R}$	univariate normal	regression
univariate, continuous, unbounded	$y \in \mathbb{R}$	Laplace or t-distribution	robust regression
univariate, continuous, unbounded	$y \in \mathbb{R}$	mixture of Gaussians	multimodal regression
univariate, continuous, bounded below	$y \in \mathbb{R}^+$	exponential or gamma	predicting magnitude
univariate, continuous, bounded	$y \in [0, 1]$	beta	predicting proportions
multivariate, continuous, unbounded	$\mathbf{y} \in \mathbb{R}^K$	multivariate normal	multivariate regression
univariate, continuous, circular	$y \in (-\pi, \pi]$	von Mises	predicting direction
univariate, discrete, binary	$y \in \{0, 1\}$	Bernoulli	binary classification
univariate, discrete, bounded	$y \in \{1, 2, \dots, K\}$	categorical	multiclass classification
univariate, discrete, bounded below	$y \in [0, 1, 2, 3, \dots]$	Poisson	predicting event counts
multivariate, discrete, permutation	$\mathbf{y} \in \text{Perm}[1, 2, \dots, K]$	Plackett-Luce	ranking

Loss Function for Classification

Binary Classification Task

Binary Cross-Entropy

Multi-Class Classification Task

Cross-Entropy Loss

Kullback Leibler Divergence Loss

Negative Log Likelihood Loss

Dice Loss Function

Dice Loss is derived from the **Dice Coefficient**, which is a statistical tool to measure the similarity or overlaps between two sets.

Unlike cross entropy loss, dice loss is particularly effective when dealing with imbalanced datasets and when the focus is on capturing fine details in the segmentation masks. It's a very popular loss function in medical image segmentation.

Dice coefficient:

$$Dice = \frac{2 \times |A \cap B|}{|A| + |B|}$$

To avoid division by zero, a small constant (smooth) is added to the numerator and denominator.

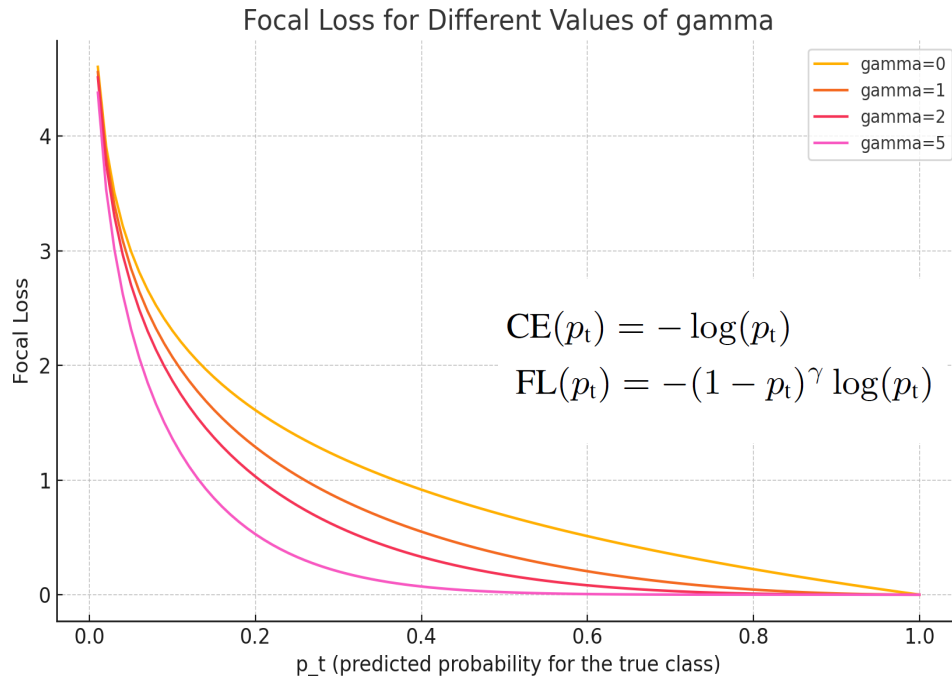
$$Dice_{smooth} = \frac{2 \times |A \cap B| + smooth}{|A| + |B| + smooth} \quad \text{and} \quad Dice\ Loss = 1 - Dice_{smooth}$$

Imbalanced Data-Loss Functions

❖ Consider Data Characteristics:

❖ Imbalanced Data: Use Weighted Cross-Entropy or Focal Loss.

❖ Outliers: Use Huber Loss or Mean Absolute Error.



1. **Weighted Likelihood:** Modify the likelihood function to emphasize minority class samples:

$$\mathcal{L} = \sum_{i=1}^N w_{y_i} \log P(y_i|x_i; \theta)$$

where w_{y_i} is inversely proportional to the class frequency.

2. **Cost-Sensitive Likelihood:** Introduce class-specific penalties:

$$\mathcal{L}_{\text{weighted}} = - \sum_{i=1}^N \frac{1}{f_{y_i}} \log P(y_i|x_i; \theta)$$

where f_{y_i} is the frequency of class y_i .

3. **Focal Loss:** Focuses on hard-to-classify examples:

$$\text{Loss}_{\text{focal}} = -\alpha(1 - p_t)^\gamma \log(p_t)$$

where α controls class weighting, and γ modulates the focus on hard examples.

Class-Balanced Loss: Reweights classes based on their effective number of samples:

$$w_c = \frac{1 - \beta}{1 - \beta^{n_c}}$$

where n_c is the number of samples in class c , and $\beta \in [0, 1)$.

Outliers-Loss Functions

Huber Loss: Combines ℓ_1 and ℓ_2 loss to handle small and large residuals differently:

$$\text{Loss}_{\text{Huber}} = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{\delta^2}{2} & \text{otherwise.} \end{cases}$$

Tukey's Biweight Loss: Suppresses large residuals:

$$\text{Loss}_{\text{Tukey}} = \begin{cases} \frac{\delta^2}{6} \left(1 - \left(1 - \frac{r^2}{\delta^2}\right)^3\right) & |r| \leq \delta, \\ \frac{\delta^2}{6} & |r| > \delta, \end{cases}$$

where $r = y - f(x)$.

Quantile Loss: Focuses on specific quantiles:

$$\text{Loss}_{\text{quantile}} = \max(\tau \cdot e, (1 - \tau) \cdot e)$$

where τ is the target quantile, and $e = y - f(x)$.

Barron, J. T. (2019). A general and adaptive robust loss function. In *CVPR*.

$$\rho(x, \alpha, c) = \begin{cases} \frac{1}{2} (x/c)^2 & \text{if } \alpha = 2 \\ \log \left(\frac{1}{2} (x/c)^2 + 1 \right) & \text{if } \alpha = 0 \\ 1 - \exp \left(-\frac{1}{2} (x/c)^2 \right) & \text{if } \alpha = -\infty \\ \frac{|\alpha-2|}{\alpha} \left(\left(\frac{(x/c)^2}{|\alpha-2|} + 1 \right)^{\alpha/2} - 1 \right) & \text{otherwise} \end{cases}$$

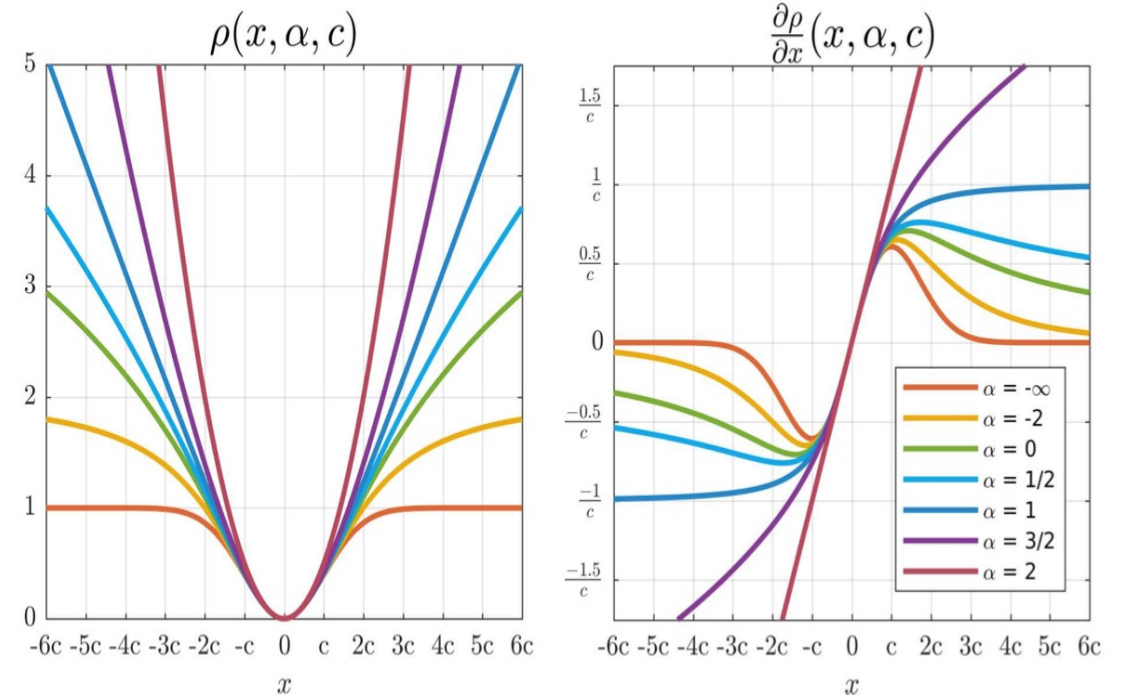


Figure 1. Our general loss function (left) and its gradient (right) for different values of its shape parameter α . Several values of α reproduce existing loss functions: L2 loss ($\alpha = 2$), Charbonnier loss ($\alpha = 1$), Cauchy loss ($\alpha = 0$), Geman-McClure loss ($\alpha = -2$), and Welsch loss ($\alpha = -\infty$).

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

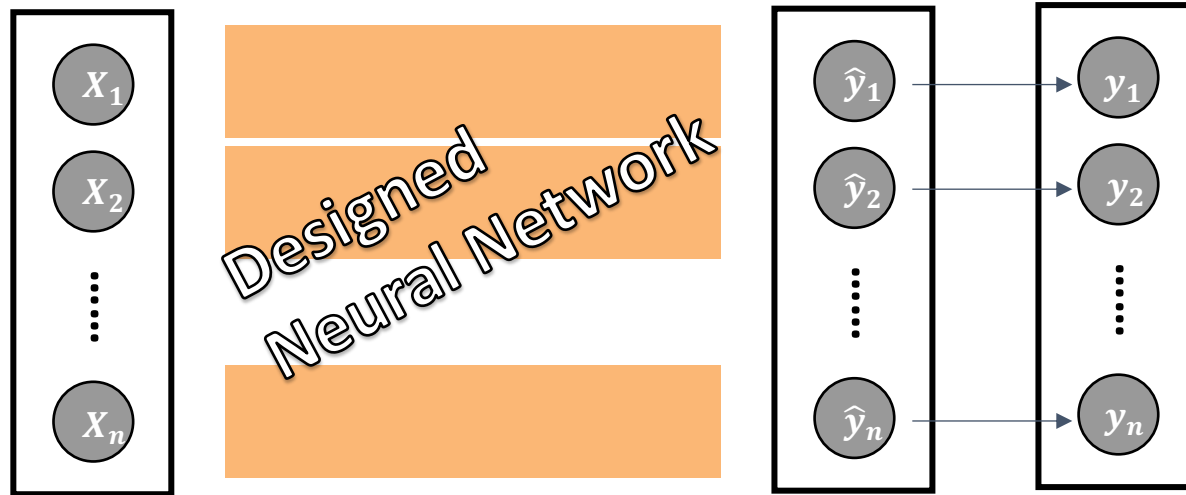
5 Optimization Techniques

6 CNNs

7 GNNs/GCNs

8 Theoretical Properties

Fitting DL Models



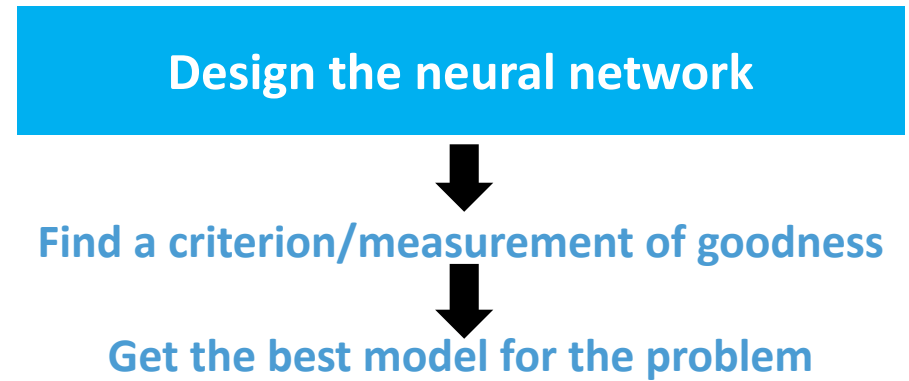
$$\begin{aligned}
 h_1 &= a[\beta_0 + \Omega_0 \mathbf{x}] \\
 h_2 &= a[\beta_1 + \Omega_1 h_1] \\
 h_3 &= a[\beta_2 + \Omega_2 h_2] \\
 &\vdots \\
 h_K &= a[\beta_{K-1} + \Omega_{K-1} h_{K-1}] \\
 y &= \beta_K + \Omega_K h_K.
 \end{aligned}$$

A **loss function** is needed here,
to measure the difference between the output and truth

Total loss:
$$L = \sum \ell(\hat{y}_i, y_i)$$

$$\hat{y}_i = \beta_K + \Omega_K \mathbf{h}_K(\mathbf{x}_i; [(\beta_0, \Omega_0), \dots, (\beta_{K-1}, \Omega_{K-1})])$$

Find the network parameters to minimize the loss



Loss Optimization

Goal: find the network weight that achieve the lowest loss.

Find the value of the parameters that help the loss function reach the lowest value.

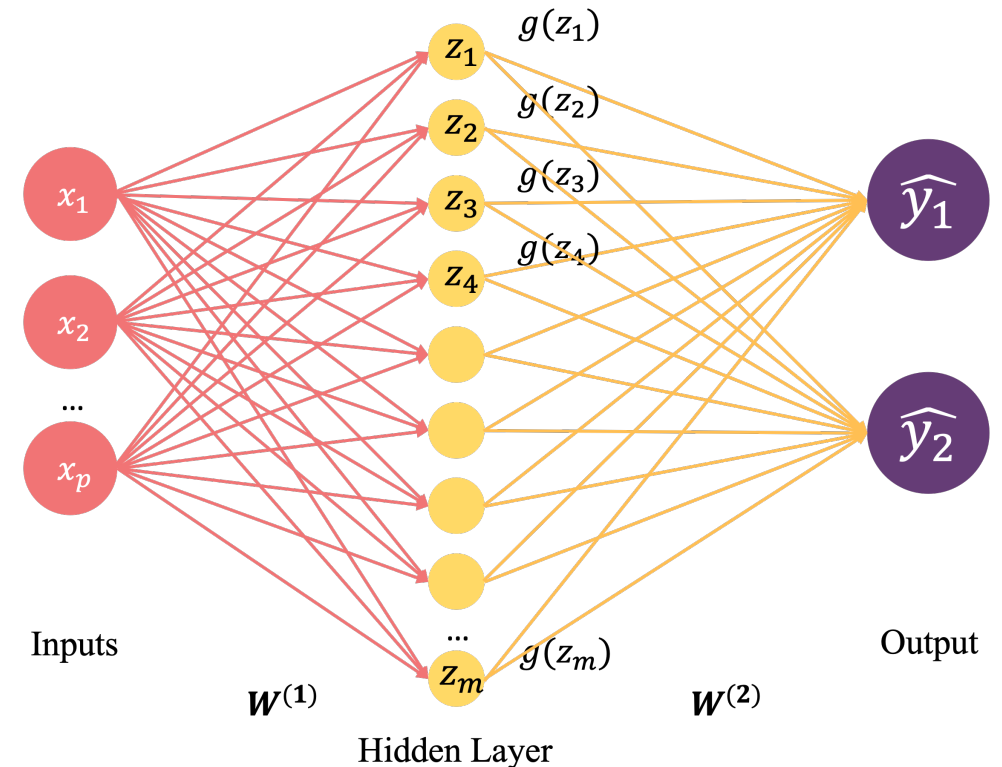
Write this goal in mathematical format:

$$\hat{W} = \underset{W}{\operatorname{argmin}} \mathcal{L}(\underbrace{f(X; W)}_{\text{Prediction}}, \underbrace{y}_{\text{True}})$$

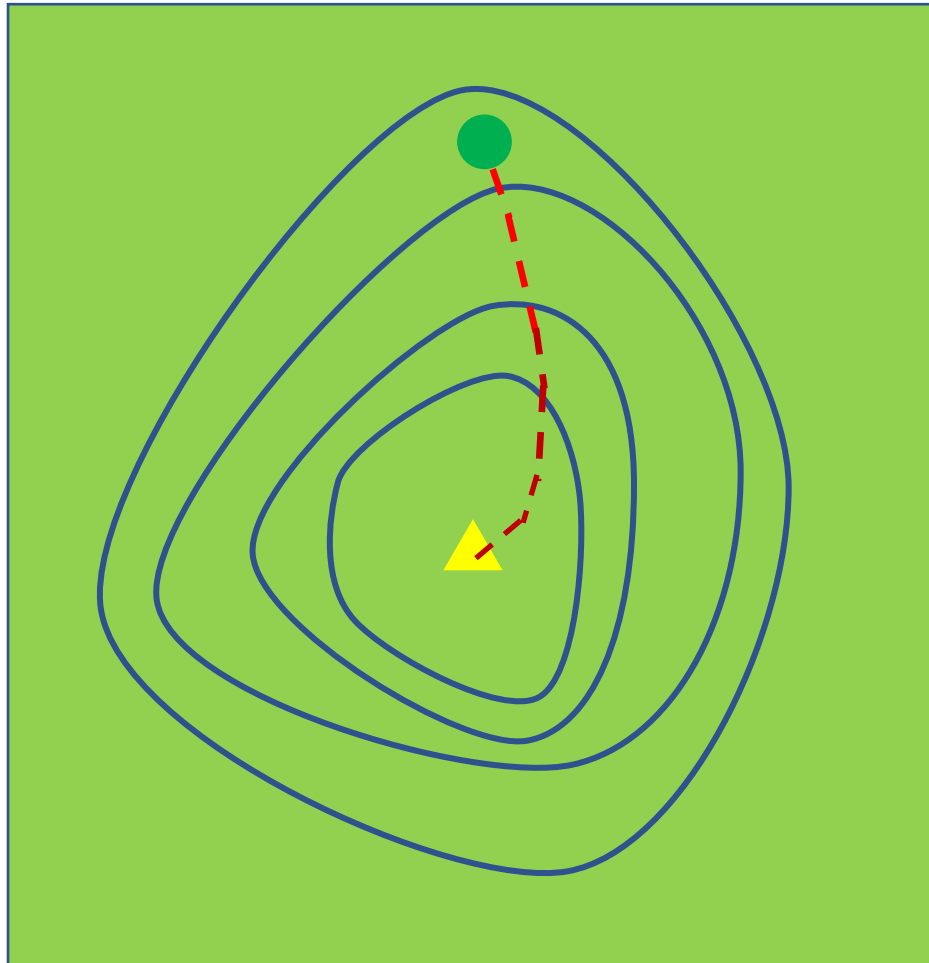
The loss function is a function of the network weights W .

$$W = [W^{(1)}, W^{(2)}, \dots]$$

contains all the weight vectors needed to be adjusted in the neural network



Gradient Descent



$$\hat{W} = \operatorname{argmin} \mathcal{L}(f(\mathbf{X}; \mathbf{W}), \mathbf{y})$$

A first-order iterative optimization algorithm for finding the minimum of a function.

Step 1. Compute the derivatives of the loss w.r.t. the parameters

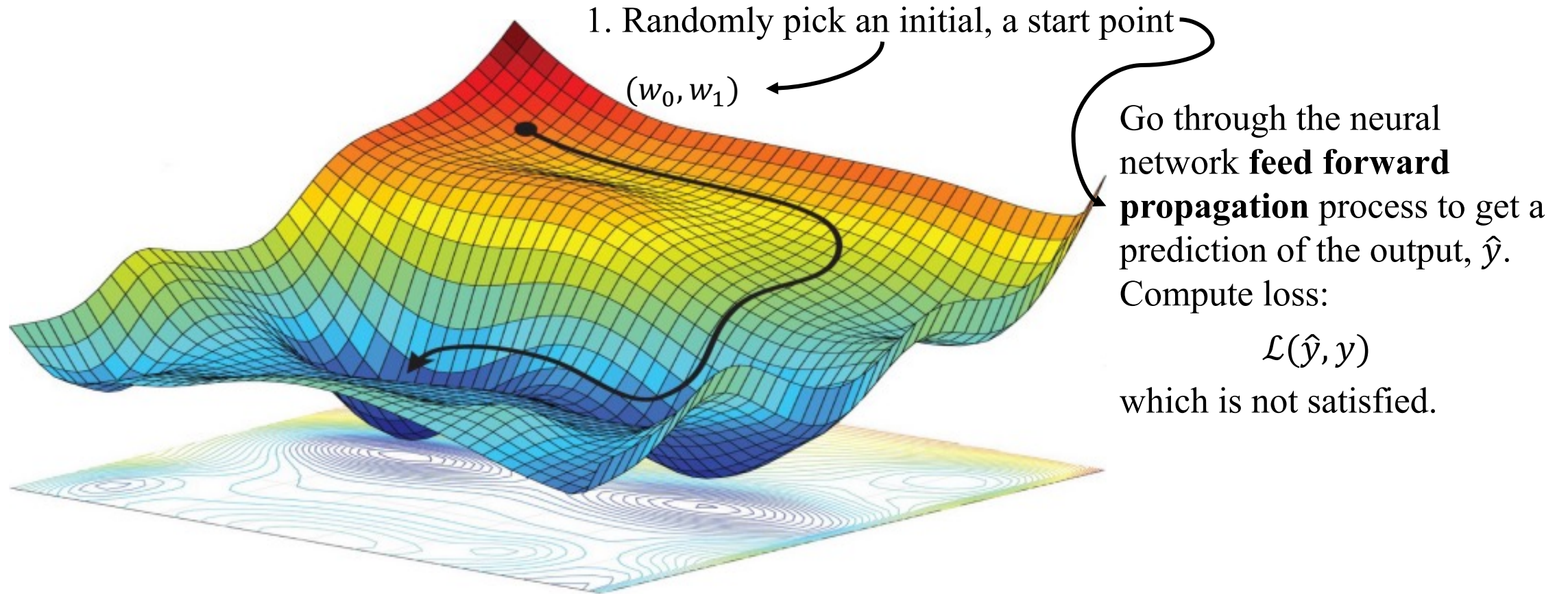
$$\frac{\partial \mathcal{L}(f(\mathbf{X}; \mathbf{W}), \mathbf{y})}{\partial \mathbf{W}}$$

Step 2. Update the parameters according to the rule:

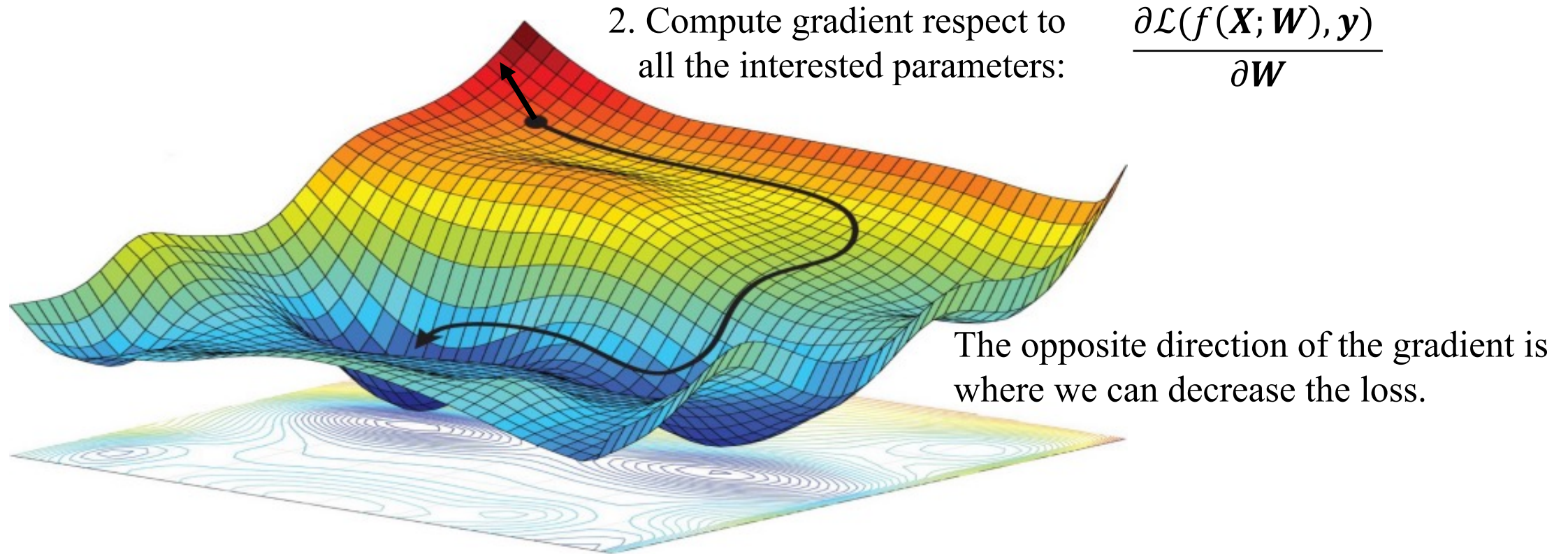
$$\mathbf{w}_{new} = \mathbf{w} - \alpha \frac{\partial \mathcal{L}(f(\mathbf{X}; \mathbf{W}), \mathbf{y})}{\partial \mathbf{W}}$$

where the positive scalar α (learning rate) determines the magnitude of the change.

Multi-Dimension Optimization Process

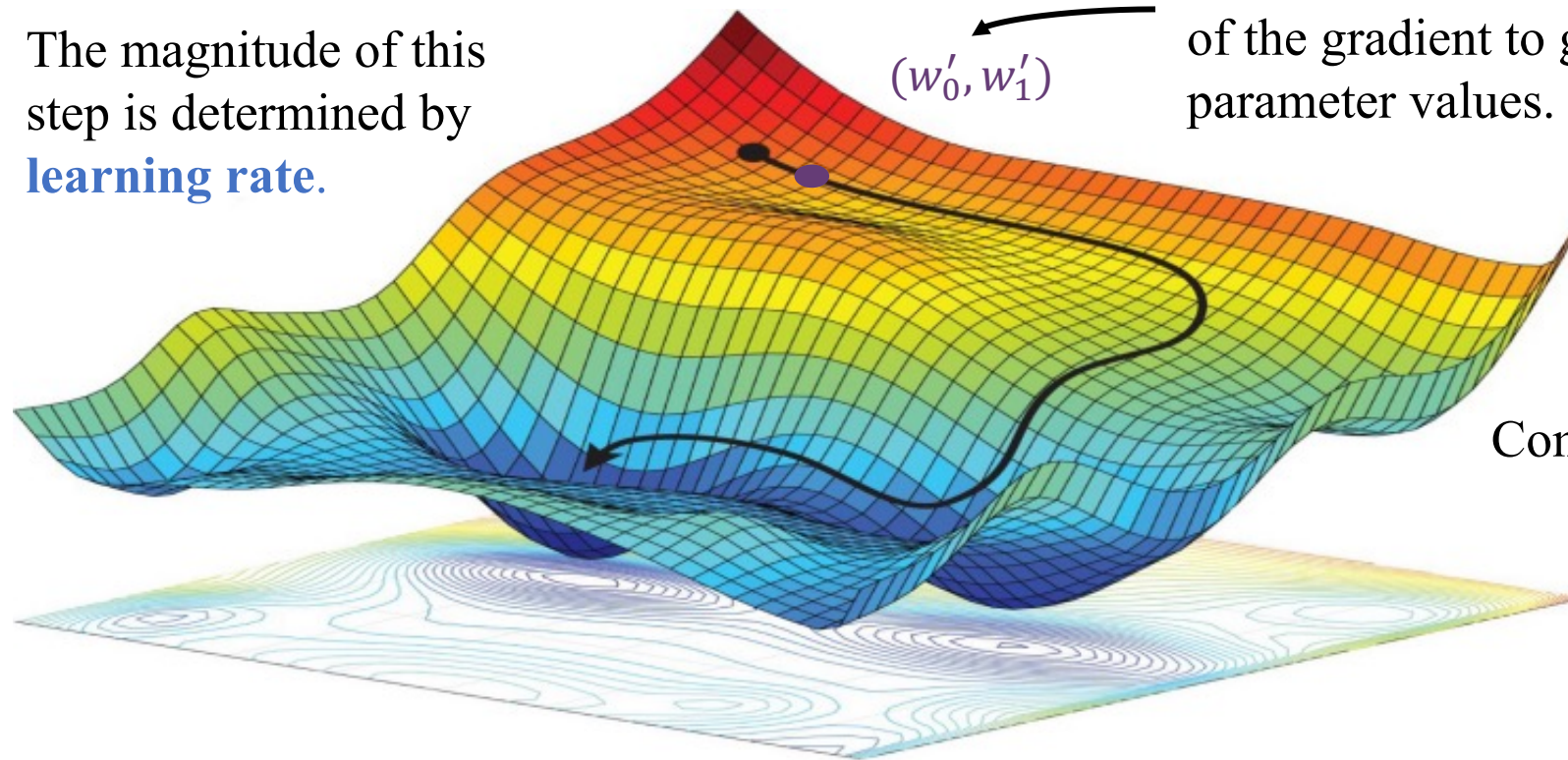


Multi-Dimension Optimization Process



Multi-Dimension Optimization Process

The magnitude of this step is determined by **learning rate**.



3. Take a small step in the opposite direction of the gradient to get a new proposal of the parameter values.

Compute loss with the new values:

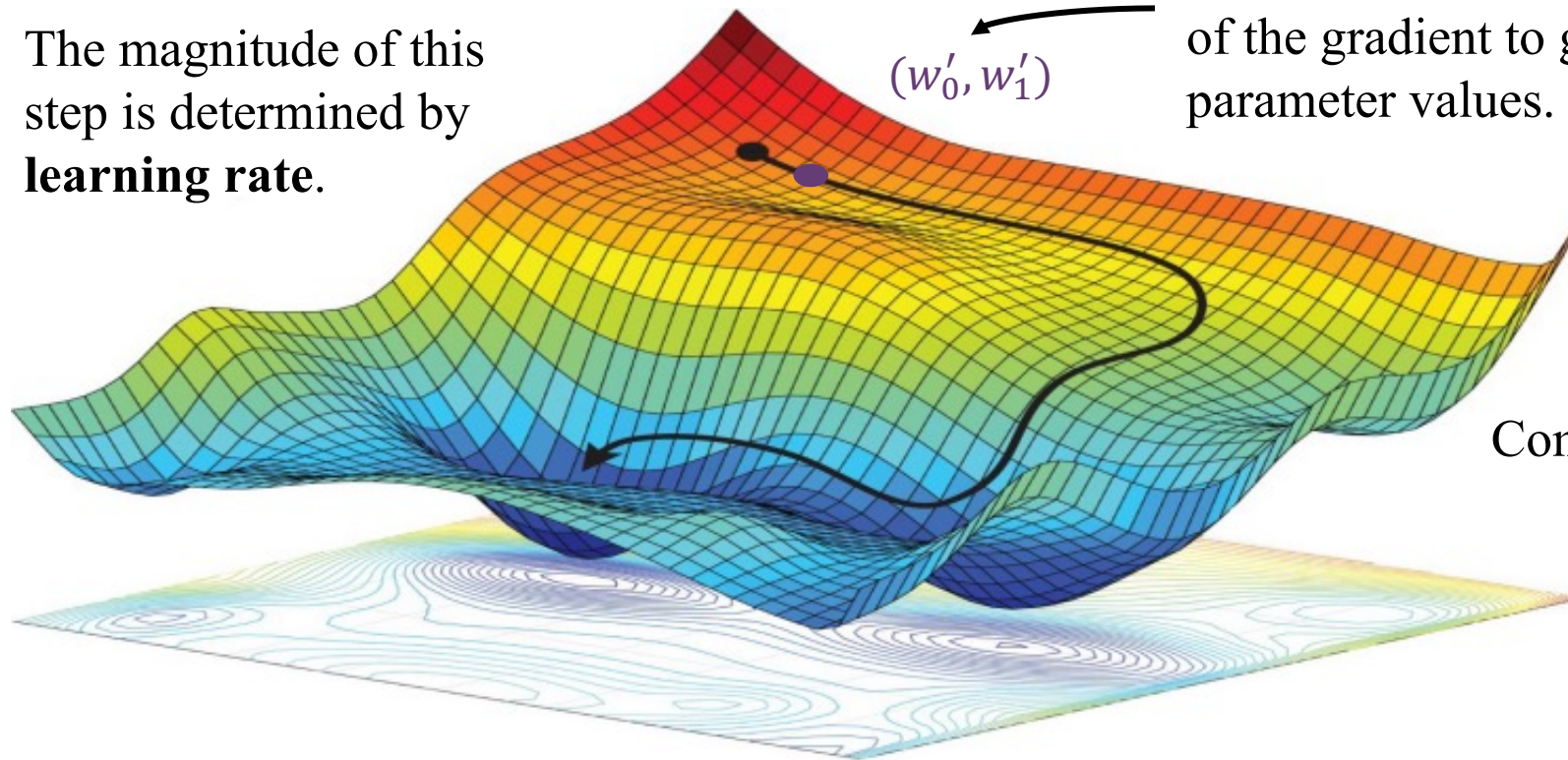
$$\mathcal{L}(f(\mathbf{X}; \mathbf{W}'), y)$$

Check if it converges.

4. Repeat steps 2 and 3 until the loss converges.

Gradient Descent

The magnitude of this step is determined by **learning rate**.



3. Take a small step in the opposite direction of the gradient to get a new proposal of the parameter values.

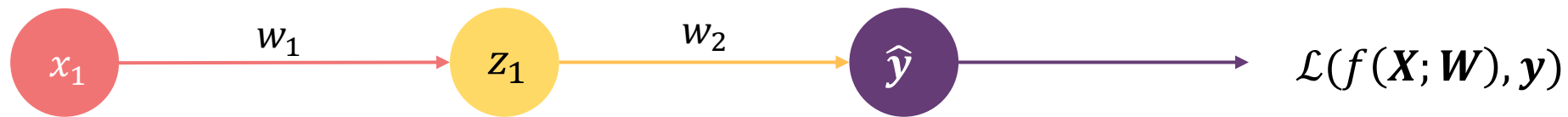
Compute loss with the new values:

$$\mathcal{L}(f(\mathbf{X}; \mathbf{W}'), y)$$

Check if it converges.

4. Repeat steps 2 and 3 until the loss converges.

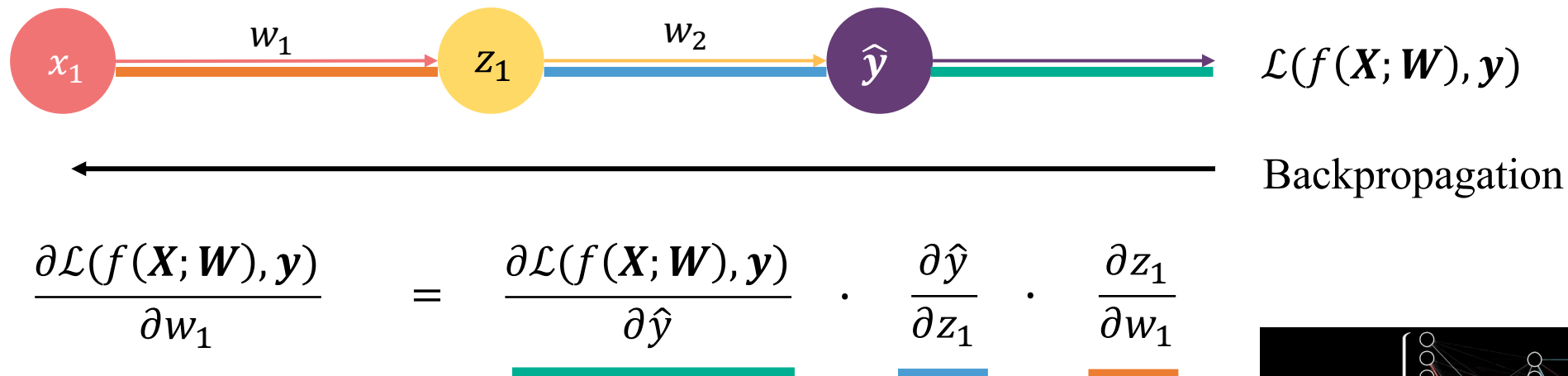
Gradient Computation: Backpropagation



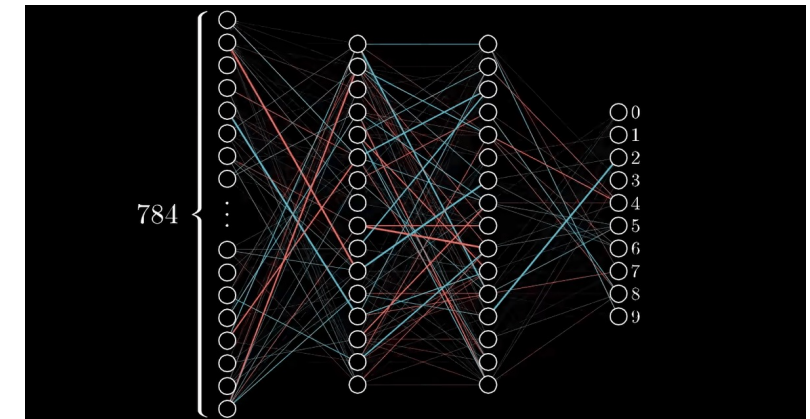
$$\frac{\partial \mathcal{L}(f(\mathbf{X}; \mathbf{W}), \mathbf{y})}{\partial w_1} = \frac{\partial \mathcal{L}(f(\mathbf{X}; \mathbf{W}), \mathbf{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} \quad \text{Chain rule}$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \quad \text{Chain rule again}$$

Gradient Computation: Backpropagation



Repeat this process for each layer, see the visual on the right:



Optimization Algorithms in PyTorch

Stochastic Gradient Descent (SGD)

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Gradient Descent with Momentum

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

AdaGrad (Adaptive Gradient Algorithm)

```
optimizer = torch.optim.Adagrad(model.parameters(), lr=0.01)
```

Adam (Adaptive Moment Estimation)

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Stochastic Gradient Descent (SGD)

Characteristics:

- Basic form of gradient descent used in neural networks.
- Fixed learning rate.
- In each iteration, randomly select a single data point (or a batch of data points) from the training set to calculate the gradient of the loss function.
- Updates parameters for each training example, leading to frequent updates with high variance.

Batch Size:

Epoch:

Advantages:

- Simple and easy to understand.
- Can escape local minima due to its inherent noise.

Disadvantages:

- Slow convergence on large datasets and high variance in updates.
- Sensitive to learning rate and other hyperparameters.

Gradient Descent with Momentum

Characteristics:

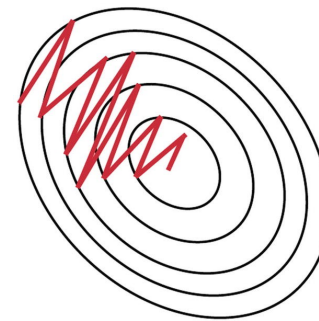
- Builds upon SGD by considering past gradients to smooth out the updates.
- Uses a momentum factor to accelerate SGD in the relevant direction.

Parameter update rule:

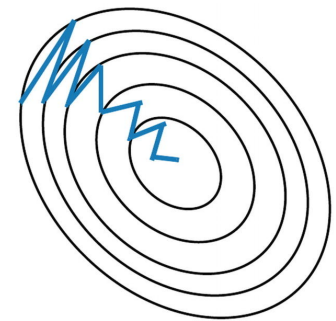
1. Update Velocity: $v = \gamma v - \alpha \nabla f(x)$.
2. Update Parameter: $x = x + v$

Advantages:

- Faster convergence than standard SGD.
- Reduces oscillations and improves stability.



Stochastic Gradient
Descent **without**
Momentum



Stochastic Gradient
Descent **with**
Momentum

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi}$$
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1},$$

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t - \alpha \beta \cdot \mathbf{m}_t]}{\partial \phi}$$
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1},$$

AdaGrad (Adaptive Gradient Algorithm)

Parameter update rule:

1. Update accumulation: $G = G + g^2$, where g is the gradient of the loss function with respect to each parameter.
2. Adjust Learning Rate: Scale the learning rate for each parameter inversely proportional to the square root of G .
3. Update Parameters: Update the parameters using the adjusted learning rate, $x = x - \frac{\alpha}{\sqrt{G + \epsilon}} \cdot g$ where α is the initial learning rate, ϵ is a small constant added to improve numerical stability.

$$\begin{aligned}\mathbf{m}_{t+1} &\leftarrow \frac{\partial L[\phi_t]}{\partial \phi} \\ \mathbf{v}_{t+1} &\leftarrow \left(\frac{\partial L[\phi_t]}{\partial \phi} \right)^2.\end{aligned}$$

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1} + \epsilon}},$$

AdaGrad (Adaptive Gradient Algorithm)

Characteristics:

- Adjusts the learning rate to each parameter, decreasing it for parameters with large gradients.
- Each parameter has its own learning rate, which can be beneficial for datasets with features of varying importance or scale.

Advantages:

- The effective learning rate decreases over time for each parameter. Eliminates the need to manually tune the learning rate.
- Well-suited for dealing with sparse features or data with different scales.

Disadvantages:

- The continuously accumulating squared gradient can lead to an excessively reduced learning rate, causing the algorithm to stop learning too early.

Adam (Adaptive Moment Estimation)

Parameter update algorithm:

1. Moving averages: two vectors m and v are used to store moving averages of the gradients and squared gradients, both initialized to zero.

2. Hyperparameters: β_1 and β_2 , close to 1 (common defaults are 0.9 and 0.999).

3. Update Moving Averages: $m = \beta_1 m + (1 - \beta_1)g$ and $v = \beta_2 v + (1 - \beta_2)g^2$.

4. Correct Bias: $\hat{m} = \frac{m}{1 - \beta_1^t}$ and $\hat{v} = \frac{v}{1 - \beta_2^t}$.
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi}$$

5. Adjust parameters: $x = x - \frac{\alpha}{\sqrt{\hat{v} + \epsilon}} \hat{m}$
$$\mathbf{v}_{t+1} \leftarrow \gamma \cdot \mathbf{v}_t + (1 - \gamma) \left(\sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi} \right)^2$$

where α is the initial learning rate, ϵ is a small constant added to improve numerical stability.

$$\begin{aligned} \tilde{\mathbf{m}}_{t+1} &\leftarrow \frac{\mathbf{m}_{t+1}}{1 - \beta^{t+1}} & \phi_{t+1} &\leftarrow \phi_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1} + \epsilon}} & \phi_{t+1} &\leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1} + \epsilon}}, \\ \tilde{\mathbf{v}}_{t+1} &\leftarrow \frac{\mathbf{v}_{t+1}}{1 - \gamma^{t+1}}. \end{aligned}$$

Adam (Adaptive Moment Estimation)

Characteristics:

- Designed to combine the advantages of two other popular optimizers: the adaptive learning rate feature of AdaGrad and the momentum feature of RMSprop.
- Different learning rates for different parameters and adjusts them throughout training.
- Corrects the bias in moving averages, especially important in the initial training phase.

Advantages:

- Combines the benefits of AdaGrad and RMSprop.
- Performs well in practice and across a wide range of non-convex optimization problems and large dataset.

Disadvantages:

- Can be memory-intensive due to storing moving averages for each parameter.
- Might not converge to the optimal solution in certain theoretical cases.

Optimization

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi}$$

➤ **Batch SGD**

➤ **Momentum**

$$\begin{aligned} \mathbf{m}_{t+1} &\leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi} \\ \phi_{t+1} &\leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}, \end{aligned}$$

$$\begin{aligned} \mathbf{m}_{t+1} &\leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi} & \tilde{\mathbf{m}}_{t+1} &\leftarrow \frac{\mathbf{m}_{t+1}}{1 - \beta^{t+1}} \\ \mathbf{v}_{t+1} &\leftarrow \gamma \cdot \mathbf{v}_t + (1 - \gamma) \left(\sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi} \right)^2 & \tilde{\mathbf{v}}_{t+1} &\leftarrow \frac{\mathbf{v}_{t+1}}{1 - \gamma^{t+1}} \end{aligned}$$

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}} + \epsilon}$$

➤ **Adaptive Moment Estimation (Adam)**

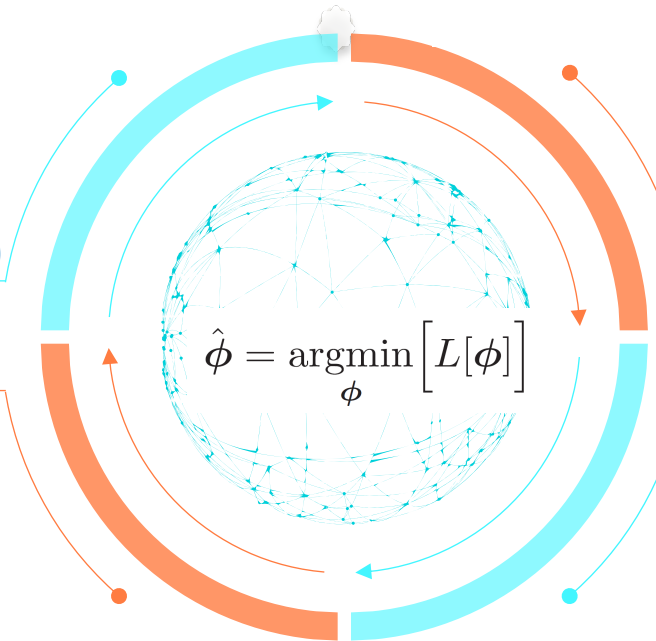
➤ **Backpropagation algorithm**

$$\begin{aligned} \mathbf{f}_0 &= \beta_0 + \Omega_0 \mathbf{x}_i \\ \mathbf{h}_k &= \mathbf{a}[\mathbf{f}_{k-1}] \\ \mathbf{f}_k &= \beta_k + \Omega_k \mathbf{h}_k \end{aligned}$$

Forward Pass

$$\begin{aligned} \frac{\partial \ell_i}{\partial \beta_k} &= \frac{\partial \ell_i}{\partial \mathbf{f}_k} & \frac{\partial \ell_i}{\partial \beta_0} &= \frac{\partial \ell_i}{\partial \mathbf{f}_0} \\ \frac{\partial \ell_i}{\partial \Omega_k} &= \frac{\partial \ell_i}{\partial \mathbf{f}_k} \mathbf{h}_k^T & \frac{\partial \ell_i}{\partial \Omega_0} &= \frac{\partial \ell_i}{\partial \mathbf{f}_0} \mathbf{x}_i^T \\ \frac{\partial \ell_i}{\partial \mathbf{f}_{k-1}} &= \mathbb{I}[\mathbf{f}_{k-1} > 0] \odot \left(\Omega_k^T \frac{\partial \ell_i}{\partial \mathbf{f}_k} \right) \end{aligned}$$

Backward Passes



Efficient Gradient Calculation

Why It's Important:

- Neural networks often contain billions to trillions of parameters (e.g., models with ~billions+parameters).
- During training, gradients need to be computed for every parameter at each iteration of the optimization process.

Challenges:

- **Computational Complexity:** Calculating gradients for all parameters in large-scale models is computationally intensive.
- **Memory Constraints:** Storing intermediate results for backpropagation in large models requires significant memory.

Solutions:

- **Backpropagation Algorithm:** Efficiently calculates gradients by applying the chain rule of differentiation.
- **Automatic Differentiation Libraries:** Frameworks like TensorFlow, PyTorch, and JAX automate gradient computation.
- **Distributed Training:** Parallelizing computations across multiple GPUs or TPUs helps handle large models.

Backpropagation Algorithm

2 Steps:

1. Forward Propagation

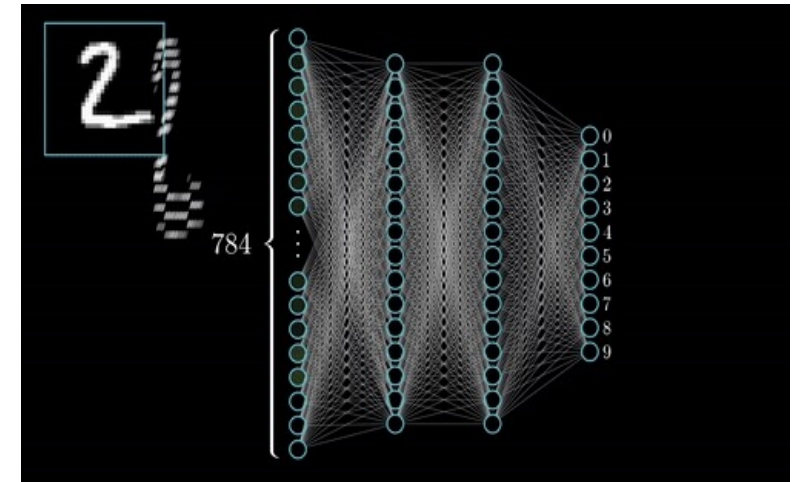
- Forward propagation is **how neural networks make predictions**.
- Involves passing input data through the network layer by layer to the output.

2. Backpropagation

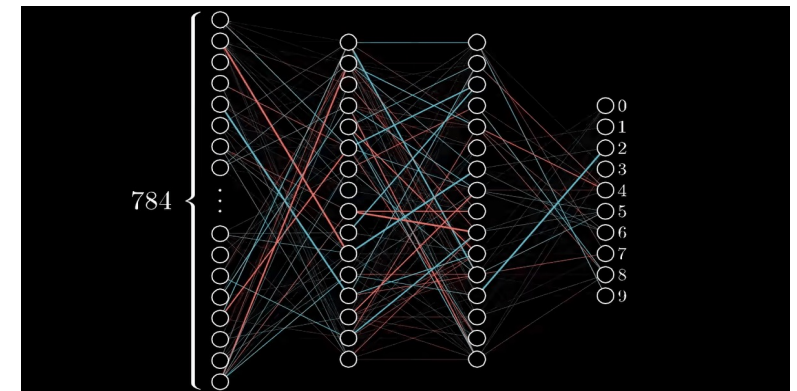
- Backpropagation is the process of adjusting the weights of the network by propagating through the neural network **backward**.
- Involves calculating the gradient of the loss function with respect to each weight by the chain rule.
- The weights are adjusted in the direction that reduces the loss.

Both steps are iteratively repeated for several epochs to minimize the loss and improve the model's accuracy.

Forward Propagation



Backpropagation

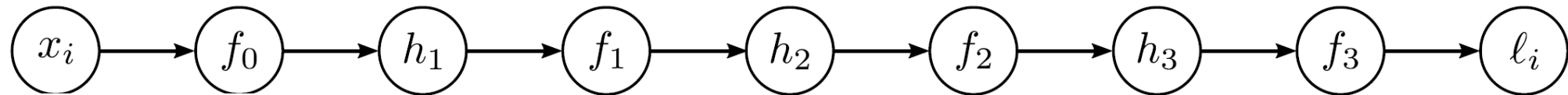


Backpropagation Algorithm

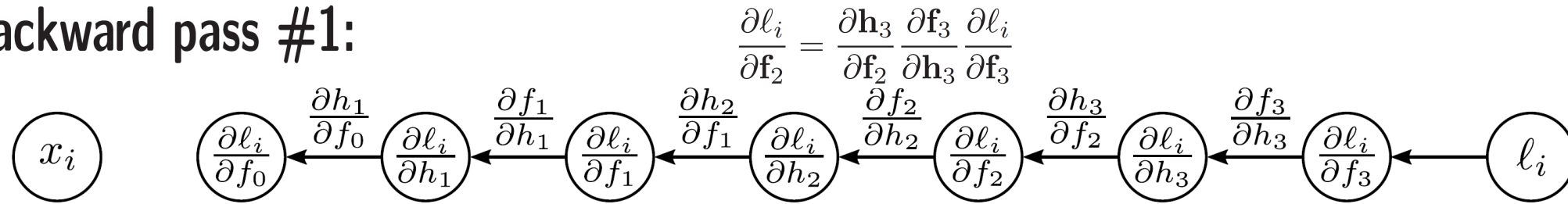
$$\ell_i = (f[x_i, \phi] - y_i)^2 \quad f[x, \phi] = \beta_3 + \omega_3 \cdot \cos[\beta_2 + \omega_2 \cdot \exp[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]]]$$

Forward pass:

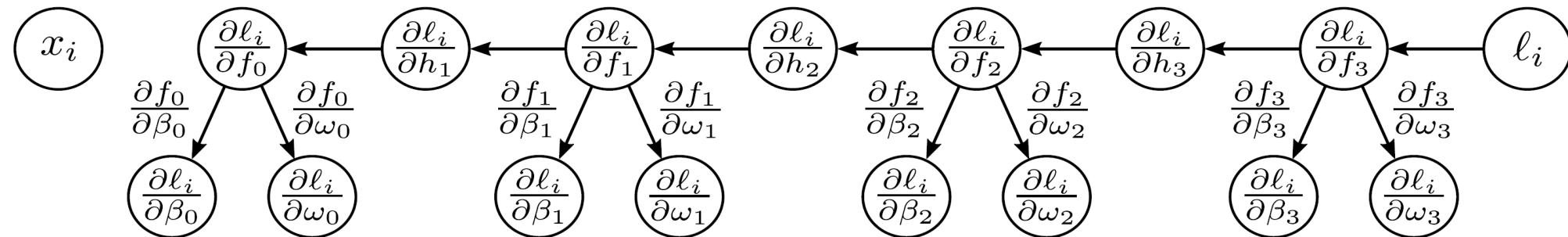
$$\begin{array}{llll} f_0 = \beta_0 + \Omega_0 x_i & f_1 = \beta_1 + \Omega_1 h_1 & f_2 = \beta_2 + \Omega_2 h_2 & f_3 = \beta_3 + \Omega_3 h_3 \\ h_1 = a[f_0] & h_2 = a[f_1] & h_3 = a[f_2] & \ell_i = l[f_3, y_i], \end{array}$$



Backward pass #1:



Backward pass #2:



Parameter Initialization

Proper initialization is critical because:

- a) **Convergence Speed:** Poor initialization can slow down the training process.
- b) **Gradient Stability:** Ensures gradients do not vanish or explode during backpropagation.
- c) **Optimization Performance:** Facilitates better navigation of the loss landscape, avoiding saddle points and bad local minima.

Challenges in Parameter Initialization:

- a. **Vanishing Gradients:** Occurs when the gradients become excessively small during backpropagation, leading to negligible weight updates. This is typically caused by: Small initial weight values and Activation functions like Sigmoid or Tanh that squash outputs to a narrow range.
- b. **Exploding Gradients:** Occurs when gradients grow exponentially during backpropagation, causing instability and divergence in the optimization process. This is typically caused by: Large initial weight values and Improper scaling of weights in deep layers.
- c. **Symmetry Breaking:** Initializing all weights to the same value (e.g., zero) causes symmetry in the network, preventing neurons in the same layer from learning distinct features.

Initialization Techniques

Zero Initialization: All weights set to 0, leading to symmetry.

Random Initialization: Weights are initialized randomly (e.g., sampled from $N(0, 1)$). Issue: Without proper scaling, it can lead to vanishing or exploding gradients.

Xavier Initialization (Glorot Initialization): Designed for Sigmoid and Tanh activation functions. Ensures variance of activations remains consistent across layers:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{\text{fan_in} + \text{fan_out}}}, \sqrt{\frac{6}{\text{fan_in} + \text{fan_out}}}\right)$$

- fan_in: Number of input connections.

He (Kaiming) Initialization: Designed for ReLU and its variants.

$$W \sim \mathcal{N}\left(0, \frac{2}{\text{fan_in}}\right)$$

- fan_out: Number of output connections.

LeCun Initialization: Suitable for activation functions like SELU: $W \sim \mathcal{N}\left(0, \frac{1}{\text{fan_in}}\right)$

Orthogonal Initialization: Ensures weights are orthogonal, maintaining variance stability across layers. Effective for RNNs and deep networks with large dimensions.

Bias Initialization: Biases are often initialized to small positive values (e.g., 0.01).

Pretrained Initializations: Using weights from pretrained models (transfer learning).

Layer-Specific Initialization: Input layers: Focus on uniform weight distribution.
Output layers: Smaller initialization to stabilize predictions.

Batch Normalization

- **Definition:** Batch Normalization (BN) is a technique used in deep learning to normalize the inputs to each layer within a neural network. It ensures that the inputs have a consistent distribution, which stabilizes and accelerates training.
- **Purpose:** **Reduce internal covariate shift:** This occurs when the distribution of inputs to a layer changes during training.
- **Benefits:**
 - a) **Improved Stability:** Keeps activations in a stable range, mitigating vanishing/exploding gradients.
 - b) **Faster Convergence:** Allows for higher learning rates and reduces sensitivity to initialization.
 - c) **Regularization Effect:** Adds noise due to batch statistics, reducing overfitting.
 - d) **Enhanced Generalization:** Produces better results on unseen data.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

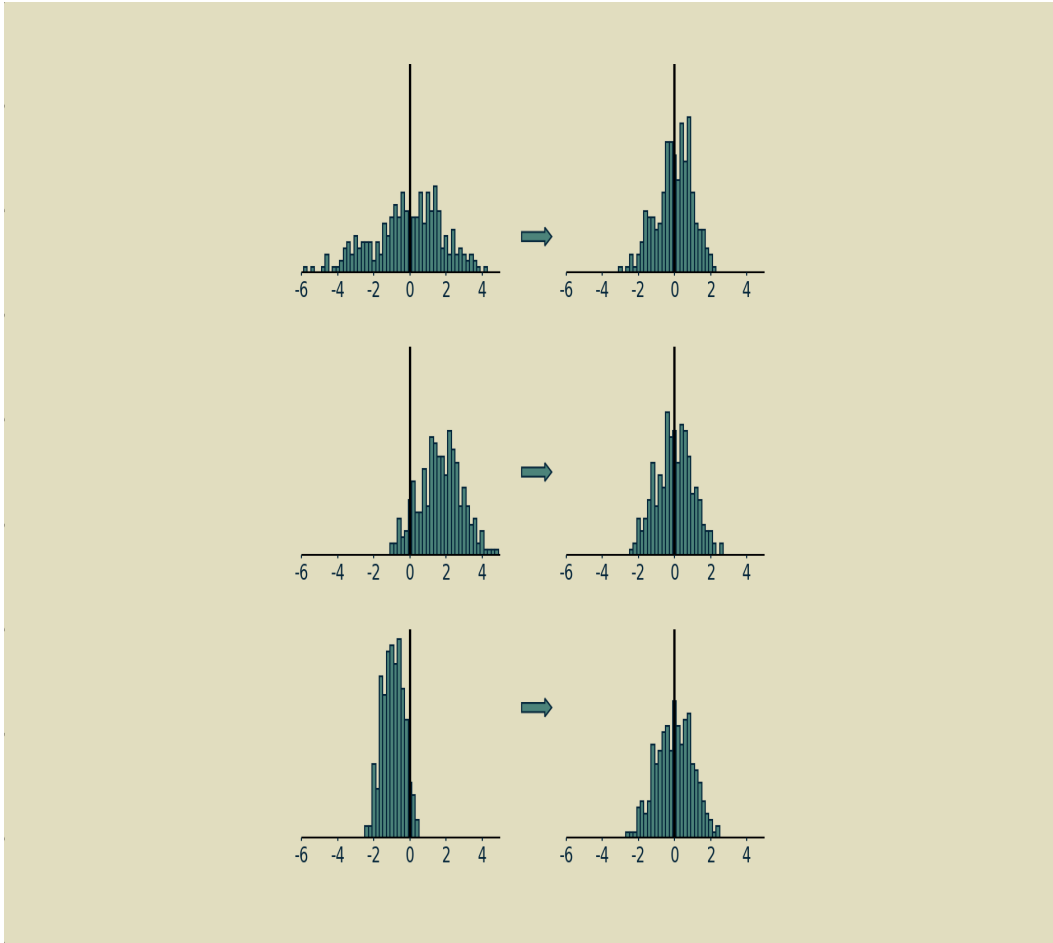
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

<https://kharshit.github.io/blog/2018/12/28/why-batch-normalization>

Batch Normalization

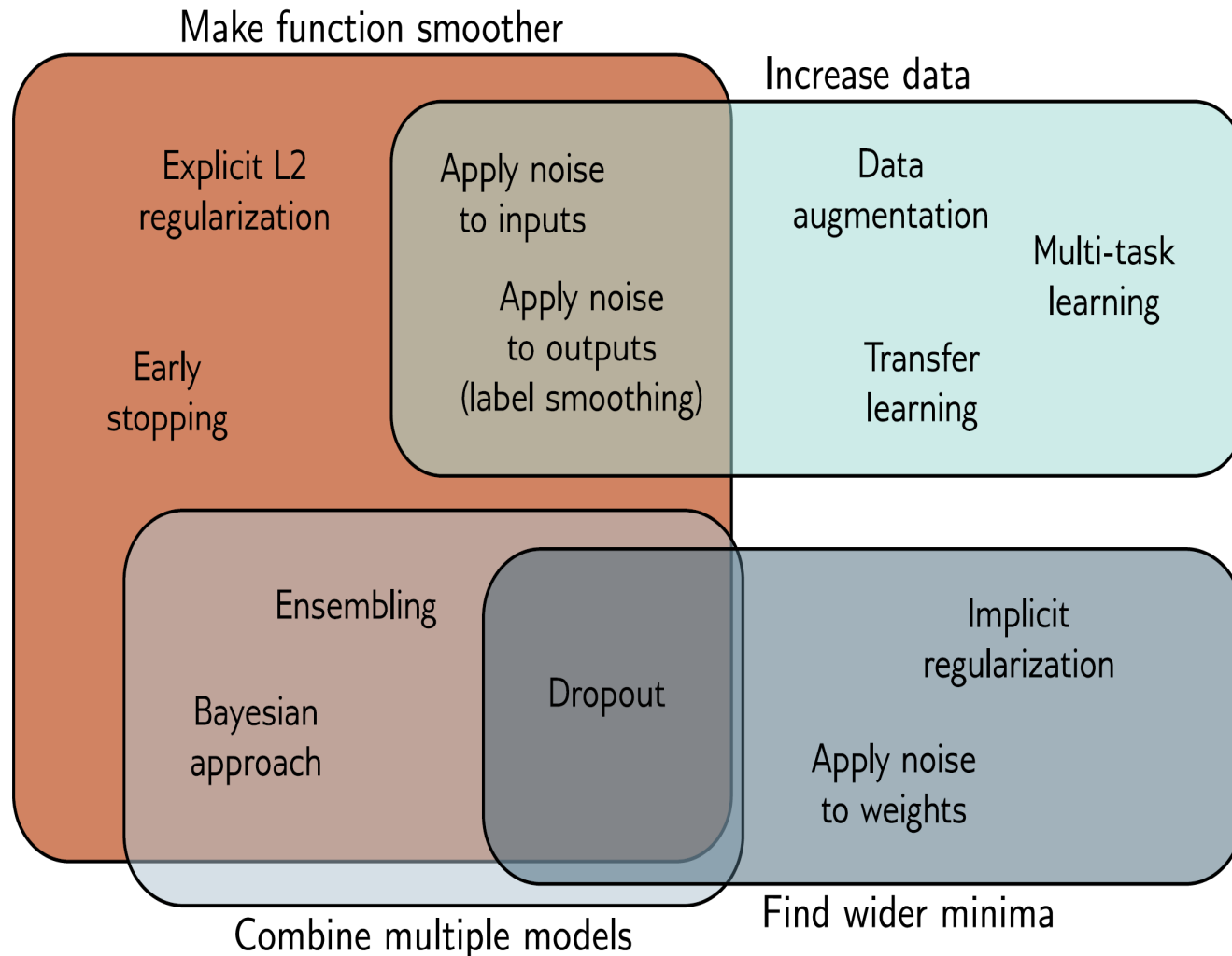


https://e2eml.school/batch_normalization



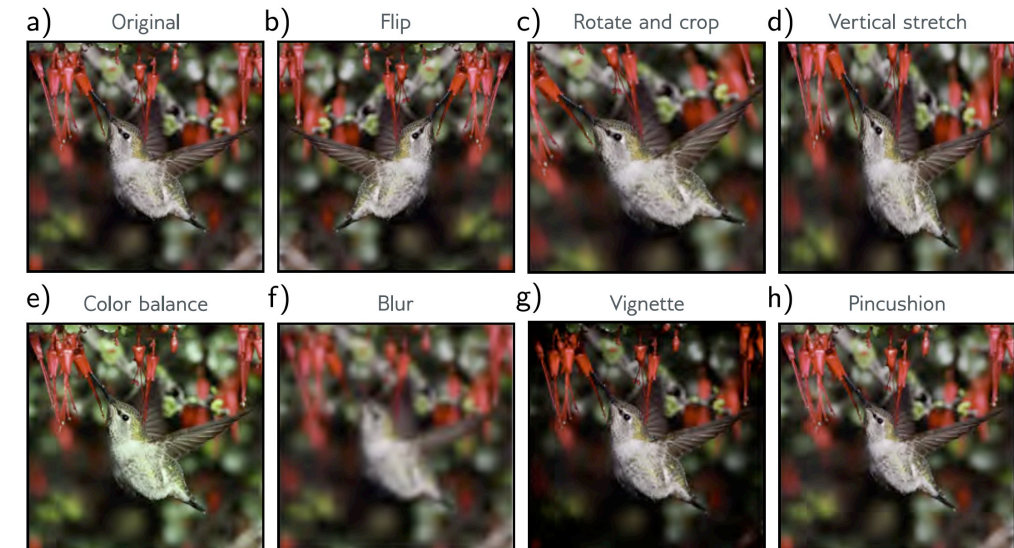
<https://kharshit.github.io/blog/2018/12/28/why-batch-normalization>

Regularization Methods



Four Mechanisms:

- ❖ Make the modeled function smoother.
- ❖ Increase the effective amount of data.
- ❖ Combine multiple models to mitigate uncertainty in the fitting process.
- ❖ Encourages the training process to converge to a wide minimum, where small errors in the estimated parameters are less important.



Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

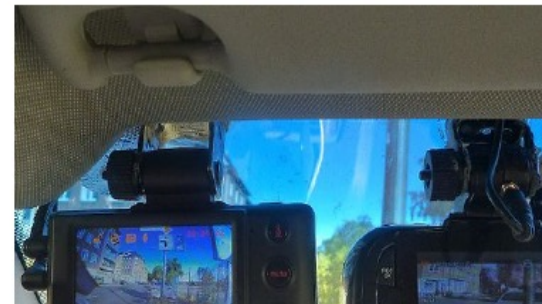
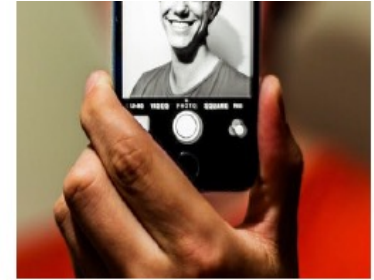
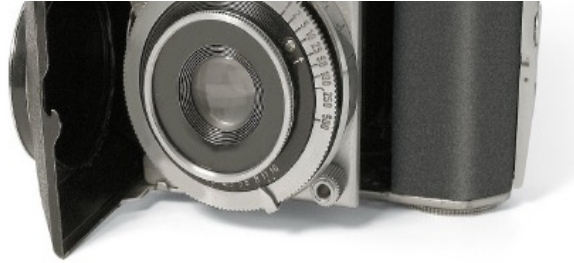
5 Optimization Techniques

6 Convolutional Neural Networks (CNN)

7 Graph Neural Networks (GNNs/GCNs)

8 Theoretical Properties

Nature Image Data is Everywhere



Major CV Tasks

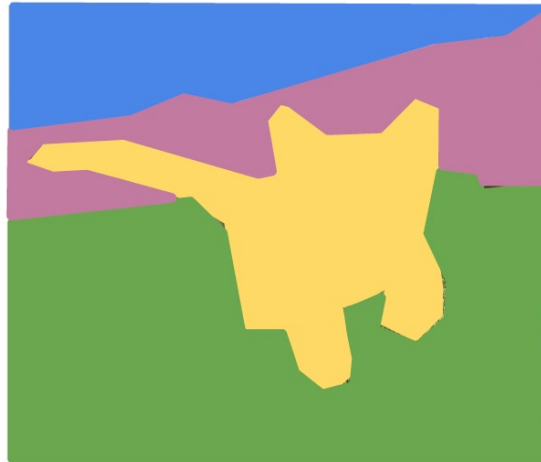
Classification



CAT

No spatial extent

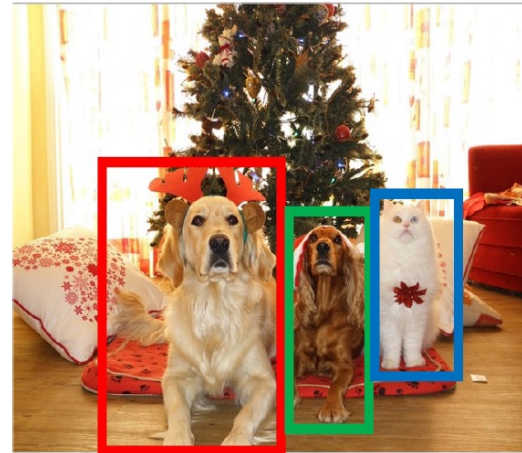
Semantic Segmentation



GRASS, CAT, TREE,
SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Object

Instance Segmentation



DOG, DOG, CAT

[This image is CC0 public domain](#)

Other CV Tasks

Video
Classification



Running? Jumping?

Multimodal Video
Understanding



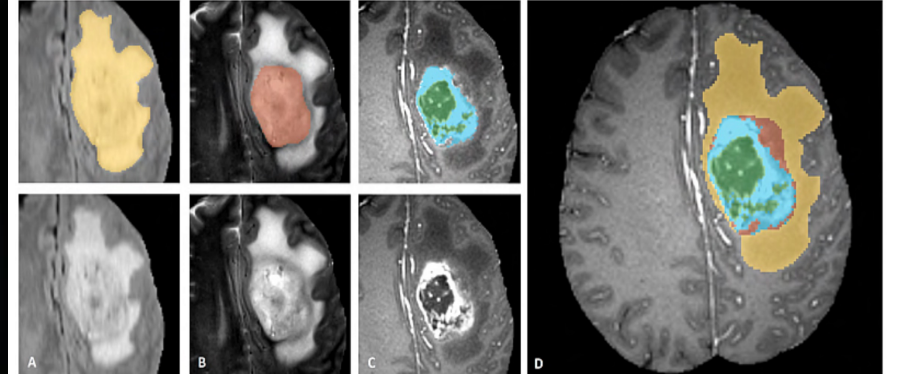
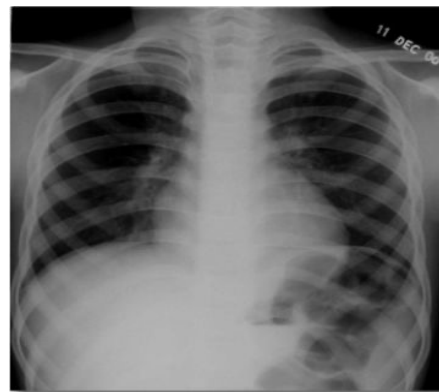
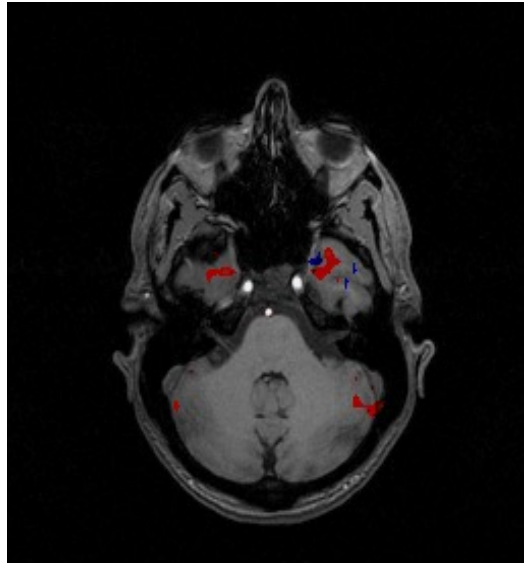
Visualization &
Understanding



Self-driving Cars



Medical Image Data is Everywhere



Scenario Challenges

Viewpoint variation



Scale variation



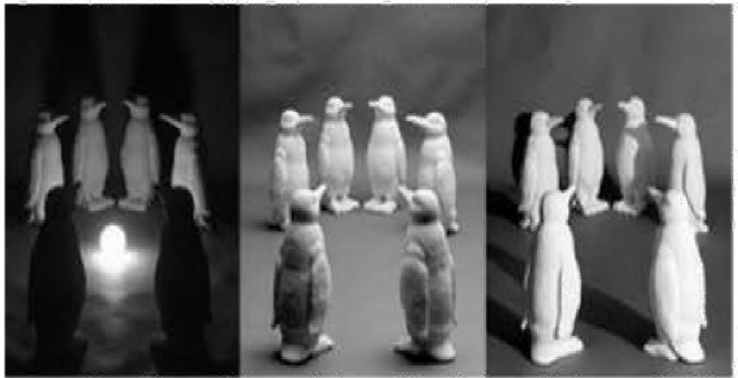
Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



High Dimensionality

- **Key Feature:**

- Images are inherently high-dimensional data. For example, a standard image in classification tasks with a resolution of 224×224 and 3 color channels (RGB) has $224 \times 224 \times 3 = 150,528$ input dimensions.
- Each pixel represents a separate input feature, and the number of features grows quadratically with image resolution.

- **Challenge:**

- Fully connected networks scale poorly with such high-dimensional data. For even a shallow network, the number of weights can exceed $150,528^2$ (~22 billion). This massive number of weights:
 - Increases the risk of overfitting, as more parameters require a proportional increase in training data.
 - Results in impractical memory and computational requirements, especially for larger images.
 - Slows down the training process significantly, making optimization difficult.

- **Real-World Implication:**

- As image resolution increases (e.g., 512×512 or beyond for high-definition images), the dimensionality becomes even more unmanageable for fully connected networks.

- **Solution:** CNNs reduce the number of parameters by using **shared weights (convolutional filters) and processing local regions of the image (kernels)**. This drastically decreases memory requirements and computational complexity.

Spatial Relationships in Pixels

- **Key Feature:**
 - Nearby pixels in an image are statistically correlated and form local patterns or textures (e.g., edges, corners, and gradients). These local relationships are critical for understanding the content of an image.
 - For example, in an image of a cat, nearby pixels may collectively form the texture of fur or the shape of an ear.
- **Challenge:**
 - Fully connected networks ignore spatial relationships by treating all input pixels equally. They lack the notion of "locality" and process the relationship between each pixel and every other pixel, regardless of their proximity.
 - This lack of spatial awareness means that a fully connected network cannot naturally exploit the structural dependencies within an image.
 - If the pixels of an image are randomly permuted in the same way for both training and testing, a fully connected network can still learn, highlighting its disregard for spatial coherence.
- **Real-World Implication:**
 - Without spatial awareness, models become inefficient and require a larger number of neurons to learn even basic patterns.
- **Solution:**
 - CNNs address this by using **local receptive fields** to capture spatial relationships. Filters (kernels) process small, overlapping regions of an image, preserving spatial coherence and focusing on local patterns. This makes CNNs particularly effective for tasks like object detection and image segmentation.

Stability Under Geometric Transformations

- **Key Feature:**
 - Images maintain their interpretation under geometric transformations such as translation, rotation, scaling, or flipping. For example:
 - A tree remains recognizable as a tree even if shifted slightly to the left or rotated by a small angle.
 - Similarly, a flipped or resized image of a cat does not change its underlying identity.
 - This invariance is essential for real-world applications like autonomous driving or medical imaging, where objects may appear in various positions or orientations.
- **Challenge:**
 - Fully connected networks treat each pixel independently and do not account for geometric transformations. A simple translation (e.g., shifting an image to the left by a few pixels) alters every pixel in the input vector, forcing the network to relearn patterns for each possible position.
 - This redundancy results in inefficient learning and requires significantly more data to cover all potential transformations.
- **Real-World Implication:**
 - Models that lack invariance to transformations are less robust in real-world scenarios where objects appear in varying contexts.
- **Solution:**
 - CNNs inherently address this issue by leveraging **translation invariance through shared filters**. These filters recognize patterns (e.g., edges or textures) regardless of their position within the image.
 - **Data augmentation techniques**, such as randomly rotating, flipping, or cropping images during training, further improve the model's ability to handle transformations.

Major Considerations

- **Noise in Images:**
 - Real-world images often contain noise (e.g., sensor artifacts, motion blur, or lighting variations). Fully connected networks struggle to differentiate between noise and meaningful patterns, further emphasizing the need for specialized architectures.
 - CNNs are more robust to noise due to their focus on local features rather than individual pixel values.
- **Scale and Hierarchy:**
 - Images often contain hierarchical features at multiple scales:
 - **Low-level** features: edges, corners.
 - **Mid-level** features: textures, patterns.
 - **High-level** features: objects or entire scenes.
 - Fully connected networks cannot naturally represent this hierarchy, while CNNs achieve this using multiple convolutional layers with increasing receptive fields.
- **Conclusion** The unique properties of unstructured image data pose significant challenges for fully connected networks. These challenges necessitate specialized architectures like CNNs, which **leverage shared weights, local receptive fields, and hierarchical feature extraction to process images efficiently**. Additionally, techniques like data augmentation and multi-scale analysis enhance the robustness of these models for real-world applications.

ImageNet

What is ImageNet?

- **Definition:** ImageNet is a large-scale visual database designed to advance research in object detection, classification, and other computer vision tasks.
- **Dataset Size:** It contains over **14 million labeled images** spanning **20,000+ categories**, with the most commonly used subset having **1,000 object categories**.

Key Features of ImageNet

a) Diversity of Classes:

Includes both broad categories (e.g., "dog," "car") and fine-grained subcategories (e.g., "golden retriever," "sports car").

b) Real-World Images:

Images collected from the internet represent real-world complexity, including cluttered backgrounds, occlusions, and multiple objects.

c) Hierarchical Organization:

Based on the WordNet hierarchy, where classes are semantically related, providing meaningful relationships between categories.

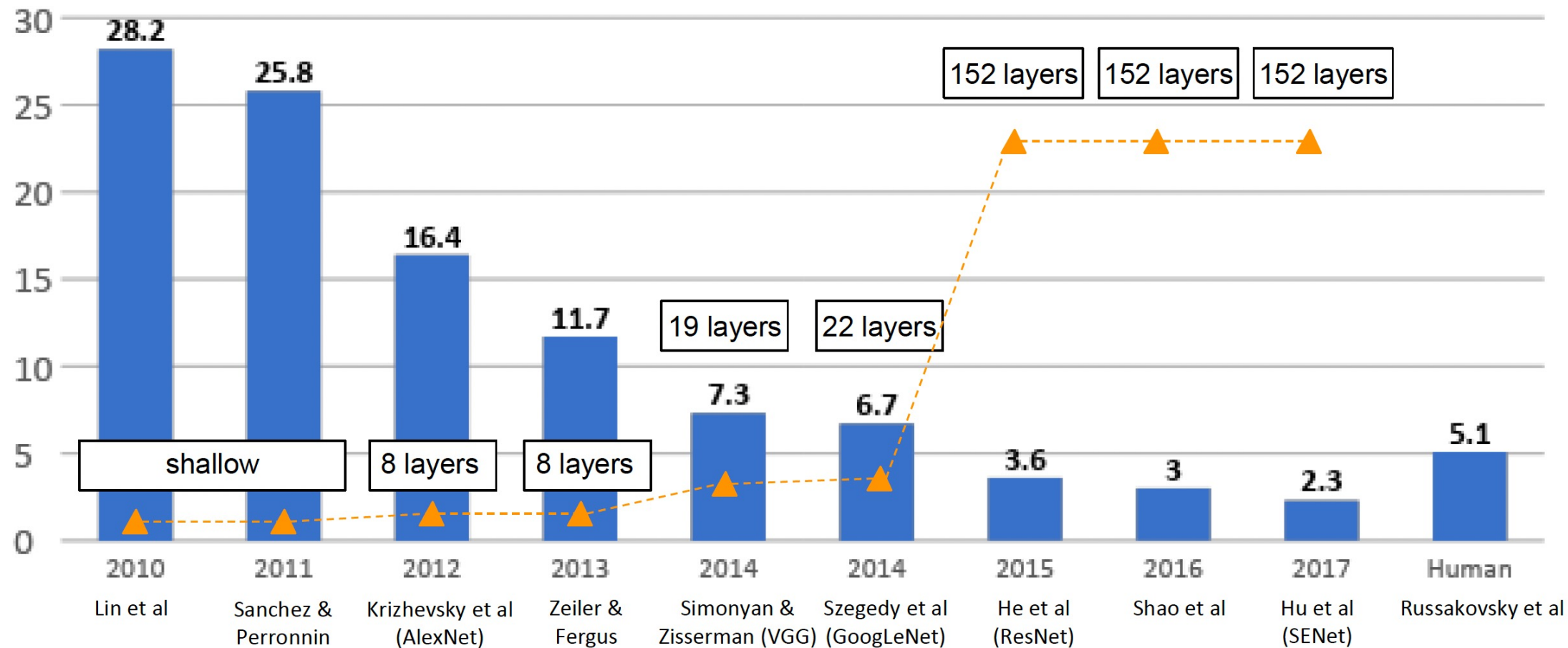


[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Introduction to CNN

What Are CNNs?

CNNs are specialized deep learning architectures designed to process data with grid-like structures, such as images and videos. By leveraging the spatial structure of data, CNNs efficiently extract and learn hierarchical features, making them particularly well-suited for computer vision tasks like image classification, object detection, and segmentation.

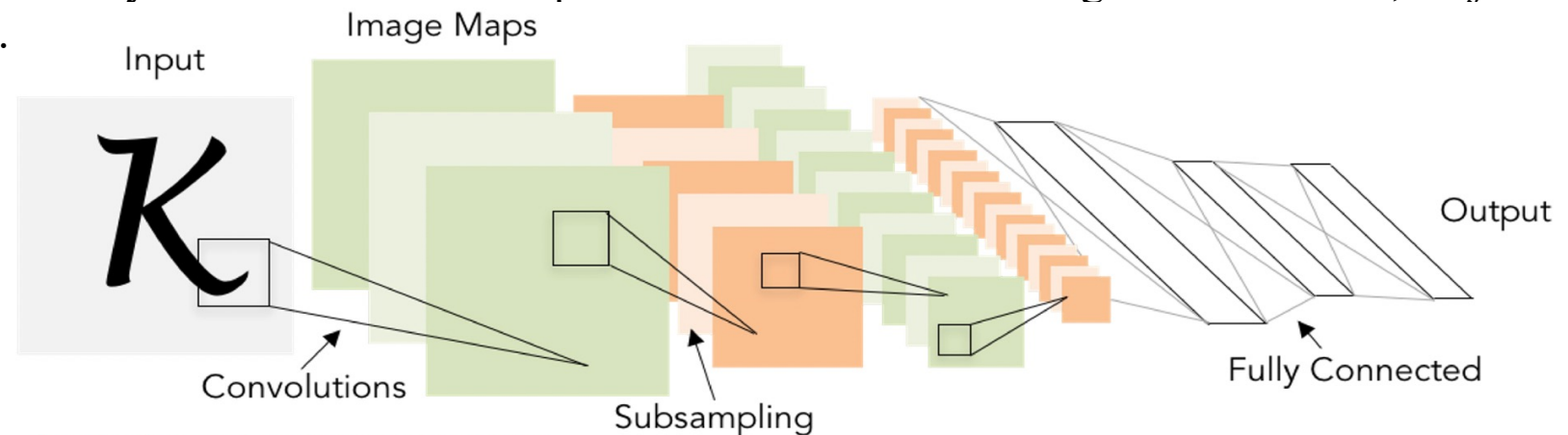


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNNs' applications

In **image and video processing**, they are widely used for tasks such as classification, object detection, segmentation, and face recognition.

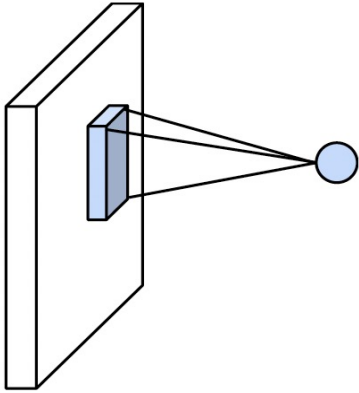
In **medical imaging**, CNNs assist in detecting tumors and anomalies in X-rays and CT scans.

In **natural language processing** (NLP), they process data as 1D inputs for tasks like sentence classification and text summarization.

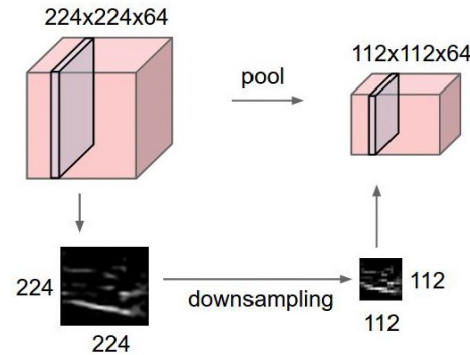
In **autonomous driving**, they enable real-time object detection for pedestrians, vehicles, and road signs.

Key Components of CNNs

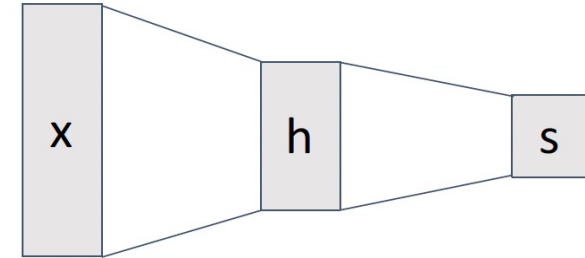
Convolution Layers



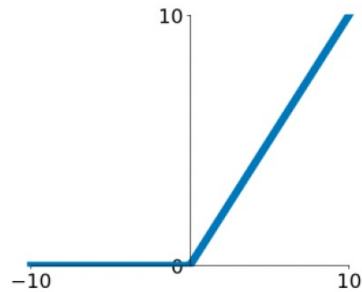
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Key Components of CNN

- Convolutional layers
- Rectified Linear Unit (ReLU)
- Pooling layers
- Fully connected layers

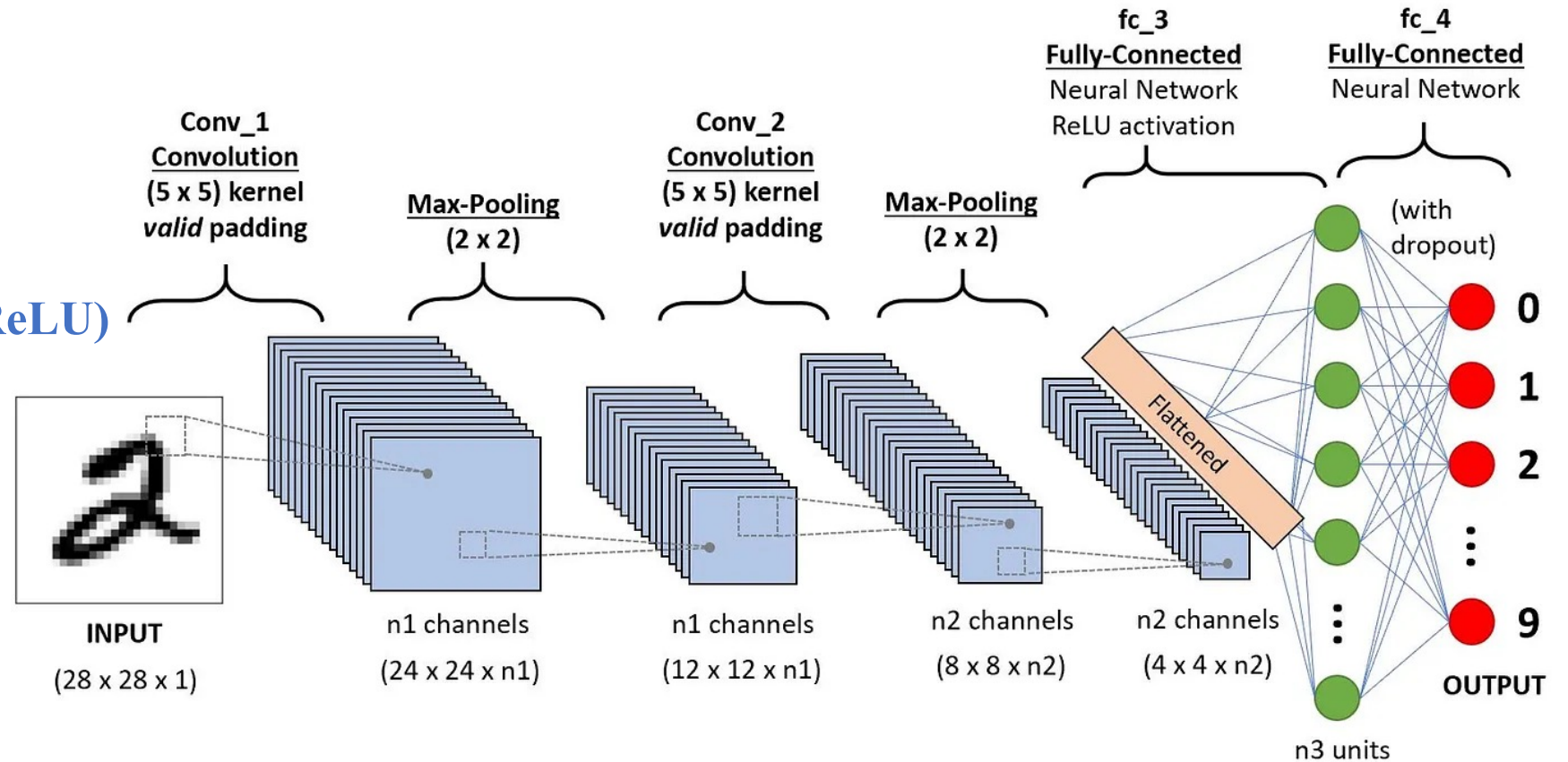
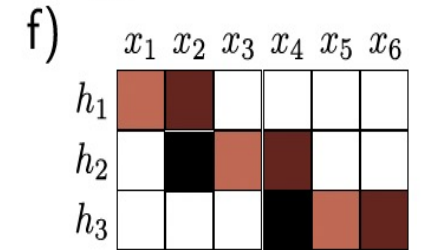
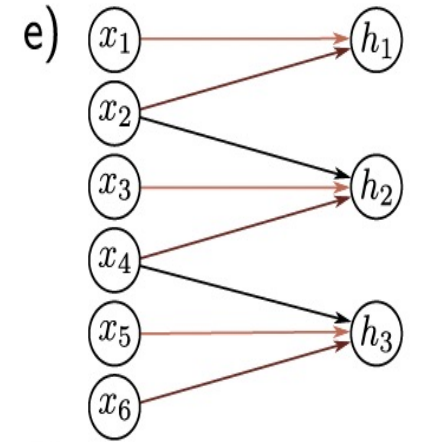
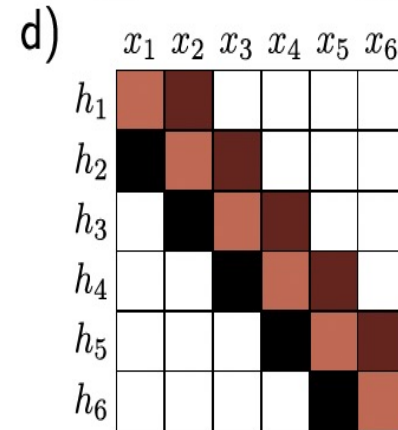
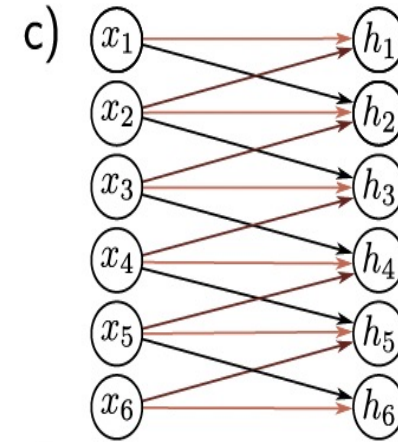
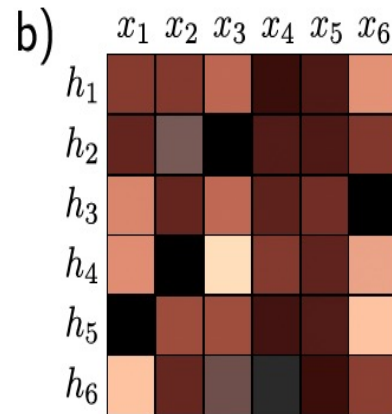
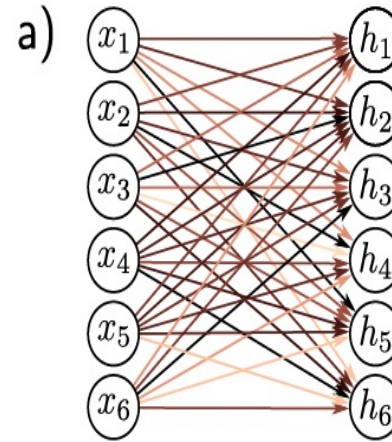


Illustration of architecture of CNNs applied to digit recognition ([source](#))

Feature Extraction Using Convolution

- **Input, kernel, and output**
- **Fully Connected Networks**
 - “fully connect” all the hidden units to all the input units. Only computationally feasible to learn features on the entire image for relatively small images.
 - order of 10^6 parameters to learn for 96x96 images. The feedforward and backpropagation computations would also be about 100 times slower, compared to 28x28 images.
- **Locally Connected Networks**



Prince (2023)

Feature Extraction Using Convolution

- **Input, kernel, and output (right figure)**
- **Fully Connected Networks**
- **Locally Connected Networks**
 - A simple solution to this problem is to limit connections between hidden and input units, allowing each hidden unit to connect to only a small subset of input units, such as a contiguous region of pixels. For other data types different than images like audio, hidden units can be connected to specific time spans. This concept of local connections is inspired by the visual cortex, where neurons respond to stimuli in specific locations.

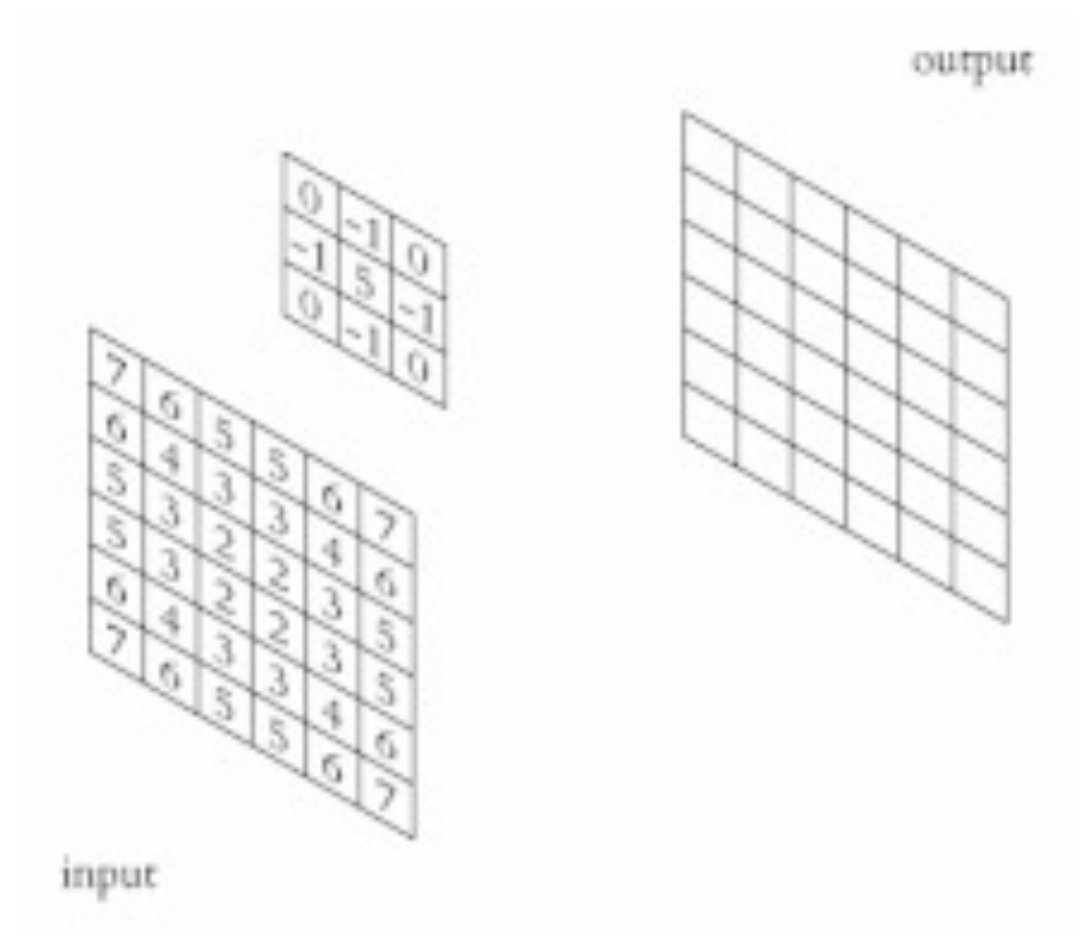


Illustration of Discrete 2D Convolution ([source](#))

Understanding the Convolution Operation

What is convolution?

Mathematically, Convolution is defined as $f, g: \mathbb{R}^n \rightarrow \mathbb{R}$:

$$(f * g)(x) = \int f(z)g(x - z)dz$$

Whenever we have discrete objects, the integral turns into a sum. For instance, in CNN, we used discrete convolution for vectors from the set of square-summable infinite-dimensional vectors defined as:

$$(f * g)(i) = \sum_a f(i)g(i - a)$$

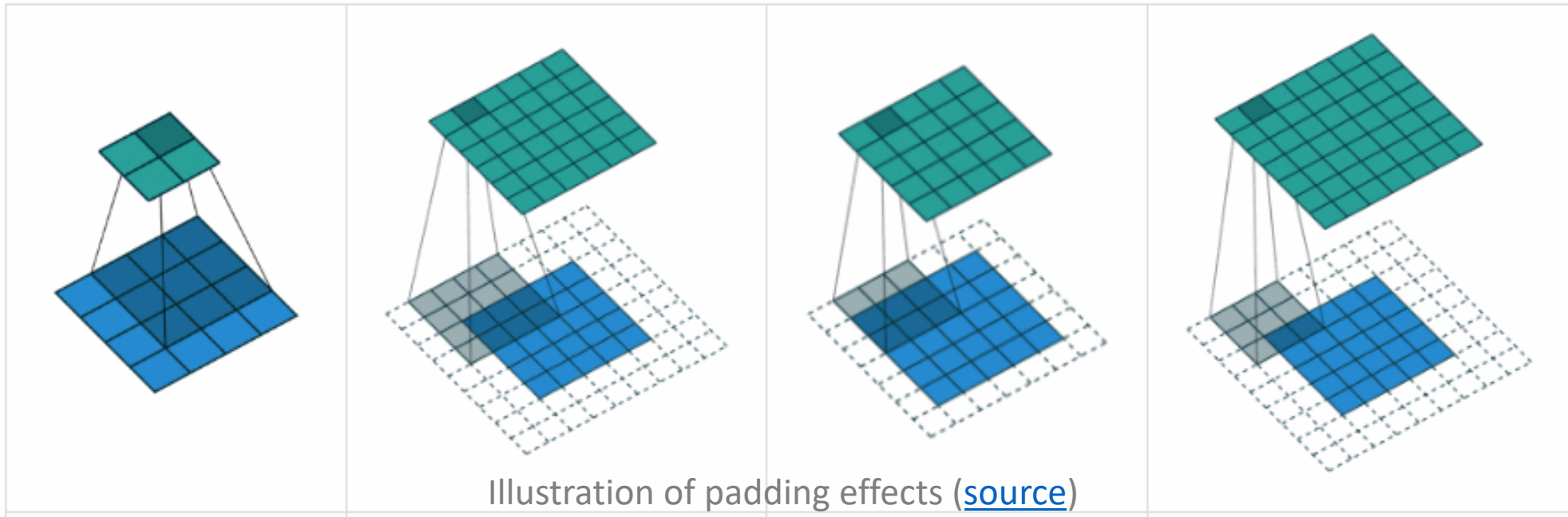
For two-dimensional tensors, we have a corresponding sum with (a,b) for f $(i-a,j-b)$ for g , respectively:

$$(f * g)(i, j) = \sum_a \sum_b f(a, b)g(i - a, j - b)$$

Padding, Stride, and Pooling

- **Padding**

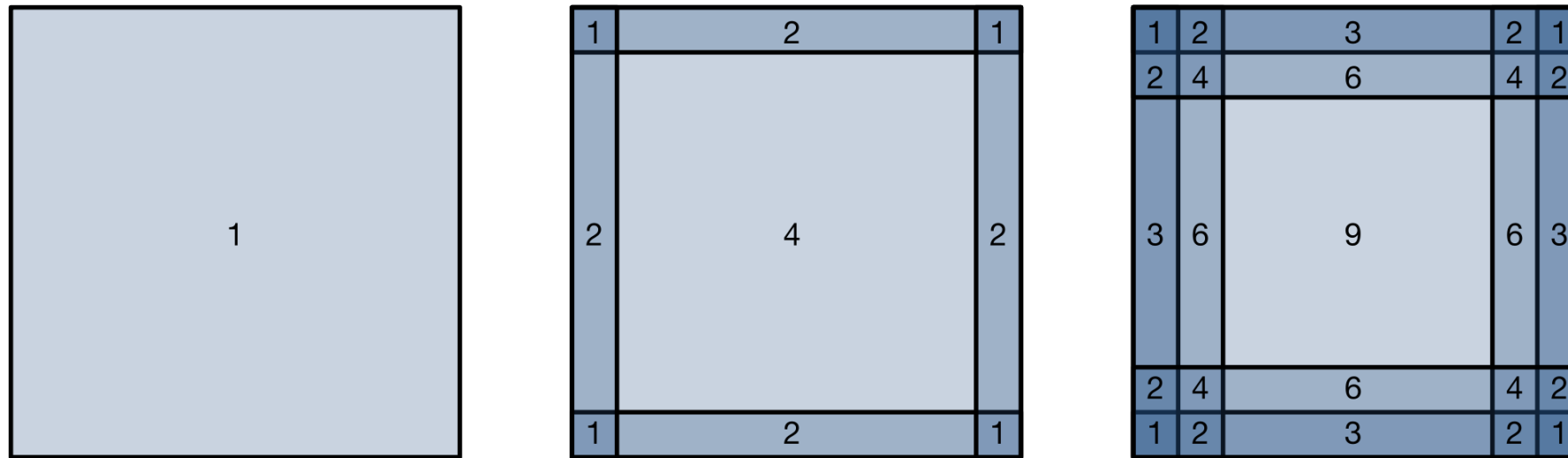
- Zero-padding and why it's necessary (The pixels at the corner in the previous images are less counted than those in the middle)
- How padding affects the dimensions of the output



Padding

One tricky issue when applying convolutional layers is that we tend to lose pixels on the perimeter of our image. The following figure depicts **the pixel utilization as a function of the convolution kernel size and the position within the image**.

We can see that the pixels in the corners are hardly used at all.

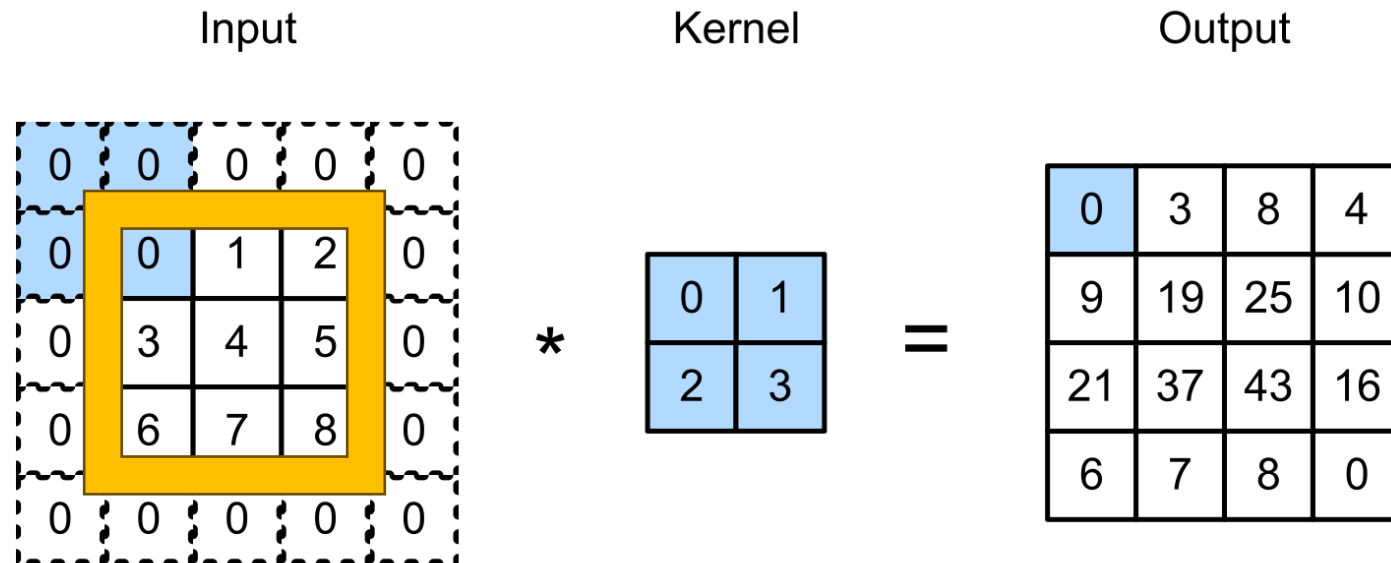


Pixel utilization for convolutions of 1x1, 2x2, and 3x3 respectively.

Padding

One straightforward solution to this problem is to add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image. Typically, we set the values of the extra pixels to zero.

Example on padding 3x3 input to 5x5 matrix:



Padding, Stride, and Pooling

Stride

- Example with stride of 1 vs. 2

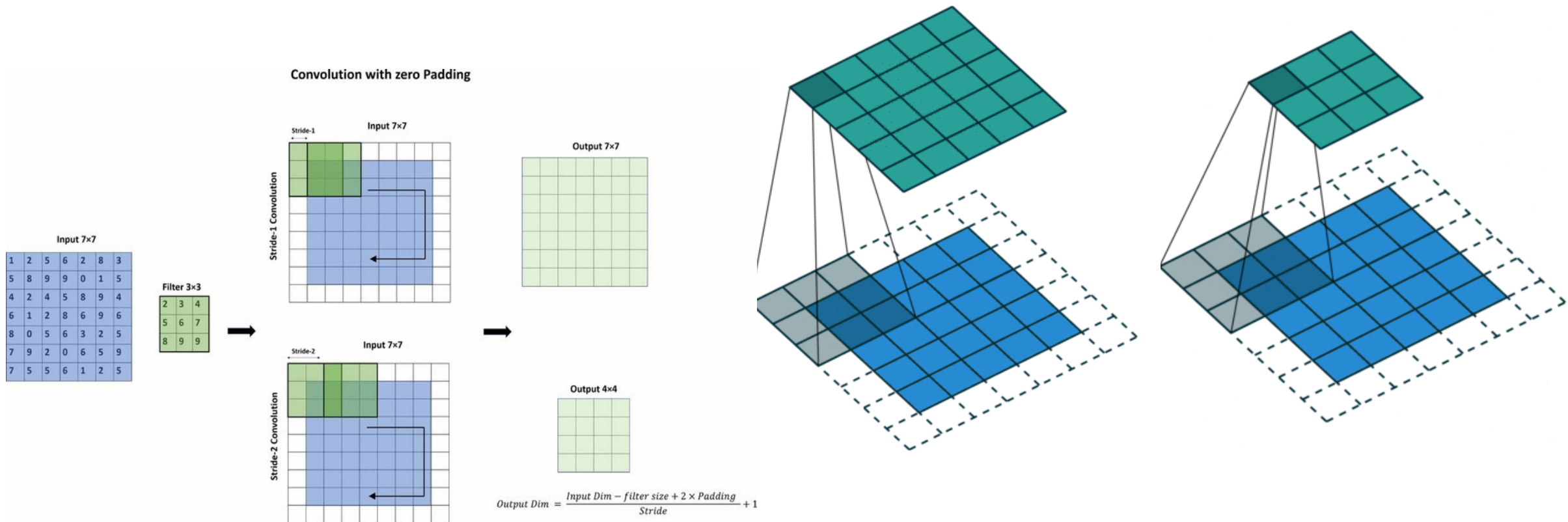
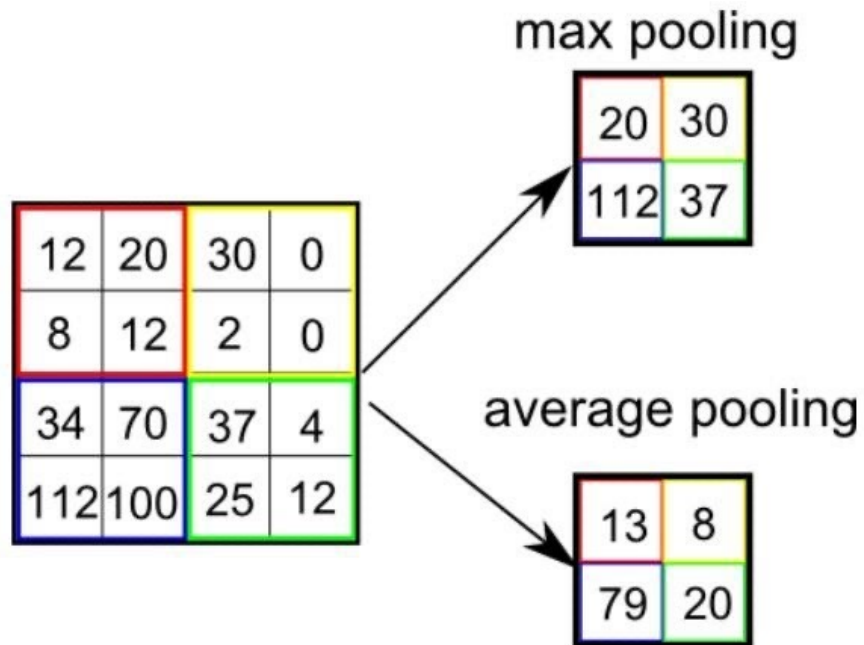


Illustration Convolution Operation with Stride Length = 1 Vs 2 ([source](#))

Padding, Stride, and Pooling

- **Pooling**

- Types: Max pooling, average pooling
- Role in reducing dimensionality
- Example: Pooling on an image



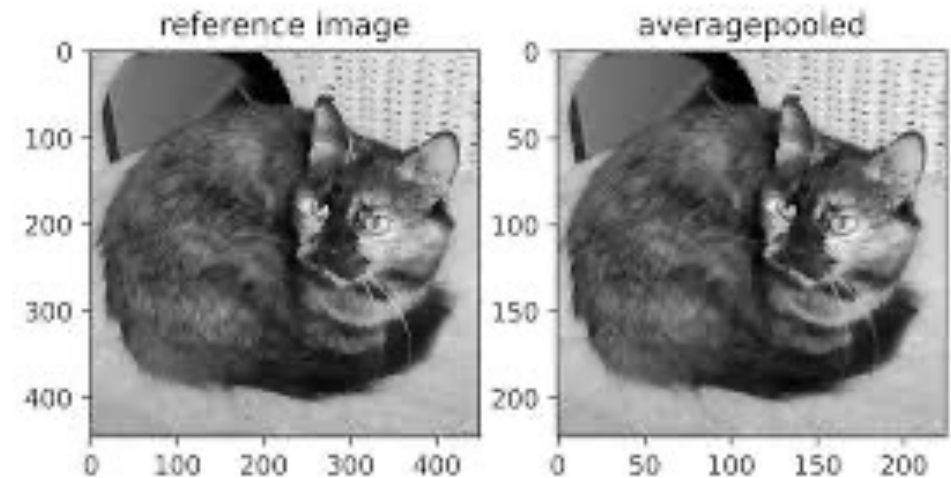
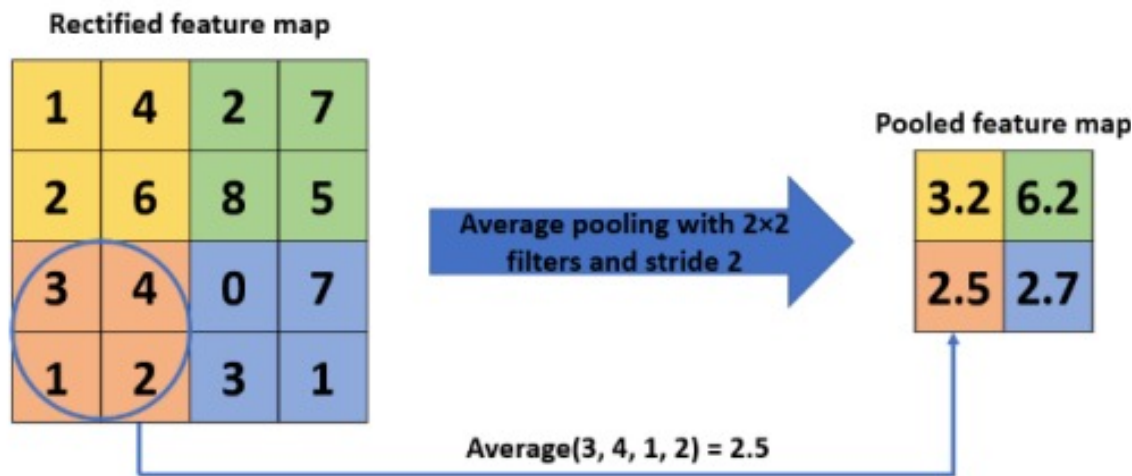
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Illustration of 3x3 pooling over 5x5 convolved feature ([source](#))

Average Pooling

Average pooling is essentially as old as CNNs. The idea is akin to downsampling an image. Rather than just taking the value of every second (or third) pixel for the lower resolution image, we can average over adjacent pixels to obtain an image with better signal-to-noise ratio since we are combining the information from multiple adjacent pixels.



<https://blog.paperspace.com/a-comprehensive-exploration-of-pooling-in-neural-networks/>

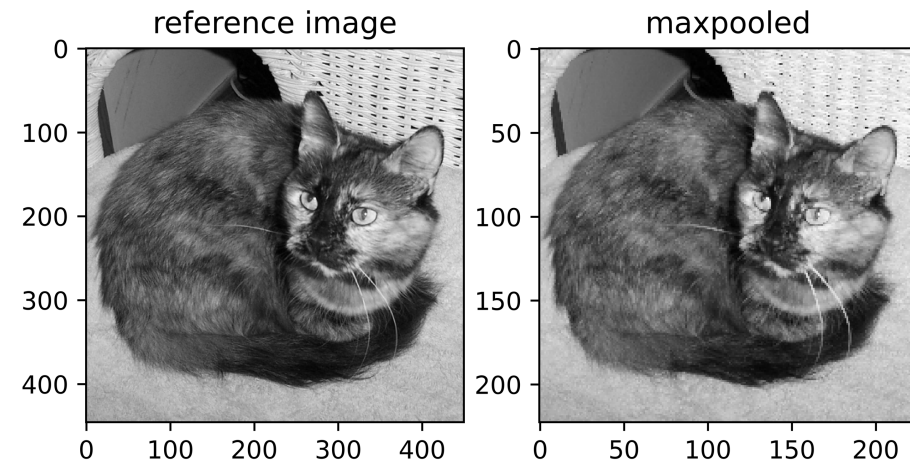
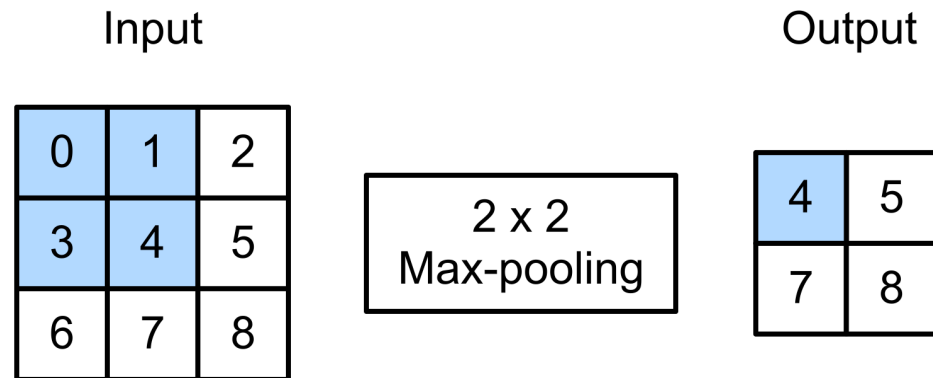
<https://www.digitalocean.com/community/tutorials/pooling-in-convolutional-neural-networks>

Maximum Pooling

Max-pooling was introduced in Riesenhuber and Poggio (1999) in the context of cognitive neuroscience to describe how information aggregation might be aggregated hierarchically for the purpose of object recognition; there already was an earlier version in speech recognition (Yamaguchi et al., 1990).

In almost all cases, max-pooling is preferable to average pooling.

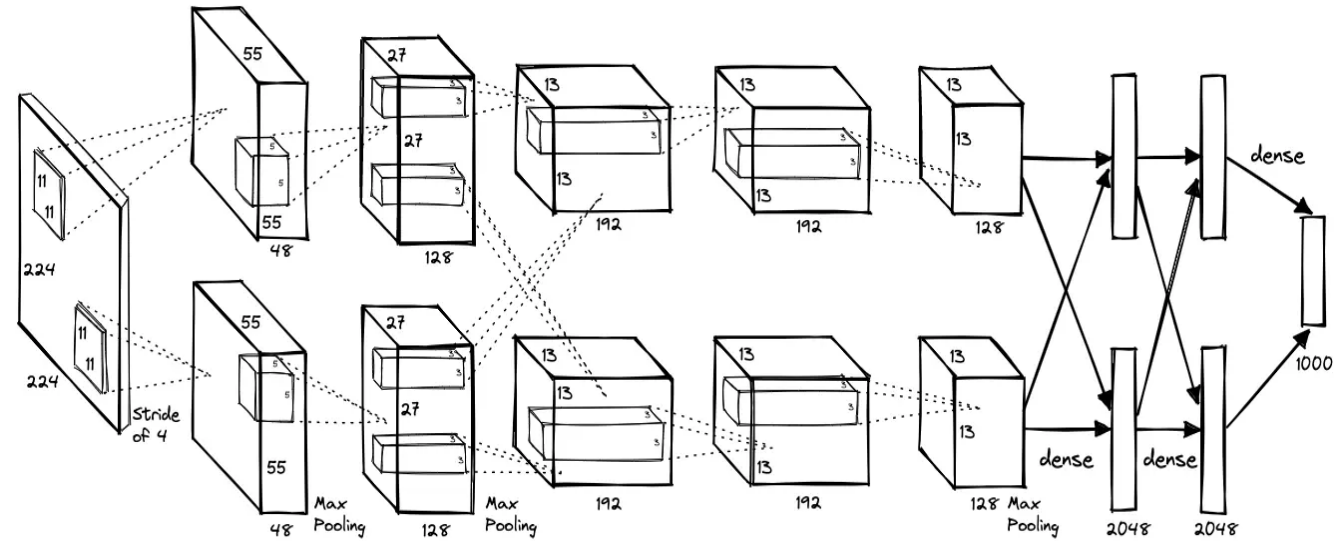
Consider example:



<https://www.digitalocean.com/community/tutorials/pooling-in-convolutional-neural-networks>

AlexNet

- With high performance hardware (GPUs from Nvidia) and sufficiently rich data-set, Krizhevsky et al. proposed AlexNet (Alom et al. 2018), which consists of **five convolution layers and three fully connected layers**.
- Each convolution layer contains a convolution kernel, a bias term, a ReLU activation function, and a local response normalization (LRN) module.
- In the 2012 ILSVRC, AlexNet won the competition with a Top-5 classification error rate of **16.4%**, became the dividing line between traditional and deep learning algorithms, and was the first deep CNN model in modern times.

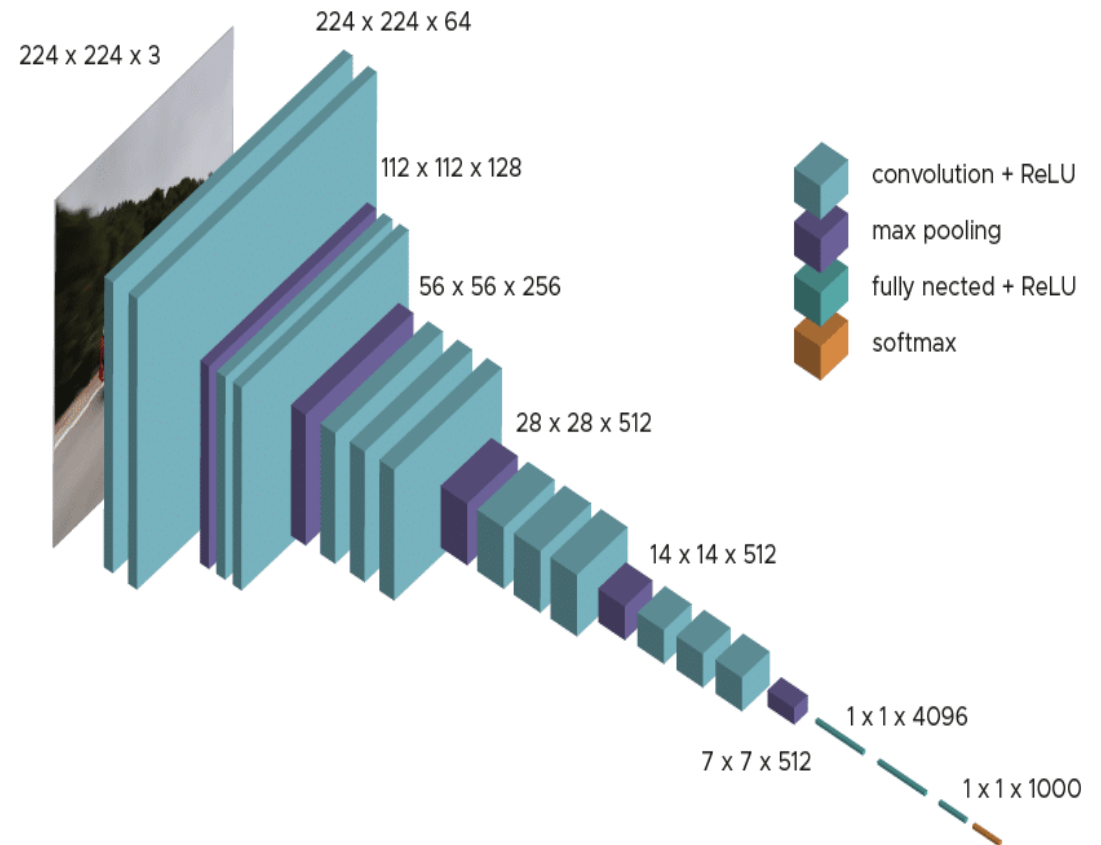


Layer	Type	Filter/Kernel Size	Number of Filters	Stride	Output Size
Input	RGB Image Input	-	-	-	$227 \times 227 \times 3$
Layer 1	Convolution + Max Pooling	11×11	96	4	$55 \times 55 \times 96$
Layer 2	Convolution + Max Pooling	5×5	256	1	$27 \times 27 \times 256$
Layer 3	Convolution	3×3	384	1	$13 \times 13 \times 384$
Layer 4	Convolution	3×3	384	1	$13 \times 13 \times 384$
Layer 5	Convolution + Max Pooling	3×3	256	1	$13 \times 13 \times 256$
Layer 6	Fully Connected	-	4096	-	4096
Layer 7	Fully Connected	-	4096	-	4096
Layer 8	Fully Connected (Output)	-	1000	-	1000 (class probabilities)

Table 1: Architecture of AlexNet (Rotated Table).

Visual Geometry Group (VGG) models

- To examine the impact of a CNN's depth on its accuracy, Karen Sengupta et al. (2019) conducted a comprehensive evaluation of the performance of network models with increasing depth, while using **smaller convolution filters (3×3)** instead of the previous 5×5 kernels and proposed a series of Visual Geometry Group (VGG) models in 2014.
- The smaller kernel size lowers the computational complexity and the number of training parameters.
- Simultaneously, VGG supports the hypothesis that performance can be enhanced by continually deepening the network topology.
- In the 2014 ILSVRC, VGG won the competition in the **Localization Task** with a Top-5 classification error rate of **7.3%**,



Sengupta et al. *Front Neurosci* (2019)

VGG Models

Layer Type	Filters	Kernel Size	Stride	Padding	Output Size
Input	-	-	-	-	$224 \times 224 \times 3$
Conv + ReLU	64	3×3	1	1	$224 \times 224 \times 64$
Conv + ReLU	64	3×3	1	1	$224 \times 224 \times 64$
Max Pooling	-	2×2	2	0	$112 \times 112 \times 64$
Conv + ReLU	128	3×3	1	1	$112 \times 112 \times 128$
Conv + ReLU	128	3×3	1	1	$112 \times 112 \times 128$
Max Pooling	-	2×2	2	0	$56 \times 56 \times 128$
Conv + ReLU	256	3×3	1	1	$56 \times 56 \times 256$
Conv + ReLU	256	3×3	1	1	$56 \times 56 \times 256$
Conv + ReLU	256	3×3	1	1	$56 \times 56 \times 256$
Max Pooling	-	2×2	2	0	$28 \times 28 \times 256$
Conv + ReLU	512	3×3	1	1	$28 \times 28 \times 512$
Conv + ReLU	512	3×3	1	1	$28 \times 28 \times 512$
Conv + ReLU	512	3×3	1	1	$28 \times 28 \times 512$
Max Pooling	-	2×2	2	0	$14 \times 14 \times 512$
Conv + ReLU	512	3×3	1	1	$14 \times 14 \times 512$
Conv + ReLU	512	3×3	1	1	$14 \times 14 \times 512$
Conv + ReLU	512	3×3	1	1	$14 \times 14 \times 512$
Max Pooling	-	2×2	2	0	$7 \times 7 \times 512$
Flatten	-	-	-	-	25088
Fully Connected	-	-	-	-	4096
Fully Connected	-	-	-	-	4096
Output (Softmax)	-	-	-	-	1000

Table 1: VGG-16 Architecture: Layers, filters, and output sizes.

a) Increased Depth:

Depth allows VGG to learn hierarchical features, improving accuracy.

b) Simple Design:

Stacks of identical convolutional layers make it easy to scale the architecture.

c) Transfer Learning:

VGG models pretrained on ImageNet are widely used for transfer learning in other tasks.

d) Small Filters:

Using 3×3 filters results in fewer parameters compared to larger filters, while maintaining the receptive field size.

e) VGG-16:

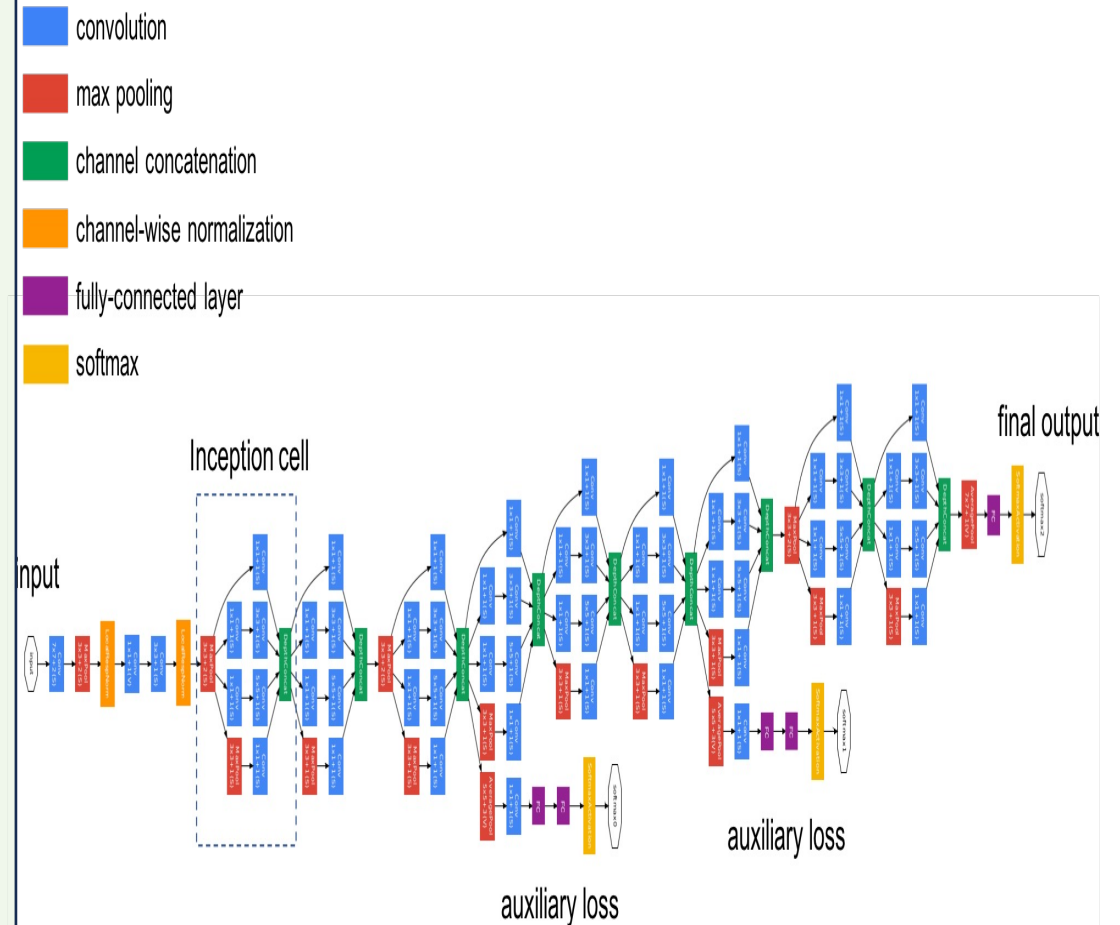
16 layers: 13 convolutional layers and 3 fully connected layers.
Parameters: ~138 million.

f) VGG-19:

19 layers: 16 convolutional layers and 3 fully connected layers.
Parameters: ~143 million.

GoogLeNet

- GoogleNet, also known as **Inception-v1**, is a deep CNN introduced by Szegedy et al. in 2014.
- It won the **ILSVRC 2014 the Classification Task** with a top-5 error rate of **6.67%**, outperforming other models.
- **Main Innovations:**
 - a) **Inception Module** enables the network to capture features at multiple scales while reducing computational cost.
 - b) **Dimension Reduction.** Uses 1×1 convolutions for reducing dimensionality before applying larger filters, significantly reducing parameters.
 - c) **Auxiliary Classifiers:** Two intermediate softmax classifiers are added to help with gradient flow and prevent vanishing gradients.
- **Motivation:** Despite having 22 layers, GoogleNet has only **~5M parameters**, significantly fewer than AlexNet (~60M) and VGG-16 (~138M). This is achieved using 1×1 convolutions for dimensionality reduction.



Szegedy et al. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015)

GoogleNet Architecture

Layer Type	Filters/Units	Kernel Size	Stride	Padding	Output Size
Input	-	-	-	-	$224 \times 224 \times 3$
Conv + ReLU	64	7×7	2	3	$112 \times 112 \times 64$
Max Pooling	-	3×3	2	0	$56 \times 56 \times 64$
Conv + ReLU	64	1×1	1	0	$56 \times 56 \times 64$
Conv + ReLU	192	3×3	1	1	$56 \times 56 \times 192$
Max Pooling	-	3×3	2	0	$28 \times 28 \times 192$
Inception Module 1	-	Multi-scale	-	-	$28 \times 28 \times 256$
Inception Module 2	-	Multi-scale	-	-	$28 \times 28 \times 480$
Max Pooling	-	3×3	2	0	$14 \times 14 \times 480$
Inception Module 3	-	Multi-scale	-	-	$14 \times 14 \times 512$
Inception Module 4	-	Multi-scale	-	-	$14 \times 14 \times 512$
Inception Module 5	-	Multi-scale	-	-	$14 \times 14 \times 528$
Auxiliary Classifier 1	1000	-	-	-	1000
Inception Module 6	-	Multi-scale	-	-	$14 \times 14 \times 832$
Auxiliary Classifier 2	1000	-	-	-	1000
Inception Module 7	-	Multi-scale	-	-	$7 \times 7 \times 1024$
Global Average Pooling	-	7×7	-	-	$1 \times 1 \times 1024$
Fully Connected	1000	-	-	-	1000

Table 1: GoogleNet Architecture: Layers, filters, and output sizes.

- **Input Layer:** $224 \times 224 \times 224 \times 3$ RGB image.
 - **Convolutional Layers:** Apply 7×7 , 1×1 , or 3×3 filters to extract features.
 - **Inception Modules:** Multi-scale processing with 1×1 , 3×3 , 5×5 , and pooling operations.
 - **Auxiliary Classifiers:** Intermediate softmax layers for training regularization.
 - **Global Average Pooling:** Replaces fully connected layers with spatial pooling across feature maps.
- Output Sizes:**
- The output size at each stage is shown, demonstrating how spatial dimensions decrease progressively.

Inception Module

1. Input feature map with dimensions $H \times W \times C_{in}$.
2. Four parallel paths:
 - 1×1 convolution.
 - 1×1 convolution followed by 3×3 convolution.
 - 1×1 convolution followed by 5×5 convolution.
 - Max pooling followed by 1×1 convolution.
3. Concatenate the outputs to produce a feature map with dimensions $H \times W \times (C_1 + C_2 + C_3 + C_4)$.

Input: Feature map dimensions $H \times W \times C_{in} = 28 \times 28 \times 192$.

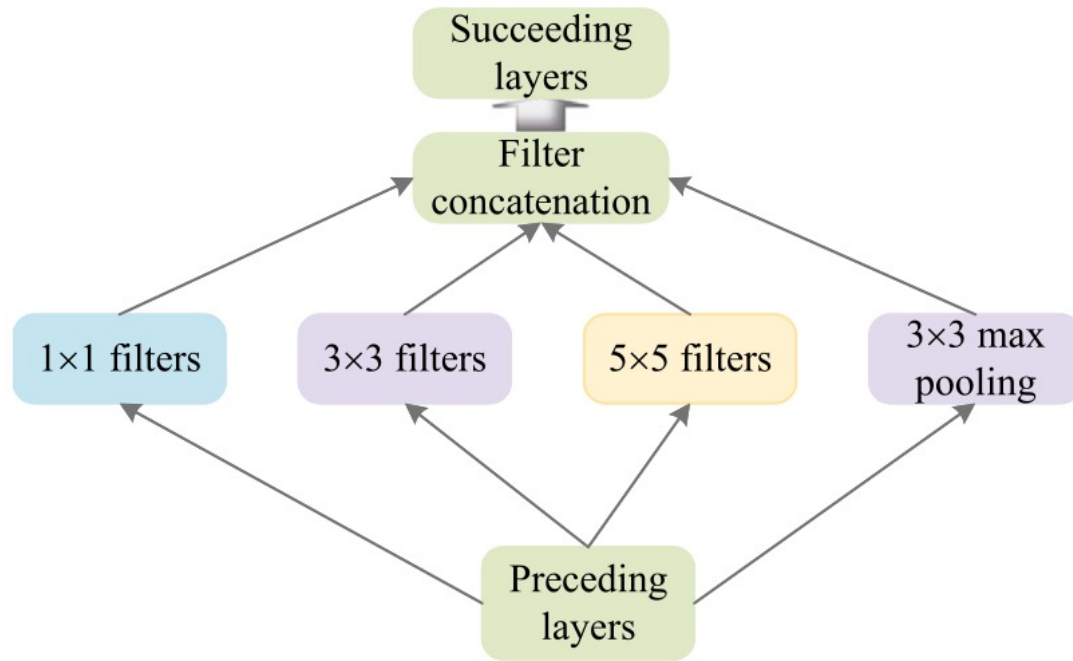
Paths:

- 1×1 Convolution: 64 filters, output $28 \times 28 \times 64$.
- $1 \times 1 + 3 \times 3$ Convolution: 96 and 128 filters, output $28 \times 28 \times 128$.
- $1 \times 1 + 5 \times 5$ Convolution: 16 and 32 filters, output $28 \times 28 \times 32$.
- Max Pooling + 1×1 Convolution: 32 filters, output $28 \times 28 \times 32$.

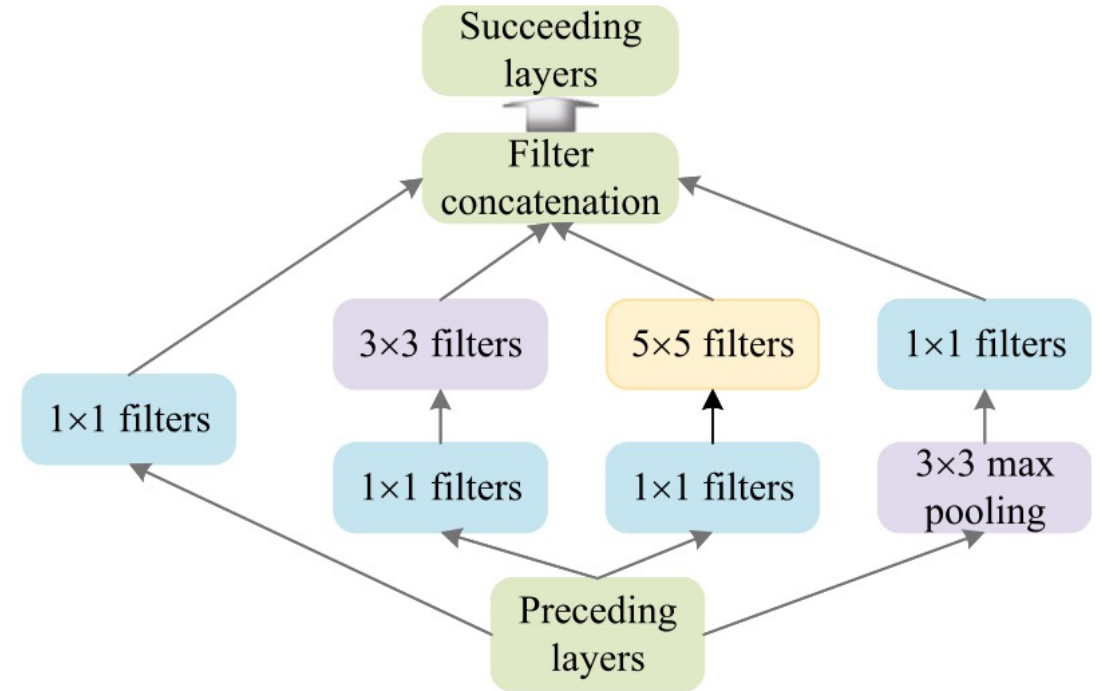
Concatenation: Final output $28 \times 28 \times (64 + 128 + 32 + 32) = 28 \times 28 \times 256$.

- **Multi-Scale Feature Extraction:** Processes feature maps at multiple scales for rich. representations.
- **Dimensionality Reduction:** 1×1 convolutions reduce computational costs while preserving important information.
- **Efficiency:** Deep networks can process large input data with fewer parameters compared to traditional architectures.
- **Improved Generalization:** Captures features across different abstraction levels.

Inception Cell



(a) Architecture of inception



(b) Architecture of inception V1

Example architecture of inception
Zhao et al. *Artificial Intelligence Review* (2024)

Residual network (ResNet)

- **Degradation Problem:** Deeper networks (e.g., >20 layers) suffered from degradation of accuracy, not just overfitting, but actual performance decline.
- **Key Idea:** Instead of learning the direct mapping ($H(x)$), ResNet learns the **residual mapping** ($F(x)=H(x)-x$). This simplifies optimization and allows gradients to flow through skip connections, improving convergence.
- **Impact:**
 - **Ease of Optimization:** Learning residuals is simpler than learning direct mappings.
 - **Deeper Architectures:** ResNet-152 outperforms shallower networks while maintaining high accuracy.
 - **State-of-the-art Results:** Top-5 error dropped to **~3.6%** on ImageNet (ILSVRC).
- **Connection to Highway Networks (Srivastava et al., 2015):** ResNet can be seen as a special, simplified case of highway layers where gates are mostly open.
- Residual connections enable building much deeper and more powerful networks by addressing gradient vanishing and “degradation” issues.

Residual Block

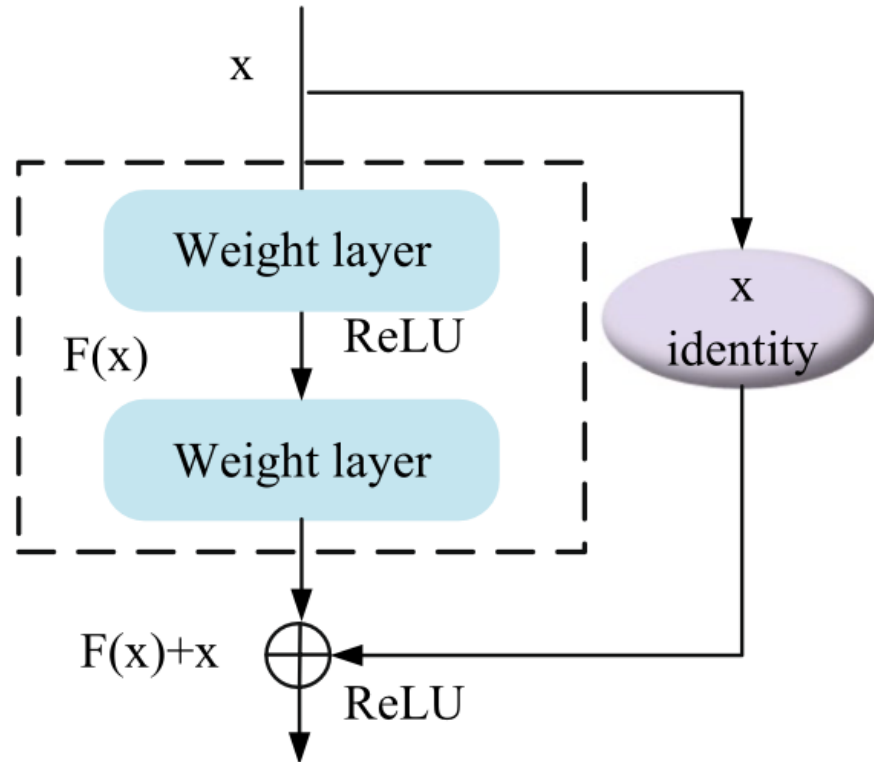


Illustration of a residual block

Zhao et al. *Artificial Intelligence Review* (2024)

Building Block:

- a) **Input:** x (feature map from the previous layer).
- b) **Path 1 (Residual Function):**
 - i. 3×3 convolution \rightarrow Batch Normalization \rightarrow ReLU.
 - ii. 3×3 convolution \rightarrow Batch Normalization.
- c) **Path 2 (Skip Connection):**
 - i. Identity mapping: Directly passes the input x .
- d) **Addition:**
 - i. Output: $F(x)+x$ (summation of the two paths).
- e) **Activation:**
 - i. Apply ReLU to the combined output.
- f) **Output:**
Final feature map retains the same dimensions as the input.

CNN Optimization Techniques

CNN optimization involves techniques to improve the performance, efficiency, and generalization of Convolutional Neural Networks during training and inference.

- **Goals:**

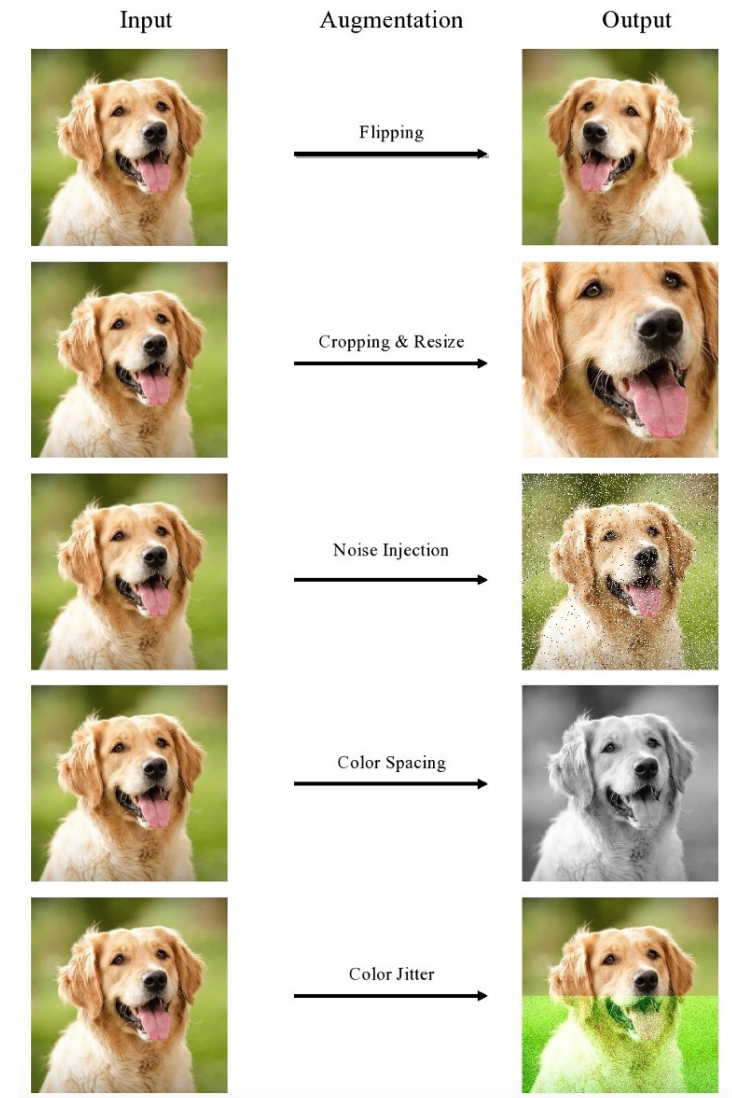
- a) Reduce overfitting.
- b) Improve convergence speed.
- c) Optimize computational resources.

- **Common Strategies:**

- a) Data Augmentation
- b) Regularization (L1, L2, Elastic Net)
- c) Dropout & Early Stopping
- d) Transfer Learning

Data Augmentation

- Data augmentation is a strategy used to artificially increase the size and diversity of a training dataset by applying transformations to the existing data.
- **Purpose:** Improve model generalization. Prevent overfitting. Compensate for limited training data.
- *Why?* Increases effective training set size without extra data collection.
- **Common Methods:**
 - Color jittering, cropping, flipping, rotations, scaling.
 - PCA-based color augmentation (as in AlexNet) (Krizhevsky et al., *Commun. ACM*, 2017).
- Transfer learning approach using well-known CNN models (GoogleNet, AlexNet, VGG16, VGG19, DenseNet, etc.) along with data augmentation techniques can be used to accelerate the training and testing process while yielding good results and performance.
- He et al. implemented data augmentation along with regularization techniques such as dropouts and weight decay (CVPR, 2016).



Teerath et al. *IEEE Access* (2024)

Data Augmentation

- **Geometric Transformations:**

Flipping: Horizontal and vertical flips.

Rotation: Rotates images by a specified angle.

Scaling: Resizes images while preserving aspect ratio.

Cropping: Extracts subregions from the image.

- **Color Transformations:**

Brightness Adjustment: Alters image brightness.

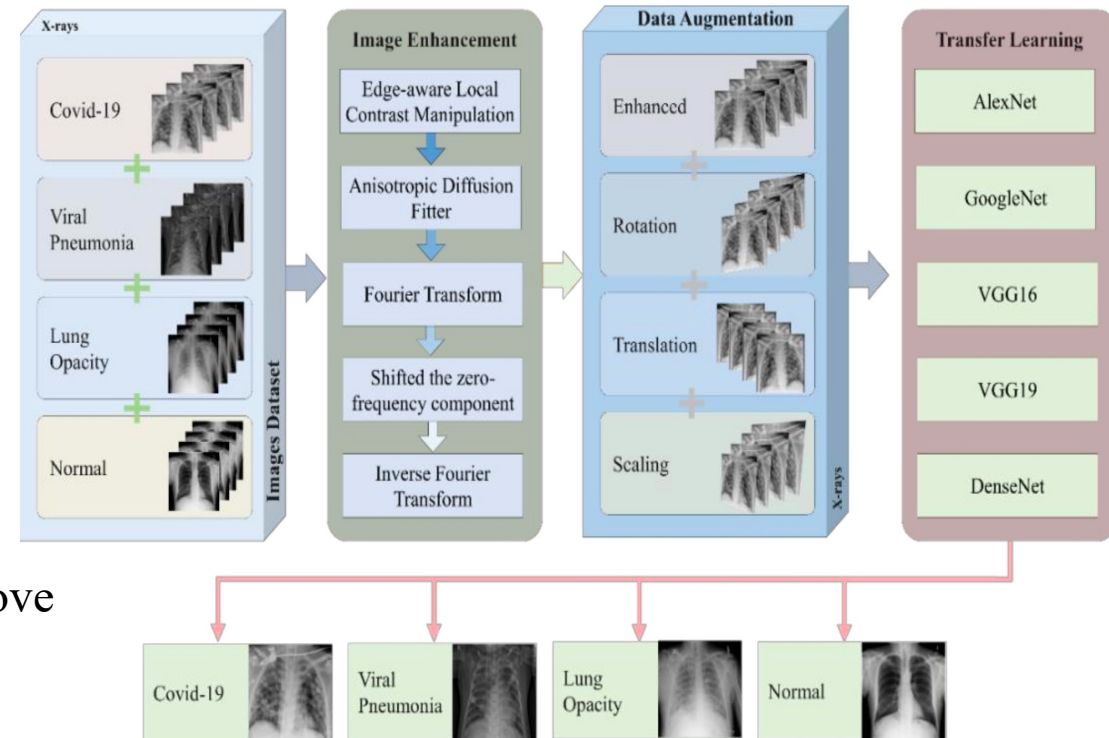
Contrast Adjustment: Modifies contrast levels.

Saturation Adjustment: Changes color saturation.

Hue Adjustment: Shifts color hues.

- **Noise Injection:** Adds random noise to images to improve robustness.

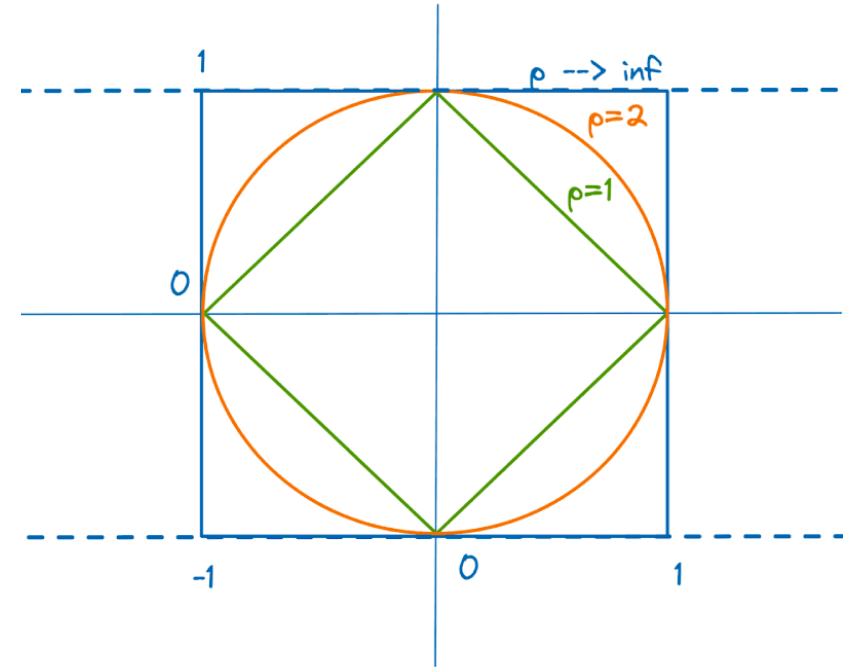
- **Affine Transformations:** Applies scaling, shearing, or translation to the images.



Example of using preprocessing techniques along with the well-known CNN models for COVID-19 and Lungs Pneumonia detection using transfer learning.
Latif et al. *AIMS Mathematics* (2024)

Regularization methods

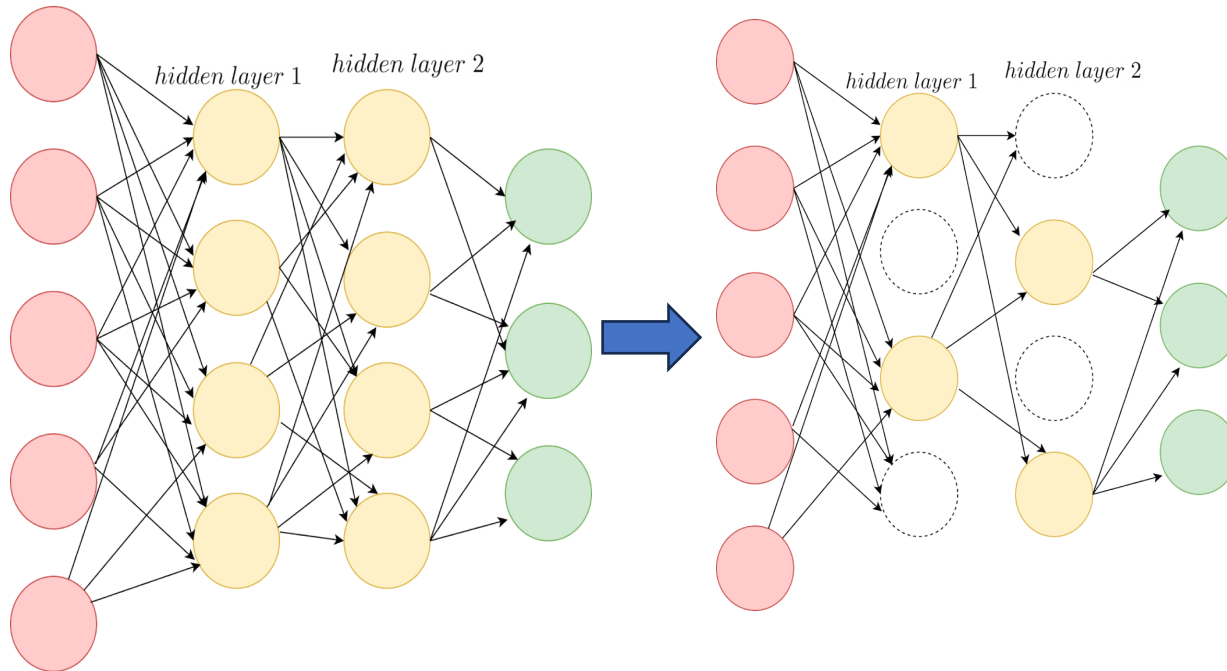
- **Definition:** Regularization refers to techniques that improve a model's generalization by reducing overfitting to the training data.
- **Why Regularization?** Deep learning models are prone to overfitting due to high capacity and complex structures. Regularization helps balance the trade-off between model complexity and performance.
- **L2 Regularization (Weight Decay)**
 - Penalizes the square of weights \rightarrow discourages large weight values, helps smooth solutions.
- **L1 Regularization (Lasso)**
 - Penalizes the absolute value of weights \rightarrow encourages sparsity (some weights become zero).
- **Elastic Net**
 - Combines L1 and L2 \rightarrow can both shrink weights and promote sparsity.



Dropout & Early Stopping

- **Dropout**

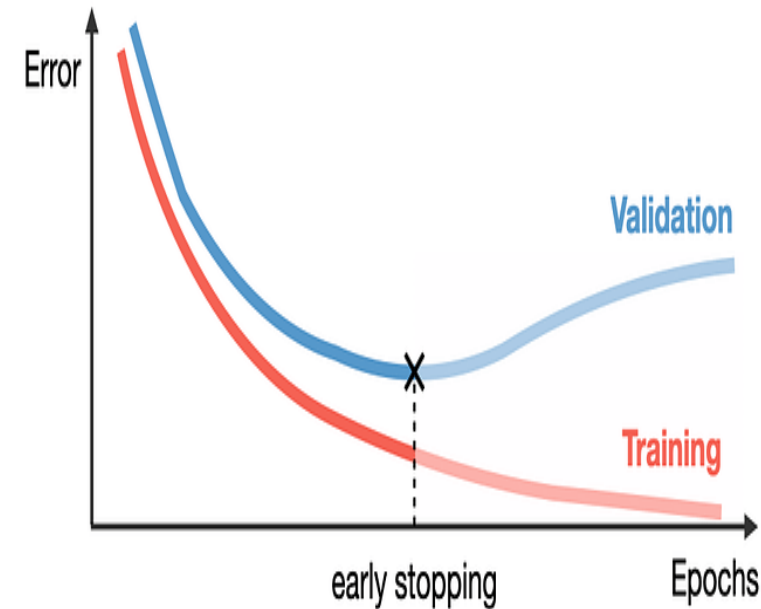
- Randomly “drops” neurons during training.
- Reduces co-adaptations among neurons → mitigates overfitting.



<https://www.pinecone.io/learn/regularization-in-neural-networks/>

- **Early Stopping**

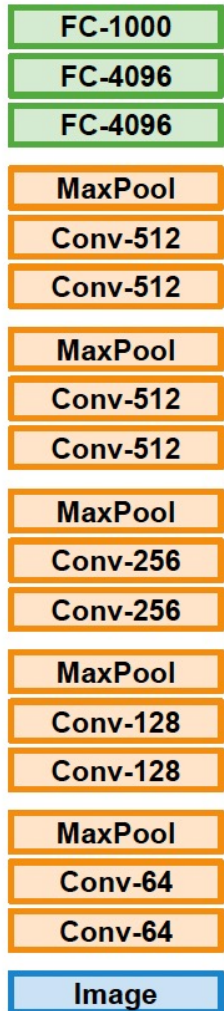
Monitors validation performance and halts training before overfitting sets in. Balances bias/variance by stopping at the optimal point.



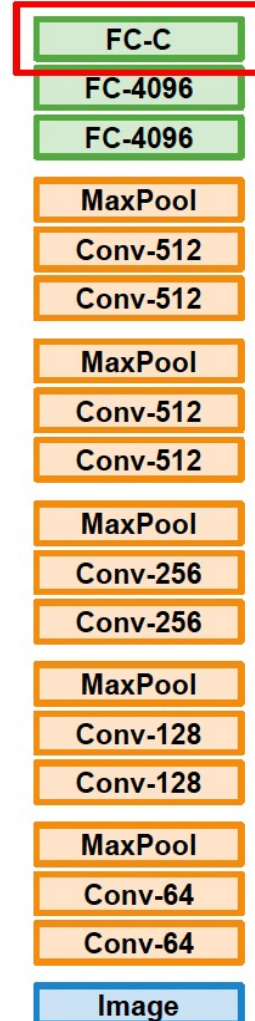
<https://www.comet.com/site/blog/4-techniques-to-tackle-overfitting-in-deep-neural-networks/>

Transfer Learning

Train on ImageNet



Small Datasets



Reinitialize
this and train

Freeze these

Bigger Datasets



Train these

With bigger
dataset, train
more layers

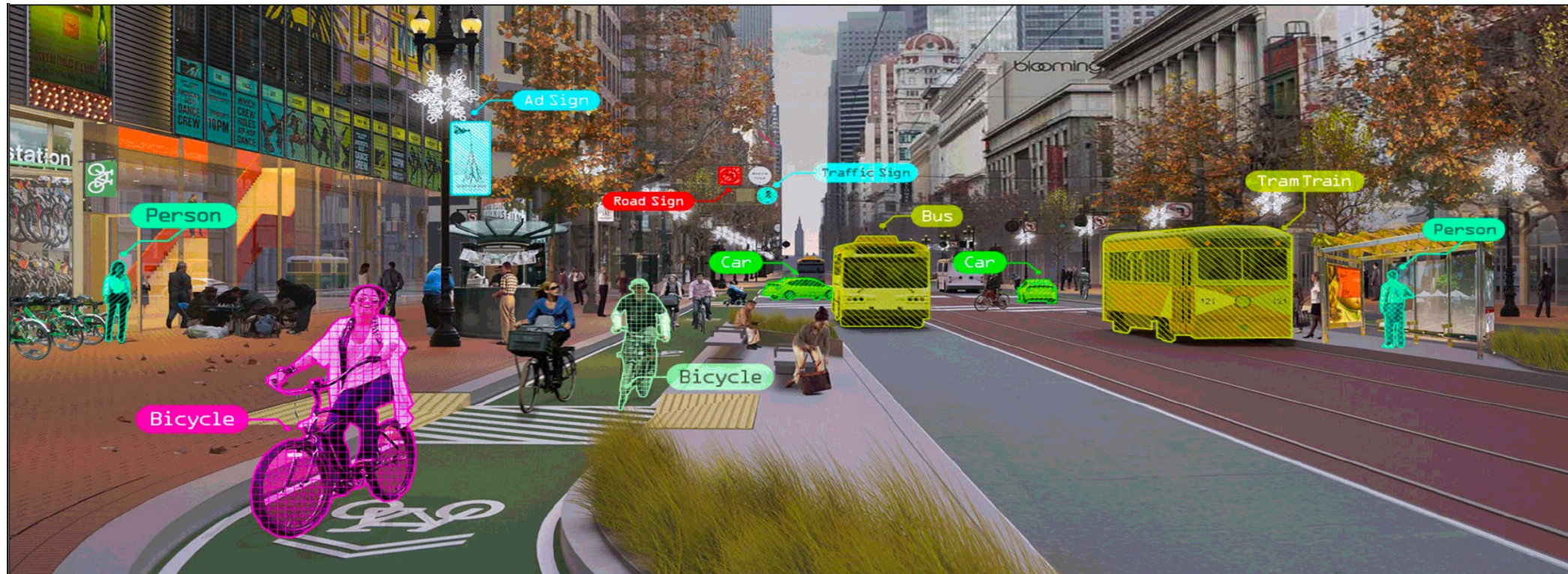
Freeze these

Lower learning rate
when finetuning;
1/10 of original LR
is good starting
point

*Donahue et al, ICML
2014*

*Razavian et al, CVPR
Workshops 2014*

Object Detection



Challenge:

- Objects can be anywhere in the scene, in any orientation, rotation, color hue, etc.
- How can we overcome this challenge?

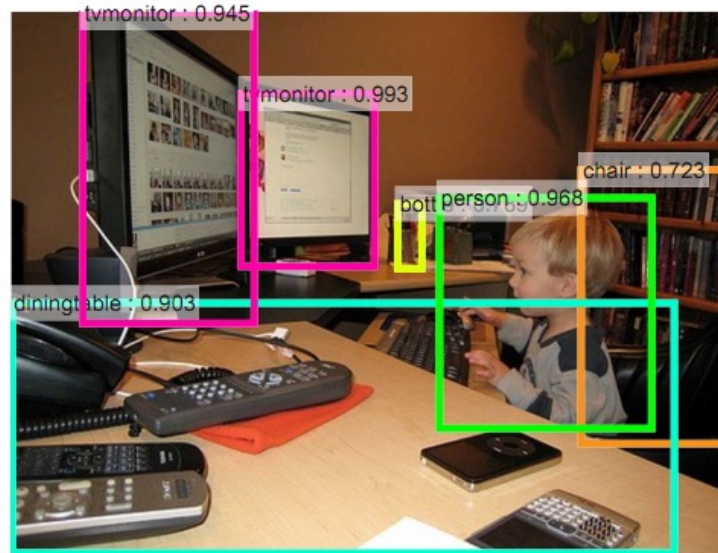
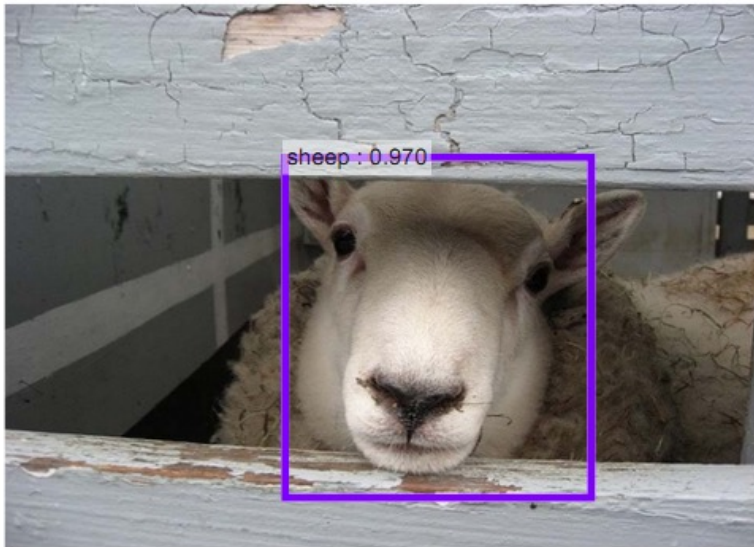
Answer:

- Learn a ton of features (millions) from the bottom up
- Learn the convolutional filters, rather than pre-computing them

What is Object Detection?

To determine: *What objects are where?*

- Object bounding box: location and size
- Object category.



(a)

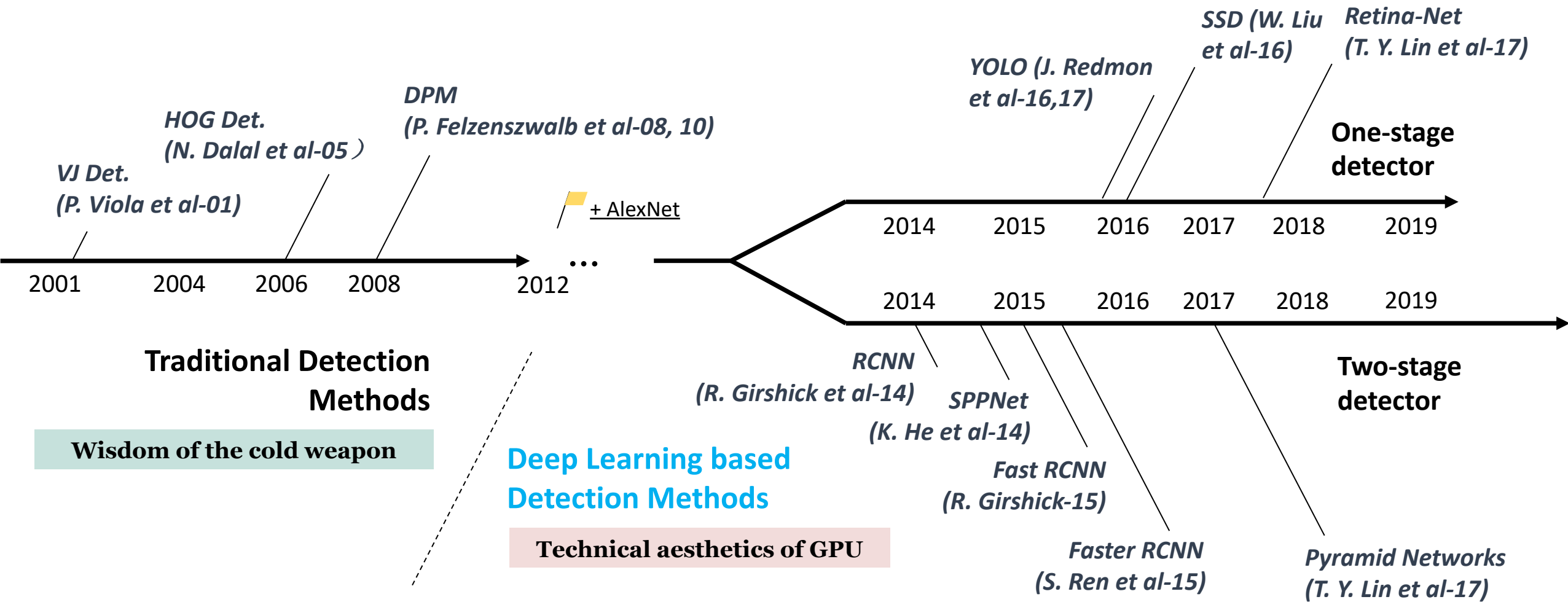
(b)



(c)

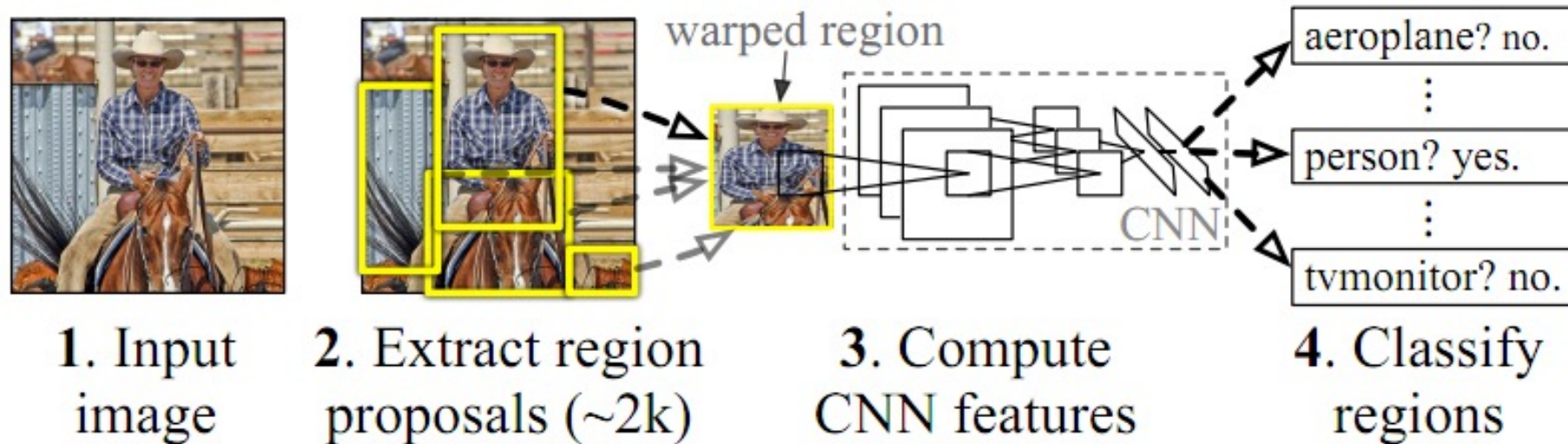
By NIPS15-Faster RCNN

Object Detection Milestones



R-CNN: Regions with CNN features

Ross B. Girshick et al., (CVPR2014)



- Object Proposal+CNN features
- Bounding Box Regression
- Fine tuning
- VOC07 mAP: 33.7→58.5

Time: 14s/image on a GPU

Drawbacks

- The redundant feature computations on a large number of overlapped proposals (>2000 boxes/img) leads to an extremely slow detection speed (14s per image with GPU).

R-CNN: Regions with CNN features

Definition: R-CNN is a deep learning framework for object detection introduced by Ross Girshick in 2014. It integrates region proposals with CNNs to detect objects in an image effectively.

Key Contributions:

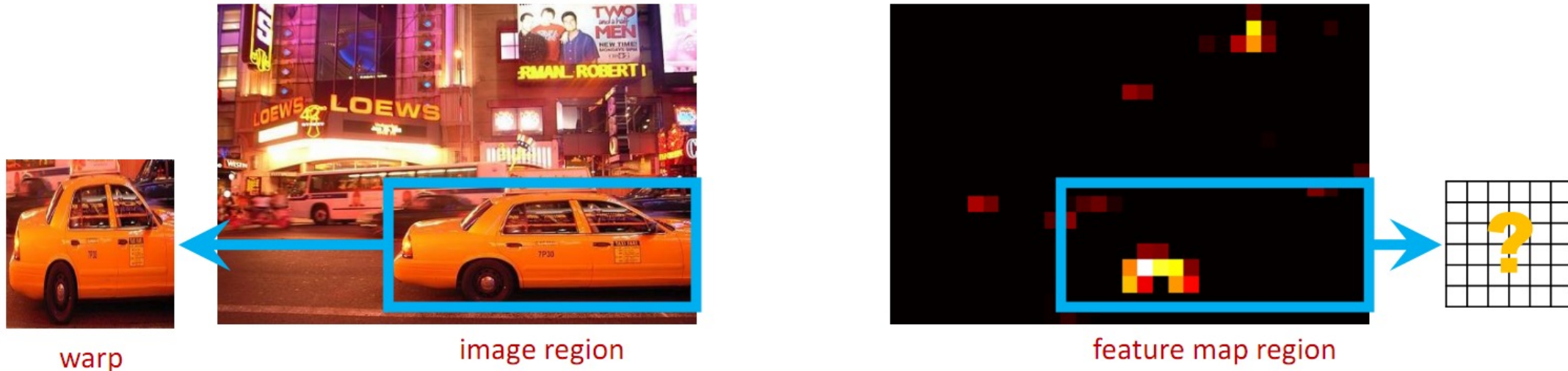
- ❖ Combines region proposals with CNN-based feature extraction.
- ❖ Demonstrates the use of transfer learning for detection tasks.
- ❖ Achieves significant performance improvements over traditional methods.

Workflow of R-CNN:

- Input image is processed using Selective Search to generate region proposals.
- Each region is resized to 224x224 and passed through a CNN to extract features.
- SVM classifiers predict object categories for the proposals.
- Bounding box regression refines the coordinates of the proposals.
- Outputs are the predicted class labels and refined bounding boxes.

SPPNet: Spatial Pyramid Pooling

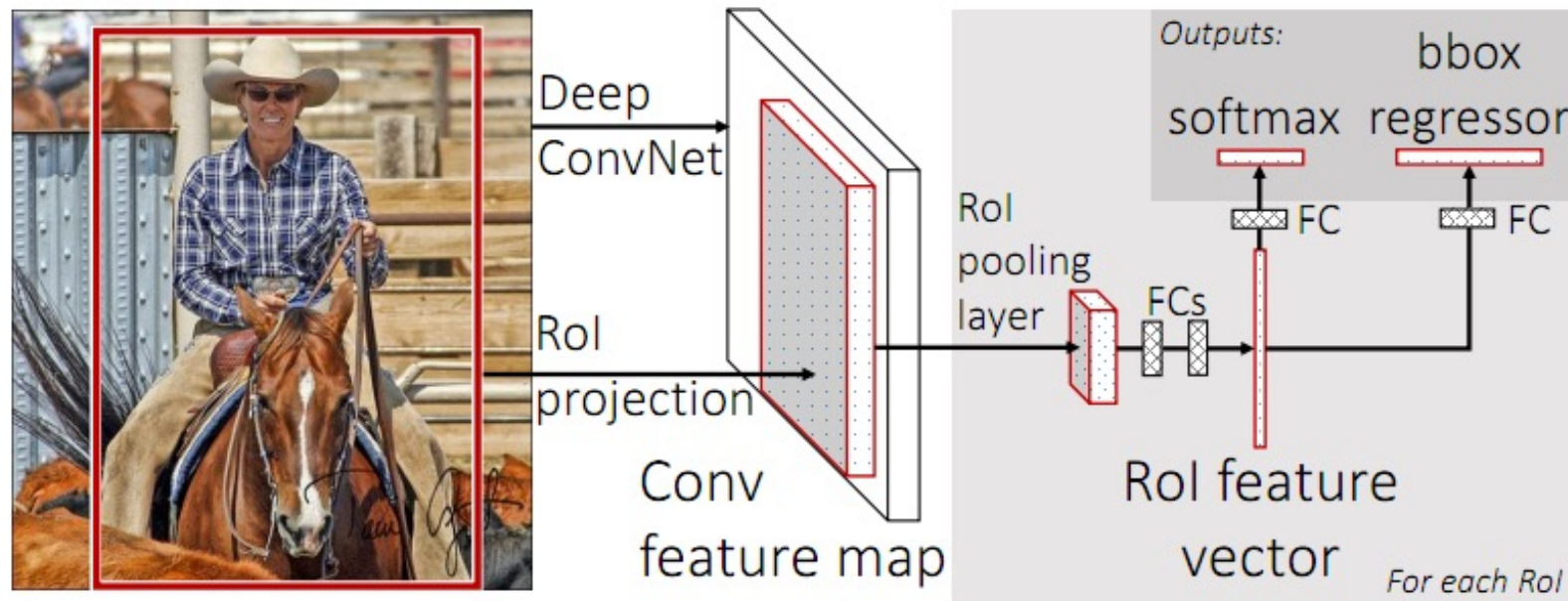
SPPnet is a deep learning framework designed to handle images of arbitrary sizes without requiring cropping or resizing. It introduces the Spatial Pyramid Pooling (SPP) layer, which allows for flexible input dimensions and improved computational efficiency.



- **Fixed-length** features are required by fully-connected layers or SVM
- But how to produce a fixed-length feature from a feature map region?
- Solutions in traditional computer vision: Bag-of-words, SPM...

Kaiming He et al., (ECCV2014)

Fast RCNN



Fast R-CNN is an object detection framework introduced by Ross Girshick in 2015. It improves upon the inefficiencies of R-CNN by introducing Region of Interest (ROI) Pooling and enabling shared computation, leading to faster and more accurate object detection.

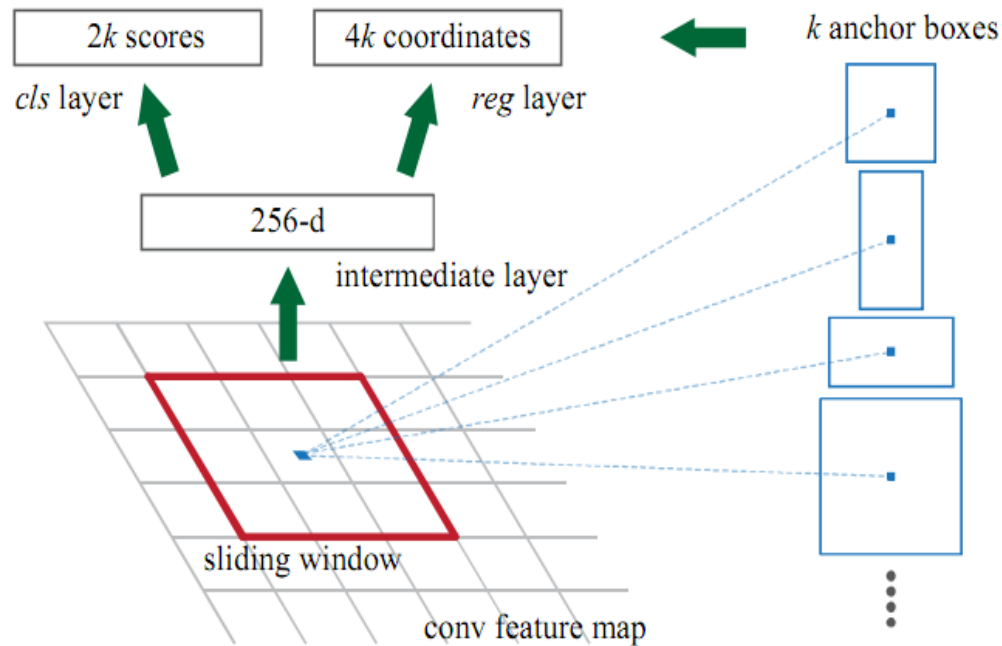
- ROI Pooling
- Multi-task loss (Clc. + BB Reg.)
- BP through ROI pooling layers
- VOC07 mAP: 58.5→70.0

Time:
0.32s/image on a GPU

Ross B. Girshick (ICCV15)

Faster RCNN

Faster R-CNN is a successor to Fast R-CNN and introduces the **Region Proposal Network (RPN)** for generating region proposals, making the detection pipeline fully end-to-end.



Anchors (reference boxes)

- Region Proposal Network
- Detection Network
- Sharing Features
- VOC07 mAP: 70.0→78.8

Time: 17 fps on a GPU

Shaoqing Ren et al., (NIPS2015)

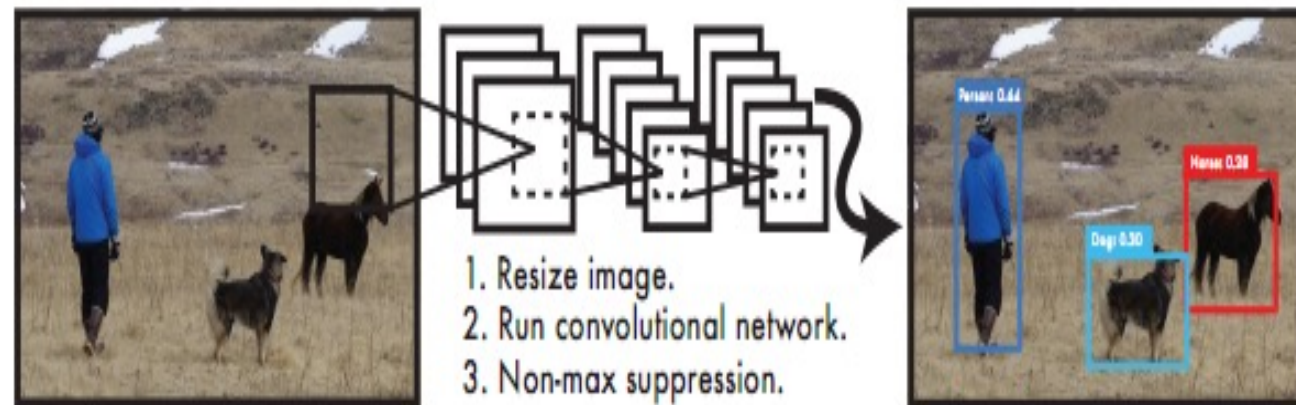
$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*).$$

You Only Look Once (YOLO)

YOLO treats object detection as a single regression problem, predicting both class probabilities and bounding box coordinates in one forward pass.
J. Redmon et al., (CVPR2016)

Key Contributions:

- ❖ Introduces a unified framework for object detection, enabling real-time performance.
- ❖ Processes the entire image in a single forward pass, improving efficiency.
- ❖ Balances speed and accuracy, making it suitable for real-world applications.



- Runs at 45fps with VOC07 mAP=63.4% and VOC12 mAP=57.9%.
- A fast version runs at 155fps with VOC07 mAP=52.7%.

Workflow of YOLO

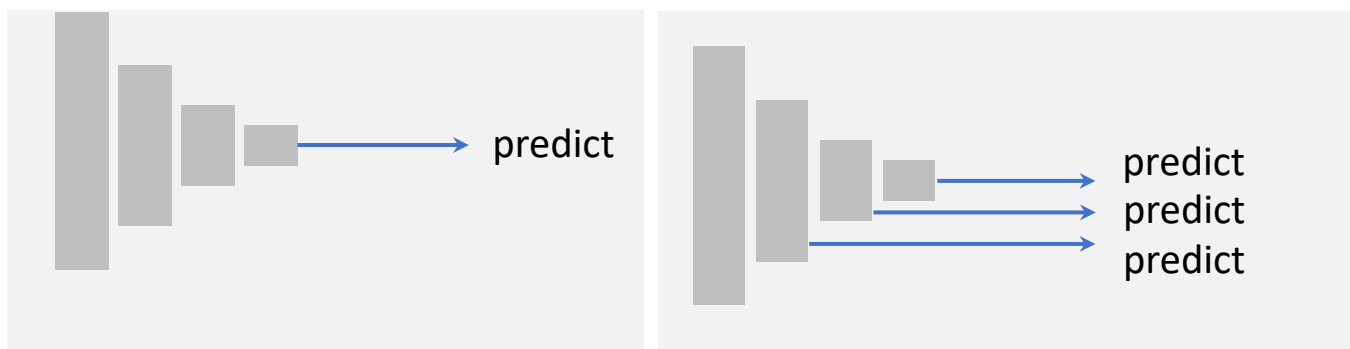
- **Input Image:** The input image is divided into an SXS grid (e.g., 7X7).
- **Feature Extraction:** A CNN processes the image to extract features.
- **Bounding Box Prediction:** Each grid cell predicts:
 - Bounding boxes (coordinates and dimensions).
 - Confidence scores for each bounding box.
- **Classification:** Each grid cell predicts class probabilities for the objects it contains.
- **Post-Processing:** Non-Maximum Suppression (NMS) removes duplicate detections and retains the most confident predictions.

SSD: Single Shot MultiBox Detector

SSD performs object detection in a single forward pass, making it fast and efficient compared to region-based methods like Faster R-CNN.

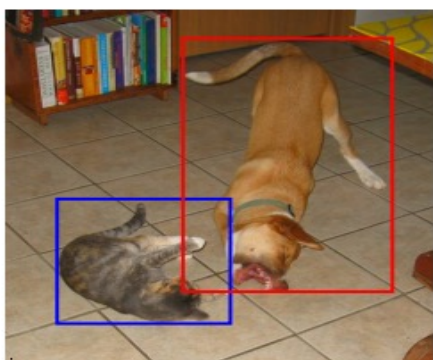
Key Contributions:

- Uses multi-scale feature maps for detecting objects of different sizes.
- Introduces default (prior) boxes for efficient bounding box predictions.
- Eliminates the need for separate region proposal steps, improving speed.

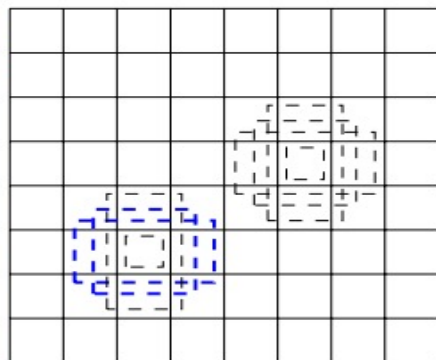


(a) YOLO

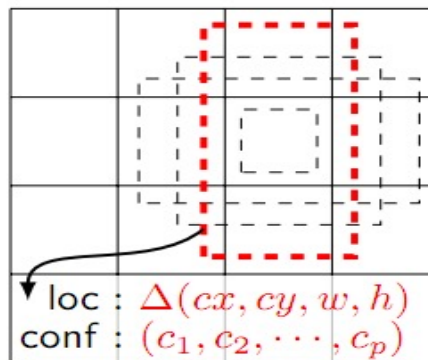
(b) SSD



(a) Image with GT boxes



(b) 8×8 feature map



(c) 4×4 feature map

Workflow of SSD

1. Input Image: The input image is processed through a backbone CNN (e.g., VGG16) to extract feature maps.

2. Multi-Scale Feature Maps: Feature maps from different layers are used to predict objects at various scales.

3. Default Boxes: Predefined bounding boxes with varying aspect ratios and scales are applied to each feature map cell.

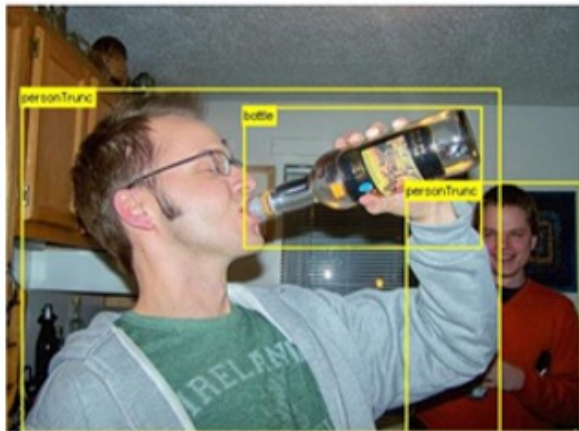
4. Predictions: Each default box predicts:

- Class probabilities for classification.
- Bounding box offsets for localization.

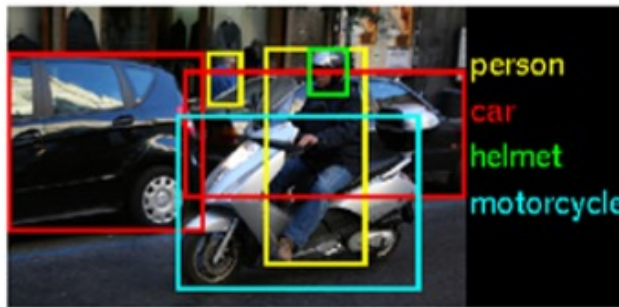
5. Post-Processing: Non-Maximum Suppression (NMS) removes redundant detections and retains the most confident predictions.

Wei Liu et al., (ECCV2016)

Detection Datasets



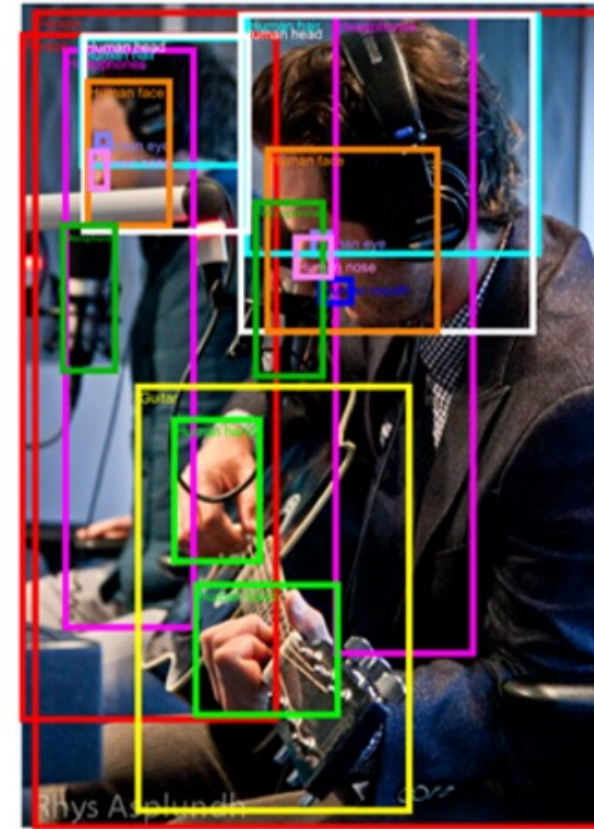
PASCAL VOC



ILSVRC



MS-COCO



Open Images

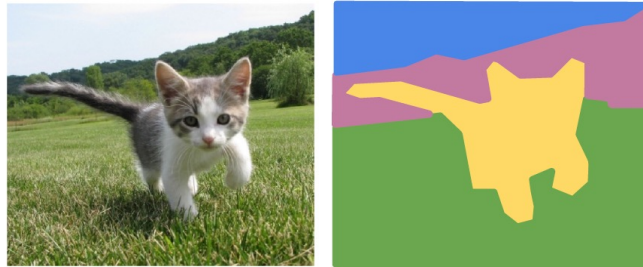
Detection Datasets

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2018	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
OID-2018	1,743,042	14,610,229	41,620	204,621	1,784,662	14,814,850	125,436	625,282

TABLE 1

Some well-known object detection datasets and their statistics.

Semantic Segmentation: The Problem



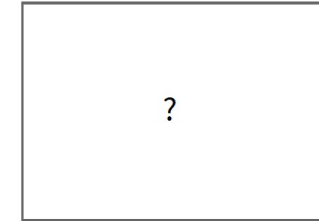
GRASS, CAT, TREE,
SKY, ...

Paired training data: for each training image, each pixel is labeled with a semantic category.

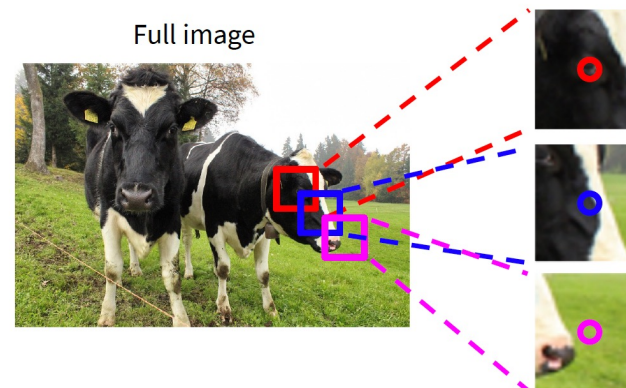


Impossible to classify without context

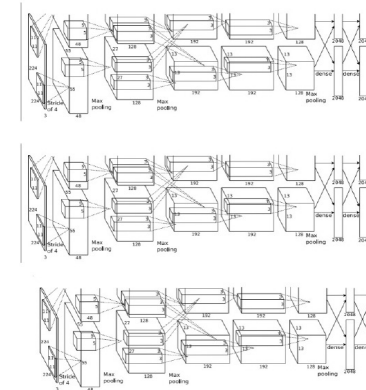
Q: how do we include context?



At test time, classify each pixel of a new image.



Classify center pixel with CNN



Cow

Cow

Grass

Q: how do we model this?

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

U-Net: Motivation

In CNNs, different layers learn different feature levels:

- **Lower layers:** Learn low-level, fine-grained details (e.g., edges, textures)
- **Higher layers:** Capture high-level, coarse-grained semantic features (e.g., shape, structure)
- This hierarchy is ideal for classification tasks but introduces limitations for pixel-level tasks like segmentation

Challenges in Medical Image Segmentation

- Medical images often suffer from:
 - Noise
 - Low contrast
 - Blurred or unclear boundaries
- Relying only on low-level features results in poor object recognition
- Relying only on high-level semantic features leads to inaccurate boundary detection

Need for Multi-Level Feature Integration

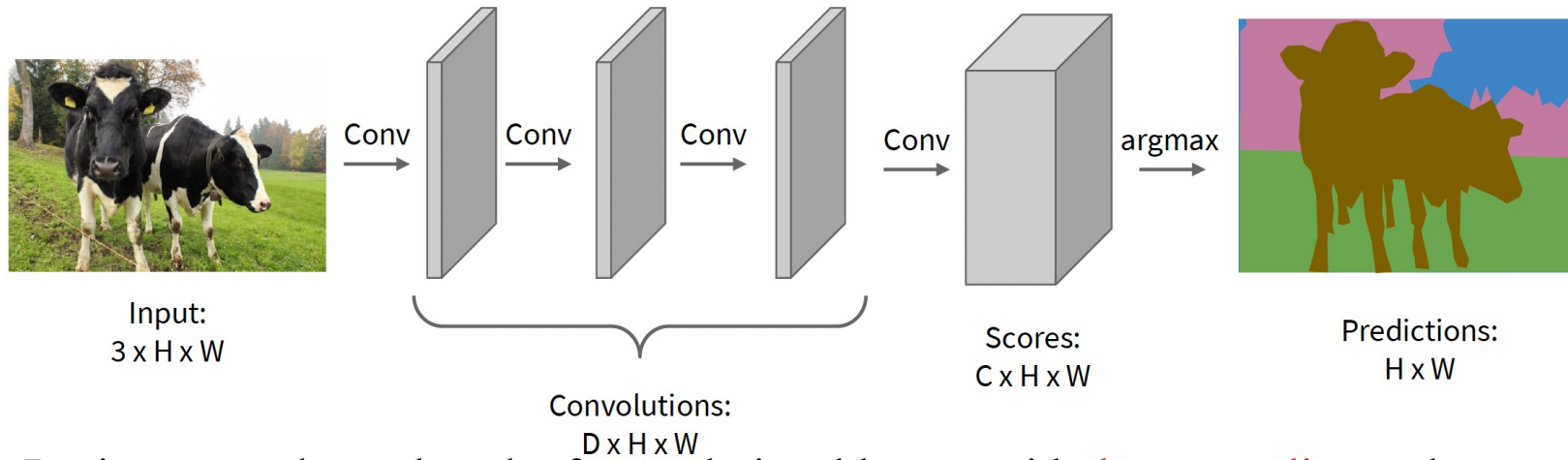
- Effective segmentation requires a combination of:
 - High-level semantic understanding (context)
 - Low-level spatial precision (details)
- General CNNs lack explicit mechanisms to combine both effectively

Encoder-Decoder Architectures

- Designed to combine high-level and low-level features
- Consist of:
 - **Encoder:** Downsamples and extracts abstract features
 - **Decoder:** Upsamples to recover spatial resolution and integrates detail
- Enables pixel-level prediction with semantic awareness

Semantic Segmentation Idea

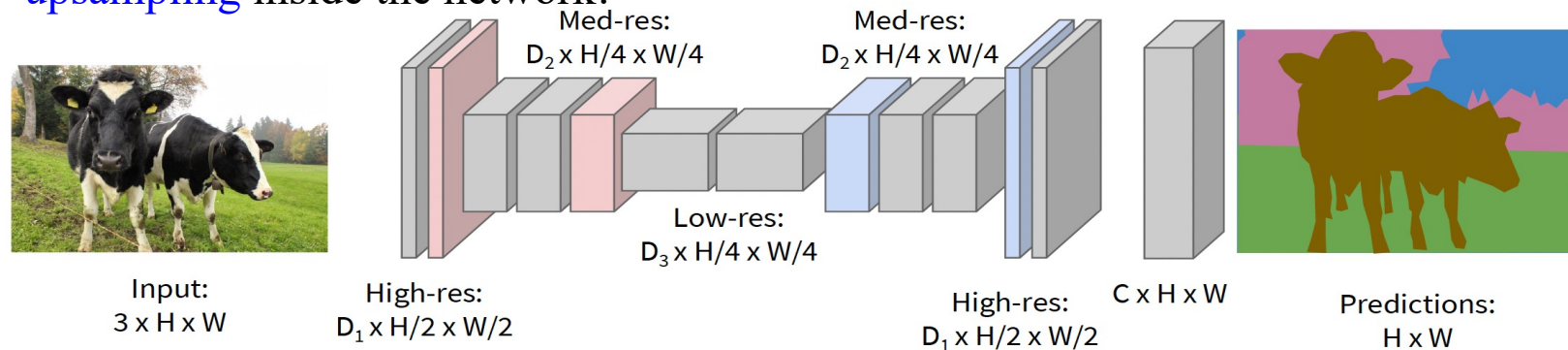
Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



Problem: convolutions at original image resolution will be very expensive ...

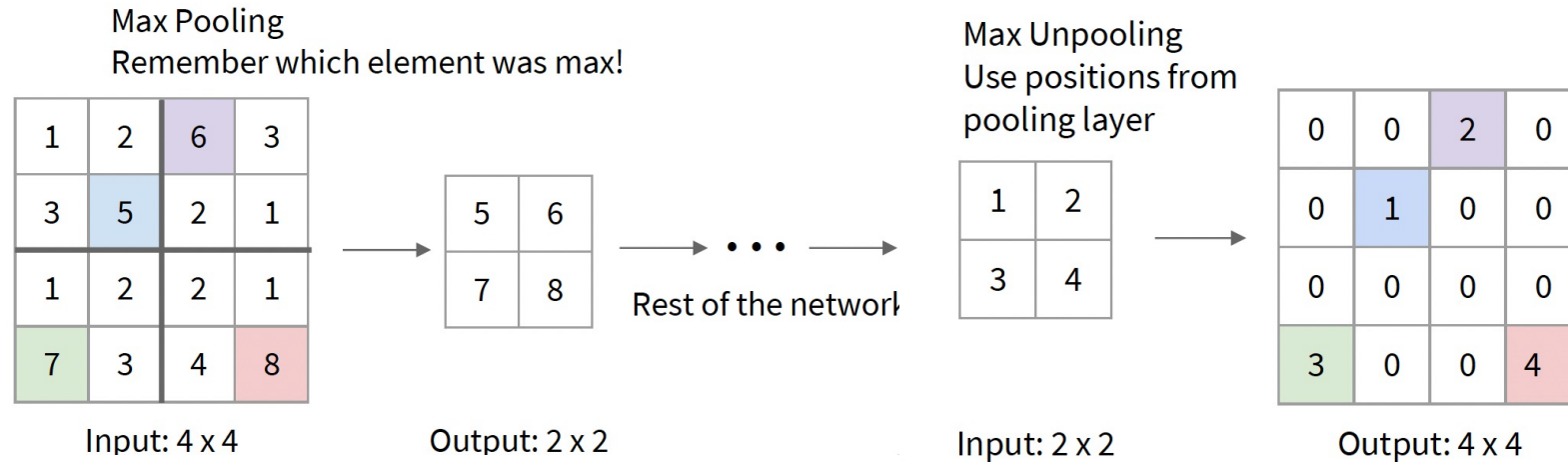
Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Downsampling:
Pooling, strided
convolution

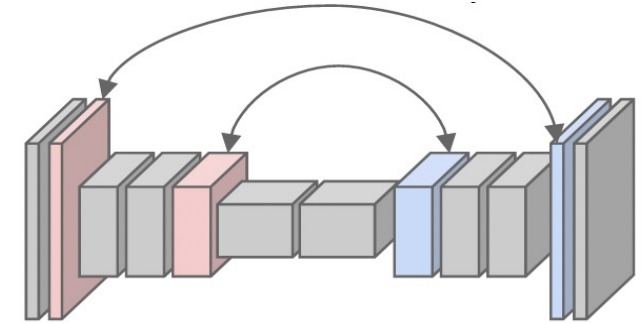


Upsampling:
Unpooling or strided
transposed convolution

Downsampling and Upsampling



Corresponding pairs of
downsampling and
upsampling layers



Common Downsampling types:

- **Max pooling:** Takes the maximum value in each window
- **Average pooling:** Computes the average value
- **Stochastic pooling:** Randomly selects an activation based on a probability distribution
- **LP-pooling:** Generalized pooling that uses the p-norm over each region
- **Global pooling:** Applies pooling over the entire feature map to reduce to a single value per channel

• **Purpose:** (i) Reduce computation; (ii) Increase receptive field; (iii) Achieve spatial invariance; (iv) Introduce regularization

Common unpooling strategies:

- **Max-unpooling with indices:**
- **Fixed-position unpooling:** inserts values at top-left corner of window
- **Interpolation-based unpooling:** uses nearest-neighbor or bilinear interpolation to expand feature maps
- **Learnable unpooling:** introduces parameters to learn where and how to upsample

Often followed by convolutional layers to refine outputs

U-Net: Vanilla Version

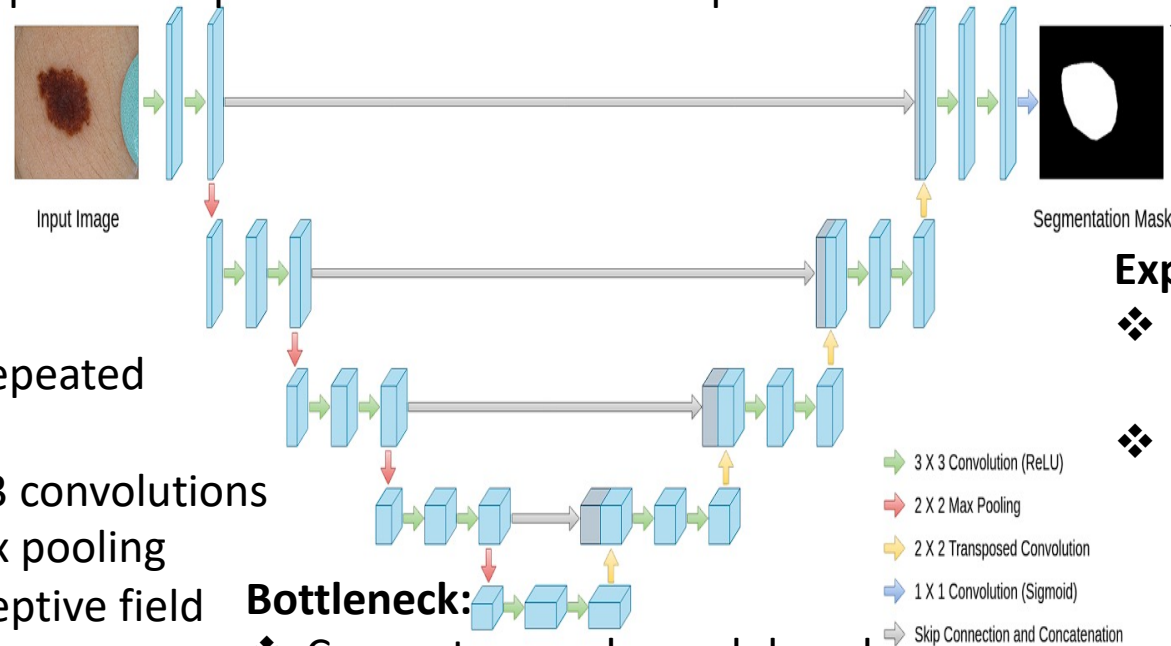
❖ U-Net is a neat end-to-end neural network with a characteristic "U" shape

Contracting Path (Encoder):

- ❖ Captures context through repeated downsampling blocks
- ❖ Each block includes two 3×3 convolutions + ReLU, followed by 2×2 max pooling
- ❖ Gradually increases the receptive field without heavy computation

Skip Connections:

- Link encoder and decoder layers at the same depth level
- Concatenate encoder feature maps with decoder inputs to combine detailed and contextual information
- Help restore spatial resolution and sharpen boundaries



Bottleneck:

- ❖ Connects encoder and decoder
- ❖ Two 3×3 convolutions + ReLU
- ❖ Reduces spatial resolution and increases depth for high-level abstraction

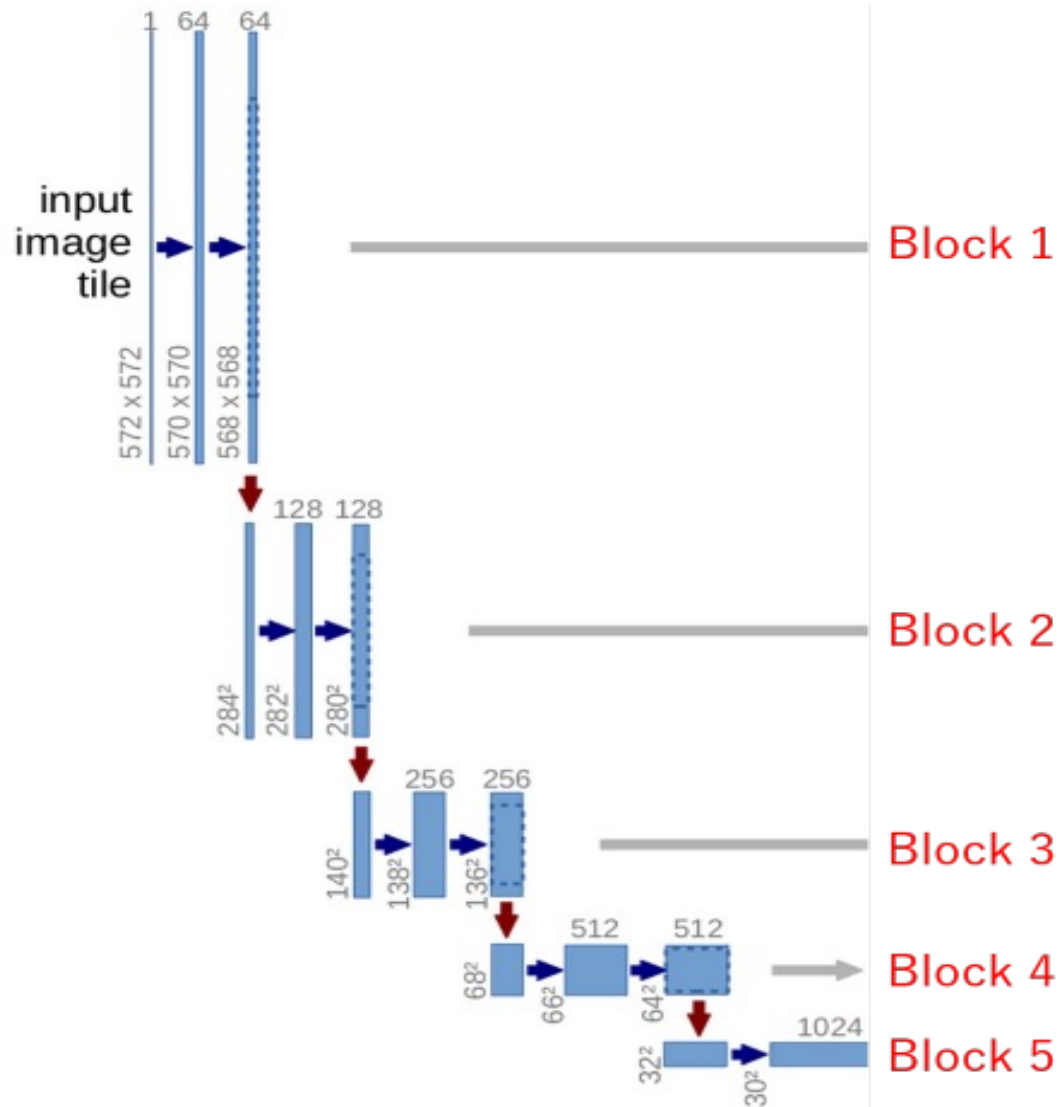
Final Output:

- ❖ A 1×1 convolution maps the final feature map to the number of target classes
- ❖ Produces a pixel-level classification map (e.g., segmentation mask)

Expanding Path (Decoder):

- ❖ Upsamples feature maps to match input resolution
- ❖ Each block includes one 2×2 transposed convolution (up-conv), two 3×3 convolutions + ReLU

Contracting Path (Encoder)



❖ Block 1:

- ❖ Input: 572×572×1 (grayscale image)
- ❖ Two 3×3 unpadded convolutions + ReLU → 64 channels
- ❖ 2×2 max pooling (stride 2) → downsampled to 284×284

❖ Block 2:

- ❖ Two 3×3 convolutions + ReLU → 128 channels
- ❖ 2×2 max pooling → 140×140

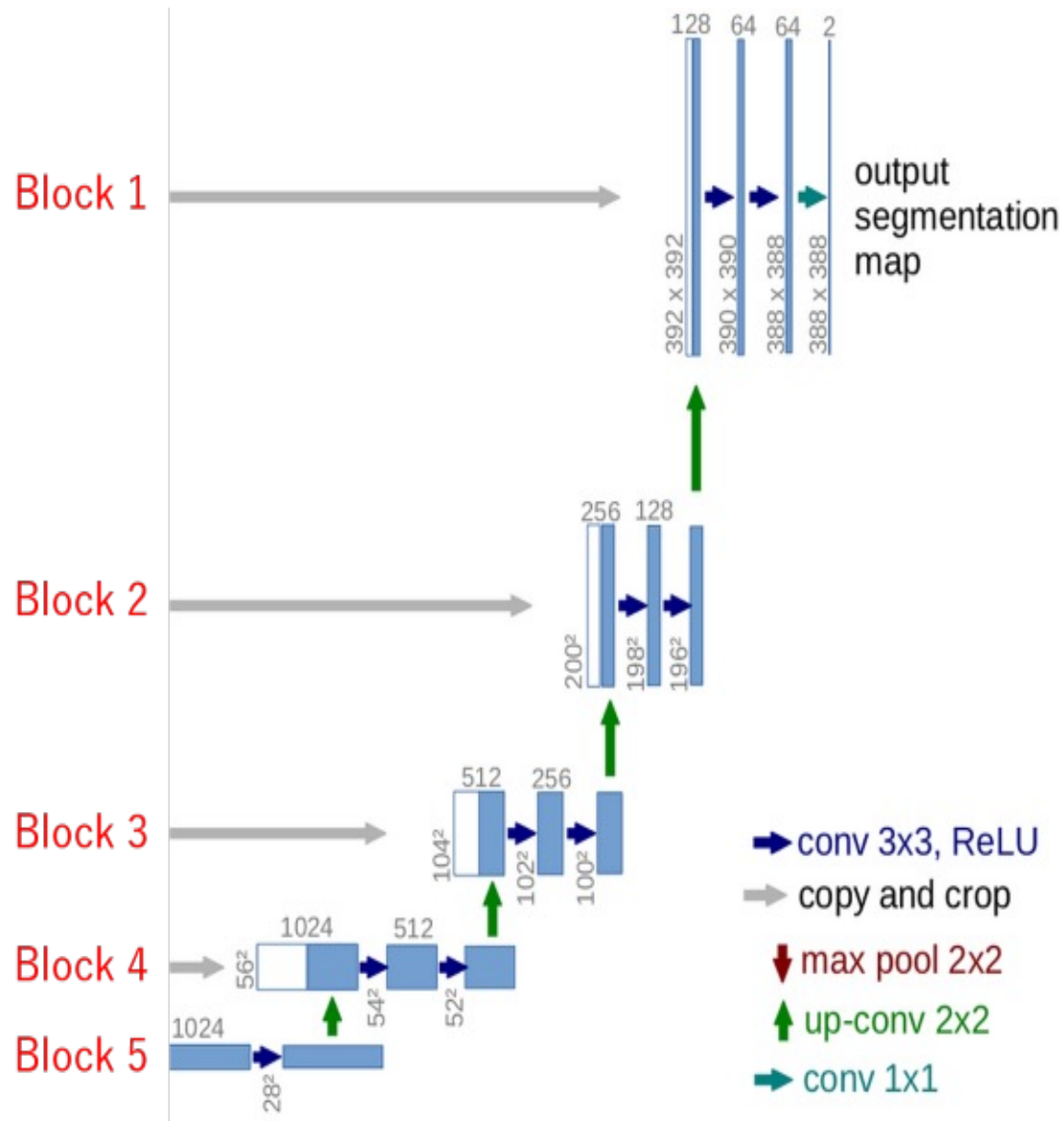
❖ Block 3 & Block 4:

- ❖ Same as previous blocks with doubled channels (256, 512)
- ❖ Max pooling after each block halves spatial dimensions

❖ Block 5 (Bottom):

- ❖ Two 3×3 convolutions + ReLU → 1024 channels
- ❖ First conv in this block included here, second used in expanding path for symmetry

Expanding Path (Decoder)



•Block 5:

- Continues from the bottom block with a second 3×3 convolution + ReLU
- Followed by a 2×2 up-convolution \rightarrow doubles spatial resolution, reduces channels to 512

•Block 4:

- Skip connection: concatenate encoder feature map (cropped to match size) \rightarrow 1024 channels
- Two 3×3 convolutions + ReLU \rightarrow reduce to 512 channels
- 2×2 up-convolution \rightarrow upsample and reduce channels to 256

•Block 3 & Block 2:

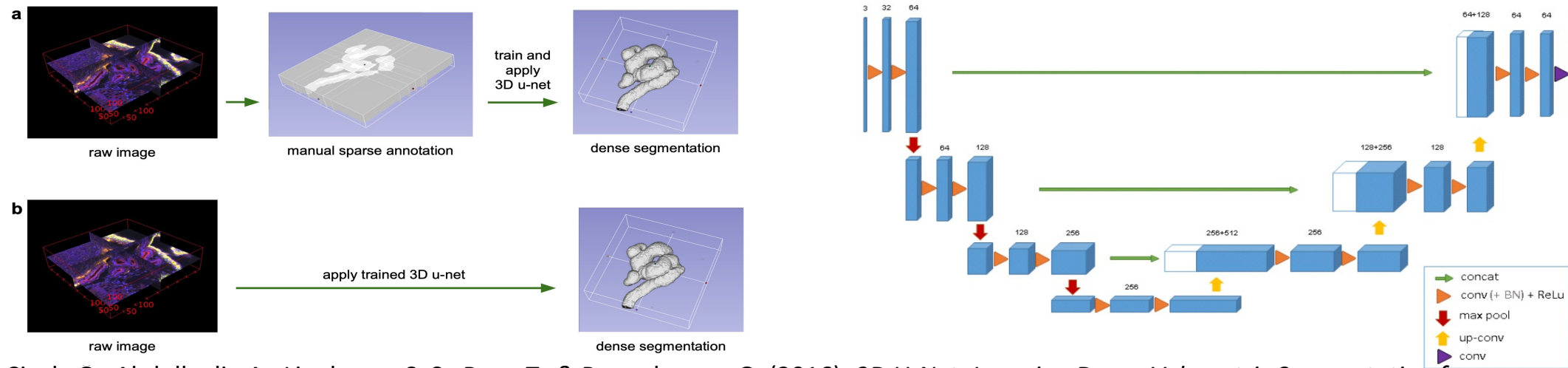
- Same as Block 4, with halved channels: $256 \rightarrow 128 \rightarrow 64$

•Block 1 (Final Block):

- After skip connection: 128 channels
- Two 3×3 convolutions + ReLU \rightarrow reduce to 64 channels
- Final 1×1 convolution \rightarrow maps to number of classes (e.g., 2 for binary)
- Followed by activation function (e.g., sigmoid for binary classification)

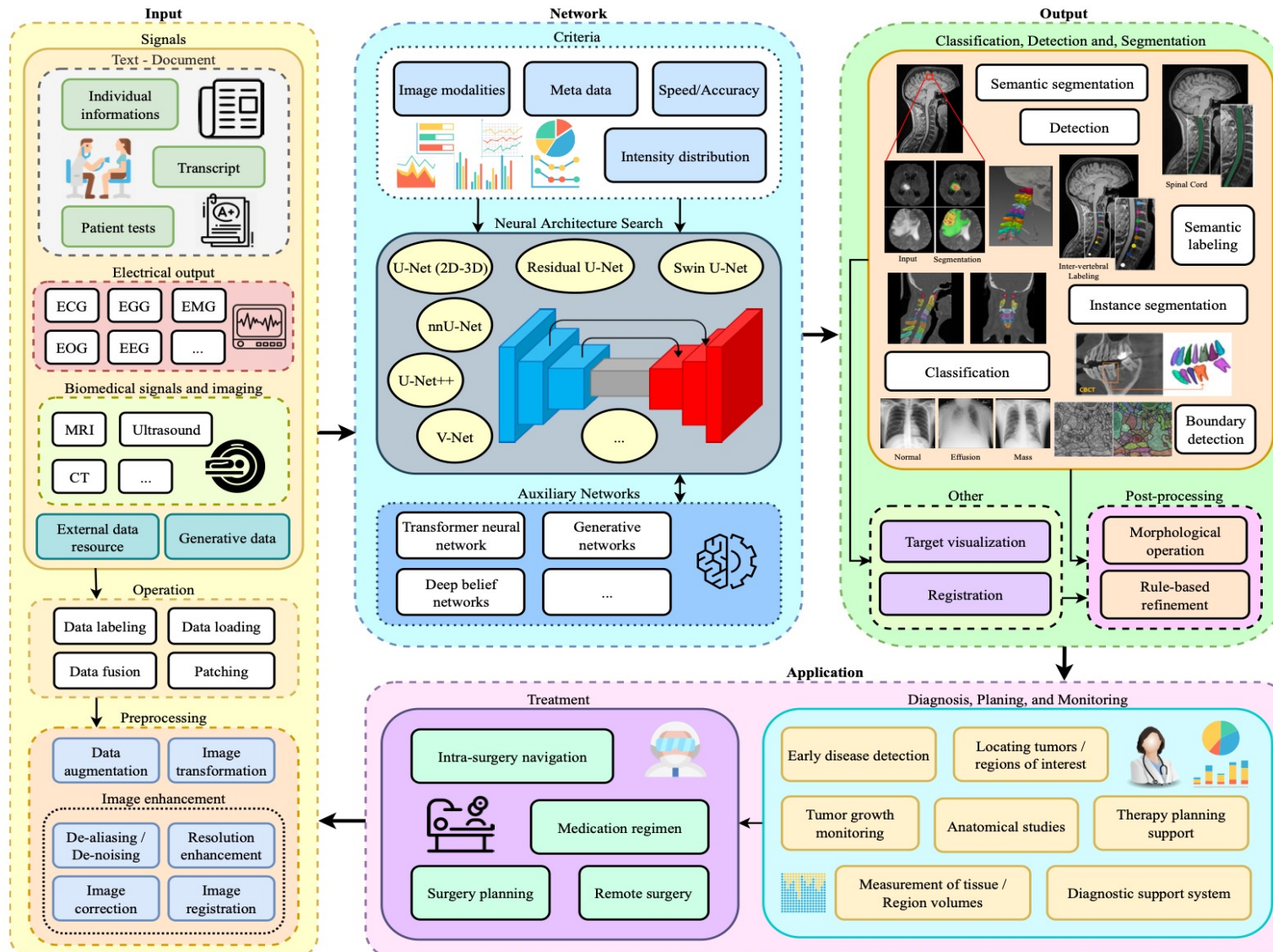
3D U-Net

- Due to the abundance and representation power of volumetric data, most medical image modalities are three-dimensional. 3D U-Net was commonly used in Brain tumor segmentation (e.g., BraTS dataset), Lung nodule detection, and liver and pancreas segmentation.
- 3D U-Net is proposed to deal with 3D medical data directly. It replaces all 2D operations with their 3D counterparts. The users can annotate some slices in the volume to be segmented. The model then learns from these sparse annotations and provides a dense 3D segmentation.
- However, due to the limitation of computational resources, it only includes three down-sampling, which cannot effectively extract deep-layer image features, leading to limited segmentation accuracy for medical images.



Çiçek, O., Abdulkadir, A., Lienkamp, S. S., Brox, T., & Ronneberger, O. (2016). *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation* (No. arXiv:1606.06650). arXiv. <https://doi.org/10.48550/arXiv.1606.06650>

U-Net in Clinical Image Analysis Pipelines



U-Net plays a central role in clinical image analysis pipelines

Overview of key stages:

- **Input Preparation:** Image acquisition, normalization, and preprocessing for consistent input format
- **Architecture Search:** Automatic selection of the most efficient U-Net variant via neural architecture search
- **Postprocessing:** Refinement of segmentation masks (e.g., morphological operations)
- **Clinical Application:** Supports decisions such as tumor growth tracking or treatment planning

Azad, R., Aghdam, E. K., Rauland, A., Jia, Y., Avval, A. H., Bozorgpour, A., Karimijafarbigloo, S., Cohen, J. P., Adeli, E., & Merhof, D. (2022). *Medical Image Segmentation Review: The success of U-Net* (No. arXiv:2211.14830). arXiv. <http://arxiv.org/abs/2211.14830>

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

5 Optimization Techniques

6 Convolutional Neural Networks (CNN)

7 Graph Neural Networks (GNNs/GCNs)

8 Theoretical Properties

Graph-Structured Data

Graph-structured data is a type of data representation where **entities (nodes)** and their **relationships (edges)** are explicitly modeled as a graph. This structure captures the connections between data points, allowing for more effective analysis of relational patterns.

Examples:

- Social networks, citation networks, multi-agent systems
- Knowledge graphs
- Recommendation System
- Protein interaction networks
- Molecules
- Road maps
- Brain networks

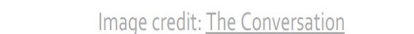
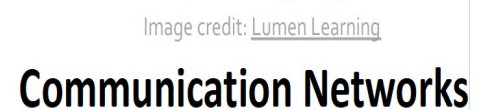
Graphs are a general language for describing and analyzing entities with relations/interactions

Why Are Graph-structured Data Important?

Graphs capture **complex relationships and dependencies** between entities:

- **Interconnected entities influence each other** (e.g., in social networks, a person's behavior depends on their connections).
- **Knowledge is structured in relational forms** (e.g., in knowledge graphs, concepts are linked based on meaning and context).
- **Biological and medical data exhibit intricate interactions** (e.g., protein-protein interaction networks, brain connectivity graphs).

By modeling data as graphs, we can better understand structures, uncover hidden patterns, and improve AI-driven decision-making.



Graph-Structured Data is Everywhere

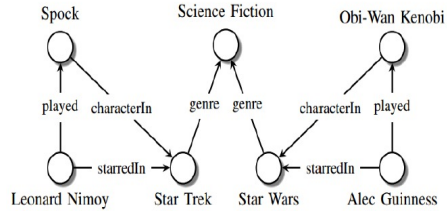


Image credit: Maximilian Nickel et al

Knowledge Graphs

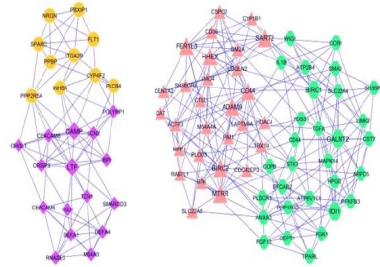


Image credit: ese.wustl.edu

Regulatory Networks

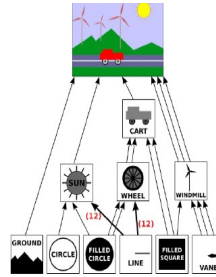


Image credit: math.hws.edu

Scene Graphs

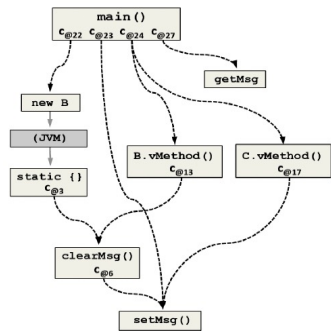
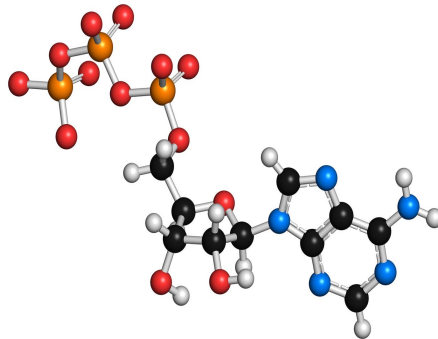


Image credit: ResearchGate

Code Graphs



Molecules

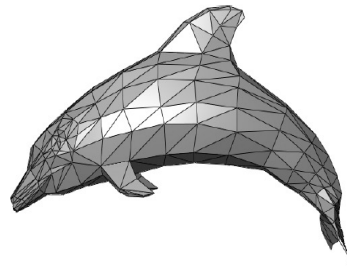
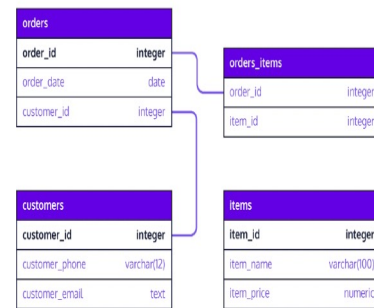
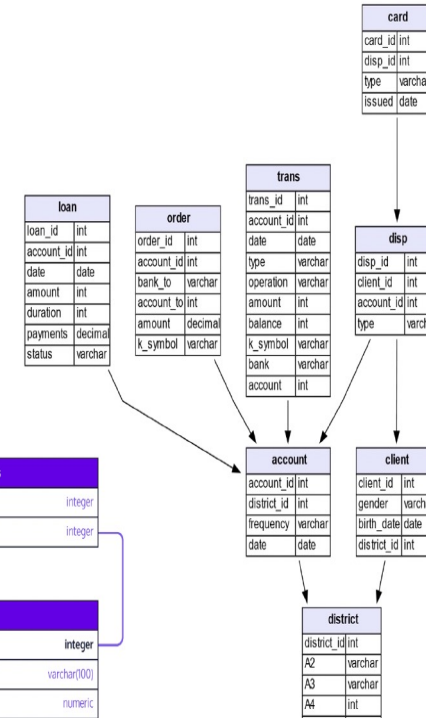


Image credit: Wikipedia

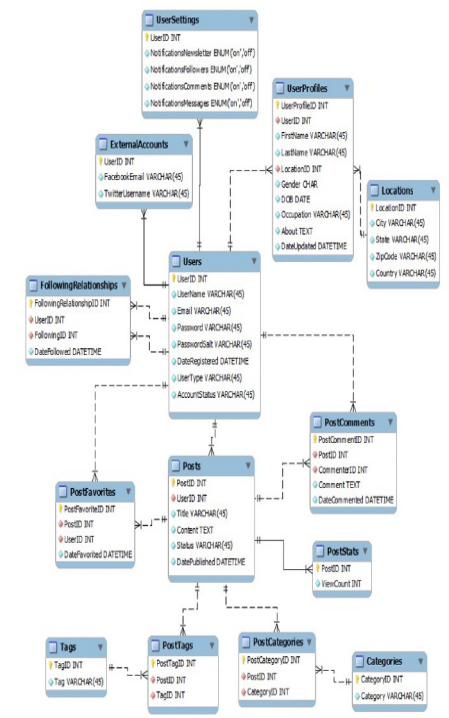
3D Shapes



Commerce



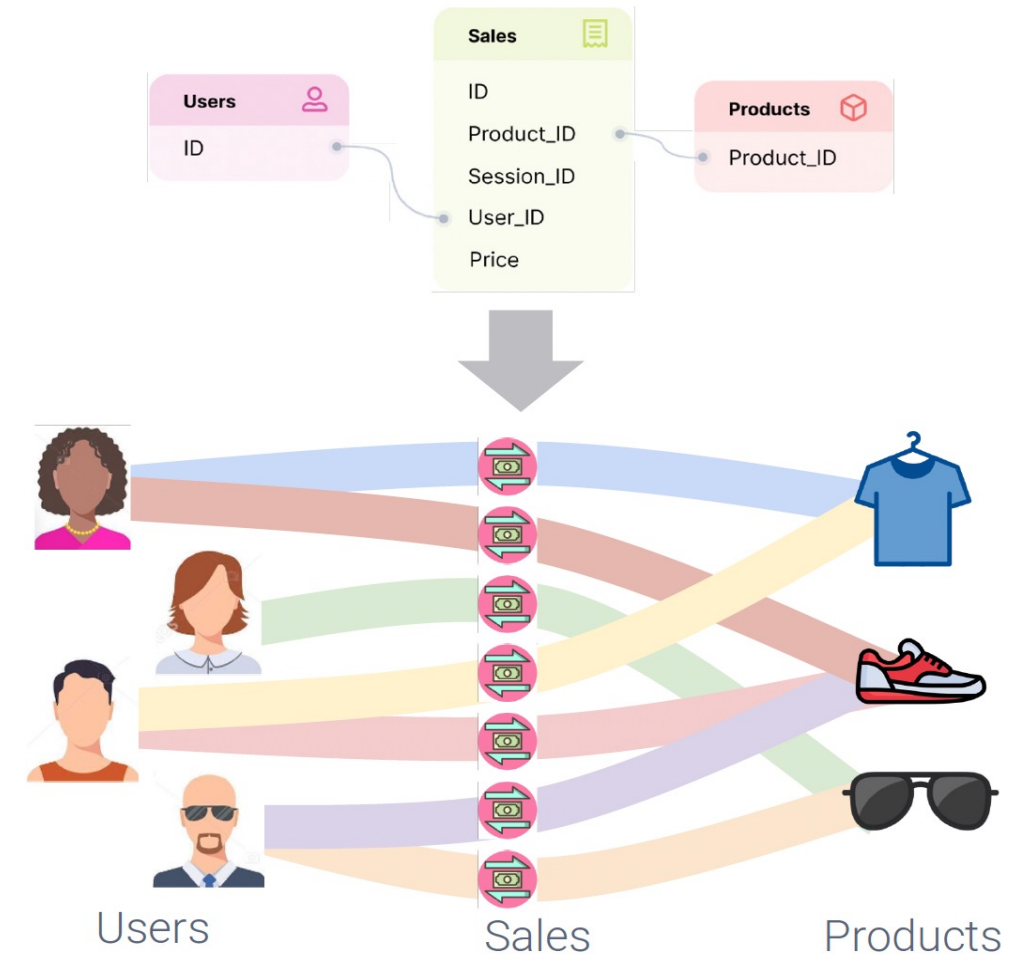
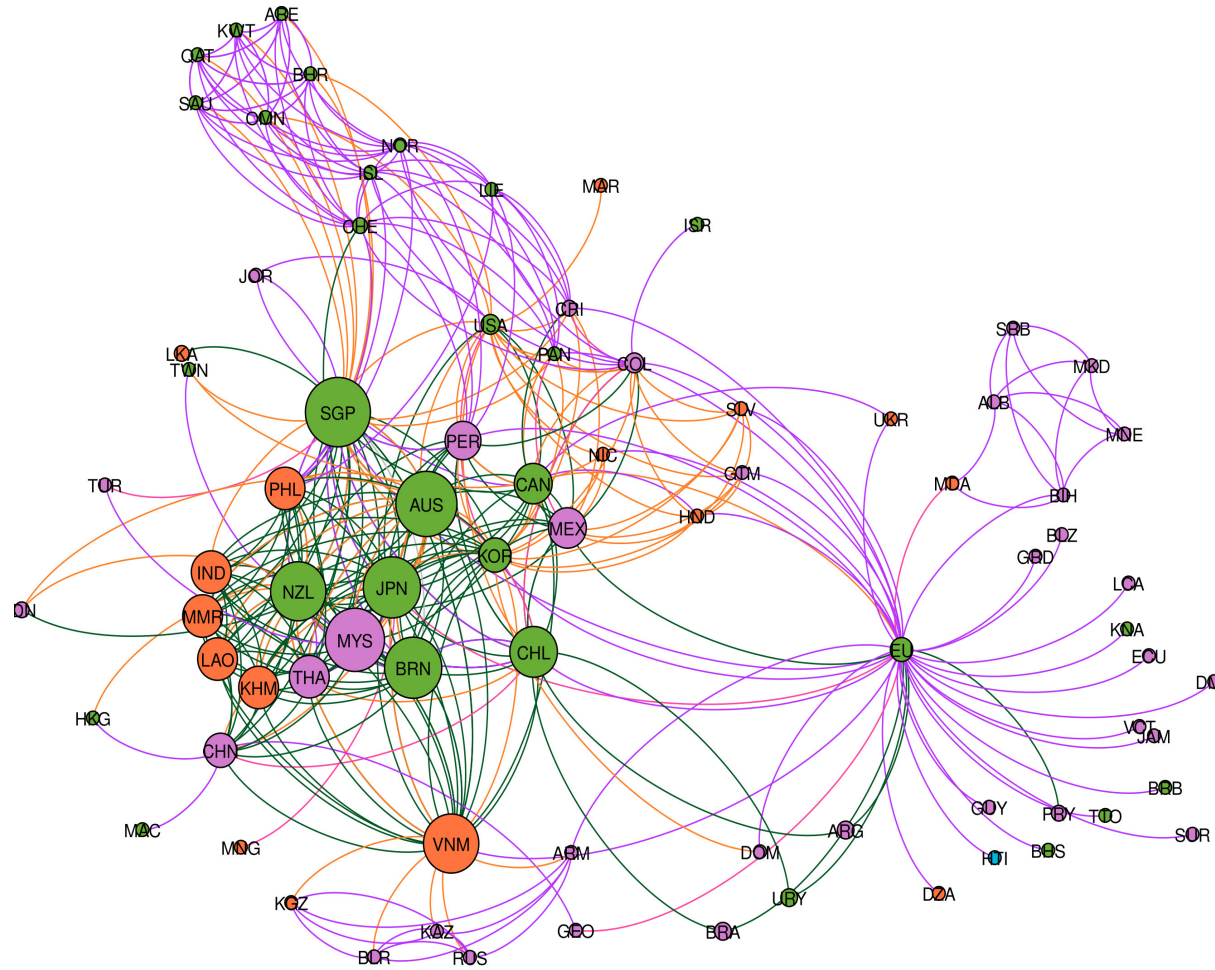
Finance



Social Media

StanfordCS224w

Graph-Structured Data is Everywhere



StanfordCS224w

<https://iit.adelaide.edu.au/news/list/2021/09/16/the-topology-of-e-commerce-governance>

MORE THAN A JOURNEY

Challenges

Graph-structured data pose significant challenges due to their irregularity, high dimensionality, and computational complexity. The major challenges include:

- Scalability and computational inefficiency
- Irregular and dynamic nature
- Data sparsity and missing values
- Complex relationships and non-Euclidean space
- Challenges in learning meaningful representations
- Privacy, security, and adversarial attacks

Homogeneous Graph

- **Key Characteristics of Homogeneous Graphs**

- ❖ **Single Node Type:** All nodes in the graph belong to the same category.
- ❖ **Single Edge Type:** All edges represent the same kind of relationship between nodes.
- ❖ **Uniform Structure:** The graph follows a consistent connectivity pattern, making it easier to apply traditional graph-based algorithms.

- **Examples of Homogeneous Graphs**

- **Social Networks (e.g., Facebook, Twitter, LinkedIn)**

- **Nodes:** Users. **Edges:** "Friends" or "Follows" relationships between users.

- **Citation Networks (e.g., Google Scholar, ArXiv, PubMed)**

- **Nodes:** Research papers. **Edges:** "Cites" relationships, where one paper references another.

- **Protein Interaction Networks (e.g., Biological Networks)**

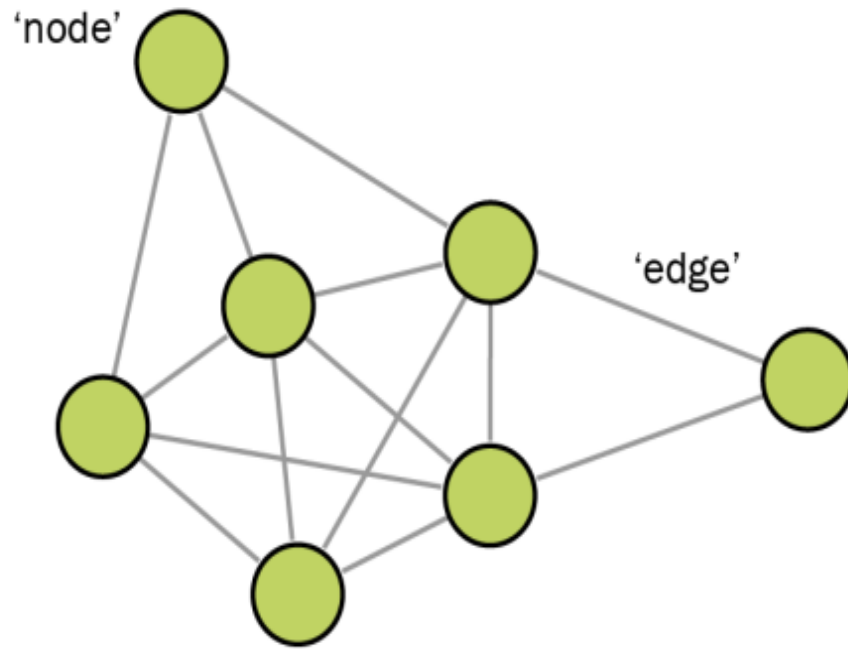
- **Nodes:** Proteins. **Edges:** "Interacts with" relationships, representing biological interactions between proteins.

How to build an effective graph?

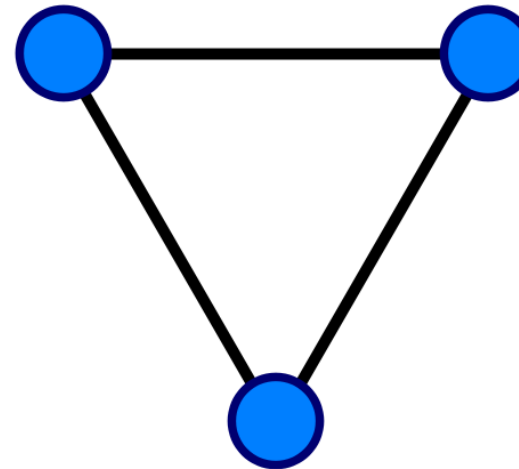
- ❖ **Nodes (or vertices)** represent the fundamental entities in a graph. They can correspond to different objects depending on the problem domain.
- ❖ **Edges (or links)** define relationships or interactions between nodes. Edges can be:
 - **Directed or undirected** (e.g., one-way vs. mutual friendships).
 - **Weighted or unweighted** (e.g., flight routes with different distances).
 - **Static or dynamic** (e.g., evolving relationships over time).
- ❖ **Choosing the Proper Network Representation.** The way we construct a graph determines our ability to extract meaningful insights. Different representations can lead to different outcomes.
 - **Cases Where Representation is Unique and Unambiguous**
 - **Cases Where Representation is Not Unique**
- ❖ **How the Choice of Links Affects the Questions You Can Study**
 - The **way you define connections** (edges) influences the type of insights you can extract.
 - If you **ignore certain relationships**, you may miss critical aspects of the data.
 - If you **add unnecessary edges**, you might introduce noise and bias in analysis.

Graph Set-up

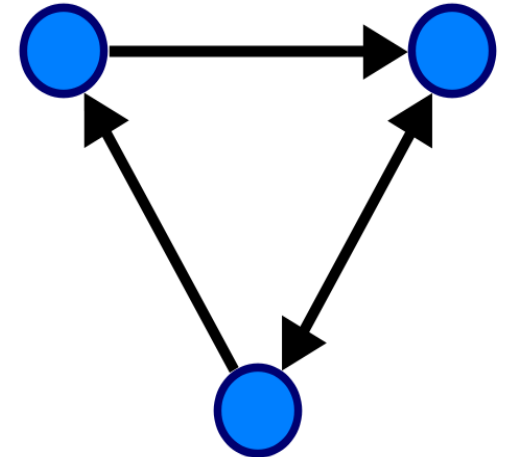
- Graph $G = (V, E)$ is defined by a set of nodes V and a set of edges E between these nodes. An edge going from node $u \in V$ to node $v \in V$ as $(u, v) \in E$.



Undirected

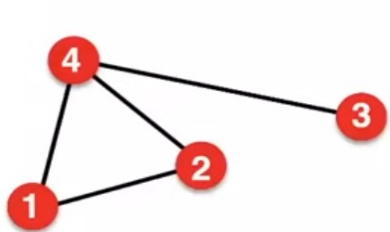


Directed



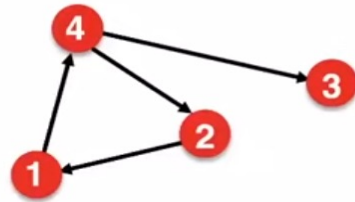
Adjacency Matrix

- A convenient way to represent graphs is through an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$. We order the nodes in the graph so that every node indexes a particular row and column in the adjacency matrix.



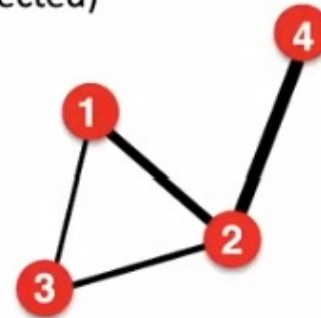
$A_{ij} = 1$ if there is a link from node i to node j
 $A_{ij} = 0$ otherwise

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

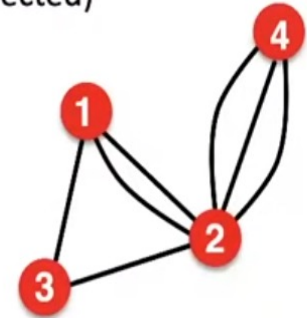


$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

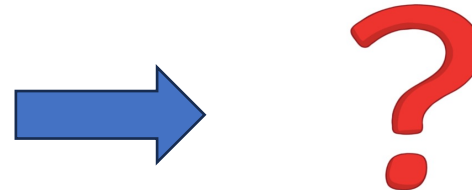
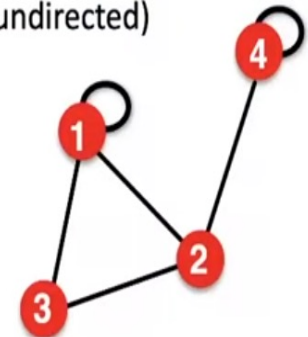
Weighted
 (undirected)



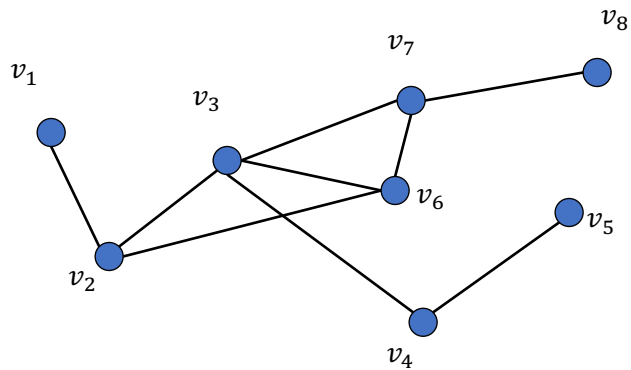
Multigraph
 (undirected)



Self-edges (self-loops)
 (undirected)



Graphs and Graph Signals



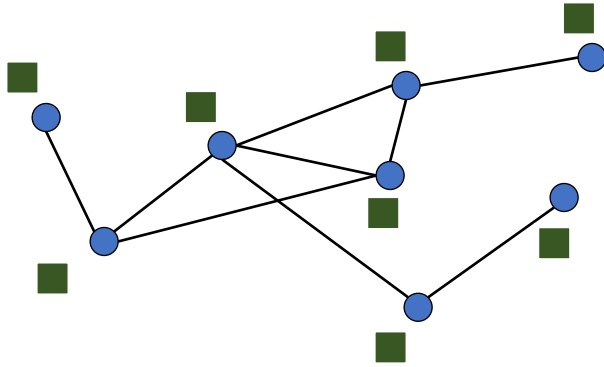
152

$$\mathcal{V} = \{v_1, \dots, v_N\}$$

$$\mathcal{E} = \{e_1, \dots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^N$

$$\mathcal{V} = \{v_1, \dots, v_N\}$$

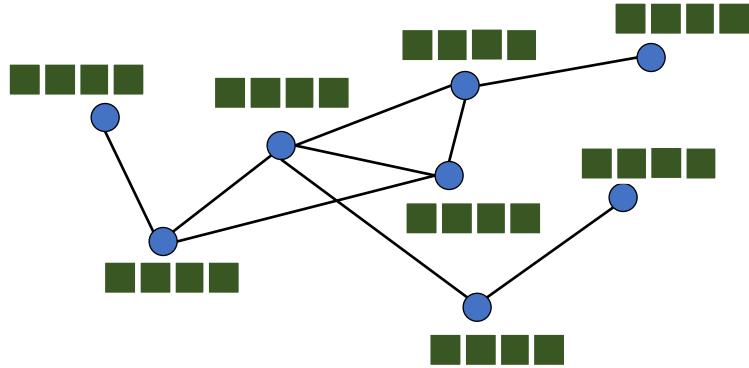
$$\mathcal{E} = \{e_1, \dots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

153

$$v \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^{N \times d}$

$$\mathcal{V} = \{v_1, \dots, v_N\}$$

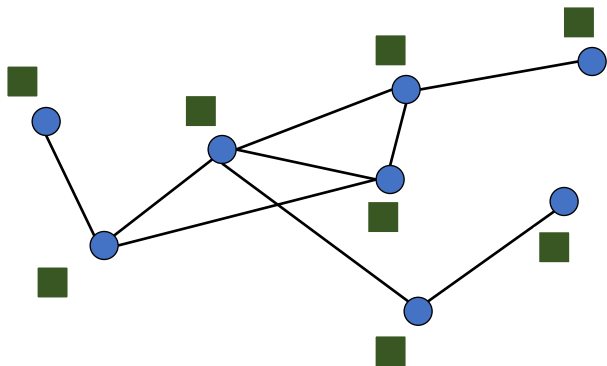
$$\mathcal{E} = \{e_1, \dots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

154

$$v \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

Graphs and Graph Signals



Graph Signal: $f : \mathcal{V} \rightarrow \mathbb{R}^N$

$$\mathcal{V} = \{v_1, \dots, v_N\}$$

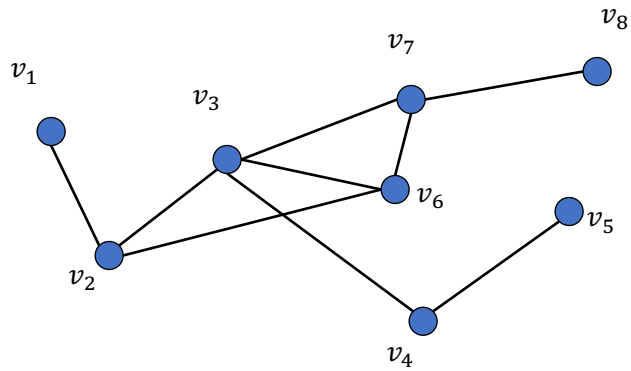
$$\mathcal{E} = \{e_1, \dots, e_M\}$$

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$$

155

$$\mathcal{V} \rightarrow \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \\ f(8) \end{bmatrix}$$

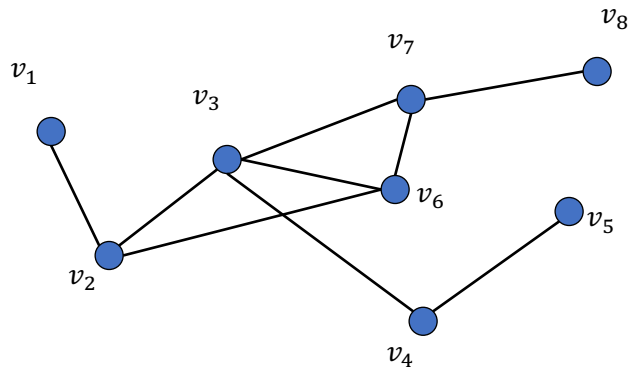
Matrix Representations of Graphs



156

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i, j] = 1$ if v_i is adjacent to v_j
 $A[i, j] = 0$, otherwise

Adjacency Matrix

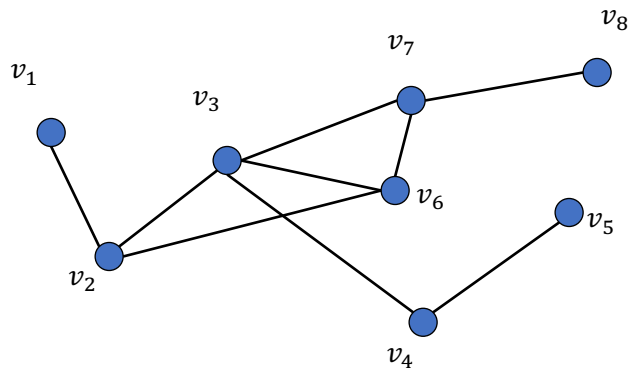
157

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

A

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i, j] = 1$ if v_i is adjacent to v_j
 $A[i, j] = 0$, otherwise

Degree Matrix: $\mathbf{D} = \text{diag}(\text{degree}(v_1), \dots, \text{degree}(v_N))$

Degree Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

\mathbf{D}

Adjacency Matrix

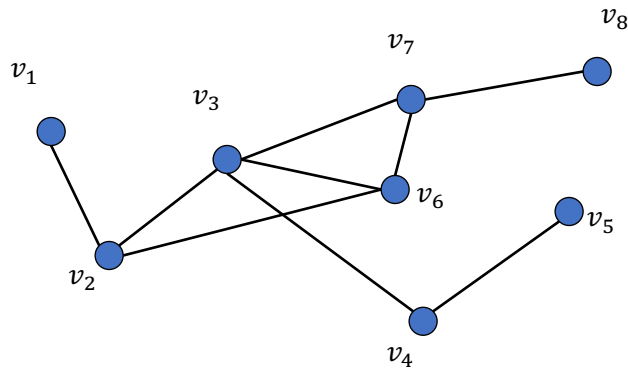
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

\mathbf{A}

158

Spectral graph theory. American Mathematical Soc.; 1997.

Matrix Representations of Graphs



Adjacency Matrix: $A[i, j] = 1$ if v_i is adjacent to v_j
 $A[i, j] = 0$, otherwise

Degree Matrix: $\mathbf{D} = \text{diag}(\text{degree}(v_1), \dots, \text{degree}(v_N))$

Degree Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

\mathbf{D}

Adjacency Matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

\mathbf{A}

159

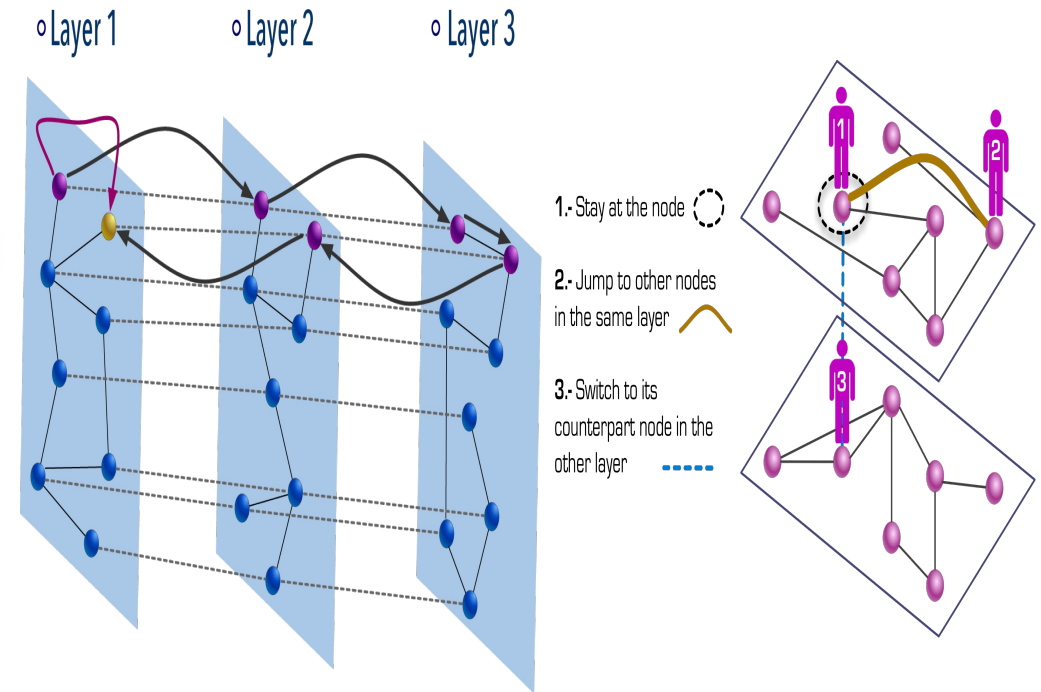
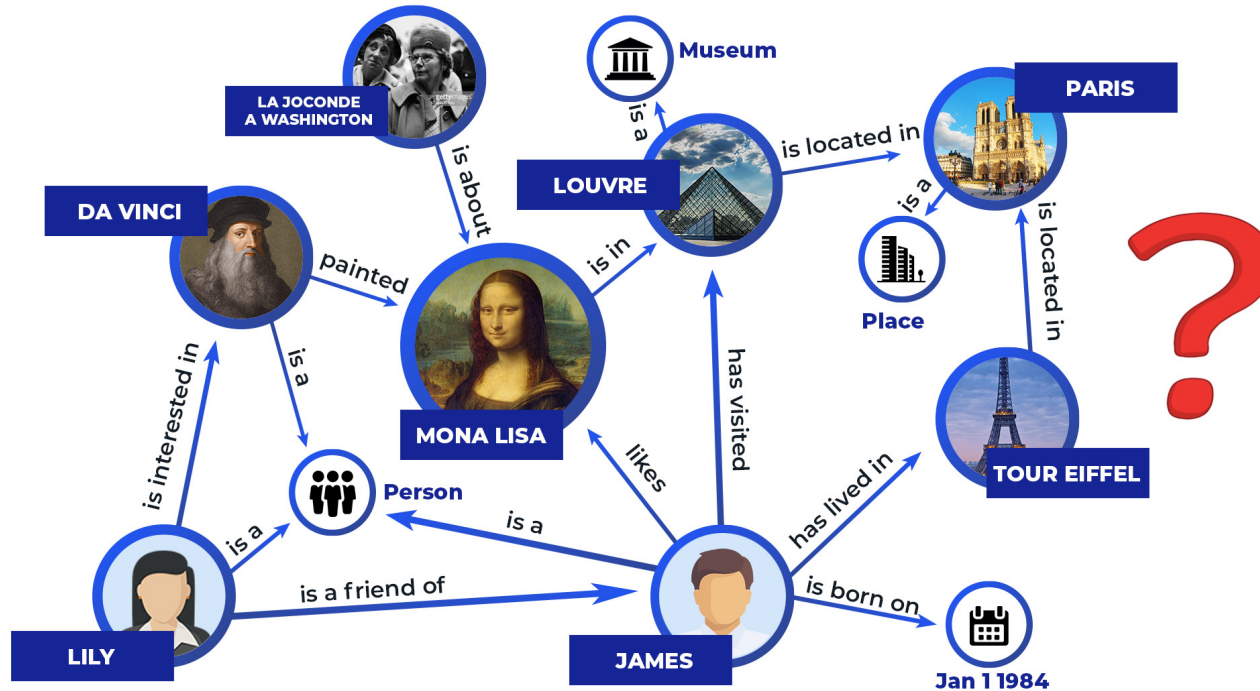
Laplacian Matrix

$$= \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

\mathbf{L}

Spectral graph theory. American Mathematical Soc.; 1997.

How to Deal with Multi-relation?



Heterogeneous Graph

Key Characteristics of Heterogeneous Graphs

- ❖ **Multiple Node Types:** Nodes represent different entities, such as users, items, papers, or institutions.
- ❖ **Multiple Edge Types:** Different relationships exist between nodes, such as "authored by," "cites"
- ❖ **Rich Semantic Information:** The diverse relationships provide deeper insights than homogeneous graphs.

Examples of Heterogeneous Graphs

➤ Academic Citation Network

Nodes: Papers, authors, journals. **Edges:** "Cites" (paper-to-paper), "Authored by" (paper-to-author).

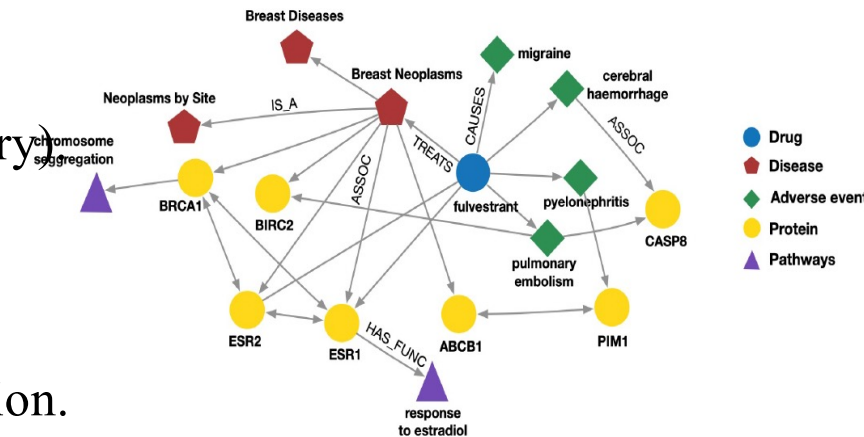
➤ Knowledge Graphs (e.g., Google Knowledge Graph, Wikidata)

Nodes: People, locations, organizations, events.

Edges: "Works at" (person-to-organization), "Located in" (place-to-country).

Why Are Heterogeneous Graphs Important?

- ❑ **More expressive** than homogeneous graphs, capturing richer information.
- ❑ **Essential for real-world applications** in social networks, recommendation systems, and knowledge graphs.
- ❑ **Enhance AI models** by incorporating multi-type relationships in representation learning.



Node, Edge, and Global Features

Node features represent **characteristics or attributes** of individual nodes for downstream tasks like **node classification, clustering, and link prediction**.

Common Types of Node Features

- ❖ **Categorical Features:** Node types (e.g., "user" or "product" in a recommendation system).
- ❖ **Numerical Features:** Values like age, price, or degree centrality.
- ❖ **Textual Features:** Descriptions, reviews, or labels in textual form.
- ❖ **Vectorized Embeddings:** Learned representations from NLP models or pre-trained embeddings.

Edge features define **relationships or interactions** between nodes for **link prediction and edge classification**.

Common Types of Edge Features

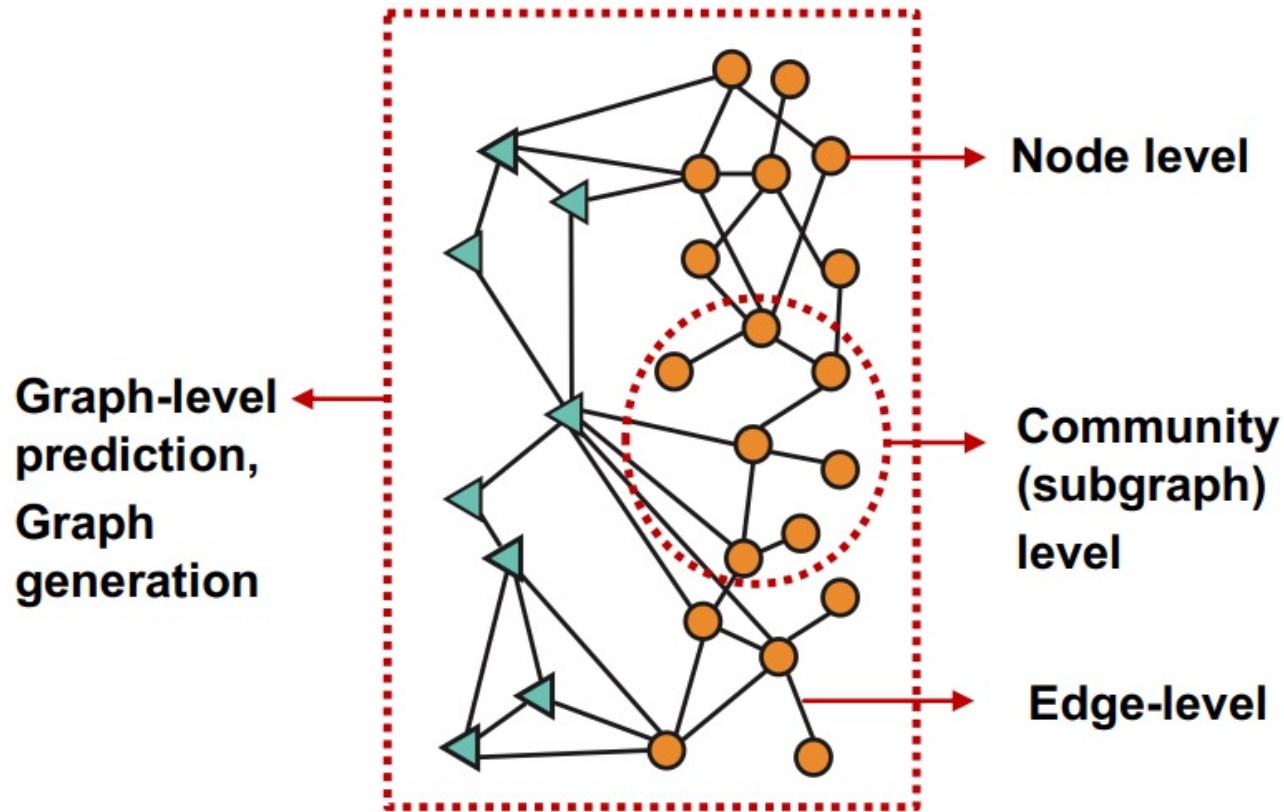
- **Weight:** The strength or importance of a connection (e.g., frequency of interactions).
- **Type:** The kind of relationship (e.g., friendship, purchase, citation).
- **Timestamp:** When the connection was established (useful for dynamic graphs).
- **Directionality:** Whether the edge is directed or undirected.

Graph-Level Features: graphs have global properties or features that apply to the entire network.

Examples include:

- **Graph Density** (How connected is the graph?).
- **Average Clustering Coefficient** (Tendency of nodes to form clusters).
- **Graph Size** (Number of nodes and edges).

Different Types of Task



Graph-based machine learning involves multiple tasks categorized by the focus of analysis. The main categories of tasks include:

▶ **Node-Level Tasks:** Predicting properties of individual nodes.

▶ **Edge-Level Tasks:** Inferring relationships between node pairs.

▶ **Community-Level Tasks:** Detecting and analyzing groups of closely connected nodes.

▶ **Graph-Level Tasks:** Understanding global graph properties.

GNN Designs

Summary

Step

1. Define Graph
2. Feature Engineering
3. Message Passing
4. Choose Architecture
5. Loss Function
6. Training
7. Evaluation
8. Deployment

Task

Nodes, edges, features

Define node and edge features

Select aggregation method

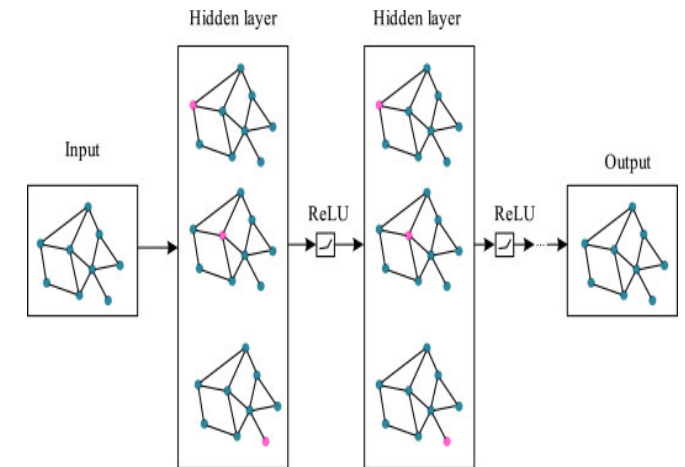
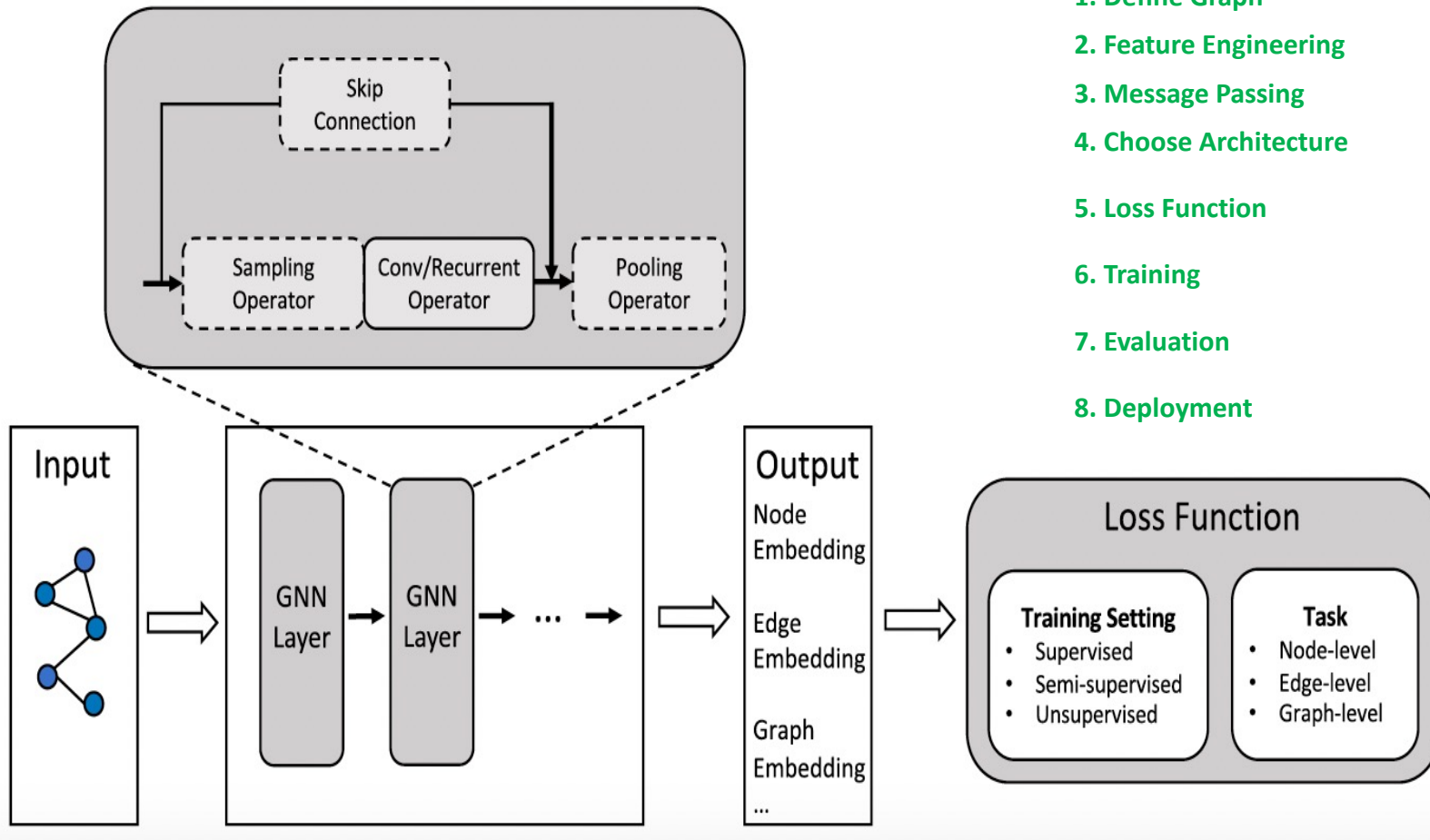
GCN, GAT, GraphSAGE, etc.

Supervised (cross-entropy), unsupervised (contrastive)

Use mini-batching and optimizers

Classification, link prediction, graph-level tasks

Optimize for inference speed



Jie Zhou, et al. (2020). AI Open

Key Modules in Graph Neural Networks

GNNs process graph-structured data by propagating and aggregating information across nodes and edges.

Three key modules in GNNs:

❖ Sampling Module

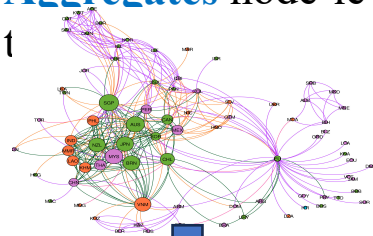
Aims to **reduce the size** of each node's neighborhood, especially for **large graphs**, preventing the neighbor explosion problem.

❖ Propagation Module

- Performs **message passing** via **convolutions** (e.g., GCNs) or **recurrent operators** (e.g., GRUs) on node features.
- Uses **skip connections** to mitigate over-smoothing and incorporate historical representations.

❖ Pooling Module

Aggregates node-level embeddings into **subgraph or graph-level** representations, extracting higher-level features needed for **classification**.



Sampling Module

- Reduces neighborhood size
- Efficient processing on large graphs

Propagation Module

- Convolution / Recurrent ops
- Skip connections mitigate over-smoothing
- Aggregates feature + topological info

Pooling Module

- Aggregates node reps into subgraph/graph-level reps
- Extracts high-level features

Tasks

- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
-

The Sampling Module

- Efficient Graph Processing via Sampling
- ▶ Direct propagation on large graphs is computationally infeasible.
- ▶ The Sampling Module reduces cost by selecting subsets of nodes or edges.
- Key Challenge:
 - ❖ **Neighbor Explosion:** The number of neighbors grows exponentially with depth. GNNs aggregate messages from each node's neighbors in the previous layer. Tracking back multiple layers can exponentially increase the neighbor set. Storing and processing all neighborhood information becomes intractable for large graphs.
 - ❖ **Computational Efficiency:** Full neighbor aggregation is impractical for large graphs.
 - ❖ **Memory Constraints:** Storing all neighborhood information for each node is infeasible.
 - ❖ **Scalability:** Enables GNNs to handle large graphs effectively.
- Common sampling techniques: Node Sampling; Layer Sampling; Subgraph Sampling.



Impact on Permutation Properties:

- ▶ Node-level predictions remain unchanged under node reordering.
- ▶ Node representations transform consistently when input ordering changes.



Impact on Task Performance:

- ▶ Preserve downstream performance in classification, link prediction, etc.
- ▶ Sampling strategies must capture essential structural information despite reduced neighborhood size.
- ▶ Aim for low variance while avoiding high computational costs.



Common Sampling Methods

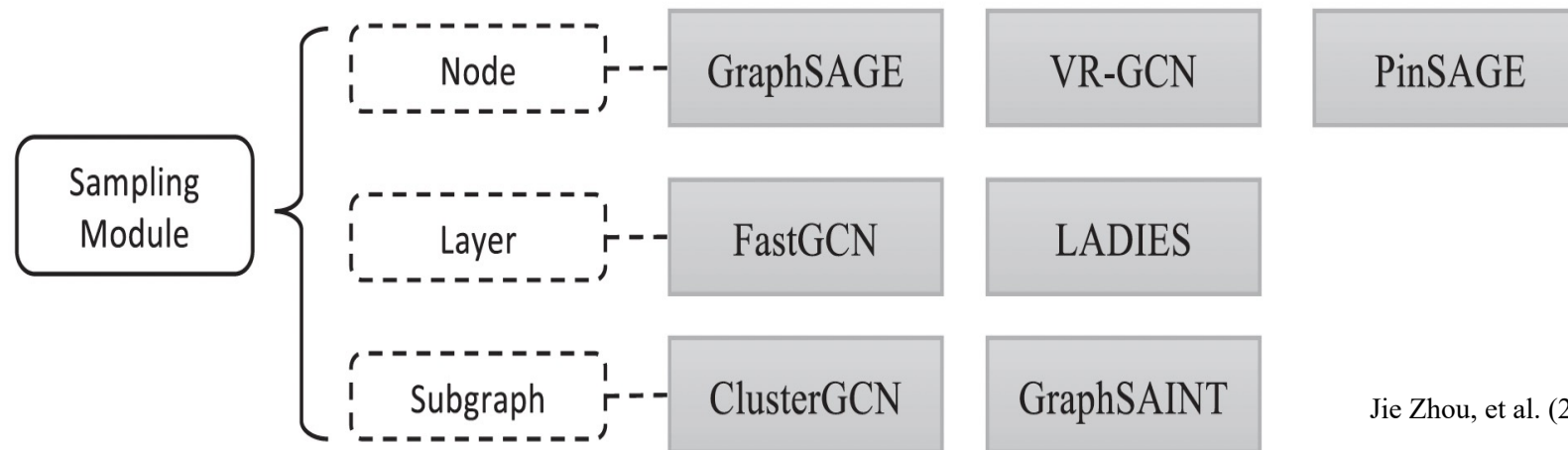
- **Node Sampling:** Selects a subset of nodes and their immediate neighbors.
-  Reduces computational complexity by limiting the number of participating nodes.
-  Often used in algorithms like GraphSAGE.

Layer Sampling: It selects a fixed number of neighbors per layer.

-  Controls exponential growth by restricting the number of aggregated neighbors.
-  Balances efficiency and performance in large-scale graphs.

Subgraph Sampling: Extracts a subgraph based on connectivity patterns.

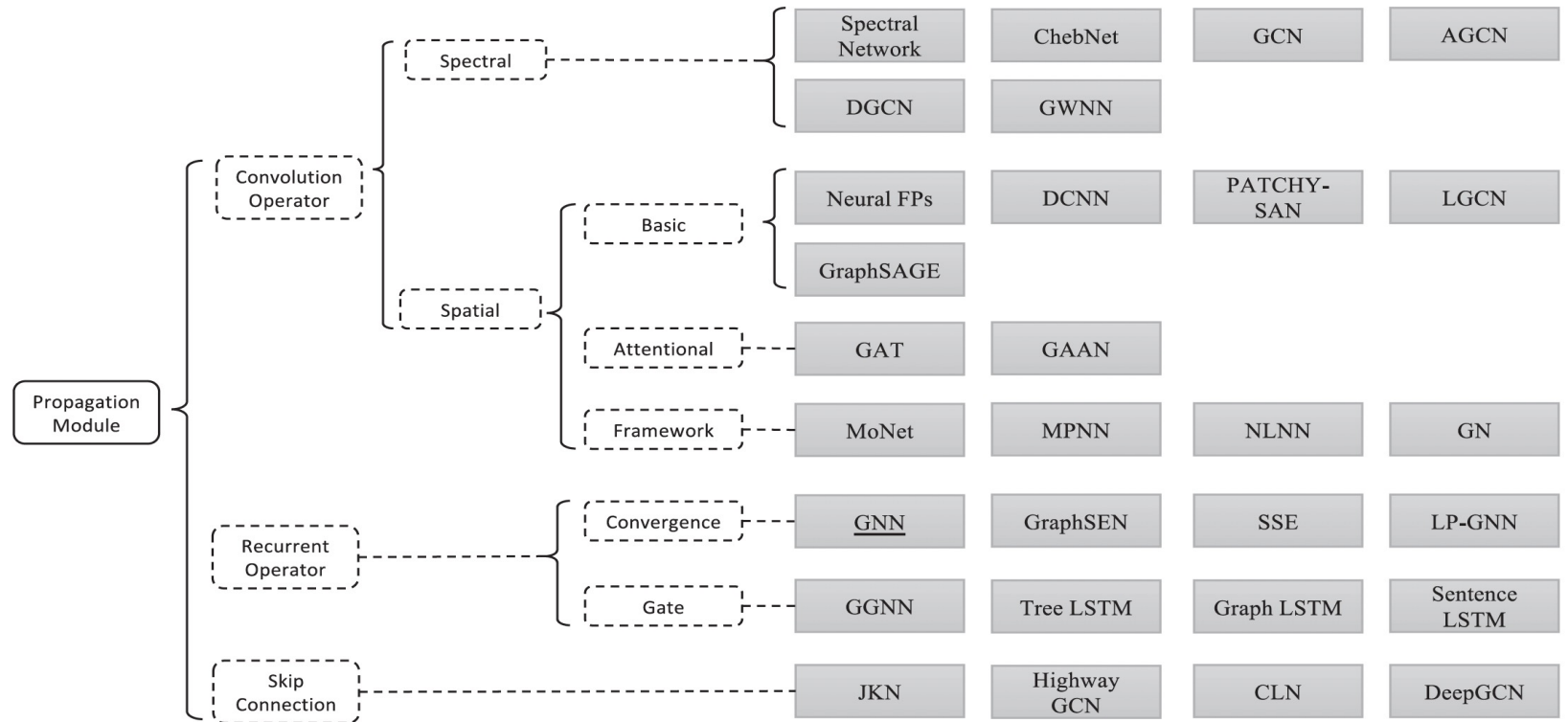
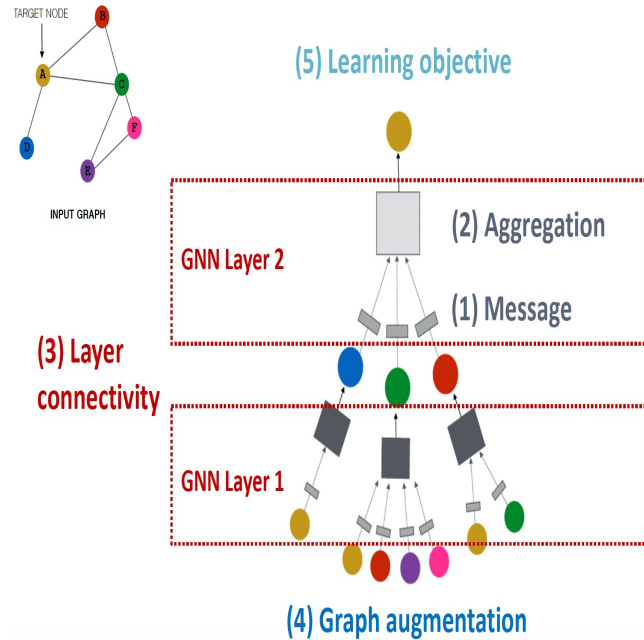
-  Useful for mini-batch training by working on graph partitions.
-  Preserves graph topology while reducing computation.



Jie Zhou, et al. (2020). AI Open

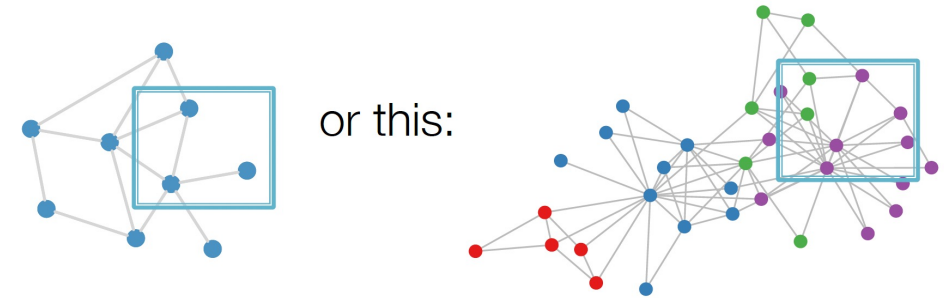
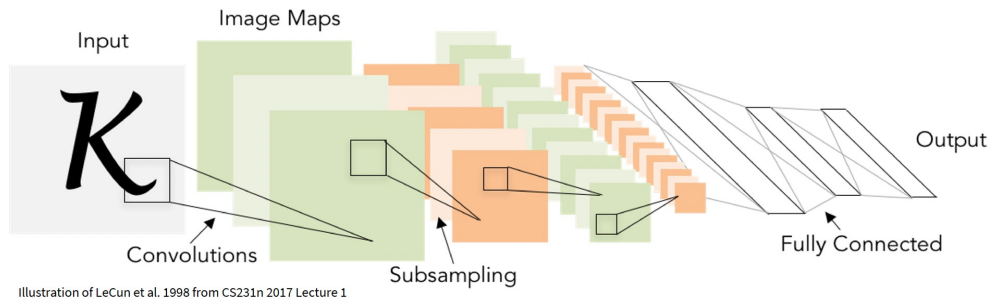
The Propagation Module

- Facilitates message passing between nodes to integrate structural and feature information.
- Key operations:
 - ❖ **Convolution Operators:** Aggregate neighbor information.
 - ❖ **Recurrent Operators:** Maintain temporal dependencies in dynamic graphs (e.g., Graph GRU, Graph LSTM).
 - ❖ **Skip Connections:** Mitigate over-smoothing by retaining historical representations.



Jie Zhou, et al. (2020). AI Open

Permutation Equivariance and Invariance

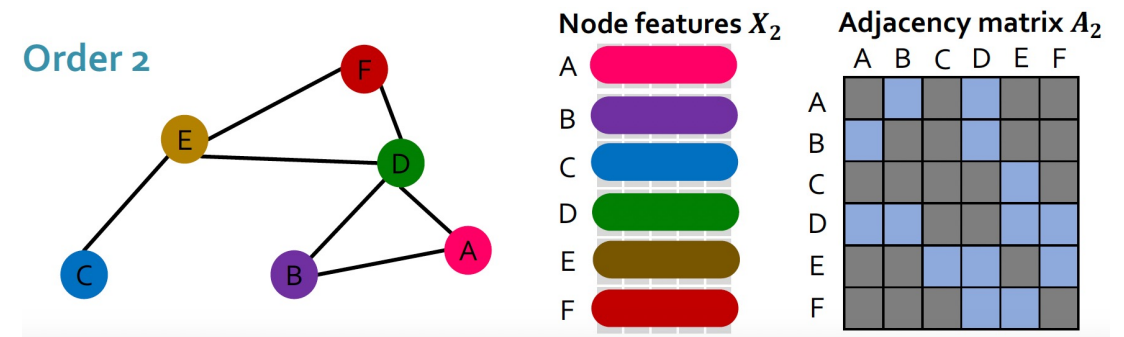
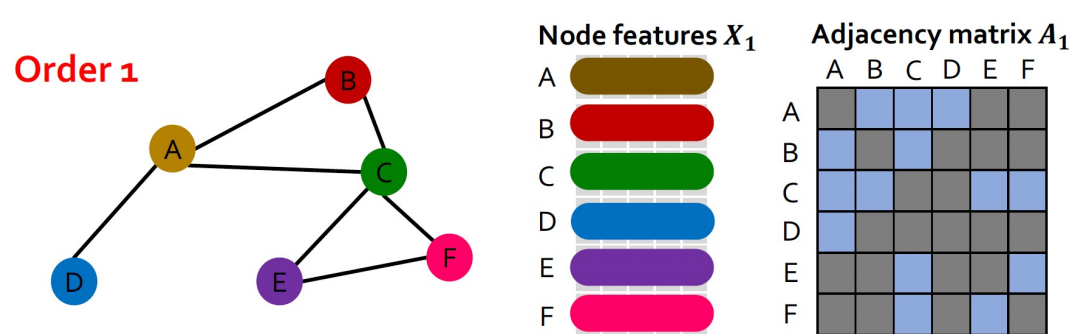


Key Observation:

- ▶ A graph does not have a fixed, canonical ordering of its nodes.
- ▶ Any permutation of node indices can still represent the same underlying graph.

Implication:

- ▶ The labeling or numbering of nodes is arbitrary.
- ▶ Reorder node IDs without changing the graph's structure.



Definition of PE and PI

Permutation Invariance:

- ▶ A function $f(\cdot)$ is *invariant* if permuting inputs does not change the output:

$$f(\pi(A, X)) = f(A, X), \quad \forall \pi \in S_n.$$

- ▶ Typical for **graph-level** tasks (e.g., entire graph classification).

Permutation on Graphs:

- ▶ Let P be an $n \times n$ permutation matrix.
- ▶ Then $A \mapsto PAP^T$, $X \mapsto PX$.
- $f(A, X) = \mathbf{1}^T X$: Permutation-**invariant**
 - Reason: $f(PAP^T, PX) = \mathbf{1}^T P X = \mathbf{1}^T X = f(A, X)$
- $f(A, X) = A X$: Permutation-**equivariant**
 - Reason: $f(PAP^T, PX) = PAP^T P X = P A X = P f(A, X)$
- $f(A, X) = X$: Permutation-**equivariant**
 - Reason: $f(PAP^T, PX) = P X = P f(A, X)$

Permutation Equivariance:

- ▶ A function $g(\cdot)$ is *equivariant* if the output is permuted in the same way:

$$g(\pi(A, X)) = \pi(g(A, X)), \quad \forall \pi \in S_n.$$

- ▶ Typical for **node-level** tasks (e.g., node embeddings, node classification).

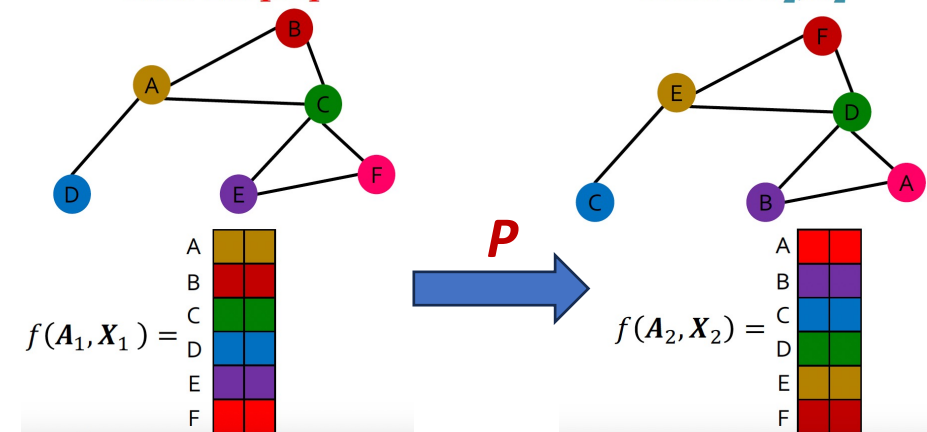
■ Permutation-invariant

$$f(A, X) = f(PAP^T, PX)$$

■ Permutation-equivariant

$$P f(A, X) = f(PAP^T, PX)$$

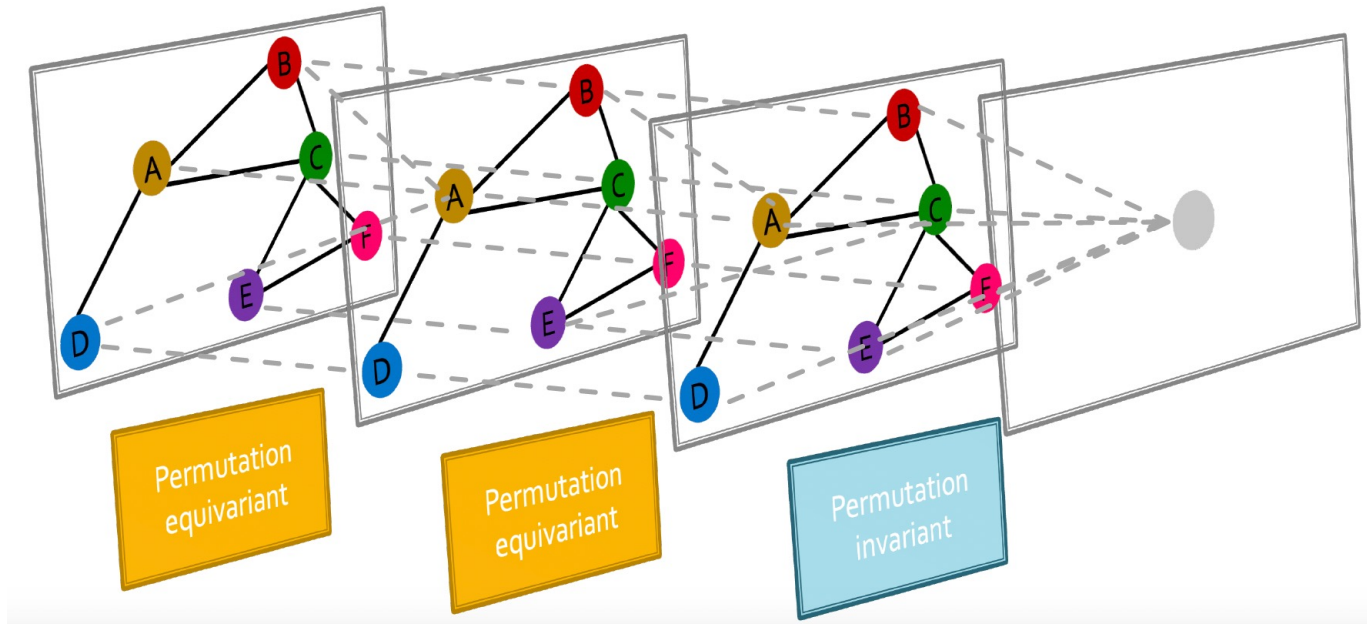
Order 1: A_1, X_1 Order 2: A_2, X_2



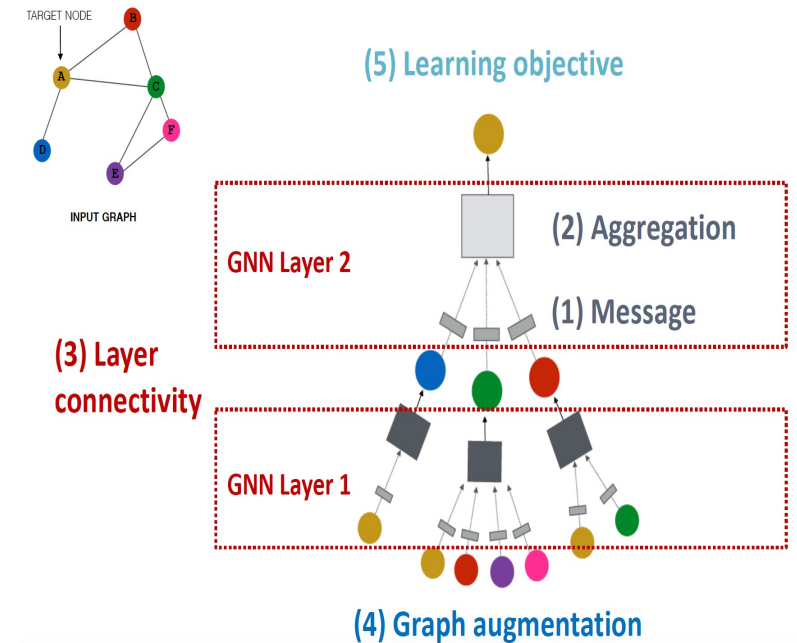
Designing GNN

- **Designing GNN Layers** must preserve or respect permutations at each update step. PI and PE are crucial for robust GNN models that handle node reorderings gracefully.
- ❖ **Sampling + Approximation:** Avoid violating permutation properties in large-scale graphs (random sampling, etc.).
- ❖ **Pooling Mechanisms:** Summation/average pooling ensures invariant graph-level outputs.
- ❖ **Challenges:** Hierarchical pooling, dynamic graphs, and advanced aggregator designs can complicate these properties.

GNN consist of multiple permutation equivariant / invariant functions.

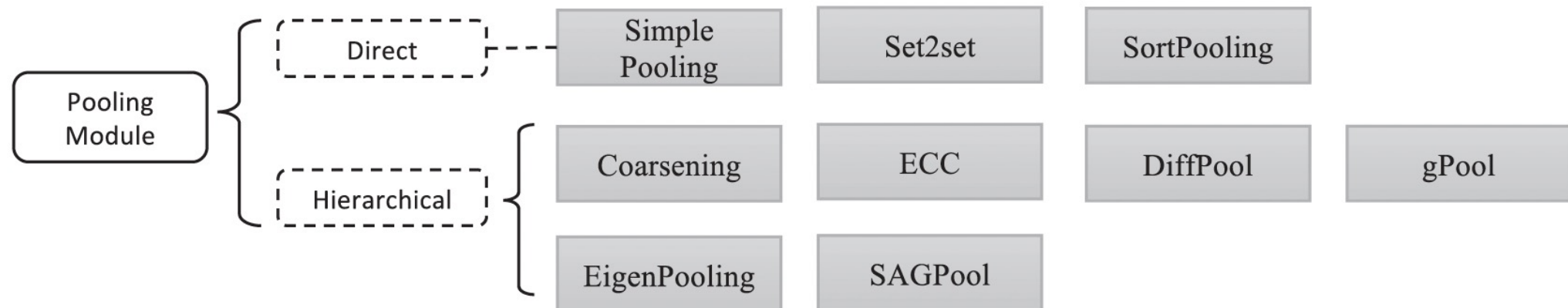


A general GNN framework



The Pooling Module

- Extracting High-Level Representations
 - ❖ Generates compact representations of subgraphs or entire graphs.
 - ❖ Essential for tasks like graph classification and hierarchical learning.
- Key pooling techniques:
 - ❖ Node Dropout Pooling: Drops less informative nodes (e.g., Top-K pooling).
 - ❖ Cluster-based Pooling: Merges similar nodes into clusters (e.g., DiffPool).
 - ❖ Attention-based Pooling: Assigns weights to nodes based on learned importance.
 - ❖ Maintaining Permutation Invariance: Ensures that graph representations remain unchanged.
- Two main categories:
 - ❖ Direct (Readout) Pooling Modules: Aggregate node embeddings into a single graph-level embedding in one step.
 - ❖ Hierarchical Pooling Modules: Iteratively coarsen (or cluster) the graph, creating a hierarchy of smaller graphs or subgraphs.



Jie Zhou, et al. (2020). AI Open

GNN Training Framework

Training Approaches

- ❖ **Supervised Learning:** Uses labeled data to train GNNs for node/graph classification.
- ❖ **Semi-supervised Learning:** Uses both labeled and unlabeled data to improve training.
- ❖ **Unsupervised Learning:** Uses self-supervision (e.g., contrastive learning) to learn node embeddings.

Prediction Tasks in GNNs

- ❖ **Node-focused:** Predicts node labels (e.g., node classification) using an MLP or softmax layer.
- ❖ **Edge-focused:** Predicts relationships between nodes (e.g., link prediction) using similarity functions or MLPs.
- ❖ **Graph-focused:** Generates graph embeddings using pooling layers for tasks like graph classification.

Cross-Entropy Loss: Example

$$L = \sum_{i \in V_{train}} y_i \log(\sigma(h_i^T \theta)) + (1 - y_i) \log(1 - \sigma(h_i^T \theta))$$

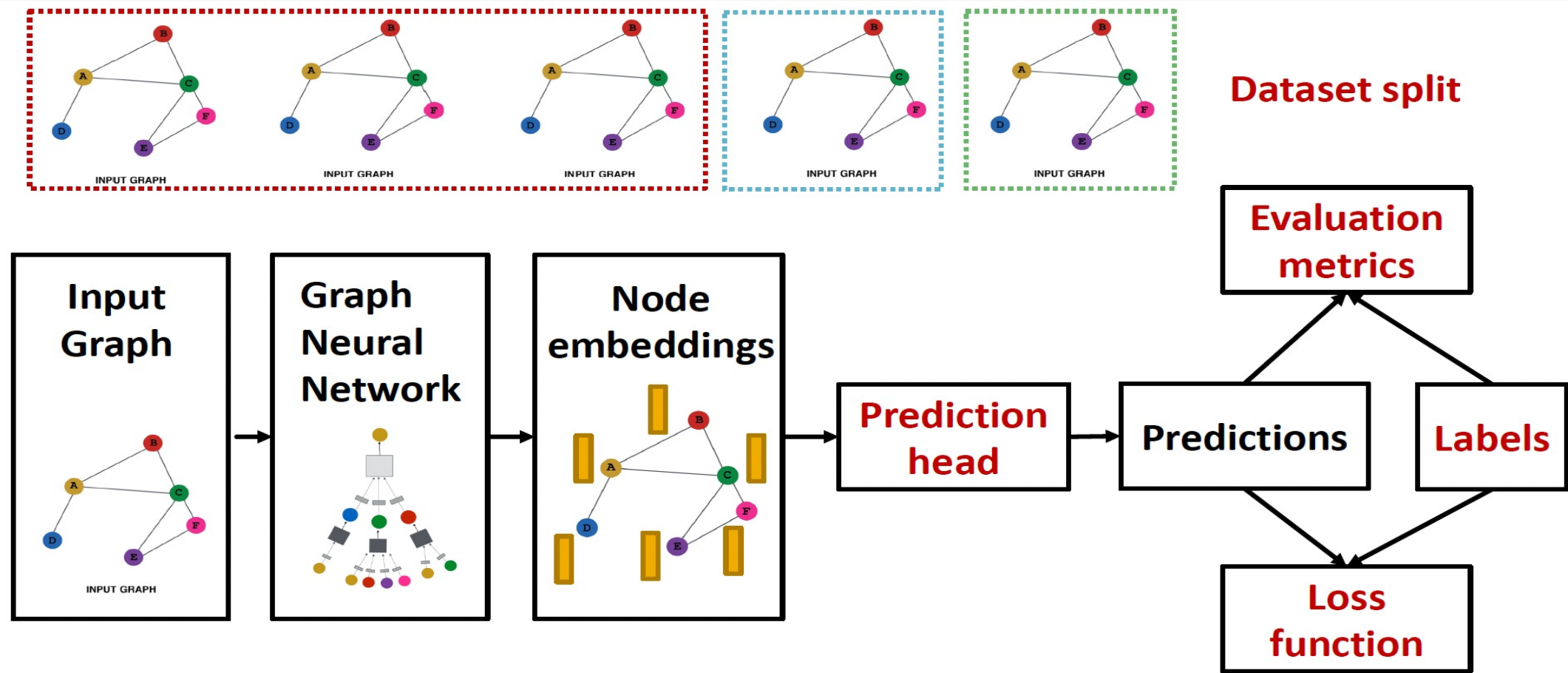
where:

- ▶ h_i is the node embedding at the last GNN layer.
- ▶ y_i is the true class label of node i .
- ▶ θ represents the classification weights.
- ▶ σ is the sigmoid function.

Types of Nodes in GNN Training

- ❖ **Training Nodes:** Used in loss computation.
- ❖ **Transductive Test Nodes:** Processed in GNN but not included in loss computation.
- ❖ **Inductive Test Nodes:** Not included in GNN computation or loss function.

GNN Training Pipeline

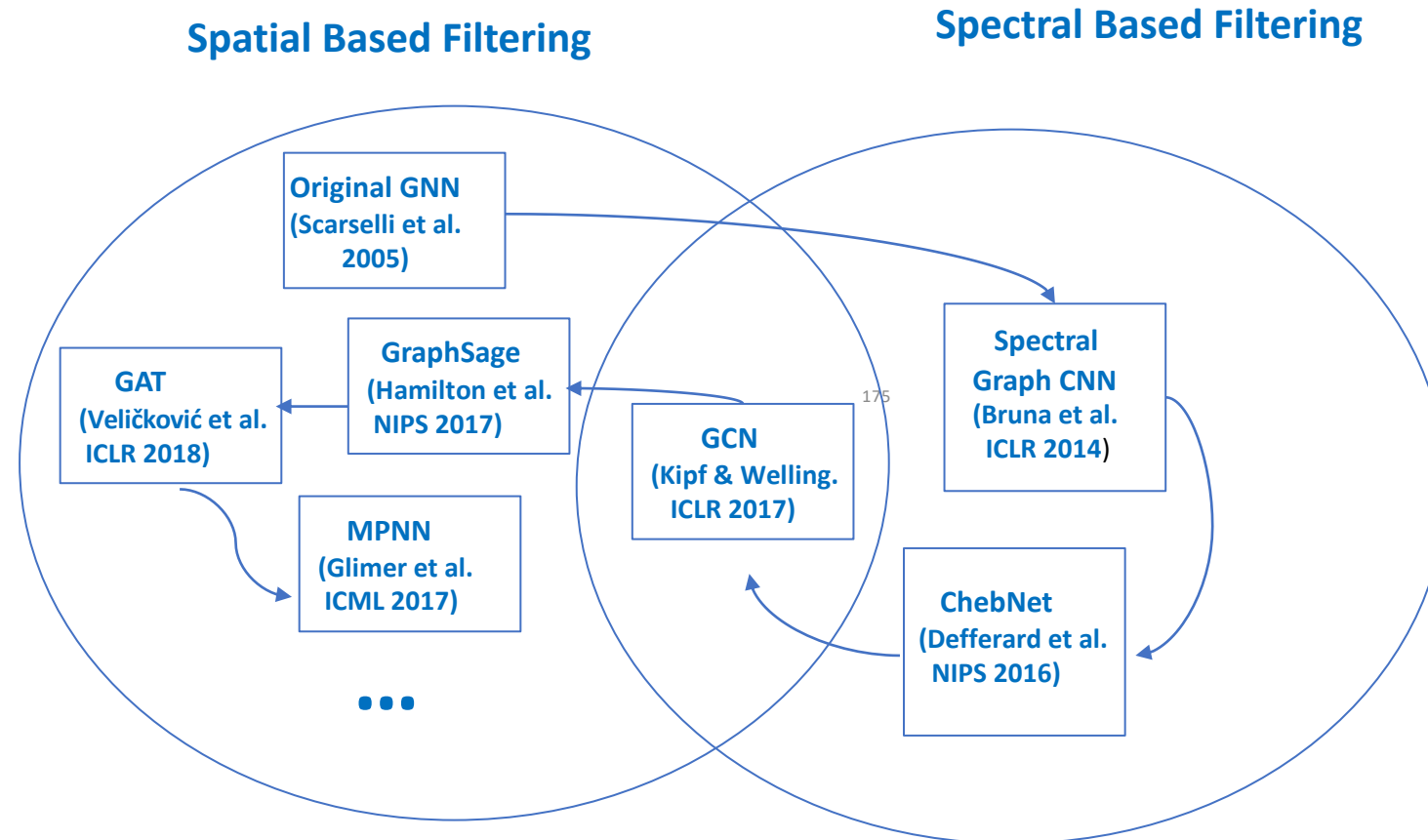


Implementation resources:

[PyG](#) provides core modules for this pipeline

[GraphGym](#) further implements the full pipeline to facilitate GNN design

Spectral and Spatial GNN Framework

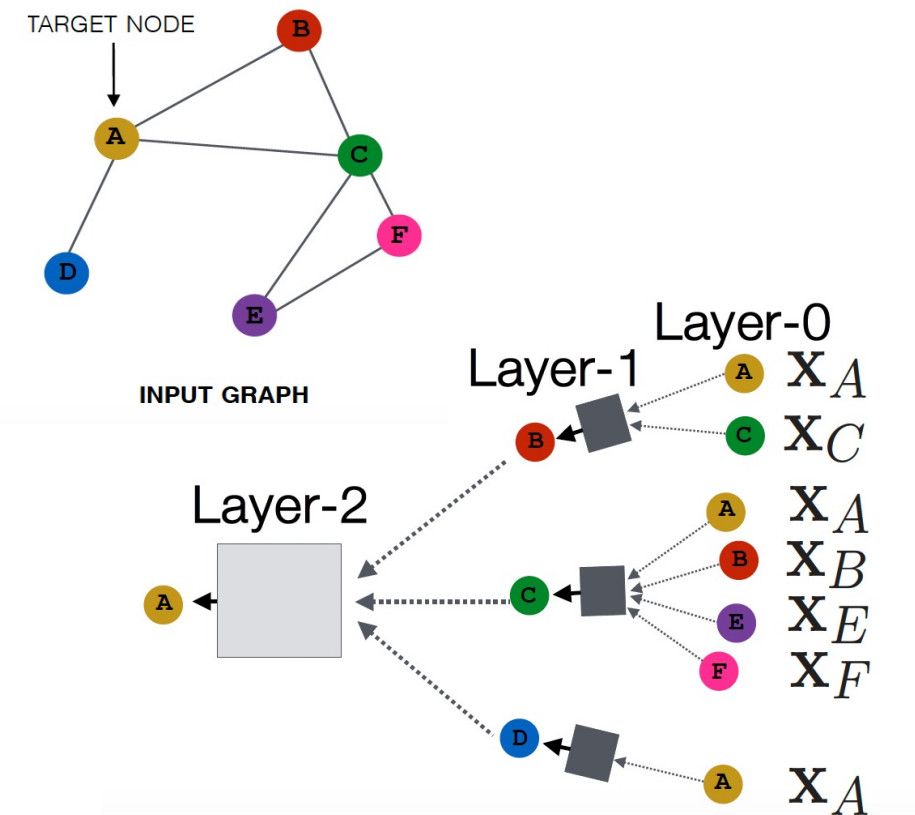
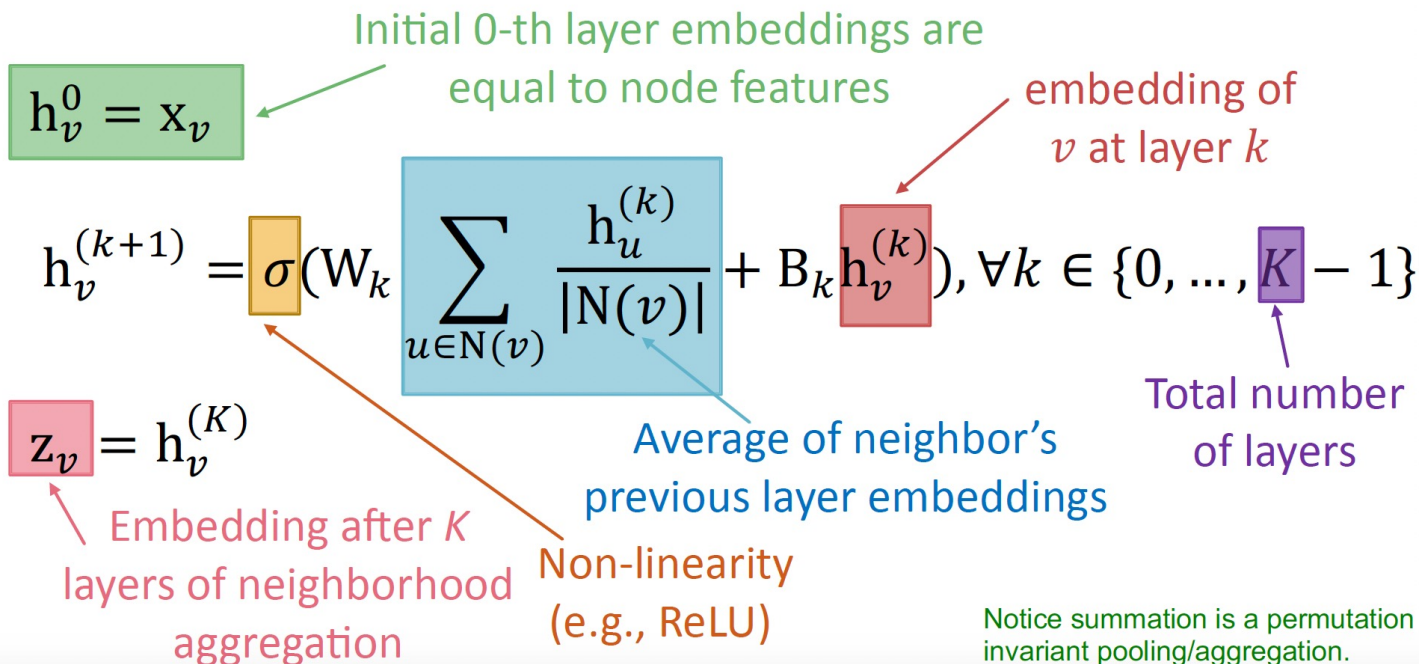


Spatial GNN Framework

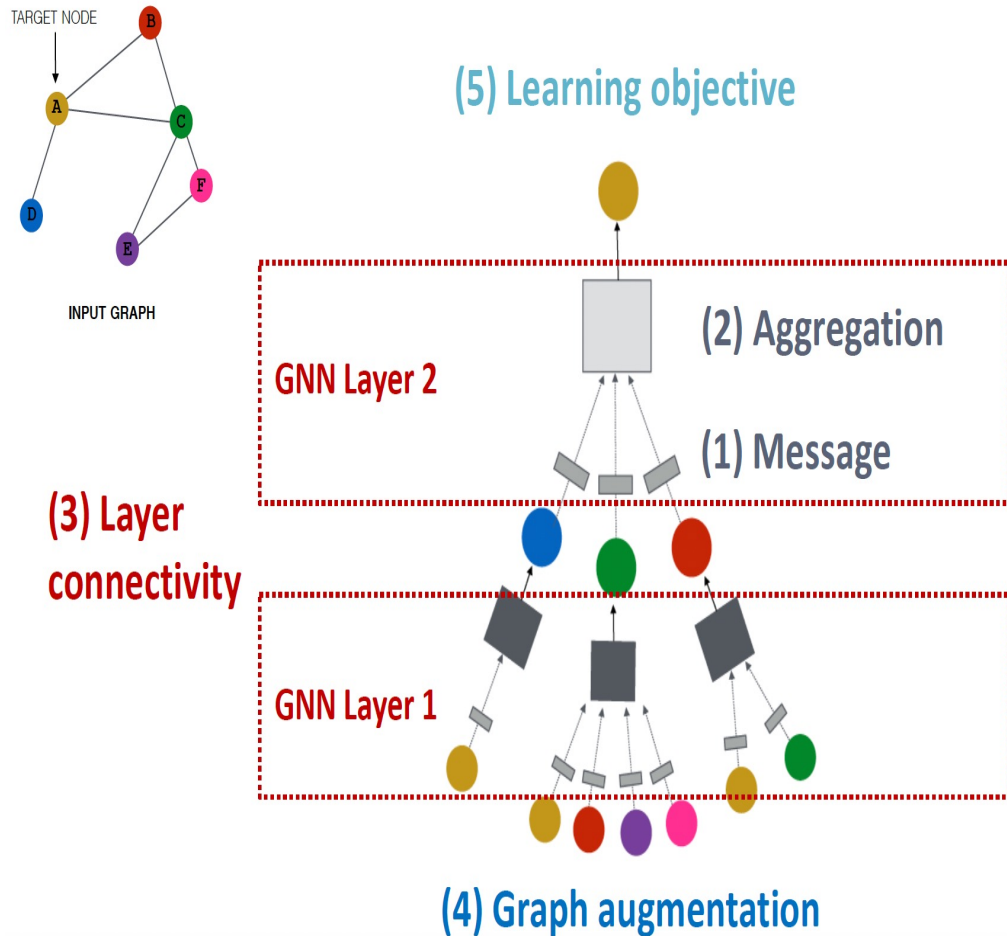
Key Concepts:

- ❖ Spatial approaches define convolutions directly on the graph using graph topology.
- ❖ Unlike spectral methods, these approaches operate in the node domain without eigen-decomposition.
- ❖ The challenge lies in handling variable neighborhood sizes and preserving local invariance.

General Spatial Convolution:



Neural Message Passing



The defining feature of a GNN is that it uses a form of *neural message passing*.

During each iteration k , a hidden embedding $h_u^{(k)}$ for node u is updated according to the information **aggregated from its neighborhood $N(u)$** , which can be expressed as follows:

$$h_u^{(k+1)} = \text{update}^{(k)} \left(h_u^{(k)}, \text{aggregate}^{(k)} \left(\{ h_v^{(k)}, \forall v \in N(u) \} \right) \right)$$

We often denote $m_{N(u)} = \text{aggregate}^{(k)} \left(\{ h_v^{(k)}, \forall v \in N(u) \} \right)$ as the “**message**” aggregated from neighborhood. The initial embeddings at $k = 0$ are set to the input features for all nodes, i.e., $h_u^{(0)} = x_u$. After running K iterations of the GNN message passing, we can use the **output of the final layer to define the embeddings for each node**, i.e., $z_u = h_u^{(K)}, \forall u \in V$.

Neural Message Passing: Intuition

Intuition Behind Message-Passing Framework

- ❖ The core idea of message passing is simple:
 - At each iteration, every node aggregates information from its 1-hop neighbors.
 - As iterations progress, nodes encode information from progressively farther regions of the graph.
- ❖ This allows nodes to capture both local and global structures over time.

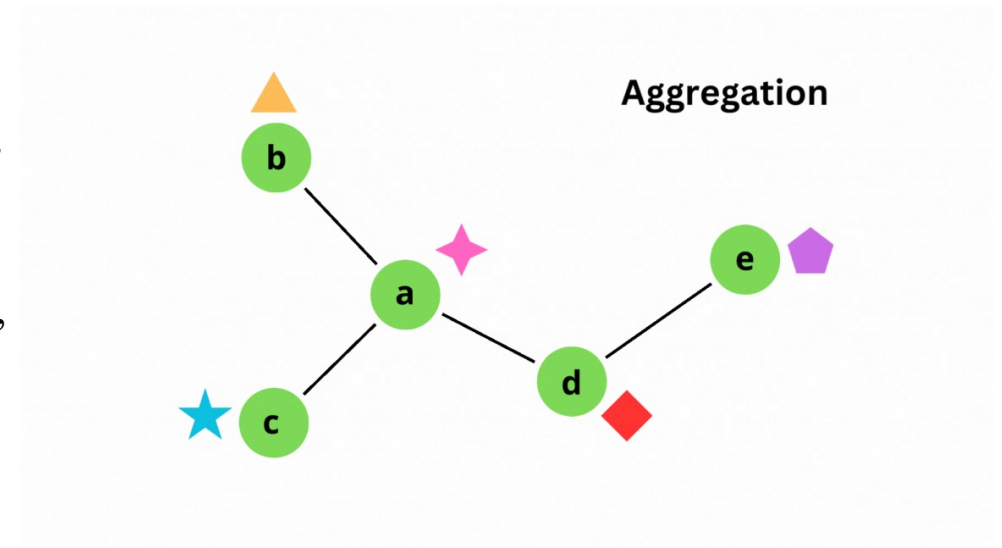
What Do Node Embeddings Encode?

Node embeddings contain two main types of information:

- **Structural Information:** Local connectivity patterns; Higher-order graph structures; the importance of a node based on its graph position (e.g., centrality measures).
- **Feature Information:** Numerical attributes (e.g., temperature, population density in spatial graphs); Categorical attributes (e.g., user preferences in recommendation systems); Learned representations from deep neural networks.

Why is Message Passing Powerful?

- ❖ Combines local and global information efficiently.
- ❖ Enables deep learning models to capture rich relational patterns.
- ❖ Supports various tasks like node classification, link prediction, and graph generation.



GNN: Basic Form

The basic GNN message passing is defined in **node-level**:

$$h_u^{(k)} = \sigma \left(W_{self}^{(k)} h_u^{(k-1)} + W_{neigh}^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)} \right)$$

where W_{self}, W_{neigh} are trainable parameter and σ denotes an elementwise non-linearity such as ReLU. Alternatively, it can also be succinctly defined in **graph-level**:

$$H^{(t)} = \sigma(H^{(k-1)} W_{self}^{(k)} + A H^{(k-1)} W_{neigh}^{(k)})$$

The basic GNN message passing can be simplified by omitting the explicit update step:

$$h_u^{(k+1)} = aggregate \left(\{ h_v^{(k)}, \forall v \in N(u) \cup \{u\} \} \right)$$

where now the aggregation is also taken over the node u itself. Adding self-loops is equivalent to sharing parameters between self and neighbor transformations.

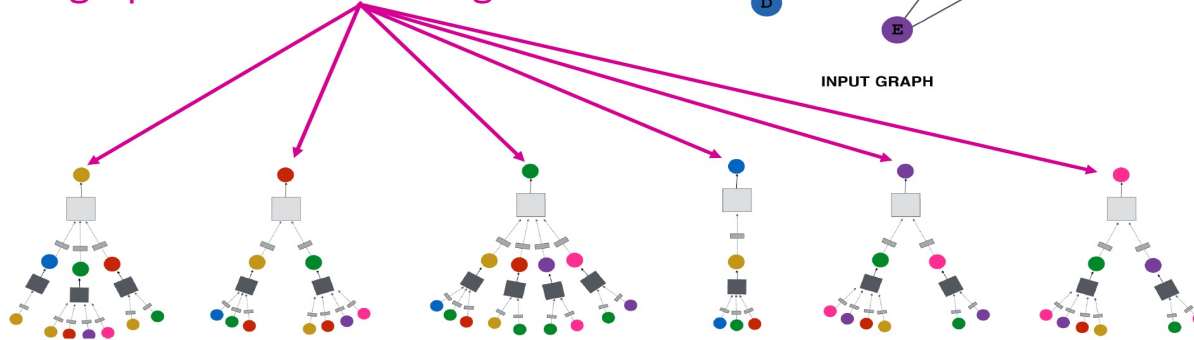
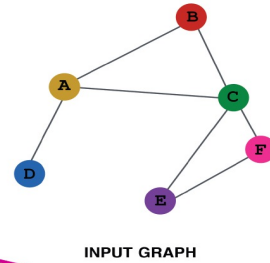
$$\mathbf{H}^{(t)} = \sigma \left((\mathbf{A} + \mathbf{I}) \mathbf{H}^{(t-1)} \mathbf{W}^{(t)} \right)$$

The self-loop GNN approach balances simplicity and efficiency but has some limitations. Self-loops make it harder to differentiate between node and neighbor information. Blurs the distinction between structural and feature information

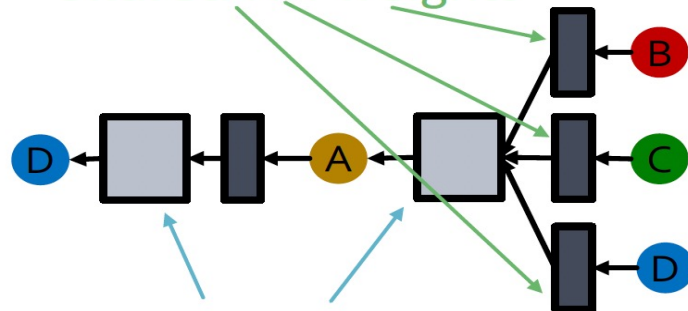
Permutation Invariant and Equivariant

- **Intuition:** Network neighborhood defines a computation graph

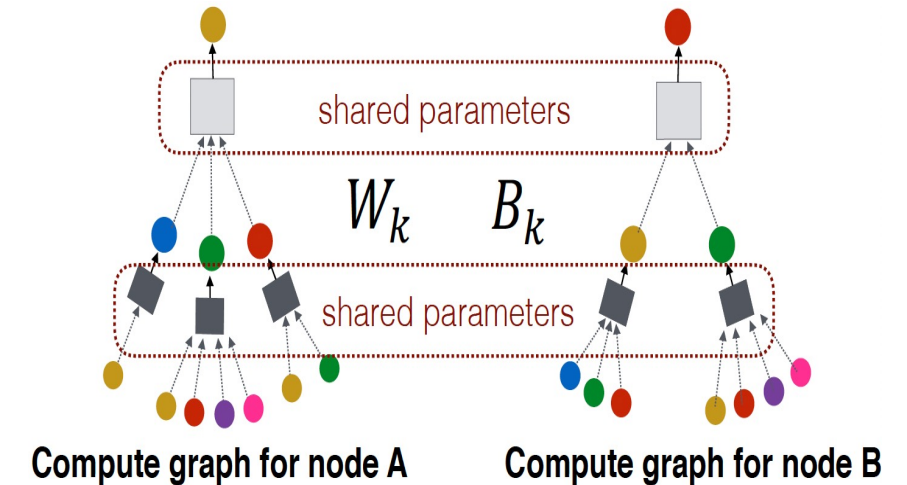
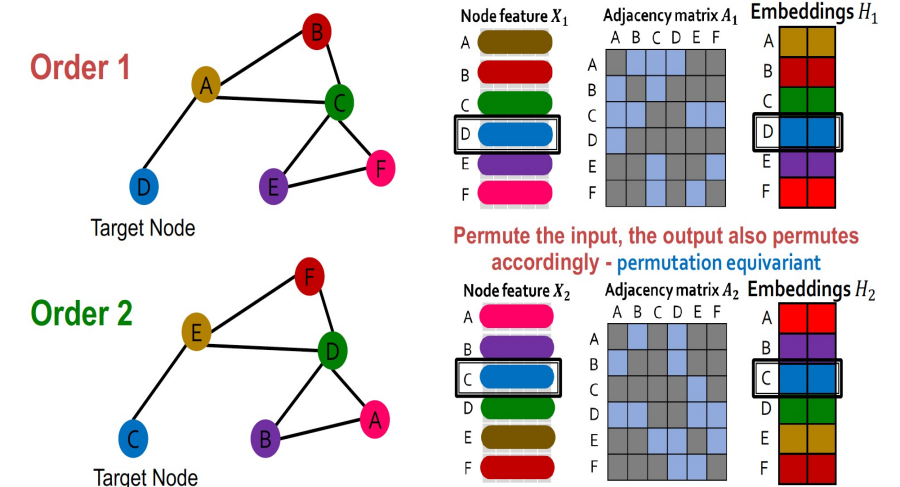
Every node defines a computation graph based on its neighborhood!



Shared NN weights



Average of neighbor's previous layer embeddings - **Permutation invariant**



Neighborhood Normalization

A basic approach is summing neighbor embeddings, but summing neighbor embeddings can create large magnitude differences. Nodes with significantly different degrees may lead to instability and optimization challenges.

$$h_u^{(k+1)} = \text{update}^{(k)} \left(h_u^{(k)}, m(N(u)) \right)$$

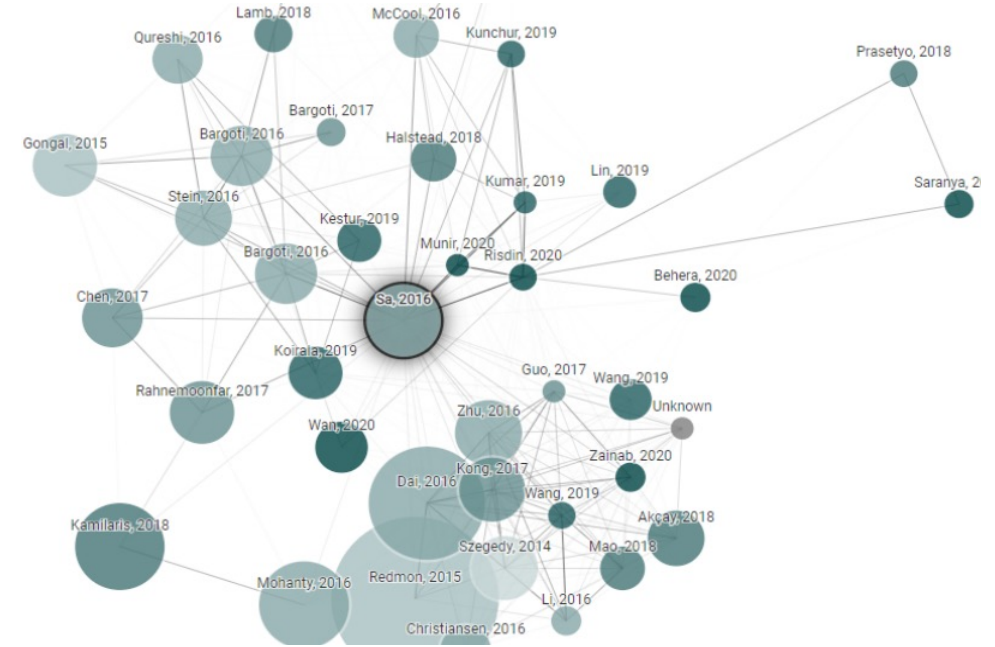
Example: A node with 100× more neighbors than another will have drastically different embedding scales. Leads to numerical instability and difficulties in optimization.

A straightforward solution is **degree-based normalization**:

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v}{|\mathcal{N}(u)|}$$

One solution to this problem is to normalize based upon the degrees of the nodes involved, which is called *symmetric normalization*:

$$m_{N(u)} = \sum_{v \in N(u)} \frac{h_v}{\sqrt{|N(u)| \times |N(v)|}}$$



Graph convolutional networks (GCNs)

$$\mathbf{h}_u^{(k)} = \sigma \left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)| |\mathcal{N}(v)|}} \right)$$

Generalized Message Passing

As the last attempt to generalize the basic neural message passing framework, now we extend the approach beyond the node level, **leveraging edge and graph-level information** at each stage.

One more generalized message passing approach can be formulized according to the following equations:

$$\begin{aligned}h_{(u,v)}^{(k)} &= \text{update}_{\text{edge}} \left(h_{(u,v)}^{(k-1)}, h_u^{(k-1)}, h_v^{(k-1)}, h_G^{(k-1)} \right) \\m_{N(u)} &= \text{aggregate}_{\text{node}} \left(\{h_{(u,v)}^{(k)}, \forall v \in N(u)\} \right) \\h_u^{(k)} &= \text{update}_{\text{node}} \left(h_u^{(k-1)}, m_{N(u)}, h_G^{(k-1)} \right) \\h_G^{(k)} &= \text{update}_{\text{graph}} \left(h_G^{(k-1)}, \{h_u^{(k)}, \forall u \in V\}, \{h_{(u,v)}^{(k)}, \forall (u,v) \in E\} \right)\end{aligned}$$

The important innovation in this framework is that we generate hidden embeddings not only for each node $h_v^{(k)}$, but also $h_{(u,v)}^{(k)}$ for each edge in the graph as well as an embedding $h_G^{(k)}$ that corresponds to the entire graph. This allows the message passing model to easily integrate edge and graph-level features and have enhanced performances compared to a standard basic GNN. Generating embeddings for edges and the entire graph also makes it trivial to **define loss functions based on the graph or edge-level classification tasks**.

Content

1 Introduction to Deep Learning

2 Neural Network Basics

3 Modern DL Model Architectures

4 Loss Functions

5 Optimization Techniques

6 Convolutional Neural Networks (CNN)

7 Graph Neural Networks (GNNs/GCNs)

8 Theoretical Properties

The Universal Approximation Theorems

Aspect	Width Version	Depth Version
Definition	A single-layer network with sufficient width can approximate any continuous function on a compact set.	A deep network with sufficient depth can approximate any Lebesgue integral function efficiently.
Focus	Number of neurons (width) in a single layer.	Number of layers (depth) in the network.
Advantages	Simple structure; can approximate any function.	More efficient; fewer parameters for the same level of approximation.
Disadvantages	Requires exponentially many neurons for high-dimensional problems.	Requires careful tuning to avoid overfitting or vanishing gradients.
Practical Implications	Rarely used due to inefficiency.	Forms the foundation of modern deep learning applications.
Efficiency	Inefficient for high-dimensional functions.	Efficient at capturing complex hierarchical relationships.
Example	Single-layer perceptron.	Deep networks like CNNs or RNNs.

Universal Approximation Theorem

Theorem 1 (Universal Approximation Theorem for Width-Bounded ReLU Networks). *For any Lebesgue-integrable function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ and any $\epsilon > 0$, there exists a fully-connected ReLU network \mathcal{A} with width $d_m \leq n + 4$, such that the function $F_{\mathcal{A}}$ represented by this network satisfies*

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| dx < \epsilon.$$

Theorem 2. *For any Lebesgue-integrable function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ satisfying that $\{x : f(x) \neq 0\}$ is a positive measure set in Lebesgue measure, and any function $F_{\mathcal{A}}$ represented by a fully-connected ReLU network \mathcal{A} with width $d_m \leq n$, the following equation holds:*

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| dx = +\infty \text{ or } \int_{\mathbb{R}^n} |f(x)| dx.$$

Theorem 3. *For any continuous function $f: [-1, 1]^n \rightarrow \mathbb{R}$ which is not constant along any direction, there exists a universal $\epsilon^* > 0$ such that for any function F_A represented by a fully-connected ReLU network with width $d_m \leq n - 1$, the L^1 distance between f and F_A is at least ϵ^* :*

$$\int_{[-1, 1]^n} |f(x) - F_A(x)| dx \geq \epsilon^*.$$

Then it's a direct comparison with Theorem 1 since in Theorem 1 the L^1 distance can be arbitrarily small.

Statistical Theory of Deep Learning

Approximation theory viewpoint

Recently, a large collection of works bridge approximation theory of neural network models with empirical processes.

Applications: Fast convergence rates of excess risks in regression and classification tasks.

Perspectives: Measuring complexities of neural networks for function approximations.

Scaling Parameters: Network width, depth, and active parameters should scale with sample size, data dimension, and function smoothness index.

Assumptions:

- Assumes global minimizers of loss functions are obtainable.
- Focuses on statistical properties without optimization concerns.
- Recognizes non-convexity of loss functions due to non-linear activation functions.

Training Dynamics Viewpoint

Understanding non-convex loss functions for neural network models is crucial. Key implications for generalization capabilities.

Key Empirical Findings: Overparameterized neural networks trained by stochastic gradient descent can fit noisy data or random noise perfectly but still generalize well.

Overparameterization Insights:

- The dynamics of deep neural networks with large enough width, trained via gradient descent (GD) in ℓ_2 -loss, behave similarly to those of functions in reproducing kernel Hilbert spaces (RKHS), where the kernel is associated with a specific network architecture.
- In the Mean-Field (MF) regime, the network parameters have the flexibility to deviate significantly from their initial values, even though it necessitates an infinite width.
- Comprehensive understanding of weight initializations and learning rate scalings in gradient-based methods.

Deep learning theory

Data $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n \sim p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$

Model $\mathbf{y}_i = f_\rho(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, 2, \dots, n,$

Assumption $\mathbb{E}(\varepsilon_i|\mathbf{x}_i) = 0$

Ideal $f_\rho := \mathbb{E}(\mathbf{y}|\mathbf{x}) = \operatorname{argmin}_{f \in \mathcal{G}} \mathcal{E}(f) := \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \rho} \left[(\mathbf{y} - f(\mathbf{x}))^2 \right]$

Estimate $\hat{f}_n = \operatorname{argmin}_{f \in \mathcal{F}(L, \mathbf{p}, \mathcal{N})} \mathcal{E}_D(f) := \operatorname{argmin}_{f \in \mathcal{F}(L, \mathbf{p}, \mathcal{N})} \left\{ \frac{1}{n} \sum_{i=1}^n (\mathbf{y}_i - f(\mathbf{x}_i))^2 \right\}$

The Risk Error $\mathcal{E}(\hat{f}_n) - \mathcal{E}(f_\rho) \leq \frac{\text{Complexity Measure of } \mathcal{F}}{n} + \frac{\text{Approx. Error}}{\sqrt{n}} + \text{Approx. Error}^2$

Approx Error $\varepsilon_{\text{Apprx}} := \sup_{f_\rho \in \mathcal{G}} \inf_{f \in \mathcal{F}(L, \mathbf{p}, \mathcal{N})} \|f - f_\rho\|_{L^p}$

Complexity $\text{VCdim}(\mathcal{F}), \text{Pdim}(\mathcal{F}) \asymp \mathcal{O}(L\mathcal{N} \log(\mathcal{N}))$

Functional Equivalence can reduce stochastic and optimization errors

Theorem 3 (Covering number of shallow neural networks)

Consider the class of shallow neural networks $\mathcal{F} := \mathcal{F}(1, d_0, d_1, B)$ parameterized by $\theta \in \Theta = [-B, B]^{\mathcal{S}}$. Suppose the radius of the domain \mathcal{X} of $f \in \mathcal{F}$ is bounded by some $B_x > 0$, and the activation σ_1 is continuous. Then for any $\epsilon > 0$, the covering number

$$\mathcal{N}(\mathcal{F}, \epsilon, \|\cdot\|_{\infty}) \leq (16B^2(B_x + 1)\sqrt{d_0}d_1/\epsilon)^{\mathcal{S}} \times \rho^{\mathcal{S}_h}/d_1!, \quad (3)$$

where ρ denotes the Lipschitz constant of σ_1 on the range of the hidden layer (i.e., $[-\sqrt{d_0}B(B_x + 1), \sqrt{d_0}B(B_x + 1)]$), and $\mathcal{S}_h = d_0d_1 + d_1$ is the total number of parameters in the linear transformation from input to the hidden layer, and $\mathcal{S} = d_0 \times d_1 + 2d_1 + 1$ is the total number of parameters.

- A reduced complexity (by $d_1!$) compared to existing studies [25, 3, 27, 23, 17]. For a shallow ReLU network with $d_1 = 128$, covering number reduced by $\approx 10^{215}$.

Theorem 4 (Covering number of deep neural networks)

Consider the class of deep neural networks $\mathcal{F} := \mathcal{F}(1, d_0, d_1, \dots, d_L, B)$ parameterized by $\theta \in \Theta = [-B, B]^{\mathcal{S}}$. Suppose the radius of the domain \mathcal{X} of $f \in \mathcal{F}$ is bounded by B_x for some $B_x > 0$, and the activations $\sigma_1, \dots, \sigma_L$ are locally Lipschitz. Then for any $\epsilon > 0$, the covering number $\mathcal{N}(\mathcal{F}, \epsilon, \|\cdot\|_{\infty})$ is bounded by

$$\frac{\left(4(L+1)(B_x+1)(2B)^{L+2}(\prod_{j=1}^L \rho_j)(\prod_{j=0}^L d_j) \cdot \epsilon^{-1}\right)^{\mathcal{S}}}{d_1! \times d_2! \times \dots \times d_L!},$$

where $\mathcal{S} = \sum_{i=0}^L d_i d_{i+1} + d_{i+1}$ and ρ_i denotes the Lipschitz constant of σ_i on the range of $(i-1)$ -th hidden layer, especially the range of $(i-1)$ -th hidden layer is bounded by $[-B^{(i)}, B^{(i)}]$ with $B^{(i)} \leq (2B)^i \prod_{j=1}^{i-1} \rho_j d_j$ for $i = 1, \dots, L$.

- A reduced complexity (by $(d_1!d_2! \dots d_L!)$) over existing studies [25, 3, 27, 23, 17].
- Increasing depth L does increase complexity. The increased hidden layer l will have a $(d_l!)$ discount on the complexity.

Deep learning theory

- Much of the current theoretical understanding is counterintuitive and falls short of explaining why deep learning or reinforcement learning methods perform effectively in real-world scenarios. There is **a big gap** between popular deep learning algorithms and current theoretical results.
- Many deep learning (DL) theoretical studies primarily focus on fully connected neural networks (FNN) within nonparametric settings, while making **unrealistic assumptions**.
- Key breakthroughs in algorithmic modeling often lack a solid mathematical foundation due to the absence of powerful tools in such complex scenarios.
- Furthermore, existing methodologies, such as traditional harmonic analysis and empirical process theory, are insufficient for addressing **heterogeneous object structures (e.g., Lie group/algebra)** commonly encountered in computer vision (CV) and natural language processing (NLP).



Theoretical Properties

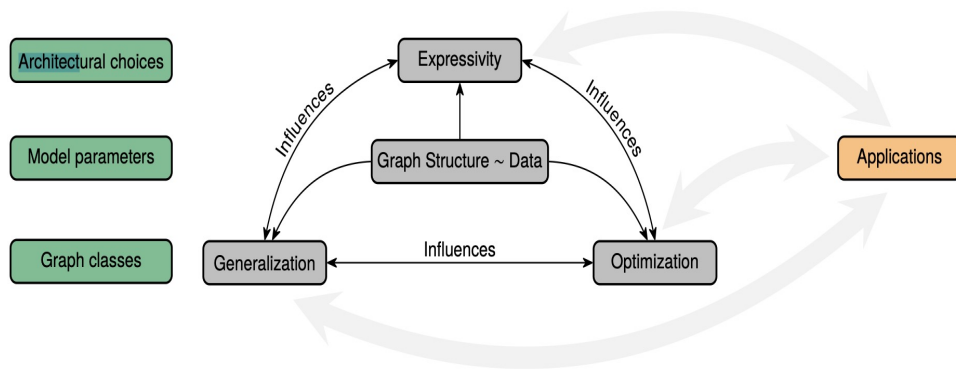


Figure 1: Interactions of the four challenges within graph machine learning: Fine-grained *expressivity*, *generalization*, *optimization*, *applications*, and their interactions. The green boxes *architectural choices* (hyperparameter and other design choices like normalization layers), *model parameters*, and *graph classes* (different types of graphs) represent aspects of all four challenges.

- ❖ **Expressivity:** What graph structures can a GNN distinguish?
 - Traditional results relate GNNs to the 1-WL test, but finer geometric notions are needed.
- ❖ **Approximation:** Under what conditions can GNNs approximate continuous, permutation-invariant functions?
 - Universal approximation results require a careful treatment of the topology of graph space.
- ❖ **Generalization:** How well do GNNs perform on unseen graphs?
 - Existing VC-dimension based bounds are loose and do not fully capture the influence of architectural choices and graph structure.

Future Directions

- ❖ Develop fine-grained expressivity results that quantify not only if two graphs are distinguishable, but how similar they are.
- ❖ Derive uniform approximation bounds for GNNs using a refined topology on graph space.
- ❖ Establish tighter generalization bounds that incorporate architectural choices and graph geometry.
- ❖ Explore the interplay between expressivity, optimization, and generalization to inform the design of more robust GNN architectures.

Morris, Christopher, Fabrizio Frasca, Nadav Dym, Haggai Maron, Ismail Ilkan Ceylan, Ron Levie, Derek Lim, Michael M. Bronstein, Martin Grohe, and Stefanie Jegelka. "Position: Future Directions in the Theory of Graph Machine Learning." In *Forty-first International Conference on Machine Learning*.

Deepset

A function f transforming a set $X = \{x_1, \dots, x_M\}$ into Y should be:

- ▶ **Permutation invariant:** The output does not change under reordering.

$$f(\{x_1, \dots, x_M\}) = f(\{x_{\pi(1)}, \dots, x_{\pi(M)}\})$$

for any permutation π .

- ▶ **Permutation equivariant:** The output follows the permutation.

$$f([x_{\pi(1)}, \dots, x_{\pi(M)}]) = [f_{\pi(1)}(x), \dots, f_{\pi(M)}(x)]$$

Theorem. A function $f(X)$ is invariant to the permutation of instances in X if and only if it can be decomposed as:

$$f(X) = \rho \left(\sum_{x \in X} \phi(x) \right)$$

where ϕ and ρ are suitable transformations.

The standard neural network layer is represented as:

$$f_{\Theta}(\mathbf{x}) = \sigma(\Theta \mathbf{x})$$

where $\Theta \in \mathbb{R}^{M \times M}$ is the weight matrix.

Theorem. A function $f_{\Theta} : \mathbb{R}^M \rightarrow \mathbb{R}^M$ is permutation equivariant if:

$$\Theta = \lambda I + \gamma(11^T)$$

where:

- ▶ I is the identity matrix,
- ▶ $1 = [1, \dots, 1]^T$,
- ▶ $\lambda, \gamma \in \mathbb{R}$.

de Finetti's theorem states that any exchangeable model can be factored as

$$p(X|\alpha, M_0) = \int d\theta \left[\prod_{m=1}^M p(x_m|\theta) \right] p(\theta|\alpha, M_0).$$

where θ is a latent feature and α, M_0 are hyper-parameters of the prior.

For Exponential Family with Conjugate Priors:

$$p(X|\alpha, M_0) = \exp \left(h \left(\alpha + \sum_m \phi(x_m), M_0 + M \right) - h(\alpha, M_0) \right)$$

PINE

Let \mathbf{f} be a continuous real-valued function defined on a compact set with the following form

$$\mathbf{f}\left(\underbrace{\mathbf{x}_{1,1}, \mathbf{x}_{1,2}, \dots, \mathbf{x}_{1,N_1}}_{G_1}, \underbrace{\mathbf{x}_{2,1}, \dots, \mathbf{x}_{2,N_2}}_{G_2}, \dots, \underbrace{\mathbf{x}_{K,1}, \dots, \mathbf{x}_{K,N_K}}_{G_K}\right),$$

where $\mathbf{x}_{k,n} \in \mathbb{R}^{M_k}$. If function \mathbf{f} is partial permutation invariant, then the PINE framework provides a

Core Representation Theorem as

$$\mathbf{f}(\cdot) = \mathbf{h}\left(\sum_{n=1}^{N_1} \mathbf{g}_1(\mathbf{x}_{1,n}), \sum_{n=1}^{N_2} \mathbf{g}_2(\mathbf{x}_{2,n}), \dots, \sum_{n=1}^{N_K} \mathbf{g}_K(\mathbf{x}_{K,n})\right) + o(1)$$

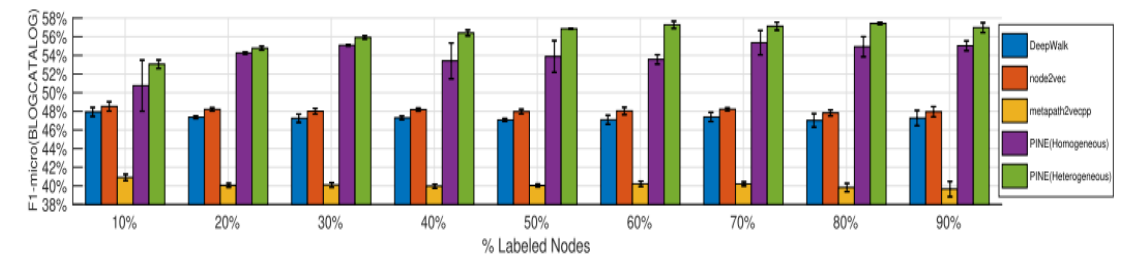
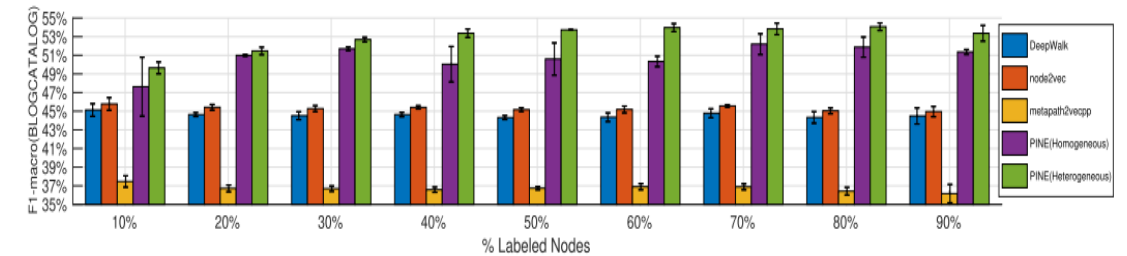
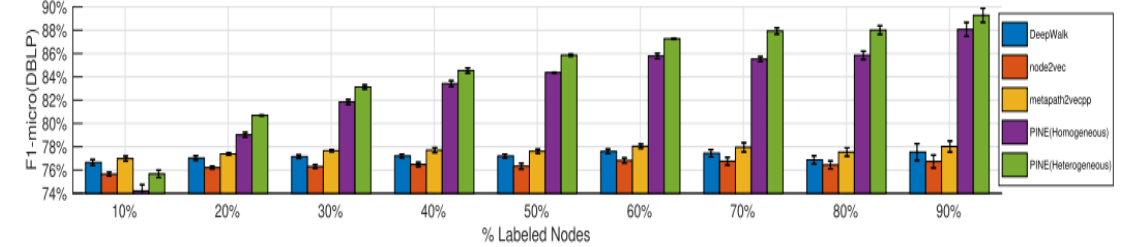
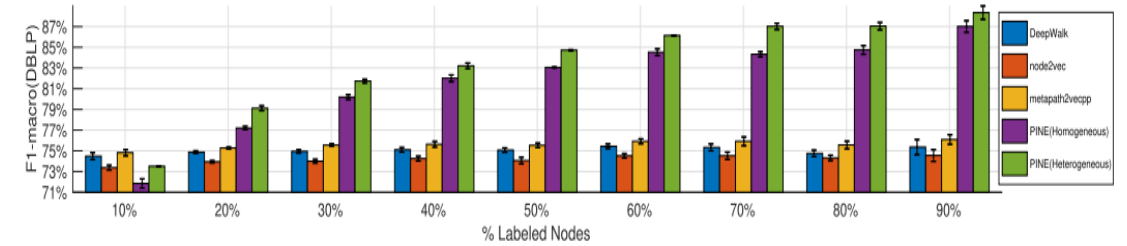
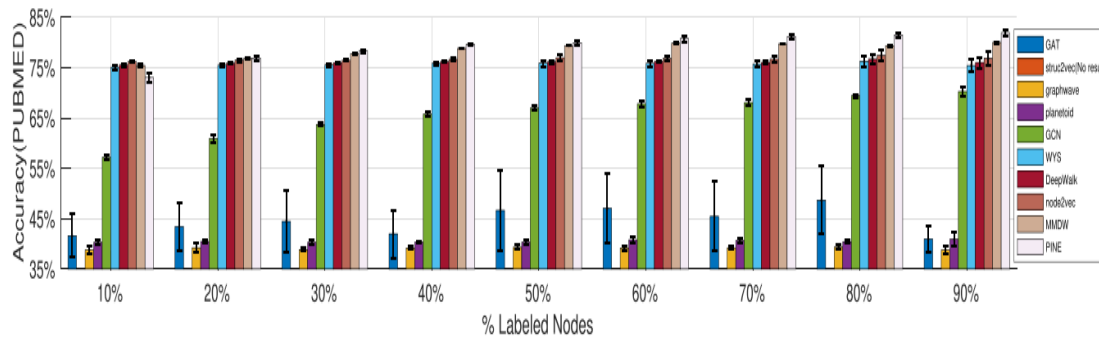
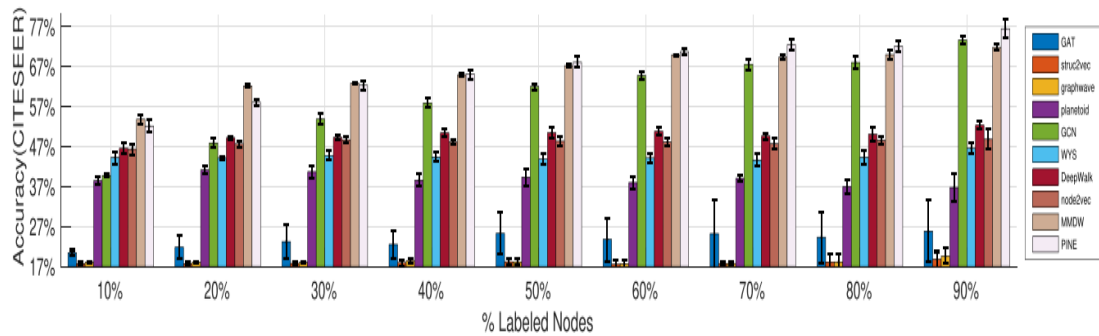
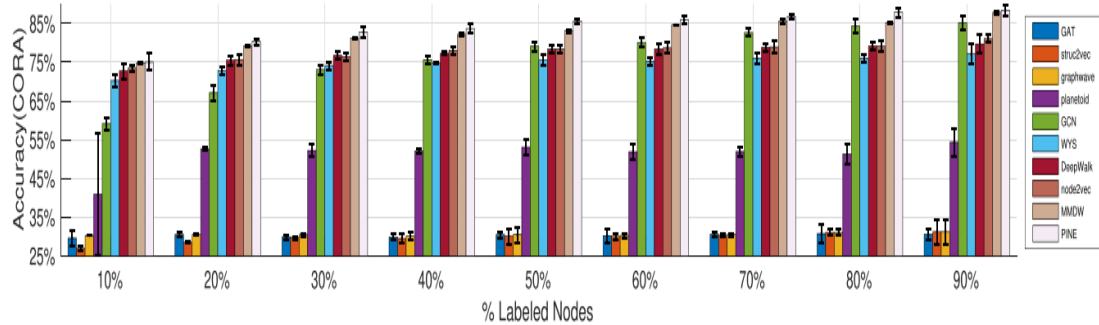
which requires $h(\cdot)$ and $g(\cdot)$ to ensure permutation invariant :

Then, PINE provides specific parameters for $h(\cdot)$ and $g(\cdot)$, which can be trained as follows:

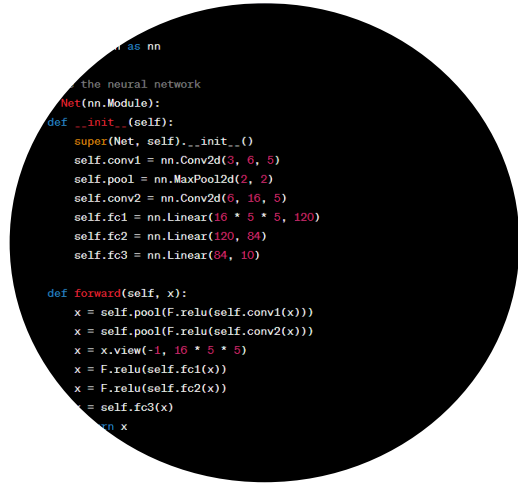
$$\mathbf{h}\left([z_1^\top, \dots, z_K^\top]^\top =: \bar{\mathbf{z}} \mid c, W, b\right) = c^\top \sigma(W\bar{\mathbf{z}} + b) \quad \mathbf{g}\left(\mathbf{x} \mid T, u, \{a_t, b'_t\}_{t=1}^T\right) = \begin{bmatrix} \sigma((u \otimes a_1)\mathbf{x} + b'_1) \\ \sigma((u \otimes a_2)\mathbf{x} + b'_2) \\ \vdots \\ \sigma((u \otimes a_T)\mathbf{x} + b'_T) \end{bmatrix}$$

Evaluation

Accuracy (%) of multi-class classification in homogeneous and heterogeneous graphs



How to succeed in this course?



Practice



Explore



Visualize

Discuss



Ask

