Deep Learning Fundamentals

Artificial Intelligence for Digital Health (AID)

M.Sc. in Digital Health – University of Pisa

Davide Bacciu (davide.bacciu@unipi.it)

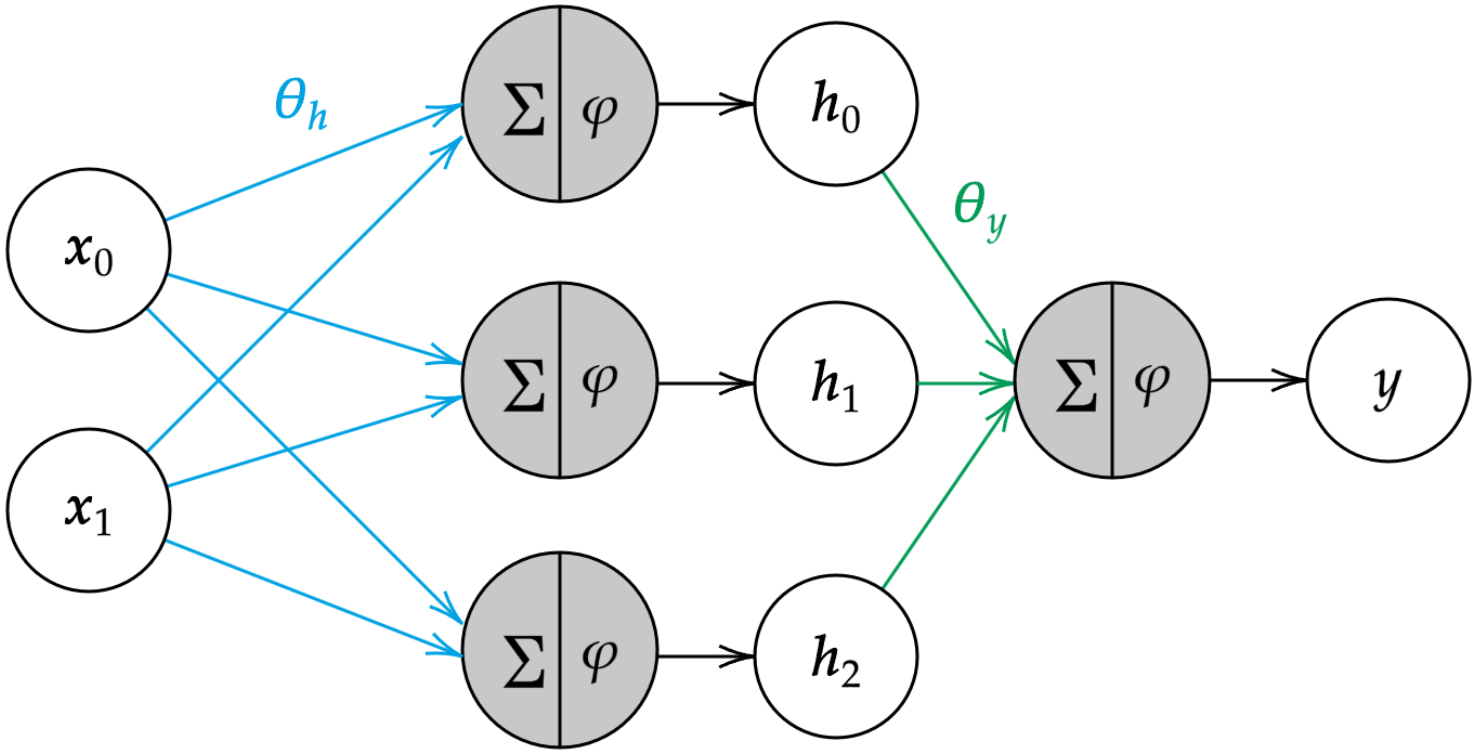
Lecture Outline

- From shallow to deep networks
- Enabling factors (tricks and else) for deep learning
  - Gradient issues
  - Activation functions
  - Normalization and regularization
  - Optimization
- Neural Autoencoders
  - The first deep neural network
  - Unsupervised learning with deep neural networks
  - AE tasks: anomaly detection, compression, denoising

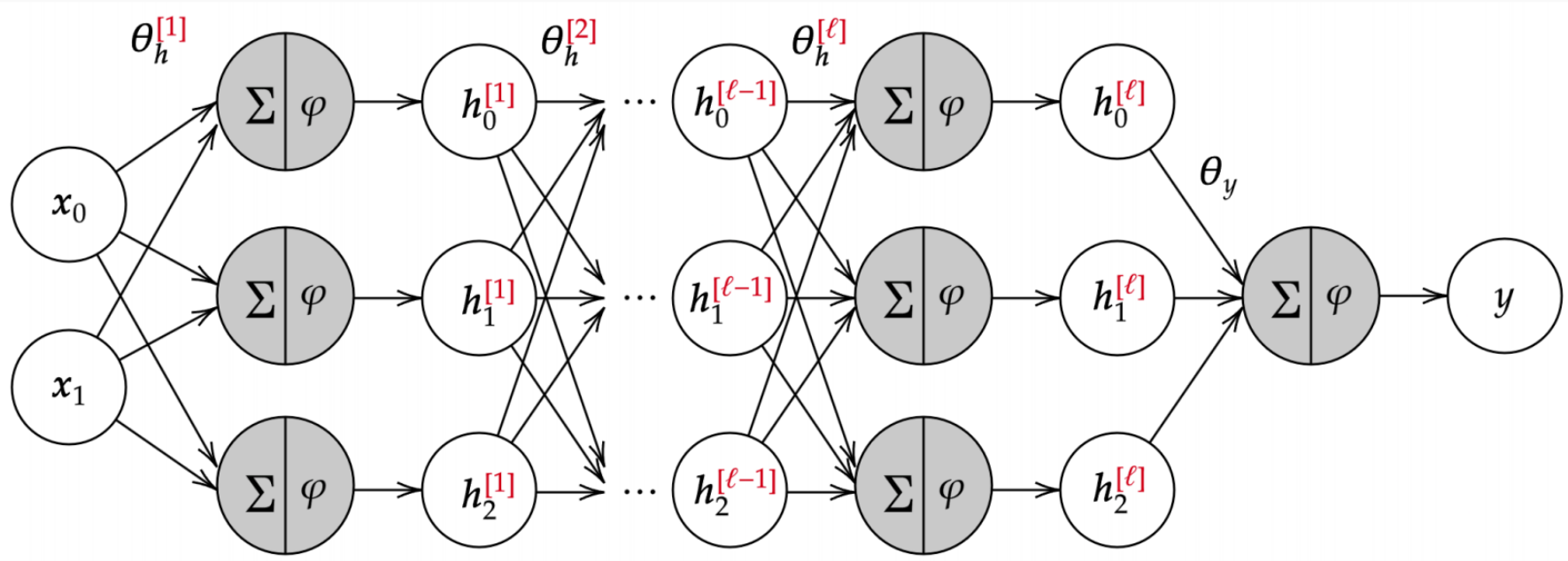
From the shallows to the deep

MLP – The shallow network

From some lectures ago..



Going deeper



- Apparently scaling up hidden layers to 2+
- Clearly much more than this

Deep neural network (DNN) computation

Transform input x through a composition of learnable non-linear functions

$$DNN_\Theta(\boldsymbol{x}) = g_{\theta^y}\left(g_{\theta^{h_l}}\left(g_{\theta^{h_{l-1}}}\left(\dots\left(g_{\theta^{h_1}}(\boldsymbol{x})\right)\right)\right)\right)$$

where

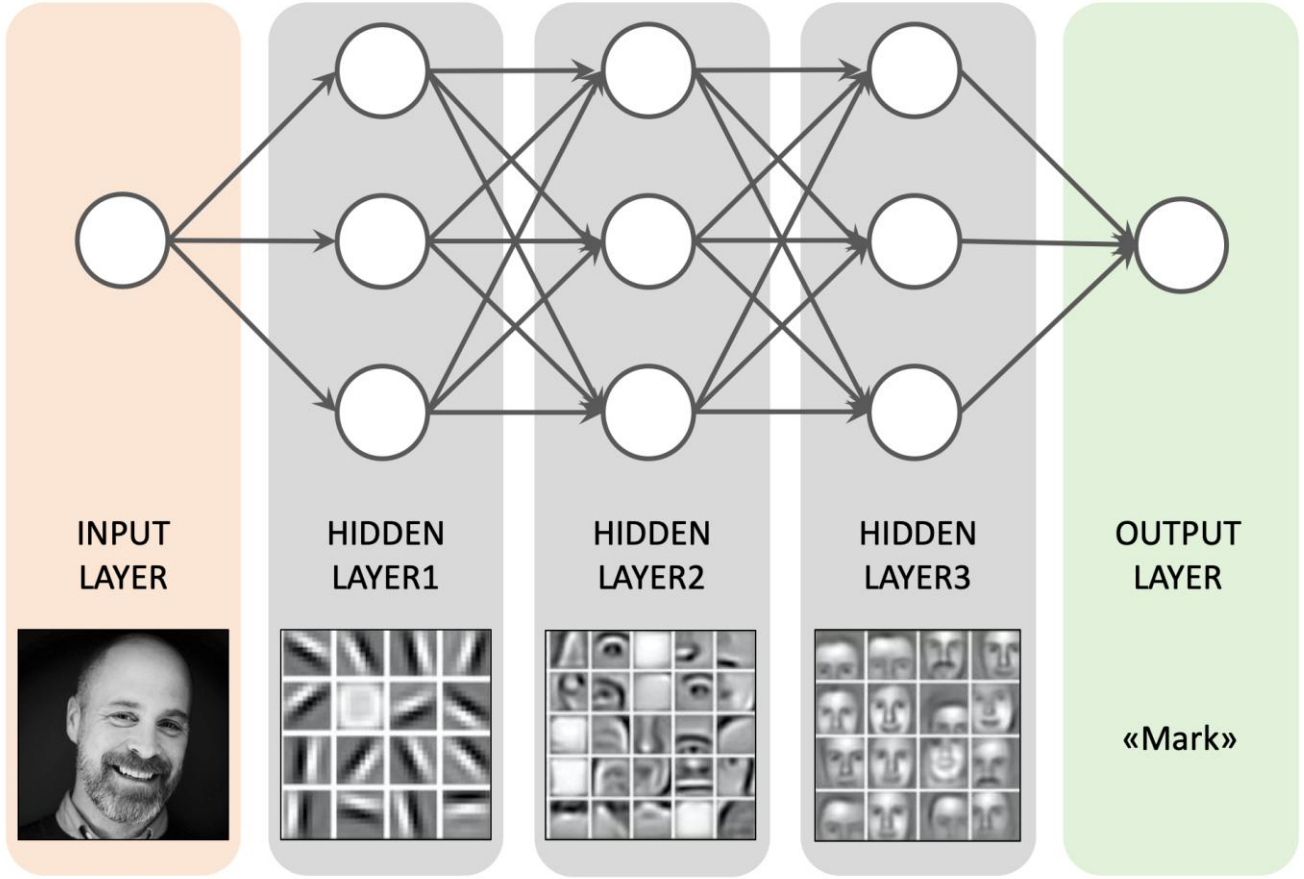
- $\boldsymbol{h}_1 = g_{\theta^{h_1}}(\boldsymbol{x}) = \varphi_1\left(\boldsymbol{\theta}^{h_1}\boldsymbol{x}\right)$
- $\boldsymbol{h}_2 = g_{\theta^{h_2}}(\boldsymbol{h}_1) = \varphi_2\left(\boldsymbol{\theta}^{h_2}\boldsymbol{h}_1\right)$
- …
- $\boldsymbol{h}_l = g_{\theta^{h_l}}(\boldsymbol{h}_{l-1}) = \varphi_l\left(\boldsymbol{\theta}^{h_l}\boldsymbol{h}_{l-1}\right)$
- $y = g_{\theta^y}(\boldsymbol{h}_l) = \varphi_y(\boldsymbol{\theta}^y\boldsymbol{h}_l)$

A set of learnable parameter matrices $\Theta = \{\boldsymbol{\theta}^{h_1}, \dots, \boldsymbol{\theta}^{h_l}, \boldsymbol{\theta}^y\}$ and layer-specific activation functions φ_l

Deep Learning: Learning Hierarchical Representations

- Early layers → Learn simple features (e.g., edges in images, basic word embeddings in text)
- Deeper layers → Learn complex features by combining simpler ones (e.g., shapes, object parts in images; syntax/semantics in text)
- Final layers → Combine high-level features for task-specific predictions
- This is especially useful in computer vision and natural language processing, where hierarchical feature extraction is critical

Hierarchical Neural Representations



- Deep learning success builds on the assumption that input is compositional
- Under such conditions going deeper is better than going wider

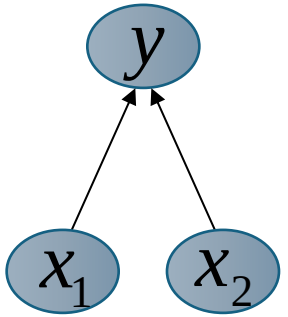
Efficient learning with depth (vs. width)

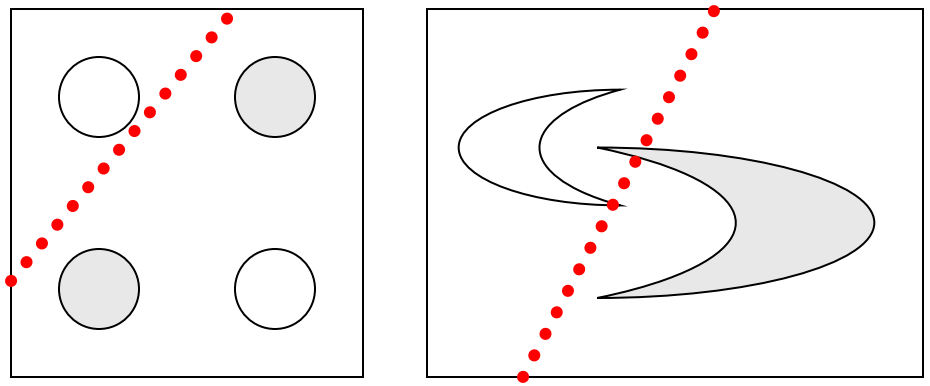
- There are theorems that prove that single-hidden-layer MLPs are universal approximators (with possibly infinite neurons)
- In practice, instead of making networks wider (more neurons per layer) is it best to go deeper (more layers)
  - Better feature abstraction → Each layer refines representations learned by previous layers
  - More efficient learning → Deep networks can represent complex functions more compactly
  - Improved generalization → Deep architectures capture meaningful patterns with fewer parameters than a shallow, wide network

Effect of Number of Layers

N hidden layers = 0 → Perceptron

- Learning separating hyperplanes



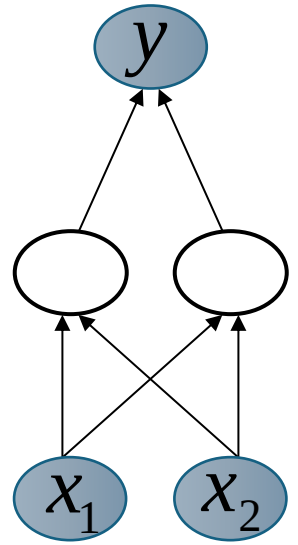


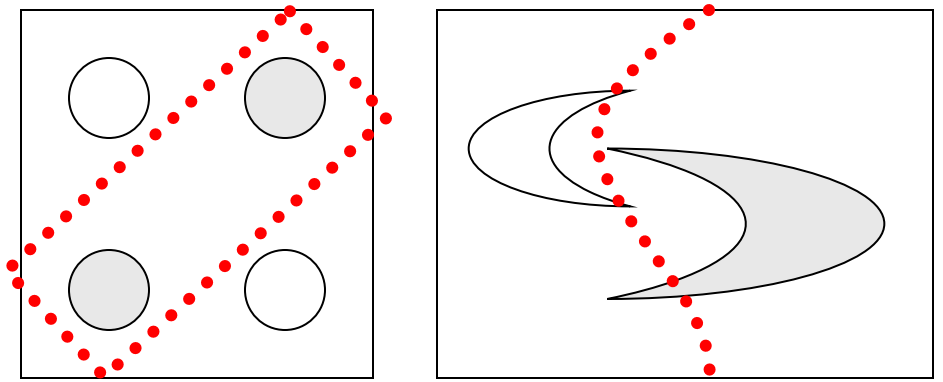
Example from to Eric Postma via Jason Eisner

Effect of Number of Layers

N hidden layers = 1 → Multilayer Perceptron

- Learning convex region boundaries (open or closed)



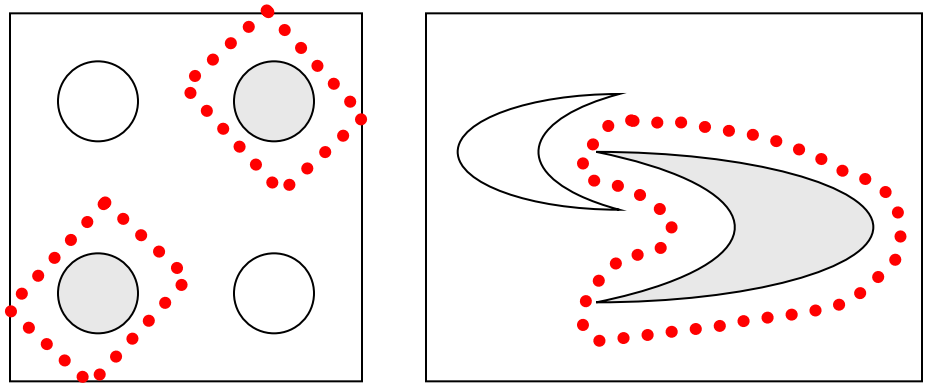


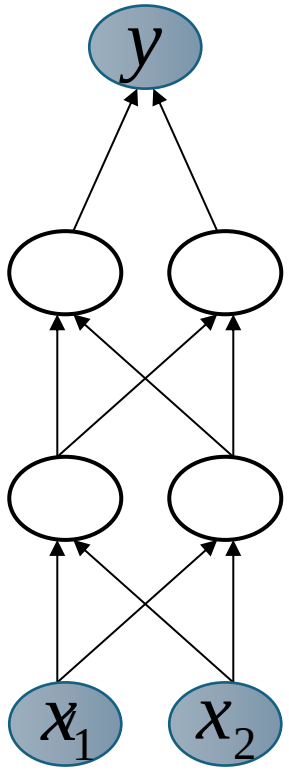
Example from to Eric Postma via Jason Eisner

Effect of Number of Layers

N hidden layers = 2 → Deep learning (sort-of)

- Learning to combine convex regions





Example from to Eric Postma via Jason Eisner

Challenges of deep learning: the role of depth (l)

- Model selection becomes expensive
  - A new bunch of hyperparameters: depth l, size of each hidden layer, …
- Overfitting becomes a taunting problem
  - Deep networks can easily memorize training data, leading to poor generalization
- Requires massive amounts of data
  - Hundreds of thousands of data: challenging for labelling and in general in the healthcare domain
- Limited usefulness if input is not compositional
  - Deep architectures may not provide much benefit over simpler models
- Inherent computational and numerical challenges
  - Deep models require large memory and powerful computing
  - Training can be unstable and slow to converge (gradient and optimization issues)

Tackling Deep Learning Challenges

Training of deep networks

- We can keep working with the stochastic gradient descent principle and the backpropagation algorithm
- Principled and smooth generalization with respect to increasing scale and model complexity
- But…

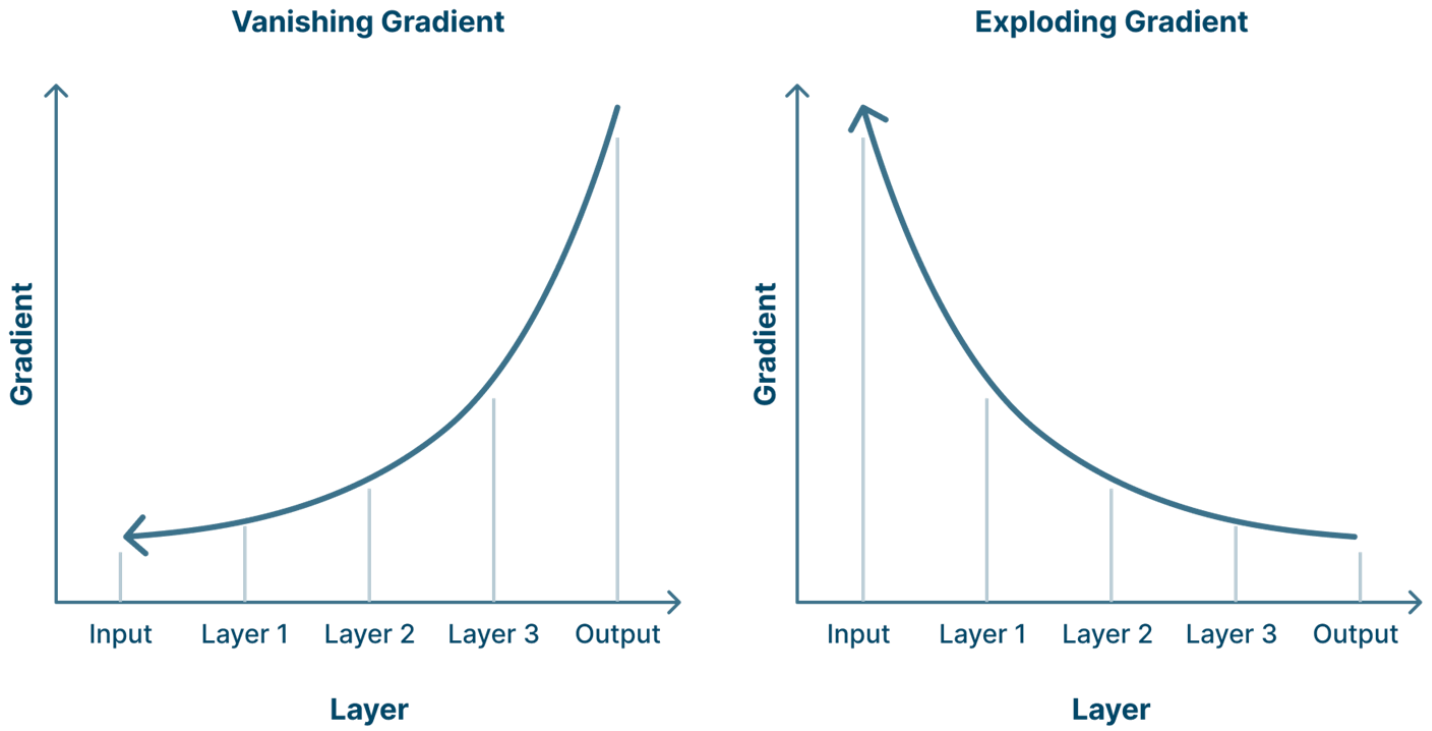
Dumbly applying backprop training does not work with deep networks → learning becomes difficult and training unstable

More complex networks may result in more articulated loss landscapes to be optimized

The original sin

The gradient vanish/explosion problem

Gradient (norm of the gradient) < 1



Gradient (norm of the gradient) > 1

A numerical problem rooted in the long chain of multiplications needed when backpropagating gradients through many layers

In the following

- A set of tips and tricks essential to smoothen and facilitate training of deep neural networks
  - Weight initialization
  - Activation functions
  - Activation normalization
  - Regularization
  - Residual connections
  - Optimization algorithms
  - Gradient clipping
- They often have deep theoretical motivations behind but for your mental safety we will refrain from deepening such motivations

Weight initialization

- Choice of initial weight values is important as this decides starting position in weight space
- Proper weight initialization is crucial to ensure stable learning, faster convergence, and improved accuracy in deep neural networks
- Guiding principles
  - Select weight values which produce midrange function signals
  - Select weight values randomly to break symmetries
  - Normalise weight values w.r.t. number of weighted connections per unit
- Try different random initialization to
  - Assess robustness
  - Have more opportunities to find optimal results

Weight initialization strategies

Random Initialization

- Breaks symmetry, ensuring different neurons learn distinct features
- Scaling strategies like sampling from small-variance Gaussian/uniform distributions are used to prevent large initial values from dominating training

Glorot (Xavier) Initialization

- Maintains balanced weight distribution across layers, smoothing gradient flow for efficient backpropagation
- Designed for sigmoid and tanh activations
- Weights are drawn from either distributions

$$U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}},\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right) \text{ or } \mathcal{N}\left(0,\frac{2}{n_{in}+n_{out}}\right)$$

with $n_{in}+n_{out}$ number of input and output units to a layer

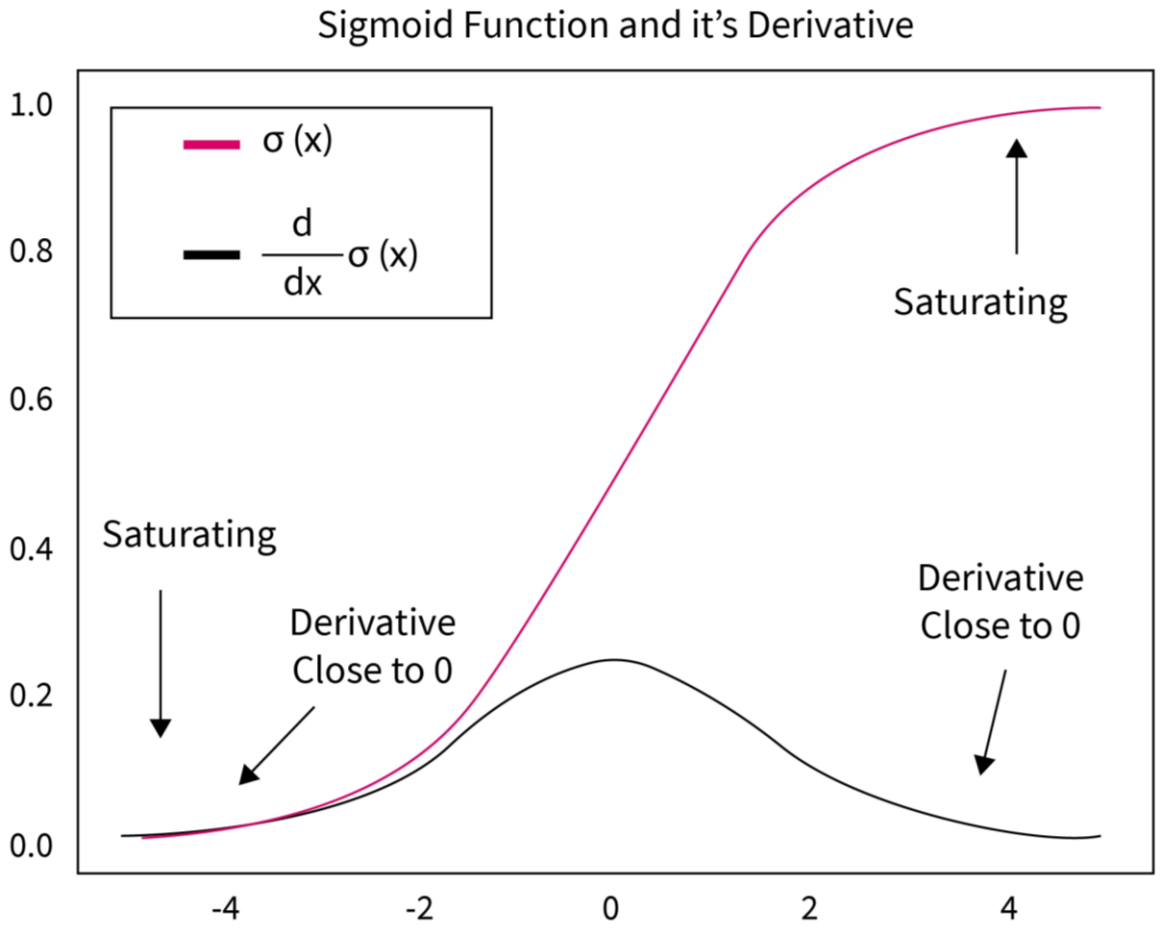
He Initialization

- Designed for for activation functions that do not have zero-centered mean (like ReLU)
- Weights are drawn from either distributions

$$U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}}},\frac{\sqrt{6}}{\sqrt{n_{in}}}\right) \text{ or } \mathcal{N}\left(0,\frac{2}{n_{in}}\right)$$

Gradient issues: on the role of activation functions

- The derivative of the activation functions play a major role in determining gradient magnitude
- Non-linear saturating activation functions (sigmoidal-like) have bad derivatives



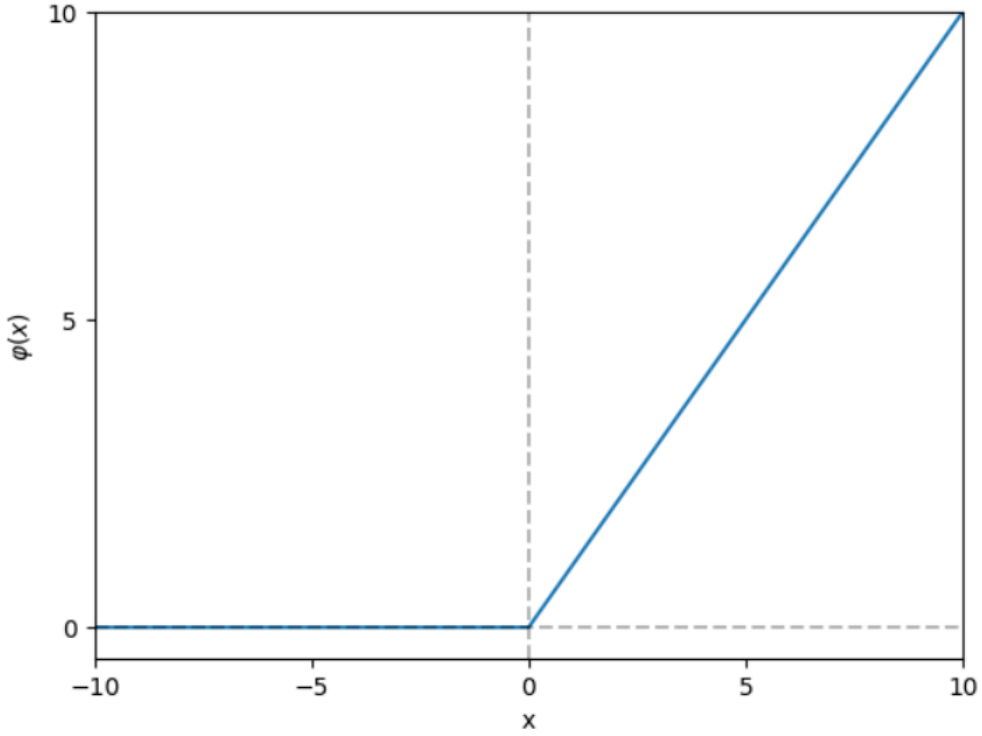
The Rectified Linear Unit (ReLU)

ReLU was instrumental to deep learning success

- Helps mitigating the vanishing gradient problem (doesn't solve it entirely!)
- Maintains (some degree of) non-linearity
- It can be efficiently computed (by GPUs)

$$ReLU(x) = \begin{cases} x, if\ x > 0 \\ 0, if\ x \leq 0 \end{cases}$$

$$\frac{dReLU(x)}{dx} = \begin{cases} 1, if\ x > 0 \\ 0, if\ x \leq 0 \end{cases}$$



Careful: it does not work nicely with all neural architectures (e.g. recurrent NNs)

Many variations along the ReLU principle

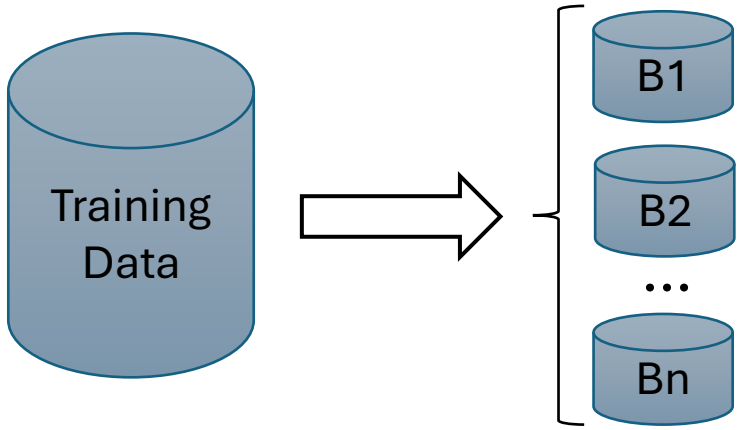
Activation Function	Formula	Fixes "Dying ReLU"?
Leaky ReLU	$\max(0.01x, x)$	✅ Yes (small slope for $x \leq 0$)
Parametric ReLU (PReLU)	$\max(\alpha x, x)$ (learns α)	✅ Learns best slope for negatives
Exponential Linear Unit (ELU)	x if $x > 0$, $\alpha(e^x - 1)$ if $x \leq 0$	✅ Smooth gradients for negatives
GELU (Gaussian Error Linear Unit)	$0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.0447x^3)))$	✅ Used in Transformers (better for NLP)
SELU (Scaled Exponential Linear Unit)	Scales output for **self-normalization**	✅ Used in Self-Normalizing Networks

If too many neurons get stuck at zero, they stop learning

Activation Normalization

- Activations are the neural representations of the input information in terms of the features learned by the hidden layers
  - If they have different scales, the model suffers from the same issues as unnormalized inputs!
- We may need activation normalization methodologies
  - Prevent vanishing/exploding gradients → Stabilize training
  - Enable faster learning → Allows higher learning rates
  - Improve generalization → Reduces reliance on precise weight initializations

Batch Normalization (BN)



For a given mini-batch B, BN normalizes activations:

1. Compute mean and variance across the batch

$$\mu_B^l = \frac{1}{N_B}\sum_{i=1}^{N_B} h_i^l \quad \text{and} \quad \sigma_B^l = \frac{1}{N_B}\sum_{i=1}^{N_B}\left(h_i^l - \mu_B^l\right)^2$$

Activation of one neuron in layer l for minibatch sample i

2. Normalize activations

$$\widehat{h_i^l} = \frac{h_i^l - \mu_B^l}{\sqrt{\left(\sigma_B^l\right)^2 + \epsilon}}$$

Need to backpropagate through these

3. Scale and shift through learnable parameters (γ, β)

$$\widehat{h_i^{l\prime}} = \gamma\widehat{h_i^l} + \beta$$

Becomes the input to layer $l+1$

- Reduces internal covariate shift (changing distribution of activations)
- Improves convergence → Training is faster and more stable

Layer Normalization (LN)

- Instead of normalizing over a batch, LN normalizes across features for each sample

1. Compute single mean and variance across the layer

$$\mu^l = \frac{1}{N_l}\sum_{j=1}^{N_l} h_j^l \quad \text{and} \quad \sigma^l = \frac{1}{N_l}\sum_{j=1}^{N_l}\left(h_j^l - \mu^l\right)^2$$

2. Normalize activations

$$\widehat{h_i^l} = \frac{h_i^l - \mu^l}{\sqrt{\left(\sigma^l\right)^2 + \epsilon}}$$

3. Scale and shift through learnable parameters (γ, β)

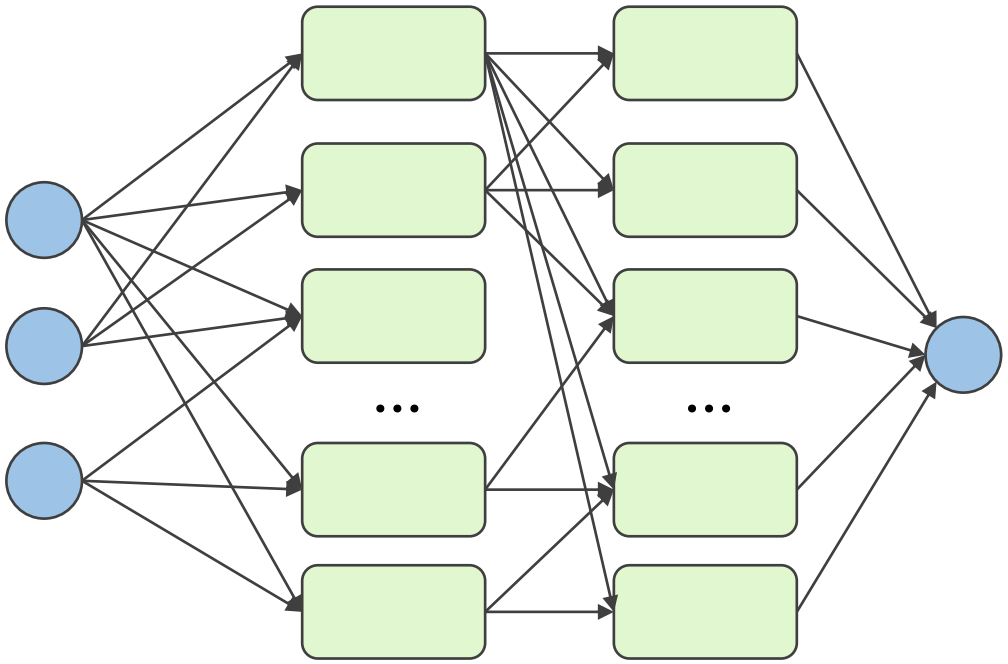
$$\widehat{h_i^{l\prime}} = \gamma \widehat{h_i^l} + \beta$$

- Used in Transformers and RNNs where batch statistics (on sequences) are less robust

Regularization

- More layers → More parameters → Higher risk of overfitting
- We already have a bunch of regularization techniques in our toolbox that use penalties on weight vector norms (L1, L2, …, combined)
- Here we focus on a different approach to regularization introduced specifically for deep learning: dropout regularization
  - Key idea: randomly disconnect units from the network during training

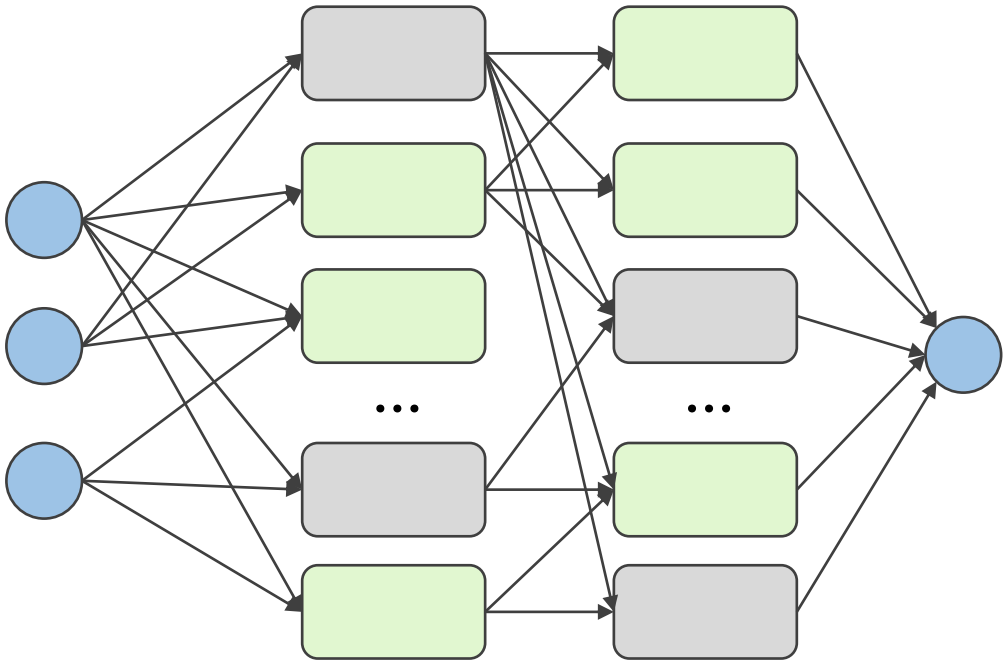
Dropout regularization



During training:

1. Randomly "drop" (set output to 0) some neurons in the hidden layers
2. This reduces the number of active neurons, effectively creating a smaller network
3. Each batch sees a different sub-network, forcing the model to learn robust features

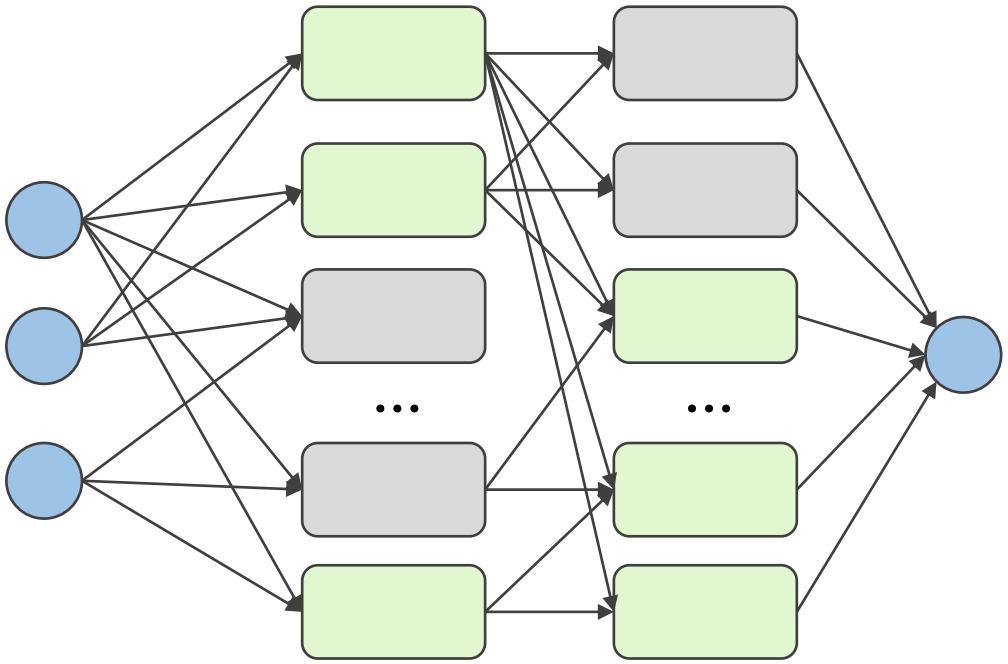
Dropout regularization



During training:

1. Randomly "drop" (set output to 0) some neurons in the hidden layers
2. This reduces the number of active neurons, effectively creating a smaller network
3. Each batch sees a different sub-network, forcing the model to learn robust features

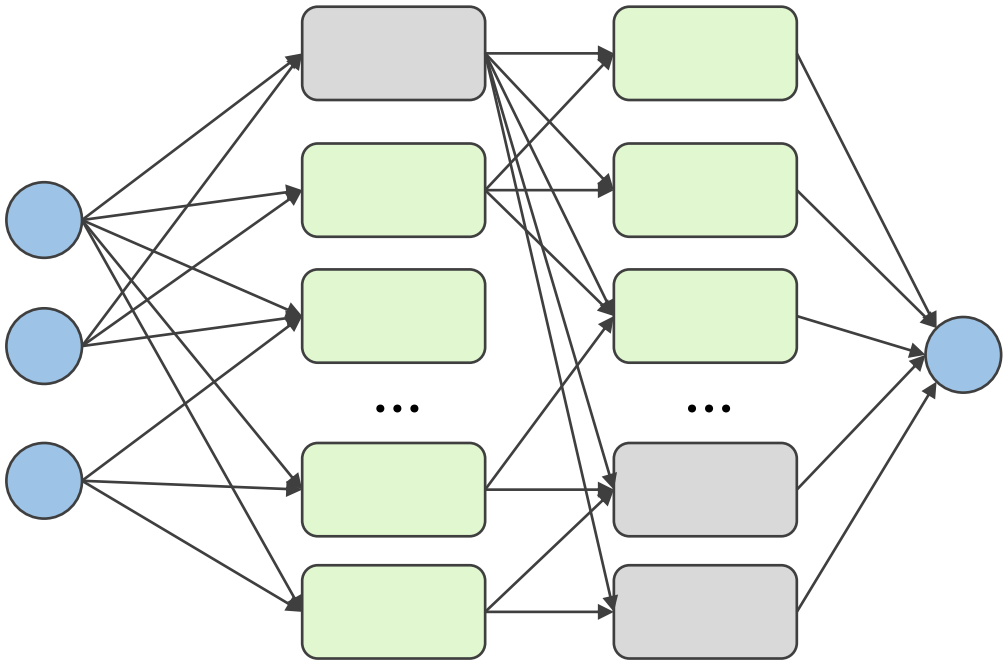
Dropout regularization



During training:

1. Randomly "drop" (set output to 0) some neurons in the hidden layers
2. This reduces the number of active neurons, effectively creating a smaller network
3. Each batch sees a different sub-network, forcing the model to learn robust features

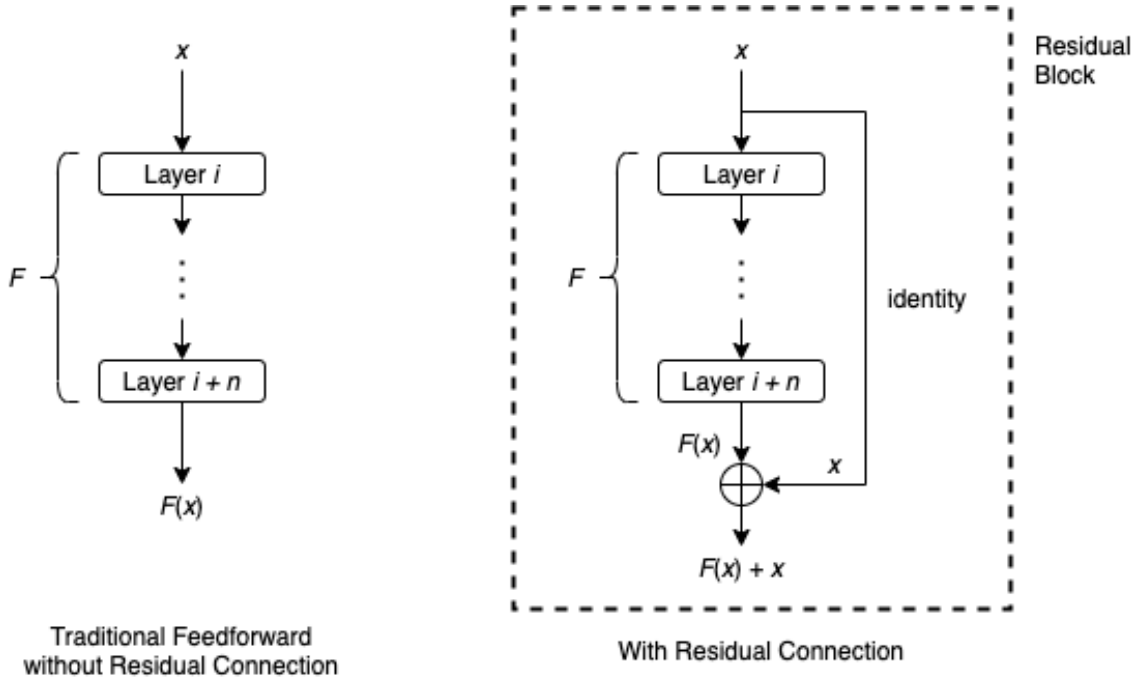
Dropout regularization



- Regulated by unit dropping hyperparameter p (Bernoulli)
- Need to adapt prediction phase
- You can also drop single connections (dropconnect)

Residual Connections

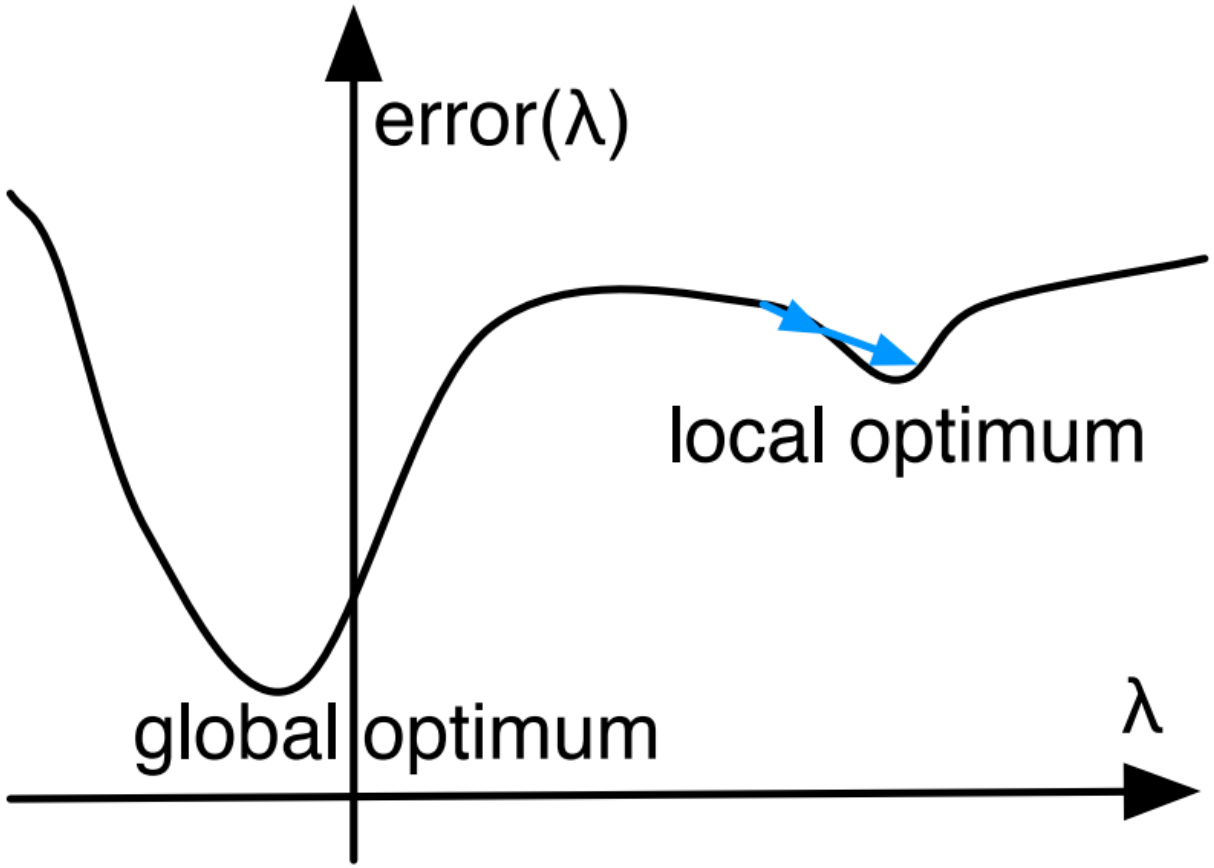
Instead of just passing information sequentially through layers, residual connections skip layers and directly connect an earlier layer's input to a later layer's output



The DNN learns the residual (difference) between layers, rather than the full transformation

- Gradients can "skip" layers and directly reach early layers
- Makes it easier for deeper networks to learn identity mappings

Cost functions are (unfortunately) more complex than simple convex functions



Local optimum

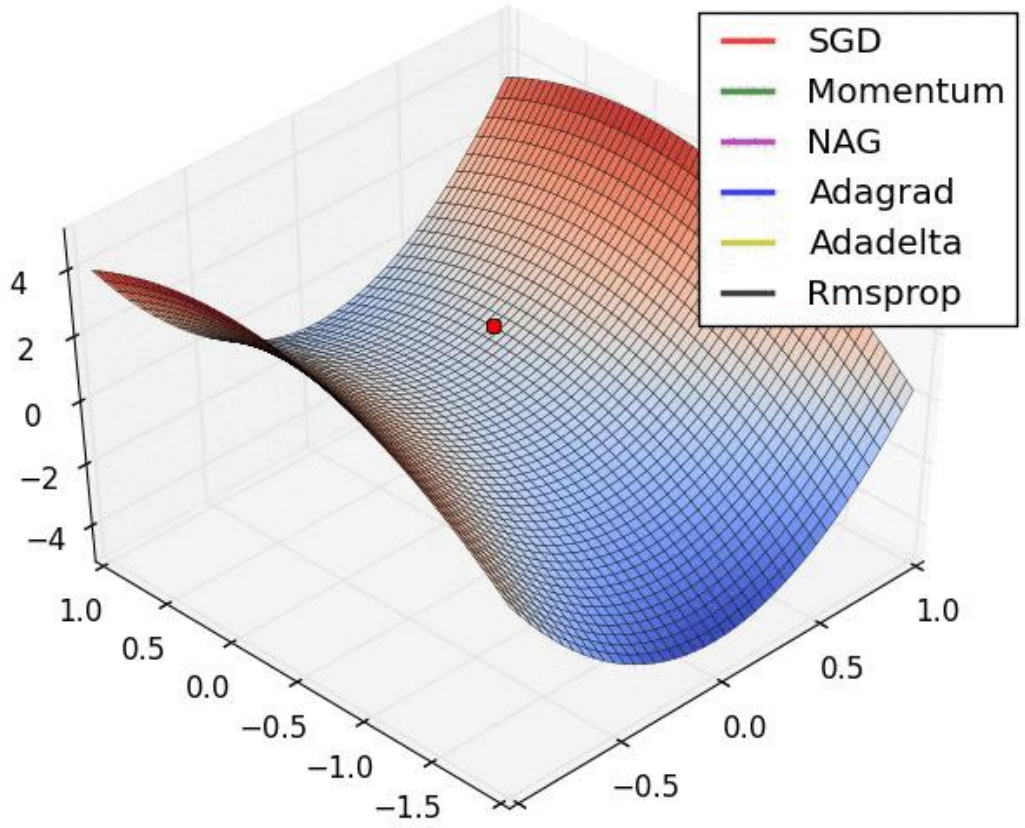
Optimizers in Deep Learning

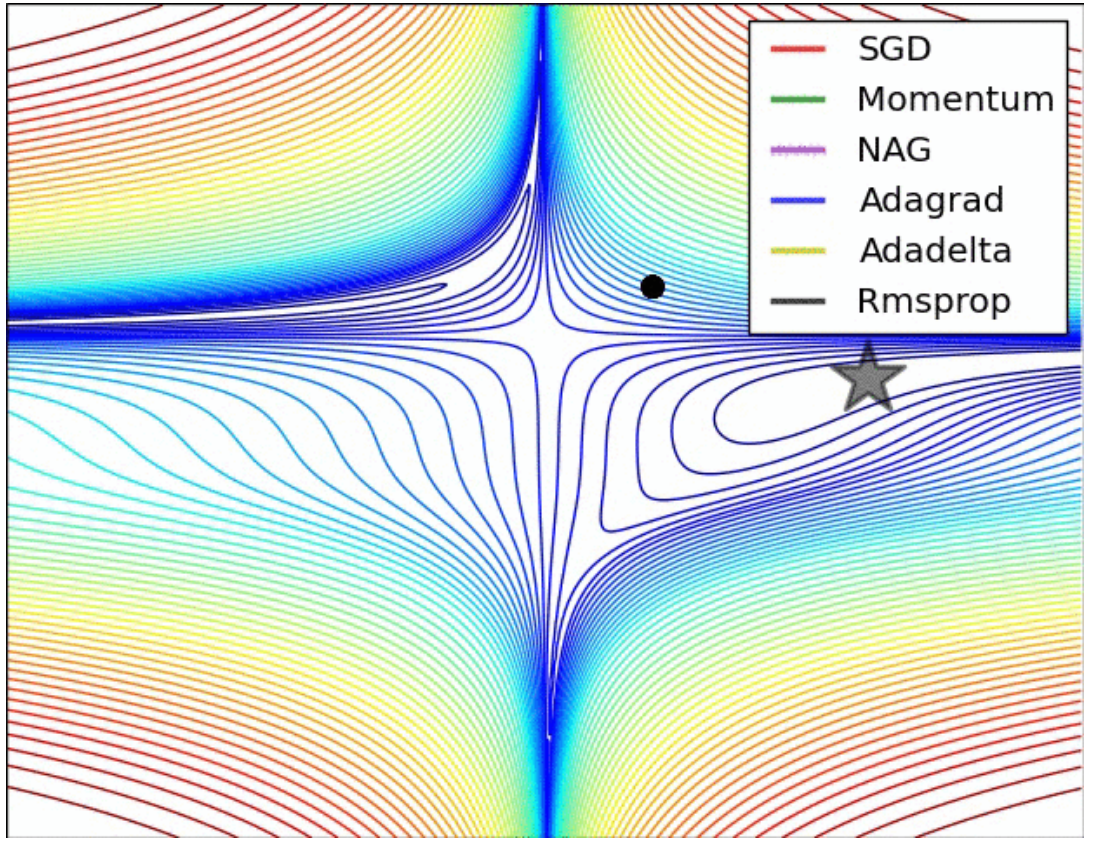
- Optimizer - A function that performs SGD to update model parameters efficiently
- Different optimizers improve SGD by modifying how gradients are used

Optimizer	Key Idea	Advantages	Disadvantages
SGD (Vanilla)	Basic gradient descent update	Simple & effective	Can get stuck in local minima, slow convergence, difficult to tune learning rate
Momentum	Adds "inertia" (exponentially weighted history of previous weights changes)	Faster convergence, escapes local minima	Can overshoot if momentum is too high
Adagrad	Adapts learning rate for each parameter	Good for sparse data (NLP)	Learning rate decays too fast, can slow down
RMSprop	Similar to Adagrad, but decays past gradients more smoothly	Works well in non-stationary environments	Requires tuning decay rate
Adam	Combines Momentum + RMSprop	Fast convergence, handles noisy gradients	Prone to overfitting in some cases

Adam is the current de-facto standard (but can depend on the task)

Optimization Algorithms

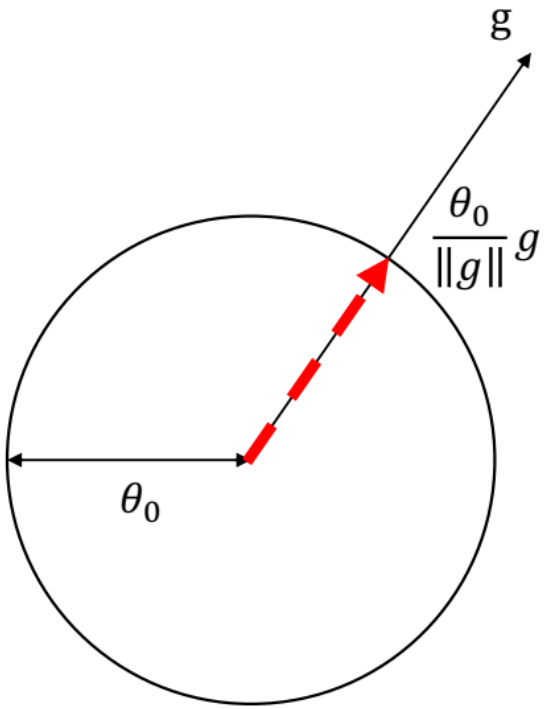




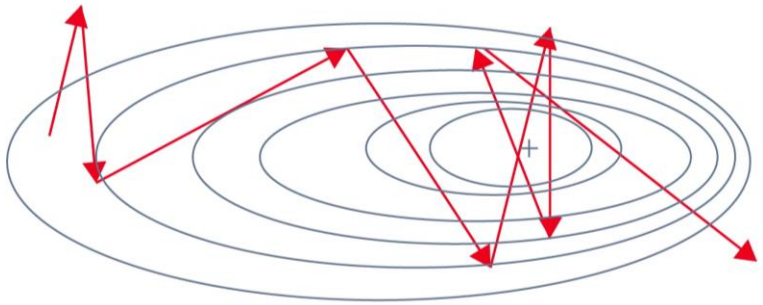
Gradient Clipping

Technique to contrast exploding gradients by limiting the gradient (magnitude) to a threshold θ_0

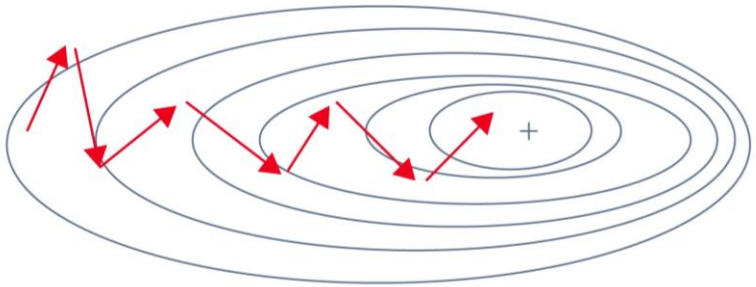
- If $\|g\| > \theta_0$ then $g = \frac{\theta_0}{\|g\|} g$



Without gradient clipping



With gradient clipping



Neural Autoencoders

Unsupervised Learning Tasks

- Neural autoencoders are an example of deep learning architecture for solving unsupervised learning tasks
- Outlier/anomaly detection: identify samples that deviate significantly from the rest of the dataset
  - Fraud detection, network security, medical diagnostics
- Compression/dimensionality reduction: reduce data complexity while preserving essential information
  - Data visualization, noise reduction, and efficient storage
- Data Generation: generate new samples similar to those in the dataset
  - Synthetic data generation: we will look into it, if we have time

Neural Autoencoder (AE)

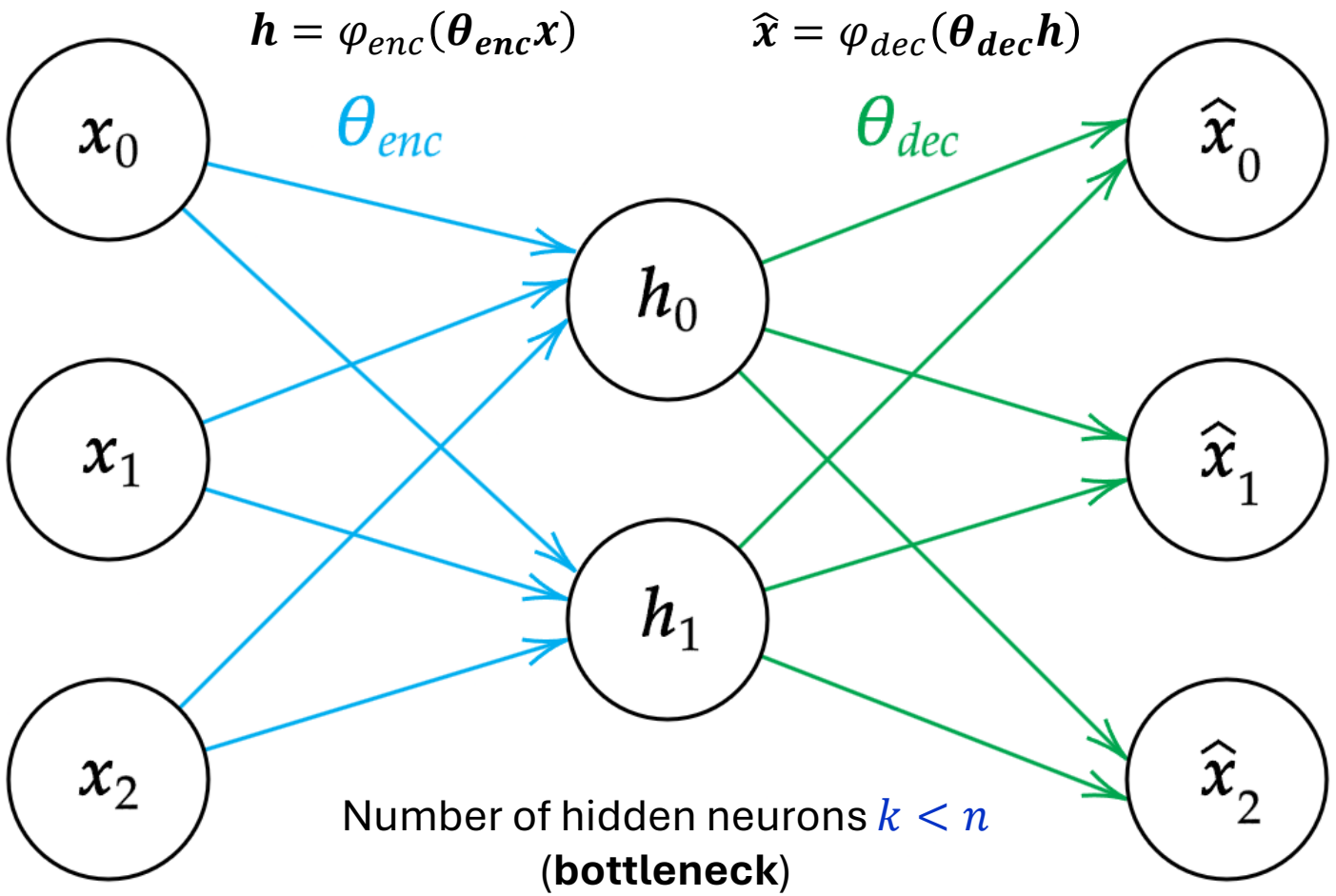
- Key idea: train a neural network that can reconstruct its input
- Key catch: to be useful, we will constrain the network to learn a compressed representation of input information

How to constrain the network?

- Bottleneck autoencoder - Make sure the hidden layer has fewer degrees of freedom (neurons) than the input
- Denoising autoencoder – Make the network reconstruct a clean information starting from an altered input

Bottleneck autoencoder

Encoder: codifies input $\boldsymbol{x}$ in a smaller neural representation $\boldsymbol{h}$



Decoder: produces a reconstruction $\tilde{\boldsymbol{x}}$ of the original input $\boldsymbol{x}$ in from its compressed representation $\boldsymbol{h}$

The number of input features is n = number of output neurons

Number of hidden neurons $k < n$ (**bottleneck**)

Bottleneck AE intuition

- Encoder: Compressing the Input
  - The encoder transforms the input **x** into a lower-dimensional representation **h** (the "bottleneck")
  - Goal: Preserve only the most useful input features (relevant factors of variation) needed for reconstruction
- Decoder: Reconstructing the Input
  - The decoder takes **h** and attempts to reconstruct **x** as closely as possible

$$MSE(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2$$

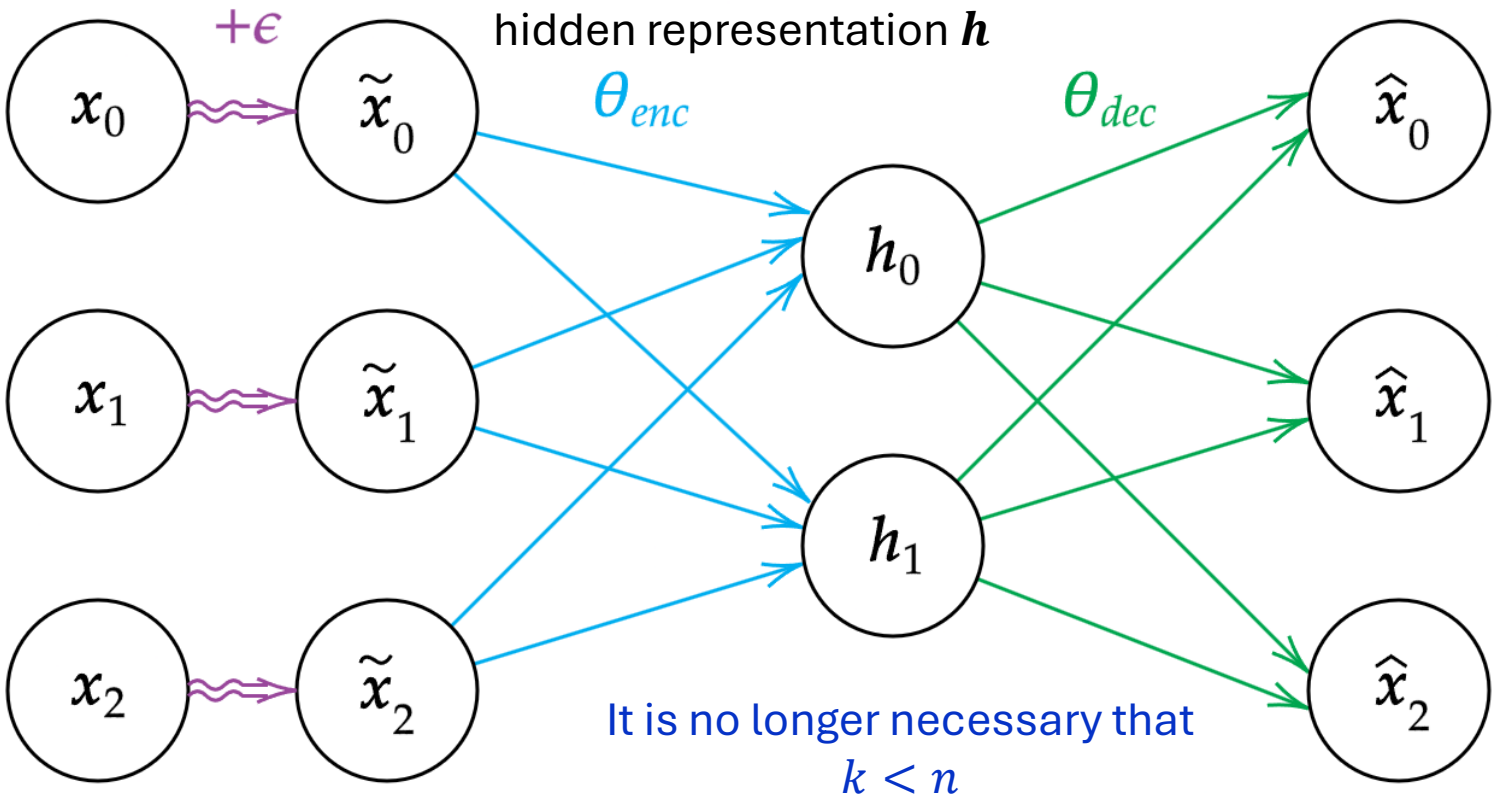
  - Reconstruction won't be perfect, since **h** contains less information than **x**, but it will also be non-trivial

Denoising autoencoder

The original input $\boldsymbol{x}$ is altered by addition of noise $\tilde{\boldsymbol{x}}$

The encoder transforms the corrupted input $\tilde{\boldsymbol{x}}$ into its hidden representation $\boldsymbol{h}$

The decoder reconstructs the original clean input $\boldsymbol{x}$ from the hidden representation $\boldsymbol{h}$

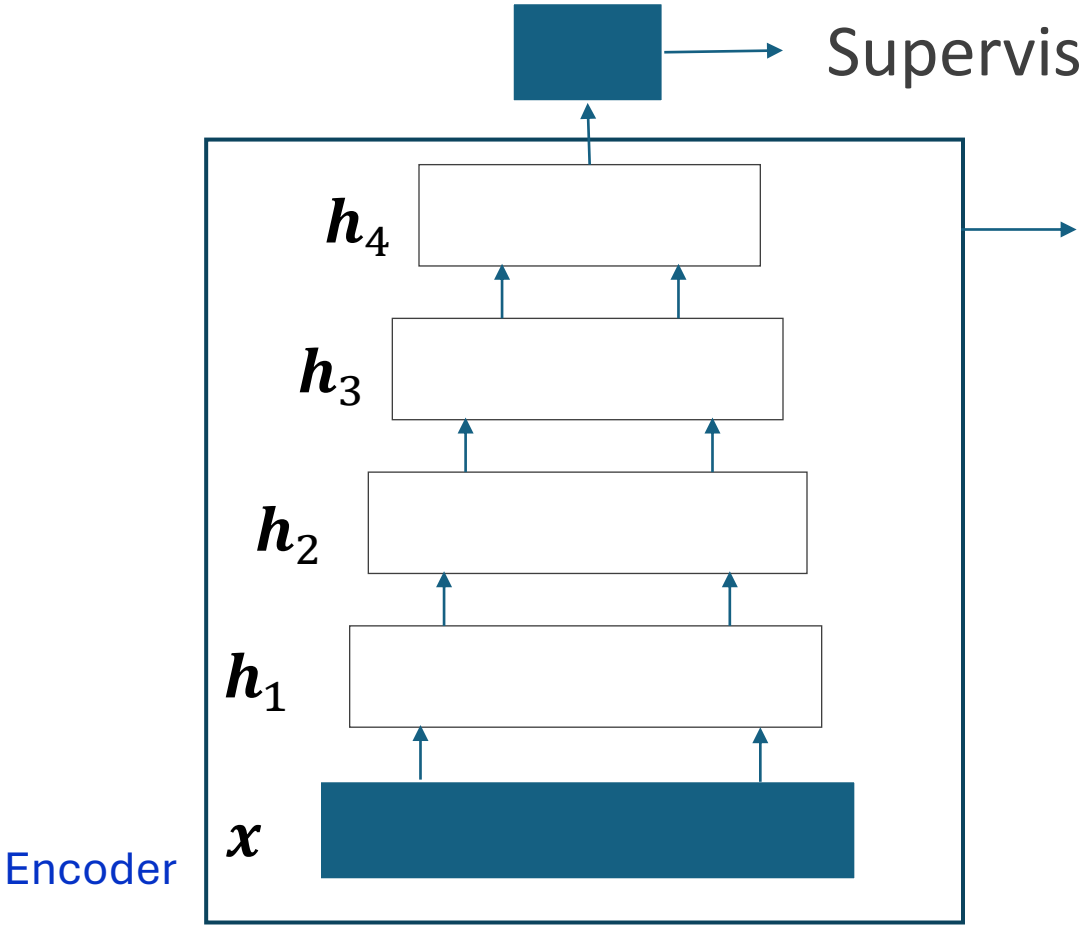


It is no longer necessary that $k < n$

Denoising AE intuition

- Noise: Corrupting the Input
  - The network never sees the clean input, but it must learn to reconstruct it (controls overfitting)
  - Mathematically, the noisy input is
    $$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon} \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2) \text{ with } \sigma^2 \text{ small}$$
  - Why? Forces the model to focus on essential patterns rather than memorizing details
- Encoder: Learning a Robust Representation
  - Extracts meaningful features to reconstruct the input while ignoring noise
- Decoder: Reconstructing the Original Input
  - The decoder tries to reconstruct the original input from its noisy version
  - The network learns to filter out noise
- The Denoising AE is trained by MSE minimization w.r.t. the clean input reconstruction

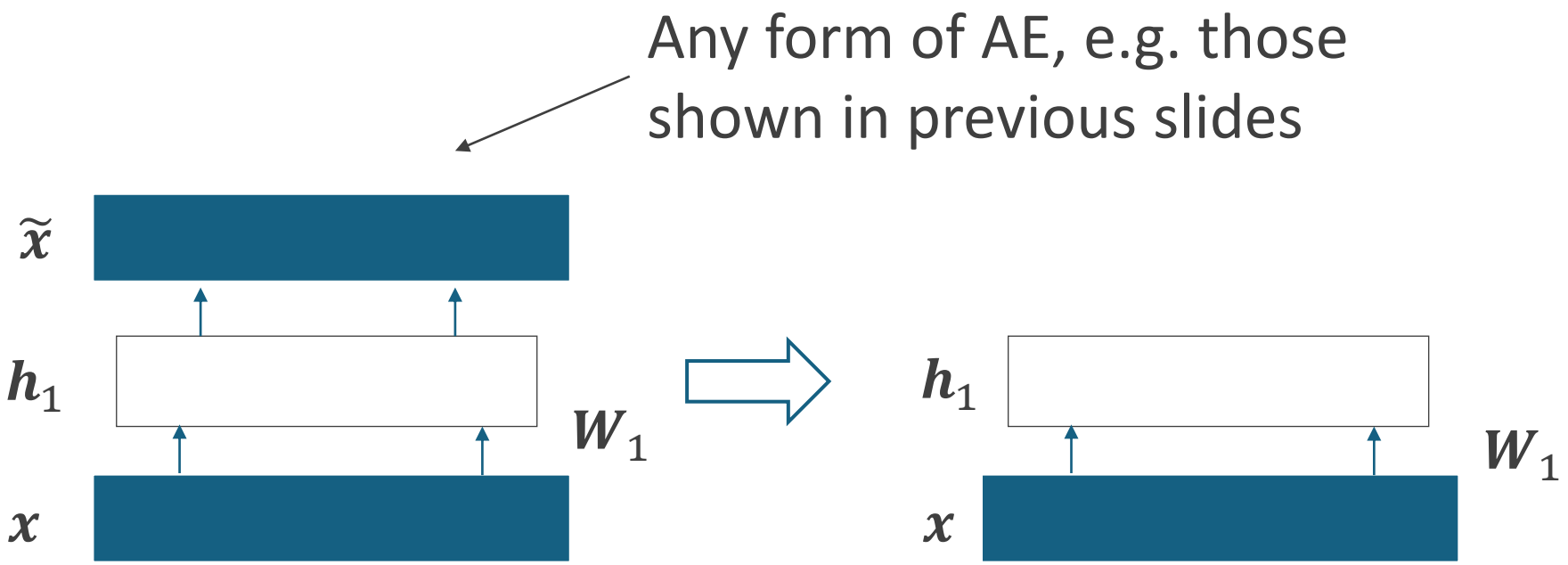
Deep Autoencoder (AE)



- Unsupervised training
- Hierarchical autoencoder
- Extracts a representation of inputs that facilitates
  - Data visualization, exploration, indexing,…
  - Realization of a supervised task

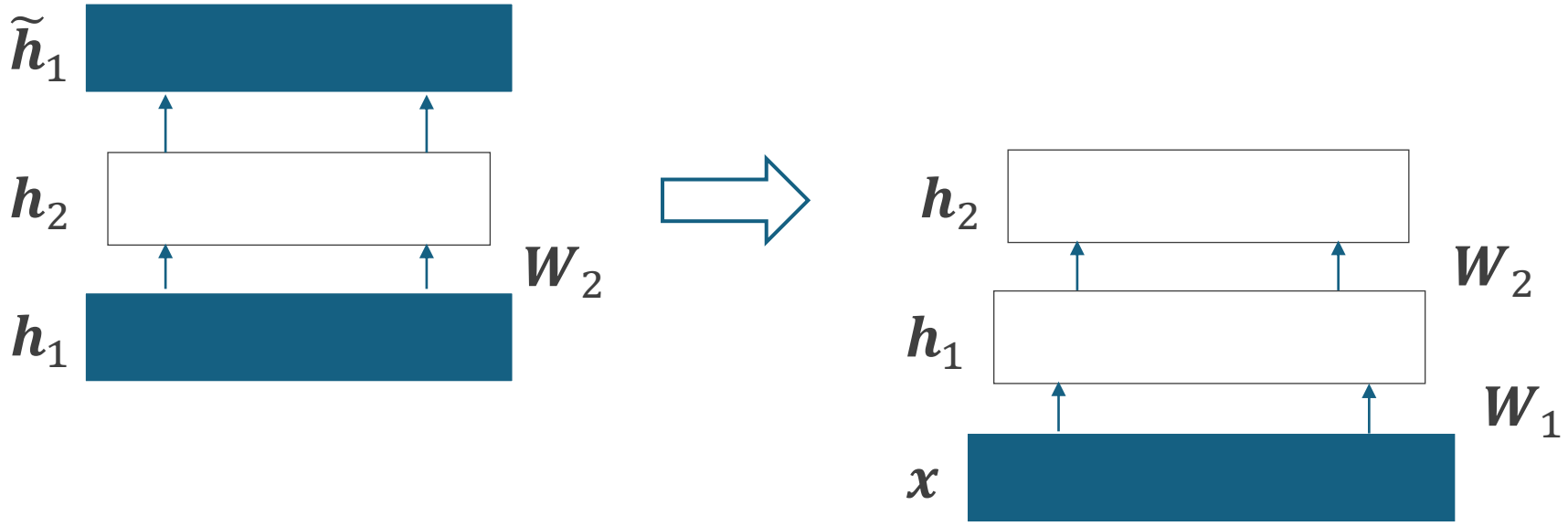
Unsupervised Layerwise Pretraining

Incremental unsupervised construction of the Deep AE



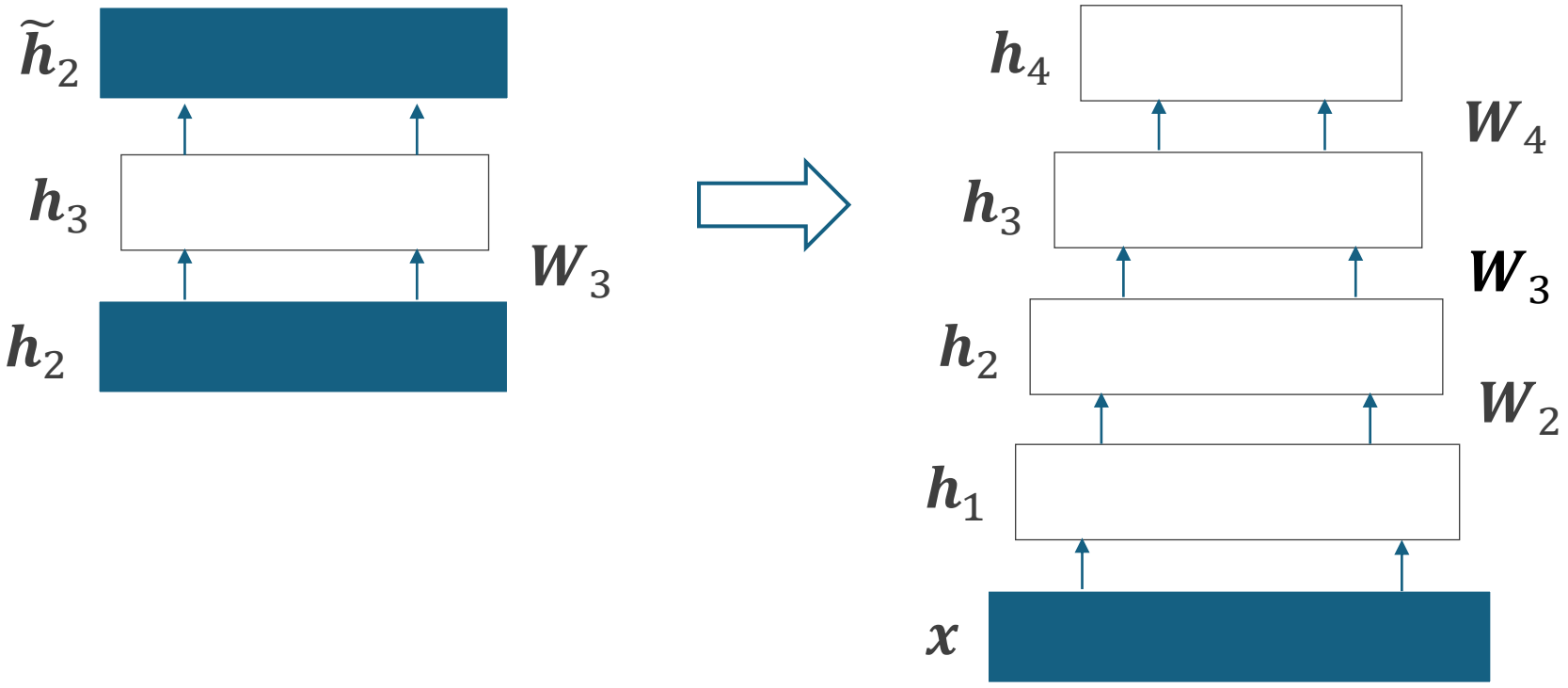
Unsupervised Layerwise Pretraining

Incremental unsupervised construction of the Deep AE



Unsupervised Layerwise Pretraining

Incremental unsupervised construction of the Deep AE



Autoencoders in use

- Anomaly/outlier detection
- Compression
- Denoising
- (and more.. such as generation)

Autoencoder for Anomaly/Outlier Detection

We want to determine if an input differs significantly from normal data

- Key idea: An autoencoder learns to reconstruct normal data, but struggles with anomalies
  - Normal inputs → The autoencoder reconstructs well (low reconstruction loss)
  - Anomalous inputs → The autoencoder fails to reconstruct (high reconstruction loss)

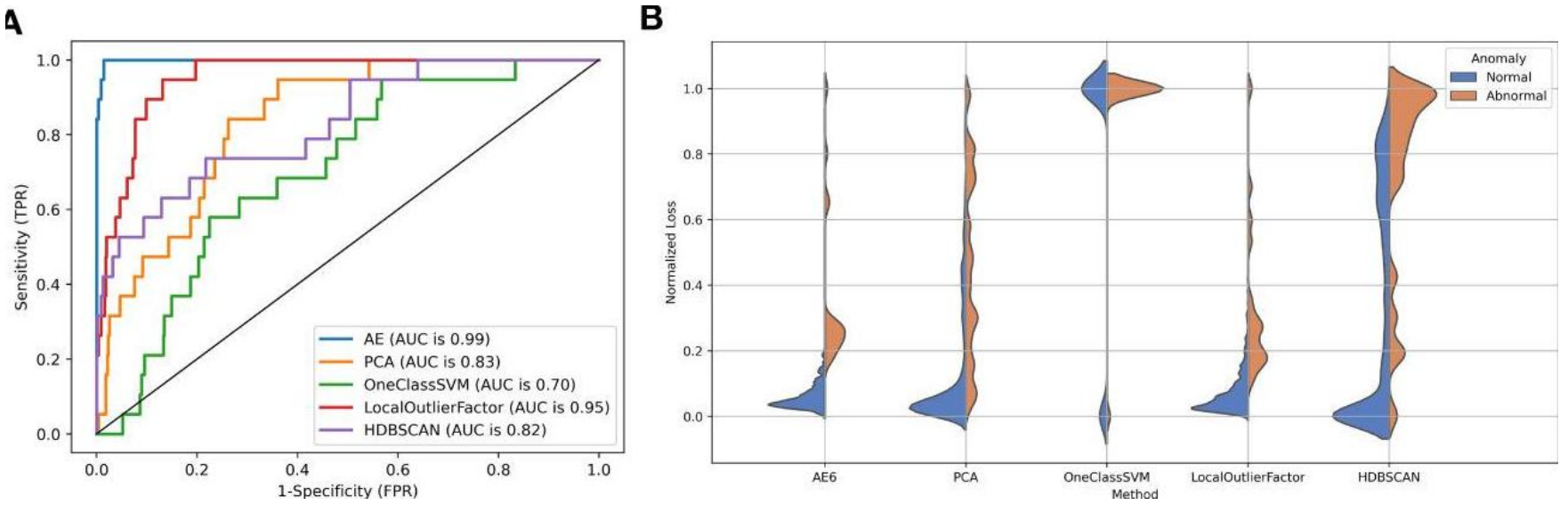
How to:

- **At training time**: Fit the autoencoder parameters using only normal data
- **At test time**: Given a new input, compute reconstruction error $\mathrm{L} = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2$
  - If $L \leq \tau$ → Input is normal
  - If $L > \tau$ → Input is anomalous
- Here τ is a threshold on reconstruction error
  - It can be computed using the average reconstruction error on ground truth normal data
  - It can also be made more robust by running a statistical test considering both average and std of normal data reconstruction

Anomaly Detection in Radiotherapy Plans

Radiotherapy plan represented by 30 input features

Features	Description	Fields	Number of features	Type	Unit
Segment	The number of segments of the field	IMRT	2	Integer	Number
SSD	Source to skin distance	IMRT/tangent	4	Float	cm
Collx1	Collimators' position in the x1 direction	IMRT/tangent	4	Float	cm
Collx2	Collimators' position in the x2 direction	IMRT/tangent	4	Float	cm
Colly1	Collimators' position in the y1 direction	IMRT/tangent	4	Float	cm
Colly2	Collimators' position in the y2 direction	IMRT/tangent	4	Float	cm
G_θ	The angle of the gantry	IMRT/tangent	4	Integer	Degree
Meterset	The MU per field	IMRT/tangent	4	Float	MU

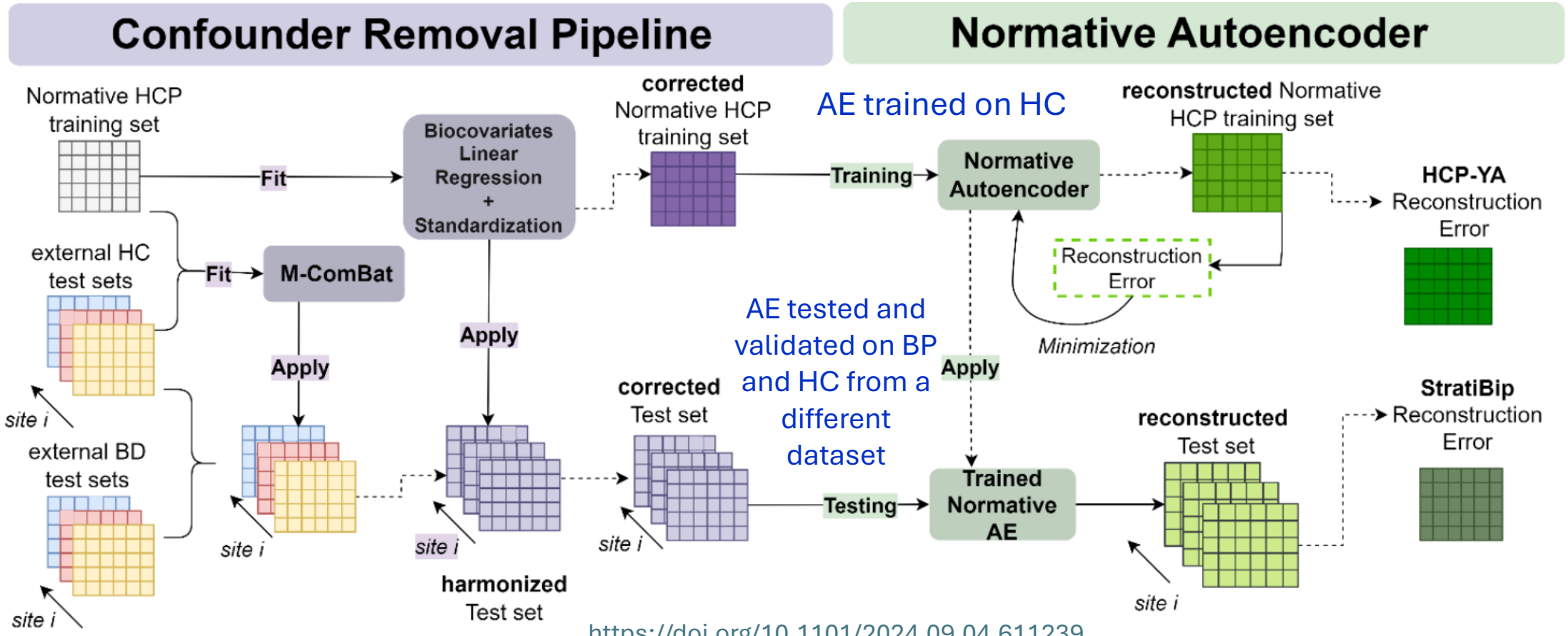


ROC curve and reconstruction error distribution confronting a deep AE with other anomaly detection methods

https://doi.org/10.3389/fonc.2023.1142947

Identifying anomalous brain morphology

Detecting morphological anomalies in Bipolar Disorder (BD) subjects as an anomaly detection task from Healthy Controls (HC)

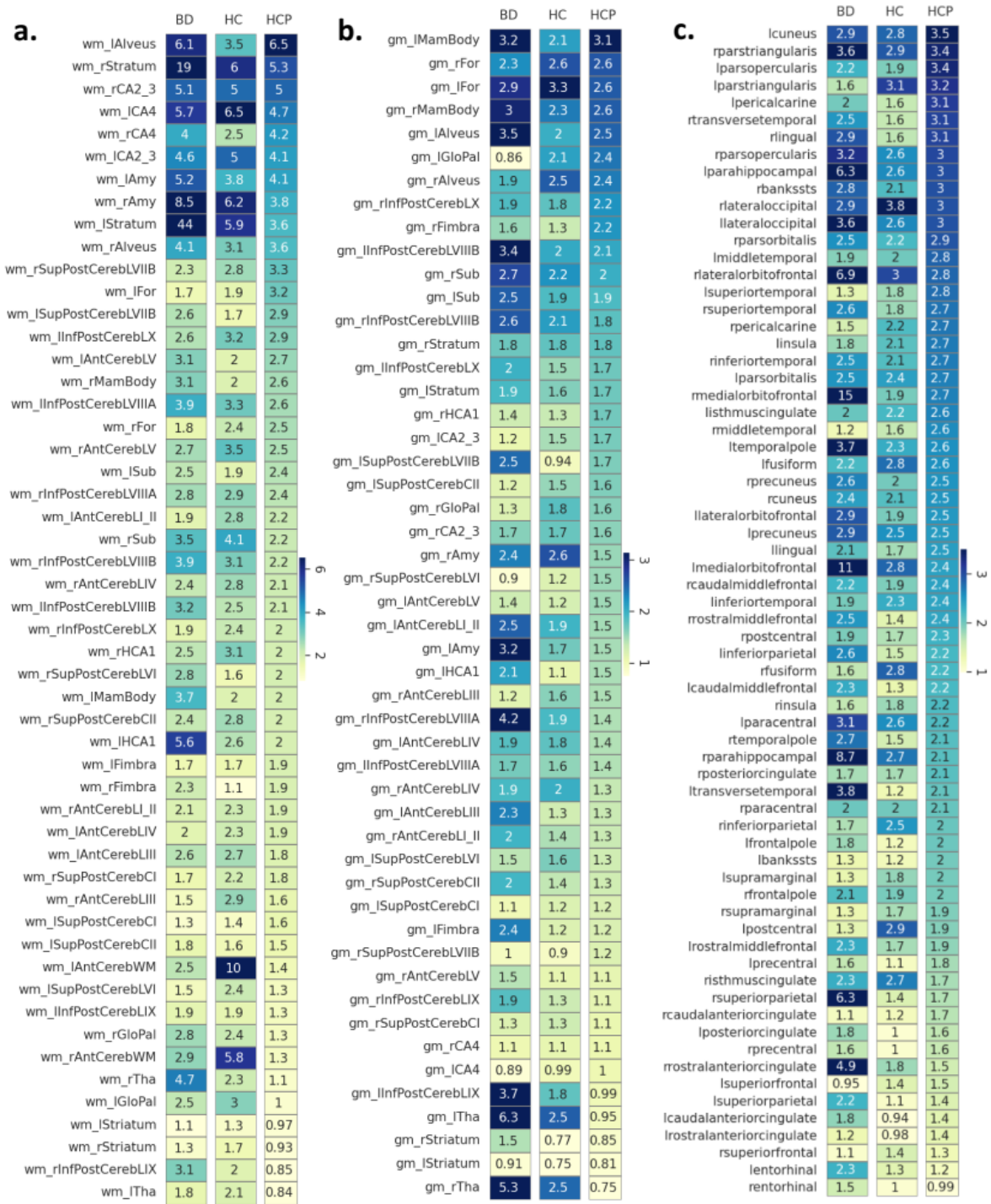


https://doi.org/10.1101/2024.09.04.611239

Identifying differentially anomalous features in BD Vs HC

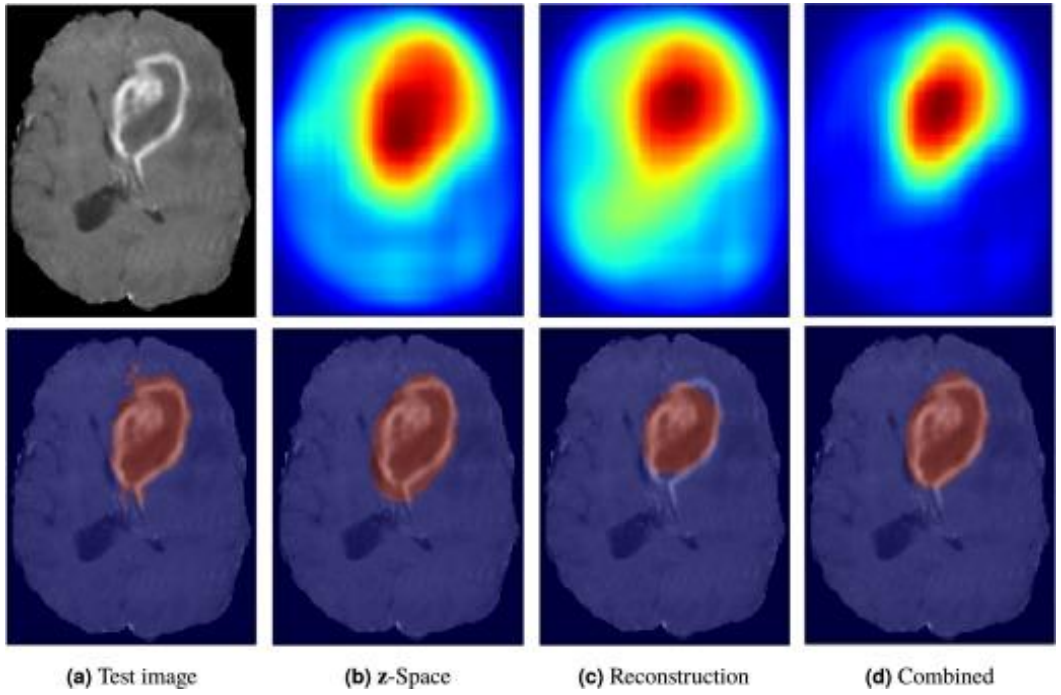
https://doi.org/10.1101/2024.09.04.611239

Features MEVs		BD-StratiBip	HC-StratiBip	HCP-YA
CT	Left parahippocampal gyrus	6.9	3	3
	Right parahippocampal gyrus	8.7	2.7	2.1
	Right lateral orbitofrontal	6.9	3	2.8
	Left medial orbitofrontal	11	2.8	2.4
	Right medial orbitofrontal	15	1.9	2.7
	Right superior parietal	6.3	1.4	1.7
	Right rostral anterior cingulate	4.9	1.8	1.5
GMV	Right Inferior posterior CerebLVIIIA	4.2	1.9	1.2
	Adjacent to left Fimbra	2.4	1.2	1.2
	Left Thalamus	6.3	2.5	1.2
	Right Thalamus	5.3	2.5	0.75
WMV	Right Stratum	19	6	5.3
	Left Stratum	44	5.9	3.6
	Left HCA1	5.6	2.6	2
	Adjacent to right Thalamus	4.7	2.3	1



Unsupervised pathology detection in biomedical images

Unsupervised pathology detection of a brain tumor image and resulting anomaly scores (growing values: blue → red)



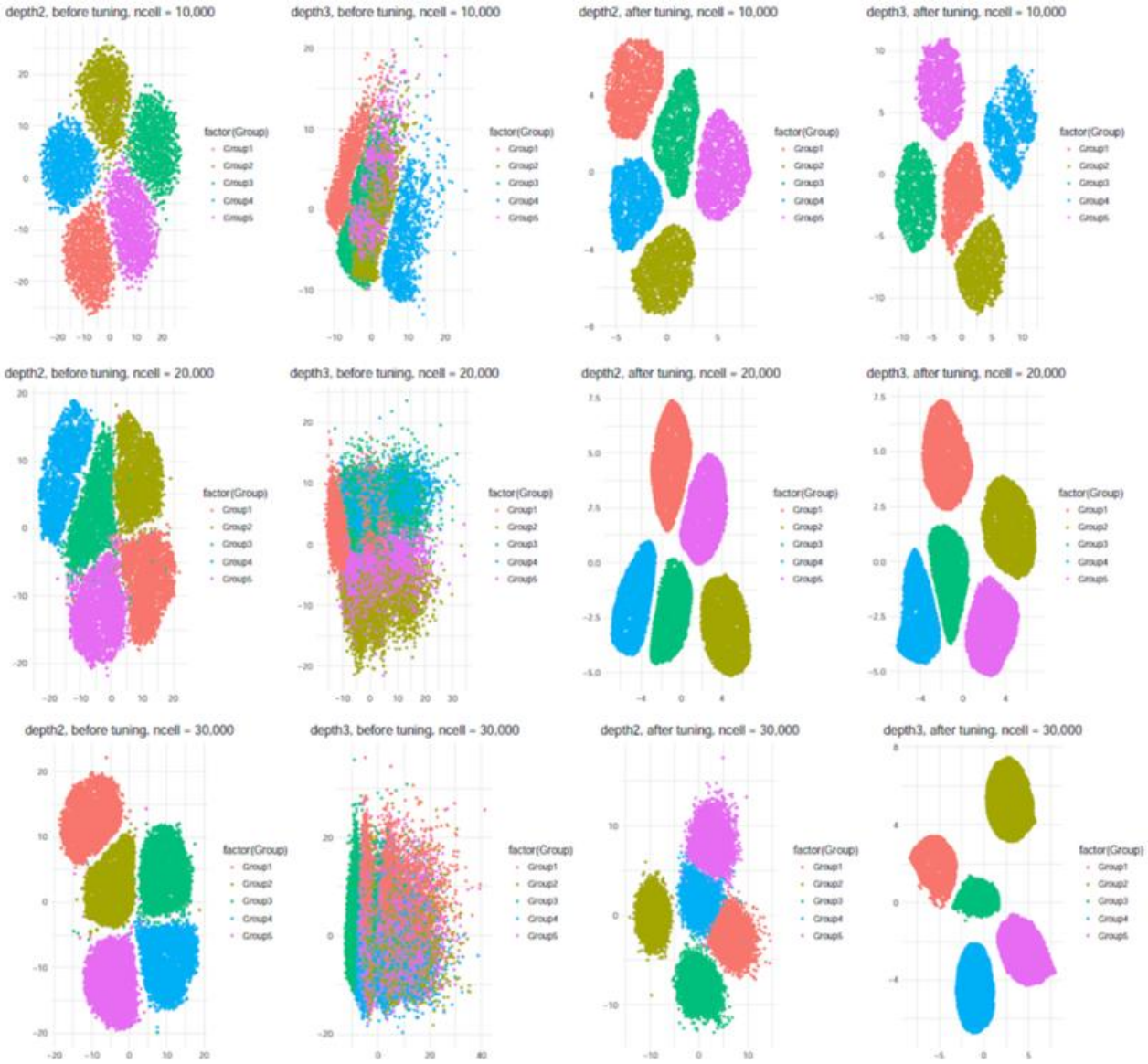
https://www.sciencedirect.com/science/article/pii/B9780128243497000153

Autoencoder for Data Compression

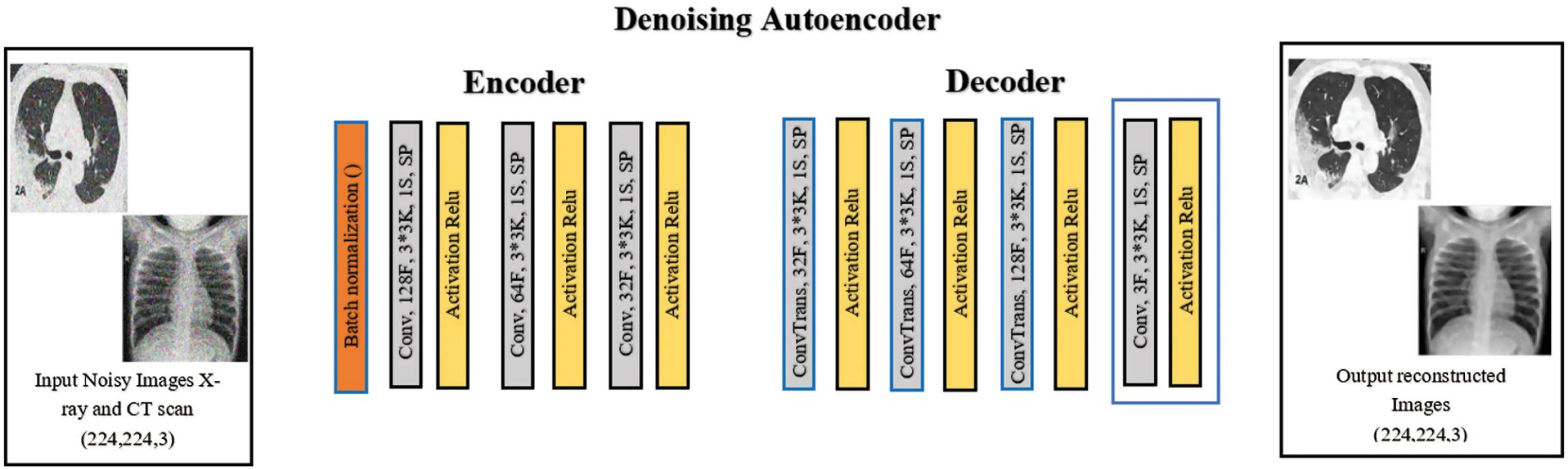
- The hidden layer of an autoencoder provides a compressed representation of the input which can be used for dimensionality reduction
- If the bottleneck size k is smaller than the input size n, the network is forced to learn a compressed but meaningful encoding
  - Nonlinear dimensionality reduction (differently from PCA)
  - When k=2, the hidden layer represents 2D coordinates and can be used for data visualization
  - When using a (deep) multi-layer bottleneck we can obtain a hierarchical dimensionality reduction
- Other uses beyond dimensionality reduction
  - Image Compression → Compress images while preserving important details.
  - Audio Compression → Reduce bitrate while keeping sound quality.
  - Latent Representations for ML Models → Extract compressed feature vectors for classification.

Autoencoders for dimensionality reduction and visualization of gene expression data

https://pmc.ncbi.nlm.nih.gov/articles/PMC6417816/



Autoencoder for Biomedical Image Denoising



https://www.techscience.com/cmc/v70n3/44978/html

Wrap-up

Take home lessons

- Deep learning is about learning hierarchical representations of input features
- Several factors are key to determine efficacy of the model
  - Activation functions
  - Activation normalization and regularization techniques
  - Weight initialization and architectural tricks (residual connectivity)
  - Optimization strategies
- Neural autoencoders are powerful tools for unsupervised learning tasks
  - Key catch: information bottleneck obliges the network to focus only on relevant factors of variation
  - Anomaly detection, information compression, dimensionality reduction, denoising
  - AEs provide refined neural representations on the top of which supervised tasks can be built

Next 2 Lectures

- Introduction to medical imaging
  - Image representation
  - Medical imaging as an inverse problem
  - Imaging modalities and their challenges
- Convolutional neural networks
  - Convolutional layers, filters/kernels, feature maps
  - Pooling layers and their role
  - Convolutional architectures and useful architectural tools
- Medical imaging tasks
  - Classification, regression, segmentation, detection et al