Neural Networks

Mohsen Moghaddam, Ph.D.

Gary C. Butler Family Associate Professor
H. Milton Stewart School of Industrial and Systems Engineering
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

SAIL
Symbiotic & Augmented Intelligence Laboratory

Spring 2026

Learning Outcomes

- Explain why linear models fail for certain problems and how neural networks learn nonlinear decision boundaries
- Describe the perceptron model and its connection to logistic regression
- Understand the role of activation functions and hidden layers in increasing model expressiveness
- Interpret neural networks as compositions of simple nonlinear units
- Explain the backpropagation algorithm using the chain rule and gradient descent
- Train and apply neural networks for binary and multi-class classification tasks
- Recognize common neural network architectures (CNNs, RNNs, autoencoders, GANs) and their typical applications
- Understand why deep learning succeeds at scale and its key practical limitations

Motivation & Historical Context

Main Approaches to Design Classifiers

Bayes rule + assumption for $p(x|y=1)$

- Assume $p(x|y=1)$ is Gaussian
- Assume $p(x|y=1)$ is fully factorized

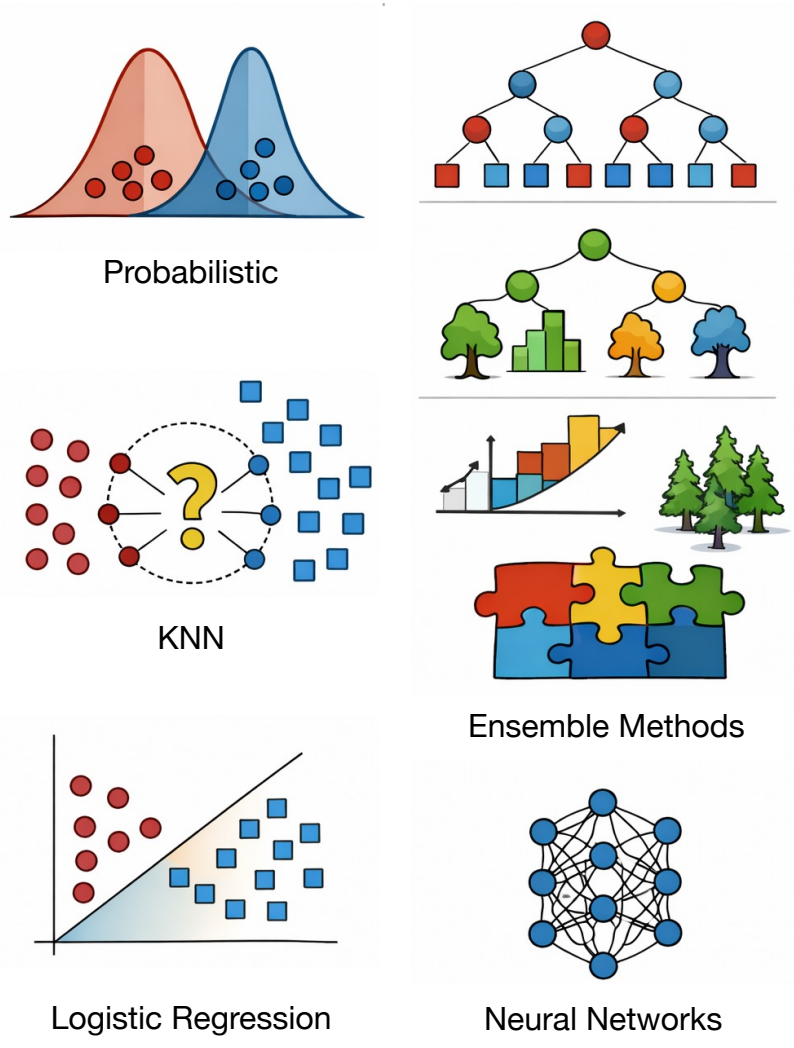
Geometric intuition

- K-nearest neighbor classifier
- Support vector machine

Directly apply decision boundary $h(x) = -\ln \frac{q_i(x)}{q_j(x)}$

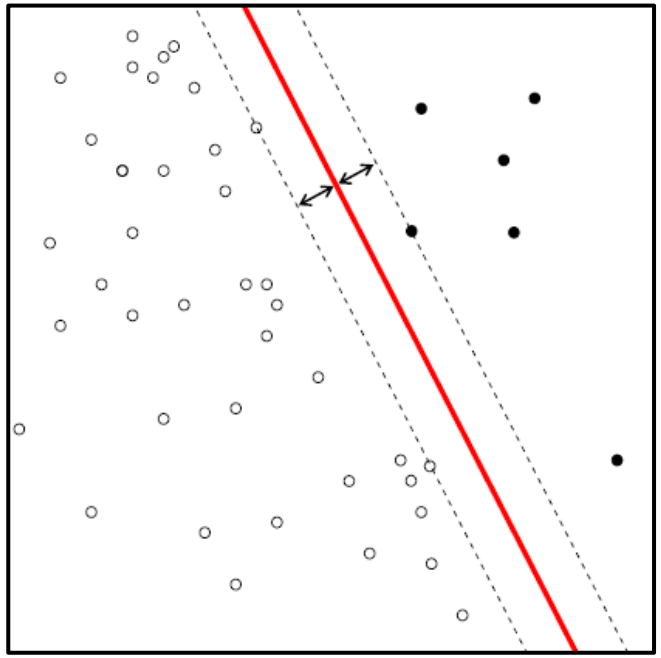
- Logistic regression
- Neural networks

  → **Philosophy:** Learn flexible internal representations that transform data to capture complex, nonlinear patterns



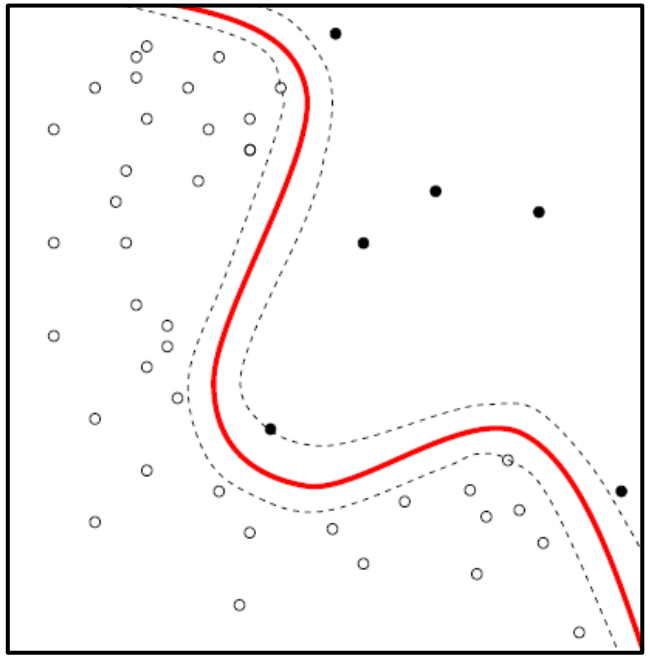
Learning Nonlinear Decision Boundary

Linearly Separable

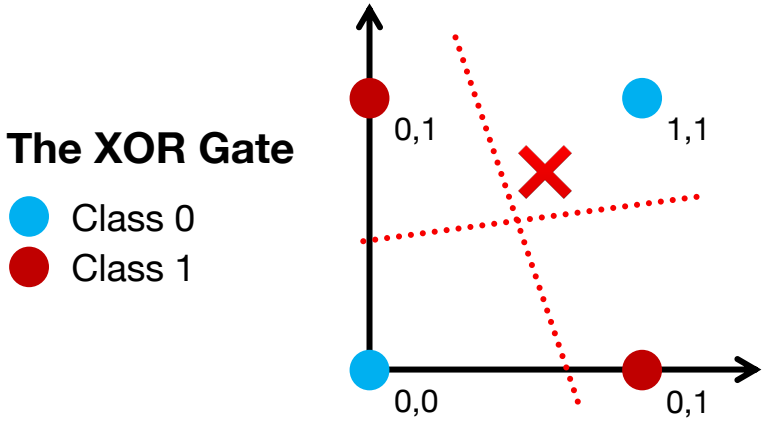


vs

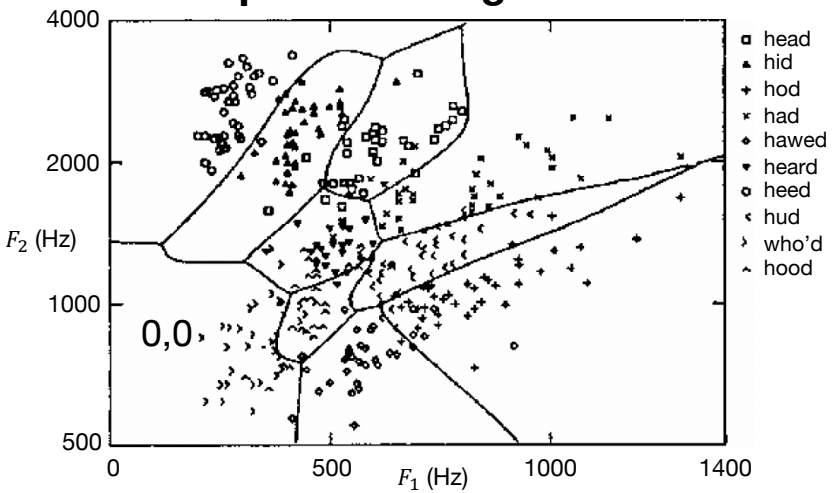
Nonlinearly Separable



The XOR Gate



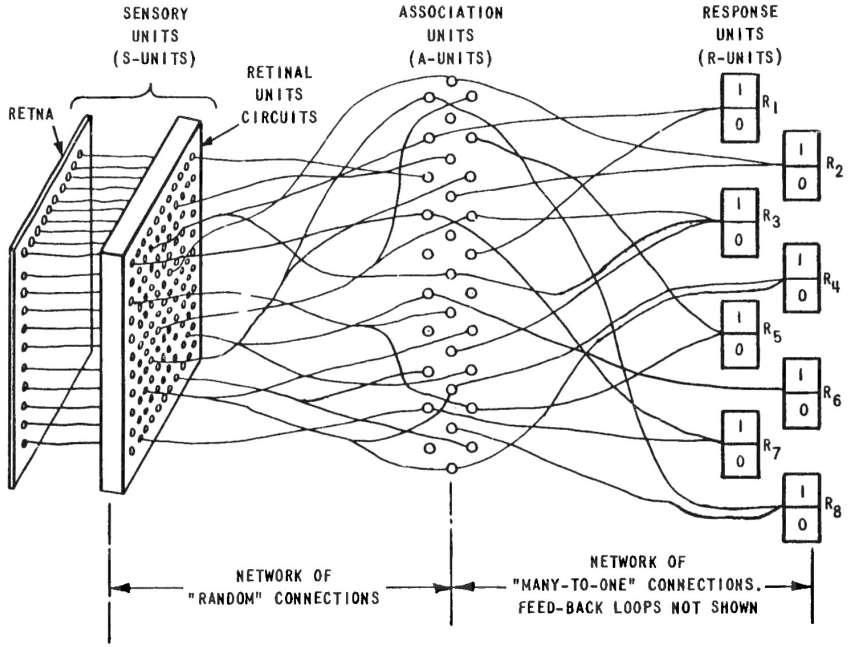
Speech Recognition



Neural networks learn nonlinear feature transformations that make classification easier

Early Artificial Intelligence

- The **perceptron algorithm** was first invented in 1958 at the Cornell Aeronautical Lab
- This machine was designed for **image recognition**: it had an array of 400 photocells, randomly connected to the “neurons”

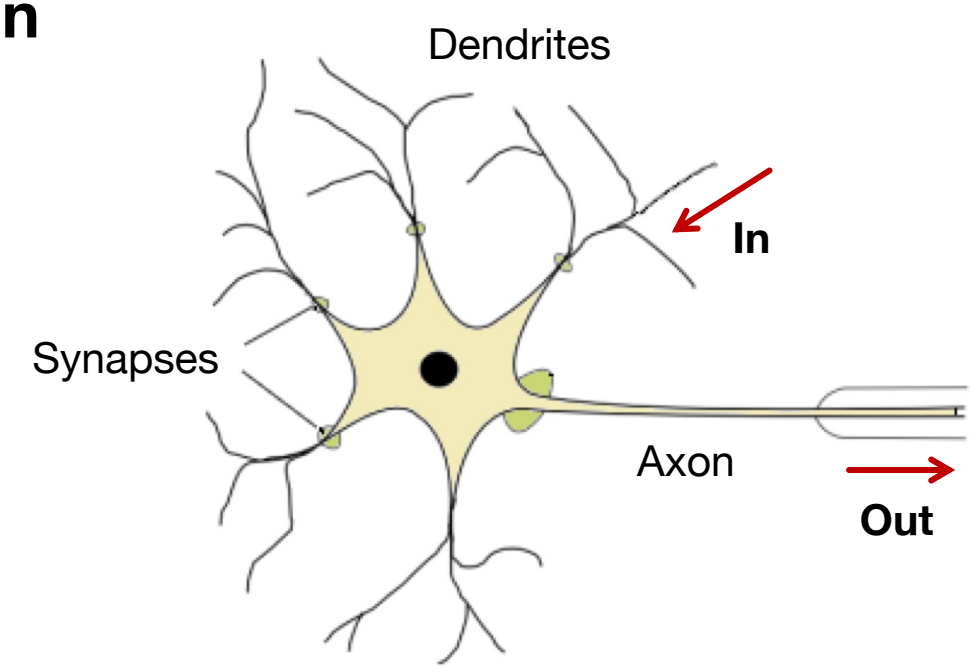


Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is a patch panel that set different combinations of input features. To the right, arrays of potentiometers that implemented the adaptive weights.

Perceptron &
Single Neuron Models

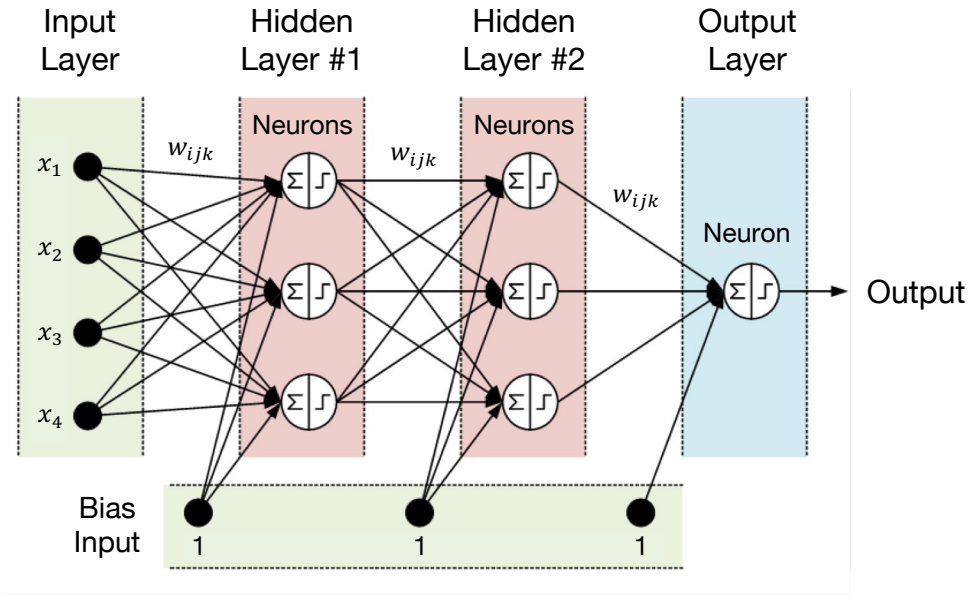
Perceptron: Artificial Neurons

Neuron



Biological → Artificial Mapping

- Dendrites → input x
- Synaptic strength → weights w
- Firing threshold → bias b
- Neuron firing → activation function



Artificial neural networks

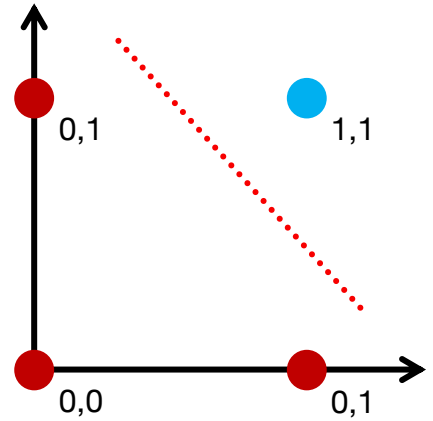
- Many neuron-like threshold switching units
- Weighted interconnections among units (feed-forward)

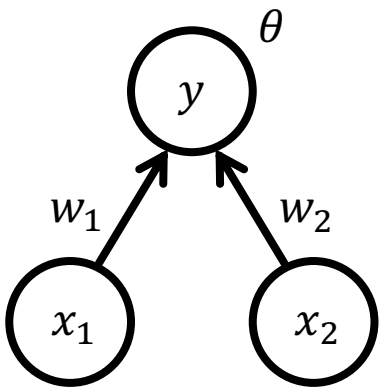
Perceptron: NAND Gate Example

NAND (Not AND) Gate

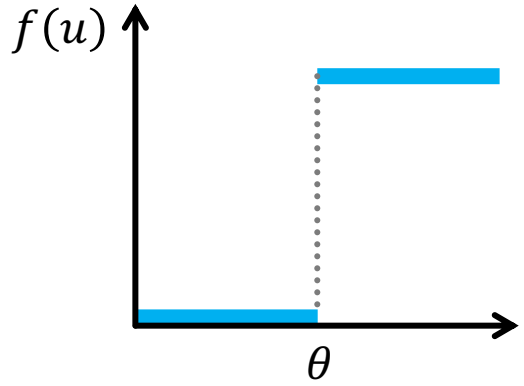
- Output true for every other input combination
- Output false only when all its inputs are true

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0





$$f(x_1w_1 + x_2w_2) = y$$
$$f(0w_1 + 0w_2) = 1$$
$$f(0w_1 + 1w_2) = 1$$
$$f(1w_1 + 0w_2) = 1$$
$$f(1w_1 + 1w_2) = 0$$



$$f(u)\begin{cases}1, & \text{for } u > \theta \\ 0, & \text{for } u \le \theta\end{cases}$$

Some possible values at $\theta = 0.5$

w_1	w_2
0.20	0.35
0.20	0.40
0.25	0.30
0.40	0.20

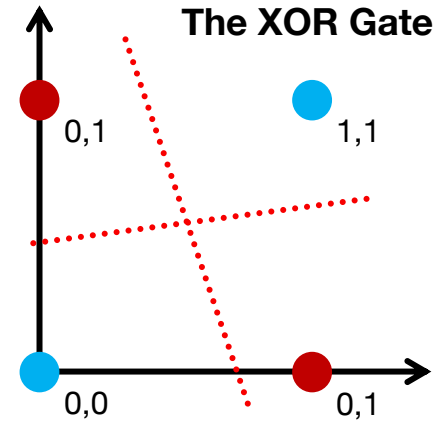
The perceptron computes: $y = \mathbb{I}(w_1x_1 + w_2x_2 > \theta)$

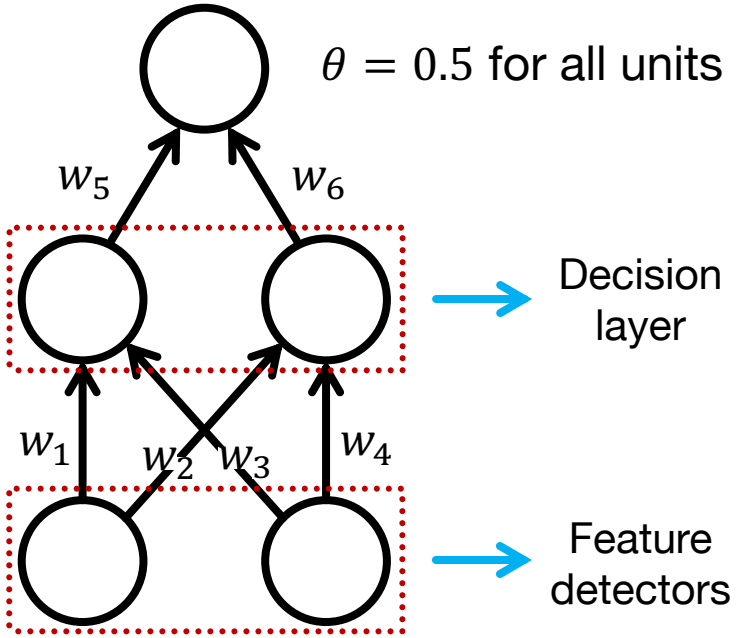
Connecting Perceptrons

XOR (Exclusive OR) Gate

Stacking perceptrons introduces intermediate representations for nonlinear classification

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





A solution set:

w_1	0.6
w_2	-0.6
w_3	-0.7
w_4	0.8
w_5	1
w_6	1

Expressive Power & Activation Functions

Expressive Capability of Neural Networks

Boolean Functions (Binary Inputs & Outputs)

- Every Boolean function can be represented by network with single hidden layer
- But this may require exponential (in number of inputs) hidden units

Continuous Functions (Real Inputs & Outputs)

- Every bounded continuous function can be approximated with arbitrary small error, by network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers
- Many recent developments for more general activation functions (e.g., **ReLU**)

Depth of a neural net trades off number of units vs. compositional structure

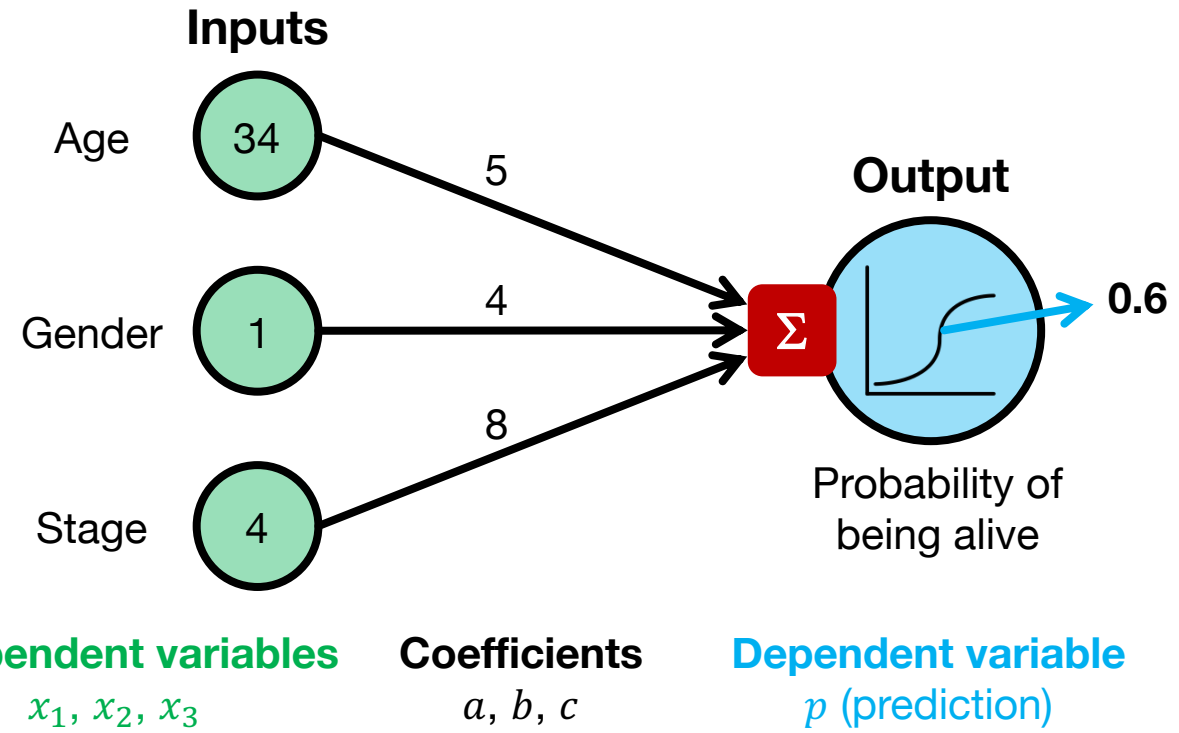
Logistic Regression as a Single Neuron

The logistic regression model (the **sigmoid unit**) is a perceptron:

- Independent variables = input variables
- Coefficients = “weights” + “bias”
- Dependent variable = output variable

$$p(y = 1|x) = \sigma(w^{\top}x + b)$$

Sigmoid function



Neural nets generalize logistic regression by stacking multiple sigmoid units

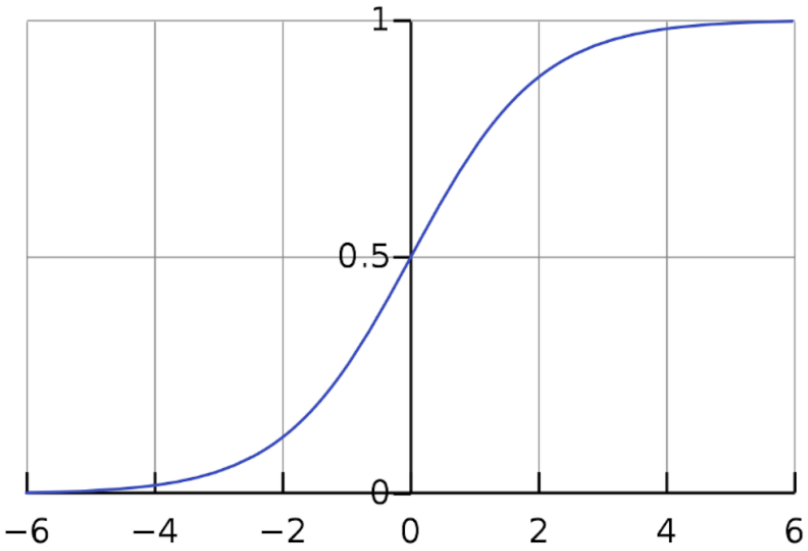
Connection to Logistic Regression

Let the original feature be denoted as u (n dimensional vector)

- We can define the augmented feature as $x = [1; u^\top]^\top$ ($n+1$ dimensional vector)
- The weight vector is $\tilde{w} = [b, w^\top]^\top$
- Assume that the posterior distribution $p(y = 1|x)$ takes a particular form:

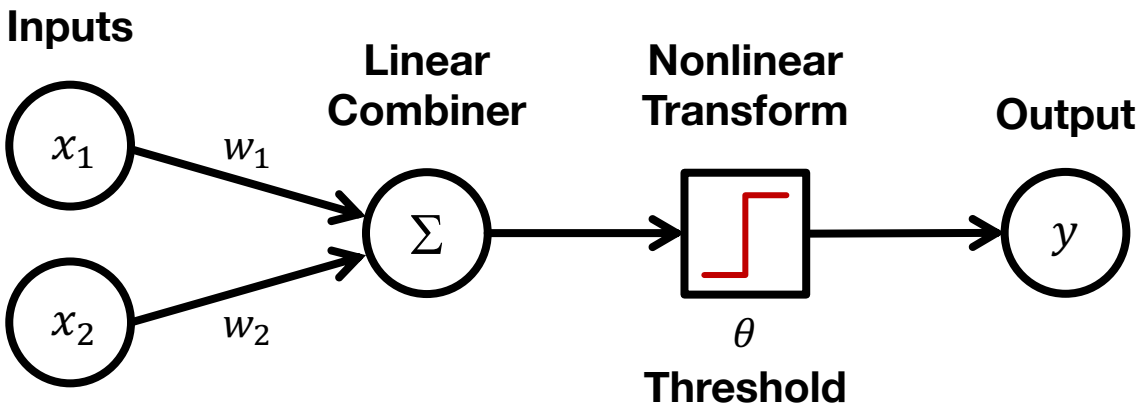
$$p(y = 1|x, \tilde{w}) = \frac{1}{1 + \exp(-\tilde{w}^\top x)} = \sigma(\tilde{w}^\top x)$$

Logistic function (Sigmoid function) → $\sigma(u) = \frac{1}{1 + \exp(-u)}$

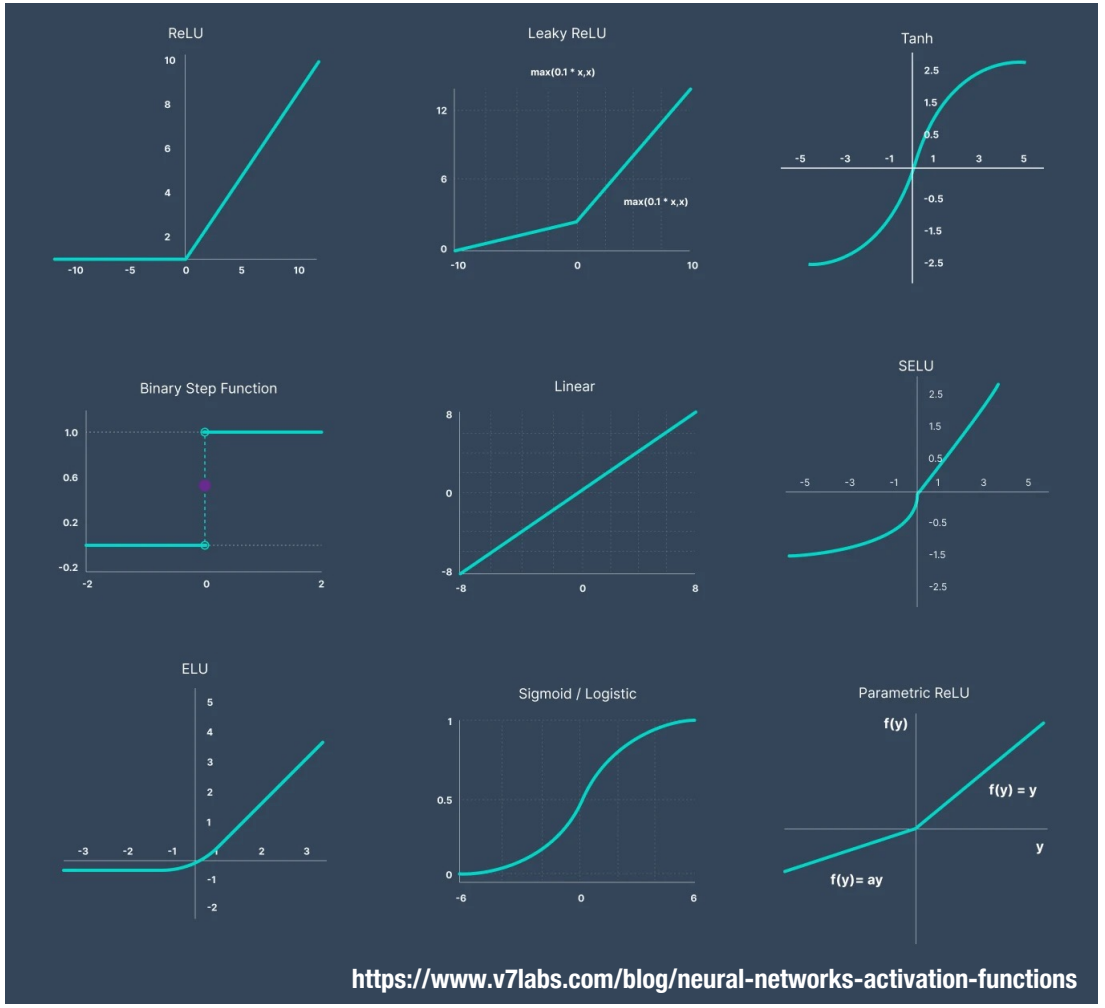


Logistic regression can be viewed as one neuron

Activation Functions

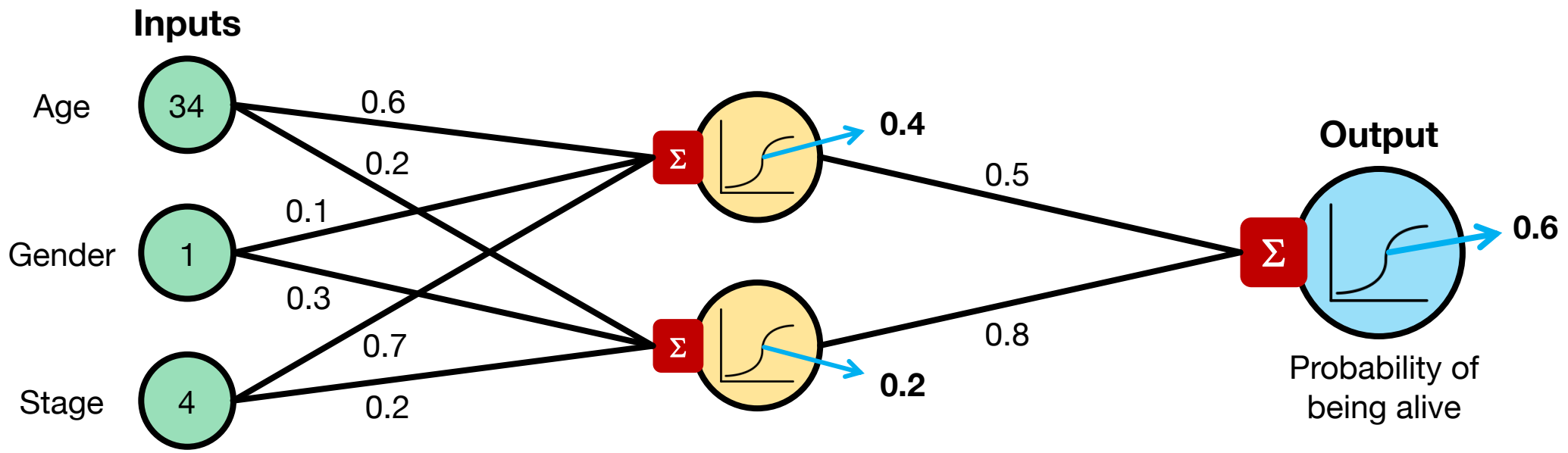


- **ReLU:** sparse, fast, dead neurons
- **Leaky ReLU:** reduces dead neurons
- **Tanh:** zero-centered, vanishing gradients
- **Binary Step:** hard threshold, no gradients
- **Linear:** no non-linearity, limits model expressiveness
- **SELU:** self-normalizing activations
- **ELU:** smooth negatives, faster convergence
- **Sigmoid/Logistic:** probabilistic, vanishing gradients
- **Parametric ReLU:** learnable negative slope



Multilayer Networks & Backpropagation

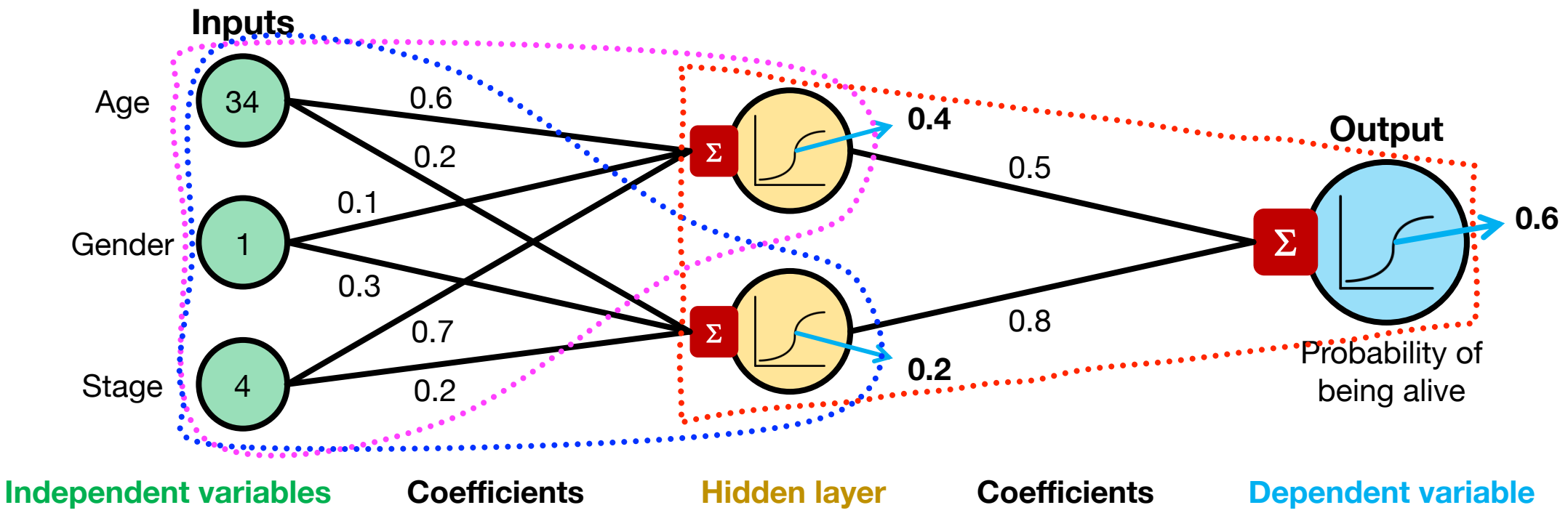
Deep Neural Network Model



- Input layer → Hidden layers → Output layer
- Parameters = all weights + biases
- **Last layer depends on task** (e.g., classification: softmax)

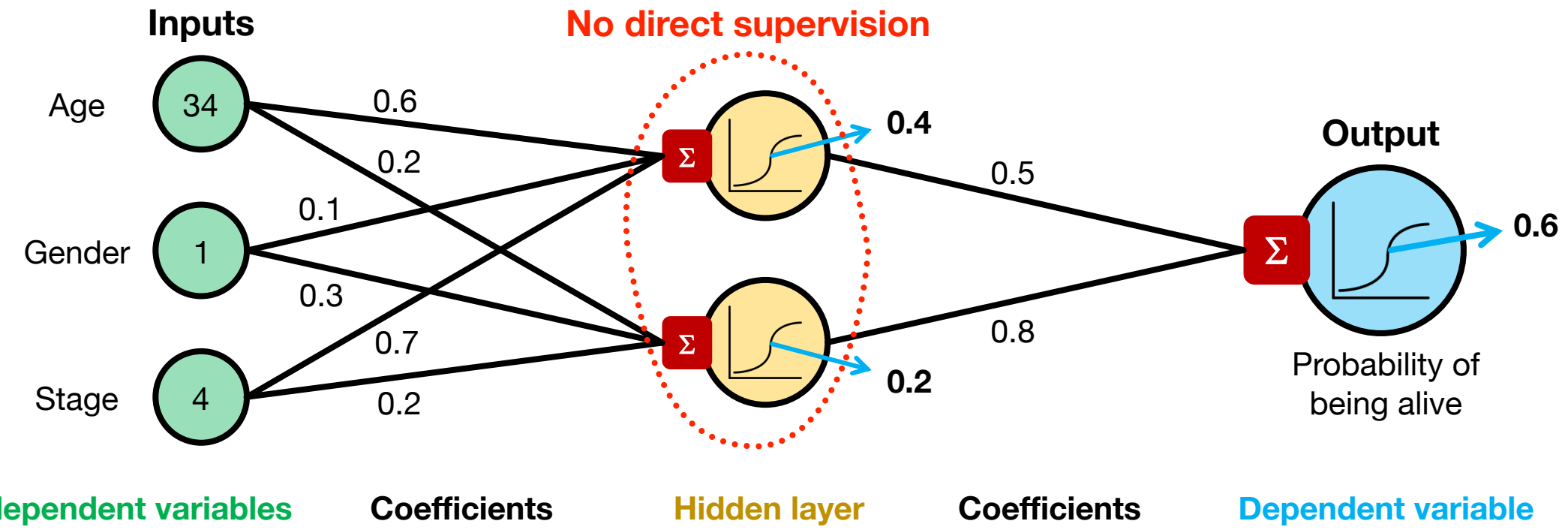
$$p(y^i = c|x^i) = \frac{\exp(\theta_c^\top x^i)}{\sum_{c'} \exp(\theta_{c'}^\top x^i)}$$

Combined Logistic Models



Neural networks: nested logistic regressions—each hidden unit outputs a transformed feature (a probabilistic prediction) used by the next layer

Overall Structure



- Hidden layers have **no direct supervision**
- Learning happens via **error propagation** from the output layer
- Turn weights and thresholds such that the neural networks will give desired prediction results

Learning Parameters in Logistic Regression

Find θ such that the **conditional likelihood of the labels** is maximized

Training examples

$$\max_{\theta} \ell(\theta) := \log \prod_{i=1}^{m} p\left(y_i | x^i, \theta\right)$$

- **Good news:** $\ell(\theta)$ is a concave function of $\theta \rightarrow$ there is a single global solution
- **Bad news:** No closed form solution exists $\rightarrow$ resort to numerical methods

Logistic regression $\rightarrow$ convex optimization
Neural networks $\rightarrow$ non-convex optimization

Backpropagation

Find w, α, β such that the output unit values are close to the actual labels (0 or 1)

α: weights connecting inputs to hidden neuron 1
β: weights connecting inputs to hidden neuron 2
w: weights connecting hidden layer to output neuron

$$\mathcal{L} = \sum_i \ell\left(y^i, \hat{y}^i\right)$$

$$\min_{w,\alpha,\beta} \ell(w,\alpha,\beta) := \sum_i^m \left(y^i - \sigma\left(w^\top z^i\right)\right)^2$$

→ Non-convex objective function

where

$$z_1^i = \sigma\left(\alpha^\top x^i\right)$$
$$z_2^i = \sigma\left(\beta^\top x^i\right)$$

→ Sigmoid functions (intermediate activation)

Use gradient decent to find a local optimum

Backpropagation efficiently computes gradients for all parameters

Gradient of the Loss $\ell(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

$$\min_{w,\alpha,\beta} \ell(w,\alpha,\beta) := \sum_{i}^{m} \left(y^i - \sigma\left(w^\top z^i\right)\right)^2$$

$$z_1^i = \sigma\left(\alpha^\top x^i\right) \qquad z_2^i = \sigma\left(\beta^\top x^i\right)$$

Let $u^i = w^\top z^i$:

$$\frac{\partial \sigma(u)}{\partial u} = \sigma(u)\left(1 - \sigma(u)\right)$$

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial w} = -\sum_{i} 2\left(y^i - \sigma\left(u^i\right)\right)\sigma\left(u^i\right)\left(1 - \sigma\left(u^i\right)\right) z^i$$

z^i is computed using α and β from previous iteration:

- Output-layer gradient = prediction error × sensitivity
- Hidden-layer gradients reuse downstream errors

Chain Rule of Derivatives

Use chain rule of derivatives (backpropagation): The "error" at layer k depends on the error at the next layer $k+1$

Let $v^i = \alpha^\top x^i$:

$$\frac{\partial \ell(w,\alpha,\beta)}{\partial \alpha} = \frac{\partial \ell(w,\alpha,\beta)}{\partial z_1^i}\frac{\partial z_1^i}{\partial \alpha}$$

$$= -\sum_i 2\left(y^i - \sigma(u^i)\right)\sigma(u^i)\left(1-\sigma(u^i)\right)w_1\sigma(v^i)\left(1-\sigma(v^i)\right)x^i$$

Forward pass: compute activations
Backward pass: compute gradients
Update parameters using gradient descent

→

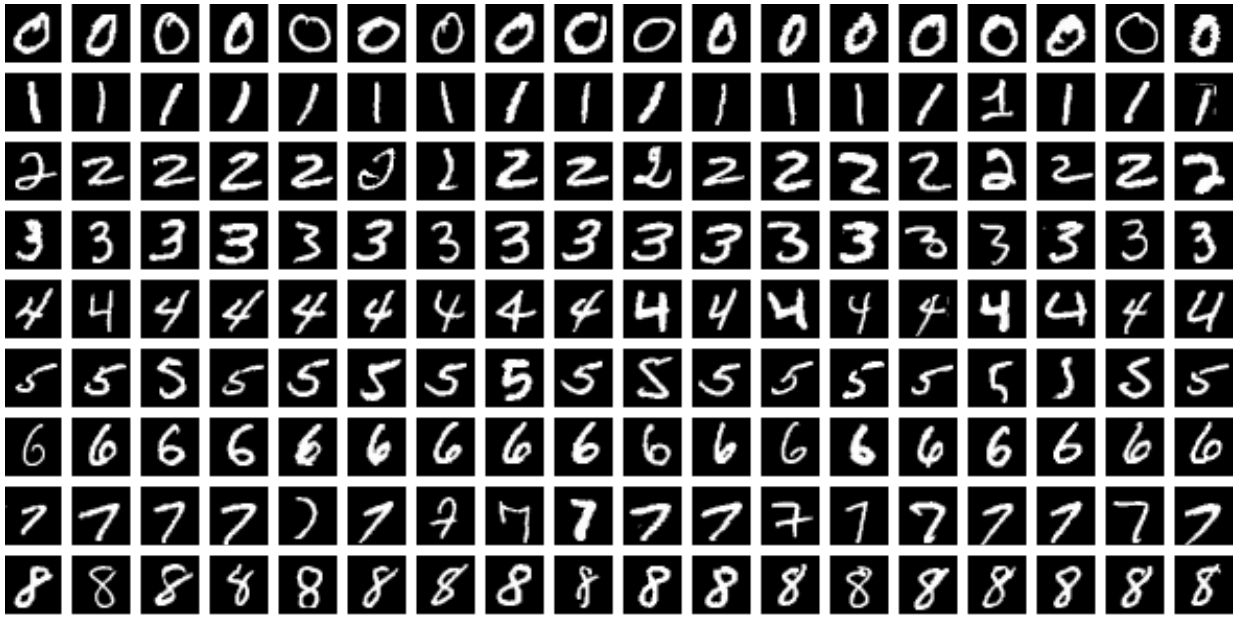
$$w \leftarrow w - \eta\nabla_w \ell$$
$$\alpha \leftarrow \alpha - \eta\nabla_\alpha \ell$$
$$\beta \leftarrow \beta - \eta\nabla_\beta \ell$$

https://youtu.be/Ilg3gGewQ5U?si=thT3-OcNZPWAauuK

Practical Examples

Handwritten Digits Classification

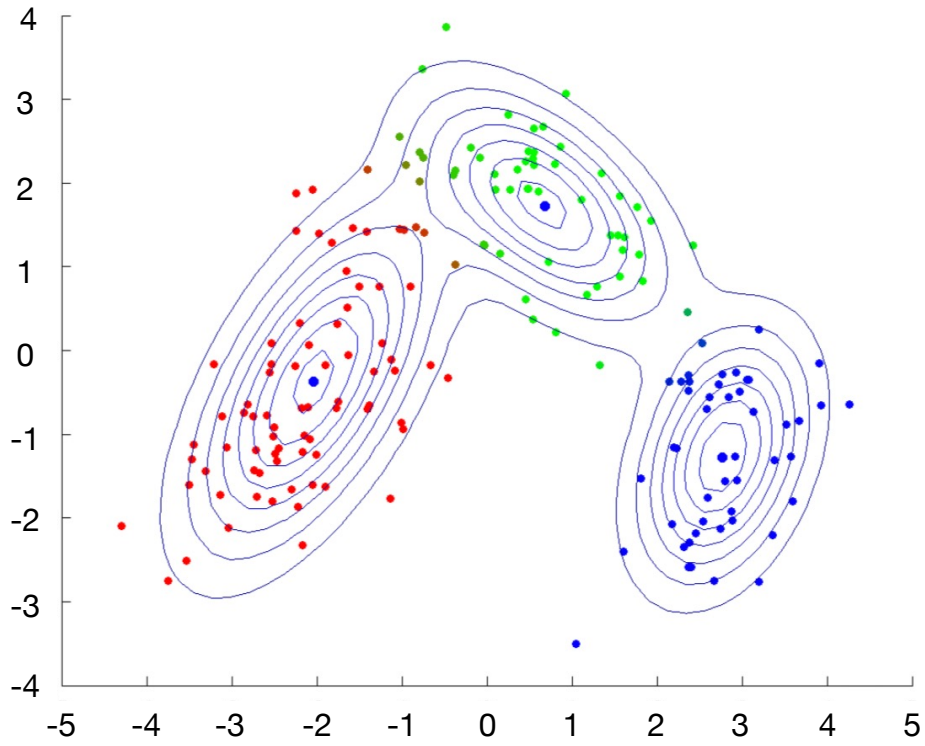
- The input data consists of 28x28 pixel handwritten digits, leading to 784 features in the dataset
- One-hidden layer
- The first layer weight matrix have the shape (784, hidden layer size)
- We can therefore visualize weight for each hidden node as a 28x28 pixel image
- Each hidden unit learns a stroke or pattern detector



https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

Wine Classification

- **Data:** Chemical analysis of wines grown in the same region in Italy but derived from three different cultivars
- 13 features, three types of wines
- Use multi-class classification in the last layer
- Neural networks reduce to flexible nonlinear classifiers even on small tabular data
- Risk of overfitting → need for regularization

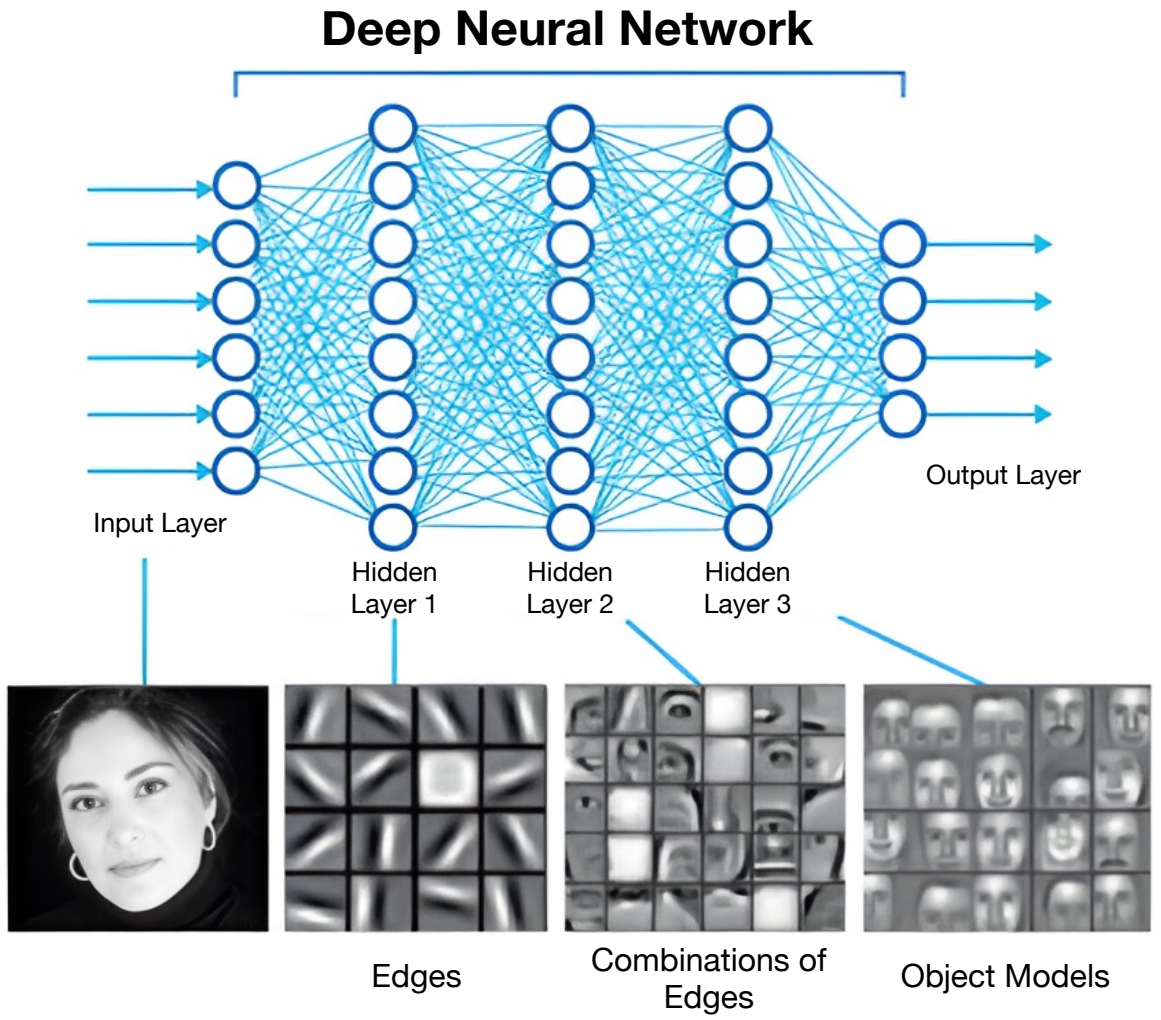


Why might a neural network overfit more easily here than on MNIST?

Deep Learning

Why Deep Learning

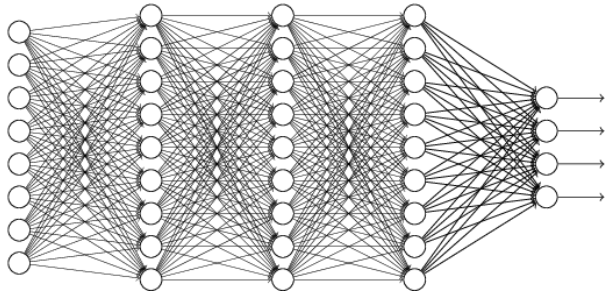
- **Representation learning:** Deep learning automatically learns meaningful features from raw data, reducing the need for manual feature engineering
- **Scalability with data:** Its performance typically improves as more data becomes available, making it well-suited for large datasets
- **GPU acceleration:** Training and inference are significantly faster by leveraging GPUs for parallel computation



Common Neural Network Structures

Full Connected Neural Network

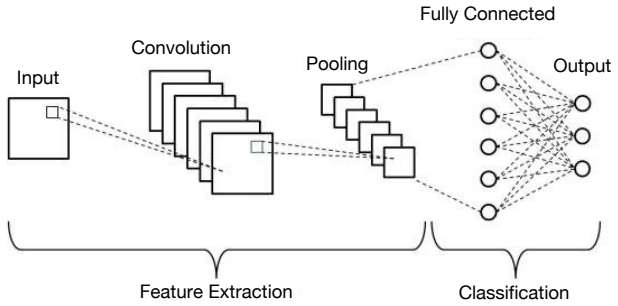
- All nodes in the k^{th} layer are connected to all nodes in the $(k+1)^{\text{th}}$ layer
- Not practical



Fully Connected NN

Convolutional Neural Network

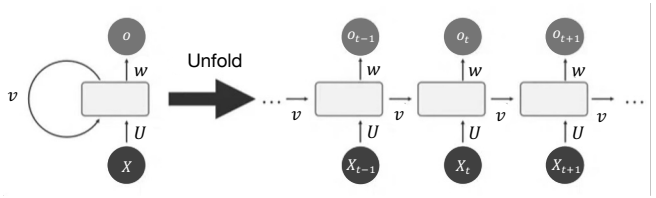
- Good for image data
- Nodes in one layer are only "locally" connected to nodes in the next layer
- Two types of layers: convolution & max pooling



CNN

Recurrent Neural Network

- Good for sequential / time series data
- Neural network structure "repeats" itself
- Attention mechanism, transformer



RNN

Residual Neural Network

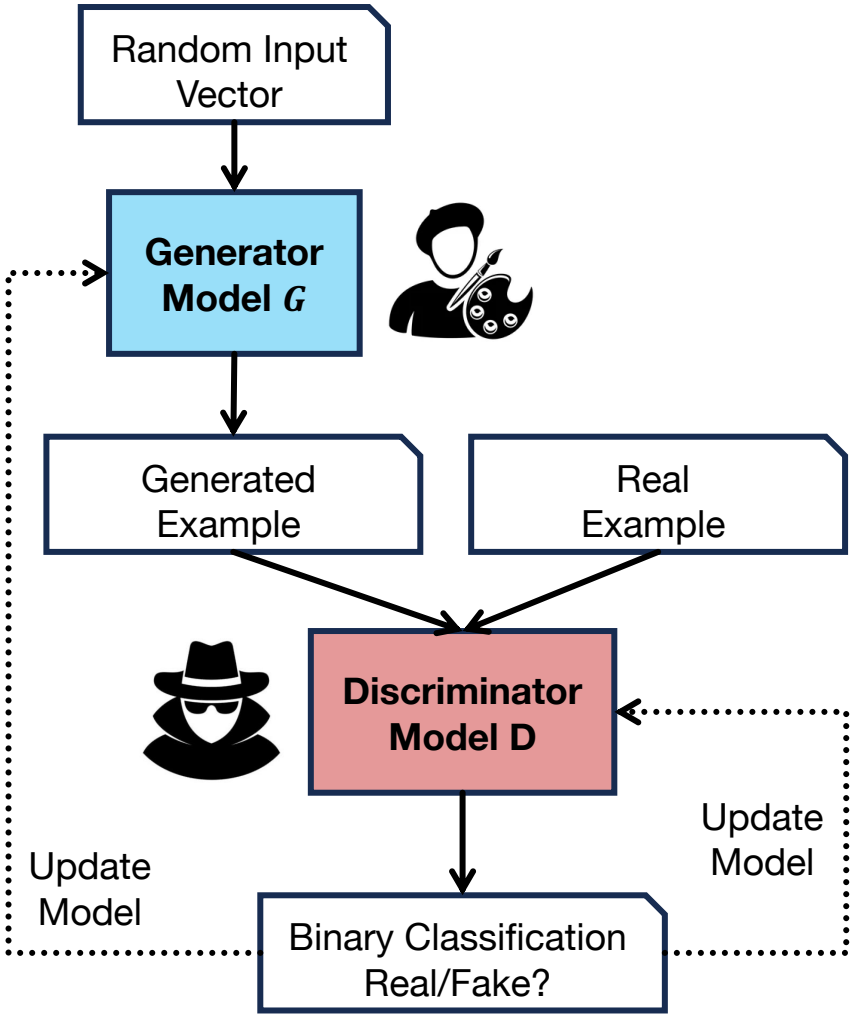
- Very deep models
- Jump over some layers

Generative Adversarial Networks

Generative model and better discriminator, trained by **competition**, not likelihood max:

$$\min_G \max_D V(D,G)$$
$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

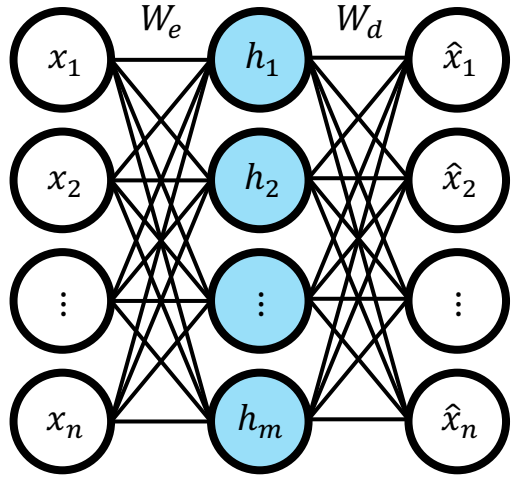
An image generated by a StyleGAN that looks deceptively like a photograph of a real person, generated by a StyleGAN based on an analysis of portraits.



Why does GAN competition help generate realistic samples?

Autoencoder

Autoencoder is a neural networks for unsupervised learning to represent the data itself



Decoder parameters

Encoder parameters

$$\min_{W_e, W_d} \sum_{i=1}^{N} \|x_i - f_{\text{dec}}(f_{\text{enc}}(x_i; W_e); W_d)\|^2$$

Decoder function

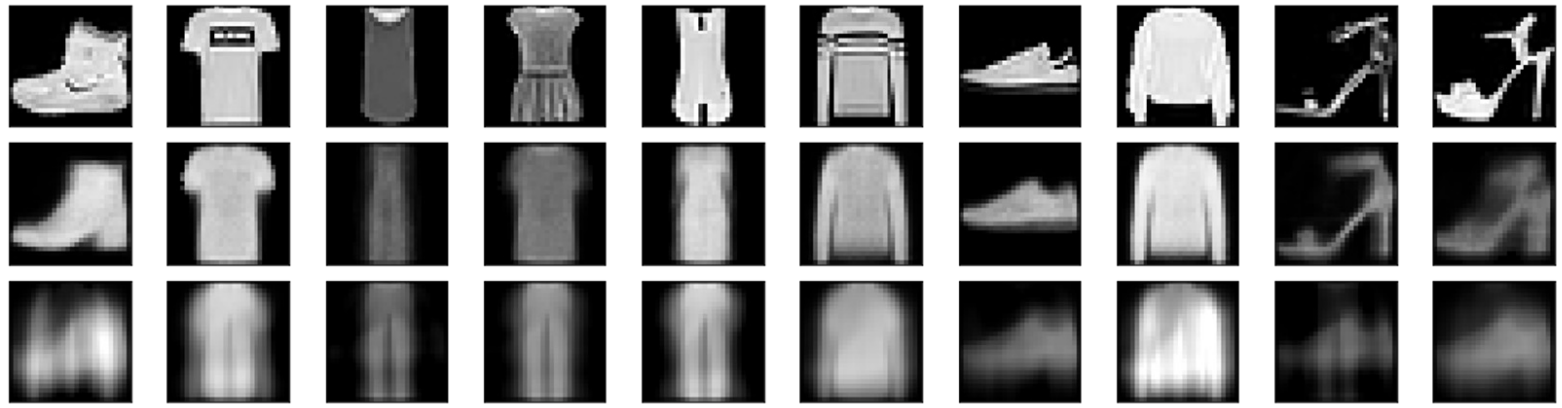
Encoder function

Fashion MNIST dataset

Original Image

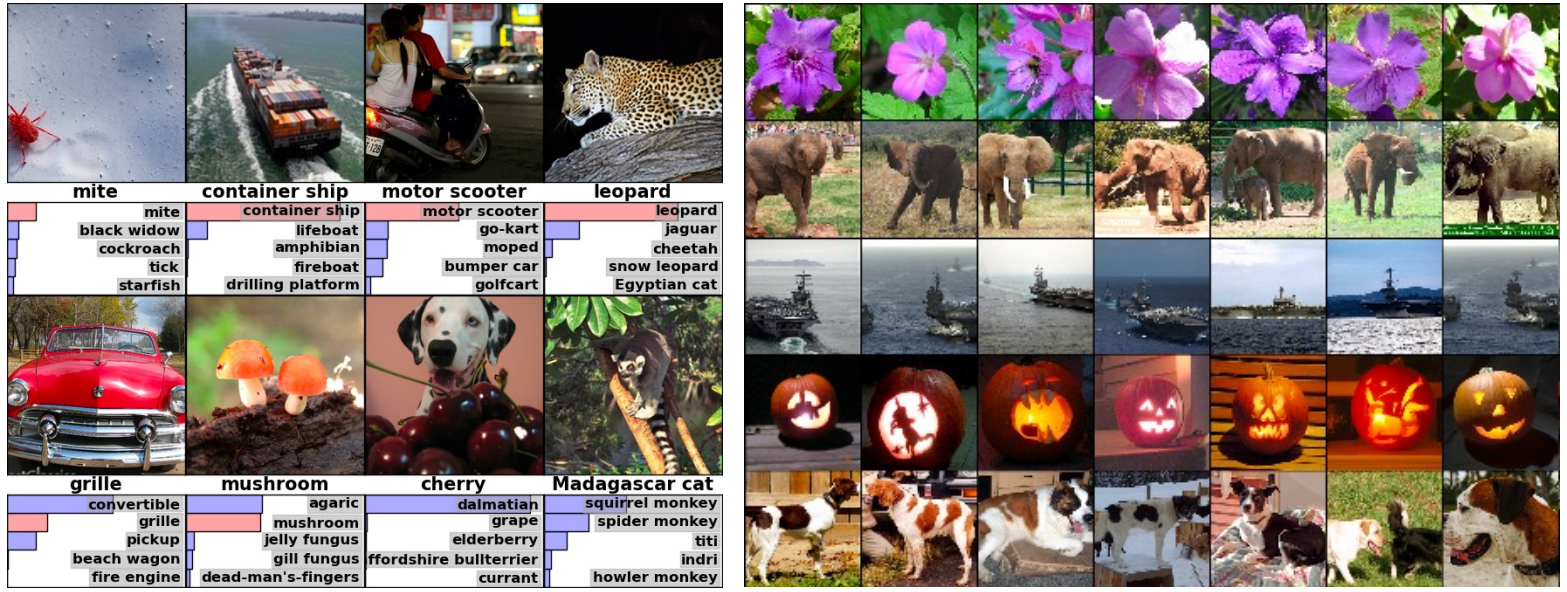
Autoencoder Reconstruction

PCA Reconstruction



ImageNet

Image classification with **1.3M color images** and **1000 classes**—need large-scale nonparametric methods to solve



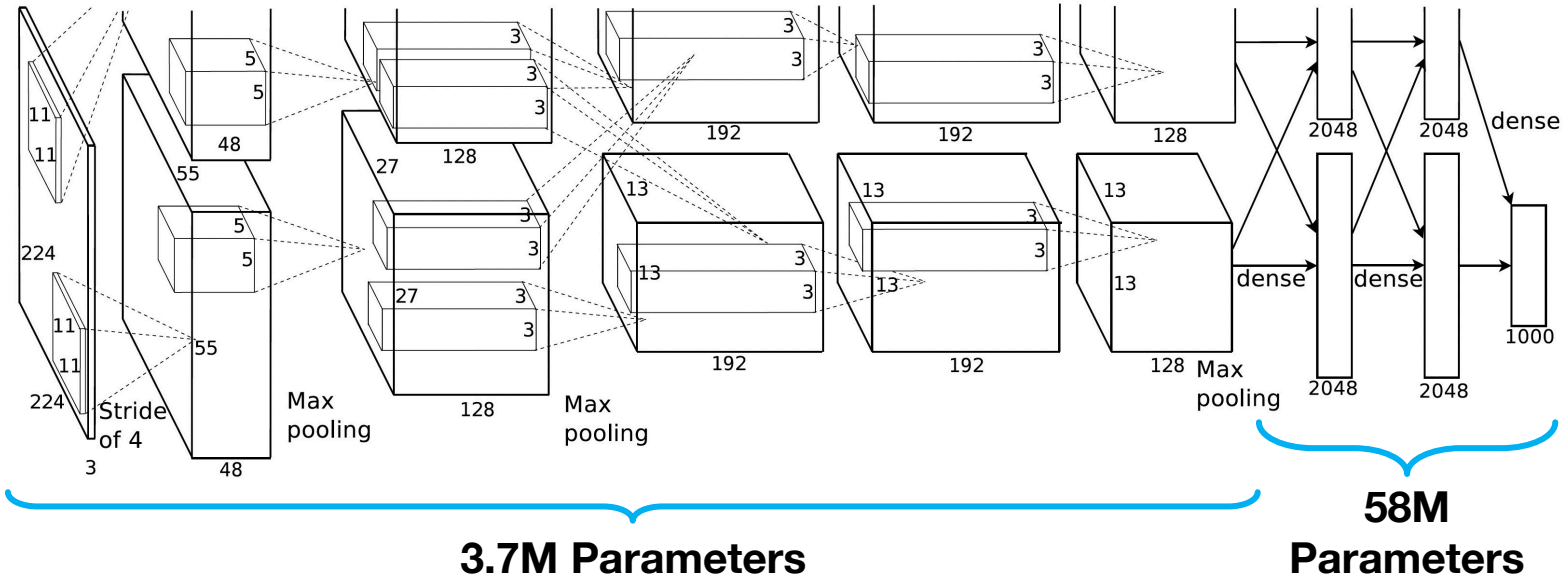
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems25 (2012).

ImageNet marked the transition from feature engineering to feature learning

Convolutional Neural Network (CNN)

8-layer CNN achieve state-of-the-art result, beating the second place by 10%

- **First 5 layers:** convolution + max pooling
- **Next 2 layers:** fully-connected, nonlinear neurons
- **Last layer:** multiclass logistic regression



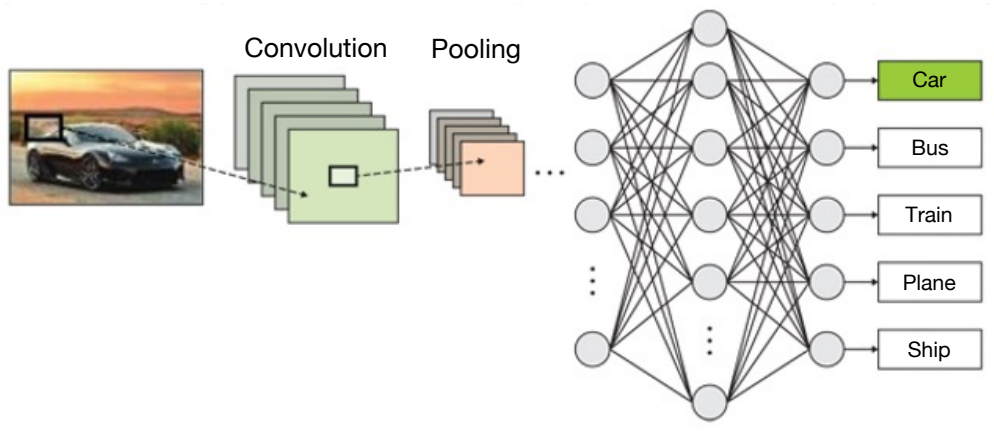
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems25 (2012).

Convolution Layer

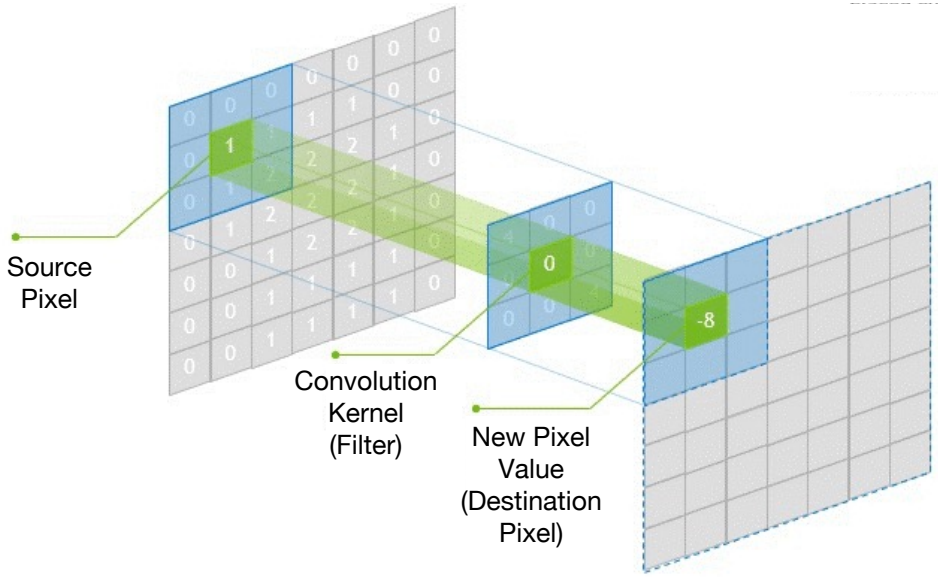
Source image I, destination image O, convolving image with a kernel K of size $2d + 1$

$$O(i,j) = \sum_{i'=i-d}^{i+d} \sum_{j'=j-d}^{j+d} I(i',j')K(i',j')$$

- Center element of the kernel is placed over the source pixel
- The source pixel is then replaced with a weighted sum of itself and nearby pixels



Credit: Nvidia



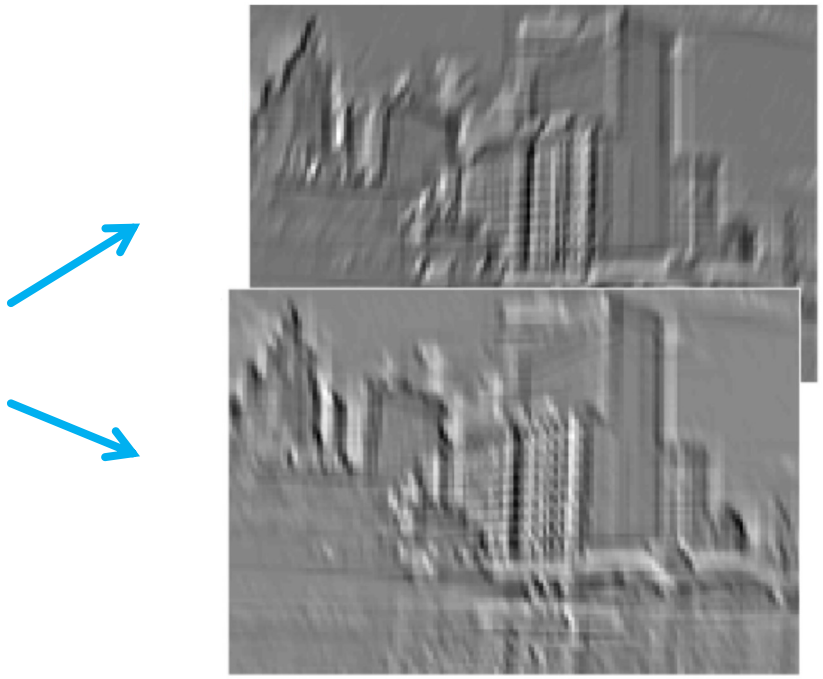
CNN Visualization

https://youtu.be/f0t-OCG79-U?si=2D5sRZRXbH3OgZfg

Convolutional Feature

Convolutional features are local patterns (such as edges, corners, or textures) extracted by sliding a shared filter across an image, enabling the network to detect the same feature at different spatial locations

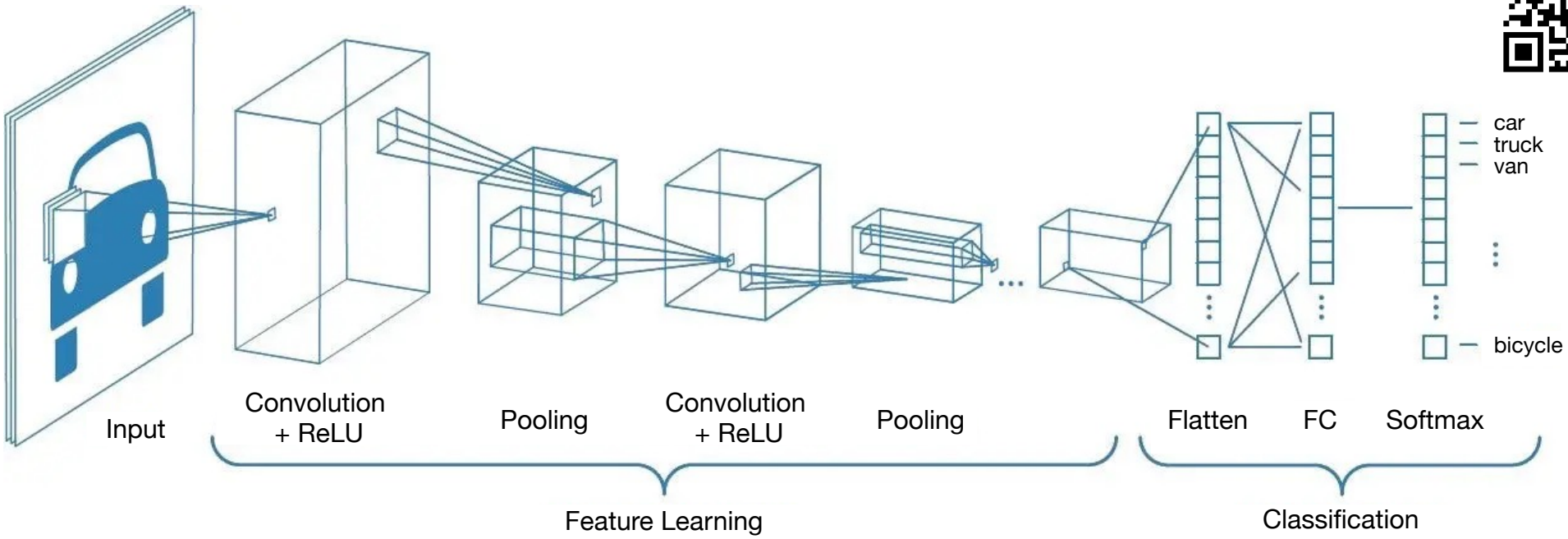
Input



Feature Map

Source: https://i.imgur.com/TqTJCrz.png

CNN Architecture

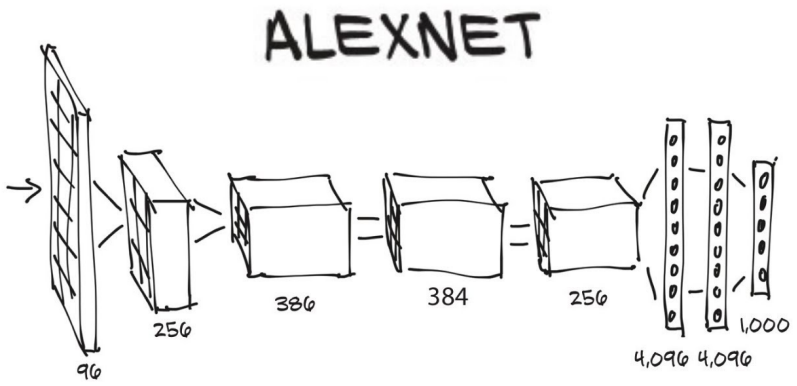


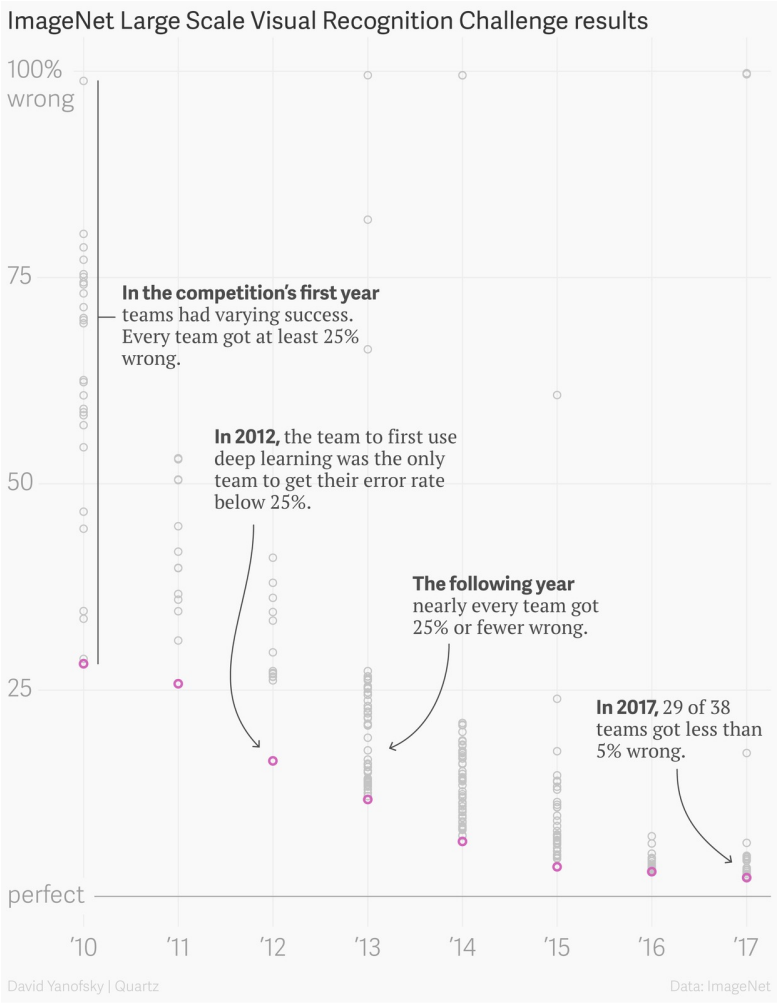
Credit: Sumit Saha

- **Convolution layers** extract local features using shared filters, preserving spatial structure
- **Pooling layers** (typically max pooling) reduce spatial resolution while retaining salient information
- This architecture provides translation invariance, parameter efficiency, and strong performance on image data

AlexNet

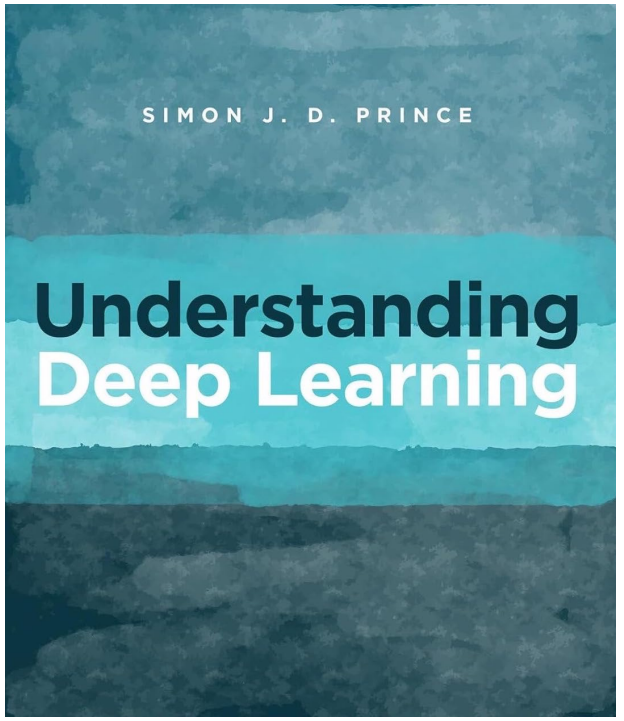
- In 2012, a CNN called AlexNet achieved a top-5 error of 15.3% in the ImageNet 2012 Challenge, more than 10.8 percentage points lower than that of the runner up
- This was made feasible due to the use of **GPUs** during training, an essential ingredient of the deep learning revolution

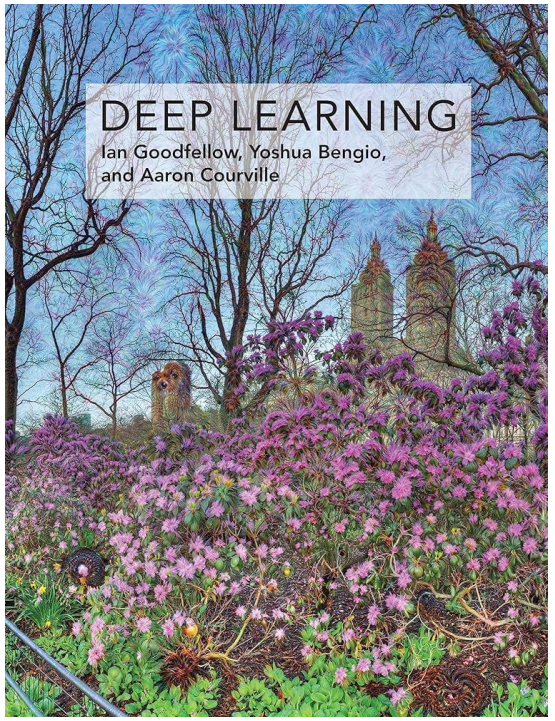




Final Thoughts

- Deep learning is one of the most impactful and rapidly evolving areas of modern machine learning
- It has driven major breakthroughs across domains such as computer vision, speech recognition, and natural language processing
- Despite its success, important challenges remain, including interpretability, reliability, and deployment in safety-critical applications





Deep learning has transformed how machines learn from data, but understanding why models work and ensuring they can be trusted remain active and important research challenges

Key Takeaways

What We Learned This Week

- Neural networks extend linear and logistic models by stacking nonlinear units to learn complex decision boundaries
- Hidden layers and activation functions enable automatic feature learning and richer representations
- Backpropagation with gradient descent efficiently trains many parameters despite non-convex optimization
- Depth provides expressive power while requiring regularization, scaling, and sufficient data to avoid overfitting
- Architecture should match data structure: fully connected for tabular, CNNs for spatial, sequence models for temporal data
- Deep learning performance scales with data and compute but raises challenges in interpretability, reliability, and deployment