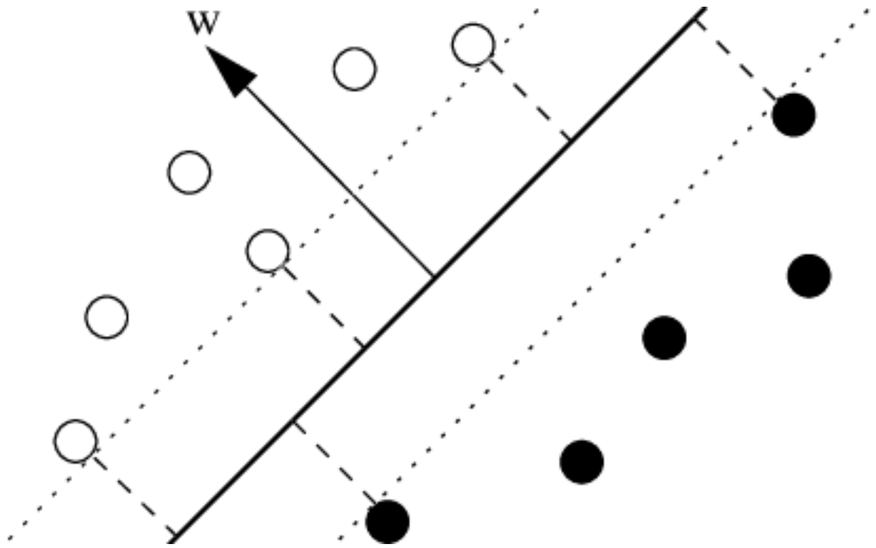
# Deep Learning: Overview

# Recap

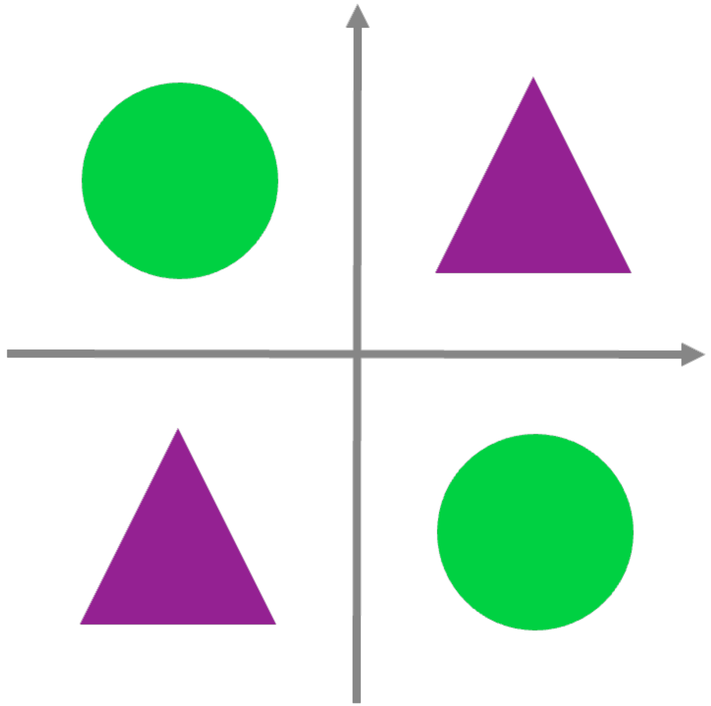
- So far, we have focused primarily on **linear models**
  - Perceptrons, logistic regression, linear regression, PCA. …

$$\min_{f:\text{linear}} \mathbb{E}_{\text{data}}[\ell(f(X), Y)]$$



# Recap

- **Strength.** Easy to optimize, Low inference cost
- **Weakness.** Limited expressive power
  - Requires the data to be linearly separable

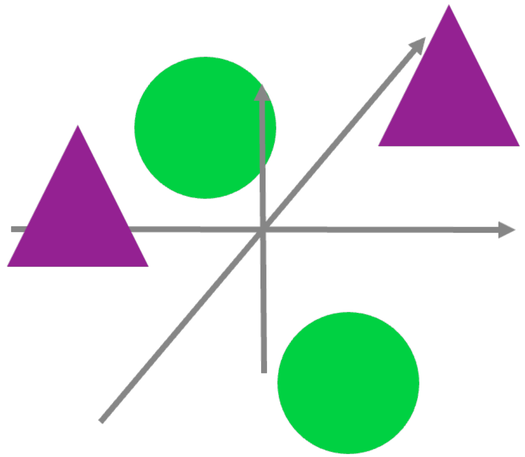


# Recap

- **Solution.** Use some nonlinear feature $\Phi(\cdot)$, and solve

$$\min_{f:\text{linear}} \mathbb{E}[\ell(f(\Phi(\mathbf{x})), y)]$$

  - The features are typically **handcrafted**
    - Example: Gaussian kernels, Polynomial kernels
    - Requires much domain expertise



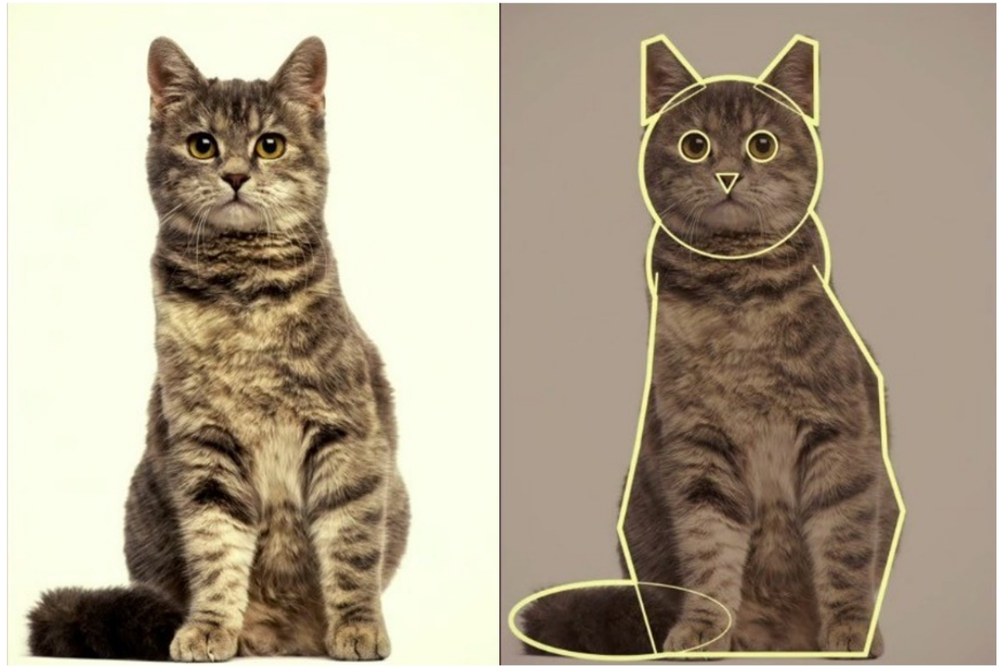
$(x_1, x_2, x_1 x_2)$

# Handcrafted Features

# Handcrafted features

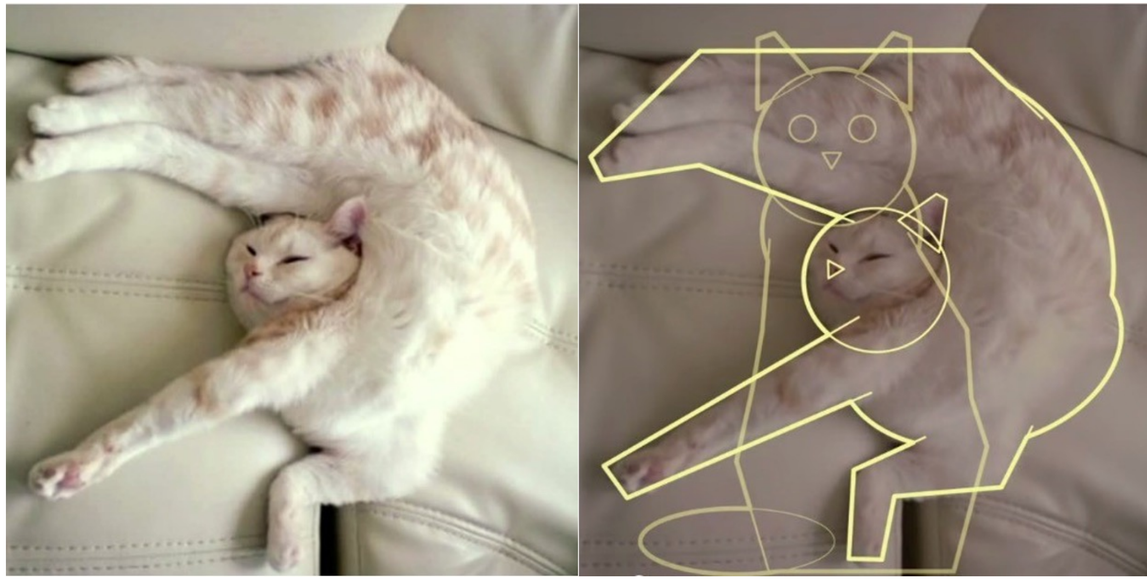
- Consider a high-dimensional and complicated data (e.g., text, images)
  - Designing a good feature can be very challenging

- Ideally, we would want something like:
  - $\phi_1(\mathbf{x}) = 1$ if ”round head”
  - $\phi_2(\mathbf{x}) = 1$ if ”two triangular ears”
  - $\phi_3(\mathbf{x}) = 1$ if ”oval tail”
  - …
  - $f(\cdot)$: Average all, and apply threshold



# Handcrafted features

- **Problem.** Identifying useful $\phi(\cdot)$ is very difficult
  - Data is too diverse!

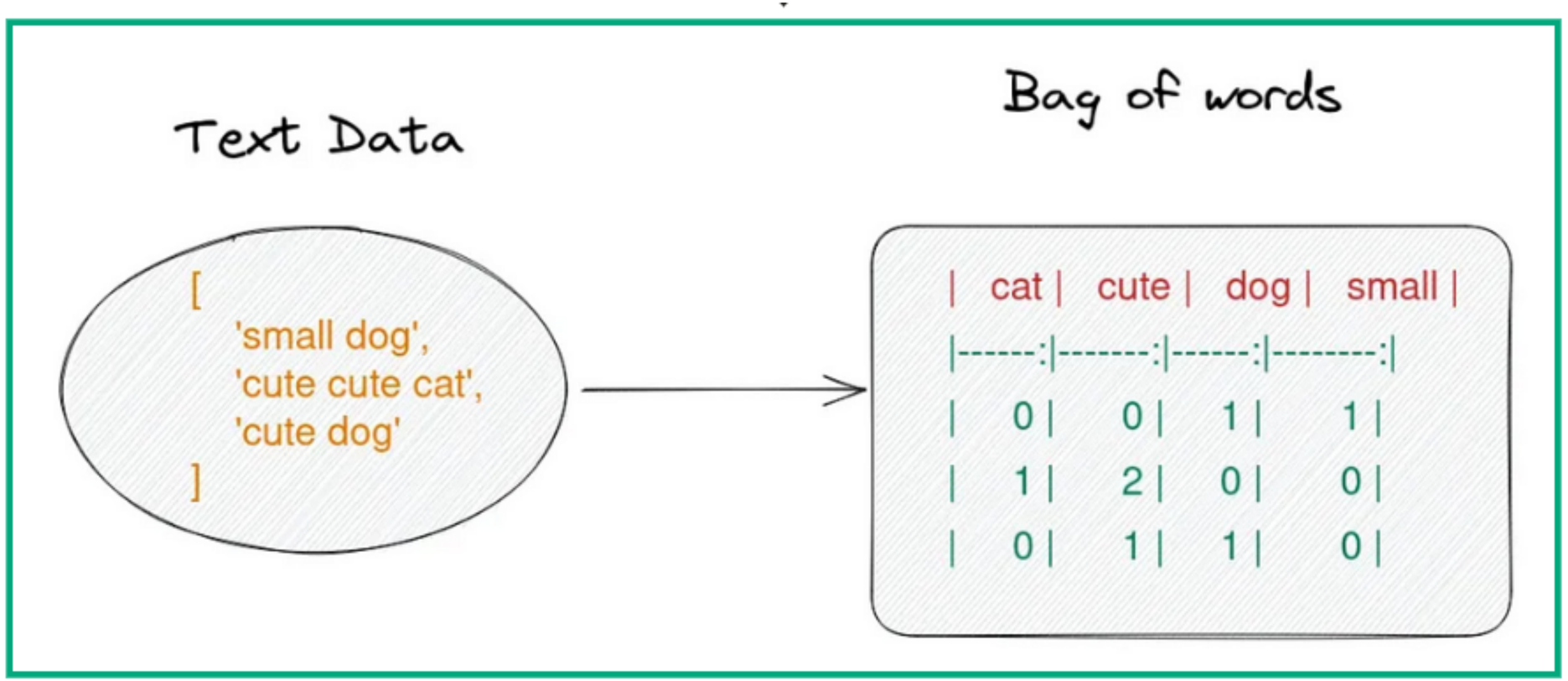


  - $\phi_1(\mathbf{x}) = 1$ if "round head" ✅
  - $\phi_2(\mathbf{x}) = 1$ if "two triangular ears" ❌
  - $\phi_3(\mathbf{x}) = 1$ if "oval tail" ❌

- Furthermore, constructing each $\phi$ is challenging as well
  - How do we construct an "oval tail detector"?

# Handcrafted features

- **Example (text).** Bag-of-Words
  - Count the frequency of words in a sentence
  - **But.** Ignores word ordering, e.g., “man bites dog” vs. “dog bites man”



# Handcrafted features

- **Example (text).** n-grams
  - Count the frequency of multi-word blocks
  - **But.** Limited “window” — cannot cover, e.g., detective novels
  - **But.** Dictionary size explodes — curse of dimensionality

| 1-Gram | 2-Gram | 3-Gram |
|---|---|---|
| The | The Margherita | The Margherita pizza |
| Margherita | Margherita pizza | Margherita pizza is |
| pizza | pizza is | pizza is not |
| is | is not | is not bad |
| not | not bad | not bad taste |
| bad | bad taste | |
| taste | | |

# Handcrafted features

- **Example (vision).** Sobel filters
  - Apply patches edge-detecting kernels to extract “shape”
  - **But.** Difficult to capture semantics
  - **But.** Difficult to be processed by linear models

$$G_x = \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \frac{1}{2}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

# Historical Bits: ImageNet

# ImageNet Challenge

- In 2010, computer vision folks turned this into a competition
  - called “ImageNet”
- **Training data.** million+ images
  - Using crowdsourced annotations, via Amazon Mechanical Turk
- **Classes.** 1000+ classes

# ImageNet Challenge

- **2010**
  - **1st.** NEC-UIUC 72% “SVM + SIFT features”
  - **2nd.** Xerox 66%
- **2011**
  - **1st.** Xerox, 74% “SVM + Fisher Kernels”
  - **2nd.** U of Amsterdam 69%
- **2012**
  - **1st.** SuperVision 84% “Deep Learning”
  - **2nd.** U of Tokyo 74% “SVM + Fisher Kernel + SIFT”
  - **3rd.** Oxford VGG 75% “SVM + Fisher Kernel + SIFT”

# Deep Learning

# Representation Learning

- Deep learning is one way to conduct **representation learning**
  - Select the feature $\Phi(\cdot)$ in a **data-driven** manner
  - So far, we have been solving the following optimization problem

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \mathbb{E}_{\text{data}}[\ell(f(\Phi(\mathbf{x})), y)]$$

Human trial-and-error 😢

Automated optimization, with data

# Representation Learning

- Deep learning is a way to conduct **representation learning**
  - Select the feature $\Phi(\cdot)$ in a **data-driven** manner
  - So far, we have been solving the following optimization problem

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \mathbb{E}_{\text{data}}[\ell(f(\Phi(\mathbf{x})), y)]$$

Automated optimization, with data

- **Idea.** Learn both $\Phi(\cdot), f(\cdot)$ from data
  - Either jointly or separately

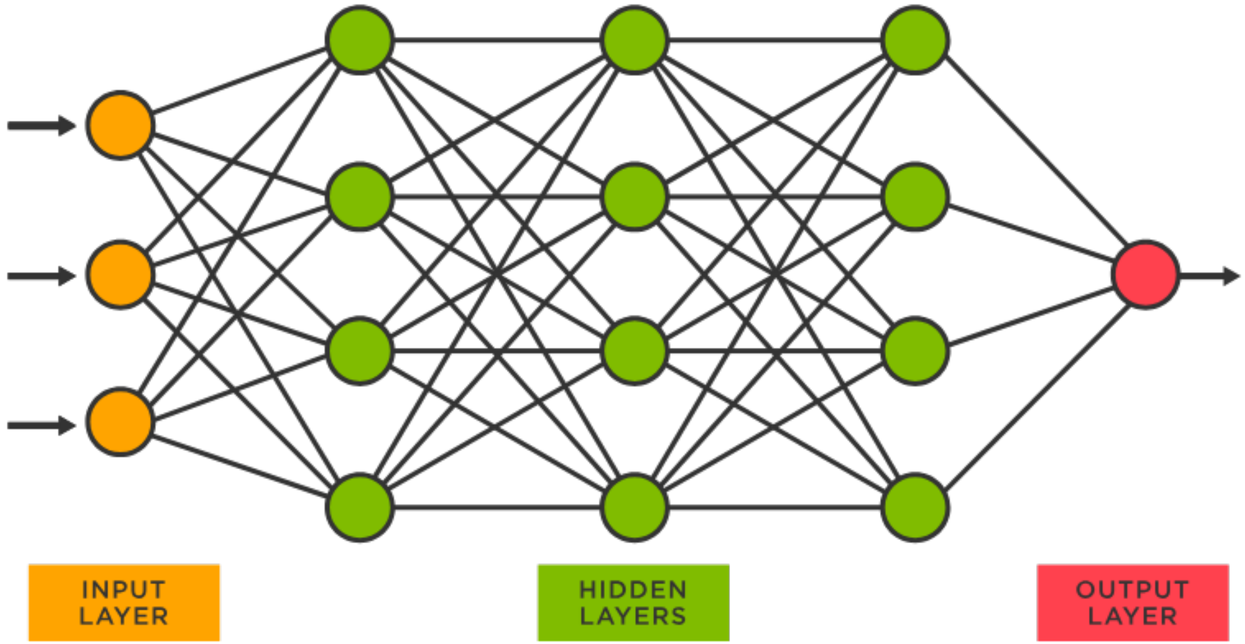
# Key questions

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \mathbb{E}_{\text{data}}[\ell(f(\Phi(\mathbf{x})), y)]$$

- For a successful representation learning, we need three things:
  - A good **search space** of $\Phi(\cdot)$
    - Should be able to express wide range of (nonlinear) functions
  - A good **optimization algorithm** to find optimal $\Phi(\cdot)$
    - A global optimum should be discoverable with small compute
  - A good **generalizability** of the discovered solution
    - Hope to suffer from small “overfitting”

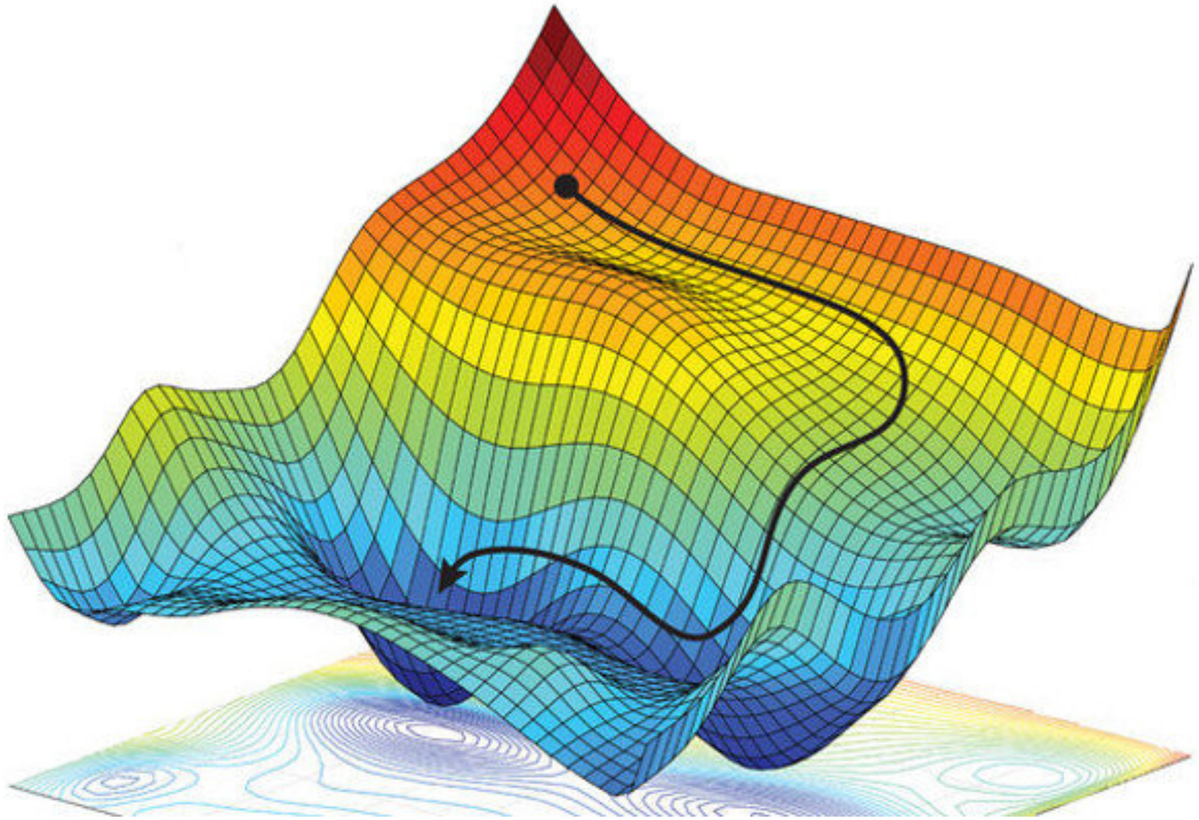
# Deep Learning

- Starting from today, we will focus on **deep learning**
- **Search space.** Neural networks
  - Repetition of simple operations
  - Can approximate any continuous function, given sufficient size
    - so-called “universal approximation property”



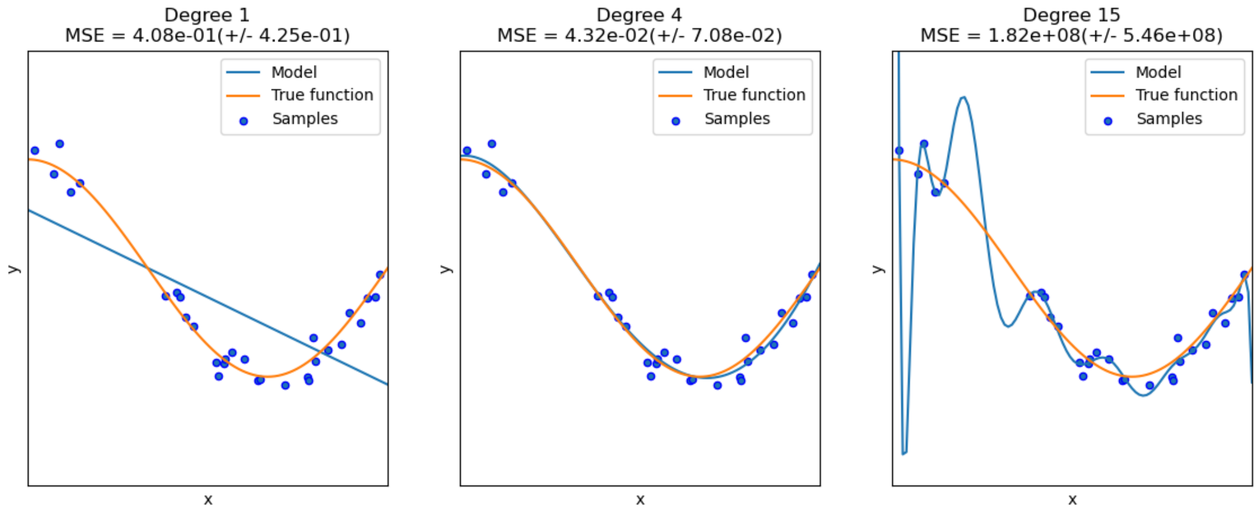
# Deep Learning

- **Optimizer.** Gradient descent + Backpropagation
  - The search space is highly nonconvex
  - Mysteriously, however, GD finds a good minima

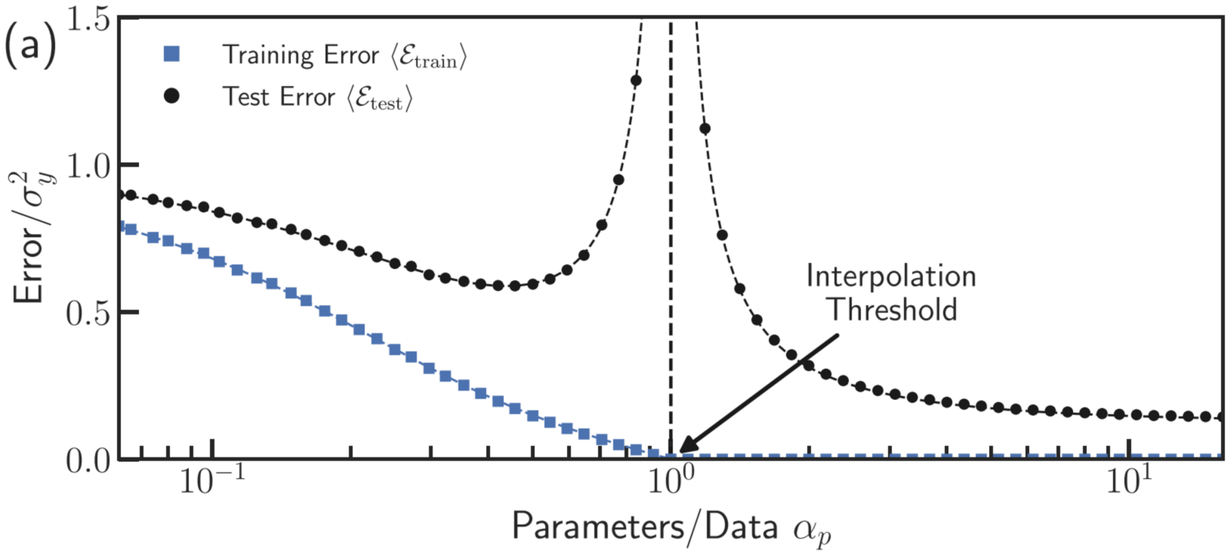


# Deep Learning

- **Generalization.** Mysteriously, generalize well…
  - Folk knowledge. Larger models generalize poorly
  - Deep learning. Larger models tend to generalize better




**Traditional ML**




**Deep Learning**

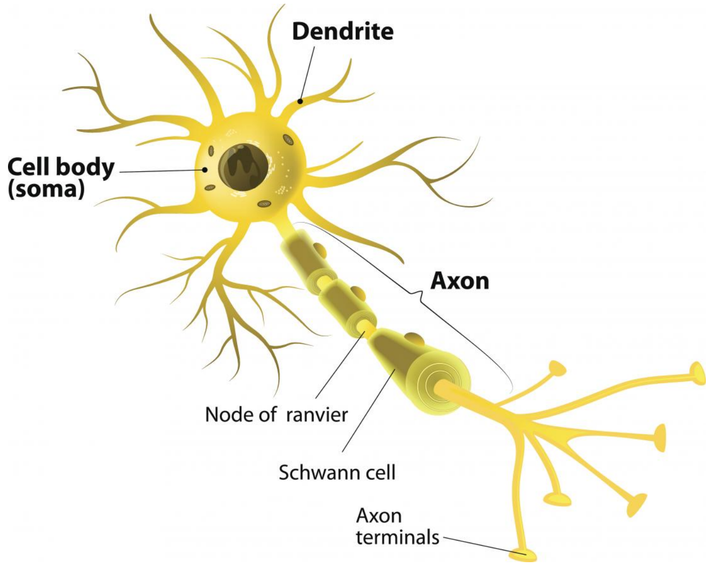
# Deep Learning

- This week, we give a brief overview of these three aspects
  - Search space
  - Optimization
  - Generalization

# Deep Neural Networks

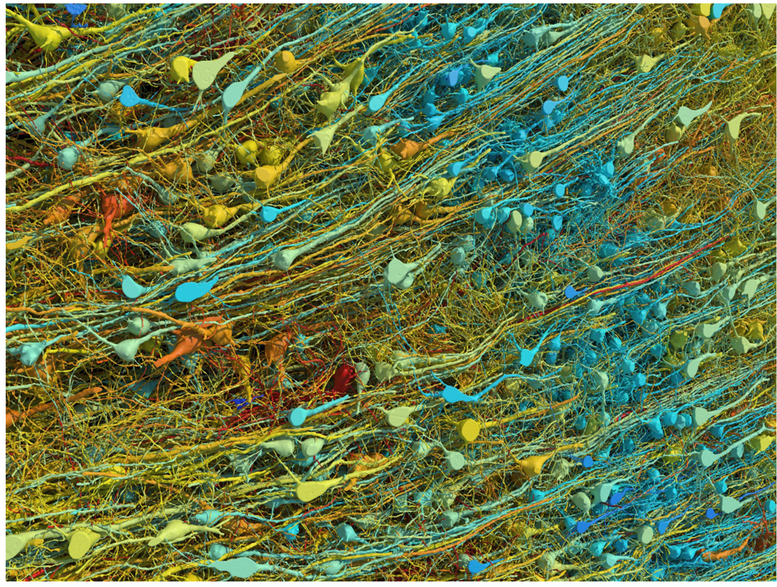
# Neural Network

- Motivated by how human processes information



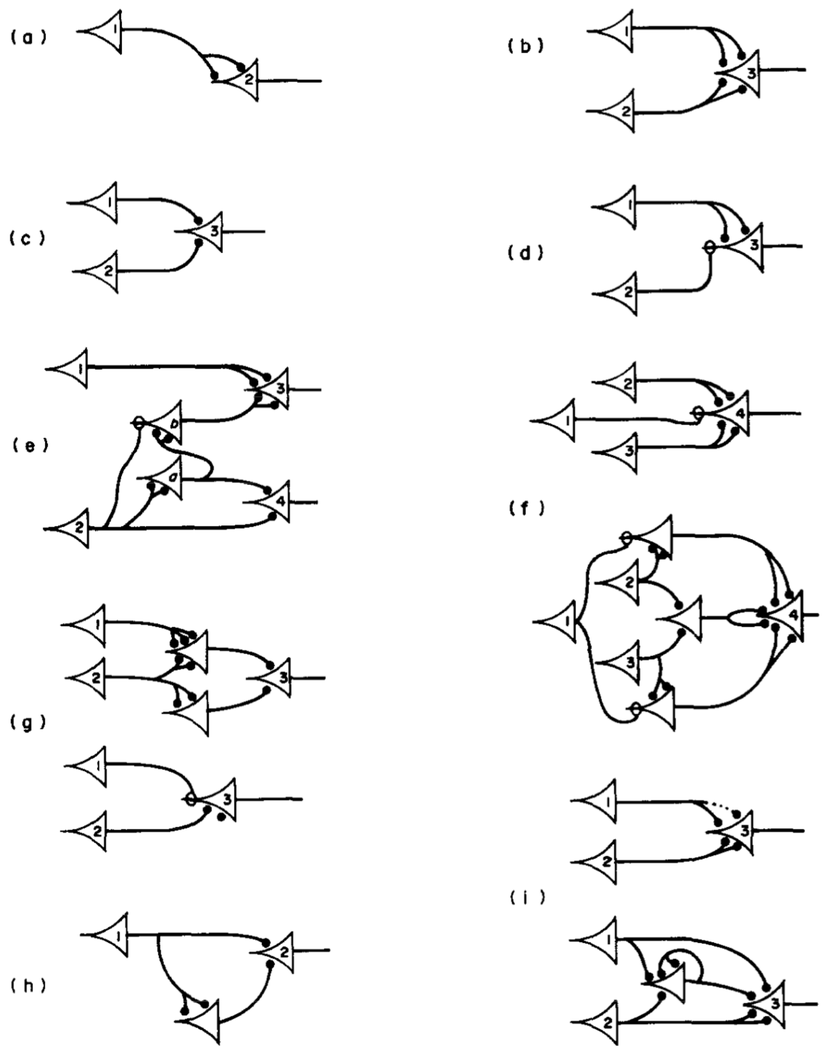

- **Neurons.** Unit of information processing
  - Aggregates electrochemical signals from other neurons
  - Fire the output, if the aggregated signal is above threshold

- **Brain.** A network of neurons
  - Can do complicated operations (a living proof-of-concept)



# Artificial Neural Network

- In 1950s, used for hard-coding functions rather than machine learning
  - Electric circuits, instead of biological
    - Build elementary circuits
    - Combine multiple circuits to express a complicated function




A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

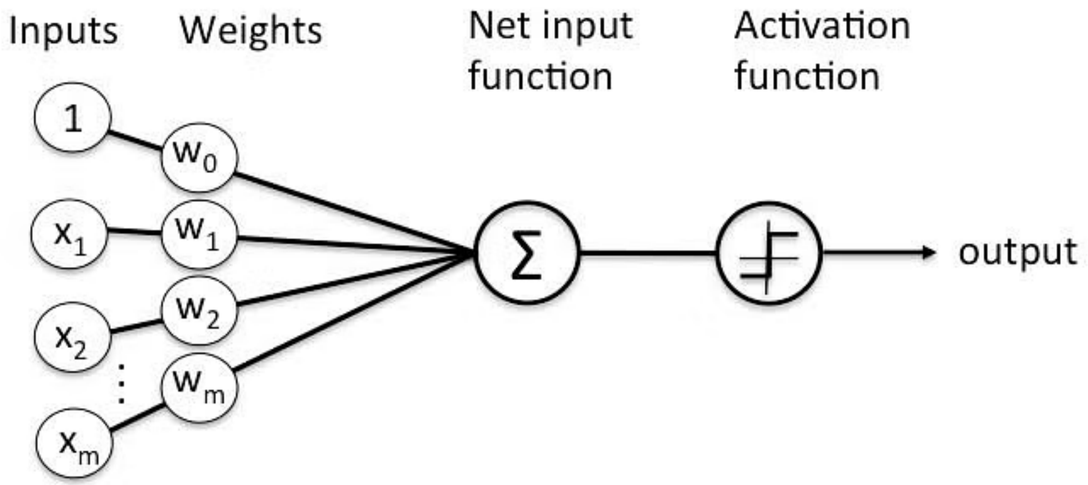
WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

# Perceptrons

- Neurons with learnable weights

$$y = \mathbf{1}\left( \sum_{i=1}^{n} w_i x_i + b \right)$$

  - Multi-dimensional input
  - Take a weighted sum
    - Fire if sum exceeds threshold
    - Otherwise, output 0




- Combination of a linear operation $\mathbf{x} \mapsto \mathbf{w}^\top \mathbf{x}$

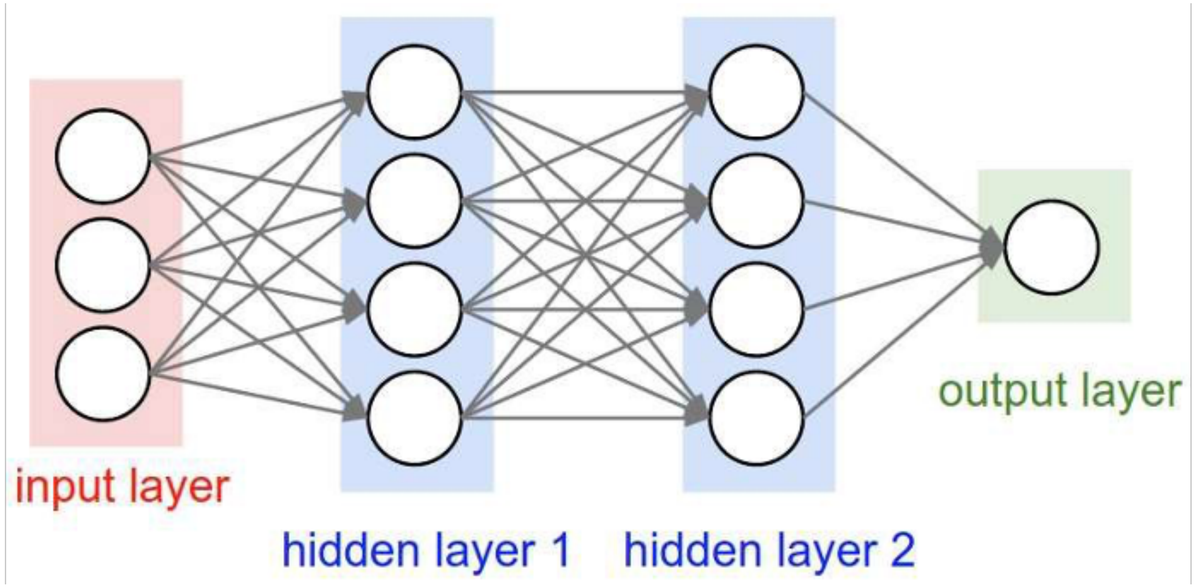
  a nonlinearity $\mathbf{x} \mapsto \mathbf{1}\{\mathbf{x} > 0\}$

# Multilayer Perceptrons

- **Idea.** Build a network of perceptrons, by parallel & serial connections
  - In the i-th, layer, do:
    - Linear operation $\mathbf{z} \mapsto \mathbf{W}_i\mathbf{z} + \mathbf{b}_i$
    - Nonlinear operation $\mathbf{z} \mapsto \sigma_i(\mathbf{z})$

weights: $\mathbf{W}_i$ biases: $\mathbf{b}_i$

activation function (not necessarily $\mathbf{1}$)



# Multilayer Perceptrons

- Our overall predictor can be written as:

$$f(\mathbf{x}) = \mathbf{W}_L \sigma_{L-1}(\mathbf{W}_{L-1}\sigma(\cdots\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \cdots))$$

linear classifier

feature

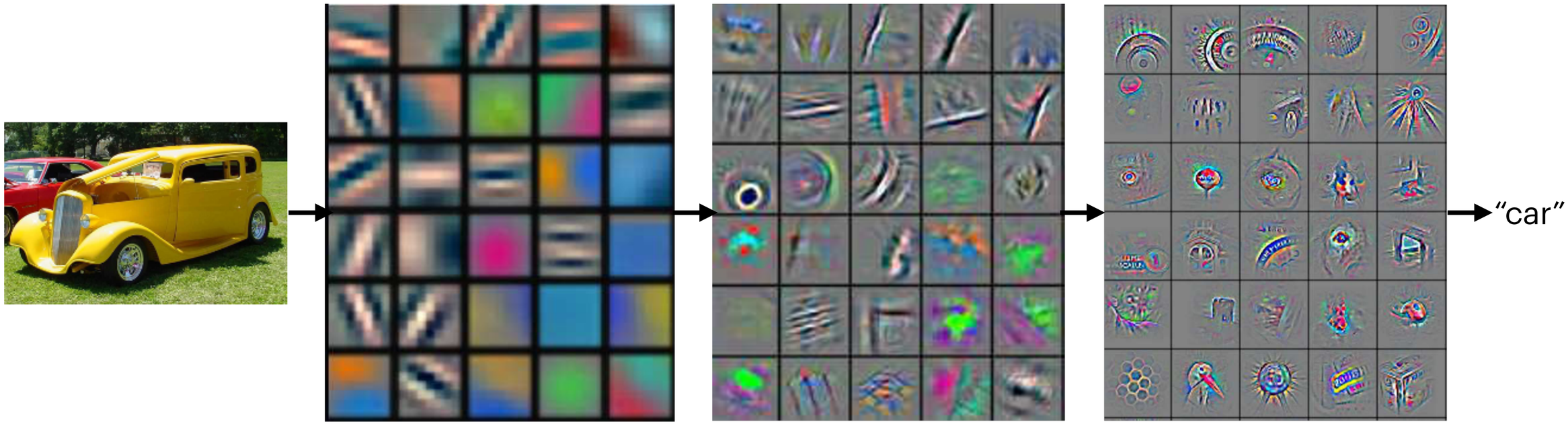
- **Note.** If we did not have activation functions, this is simply a linear model

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{W}_L(\mathbf{W}_{L-1}\cdots(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\cdots) \\ &= \tilde{\mathbf{W}}\mathbf{x} + \tilde{\mathbf{b}} \end{aligned}$$

# Multilayer Perceptrons

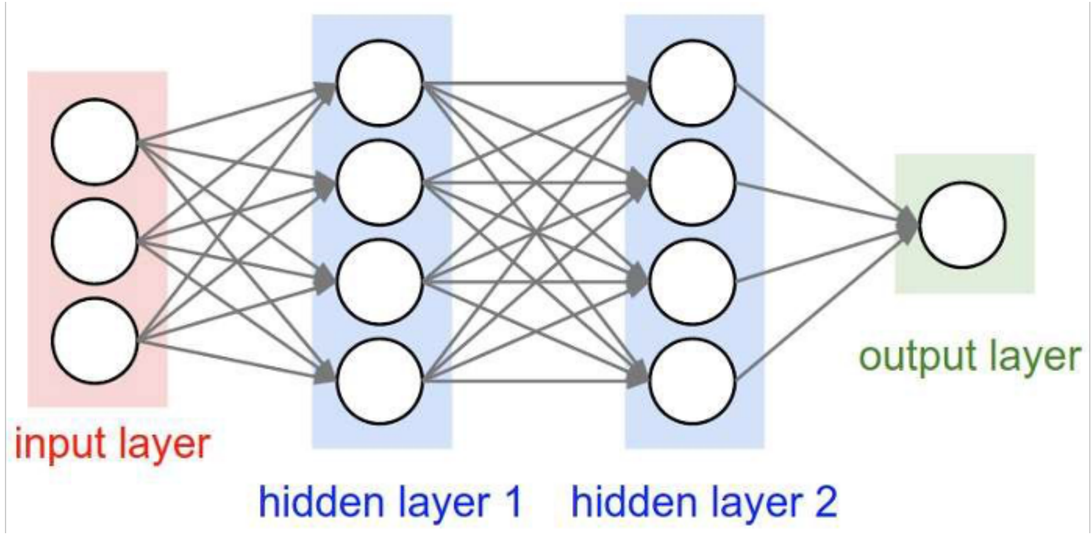
- As the layer gets deeper, the each neuron captures more and more complicated patterns

- **Example.** Visualization of the pattern that maximally activates a neuron



# Properties

- **Computation.** Very easy to compute
  - Mostly linear operations
    - We can do parellel processing w/ GPUs
- **Flexibility.** Modular, and thus very flexible
  - Can replace the “building blocks”
  - Can change the size — add or remove layers/neurons



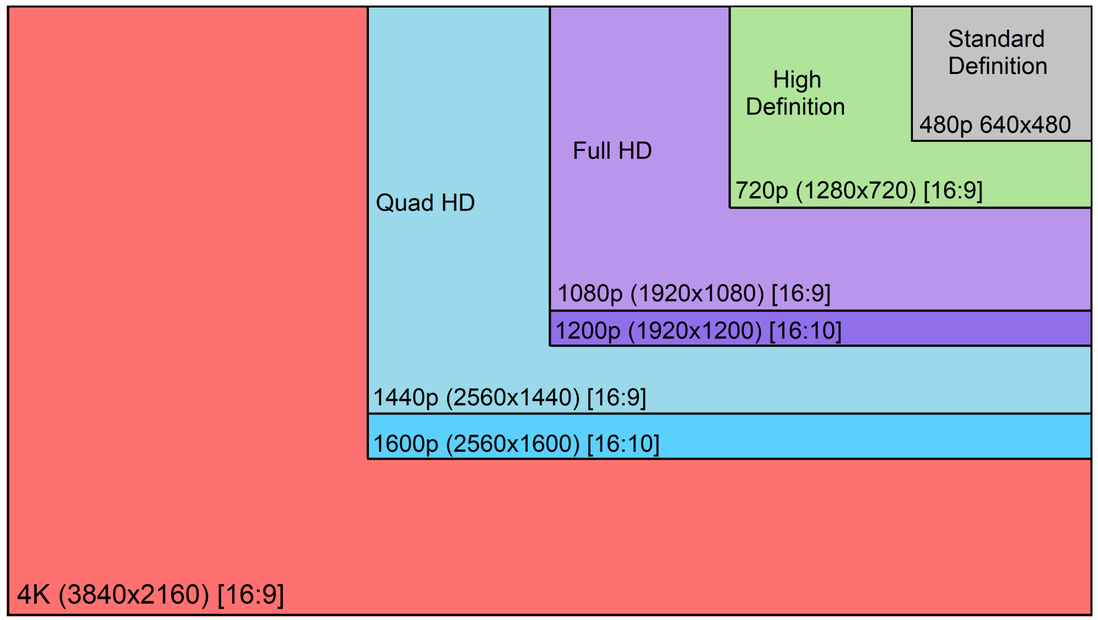
# Building blocks

- There are countless network architectures, employing:
  - Other linear operations, instead of simple matrix multiplication
    - e.g., convolution
  - Other nonlinear operations, instead of $\mathbf{1}\{\cdot\}$
    - e.g., ReLU, sigmoid

- **Advanced.** Many other operations
  - Residual connection
  - self-attention
  - normalization, …

# Building blocks

- **Linear ops.** Vanilla linear is bad, especially for high-dimensional data
  - Example 1. For 4K images, a 2-layer net with 1000 hidden neurons need 7.5 billion parameters (30GB of memory)
  - Example 2. For texts, the input can be of variable length




**Până la mijlocul lui iulie, procentul a urcat la 40%.** La începutul lui august, era 52%.

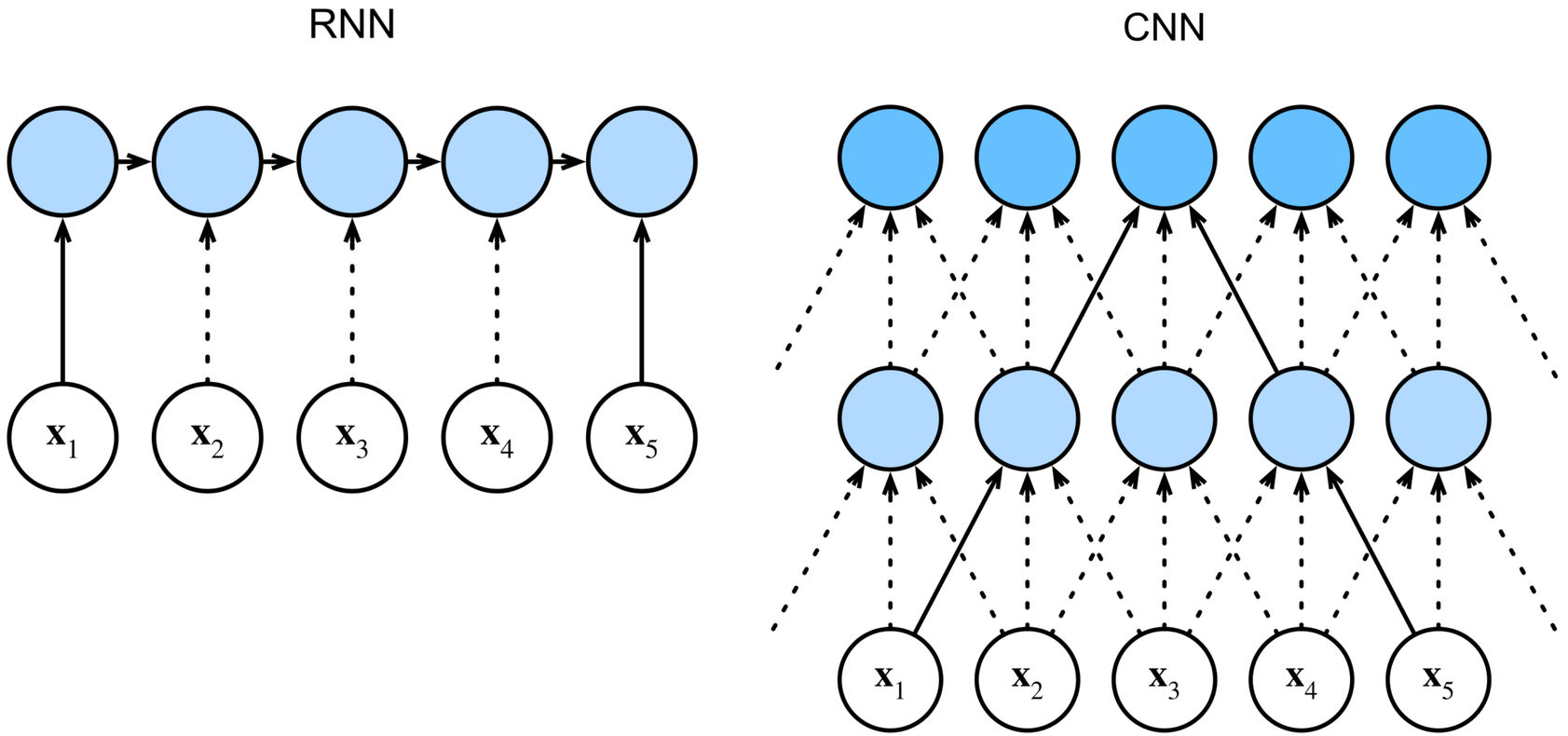
— Source

**By mid-July, it was 40 percent.** In early August, it was 52 percent.

— Reference

# Building blocks

- A common solution is to develop customized modules for each data domain
  - recurrent module for text, convolutional module for vision
  - will be discussed in later lectures

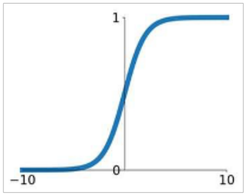


# Building blocks

- **Activation.** Difficult to conduct gradient descent with $\mathbf{1}\{\cdot\}$
  - Gradient is almost always $\mathbf{0}$
  - Alternatives are:
    - Saturating activations. Surrogates of $\mathbf{1}\{\cdot\}$
    - Non-saturating activations. Modern pick
      - Default: ReLU (Rectified linear unit)
        - Better optimization properties (discussed later)
        - Better computation: easy to compute the output & gradients

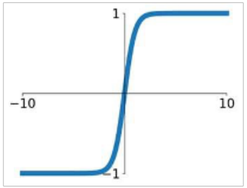
**Sigmoid**

$\sigma(x) = \frac{1}{1+e^{-x}}$



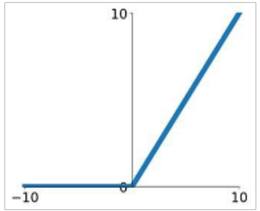

**Tanh**

$\tanh(x)$



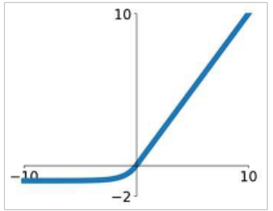

**ReLU**

$\max(0, x)$




**ELU**

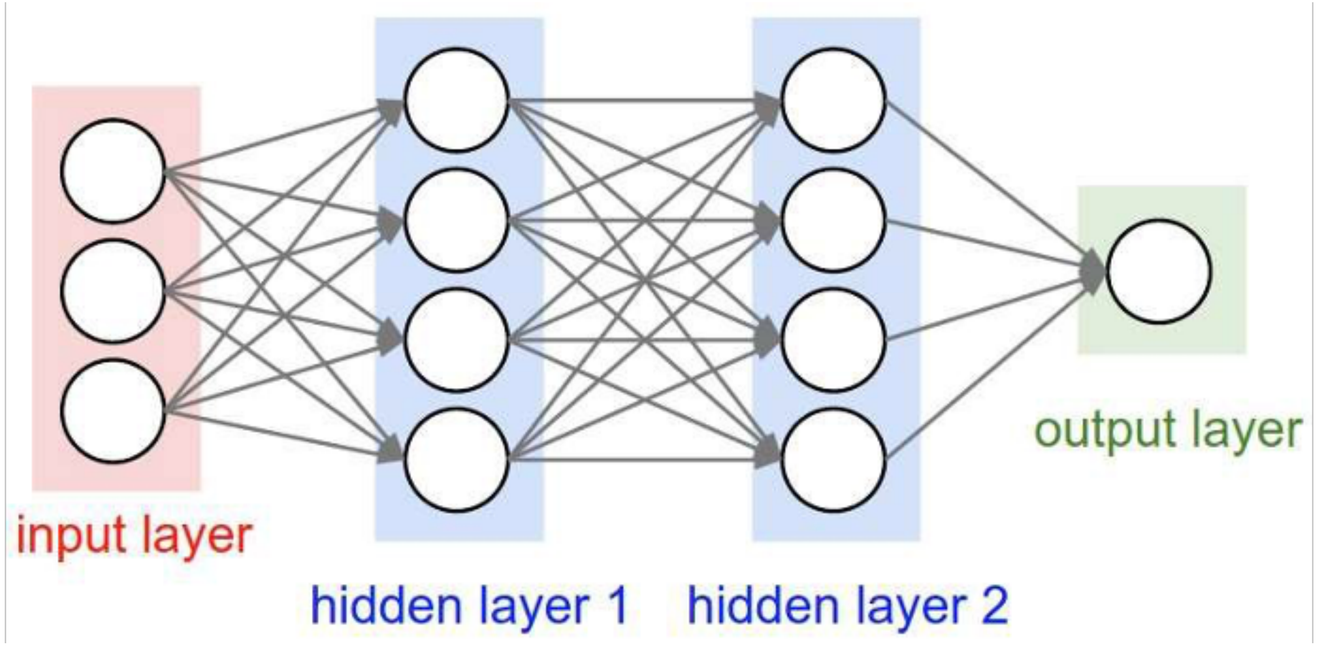
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Width and Depth

- **Width.** The number of neurons in each layer (typically the widest)
- **Depth.** The number of layers

- Exercise. How many layers does this network have?
  - What is its width?



# Width and Depth

- Larger the network, we typically have:
  - High training accuracy on complicated dataset
  - Less SGD steps needed to converge
  - More per-step computation
  - Higher inference cost
    - Width: Memory, Depth: Latency

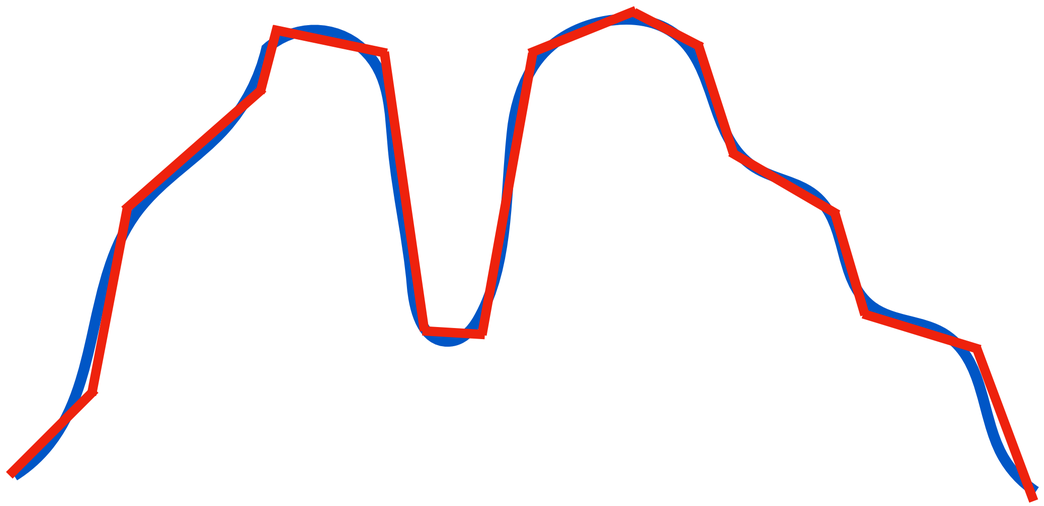
- In terms of generalization:
  - Width: Training accuracy ⬆ Generalization ⬆
  - Depth: Training accuracy ⬆ ⬆ Generalization ⬇

# Theoretical Properties: Approximation

# Universal Approximation

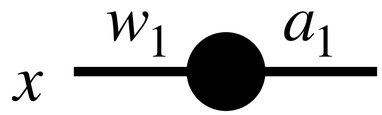
- A cool aspect of neural network is that they can represent any function
  - These properties are called “universal approximation properties”
- **Theorem.** Given any function $g(\cdot)$ and $\epsilon > 0$,
  one can find a <u>two-layer</u> ReLU neural network $f(\cdot)$ such that

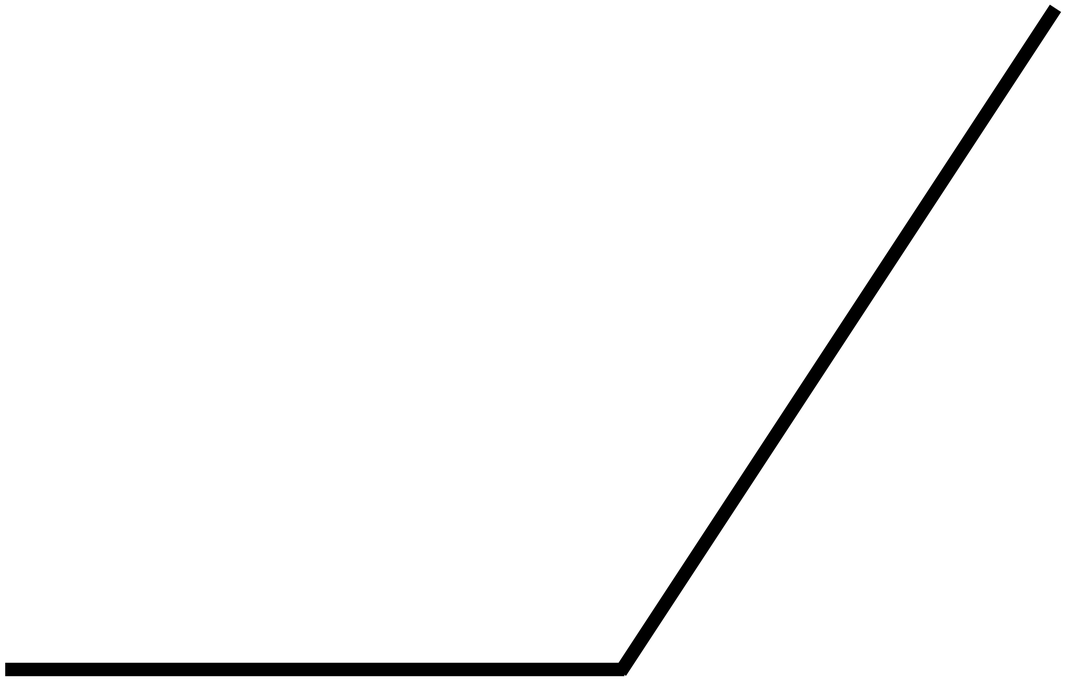
$$\sup_{x \in [0,1]} |g(x) - f(x)| \leq \epsilon$$



# Proof idea

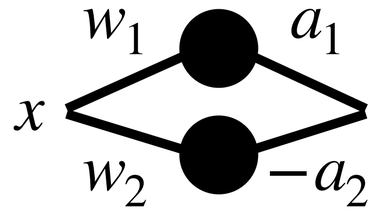
- A single ReLU neuron looks like this:

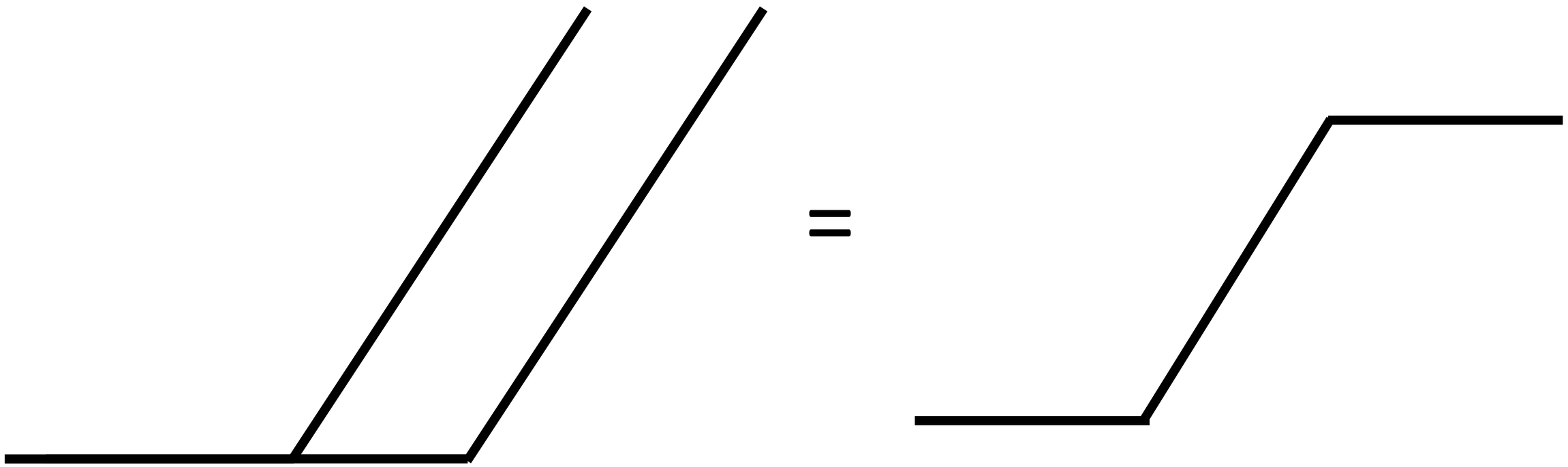


# Proof idea

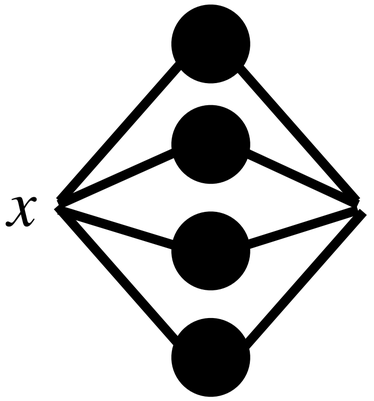
- Two parallel ReLU can make a hard sigmoid

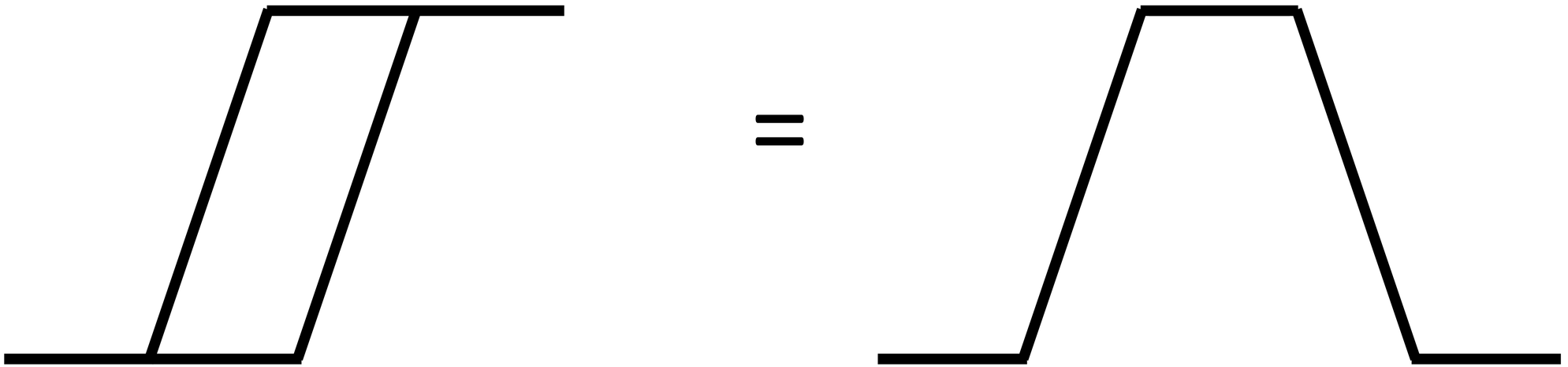


# Proof idea

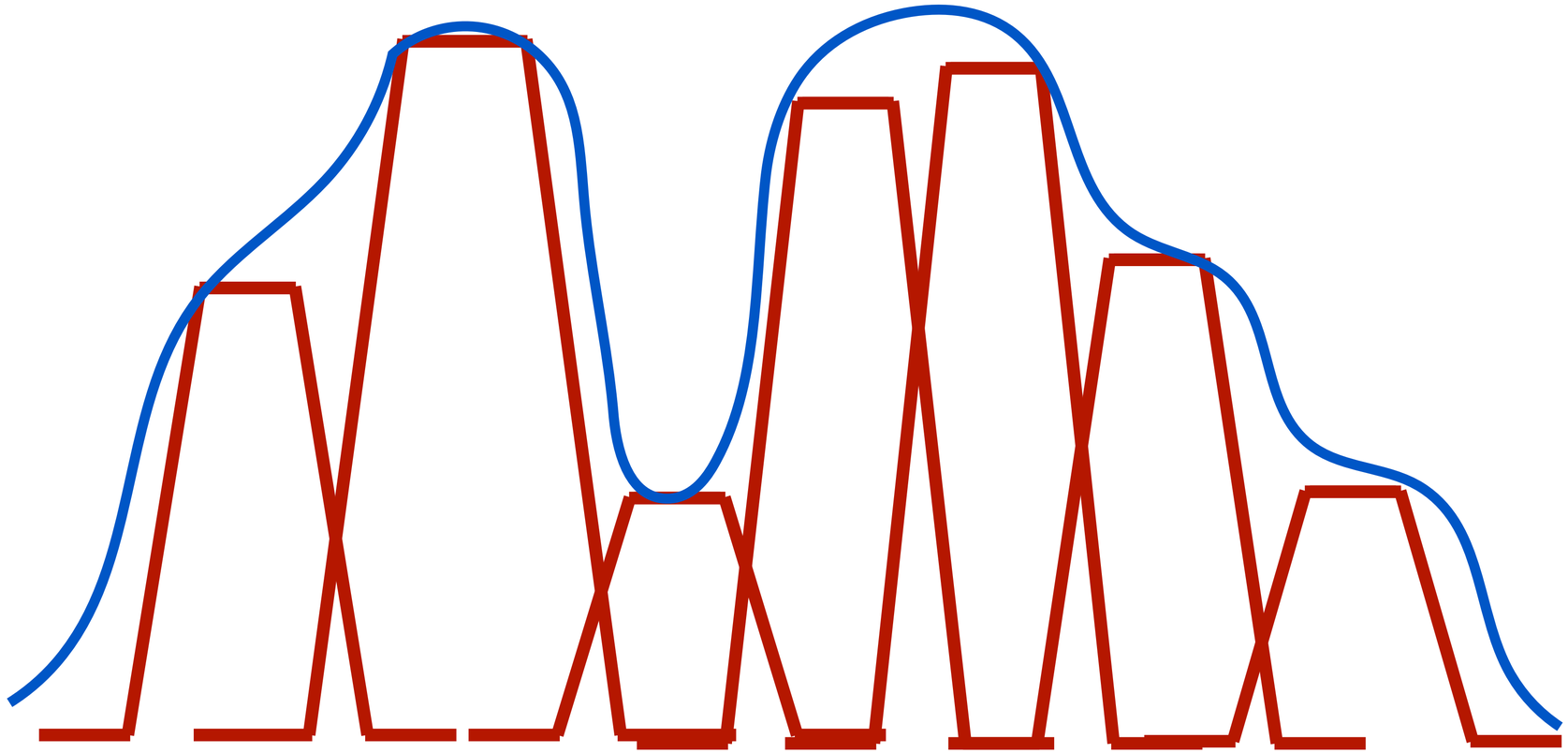
- Two hard sigmoid can make a bump
  - i.e., four ReLU neurons



# Proof idea

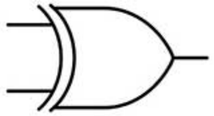
- We can use many narrow bumps to approximate the target function:
  - If we use N bumps, total 4N ReLU neurons needed



# Next up

- Optimization aspects
  - SGD
  - Backpropagation

- **Brainteaser**
  - Persuade yourself that width-2 depth-2 neural net with $\mathbf{1}\{\cdot\}$ activation can represent XOR




| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

# </lecture 11>