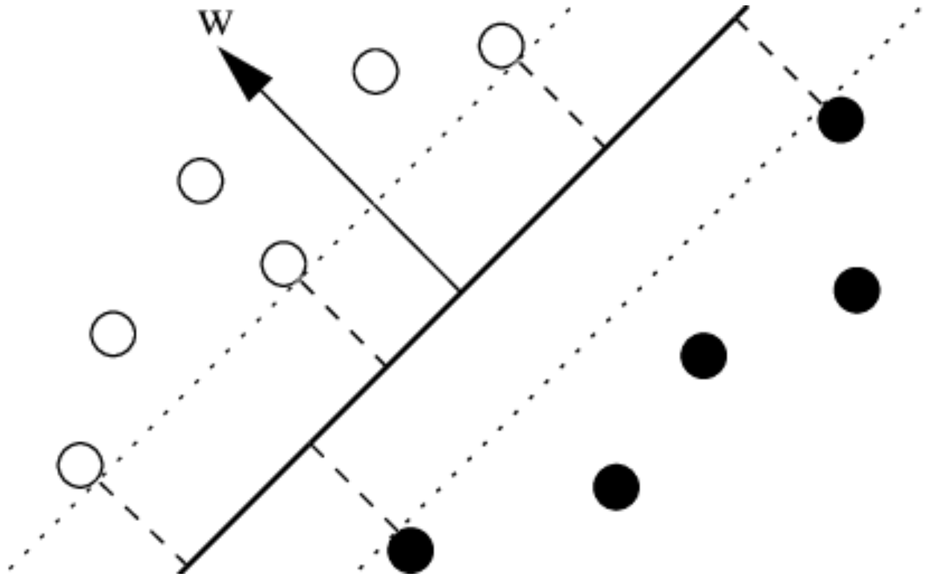
Deep Learning Overview

EECE454 Intro. to Machine Learning Systems

Fall 2024

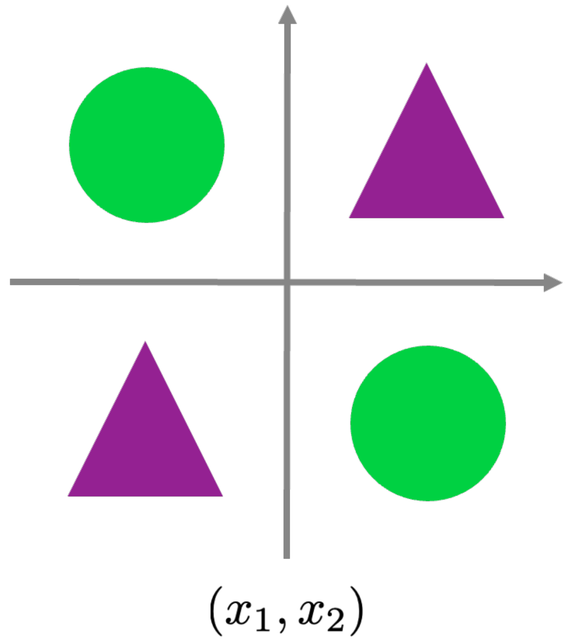
Recap: Linear Model

- We have studied many **linear models**
  - Perceptrons, Logistic regression, Linear regression, Dimensionality reduction, ...



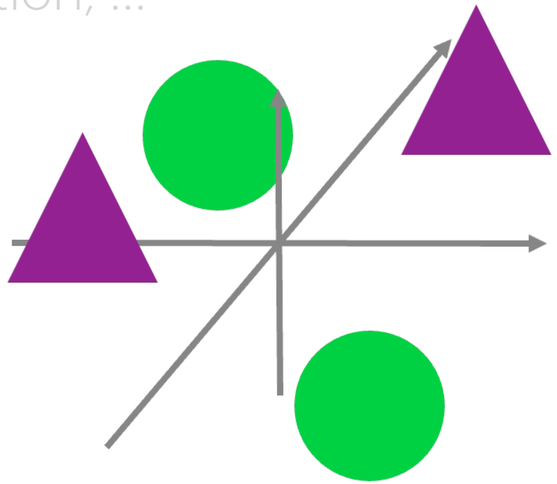
Recap: Linear Model

- We have studied many **linear models**
  - Perceptrons, Logistic regression, Linear regression, Dimensionality reduction, ...
  - Strengths. Easy to optimize, low inference cost
  - Limitations. Limited expressive power
    - Requires linearly separable data



Recap: Linear Model

- We have studied many **linear models**
  - Perceptrons, Logistic regression, Linear regression, Dimensionality reduction, …
  - Strengths. Easy to optimize, low inference cost
  - Limitations. Limited expressive power
    - Requires linearly separable data



- **Solution.** Use nonlinear feature map $\Phi(\cdot)$ that makes data linearly separable

$(x_1, x_2, x_1 x_2)$

  - A good model may exist in higher-dimensional spaces
  - We have studied many handcrafted features… which may not be effective for complicated data

Image source: CS231n @ Stanford

Visual Features

- Suppose that we train a **cat detector**
  - Use some domain knowledge to build good features

$\phi_1(\mathbf{x}) =$ "round head"

$\phi_2(\mathbf{x}) =$ "two triangular ears"

$\phi_3(\mathbf{x}) =$ "two round eyes"

$\phi_4(\mathbf{x}) =$ "oval tail"

$\phi_5(\mathbf{x}) = \ldots$

  - $f(\cdot)$: Declare cat if all are true

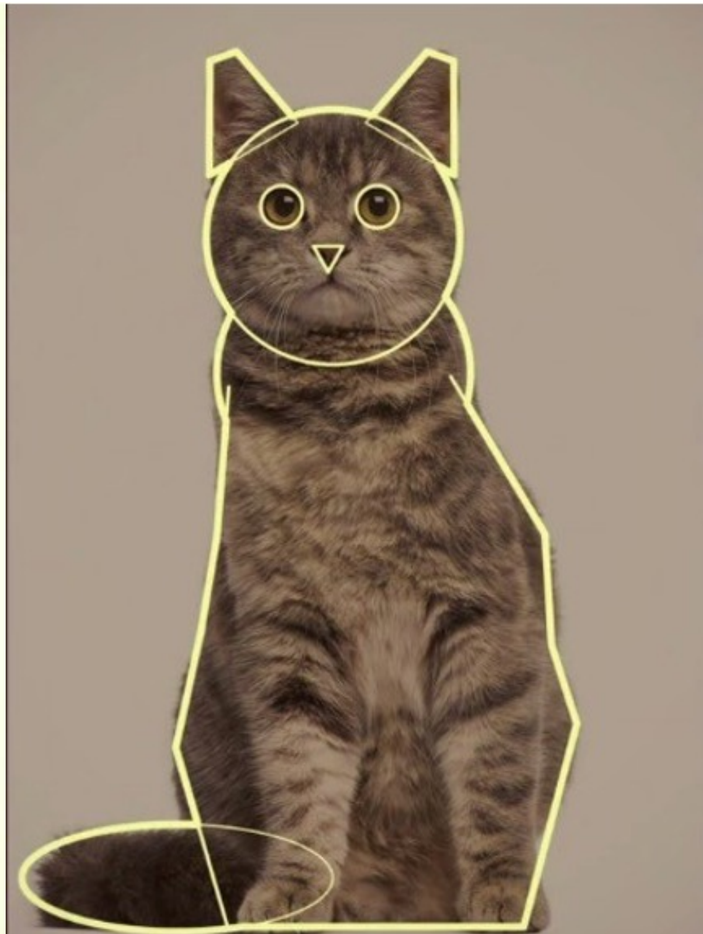


Image source: CS231n @ Stanford

Visual Features

- **Problem.** Given the variety of data, the selected features may be quite suboptimal.

$\phi_1(\mathbf{x}) =$ "round head" **O**

$\phi_2(\mathbf{x}) =$ "two triangular ears" **X**

$\phi_3(\mathbf{x}) =$ "two round eyes" **X**

$\phi_4(\mathbf{x}) =$ "oval tail" **X**

$\phi_5(\mathbf{x}) = \ldots$

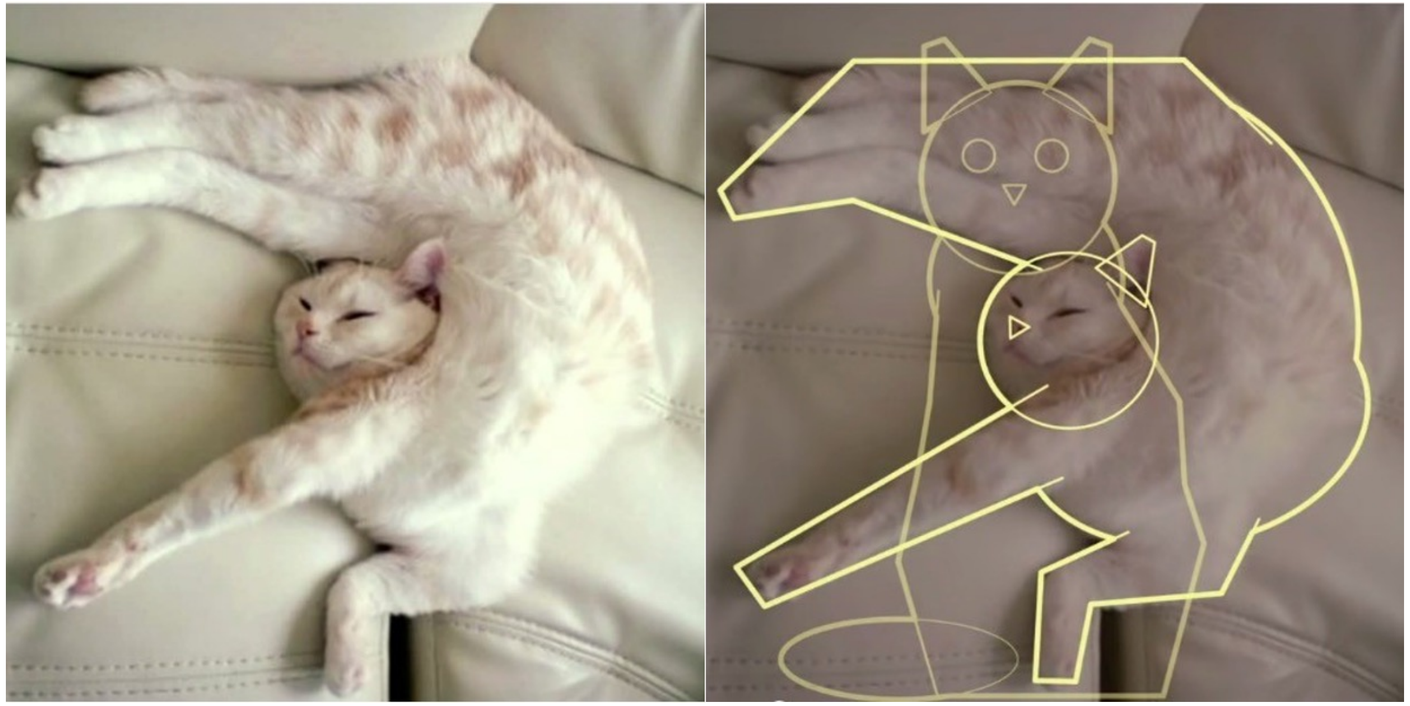


Image source: CS231n @ Stanford

Visual Features

- **Problem.** Given the variety of data, the selected features may be quite suboptimal.
  - We may want to build a cat detector that works well for any case… **how?**

Representation learning

- **Idea.** Also select the feature map $\Phi(\cdot)$ in a **data-driven** manner
  - Formally, we have been solving the following optimization problem

$$\min_{\Phi(\cdot)} \; \min_{\text{linear } f(\cdot)} \; \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(\Phi(\mathbf{x}_i))\Big)$$

Human trial-and-error 😢

Automated optimization, with data

Representation learning

- **Idea.** Also select the feature map $\Phi(\cdot)$ in a **data-driven** manner
  - Formally, we have been solving the following optimization problem

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(\Phi(\mathbf{x}_i))\Big)$$

Automated optimization, with data

- New paradigm. Learn both $\Phi(\cdot)$ and $f(\cdot)$ from data
  - Jointly optimize Φ, f with labeled data (if plenty)
  - Obtain Φ from unlabeled data, and train f with labeled data
  - Obtain Φ from unlabeled data, and train Φ, f with labeled data

Key questions

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(\Phi(\mathbf{x}_i))\Big)$$

- For a successful representation learning, we need **three things**
  - a good search space of $\Phi(\cdot)$
    - need to be able to express a very wide range of functions

Key questions

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(\Phi(\mathbf{x}_i))\Big)$$

- For a successful representation learning, we need **three things**
  - a good search space of $\Phi(\cdot)$
    - need to be able to express a very wide range of functions
  - a good optimization algorithm to find Φ
    - the global optimum can be found with low computational cost

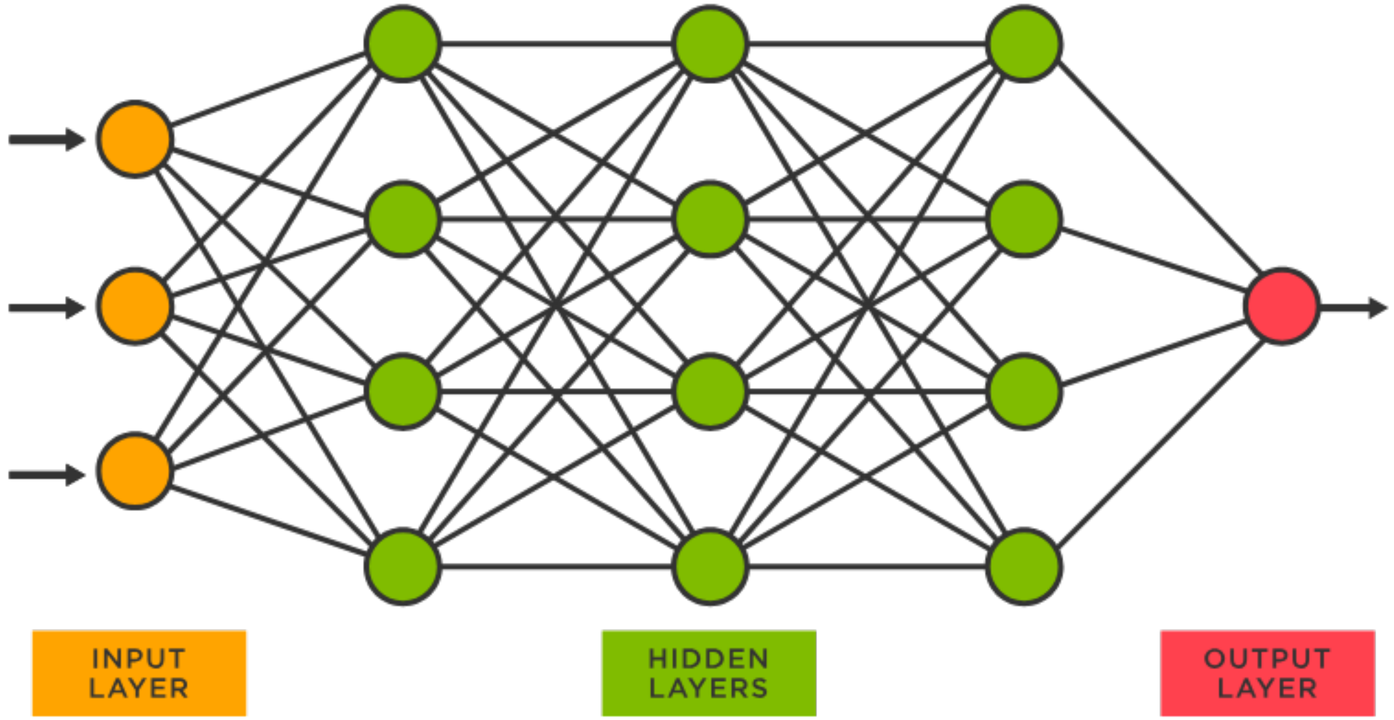
Key questions

$$\min_{\Phi(\cdot)} \min_{\text{linear } f(\cdot)} \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(\Phi(\mathbf{x}_i))\Big)$$

- For a successful representation learning, we need **three things**
  - a good search space of $\Phi(\cdot)$
    - need to be able to express a very wide range of functions
  - a good optimization algorithm to find Φ
    - the global optimum can be found with low computational cost
  - a good generalizability of discovered solution $\hat{\Phi}$
    - once we find a solution, we do not want it to overfit to the training data

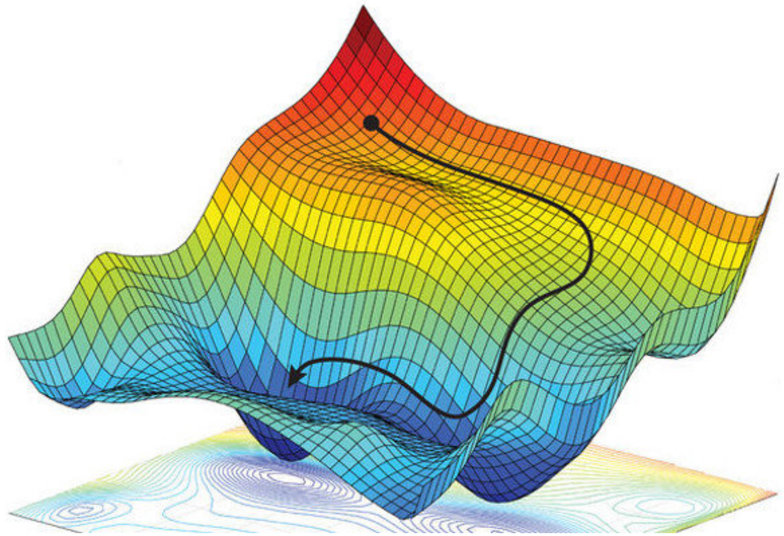
Deep Learning

- From today, we will focus on deep learning
  - **Search Space.** Neural networks
    - Neural networks with sufficient size can fit any continuous function (so-called “universal approximation theorem”)



Deep Learning

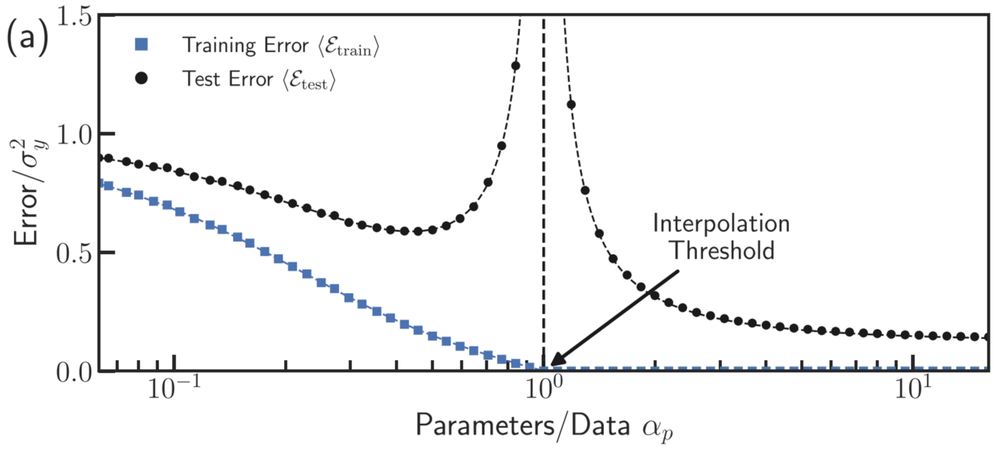
- From today, we will focus on deep learning
  - **Search Space.** Neural networks
    - Neural networks with sufficient size can fit any continuous function (so-called "universal approximation theorem")
  - **Optimization algorithm.** Gradient descent + Backpropagation
    - The search space is highly nonconvex, but mysteriously GD finds a good minima



Deep Learning

- From today, we will focus on deep learning
  - **Search Space.** Neural networks
    - Neural networks with sufficient size can fit any continuous function (so-called "universal approximation theorem")
  - **Optimization algorithm.** Gradient descent + Backpropagation
    - The search space is highly nonconvex, but mysteriously GD finds a good minima
  - **Generalization.** Also, mysteriously, using SGD leads to a good generalization
    - Counterintuitively, bigger networks tend to generalize better
    - Requires some hyperparameter tuning, though

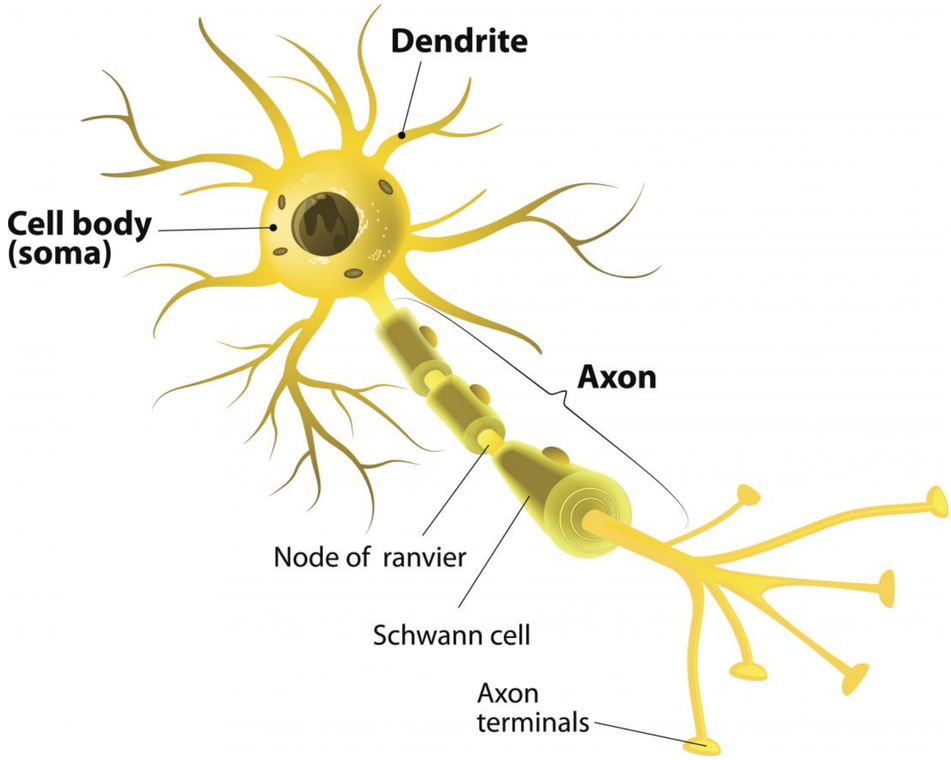
- **This week.** An overview on these aspects



Deep neural networks

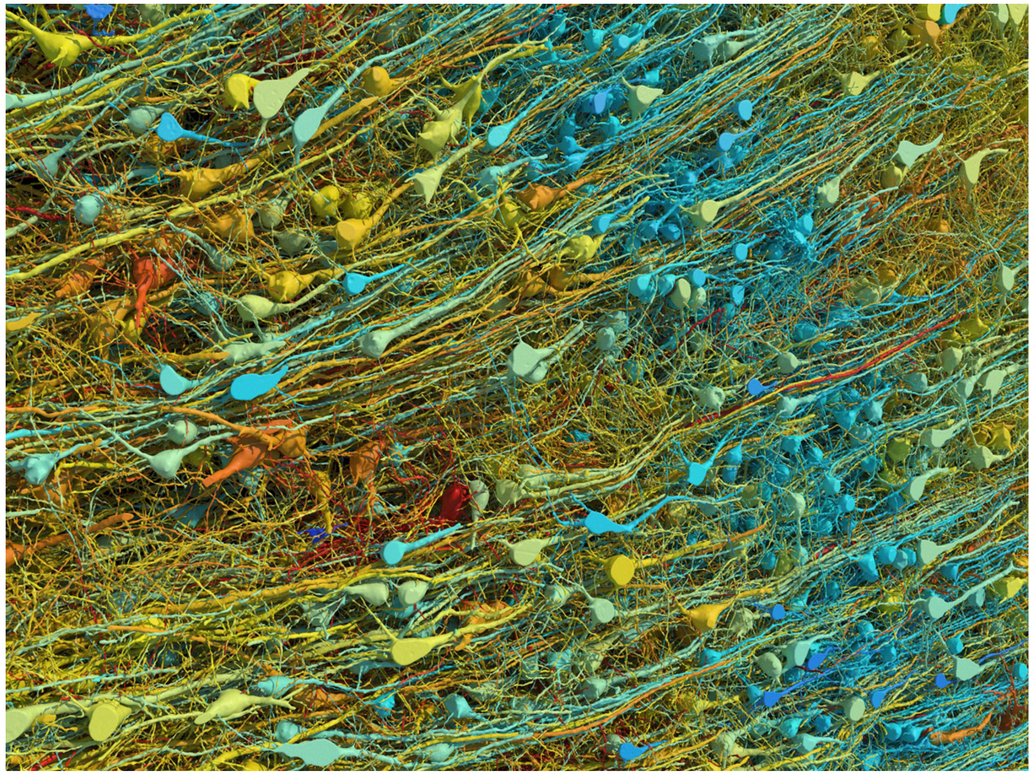
Neural network

- How human processes information
  - **Neurons** are a basic unit of information processing
    - Takes many input electrochemical signals
    - Aggregate the signals
    - Fire the pulse, if certain condition is met



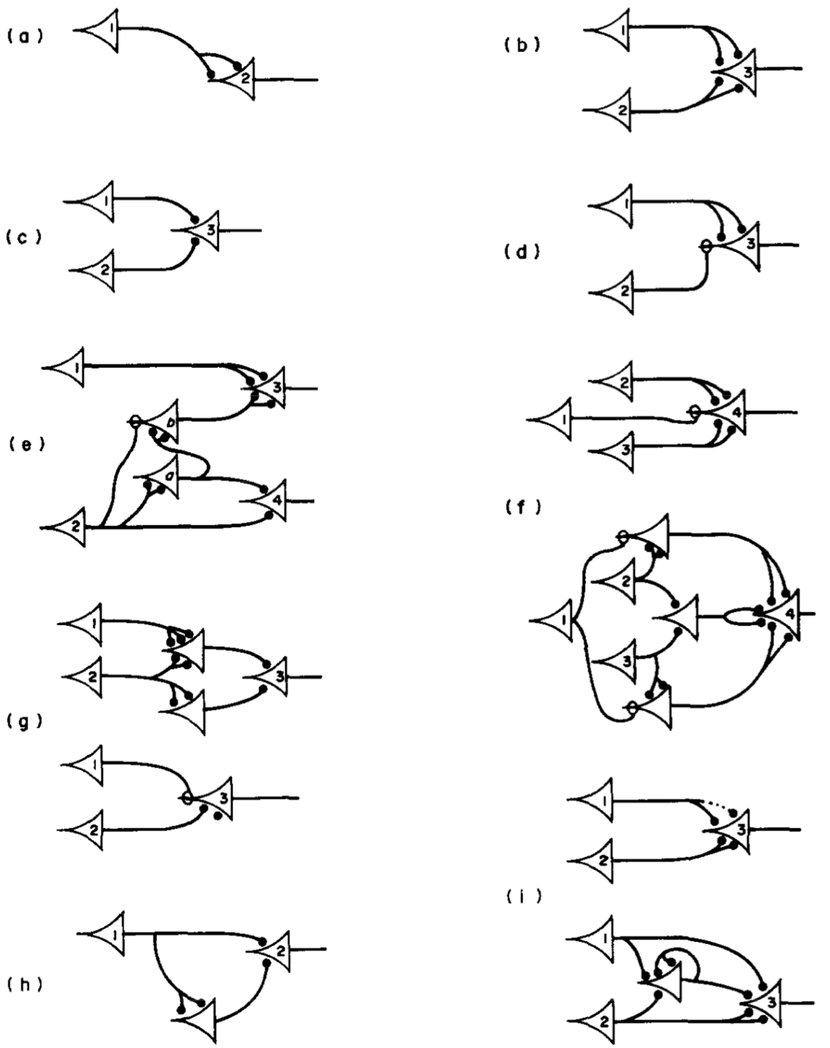
Neural network

- How human processes information
  - **Neurons** are a basic unit of information processing
    - Takes many input electrochemical signals
    - Aggregate the signals
    - Fire the pulse, if certain condition is met
  - Build a **neural network** to do complex operations
    - The signal flows through multiple layers of neurons, being sequentially processed



Artificial neural network

- Heavily motivated by the biological neurons
  - Use electric circuits instead
  - Build similar information-processing functions by combining multiple unit circuits



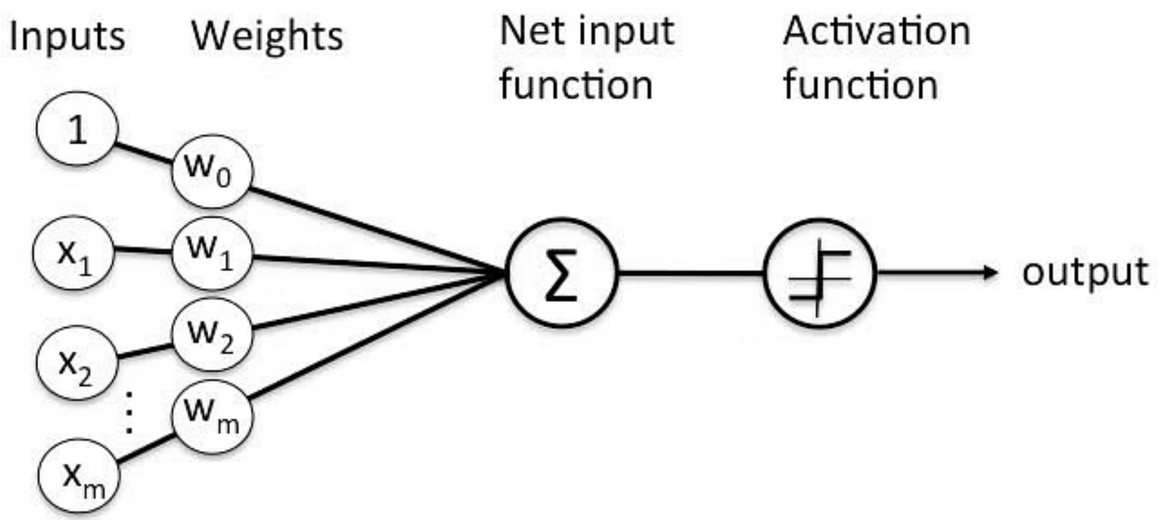
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ 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

- Each neuron is represented by a **perceptron**

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



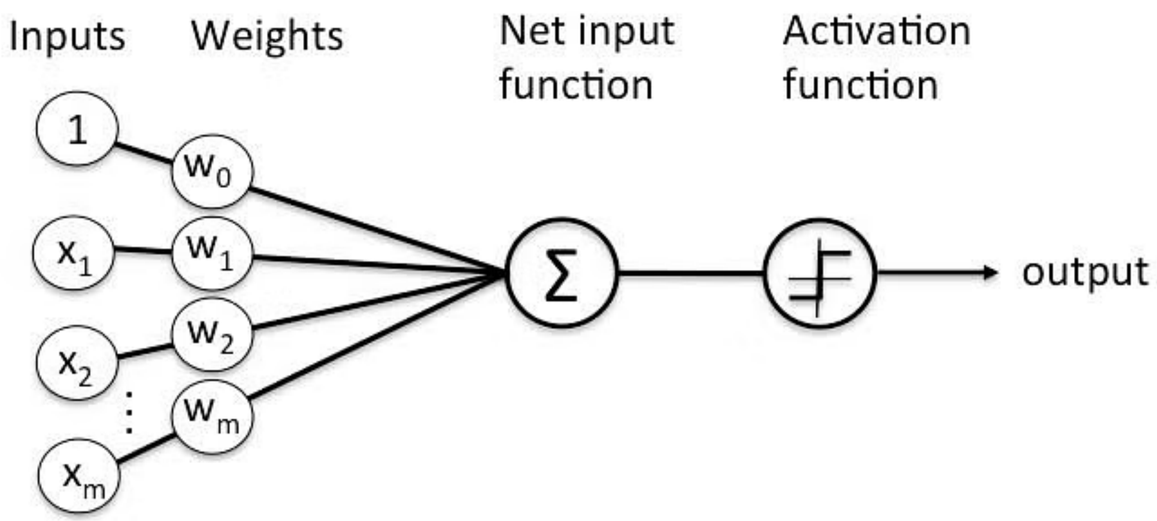
  - Takes multiple inputs
  - Take a weighted sum
    - If the result exceeds some value, fire (i.e., output 1).
    - Otherwise, output 0.

Perceptrons

- Each neuron is represented by a **perceptron**

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

  - Takes multiple inputs
  - Take a weighted sum
    - If the result exceeds some value, fire (i.e., output 1).
    - Otherwise, output 0.



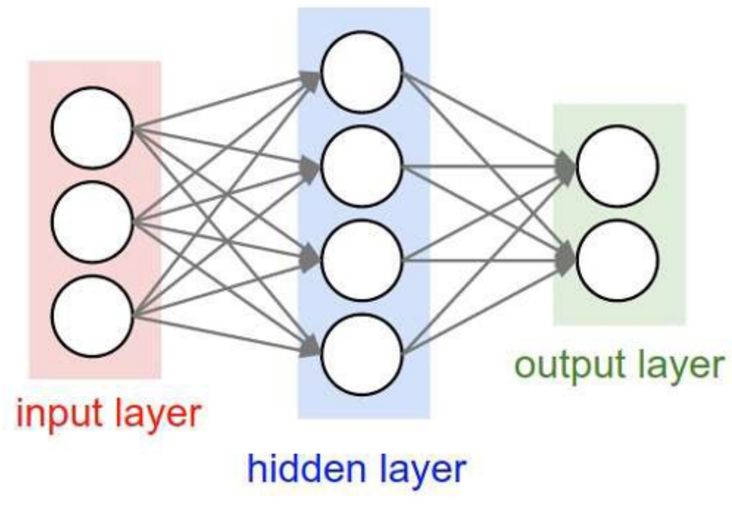
- A combination of a linear operation $\mathbf{x} \mapsto \mathbf{w}^\top \mathbf{x}$

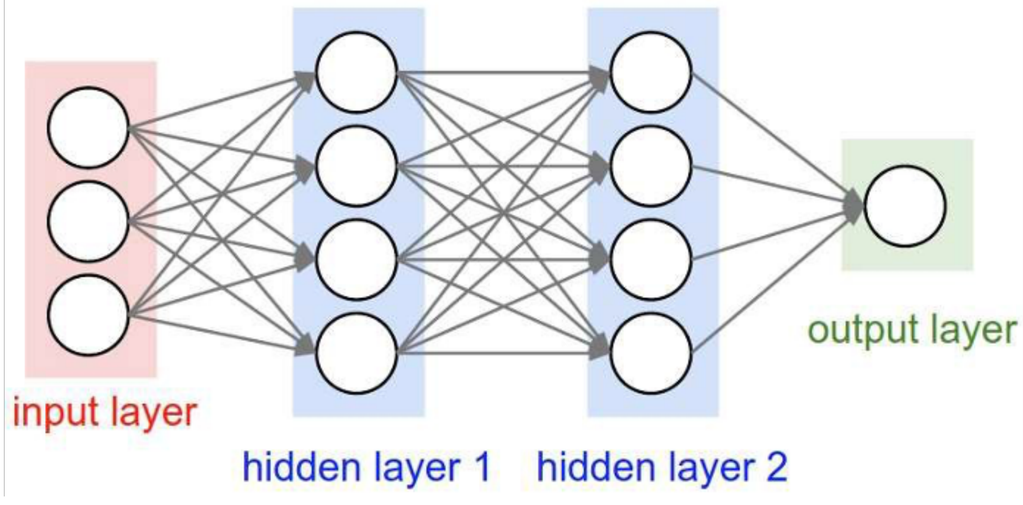
  a nonlinearity $\mathbf{x} \mapsto \mathbf{1}[\mathbf{x} > 0]$

Multilayer Perceptrons

- **Idea.** Use **multiple layers** of parallel perceptrons.
  - In the i-th layer, conduct
    - Linear operation $\mathbf{z} \mapsto \mathbf{W}_i \mathbf{z} + \mathbf{b}_i$ (weights; biases)
    - Activation function $\mathbf{z} \mapsto \sigma_i(\mathbf{z})$ (not necessarily $\mathbf{1}$; typically applied entrywise)

hidden layer activation; internal representation; ...





Multilayer Perceptrons

- **Idea.** Use **multiple layers** of parallel perceptrons.
  - In the i-th layer, conduct
    - Linear operation $\mathbf{z} \mapsto \mathbf{W}_i \mathbf{z} + \mathbf{b}_i$
    - Activation function $\mathbf{z} \mapsto \sigma_i(\mathbf{z})$ (not necessarily $\mathbf{1}$; typically applied entrywise)
- Ignoring the bias term $\mathbf{b}_i$, our predictor can be written as:

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

feature map

linear classifier

Multilayer Perceptrons

- **Idea.** Use **multiple layers** of parallel perceptrons.
  - In the i-th layer, conduct
    - Linear operation $\mathbf{z} \mapsto \mathbf{W}_i \mathbf{z} + \mathbf{b}_i$
    - Activation function $\mathbf{z} \mapsto \sigma_i(\mathbf{z})$ (not necessarily $\mathbf{1}$; typically applied entrywise)
- Ignoring the bias term $\mathbf{b}_i$, our predictor can be written as:

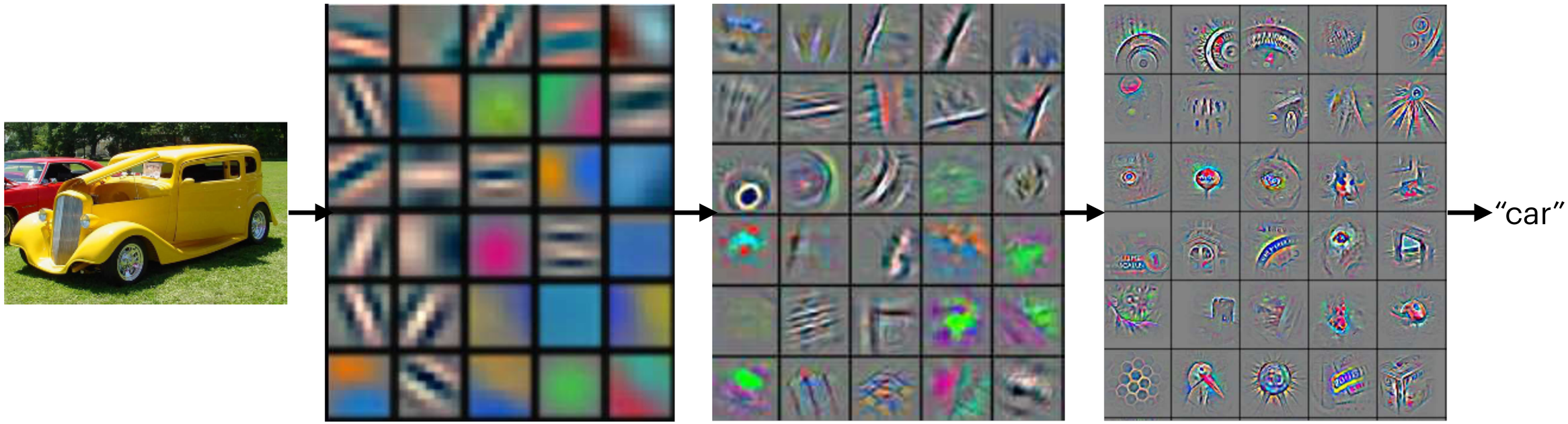
$$f(\mathbf{x}) = \mathbf{W}_L \sigma_{L-1}(\mathbf{W}_{L-1} \sigma(\cdots \sigma(\mathbf{W}_1 \mathbf{x}) \cdots))$$

- Note. Without activations, not different from a linear model

$$f(\mathbf{x}) = \mathbf{W}_L \mathbf{W}_{L-1} \cdots \mathbf{W}_1 \mathbf{x} = \tilde{\mathbf{W}} \mathbf{x}$$

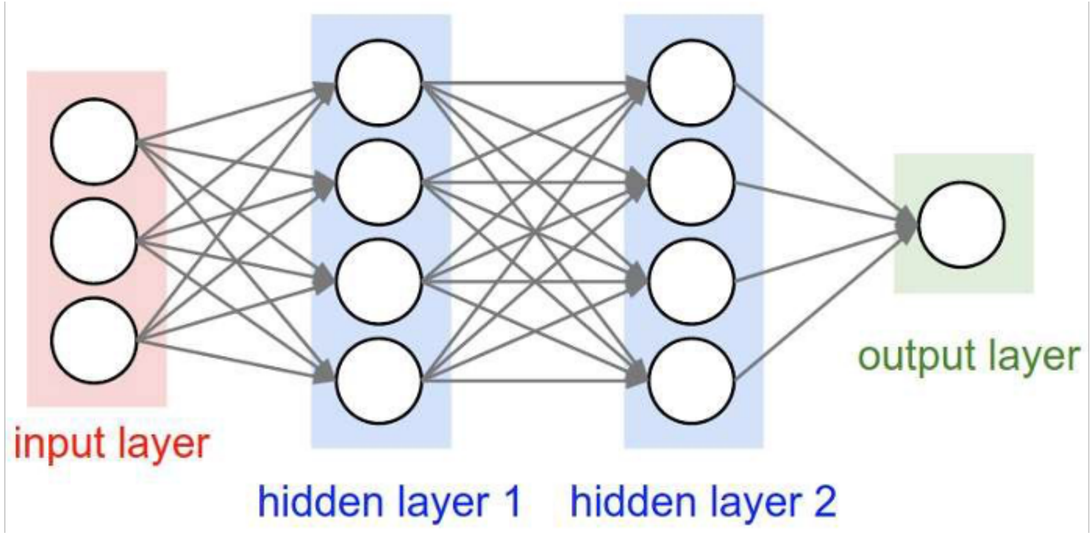
Multilayer Perceptrons

- The features captured by the neurons **grow more complicated**, as the **layer gets deeper**
  - Sometimes, even become human-distinguishable patterns



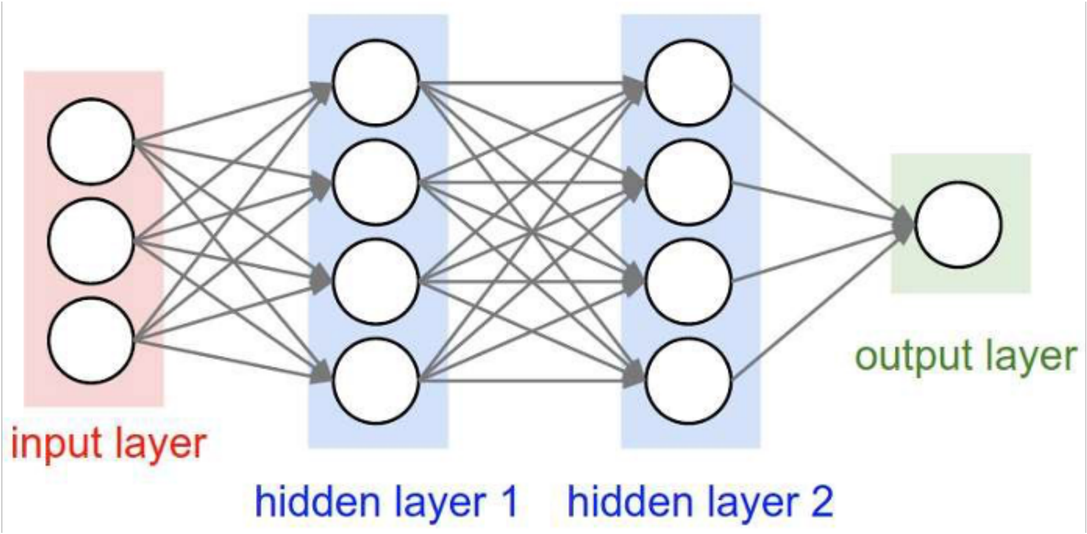
Why are DNNs so cool?

- **Compute.** Easy to compute
  - Mostly linear operations
  - Admits parallel computations



Why are DNNs so cool?

- **Compute.** Easy to compute
  - Mostly linear operations
  - Admits parallel computations
- **Flexible.** Very flexible in size, very modular
  - Can add new layers or neurons freely
  - Design new elementary operations and combine



Building a deep neural network

Modules of a deep network

- **Basic ver.** Repeat the linear operation $\mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}$

  nonlinear activation $\mathbf{x} \mapsto \mathbf{1}[\mathbf{x} \geq 0]$

Modules of a deep network

- **Basic ver.** Repeat the linear operation $\mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}$

  nonlinear activation $\mathbf{x} \mapsto \mathbf{1}[\mathbf{x} \geq 0]$

- Through history, the **MLP parameterization** has evolved through…
  - Replacing linear operation with other simple operations
    - e.g., Convolutional layers (later)
  - Replacing the step activation with other activations
    - e.g., ReLU
  - Modifying the size of the network
    - e.g., Width & Depth

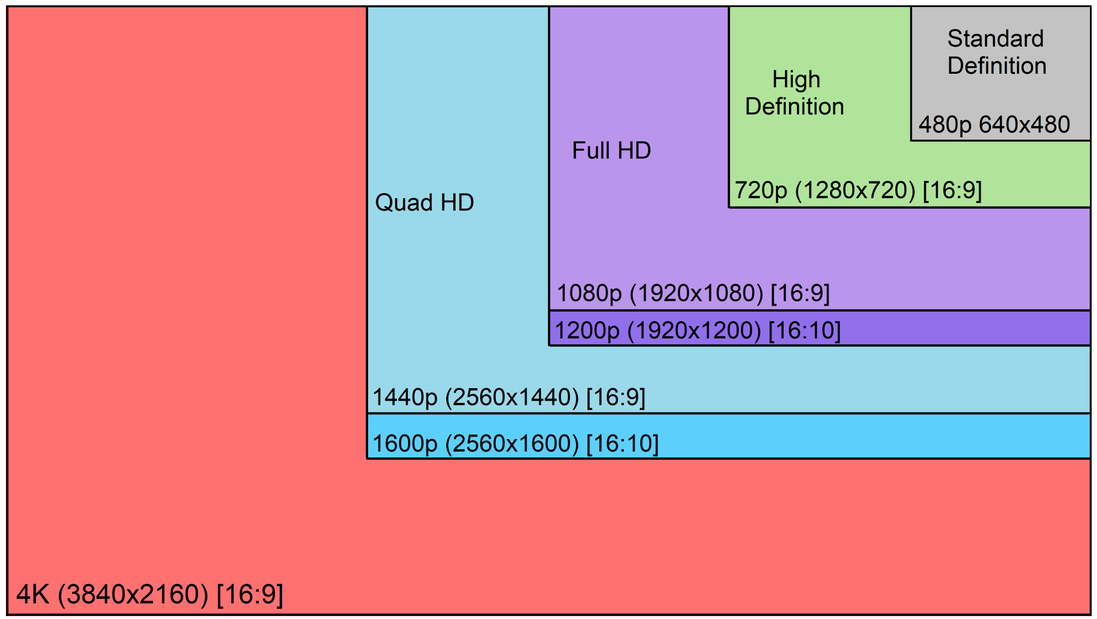
- **Advanced.** Residual connection, self-attention layer, normalization layers, …

Other linear operations

- **Problem.** Vanilla MLP may not work for all data domains

  - Example 1. For 4k images, a 2-layer network with 1,000 hidden neurons need 7.5B parameters (= 30GB of memory)

  - Example 2. For text data, the input can have various number of words



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

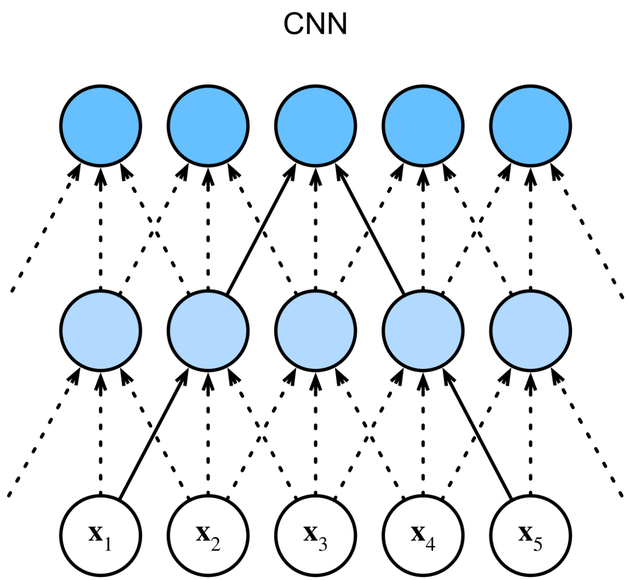
— Source

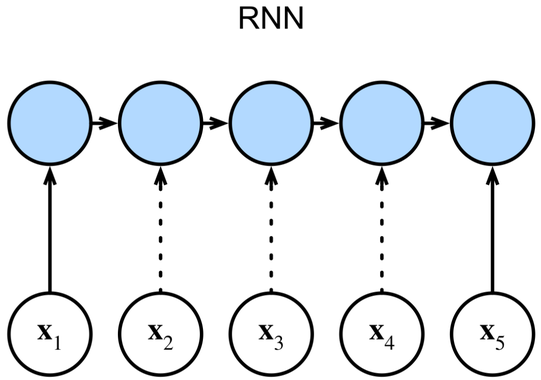
By mid-July, it was 40 percent. In early August, it was 52 percent.

— Reference

Other linear operations

- **Problem.** Vanilla MLP may not work for all data domains
  - Example 1. For 4k images, a 2-layer network with 1,000 hidden neurons need 7.5B parameters (= 30GB of memory)
  - Example 2. For text data, the input can have various number of words
- **Solution.** Develop customized modules for each data domain (more on this later)





Activation functions

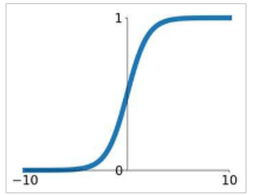
- **Recall.** It is difficult to use gradient descent with $\mathbf{1}[\,\cdot\,]$
  - Gradient is almost always 0

Activation functions

- **Recall.** It is difficult to use gradient descent with $\mathbf{1}[\,\cdot\,]$
  - Gradient is almost always 0
- **Idea.** Use surrogates!
  - Old. Stick to "saturating activation" reminiscent of the step function.

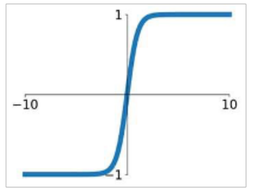
Sigmoid

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



Tanh

$\tanh(x)$

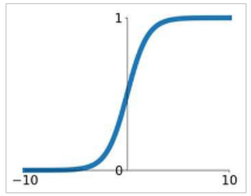


Activation functions

- **Recall.** It is difficult to use gradient descent with $\mathbf{1}[\cdot]$
  - Gradient is almost always 0
- **Idea.** Use surrogates!
  - Old. Stick to "saturating activation" reminiscent of the step function.
  - Modern. Other choices often work better
    - Default. ReLU (Rectified Linear Unit)
      - Better optimization (discussed later)
      - Better computation: easy to compute the output & gradients

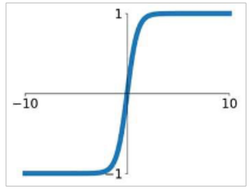
Sigmoid

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



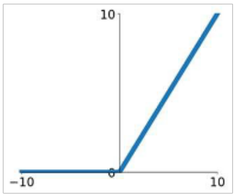
Tanh

$\tanh(x)$



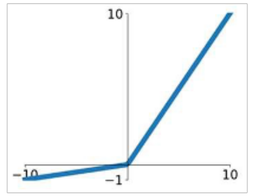
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

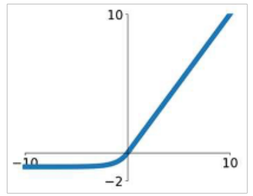


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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

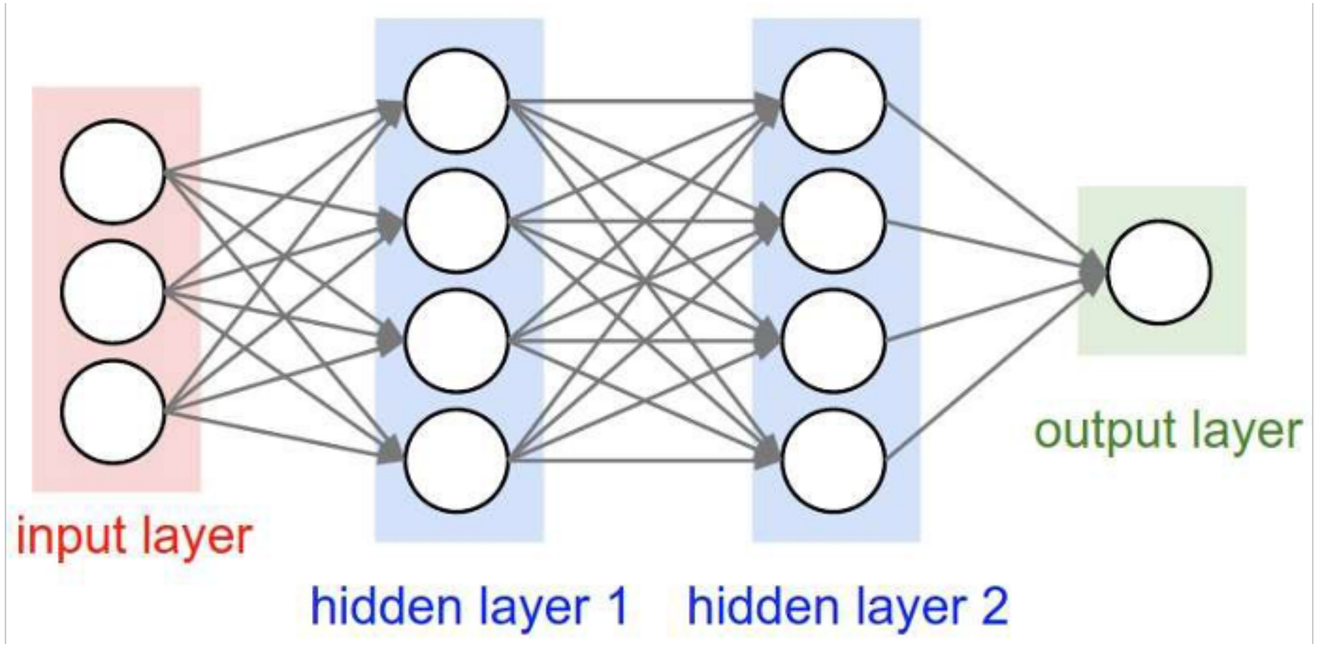


Width and Depth

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

- **Depth.** The number of layers

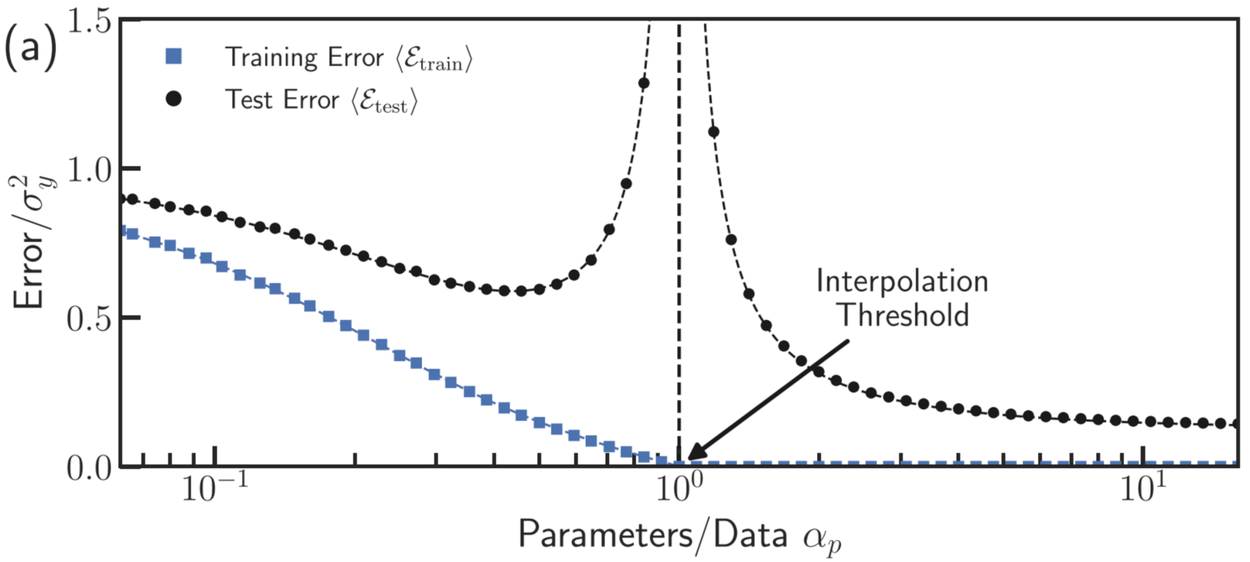
  - Example. a 3-layer network with width 4
    (alternatively, a width-4 network with two hidden layers)



Width and Depth

- **Width.** The number of neurons in each layer (typically the widest)
- **Depth.** The number of layers
  - Example. a 3-layer network with width 4
    (alternatively, a width-4 network with two hidden layers)

- **Determining the size.** Consider the following tradeoff
  - Larger size means...
    - Can fit complicated dataset
    - Less GD steps needed
    - More compute for training & inference
    - Worse/better generalization

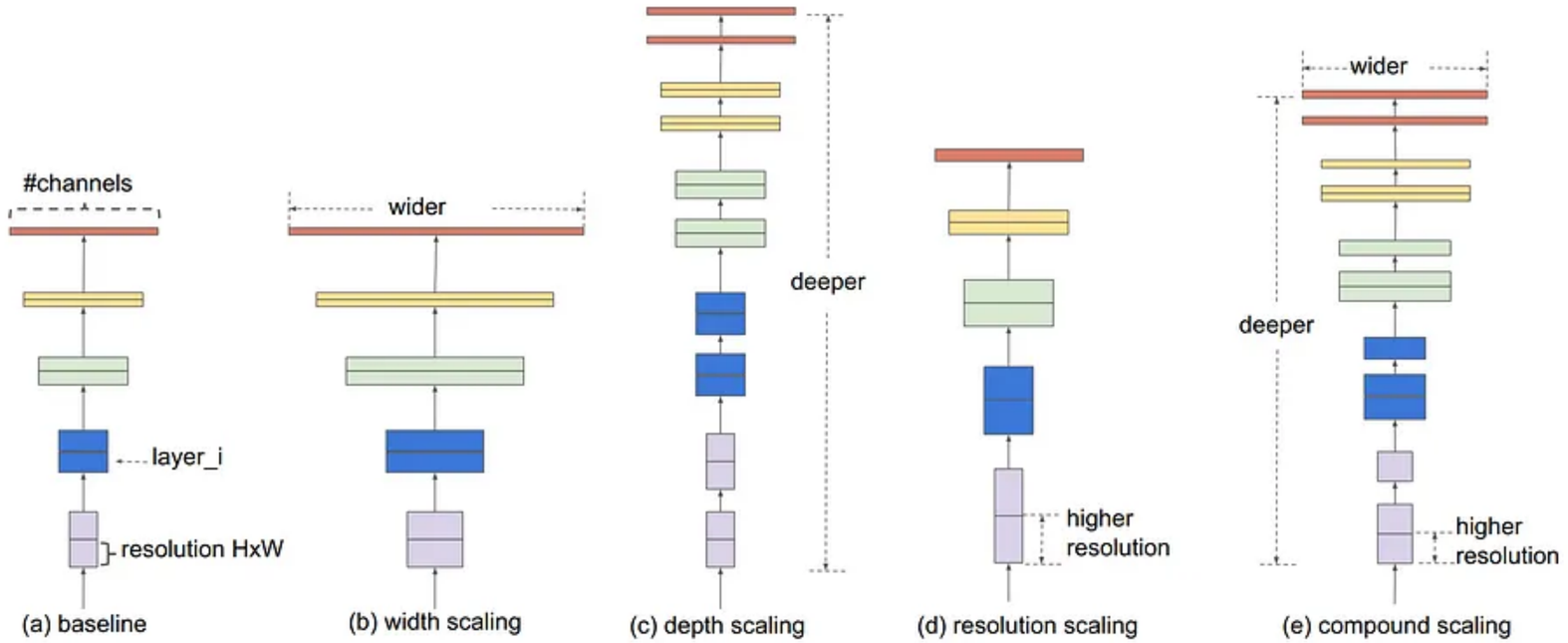


Width and Depth

- **Depth vs Width.** Contributes differently
  - Width. Expressive power ⬆ Generalization ⬆ (beware; RAM bottlenecks)
  - Depth. Expressive power ⬆ ⬆ Generalization ➡ (beware; latency)

Width and Depth

- **Depth vs Width.** Contributes differently
  - Width. Expressive power ⬆ Generalization ⬆ (beware; RAM bottlenecks)
  - Depth. Expressive power ⬆ ⬆ Generalization ➡ (beware; latency)
- **Common wisdom.** Empirically, scaling both width & depth seems to be most "efficient"
  - More involved. Do an explicit search (neural architecture search)



Tan and Le., "EfficientNet: : Rethinking Model Scaling for Convolutional Neural Networks," ICML 2019

Theoretical properties: Approximation

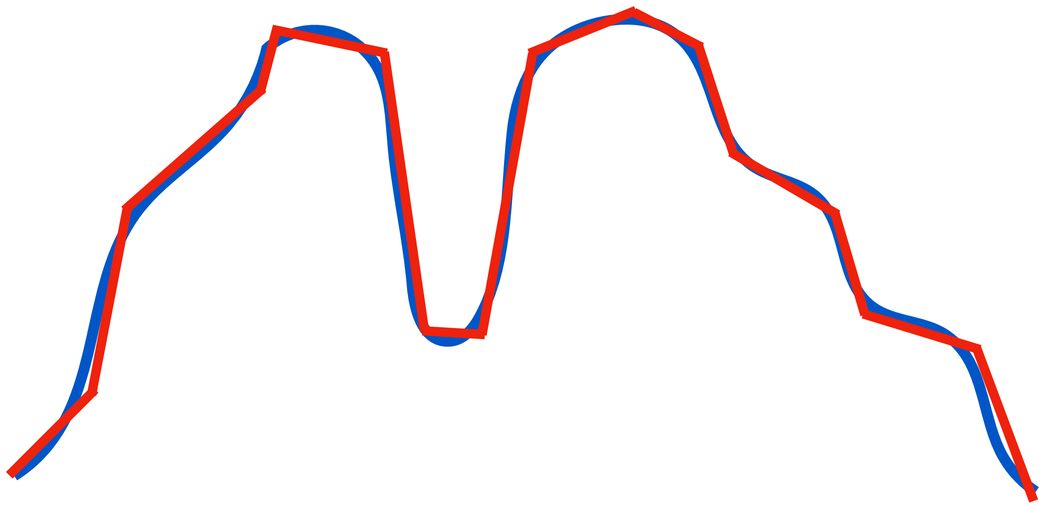
Universal approximation

- A cool aspect of **deep neural network features** is that they can represent any function!
    - Theoretical results, called “universal approximation theorems”

Universal approximation

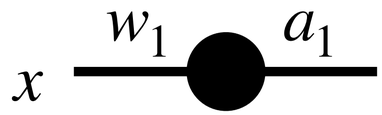
- A cool aspect of **deep neural network features** is that they can represent any function!
  - Theoretical results, called "universal approximation theorems"
- **Theorem.** Given any function $g(\cdot)$ and $\epsilon > 0$,
  one can find a <u>two-layer</u> ReLU neural network $f(\cdot)$ such that

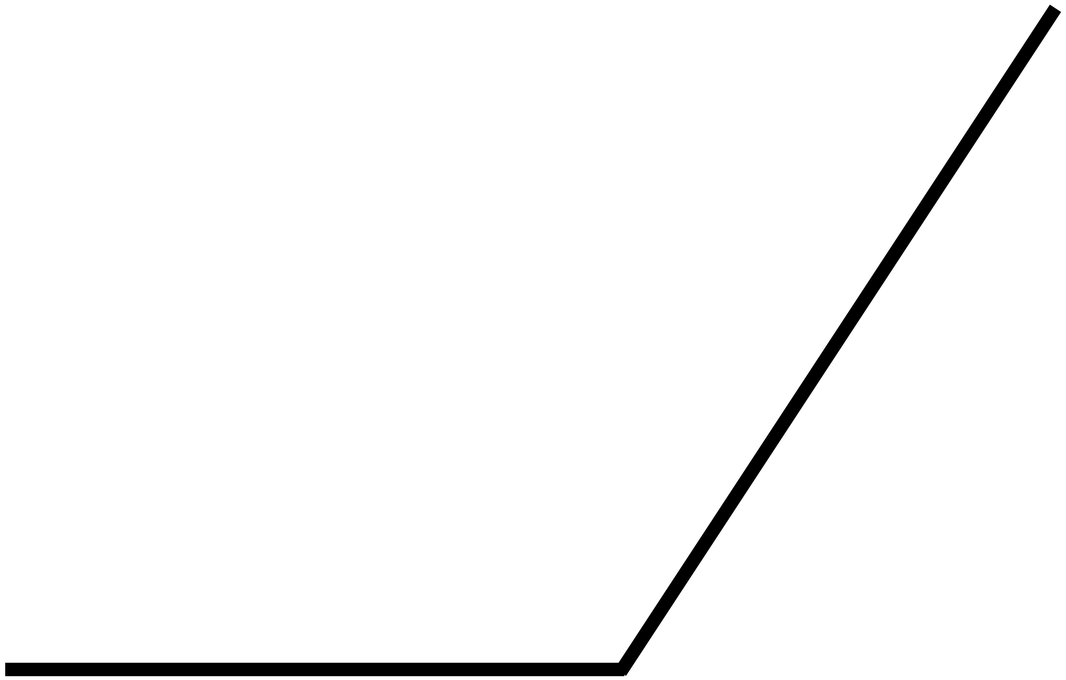
$$\sup_{x \in [0,1]} |g(x) - f(x)| \leq \epsilon$$



Proof idea

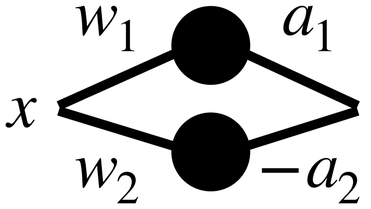
- A single ReLU neuron looks like this

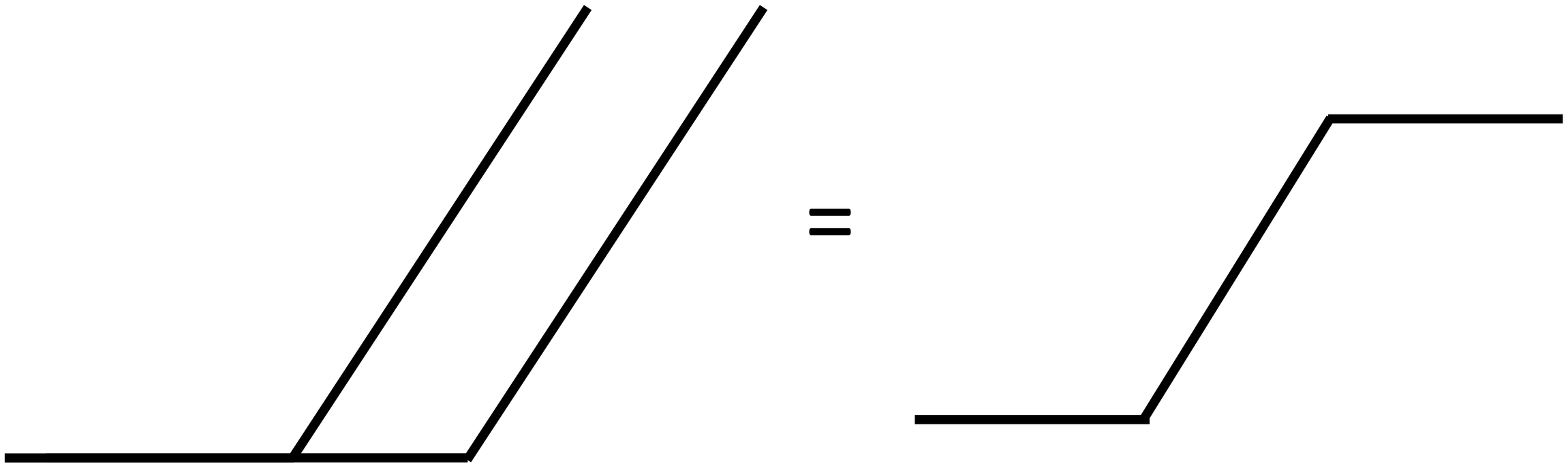




Proof idea

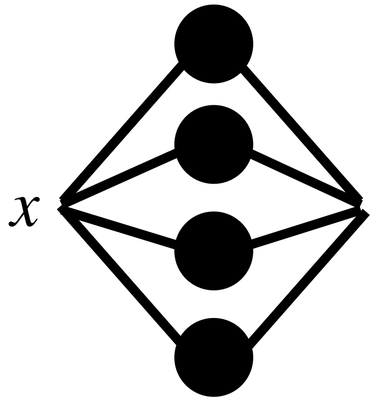
- A single ReLU neuron looks like this
- Difference of two single neurons makes the hard sigmoid

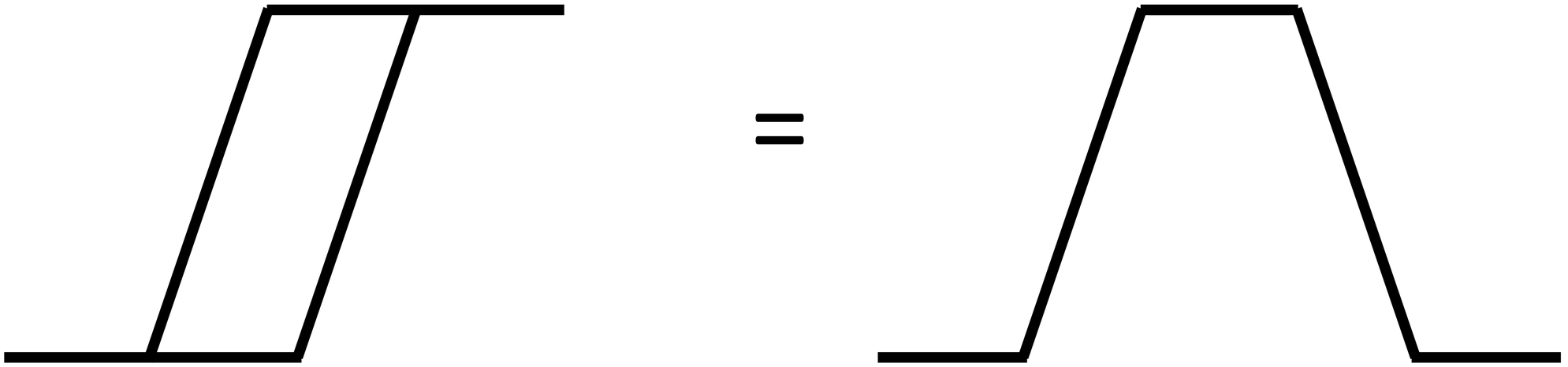




Proof idea

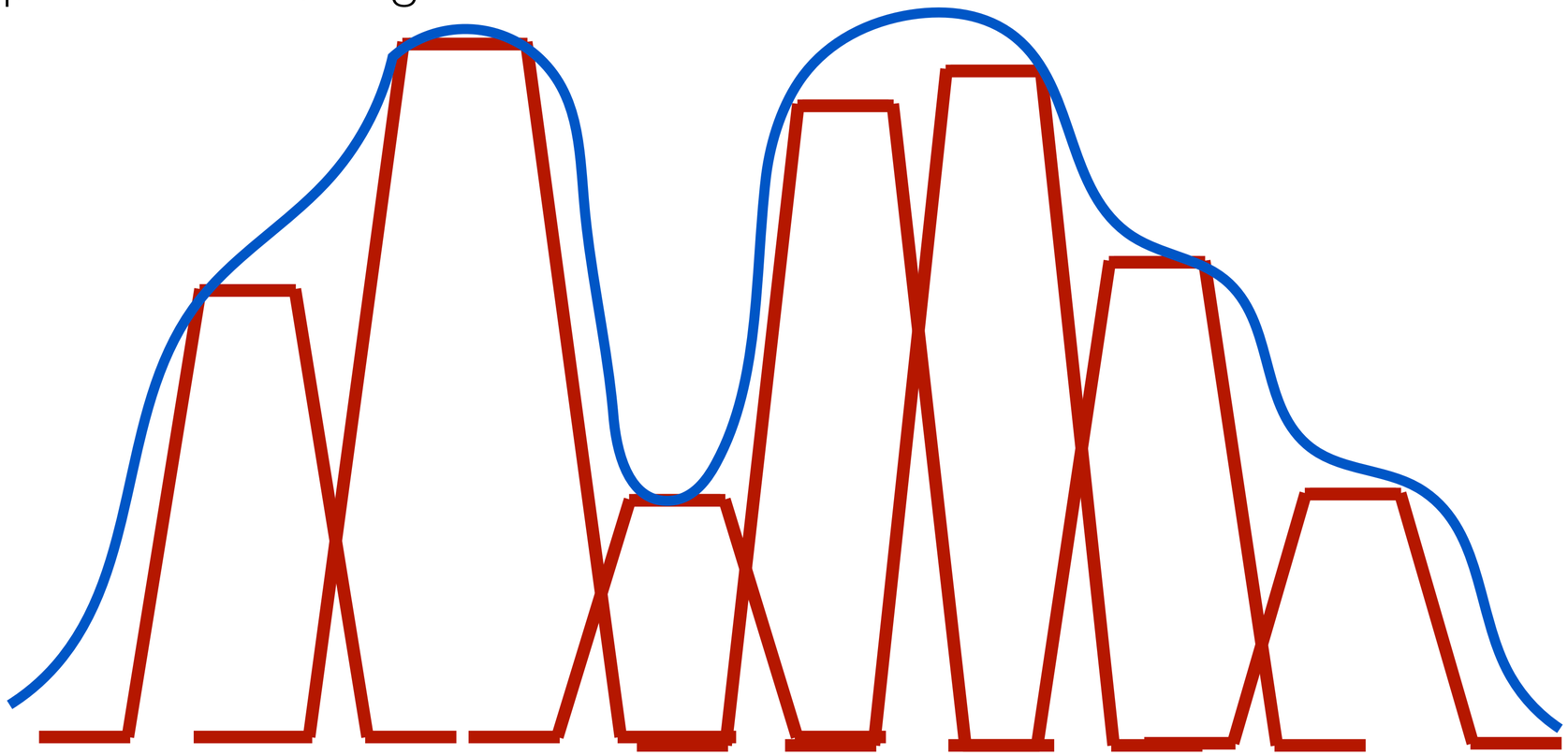
- A single ReLU neuron looks like this
- Difference of two single neurons makes the hard sigmoid
- Difference of two hard sigmoids makes a bump





Proof idea

- A single ReLU neuron looks like this
- Difference of two single neurons makes the hard sigmoid
- Difference of two hard sigmoids makes a bump
- Use bumps to approximate the target function

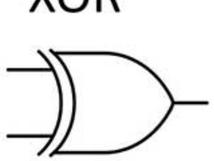


Next up

- Optimization aspects; GD and backpropagation

- **References**
  - The little book of deep learning, https://fleuret.org/francois/lbdl.html
  - Understanding deep learning, https://udlbook.github.io/udlbook/

- **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

Cheers