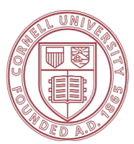
4950/6950 Fall 2026
Special Topics in ECE: AI Hardware

Basics of Neural Networks



Cornell University

CSL

Agenda

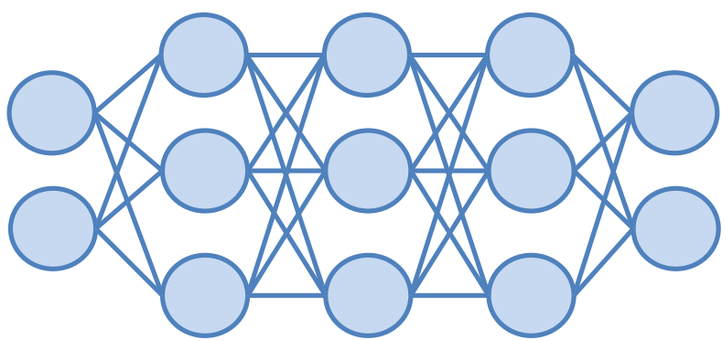
- Neural network in a nutshell
- Perceptron: an artificial neuron
- Multi-layer perceptrons (MLPs)
- Convolutional neural networks (CNNs)

Neural Networks Brief History

- **1940s – 1950s:** First artificial neural networks proposed based on biological structures in the human visual cortex
- **1980s – 2000s:** Neural networks (NNs) considered inferior to other simpler algorithms (e.g., SVM)
- **Mid 2000s**: NN research considered “dead”, machine learning conferences outright reject most NN papers
- **2010s**: Deep convolutional neural networks (DNNs/CNNs) won large-scale image and document classification contests
- **2017 – Now**: Transformer-based models and later LLMs have risen to dominance

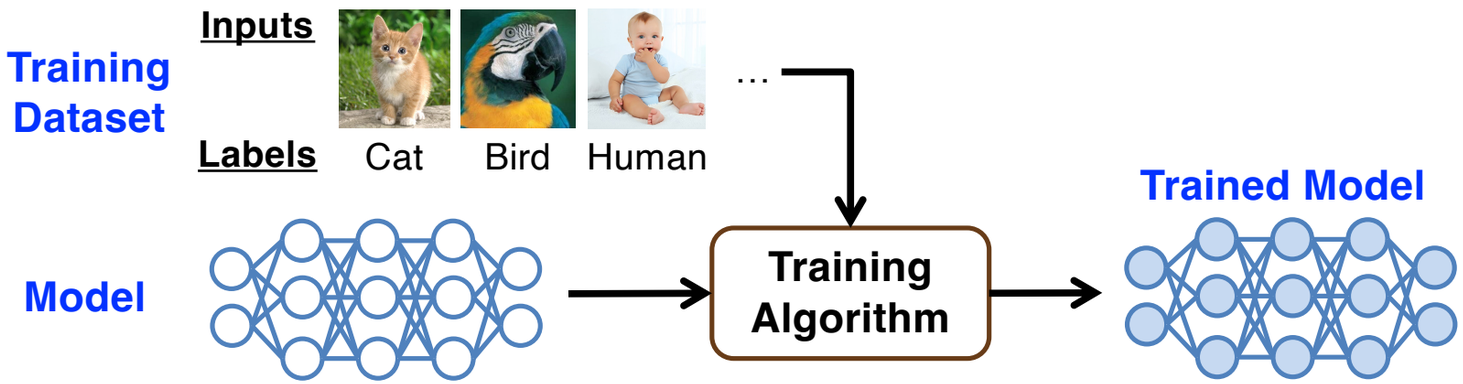
Neural Network (NN) in a Nutshell

- An ML model that **learns to map inputs to outputs by adjusting the learnable parameters (weights) of interconnected computational units**
  - Typically depends on a large volume of data and mostly *supervised*
  - Modern networks consist of a stack of connected *layers*, such as full connected convolutional, attention
    - Learns a function that encodes a hierarchy of abstract *features*

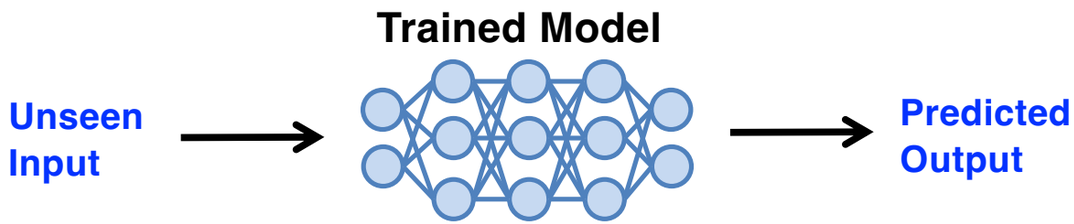


NN Training and Inference

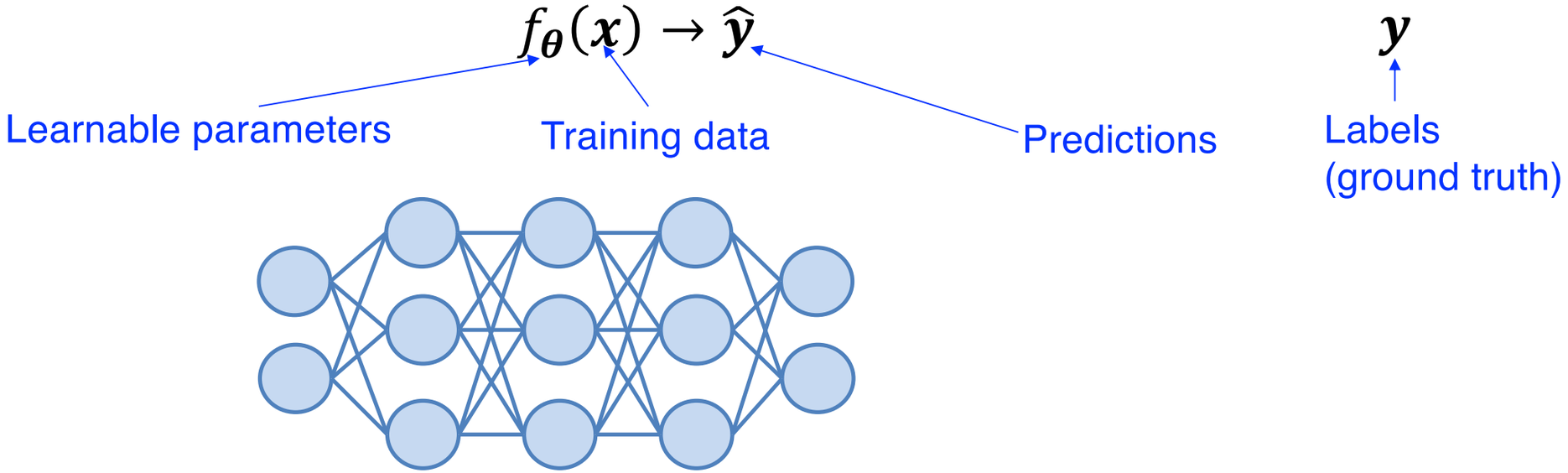
- **Training:** the process an NN model learn to accomplish a **specific task** using a **predetermined dataset**



- **Inference:** the use of a trained NN model to perform the **specific task** on **unseen data**



Basics of NN Training (1)



- Objective: $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum [Loss(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})]$
  - Given N training samples: $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} Loss(f_{\boldsymbol{\theta}}(x_i), y_i)$
- The parameters are typically updated in an iteratively way using gradient descent (over multiple steps)

Basics of NN Training (2)

Let $\mathbf{w}$ denote the set of learnable parameters (weights, bias, etc.)

- Forward pass: $\hat{y}_i = f(\boldsymbol{w}, x_i)$
- Objective: $L(\boldsymbol{w}) = \sum_{i=1}^{N} Loss(\hat{y}_i, y_i) + \lambda \|\boldsymbol{w}\|^2$

  Loss function — Regularization
- Backward pass: $\mathbf{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} L(\boldsymbol{w})$

  Learning rate — Gradients

Artificial Neuron

- The simplest possible neural network contains only one “neuron”, which is described by the following equation:

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

x: **input** (vector)
w: **weights** (vector)
b: **bias** (scalar)
σ: **activation function**
y: **activation** (scalar)

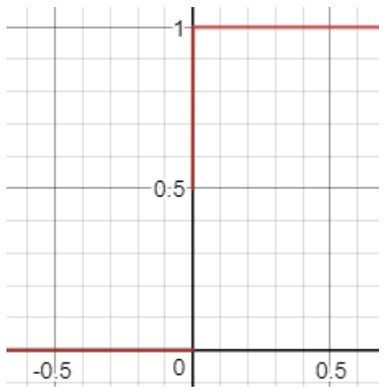
- When σ is the unit step (or sign) function, we have a **perceptron** *
  - Introduced in 1957 by Frank Rosenblatt

*** Perceptron is often used as a synonym for artificial neuron, where σ could be any activation function**

Activation Functions

- The activation function σ is non-linear

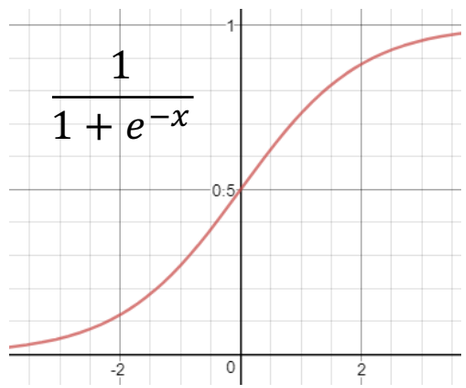
Unit Step (Sign)



Hard Yes/No decision

Used in the initial perceptrons

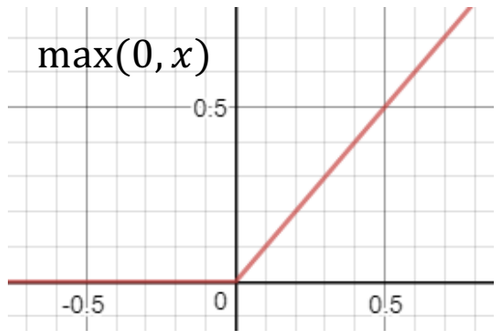
Sigmoid



Soft probability

Used in early neural nets

ReLU

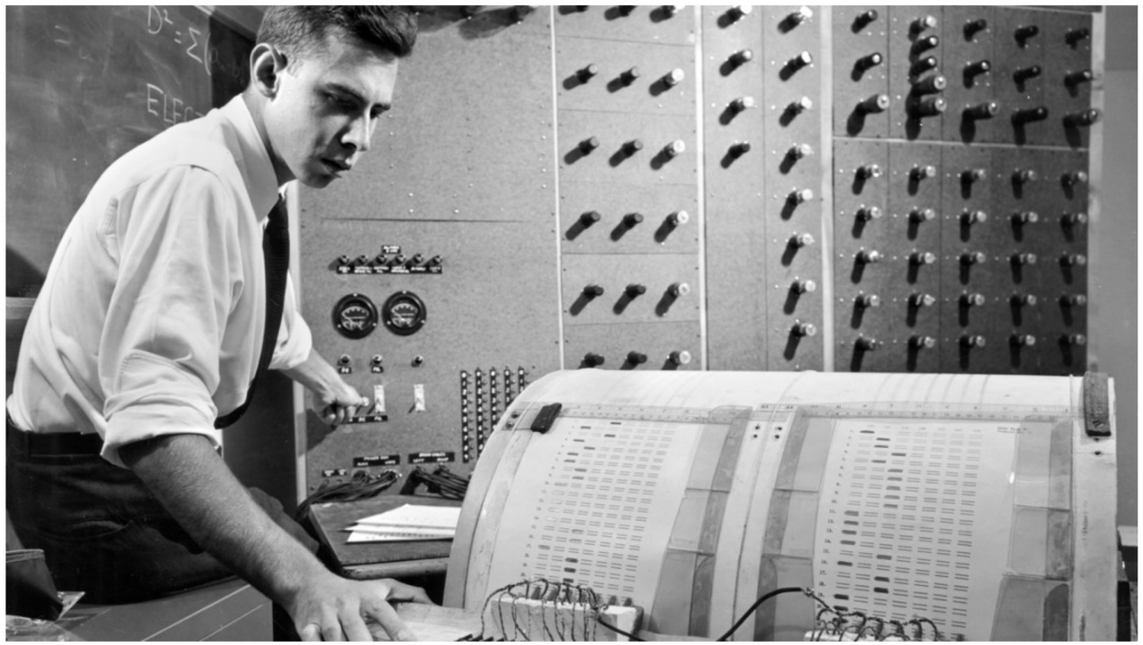


Simple and preserves gradients for positive inputs

Made deep networks easier to train

Modern LLMs typically use smooth, gated activation functions like GELU, SiLU, or variants such as SwiGLU, and GeGLU.

“The foundations for all of this artificial intelligence were laid at Cornell” – Thorsten Joachims, Professor in CIS



Division of Rare and Manuscript Collections

Frank Rosenblatt ’50, Ph.D. ’56, works on the “electronic profile analyzing computer” – a precursor to the perceptron.

Professor’s perceptron paved the way for AI – 60 years too soon

“In July 1958, the U.S. Office of Naval Research unveiled a remarkable invention.

An IBM 704 – a 5-ton computer the size of a room – was fed a series of punch cards. **After 50 trials, the computer taught itself to distinguish cards marked on the left from cards marked on the right.**

It was a demonstration of the “perceptron” – “the first machine which is capable of having an original idea,” according to its creator, Frank Rosenblatt ’50, Ph.D. ’56.

At the time, Rosenblatt – who later became an associate professor of neurobiology and behavior in Cornell’s Division of Biological Sciences – was a research psychologist and project engineer at the Cornell Aeronautical Laboratory in Buffalo, New York.

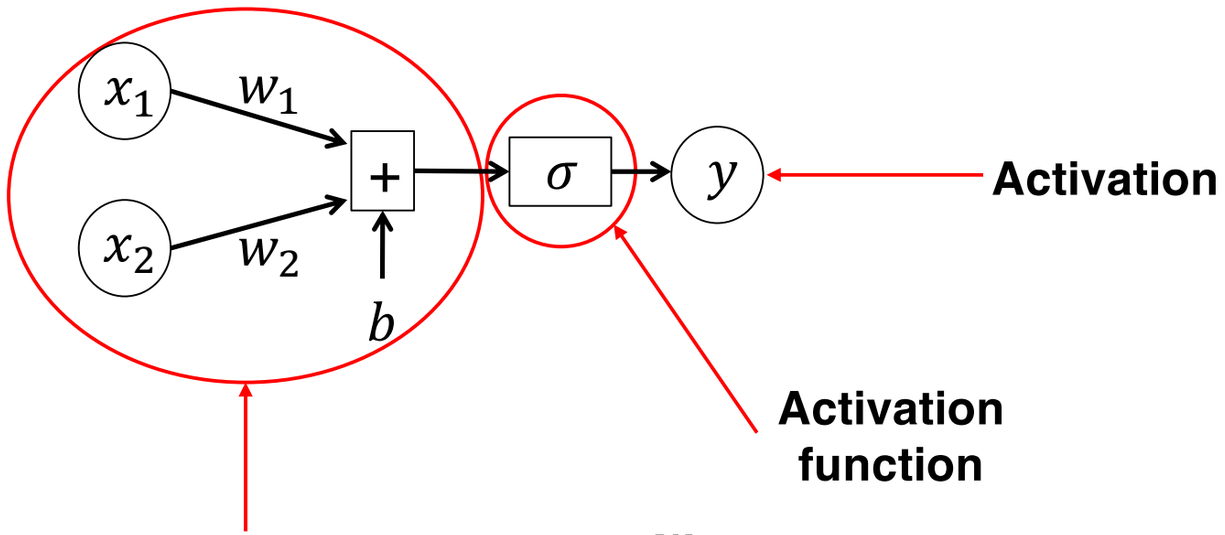
“Stories about the creation of machines having human qualities have long been a fascinating province in the realm of science fiction,” Rosenblatt wrote in 1958. “Yet we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control.”

He was right – but it took half a century to prove it.”

Source: news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

Breaking Down the "Neuron"

A 2-input, 1-output neuron



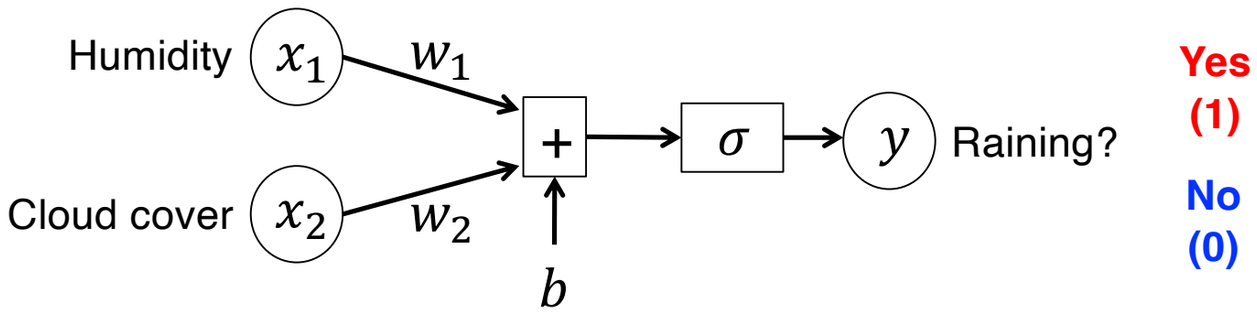
Linear function of $[x_1, x_2]$ and $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$

Vectorized form*: $\boldsymbol{xw} + b$

* More precisely an affine function when bias is nonzero

Breaking Down the "Neuron"

A 2-input, 1-output neuron



A real-life analogue

A Real-life Classification Problem

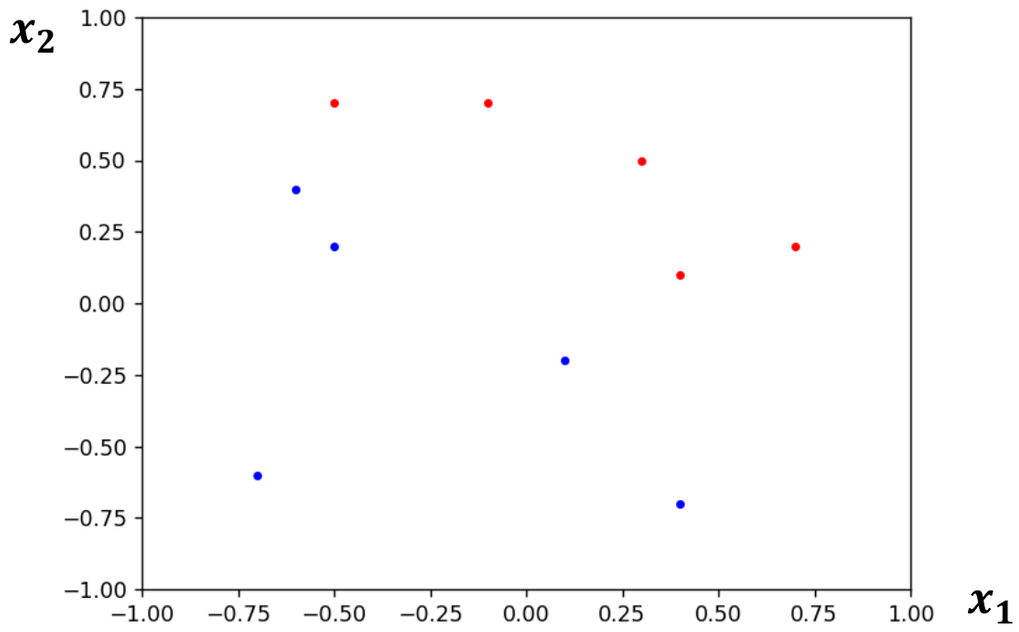
- **Inputs**: Pairs of numbers (x_1, x_2)
- **Labels**: 0 or 1 (binary decision problem)

- Real-life analogue:
  - Label = Raining or Not Raining
  - x_1 = Relative humidity
  - x_2 = Cloud coverage

x_1	x_2	Label
-0.7	-0.6	0
-0.6	0.4	0
-0.5	0.7	1
-0.5	-0.2	0
-0.1	0.7	1
0.1	-0.2	0
0.3	0.5	1
0.4	0.1	1
0.4	-0.7	0
0.7	0.2	1

Training Set

Visualizing the Data

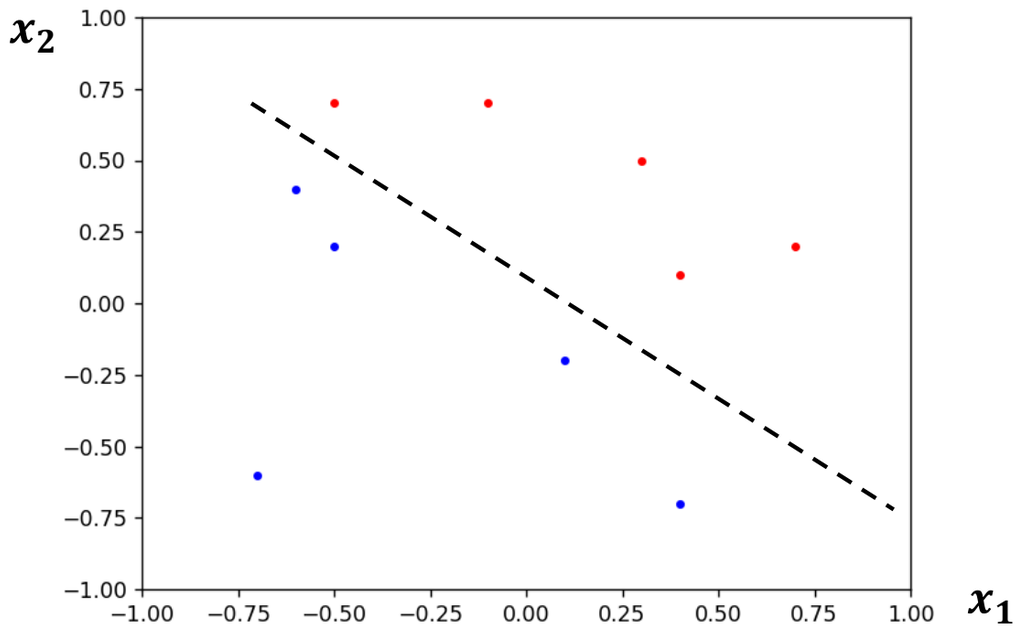


Plot of the data points

x_1	x_2	Label
-0.7	-0.6	0
-0.6	0.4	0
-0.5	0.7	1
-0.5	-0.2	0
-0.1	0.7	1
0.1	-0.2	0
0.3	0.5	1
0.4	0.1	1
0.4	-0.7	0
0.7	0.2	1

Training Set

Decision Boundary



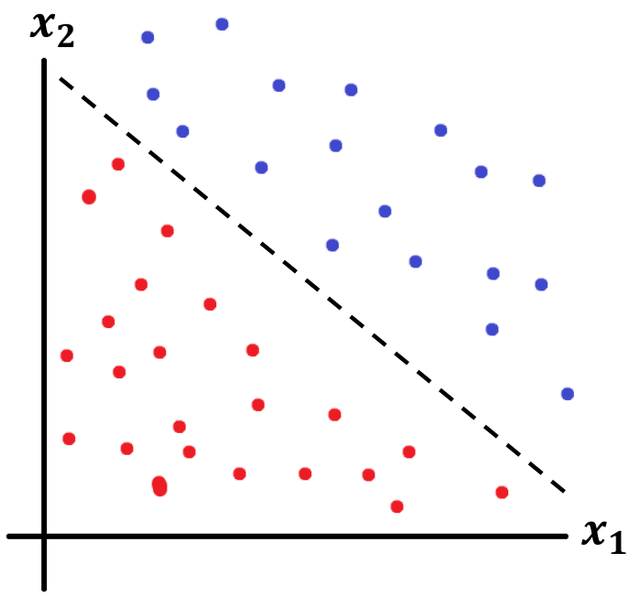
In this case, the data points can be classified with a **linear decision boundary** produced by a perceptron

x_1	x_2	Label
-0.7	-0.6	0
-0.6	0.4	0
-0.5	0.7	1
-0.5	-0.2	0
-0.1	0.7	1
0.1	-0.2	0
0.3	0.5	1
0.4	0.1	1
0.4	-0.7	0
0.7	0.2	1

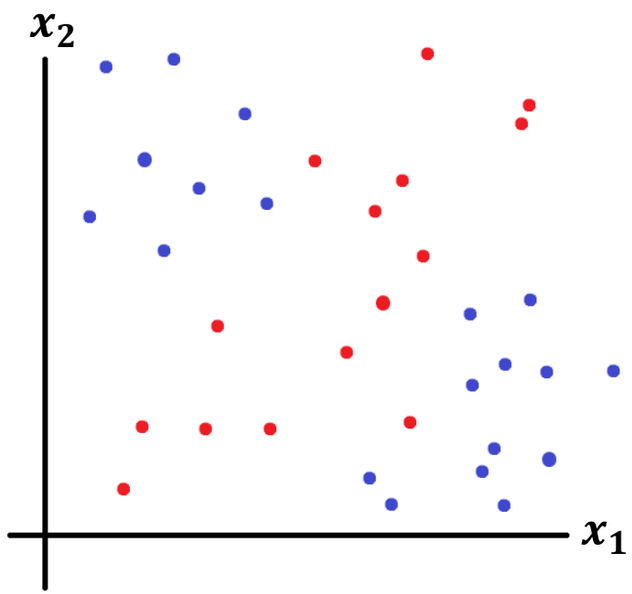
Training Set

Decision Boundary

- A single perceptron (with a step activation) can represent and learn any linearly separable binary function
  - $y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$ defines a **linear decision boundary**
- Many datasets are not linearly separable though!



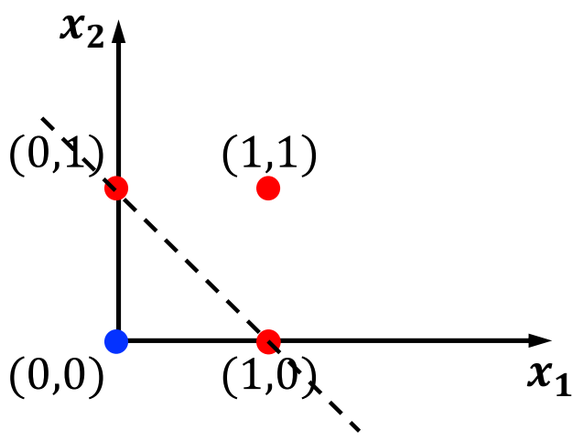
Linearly Separable
Possible to be classified by a single neuron



Non-Linearly Separable
Impossible to be classified by a single neuron

Another Example

x_2	x_1	OR		
0	0	0	$w_1x_1 + w_2x_2 + b < 0$	$\Rightarrow$ $b < 0$
0	1	1	$w_1x_1 + w_2x_2 + b \geq 0$	$\Rightarrow$ $w_1 \geq -b$
1	0	1	$w_1x_1 + w_2x_2 + b \geq 0$	$\Rightarrow$ $w_2 \geq -b$
1	1	1	$w_1x_1 + w_2x_2 + b < 0$	$\Rightarrow$ $w_1 + w_2 \geq -b$

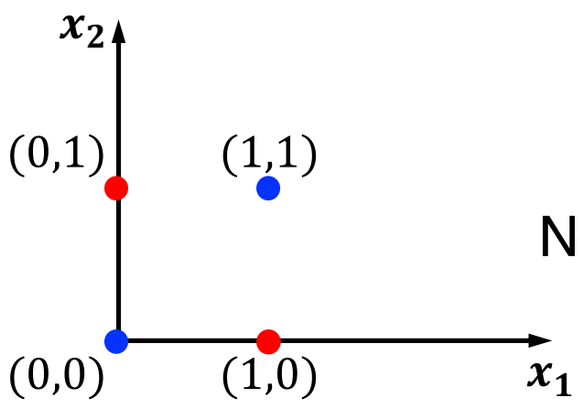


One of the feasible solutions

$w_1 = 1;\ w_2 = 1;\ b = -1$

Yet Another Example (The XOR Problem)

x_2	x_1	XOR			
0	0	0	$w_1x_1 + w_2x_2 + b < 0$	⇨	$b < 0$
0	1	1	$w_1x_1 + w_2x_2 + b \geq 0$	⇨	$w_1 \geq -b$
1	0	1	$w_1x_1 + w_2x_2 + b \geq 0$	⇨	$w_2 \geq -b$
1	1	0	$w_1x_1 + w_2x_2 + b < 0$	⇨	$w_1 + w_2 < -b$

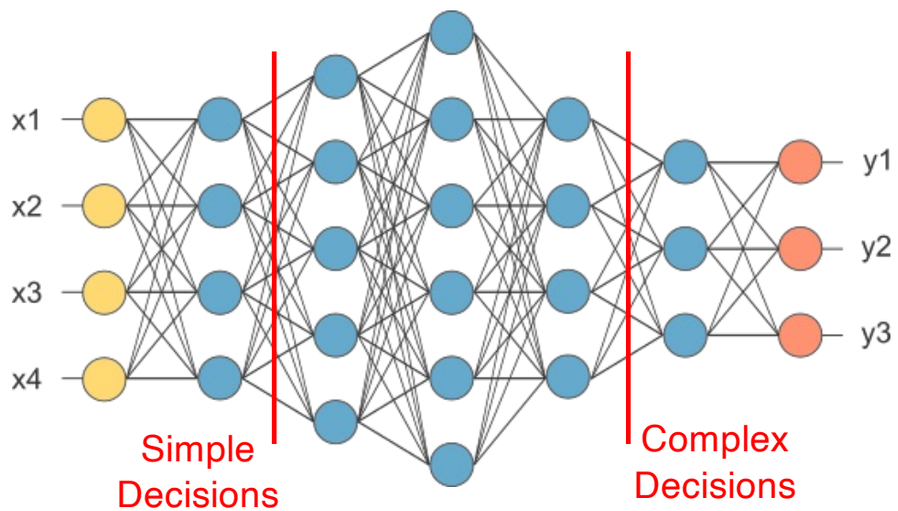


Not linearly separable!

Combining Neurons

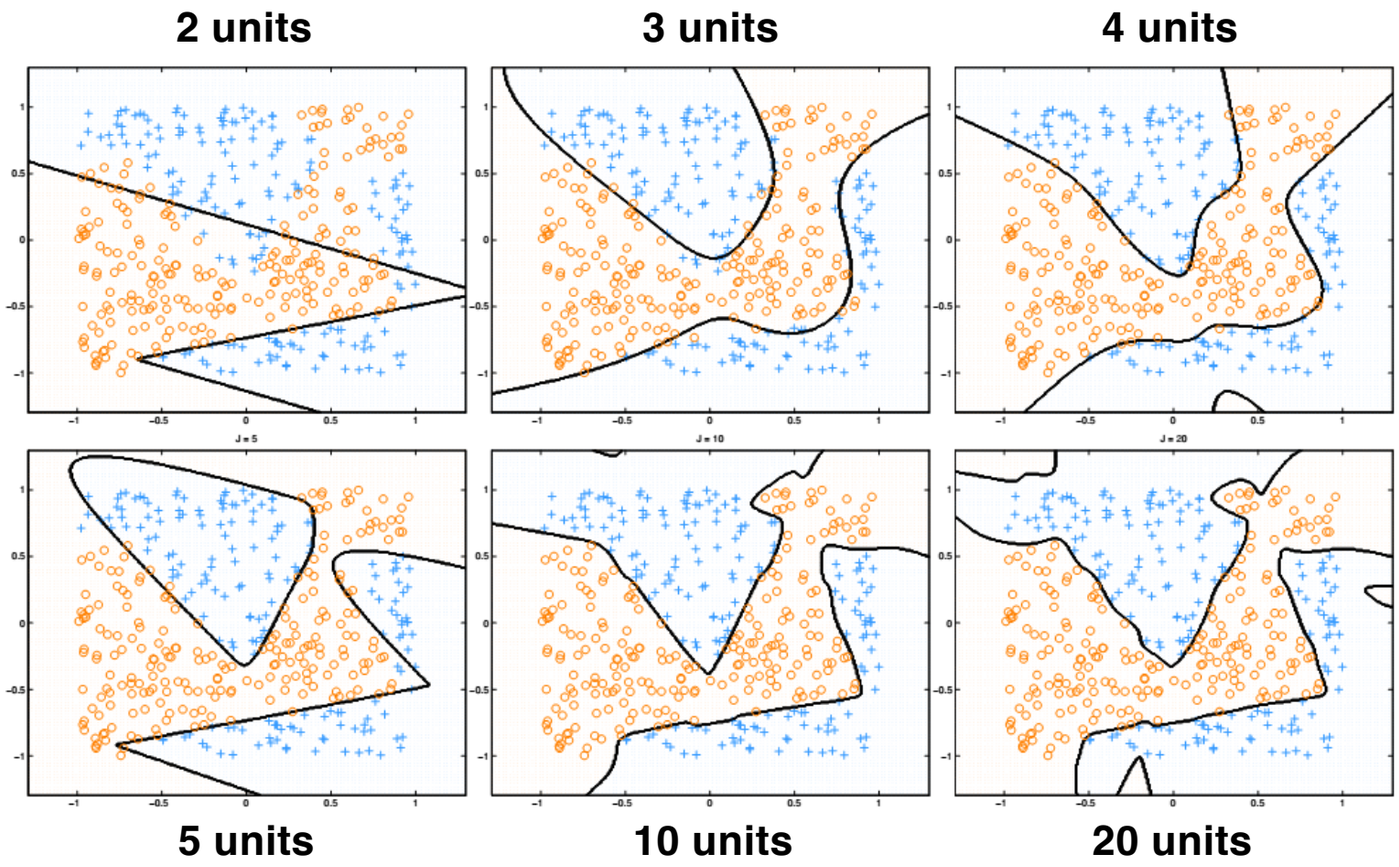
- A single neuron can only make a simple decision

- Feeding neurons into each other allows a neural network to learn **complex decision boundaries**



Source: http://www.opennn.net/

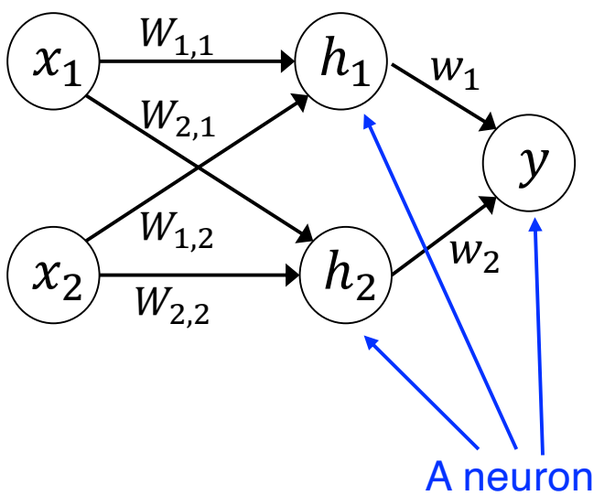
Complex Decision Boundaries



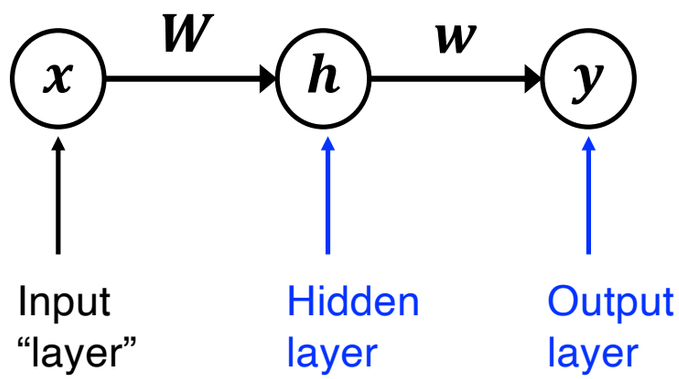
Source: https://www.carl-olsson.com/fall-semester-2013/

Multi-Layer Perceptron (MLP)

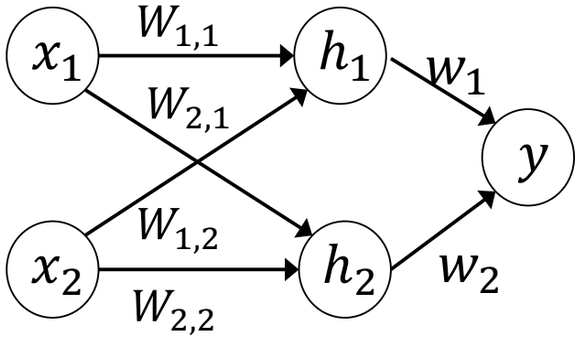
- MLP is a fully connected class of **feedforward** artificial neural network
  - Fully connected (FC): **every neuron interacts with with every input**
  - Each hidden and output layer in an MLP is an FC layer, also called a dense layer

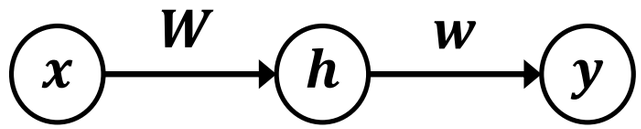


A vectorized (tensorized) view of MLP



Exercise: MLP Inference





- The hidden layer uses ReLU, i.e., $\mathrm{y} = \max(0, x)$
- $\boldsymbol{c}$ denotes the biases of the hidden layer
- b is the bias of the output neuron

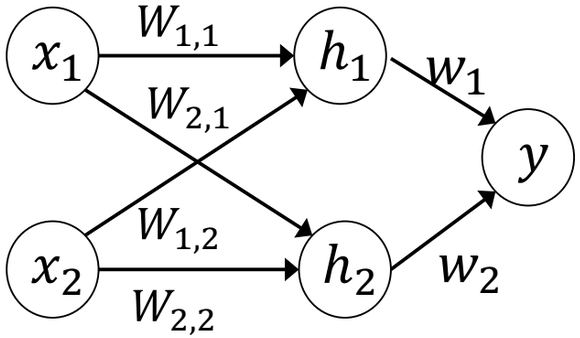
Learned parameters

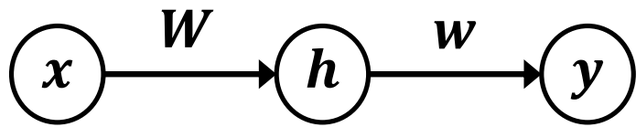
$$\boldsymbol{W} = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix} \quad \boldsymbol{c} = [\mathbf{0}, -\mathbf{1}] \quad \boldsymbol{w} = \begin{bmatrix} \mathbf{1} \\ -\mathbf{2} \end{bmatrix} \quad b = 0$$

Inference

$\boldsymbol{x} = [\mathbf{0}, \mathbf{1}]$ ⇨ $y = ?$ $\boldsymbol{x} = [\mathbf{1}, \mathbf{0}]$ ⇨ $y = ?$

Exercise: MLP Inference





- The hidden layer uses ReLU, i.e., $\mathrm{y} = \max(0, x)$
- $\boldsymbol{c}$ denotes the biases of the hidden layer
- b is the bias of the output neuron

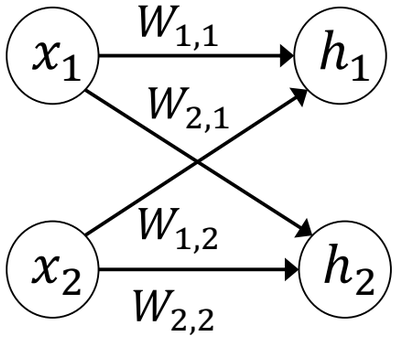
Learned parameters

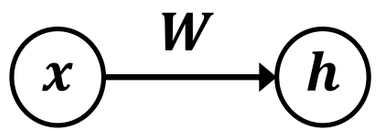
$$\boldsymbol{W} = \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} \end{bmatrix} \quad \boldsymbol{c} = [\mathbf{0}, -\mathbf{1}] \quad \boldsymbol{w} = \begin{bmatrix} \mathbf{1} \\ -\mathbf{2} \end{bmatrix} \quad b = 0$$

Inference

$\boldsymbol{x} = [\mathbf{0}, \mathbf{0}]$ ⇨ $y = ?$ $\boldsymbol{x} = [\mathbf{1}, \mathbf{1}]$ ⇨ $y = ?$

Tensorized View of an MLP Layer





Symbol	Dimension	This example
B	batch size	1
I	input dimension	2
D	hidden dimension	2

$$\boldsymbol{h} = \mathbf{x}\mathbf{W}$$

$\boldsymbol{h}$: [1, D] $\boldsymbol{x}$: [1, I] $\boldsymbol{W}$: [I, D]

In ML, tensor = multi-dimensional numerical array; here W: [I, D] indicates a 2D tensor (matrix) with I rows and D columns

- For an input, each MLP layer multiplies a weight matrix $\boldsymbol{W}$ by its corresponding vector $\boldsymbol{x}$
  - Assume bias is 0 here (many modern LLM linear layers omit bias entirely)
  - Matrix-vector multiplication (**MV** or **GEMV** for short) is an important operation for AI models

Batching

- A **batch** is a group of independent inputs processed together in one forward/backward pass
  - With batching, **B** inputs are stacked into a matrix, and the same layer processes all of them at once

Symbol	Dimension
B	**batch size**
I	input dimension
D	hidden dimension

$$\boldsymbol{H} = \mathrm{XW}$$

$\boldsymbol{H}$: [B, D] $\boldsymbol{X}$: [B, I] $\boldsymbol{W}$: [I, D]

- **Matrix multiplication** is the core operation behind MLPs and a foundational compute pattern across modern AI models
  - Common shorthand: MatMul, **MM**, **GEMM** (General Matrix-Matrix Multiplication)
  - A batch turns many MVs into one MM

Deep Neural Network (DNN)

- MLPs are neural networks with at least three layers, while DNNs typically have even more (hidden) layers

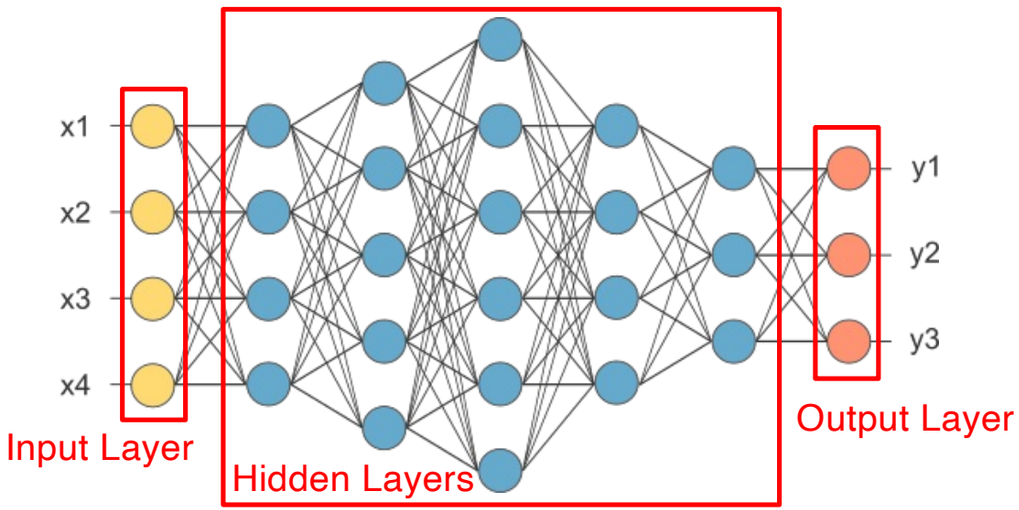
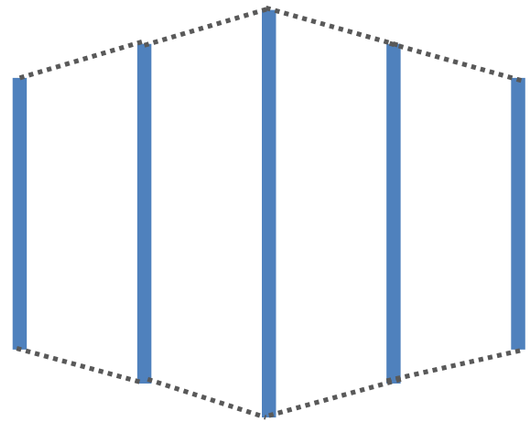


Image credit: http://www.opennn.net/



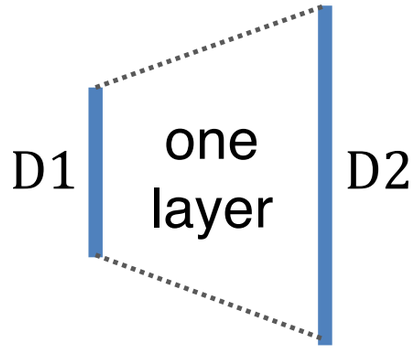
An abstract view (hidden layers)

Computational Requirements of DNN Inference

Parameters

(in bytes, actual size depending on precision)

$$\mathbf{D1 \times D2}$$

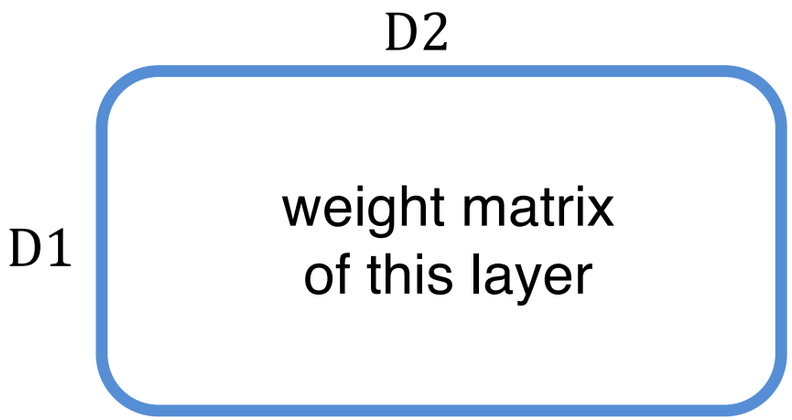


Symbol	Dimension
B	batch size
D	hidden dimension

Compute

(in FLOPs, each multiply-add counts as 2 operations)

$$\underline{2} \times \mathbf{B} \times \mathbf{D1} \times \mathbf{D2}$$



Memory

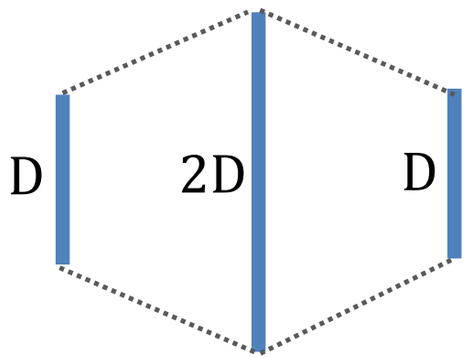
(in bytes, actual size depending on precision)

$$\mathbf{D1 \times D2}\ (\text{params}) +$$
$$\mathbf{B \times D2}\ (\text{peak activations})$$

Visuals adopted from “Street Fighting Transformers” by Sasha Rush [YouTube link]
(Note that the video uses the W∗X convention and counts each multiply-add as one operation, which differs from our convention)

Exercise

Parameters



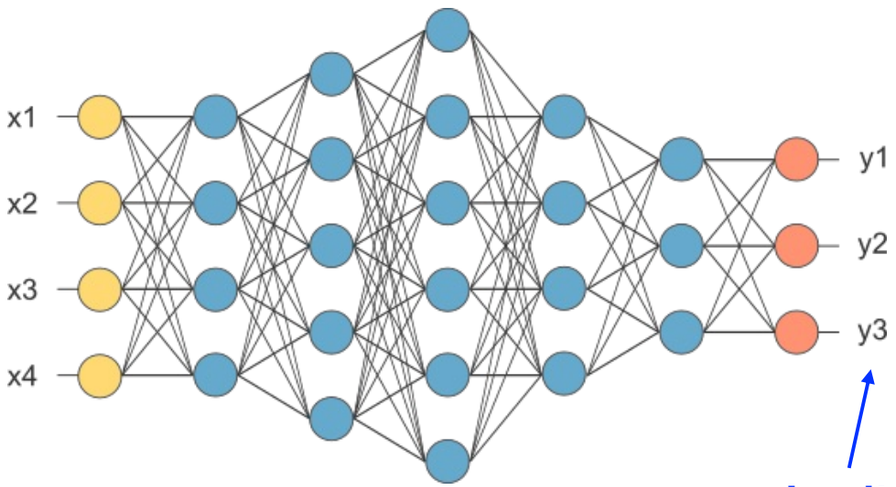
Symbol	Dimension
B	batch size
D	hidden dimension

Compute

Memory

Logits and Softmax

- DNN's final layer produces **logits**, i.e., outputs of the last linear transformation
- **Softmax** converts the logits into class probabilities (sum to 1)



$$p_i = \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_k}}$$

y_i: logit (raw score) of class i
K: number of output classes
p_i: predicted probability

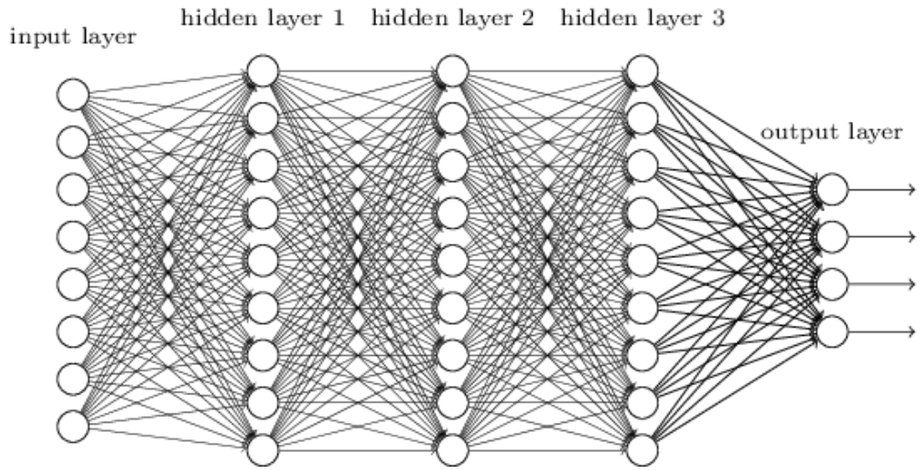
Logits y		Probabilities p
Cat **2.0**	→	Cat **0.66**
Car **1.0**		Car **0.24**
Dog **0.1**		Dog **0.10**

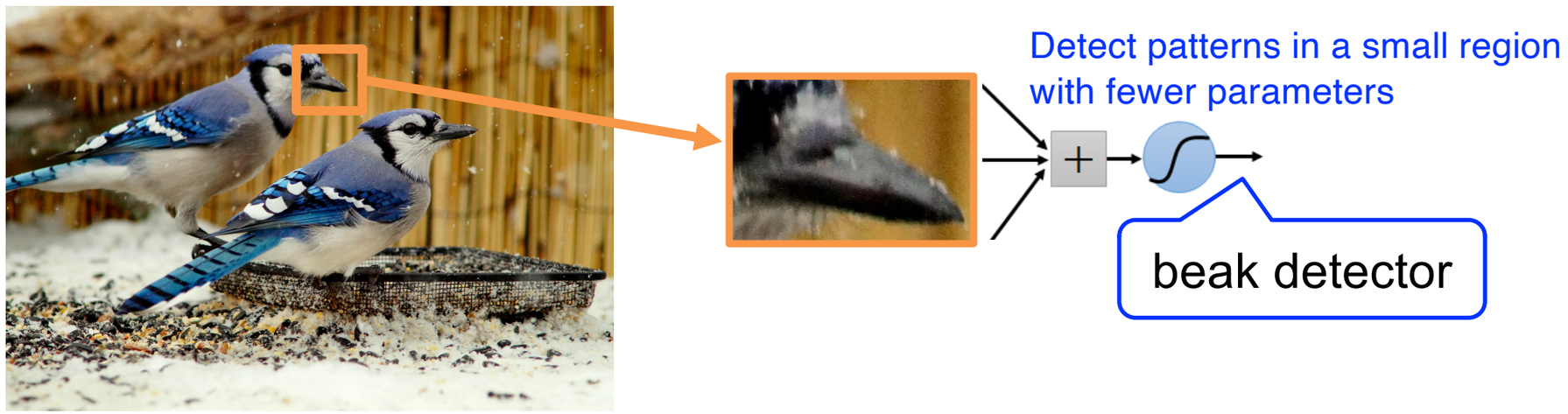
Softmax preserves ranking but changes scale, **turning a vector of logits into a probability distribution over the output classes**

- Inference: select the class with the highest probability (i.e., argmax)
  - Other strategies: top-k predictions, sampling, etc.

Neural Networks for Images

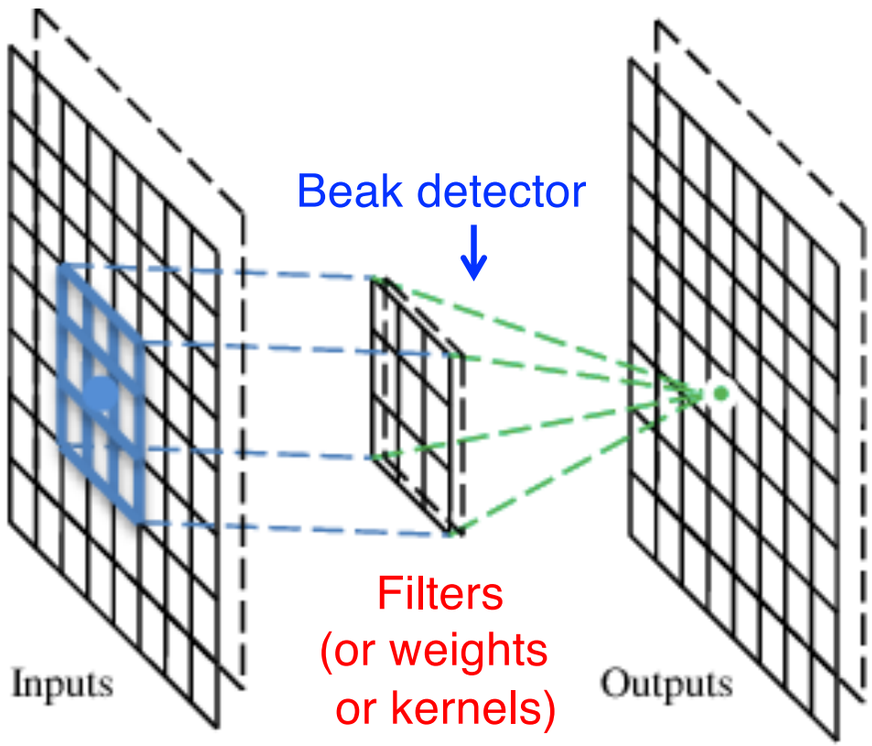
- So far, we have seen neural networks built from **fully-connected layers**
- Do we really need all the edges for learning an image?
  - Important patterns are typically much smaller than the whole image
  - Images are also shift-invariant (e.g., a bird is a bird even when shifted in the image)



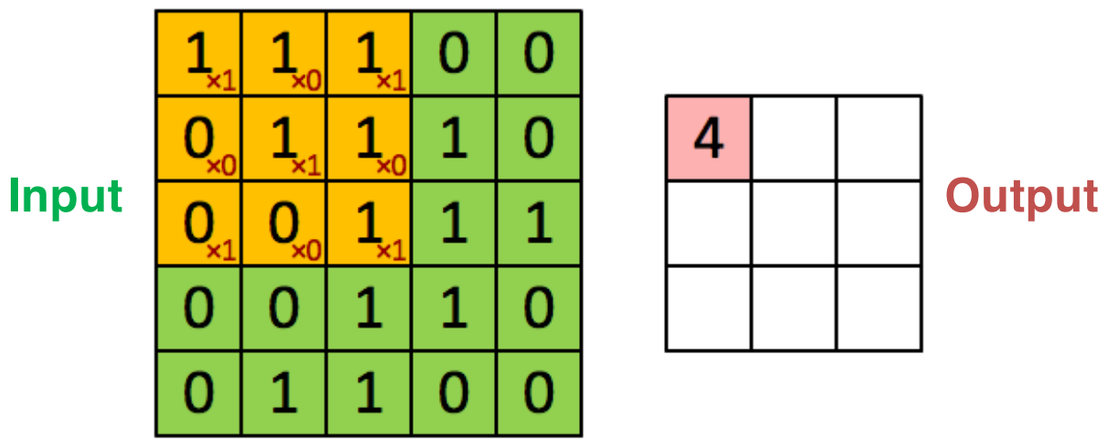


A Convolutional (conv) Layer

- A conv layer has a set of learnable filters that perform convolution operation



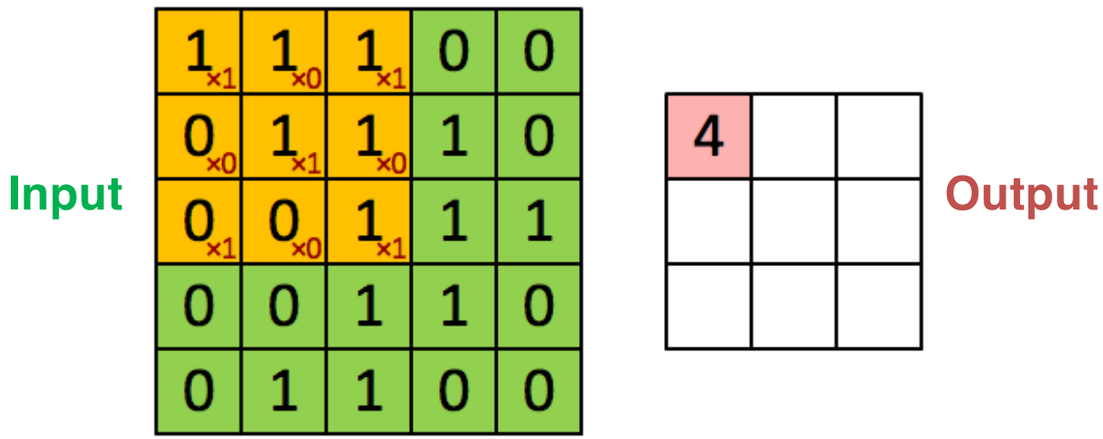
The Convolutional Filter



- Each 2D input (image) convolves with a **weight filter** to produce a 2D outputs (basically an image of features)

Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

The Convolutional Filter

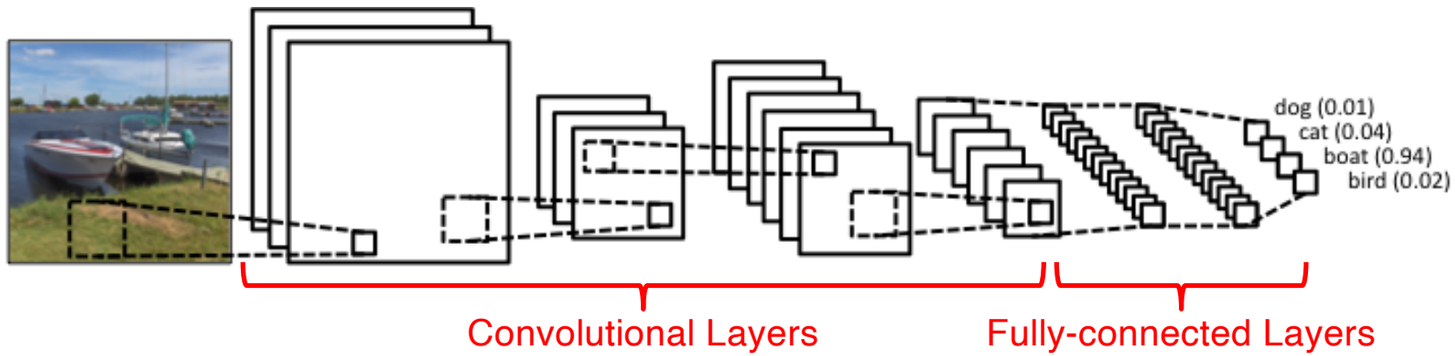


- Each point (“pixel”) in the output encodes both a decision and its spatial location
- **Can detect the pattern anywhere in the image!**

Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Network (CNN)

- A CNN stacks multiple conv layers (and some other layers)

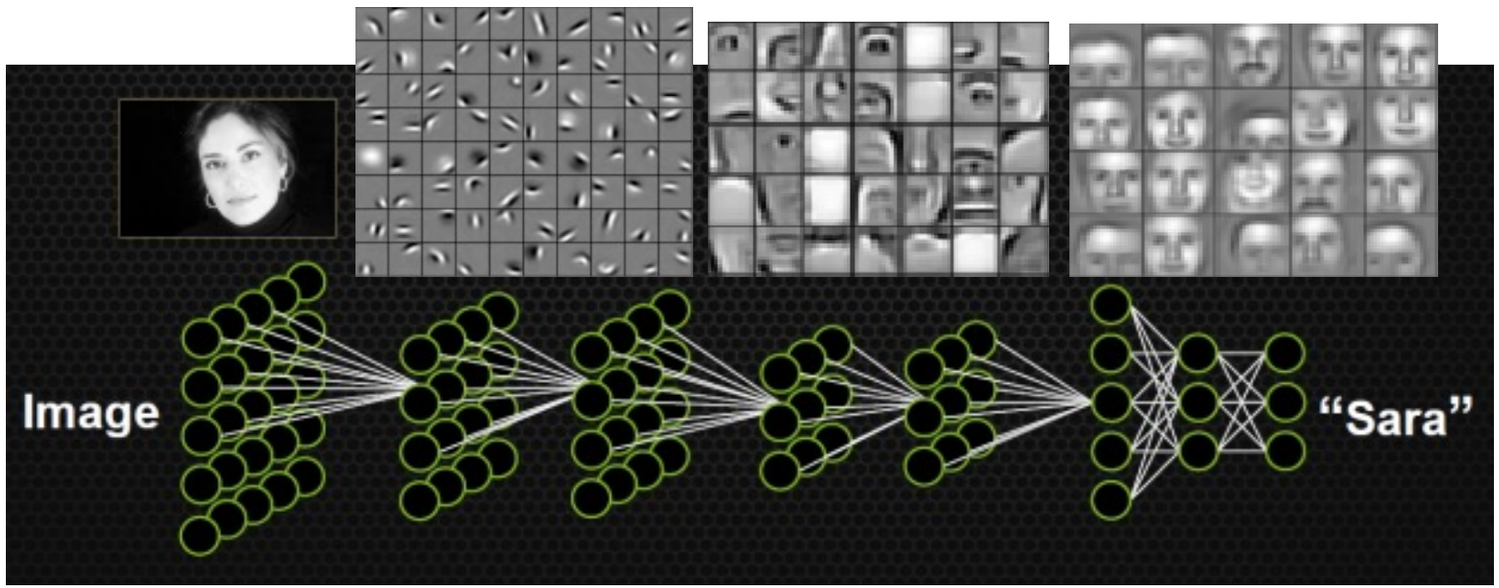


  - Convolutional (conv) layers learn visual features
  - Feature maps get downsampled through the network
  - Fully-connected (dense) layers perform classification using the visual features

Source: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Learning Complex Features

- CNNs combine simple features into complex patterns
  - Early conv layers = edges, textures, ridges
  - Later conv layers = eyes, noses, mouths

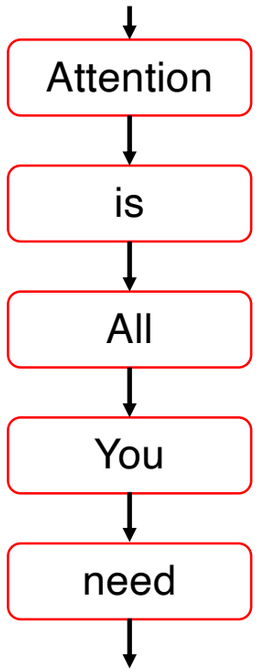


Source: H. Lee, et al. "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks", CACM 2011.

Transformer (Covered in a Later Lecture)

RNN

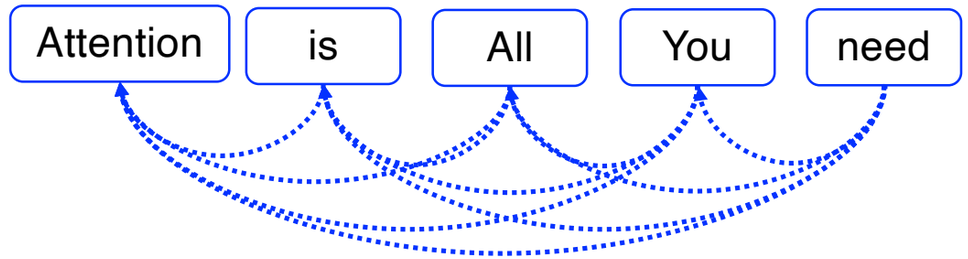
Step-by-step processing



slow in modern hardware

Transformer

Parallel context processing



Compute attention between tokens in parallel

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

Next Lecture

- Hardware Specialization

Acknowledgements

- The lecture slides contain/adapt materials from
  - Ritchie Zhao (Cornell PhD, now NVIDIA), Jordan Dotzel (Cornell PhD, now Google), Yichi Zhang (Cornell PhD, now Google), Yixiao Du (Cornell PhD)
  - CS898 by Prof. Ming Li (University of Waterloo)