SSM Course 2024-25

Introduction to Deep Learning

Mirco Musolesi

mircomusolesi@acm.org

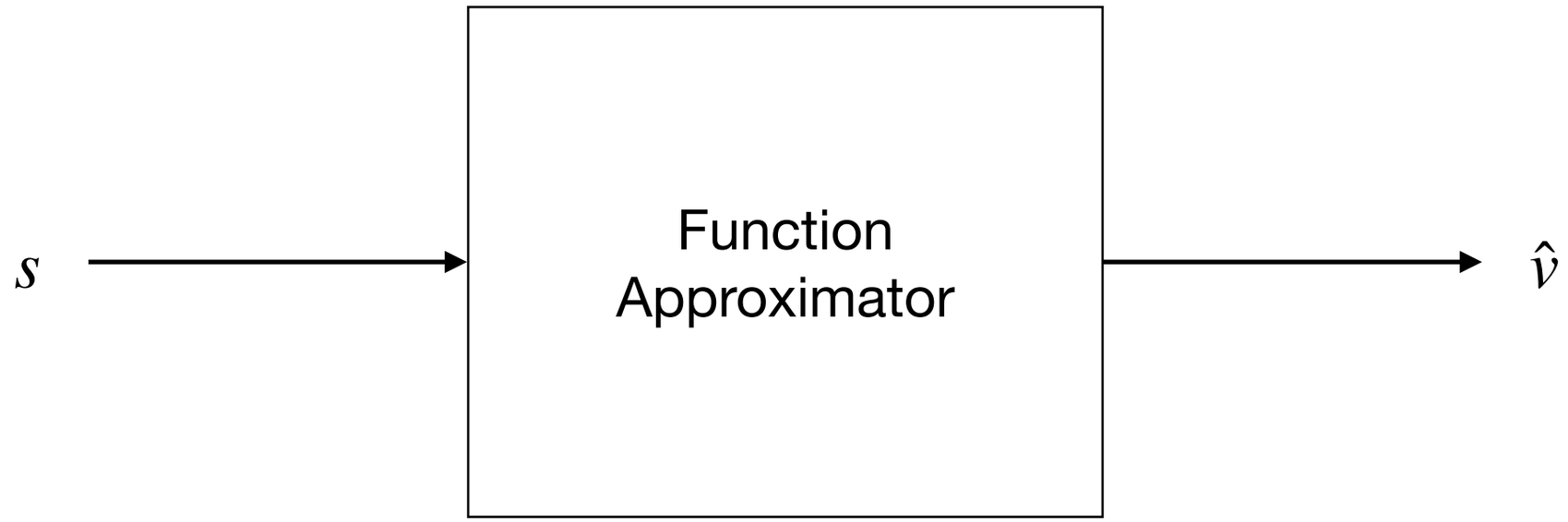
Going Beyond Tabular Methods

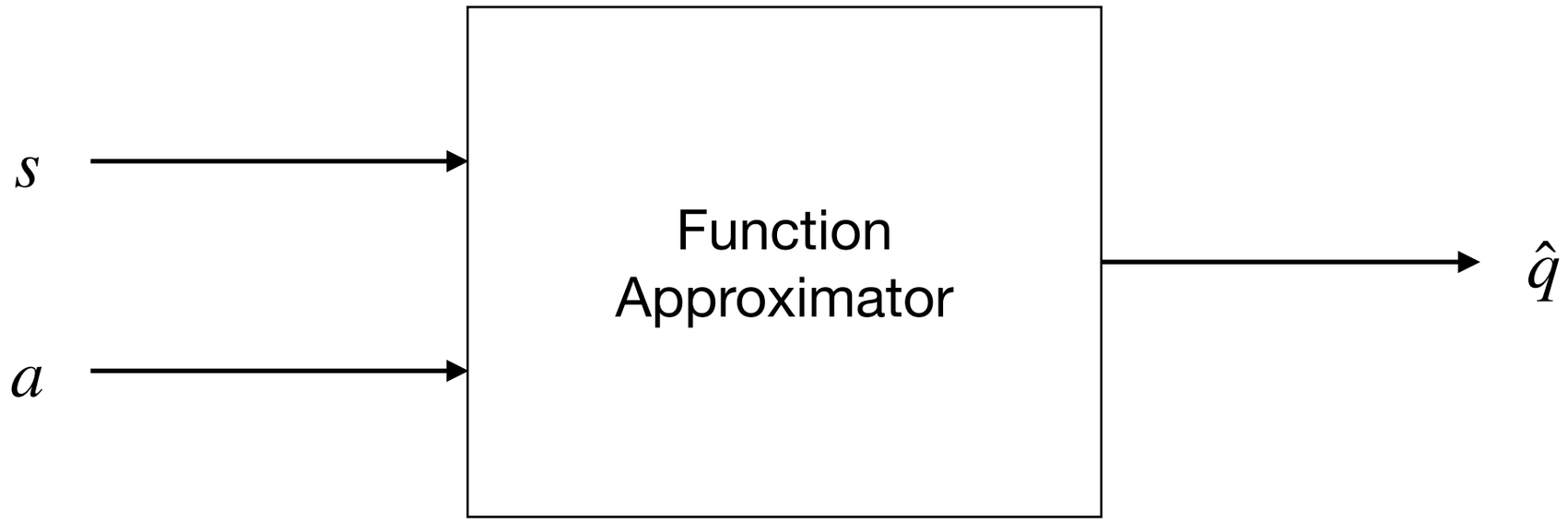
- Tabular methods derive their name from the fact that the state-action space can fit in a table
  - Table with 1 row per state-action entry.
- They are among the most-used methods in RL.
- However, what happens if you can't fit all the state-action entry in a table?
  - We need *function approximation* rather than tables.

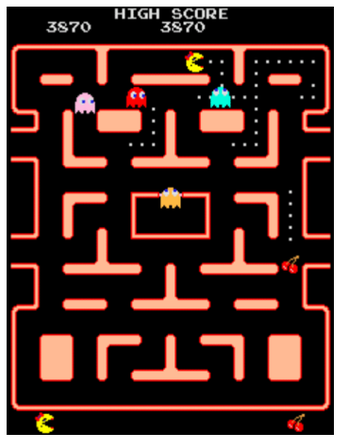
Going Beyond Tabular Methods

- Function Approximation will provide a mapping between a state or state-action to a value function.

- More precisely, a value-function approximation is a function with in input the state (or the state and action), which gives in output the value for the state (or the state and action).


s
Function
Approximator
$\hat{v}$


s
a
Function
Approximator
$\hat{q}$

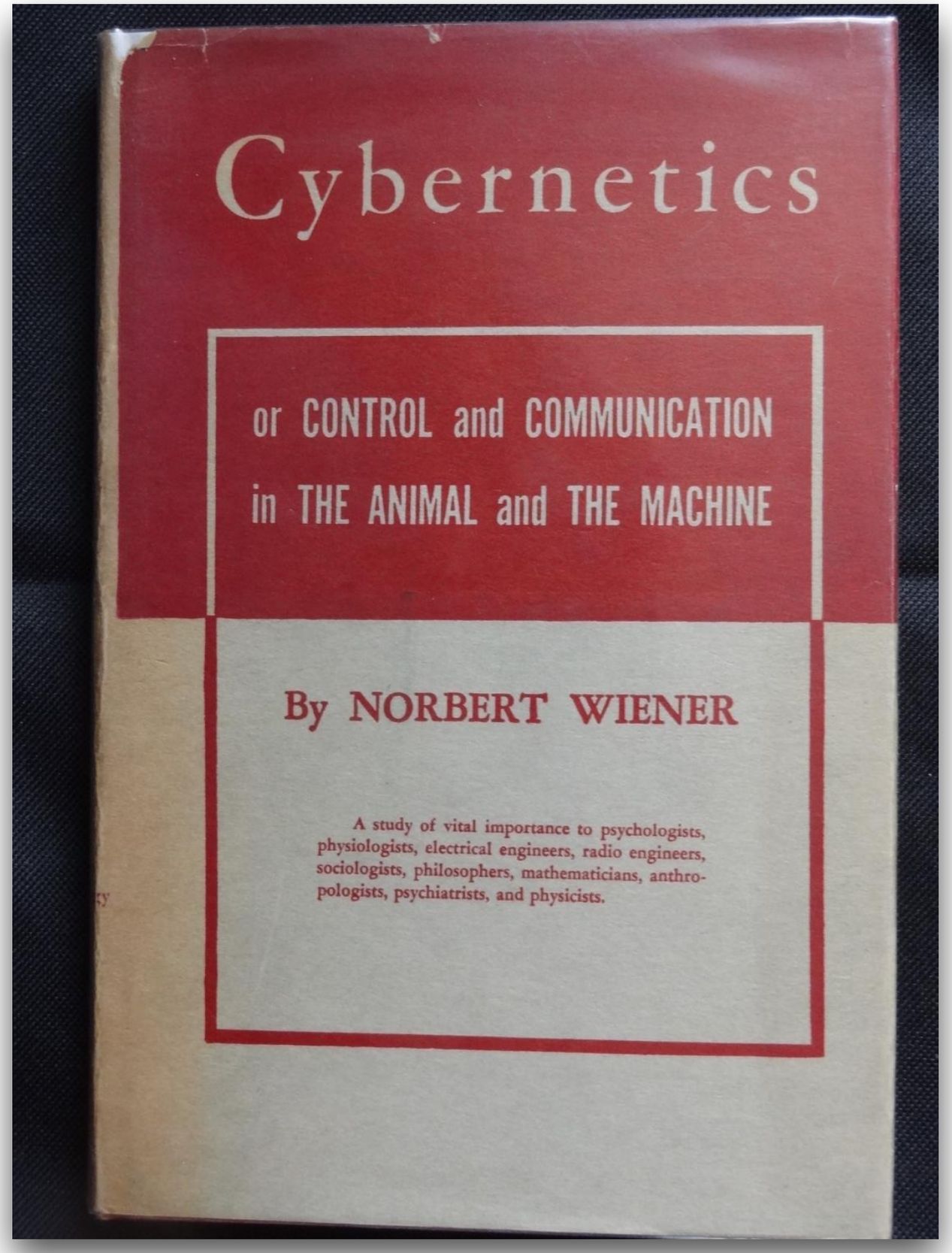


Function Approximator

$\hat{q}$

From Theories of Biological Learning to Deep Learning

- Three waves:
  - Cybernetics (1940s-1960s)
  - Connectionism (1980s-1990s)
  - Deep learning (2006-today)
- Some of the earliest learning algorithms were intended to be computational models of the brain. As a result, one of the names used for deep learning is *artificial neural networks (ANNs)*.

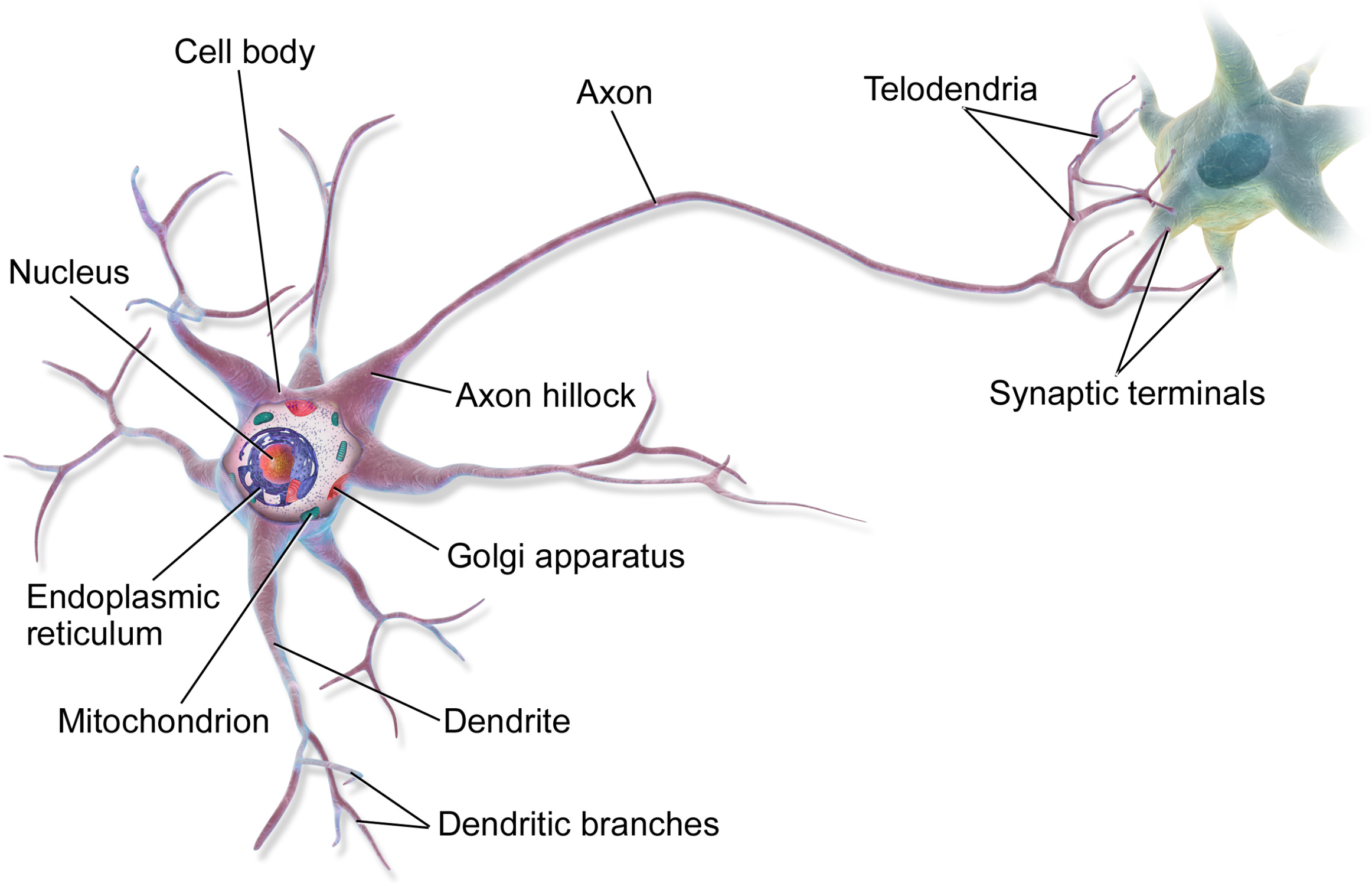

Cybernetics
or CONTROL and COMMUNICATION
in THE ANIMAL and THE MACHINE
By NORBERT WIENER
A study of vital importance to psychologists, physiologists, electrical engineers, radio engineers, sociologists, philosophers, mathematicians, anthropologists, psychiatrists, and physicists.

Artificial Neural Networks and Neuroscience

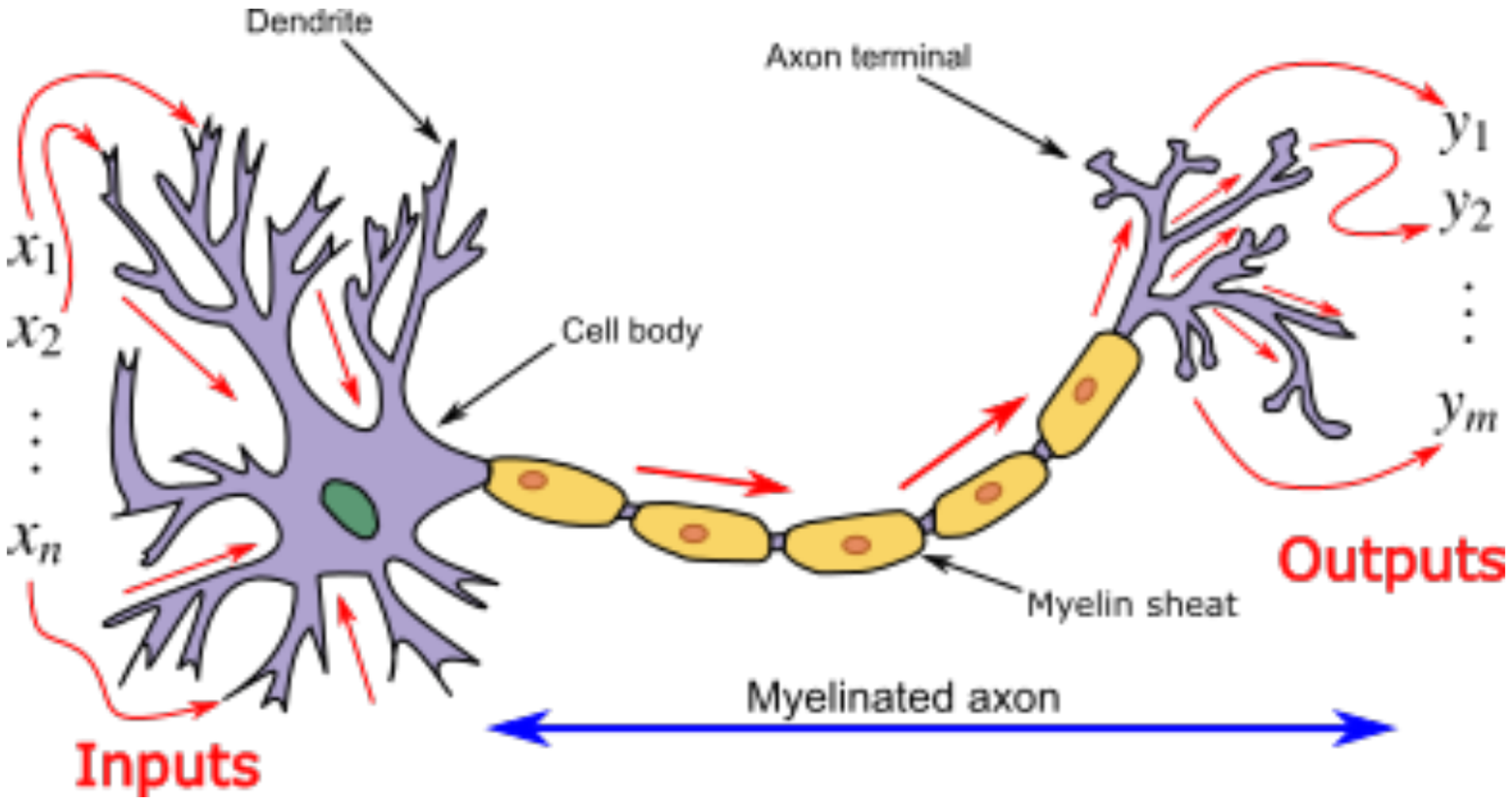
- The earliest predecessors of modern deep learning were simple linear models motivated from a neuroscience perspective.

- These models were designed to take a series of n input values $x_1, x_2, \ldots, x_n$ and associate them to an output y.

- These models would be based or learn a set of weights:
$y = f(\mathbf{x}, \mathbf{w}) = w_1 x_1 + \ldots + w_n x_n$



Credit: Wikimedia



Credit: Wikimedia

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

I. Introduction

Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from less than one meter per second in thin axons, which are usually short, to more than 150 meters per second in thick axons, which are usually long. The time for axonal conduction is consequently of little impor-

der the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

I. Introduction

Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurrence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from less than one meter per second in thin axons, which are usually short, to more than 150 meters per second in thick axons, which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequally remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this depends upon irreciprocity of individual synapses or merely upon prevalent anatomical configurations. To suppose the latter requires no hypothesis *ad hoc* and explains known exceptions, but any assumption as to cause is compatible with the calculus to come. No case is known in which excitation through a single synapse has elicited a nervous impulse in any neuron, whereas any

The McCulloch-Pitts Neural Model

- In “A Logical Calculus of the Ideas Imminent in Nervous Activity” (1943), McCulloch and Pitts suggested a model about how thought executes.

- This is the original inspiration of current deep learning models.

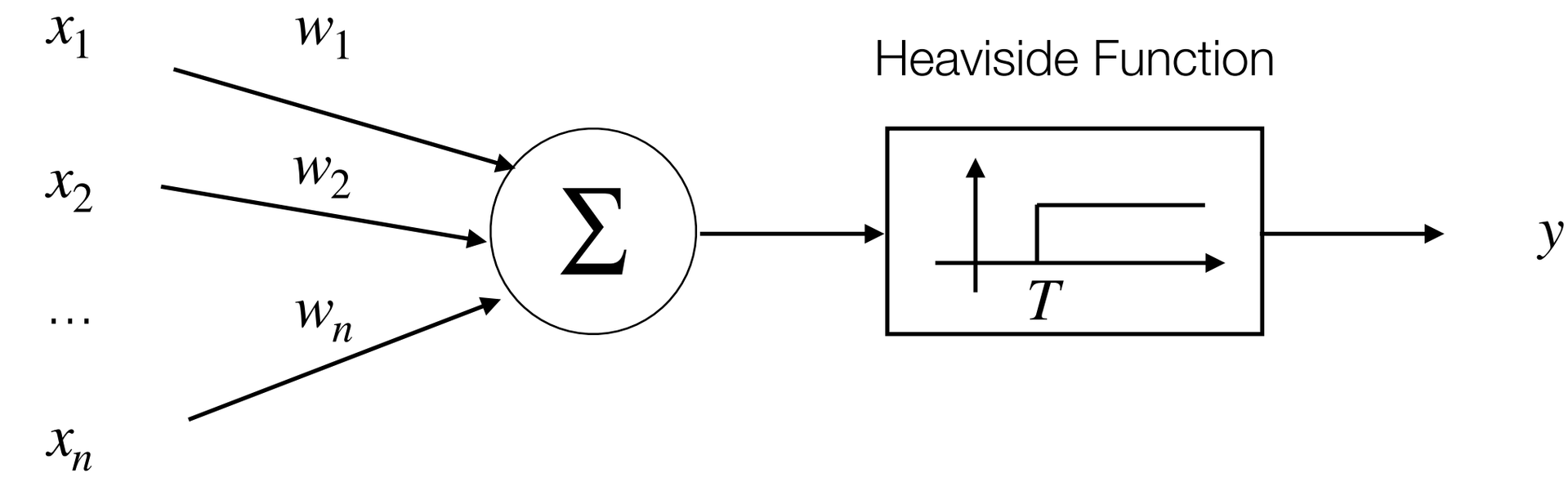
- The set of operations is defined over two values:

  - True (1)

  - False (0)

- The calculus contained NOT, AND, OR. By changing the (fixed) values of the weights, you can obtain different functions.

McCulloch-Pitts Model of Neuron



In the McCulloch-Pitts model, the values of the weights are fixed.

The Organization of Behavior

A NEUROPSYCHOLOGICAL THEORY

D. O. HEBB

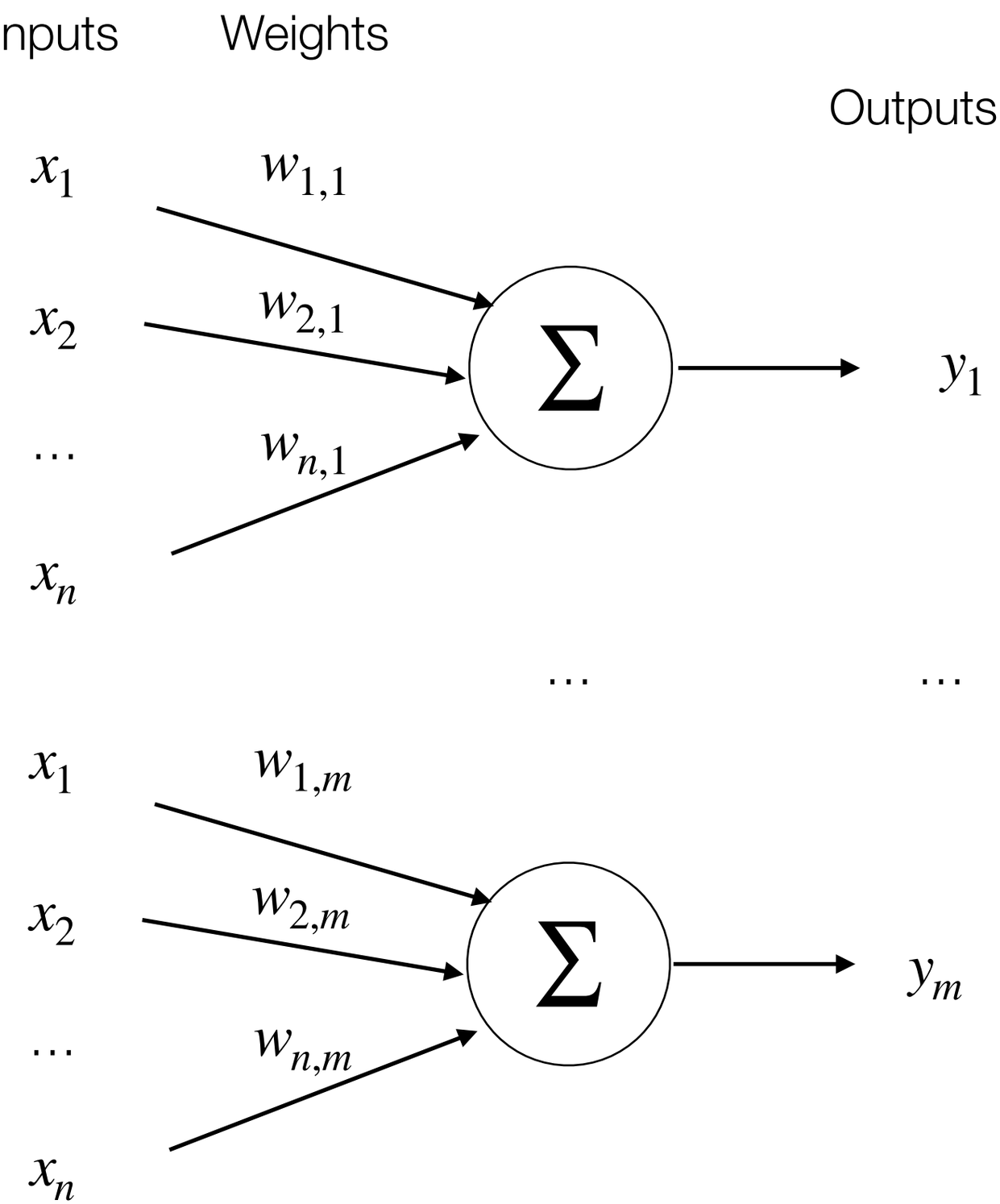
McGill University

1949

New York · JOHN WILEY & SONS, *Inc.*

London · CHAPMAN & HALL, *Limited*

Hebb's Model of Neuron



The Hebbian network model has n-node input layer:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$$

and an m-node output layer

$$\mathbf{y} = [y_1, y_2, \ldots, y_m]^T$$

Each output is connected to all input as follow:

$$y_i = \sum_{j=1}^{n} w_{j,i} x_j$$

The learning rule is the following:

$$w_{j,i}^{new} \leftarrow w_{j,i}^{old} + \eta x_j y_i$$

η is the learning rate.

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

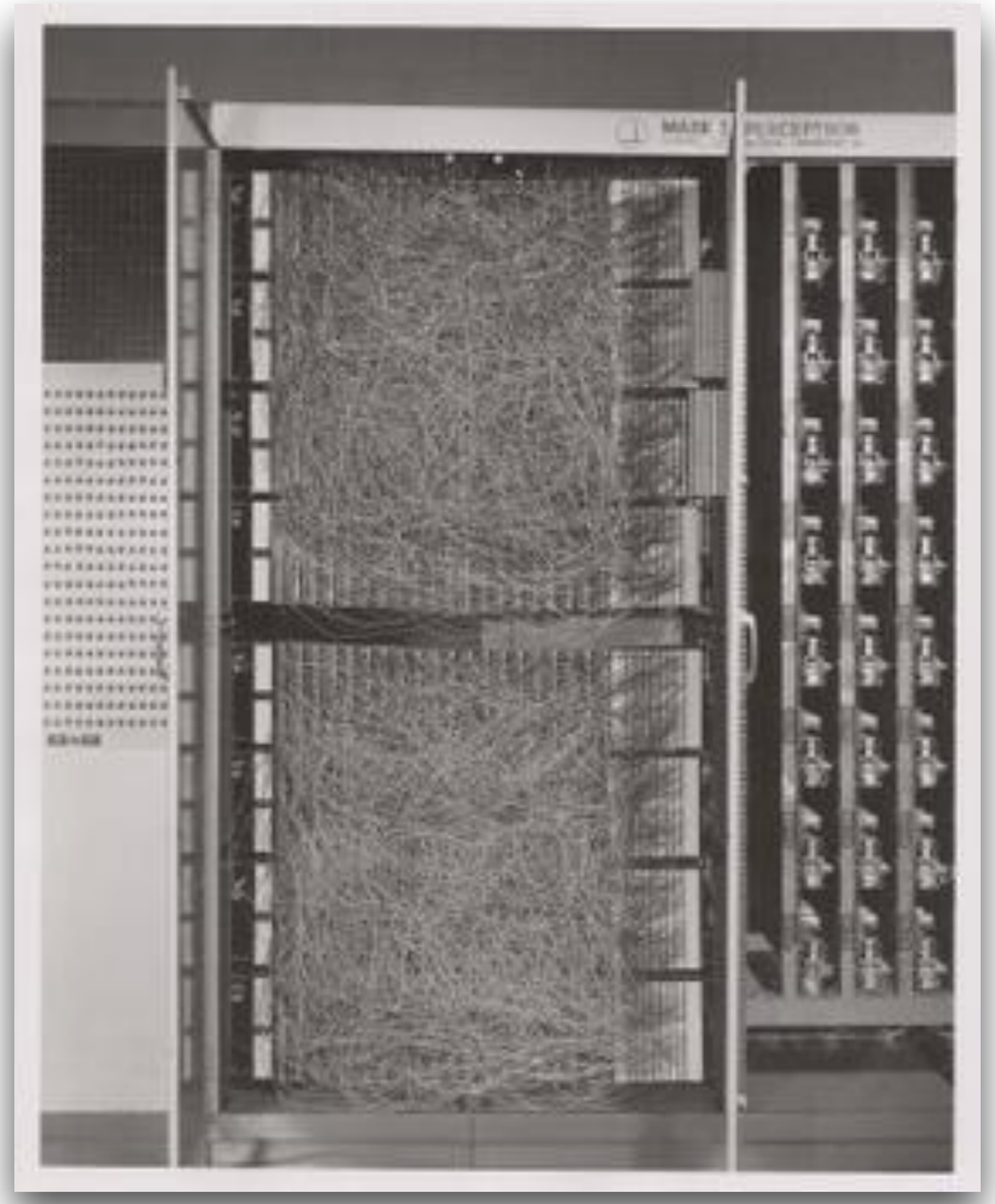
1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility,

Rosenblatt's Perceptron Model

- Frank Rosenblatt's perceptron models the first one with variables weights that were learned from examples:
  - Learning the weights of categories given examples of those categories.
- The perceptron was intended to be a machine rather than a program.
  - First implementation was actually for IBM 704.

IBM 704



Mark I Perceptron

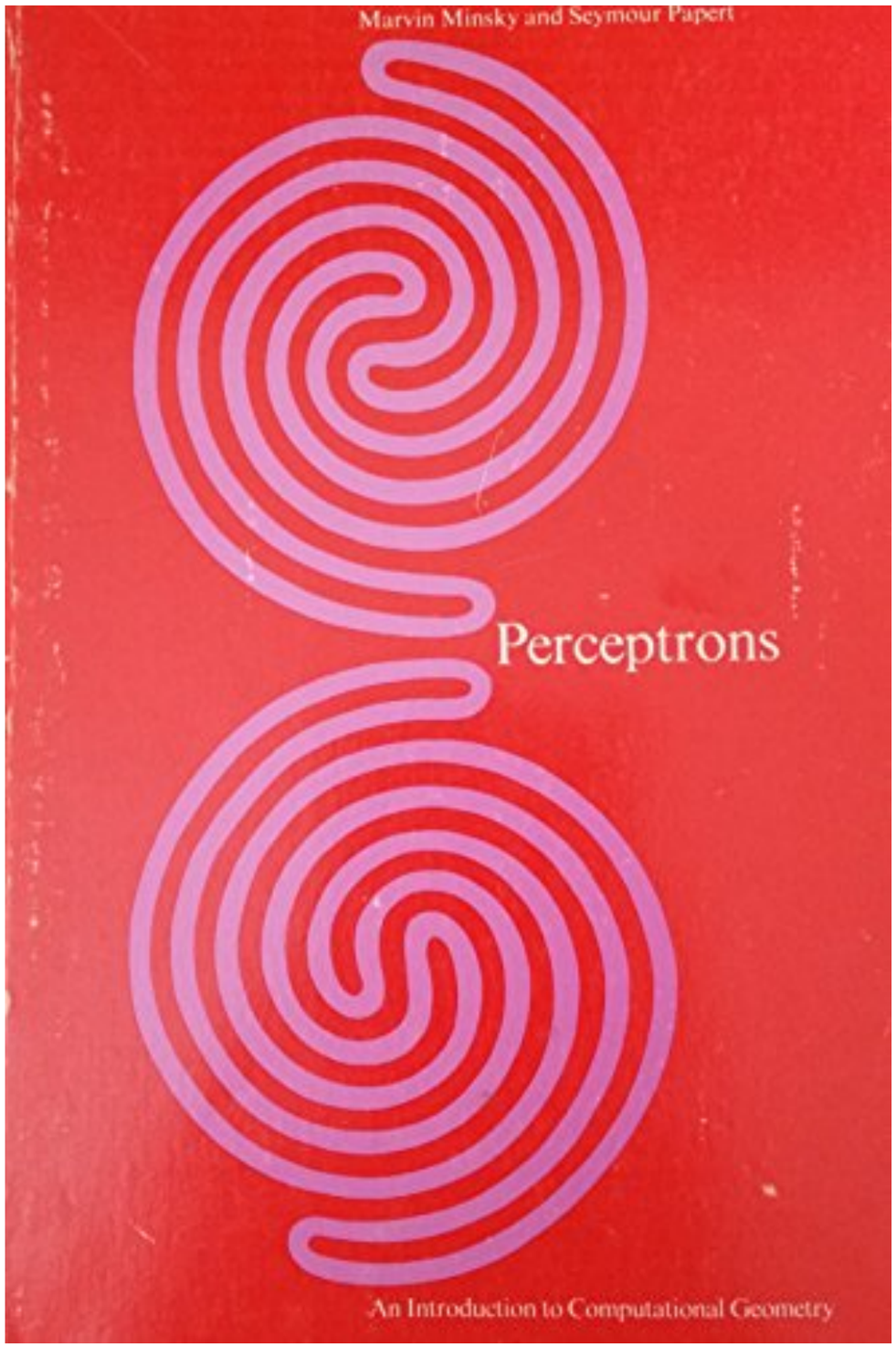
Built for image recognition

400 photocells randomly connected to the neurons

Weights encoded in potentiometers

Limitations of Perceptrons

- Linear models have many limitations.
- Most famously, they cannot learn the XOR function where $f([0,1], \mathbf{w}) = 1$ and $f([1,0], \mathbf{w}) = 1$ but $f([1,1], \mathbf{w}) = 0$ and $f([0,0], \mathbf{w}) = 0$.
- This was observed by Minsky and Papert in 1969 in *Perceptrons*.
- This was the first major dip in the popularity of neural networks.


Marvin Minsky and Seymour Papert
Perceptrons
An Introduction to Computational Geometry

Neurocognitron and (Convolutional) Multi-layer Neural Networks

- Neuroscience can be an inspiration for the design of novel architectures and solutions.

- The basic idea of having multiple computational units that become intelligent via their interactions with each others is inspired by the brain.

- The *neurocognitron* introduced by Fukushima can be considered as a basis for the modern convolutional networks architectures.

- The neurocognitron was the basis of the modern convolutional network architectures (see Yann LeCun et al.'s LeNet architecture).

Biol. Cybernetics 20, 121—136 (1975)

Cognitron: A Self-organizing Multilayered Neural Network

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan

Received: February 4, 1975

Abstract

A new hypothesis for the organization of synapses between neurons is proposed: "The synapse from neuron x to neuron y is reinforced when x fires provided that no neuron in the vicinity of y is firing stronger than y". By introducing this hypothesis, a new algorithm with which a multilayered neural network is effectively organized can be deduced. A self-organizing multilayered neural network, which is named "cognitron", is constructed following this algorithm, and is simulated on a digital computer. Unlike the organization of a usual brain models such as a three-layered perceptron, the self-organization of a cognitron progresses favorably without having a "teacher" which instructs in all particulars how the individual cells respond. After repetitive presentations of several stimulus patterns, the cognitron is self-organized in such a way that the receptive fields of the cells become relatively larger in a deeper layer. Each cell in the final layer integrates the information from

At present, however, the algorithm with which a neural network is self-organized is not known. Although several hypothesis for it have been proposed, none of them has been physiologically substantiated.

The three-layered perceptron proposed by Rosenblatt (1962) is one of the examples of the brain models based on such hypotheses. For a while after the perceptron was proposed, its capability for information processing was greatly expected, and many research works on it have been made. With the progress of the researches, however, it was gradually revealed that the capability of the perceptron is not so large as it had been expected at the beginning.

Although the perceptron consists of only three

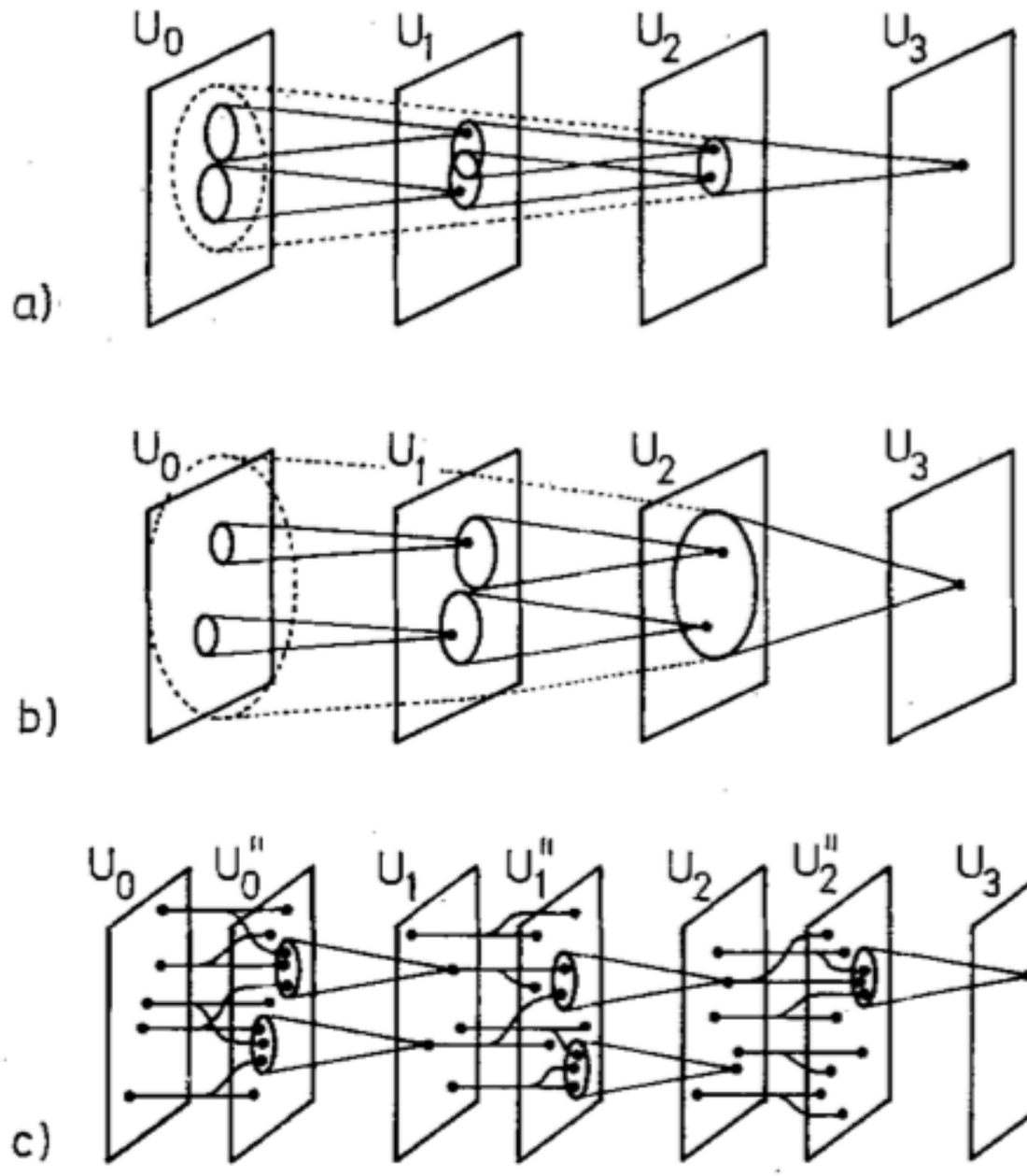


Fig. 4a–c. Three possible methods for interconnecting layers. The connectable area of each cell is differently chosen in these three methods. Method c is adopted for the cognitron discussed in this paper

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

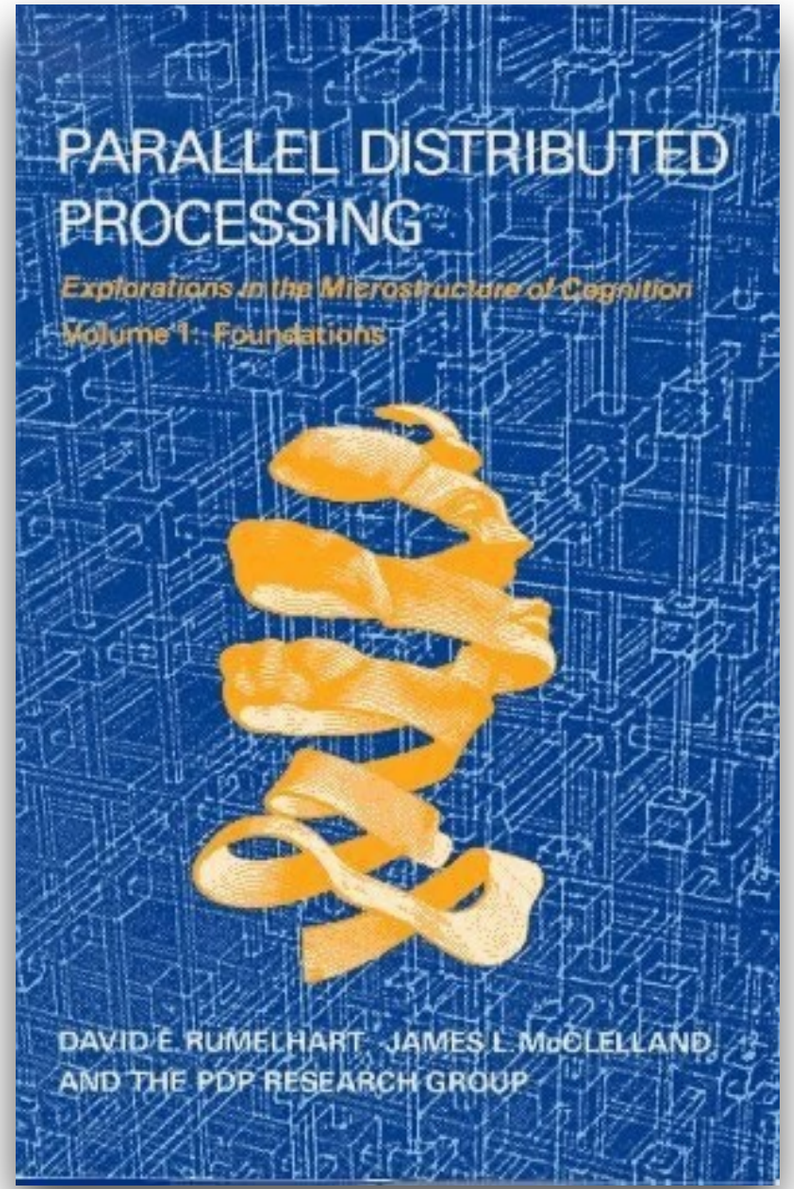
I. Introduction

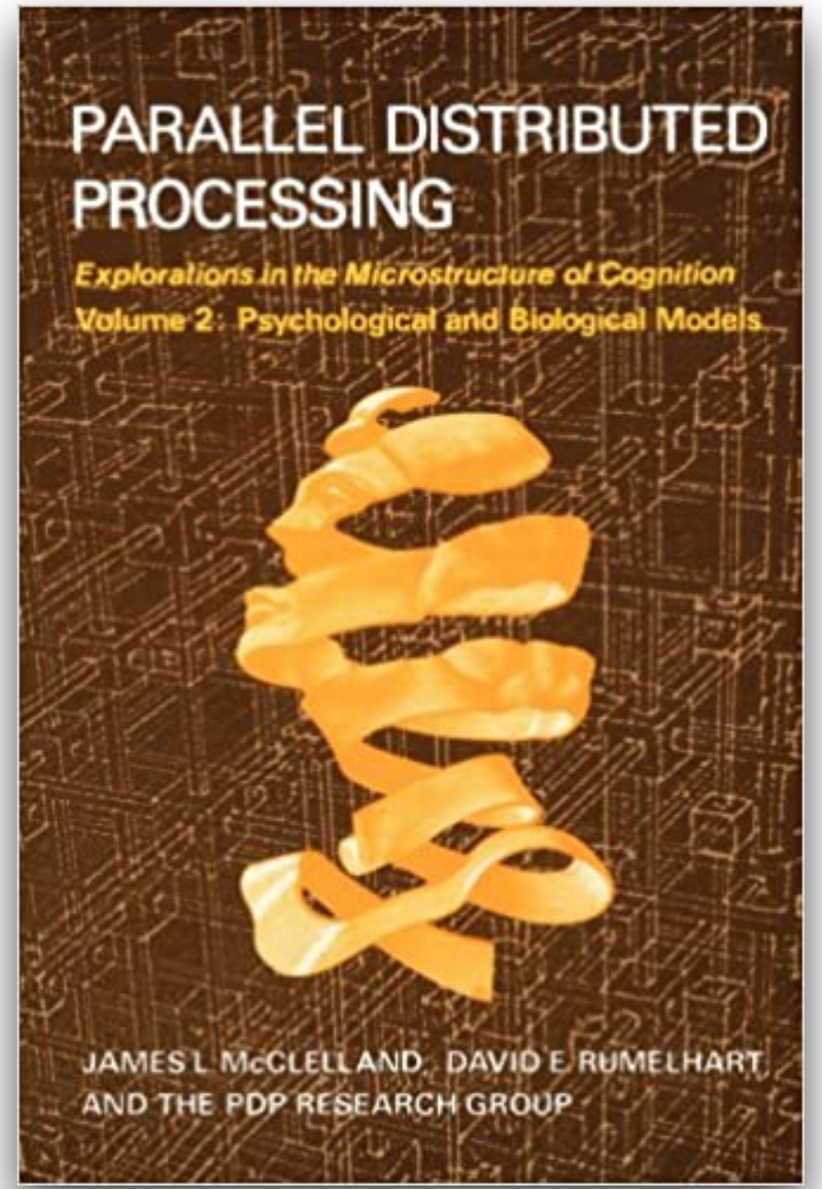
Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics. This is made possible by recent progress in machine learning and computer technology. Using character recognition as a case study, we show that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that operate directly on pixel images. Using document understanding as a case study, we show that the traditional way of building recognition systems by manually integrating individually designed modules can be replaced by a unified and well-principled design paradigm, called *Graph Transformer Networks*, that allows training

Connectionism

- The second wave of neural network research was in 1980s and started in the cognitive science. It was called connectionism or parallel distributed processing.
  - This followed the first winter (mid 70s-1980).
- The focus was on devising models of cognition combining symbolic reasoning and artificial neural network models.
- Many ideas are inspired by Hebb's models.
- The idea of *distributed representation*, i.e., using the raw data without devising features or pre-categorisation of the inputs was introduced by this research movement.
- The other key contribution of connectionism was the development of the *back-propagation algorithm for* training neural networks, which is central in deep learning.


PARALLEL DISTRIBUTED PROCESSING
Explorations in the Microstructure of Cognition
Volume 1: Foundations
DAVID E. RUMELHART, JAMES L. McCLELLAND,
AND THE PDP RESEARCH GROUP


PARALLEL DISTRIBUTED PROCESSING
Explorations in the Microstructure of Cognition
Volume 2: Psychological and Biological Models
JAMES L. McCLELLAND, DAVID E. RUMELHART,
AND THE PDP RESEARCH GROUP

Second AI Winter and Current AI Summer

- The second wave of neural networks lasted until mid 1990s.
  - Loss of interest and lot of disappointment due to unrealistic goals led to a new “winter”.
- During the second winter, a lot of work continued especially in Canada (and NYU).
- The summer returned in 2006 when Geoffrey Hinton showed that a particular neural network called a deep belief network could be very efficiently trained (the strategy is called *greedy layer-wise pre-training*).

A fast learning algorithm for deep belief nets *

Geoffrey E. Hinton and **Simon Osindero**
Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4
{hinton, osindero}@cs.toronto.edu

Yee-Whye Teh
Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg

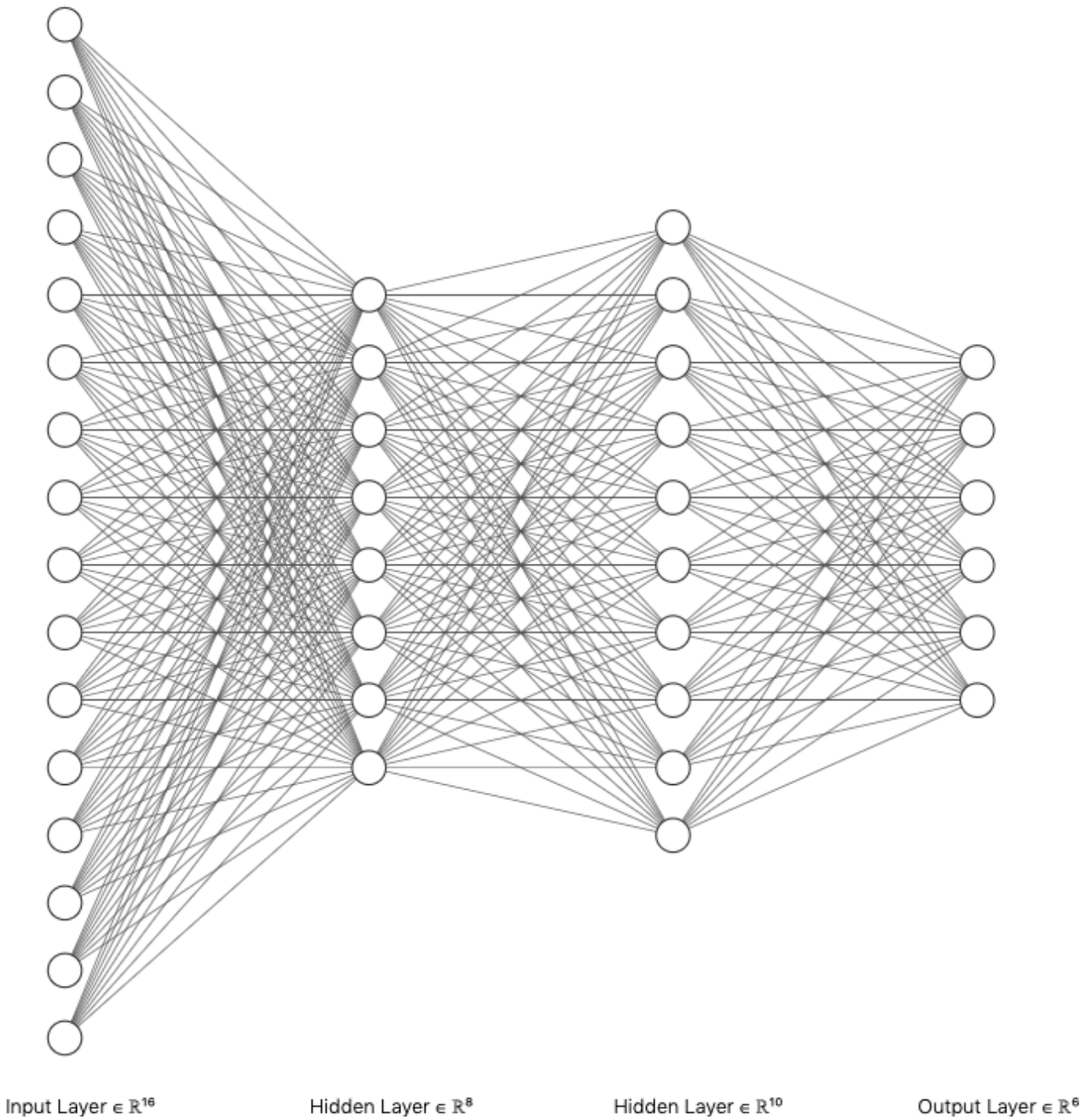
Abstract

We show how to use "complementary priors" to eliminate the explaining away effects that make inference difficult in densely-connected belief nets that have many hidden layers. Using complementary priors, we derive a fast, greedy algorithm that can learn deep, directed belief networks one layer at a time, provided the top two layers form an undirected associative memory. The fast, greedy algorithm is used to initialize a slower learning procedure that fine-tunes the weights using a contrastive version of the wake-sleep algorithm. After fine-tuning, a network with three hidden layers forms a very good generative model of the joint distribution of handwritten digit images and their labels. This generative model gives better digit classification than the best discriminative learning algorithms. The low-dimensional manifolds on which the digits lie are modelled by

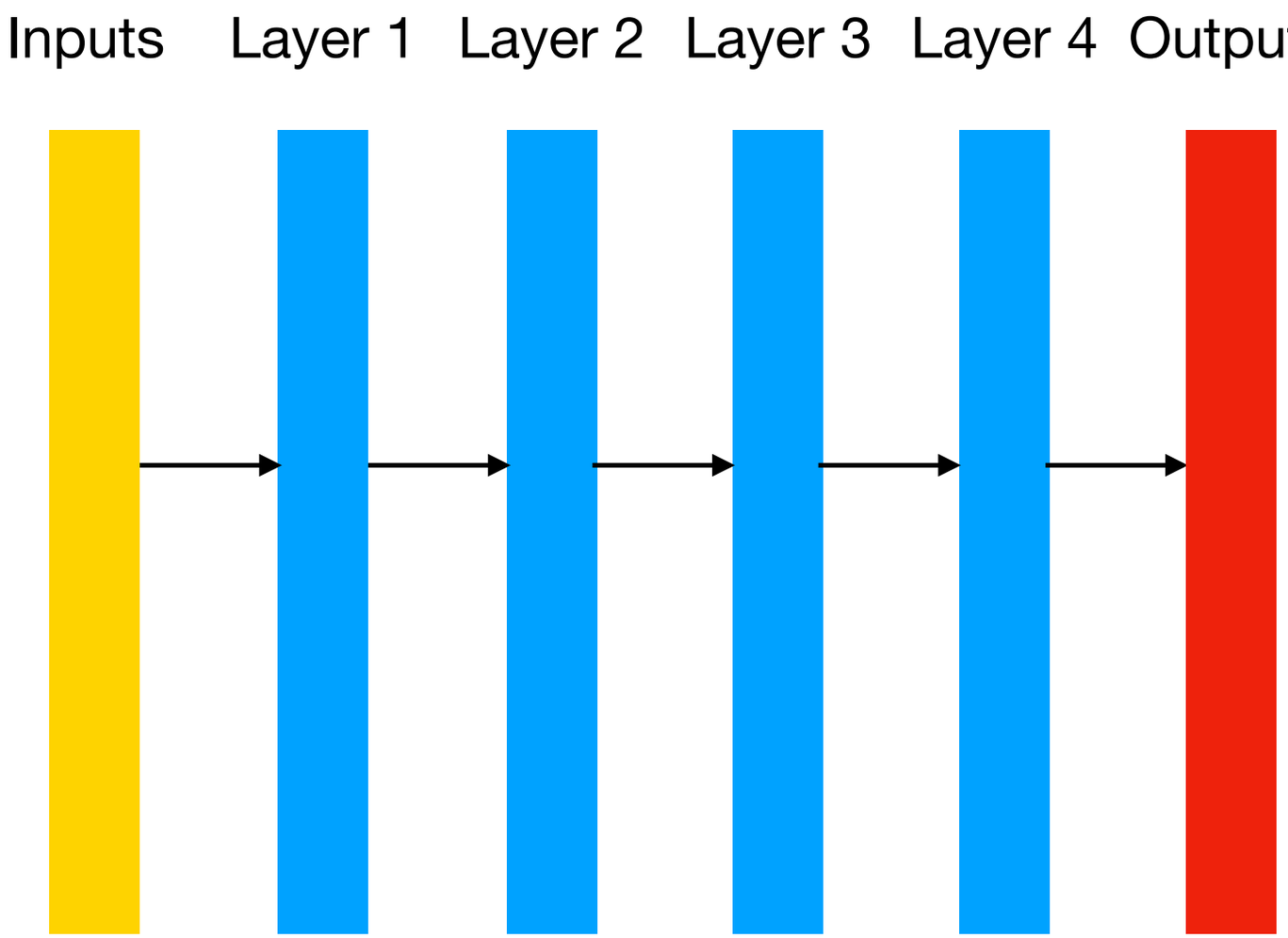
remaining hidden layers form a directed acyclic graph that converts the representations in the associative memory into observable variables such as the pixels of an image. This hybrid model has some attractive features:

1. There is a fast, greedy learning algorithm that can find a fairly good set of parameters quickly, even in deep networks with millions of parameters and many hidden layers.
2. The learning algorithm is unsupervised but can be applied to labeled data by learning a model that generates both the label and the data.
3. There is a fine-tuning algorithm that learns an excellent generative model which outperforms discriminative methods on the MNIST database of hand-written digits.
4. The generative model makes it easy to interpret the distributed representations in the deep hidden layers.
5. The inference required for forming a percept is both fast

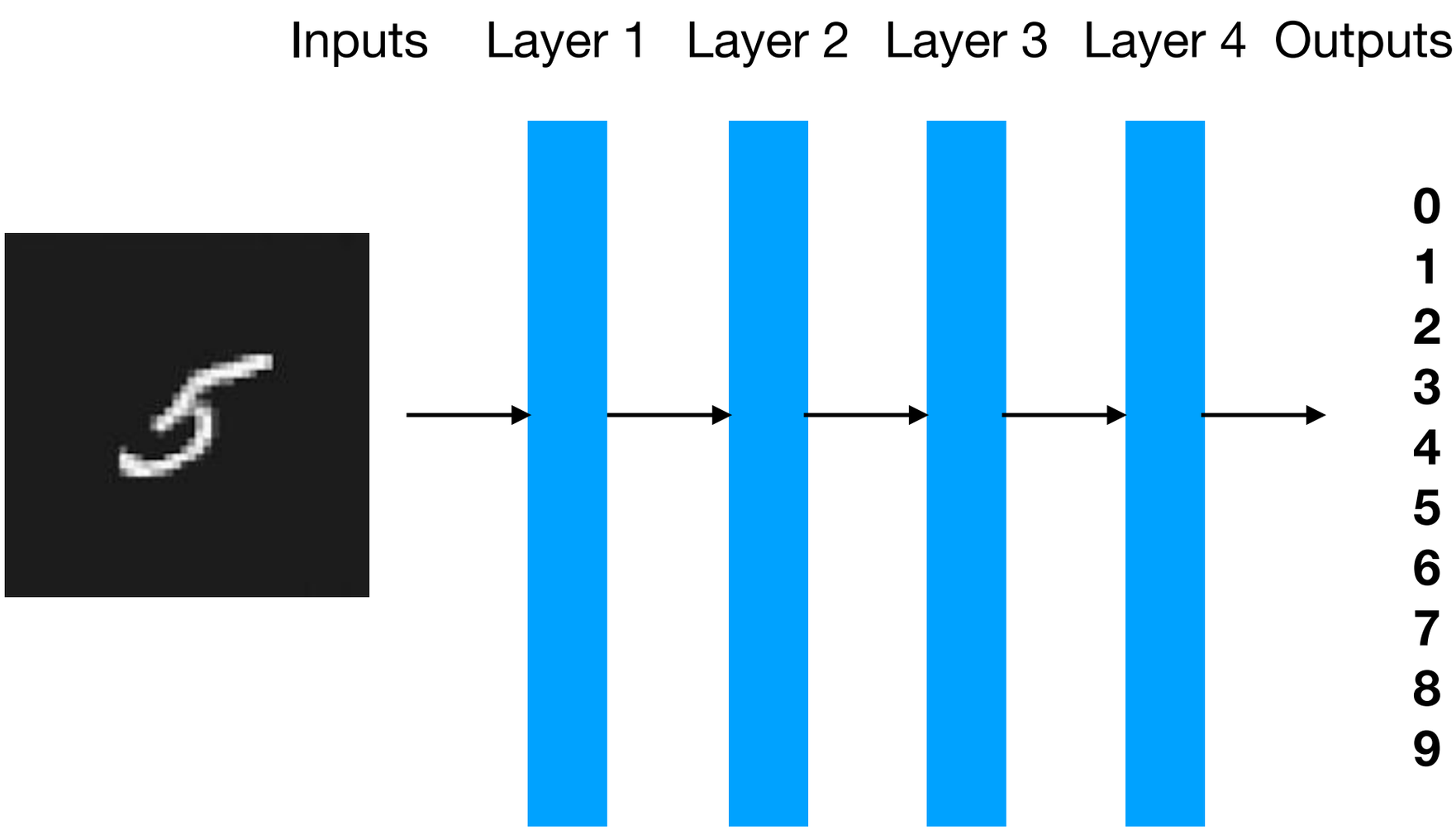
Deep Neural Networks



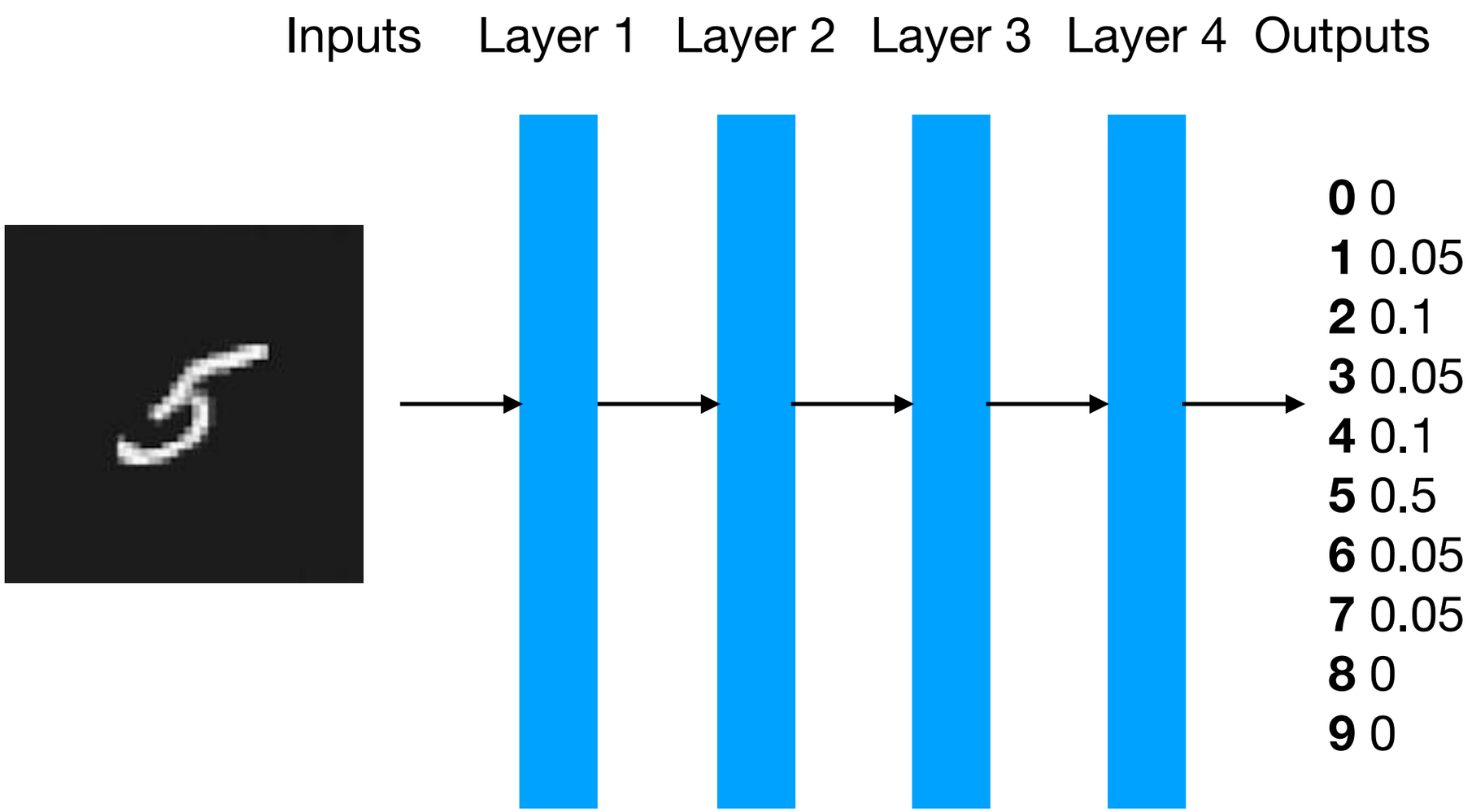
Deep Neural Networks



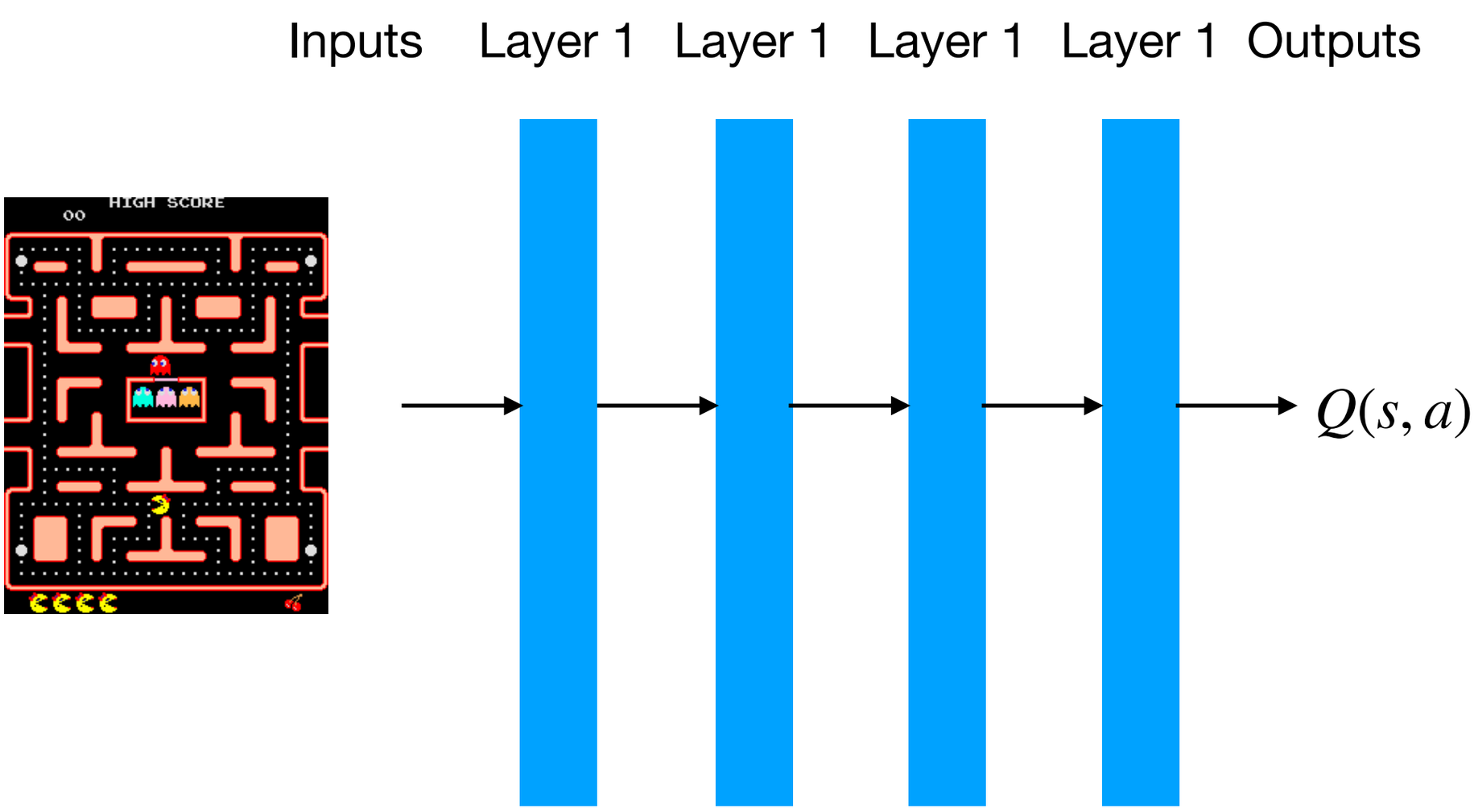
Deep Neural Networks



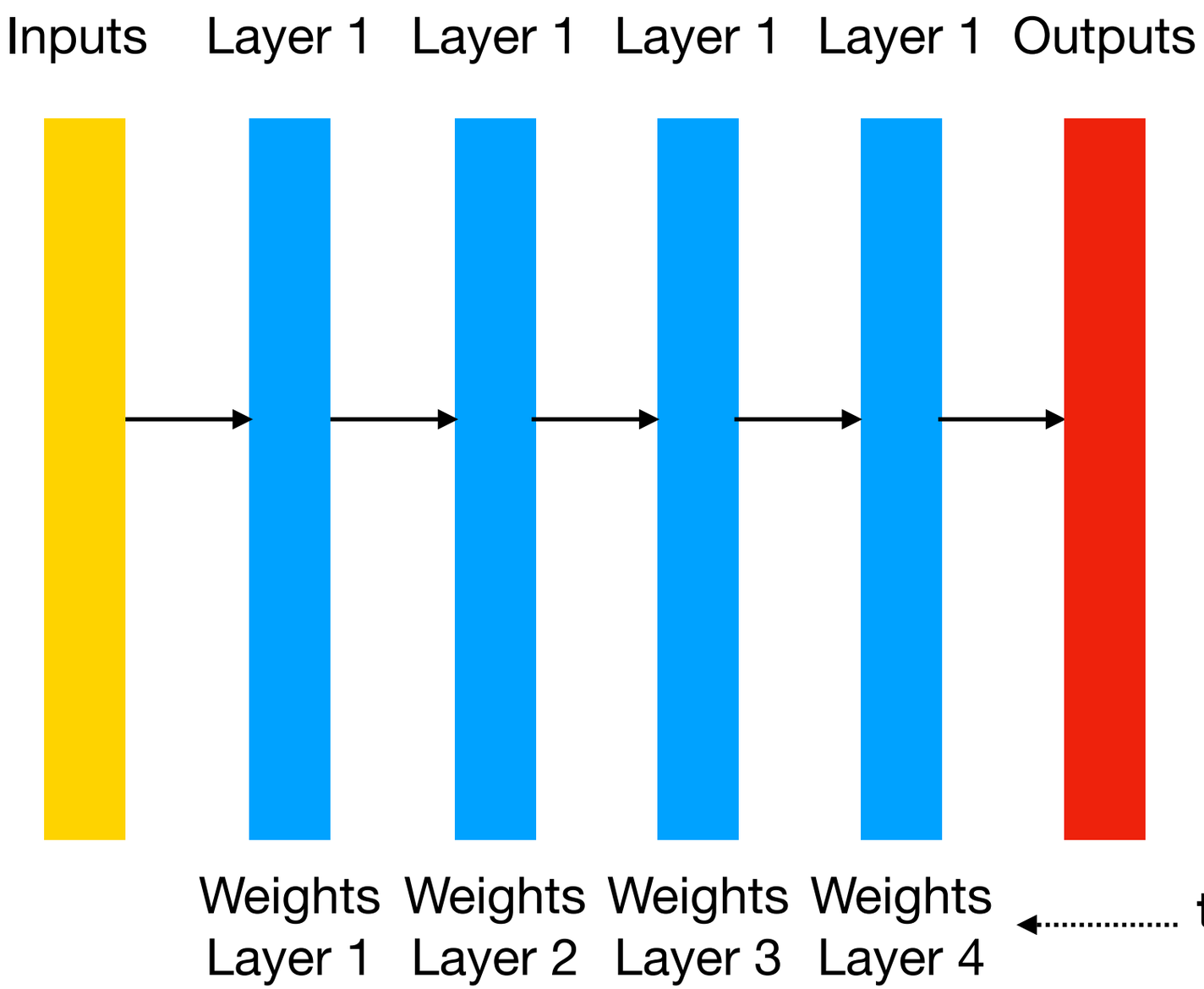
Deep Neural Networks



Deep Neural Networks

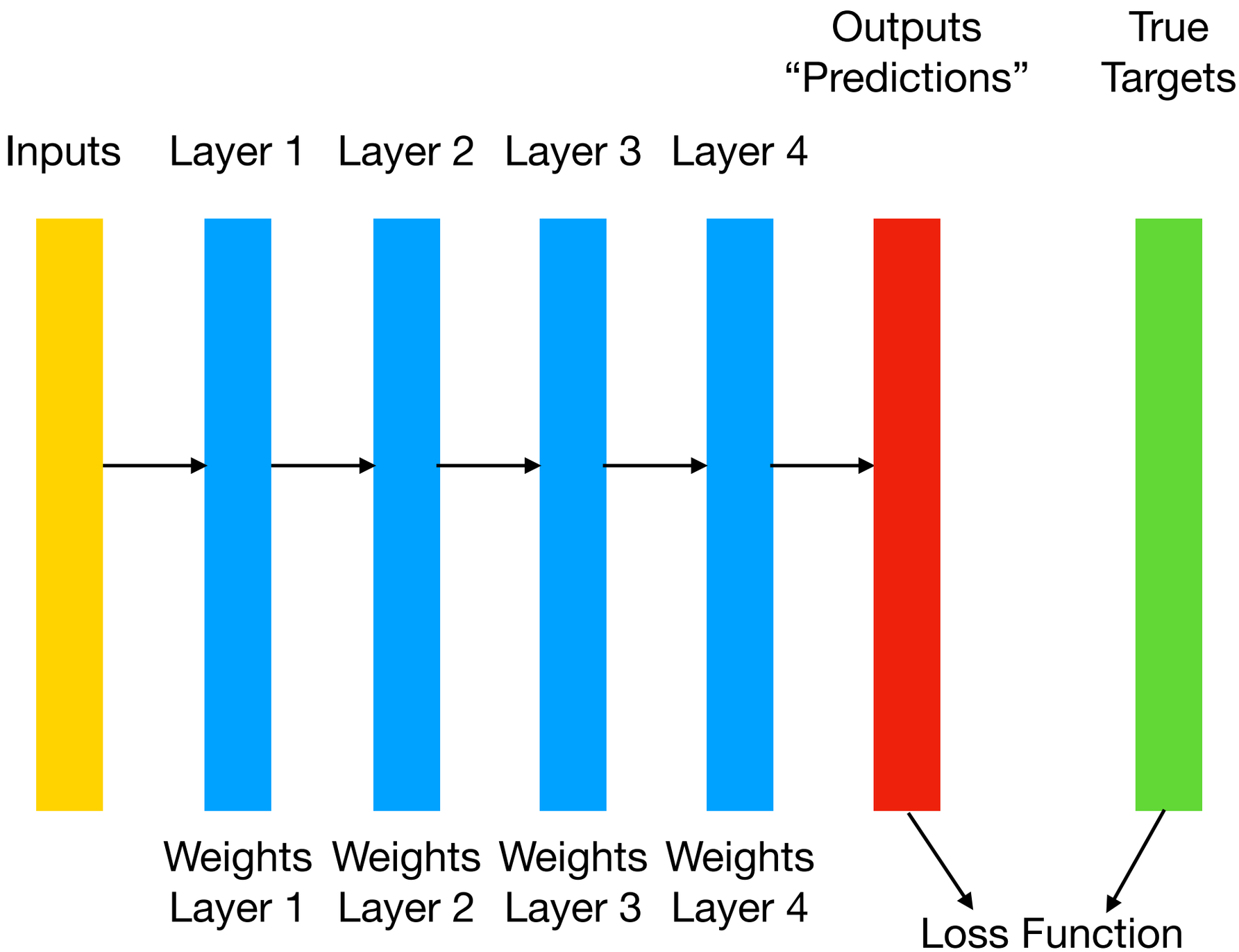


Deep Neural Networks

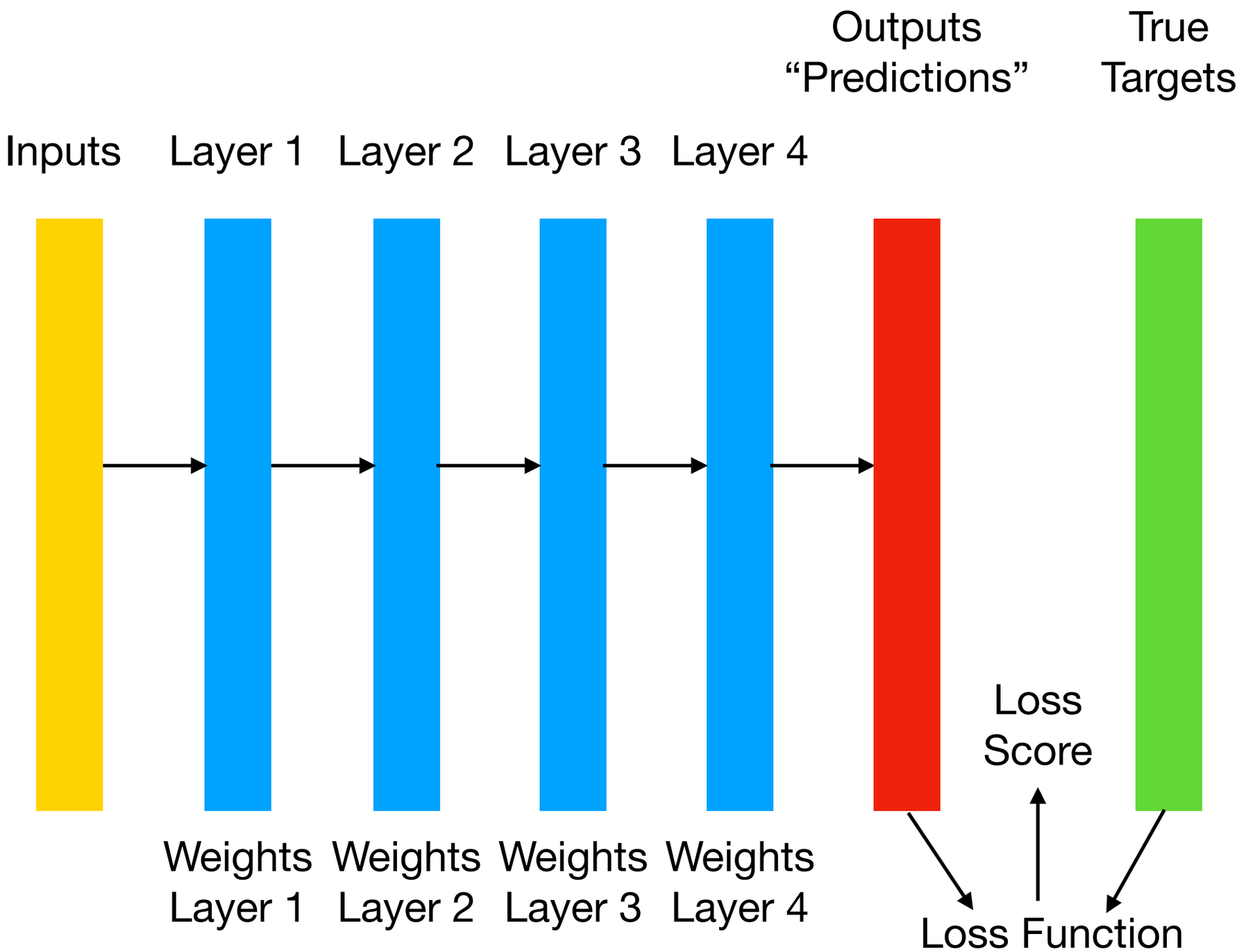


The goal is to find the right values for these weights.

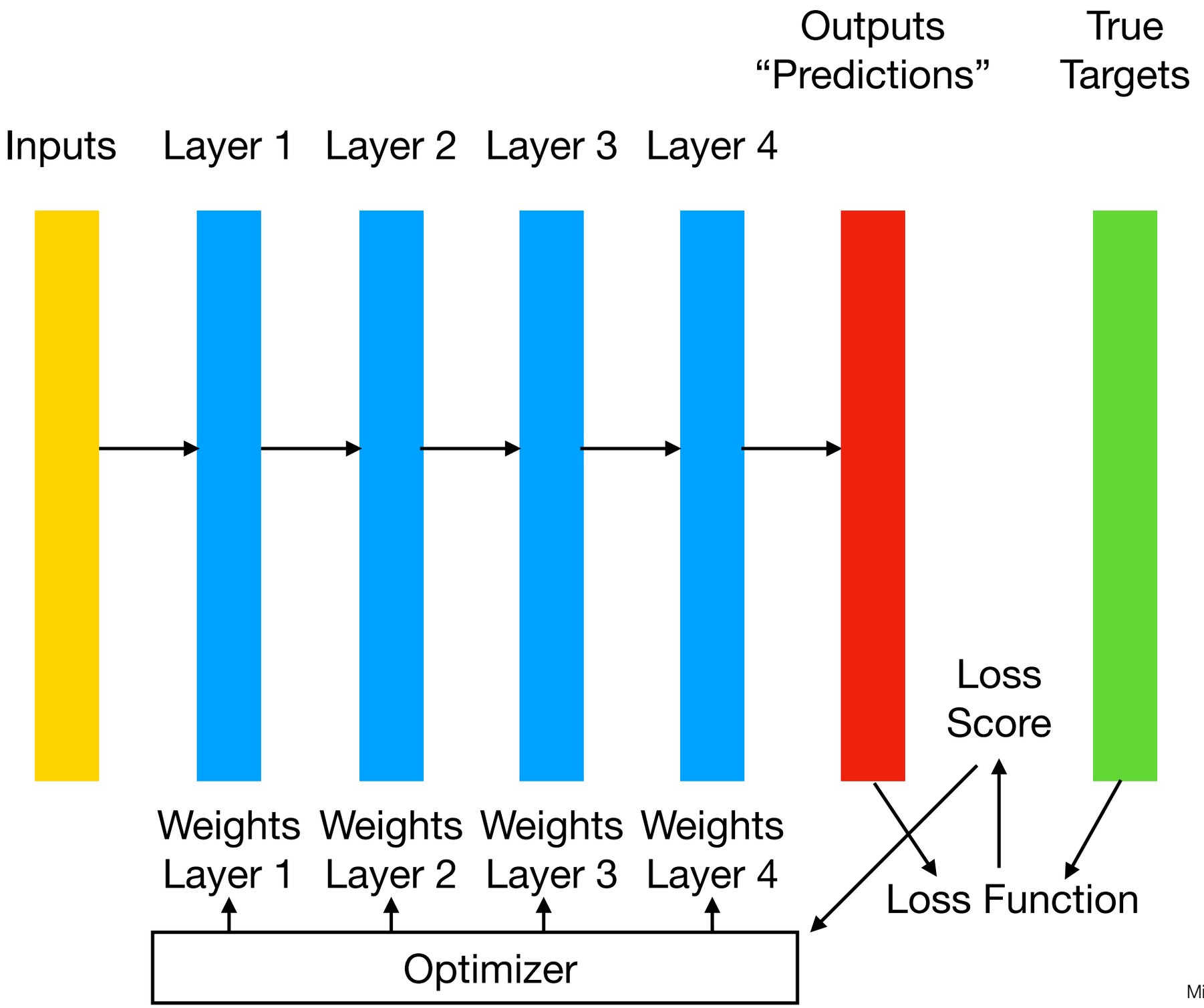
Deep Neural Networks



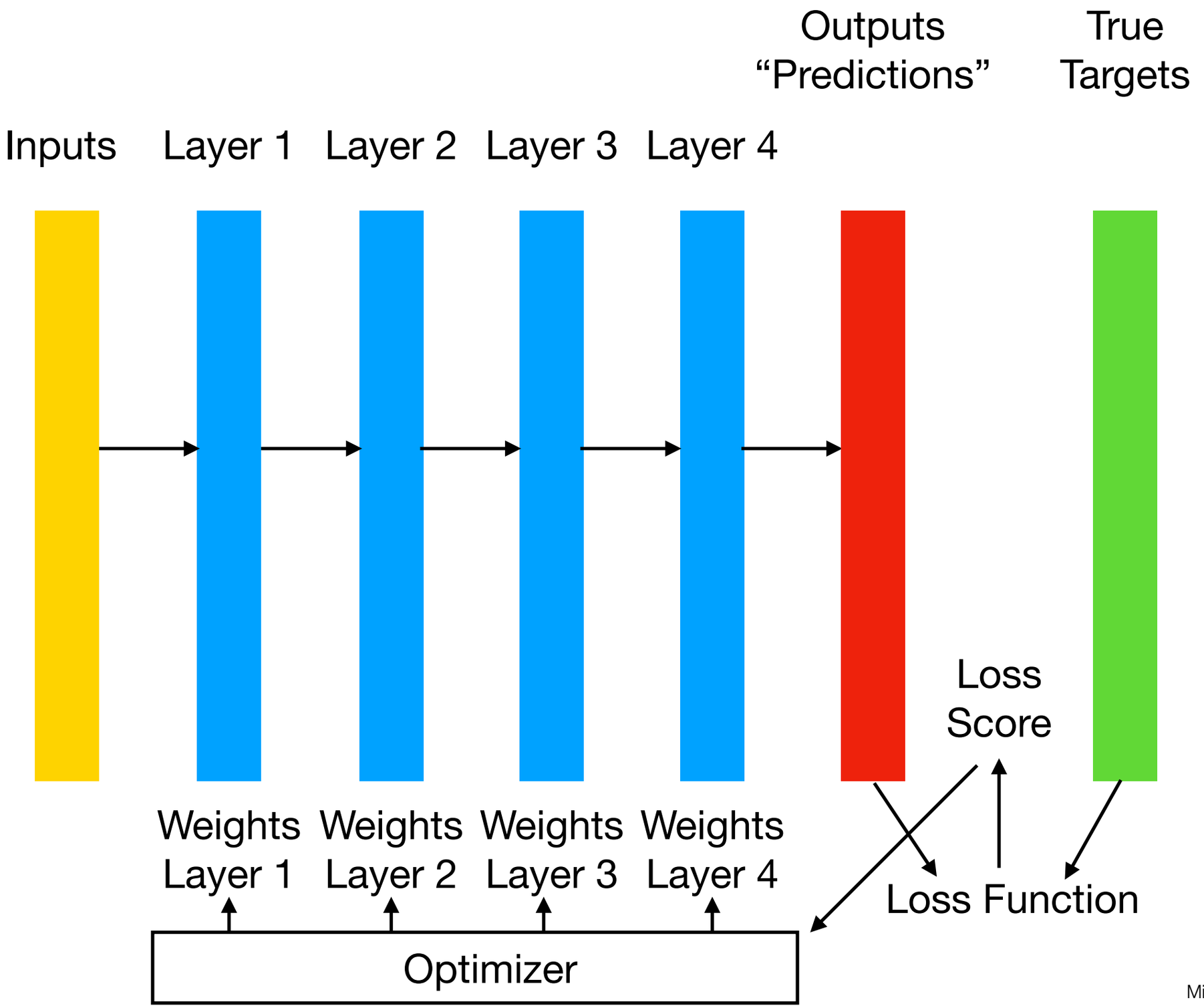
Deep Neural Networks



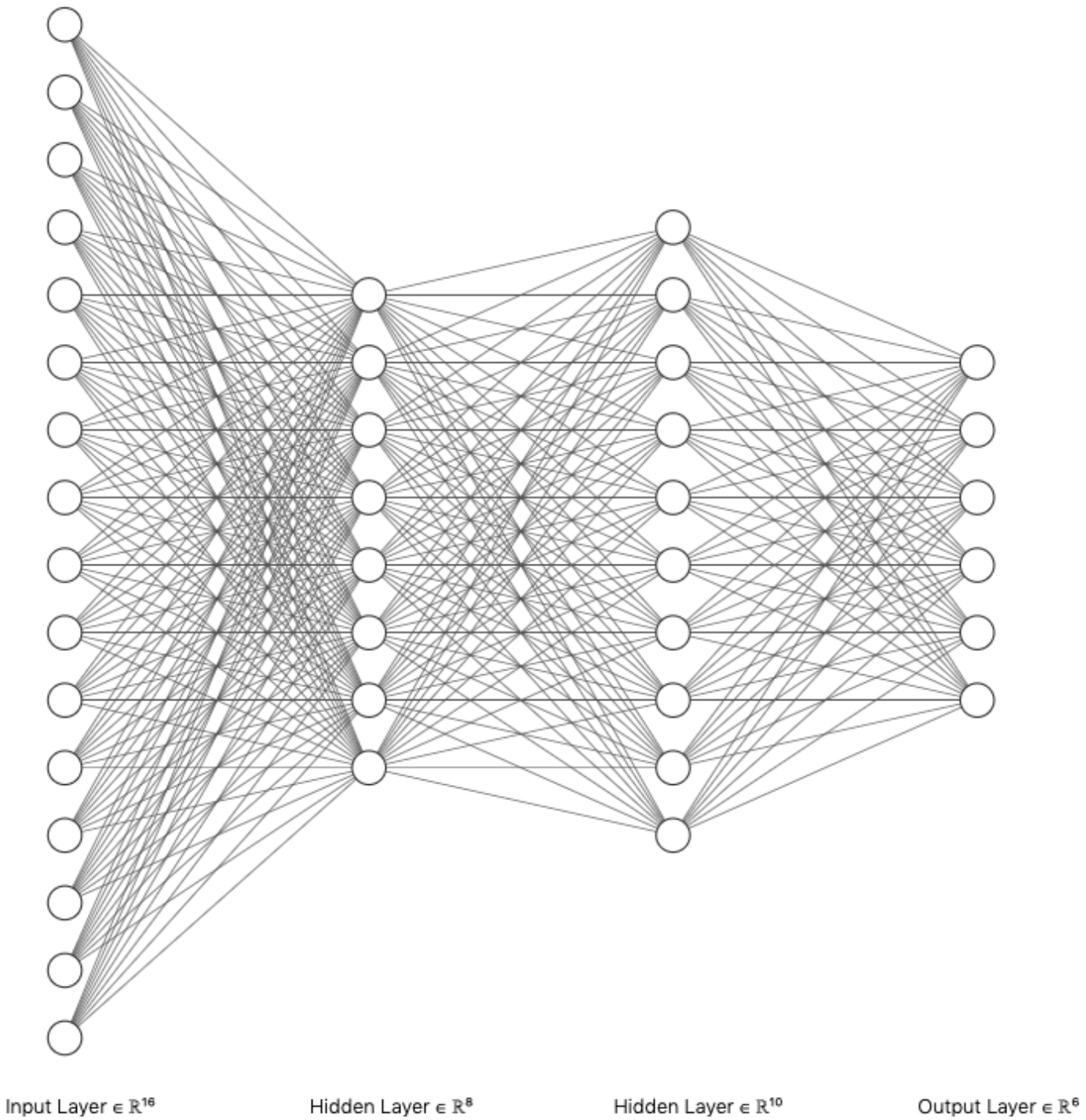
Deep Neural Networks



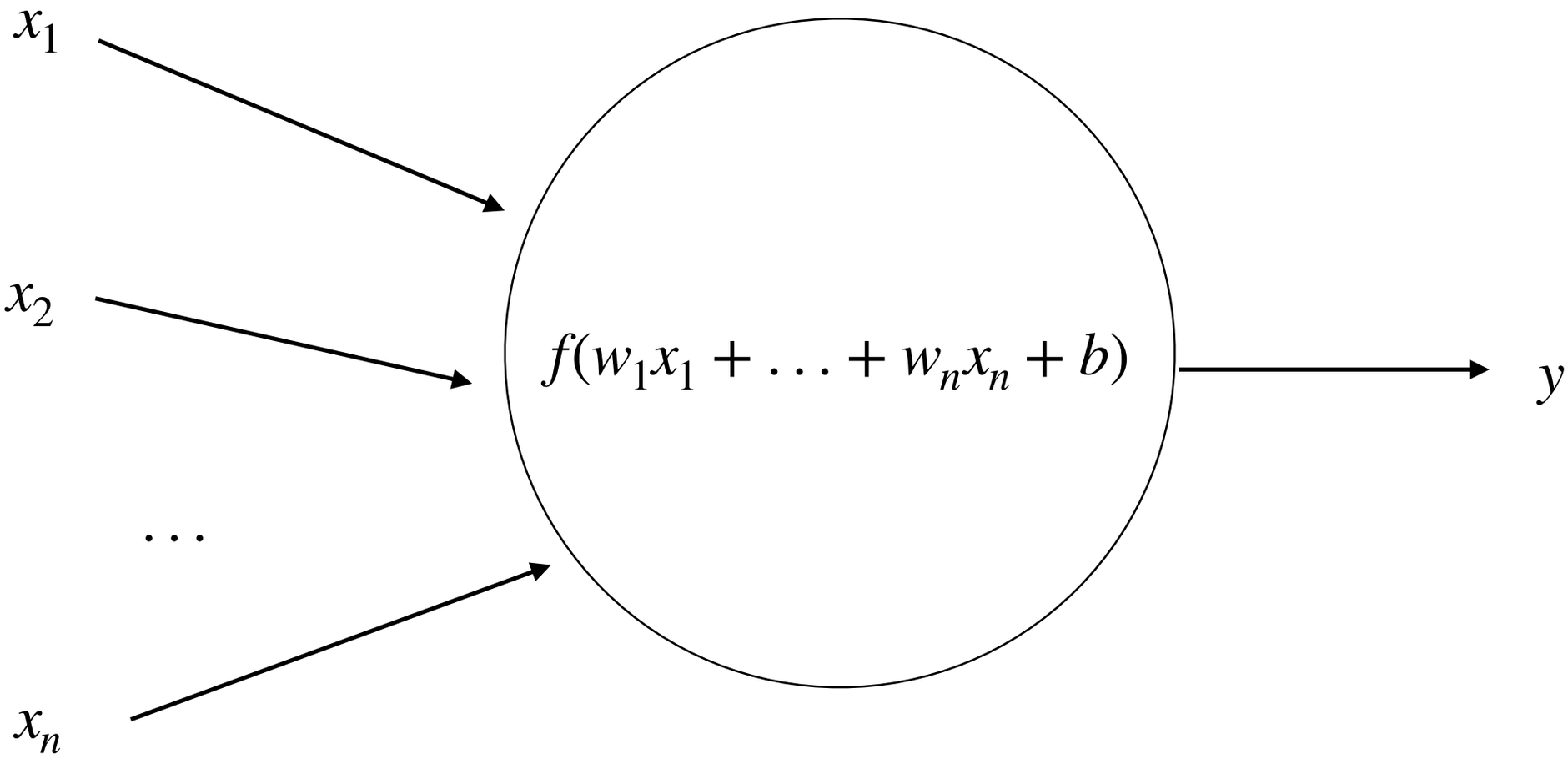
Deep Neural Networks



Deep Neural Networks



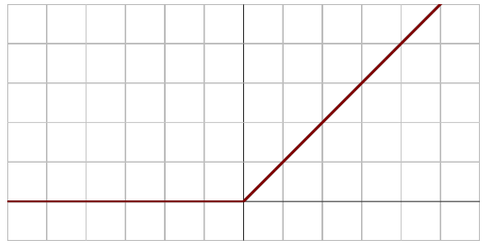
Nodes/Units/Neurons



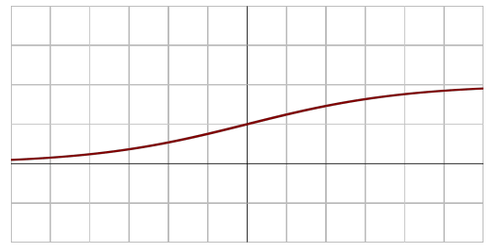
f is called the activation function, b is usually called the bias

Activations Functions

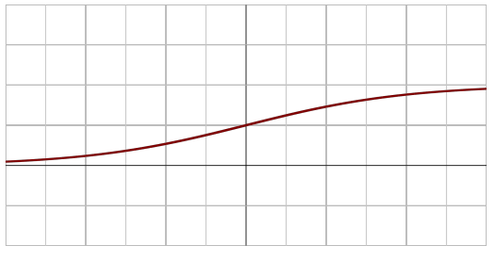
- They are generally used to add non-linearity.
- Examples:
  - *Rectified Linear Unit*: it returns the max between 0 and the value in input. In other words, given the value z in input it returns $max(0,z)$.



  - *Logistic sigmoid*: given the value in input z, it returns $\frac{1}{1+e^{z}}$.

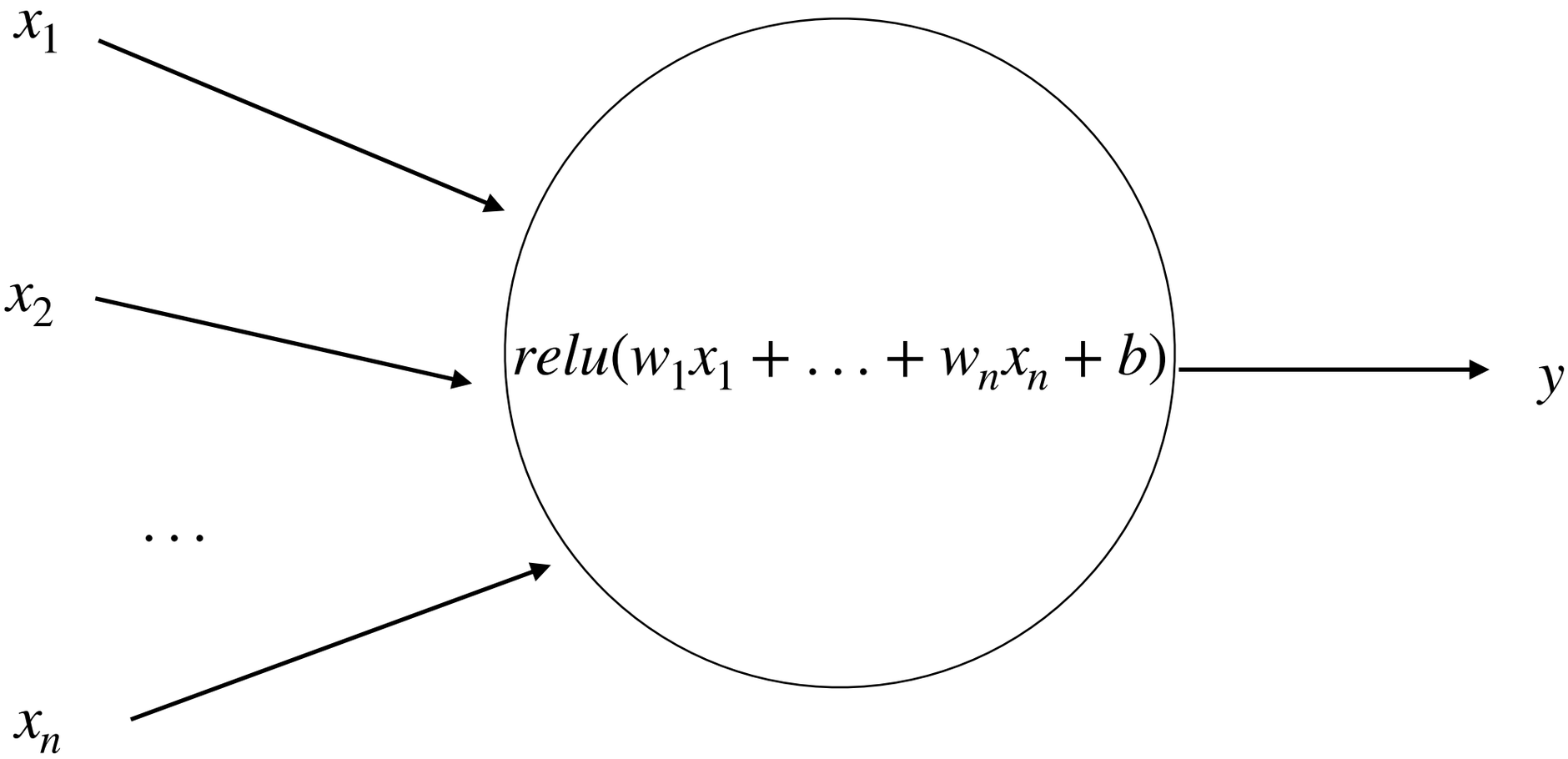


  - *Arctan*: given the value in input z, it returns $tan^{-1}(z)$.



Credit: Wikimedia

Nodes/Units/Neurons



Note that here the function in input of relu is 1-dimensional.

Softmax Function

- Another function that we will use is *softmax*.

- But please note that softmax is not like the activation functions that we discussed before. The activations functions that we discussed before take in input real numbers and returns a real number.

- A softmax function receives in input a vector of real numbers of dimension n and returns a vector of real numbers of dimension n.

- *Softmax*: given a vector of real numbers in input $\mathbf{z}$ of dimension n, it normalises it into a probability distribution consisting of n probabilities proportional to the exponentials of each element z_i of the vector $\mathbf{z}$. More formally,

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \text{ for } i = 1,..n.$$

Gradient-based Optimisation

- We will now discuss a high-level description of the learning process of the network, usually called *gradient-based optimization*.

- Each neural layer transforms his input layer as follows:

$$output = f(w_1 x_1 + \ldots + w_n x_n + b)$$

- And in the case of a relu function, we will have

$$output = relu(w_1 x_1 + \ldots + w_n x_n + b)$$

- Note that this is a simplified notation for one layer, it should be $w_{1,i}$ for layer i.

Gradient-based Optimisation

- The learning is based on the gradual adjustment of the weight based on a feedback signal, i.e., the loss described above.

- The training is based on the following training loop:
  - Draw a batch of training examples $\mathbf{x}$ and corresponding targets $\mathbf{y}_{target}$.
  - Run the network on $\mathbf{x}$ (forward pass) to obtain predictions $\mathbf{y}_{pred}$.
  - Compute the loss of the network on the batch, a measure of the mismatch between $\mathbf{y}_{pred}$ and $\mathbf{y}_{target}$.
  - Update all weights of the networks in a way that reduces the loss of this batch.

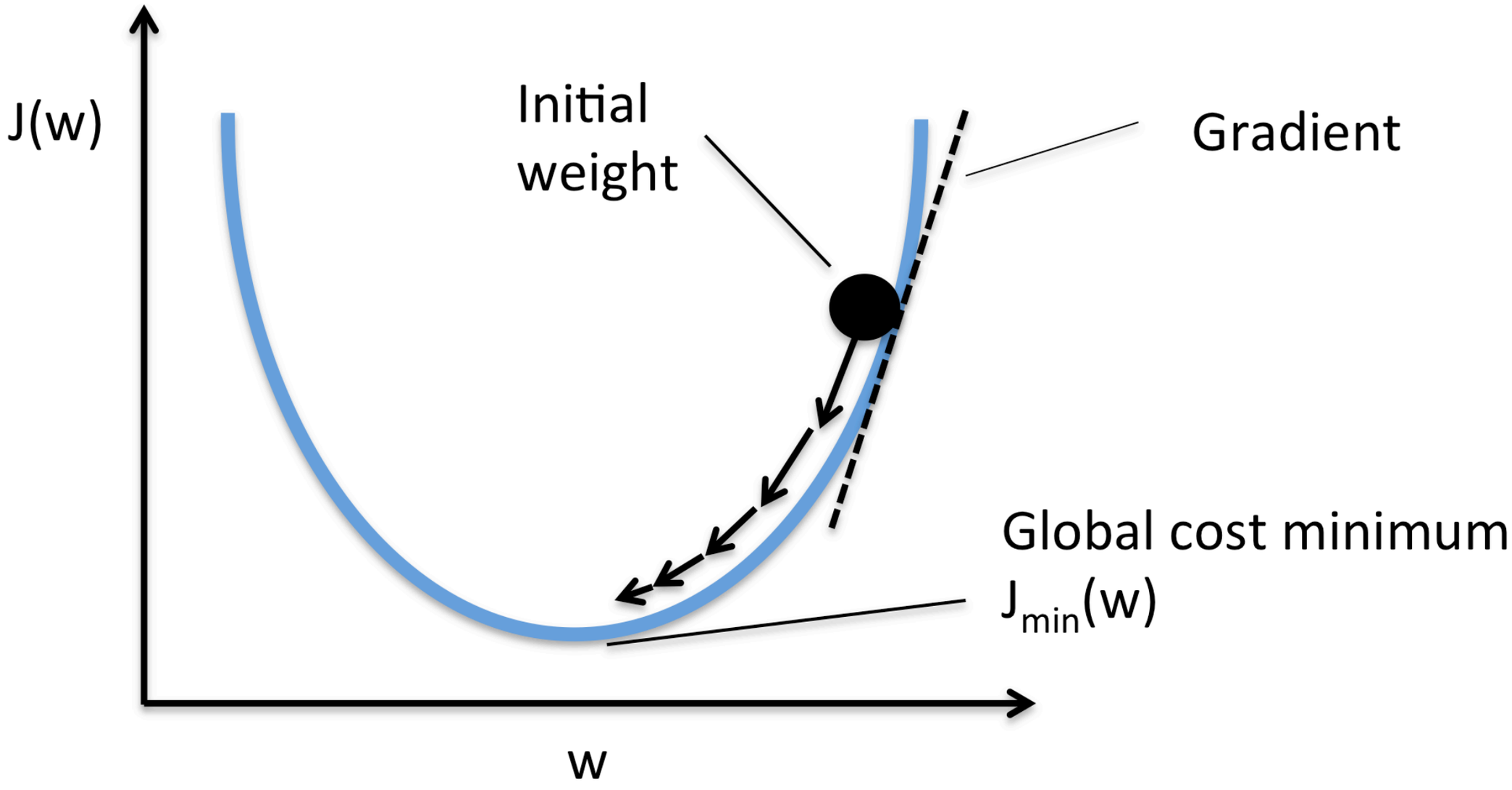
Stochastic Gradient Descent

- Given a differentiable function, it's theoretically possible to find its minimum analytically.

- However, the function is intractable for real networks. The only way is to try to approximate the weights using the procedure described above.

- More precisely, since it is a *differentiable* function, we can use the gradient, which provides an efficient way to perform the correction mention before.

Gradient-based Optimisation



Credit: Sebastian Raschka

Stochastic Gradient Descent

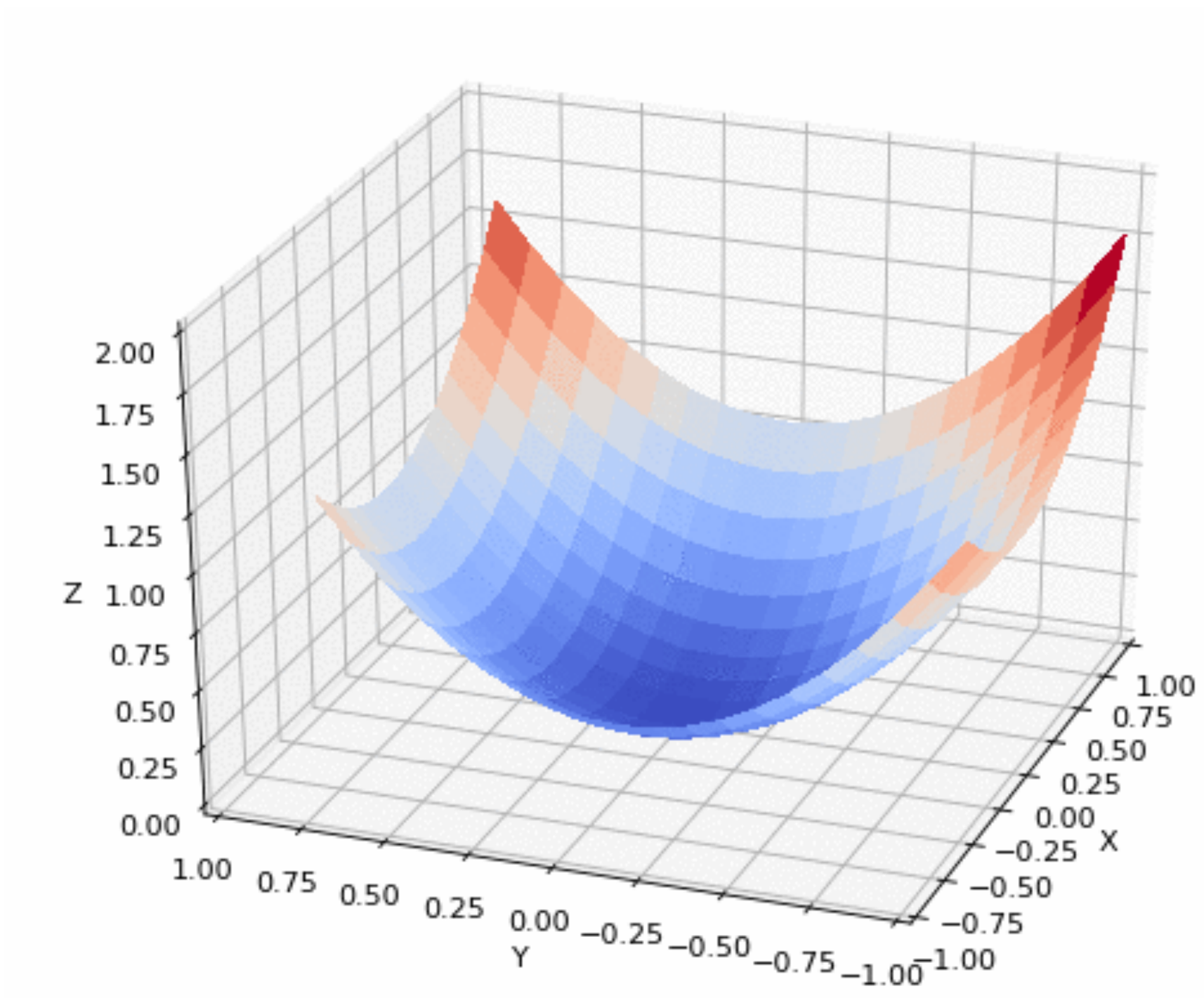
- More formally:
  - Draw a batch of training example $\mathbf{x}$ and corresponding targets $\mathbf{y}_{target}$.
  - Run the network on $\mathbf{x}$ (forward pass) to obtain predictions $\mathbf{y}_{pred}$.
  - Compute the loss of the network on the batch, a measure of the mismatch between $\mathbf{y}_{pred}$ and $\mathbf{y}_{target}$.
  - Compute the gradient of the loss with regard to the network's parameters (backward pass).
  - Move the parameters in the opposite direction from the gradient with: $w_j \leftarrow w_j + \Delta w_j = w_j - \eta \frac{\partial J}{\partial w_j}$

    where J is the loss (cost) function.
  - If you have a batch of samples of dimension k:

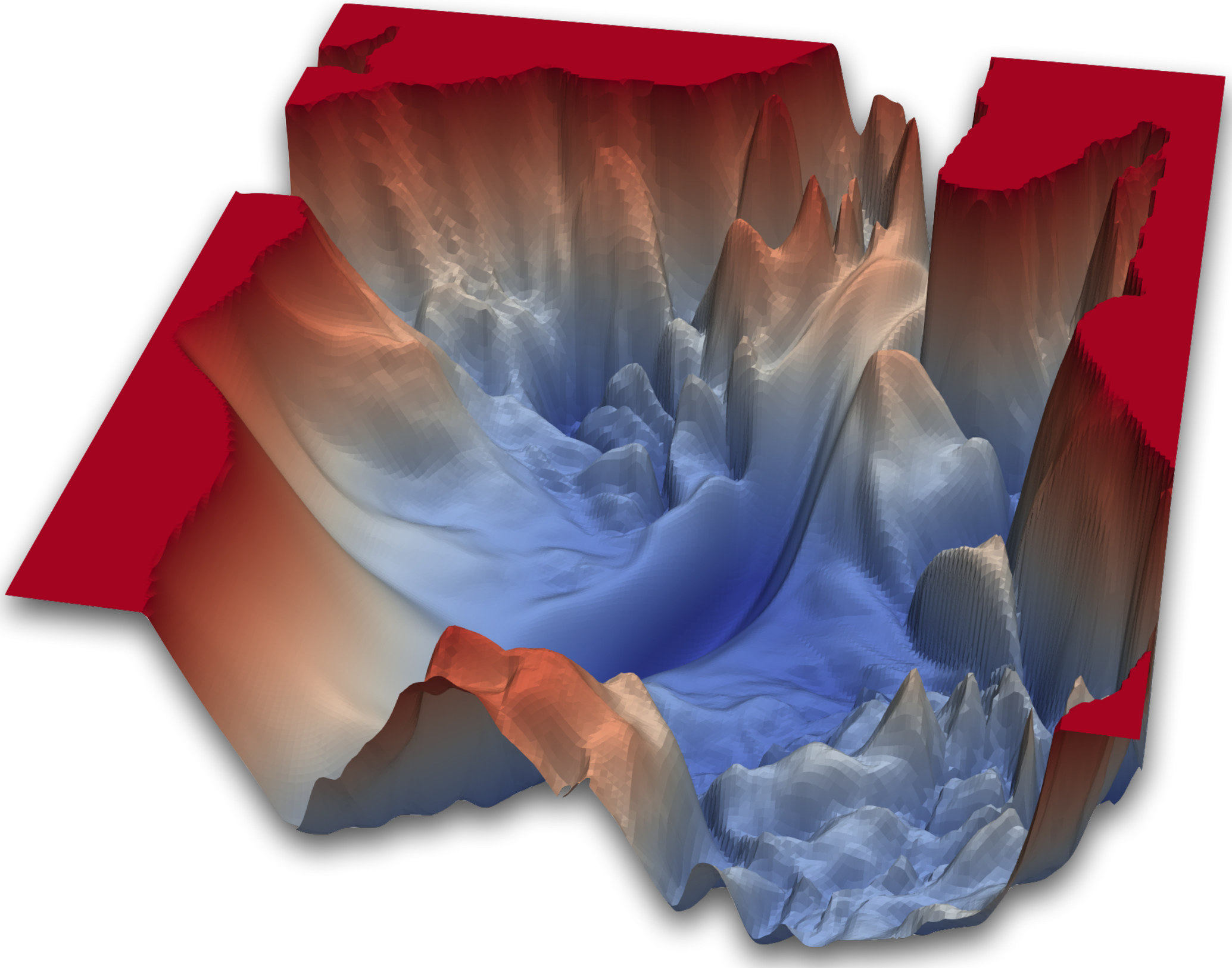
$w_j \leftarrow w_j + \Delta w_j = w_j - \eta \; average(\frac{\partial J_k}{\partial w_j})$ for all the k samples of the batch.

Stochastic Gradient Descent

- This is called the mini-batch stochastic gradient descent (mini-batch SGD).
- The loss function J is a function of $f(\mathbf{x})$, which is a function of the weights.
  - Essentially, you calculate the value $f(\mathbf{x})$, which is a function of the weights of the network.
    - Therefore, by definition, the derivative of the loss function that you are going to apply will be a function of the weights.
- The term *stochastic* refers to the fact that each batch of data is drawn randomly.
- The algorithm described above was based on a simplified model with a single function in a sense.
- You can think about a network composed of three layers, e.g., three tensor operations on the network itself.



https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/



https://www.cs.umd.edu/~tomg/projects/landscapes/

Backpropagation Algorithm

- Suppose that you have three tensor operations/layers f, g, h with weights $\mathbf{W}^1$, $\mathbf{W}^2$ and $\mathbf{W}^3$ respectively for the first, second, third layer. You will have the following function:

$$y_{pred} = f(\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3, \mathbf{x}) = f(\mathbf{W}^3, g(\mathbf{W}^2, h(\mathbf{W}^1, \mathbf{x})))$$

with $f()$ the *rightmost* function/layer and so on. In other words, the input layer is connected to $h()$, which is connected to $g()$, which is connected to $f()$, which returns the final result.

- A network is a sort of chain of layers. You can derive the value of the "correction" by applying the chain rule of the derivatives backwards.
  - Remember the chain rule $(f(g(x)))' = f'(g(x))g'(x)$.

Backpropagation Algorithm

- The update of the weights starts from the right-most layer *back* to the left-most layer. For this reason, this is called *backpropagation* algorithm.

- More specifically, backpropagation starts with the calculation of the gradient of final loss value and works backwards from the right-most layers to the left-most layers, applying the chain rule to compute the contribution that each weight had in the loss value.

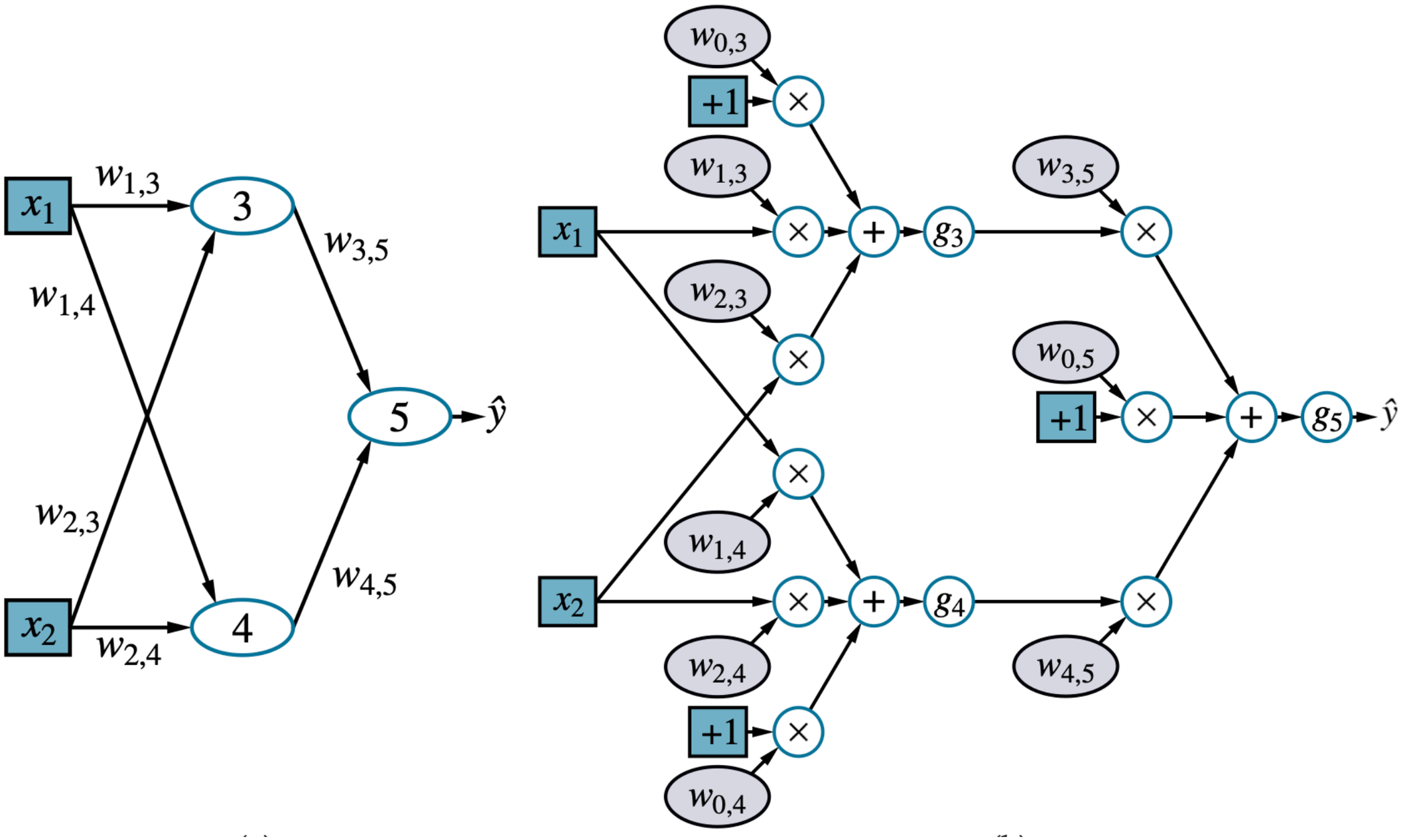
- Nowadays, we do not calculate the partial derivates manually, but we use frameworks like TensorFlow and Pytorch that support symbolic differentiation for the calculation of the gradient.

- TensorFlow and PyTorch support the automatic updates of the weights described above.

- More theoretical details can be found in:

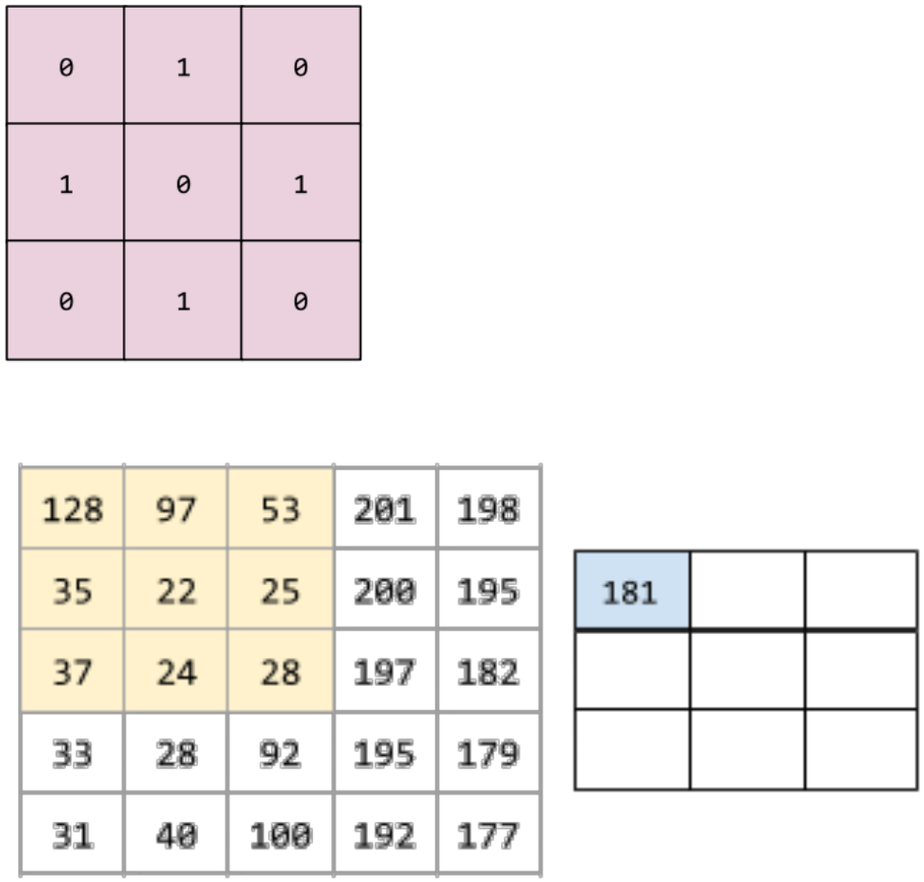
  Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. MIT Press. 2016.

Networks and Computational Graphs



From: Stuart Russell and Peter Norvig. Introduction to Artificial Intelligence. 4th Edition. 2020.

Convolutional Networks



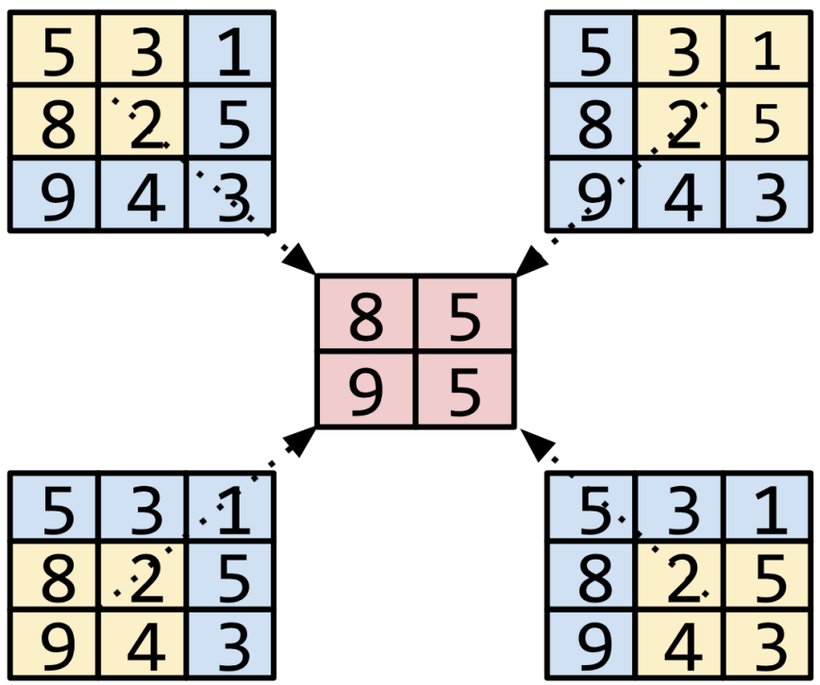
- *Convolutional networks* are networks that contain a mix of convolutional layers, pooling layers and dense layers.

- A *convolutional layer* is a layer of a deep neural network, which contains a convolutional filter.

- A convolutional filter is a matrix having the same rank as the input matrix but a smaller shape.

Convolutional Networks

5	3	1
8	2	5
9	4	3



- A *pooling layer* reduces a matrix (or matrices created by an earlier convolutional layer to a smaller matrix). Pooling usually involves taking either maximum of average value across the pooled area.

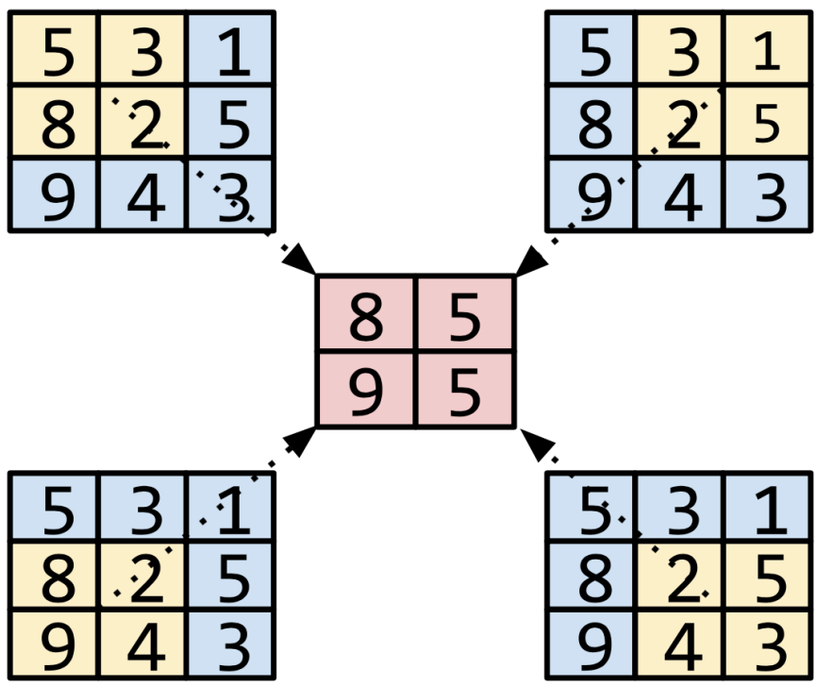
- A *pooling operation* divides the matrix into slices and then slides that convolutional operation by strides.

- A *stride* is the delta in each dimension of the convolutional operation.

Convolutional Networks

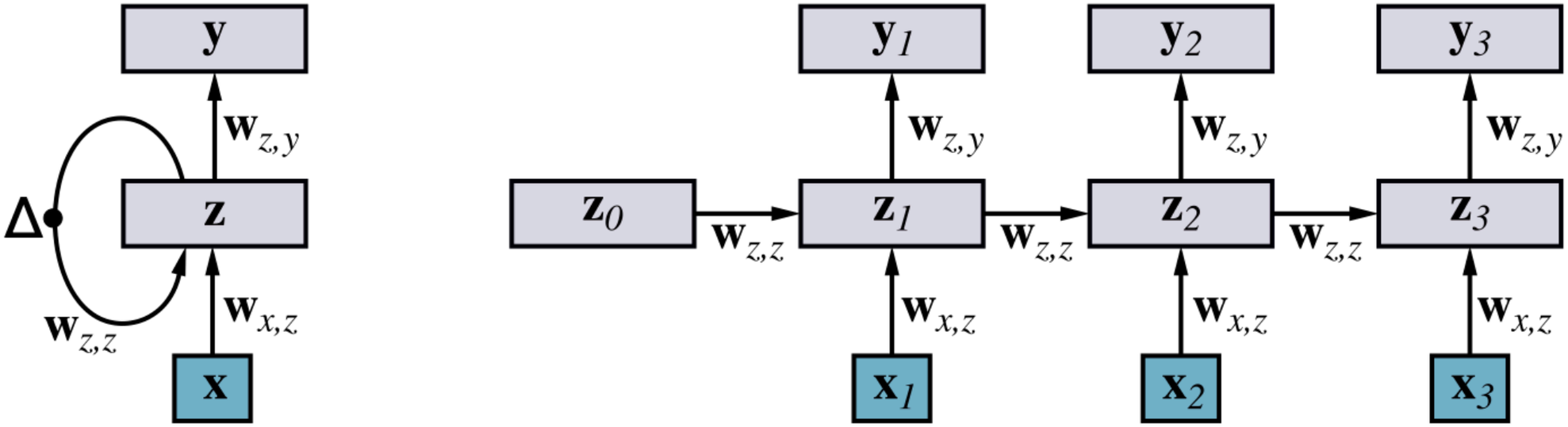
5	3	1
8	2	5
9	4	3

- Pooling helps enforce translational invariance, which allows algorithms to classify images when the position of the objects within the images change, in the input matrix.



- Pooling for vision applications is usually called *spatial pooling*.
- Pooling for time-series applications is usually referred to as *temporal pooling*.
- You can also hear the expressions *subsampling* and *downsampling*.

Recurrent Neural Networks



LSTMs

- It is a kind of RNN that does not suffer from the problem of vanishing gradients.

- In fact, an LSTM can choose to remember part of the input, copying it over to the next time step and to forget other parts.

- LSTM stands from long short-term memory. LSTM is a kind of RNN with gating units that does not suffer vanishing gradients.

LONG SHORT-TERM MEMORY

Neural Computation 9(8):1735–1780, 1997

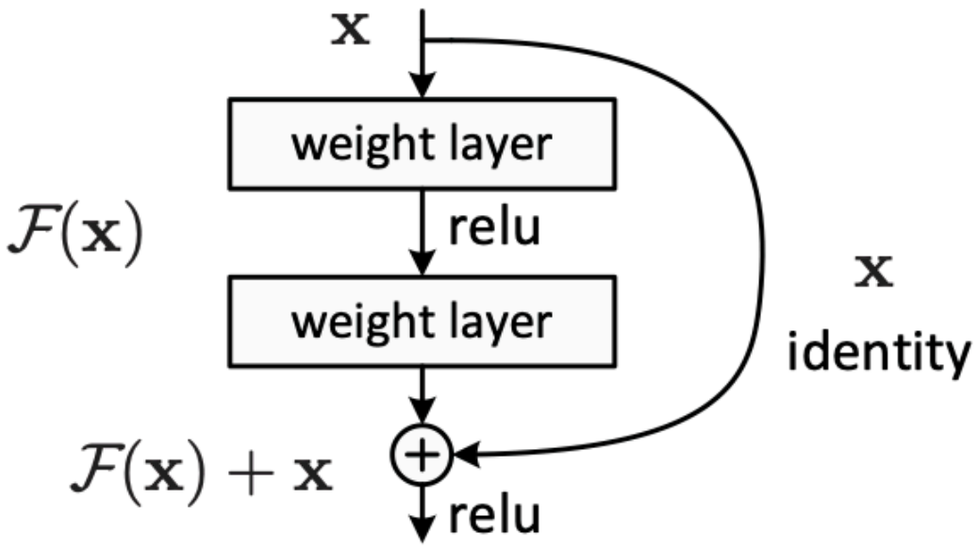
Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

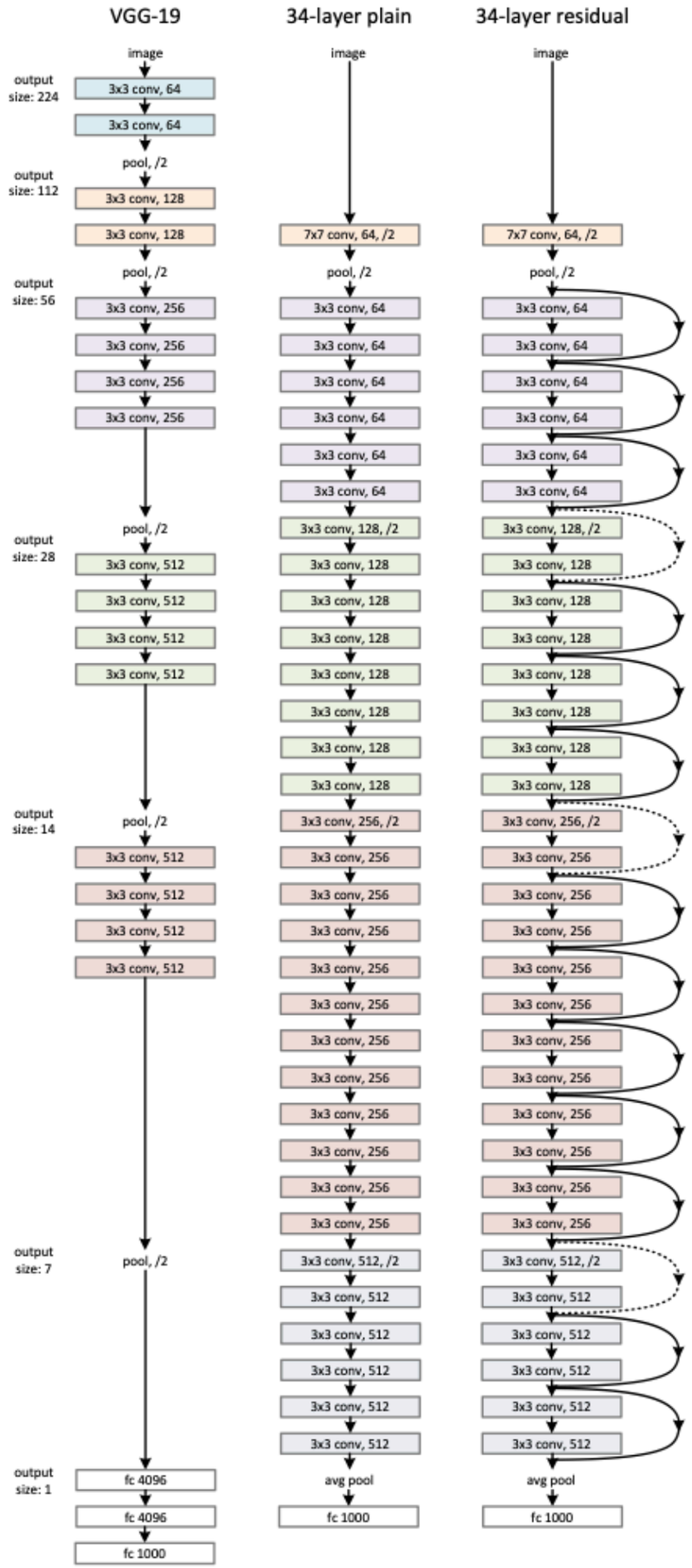
Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never

Residual Networks





Deep Learning Applications

- The number of application of deep learning is increasing everyday:
  - Image and video processing and vision;
  - Machine translation;
  - Speech generation;
  - Applications to many scientific fields (astronomy, biology, etc.).
    - See for example the problem of protein folding.
- One of the biggest achievement is the extension of the domain of reinforcement learning.
  - We refer to the convergence of deep learning and reinforcement learning as *deep reinforcement learning*.
  - Applications of deep reinforcement learning include games, robotics, etc.

Neuroscience and Deep Learning: Some Caveats

- Neuroscience can be an inspiration, but we should remember that we are trying to “engineer” a system.
- Actual neurons are not based on the simple functions that we use in our systems.
  - At the moment, more complex functions haven’t led to improve performance yet.
- Neuroscience has inspired the design of several neural architectures, but our knowledge is limited in terms of how the brain actually learn.
- For this reason, neuroscience is of limited help for improving the design of the learning algorithms themselves.
- Deep learning is not an attempt to simulate the brain!

References

- Chapters 1, 16 and 20 of Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. MIT Press. 2016.

Attribution Notice

- Portion of the material in the slides about convolutional networks, recurrent networks are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 4.0 Attribution License.

- Source: https://developers.google.com/machine-learning/glossary/

- Attribution license: https://creativecommons.org/licenses/by/4.0/