Autonomous and Adaptive Systems

Introduction to Deep Learning II

Mirco Musolesi

mircomusolesi@acm.org

Neurocognitron and Convolutional 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

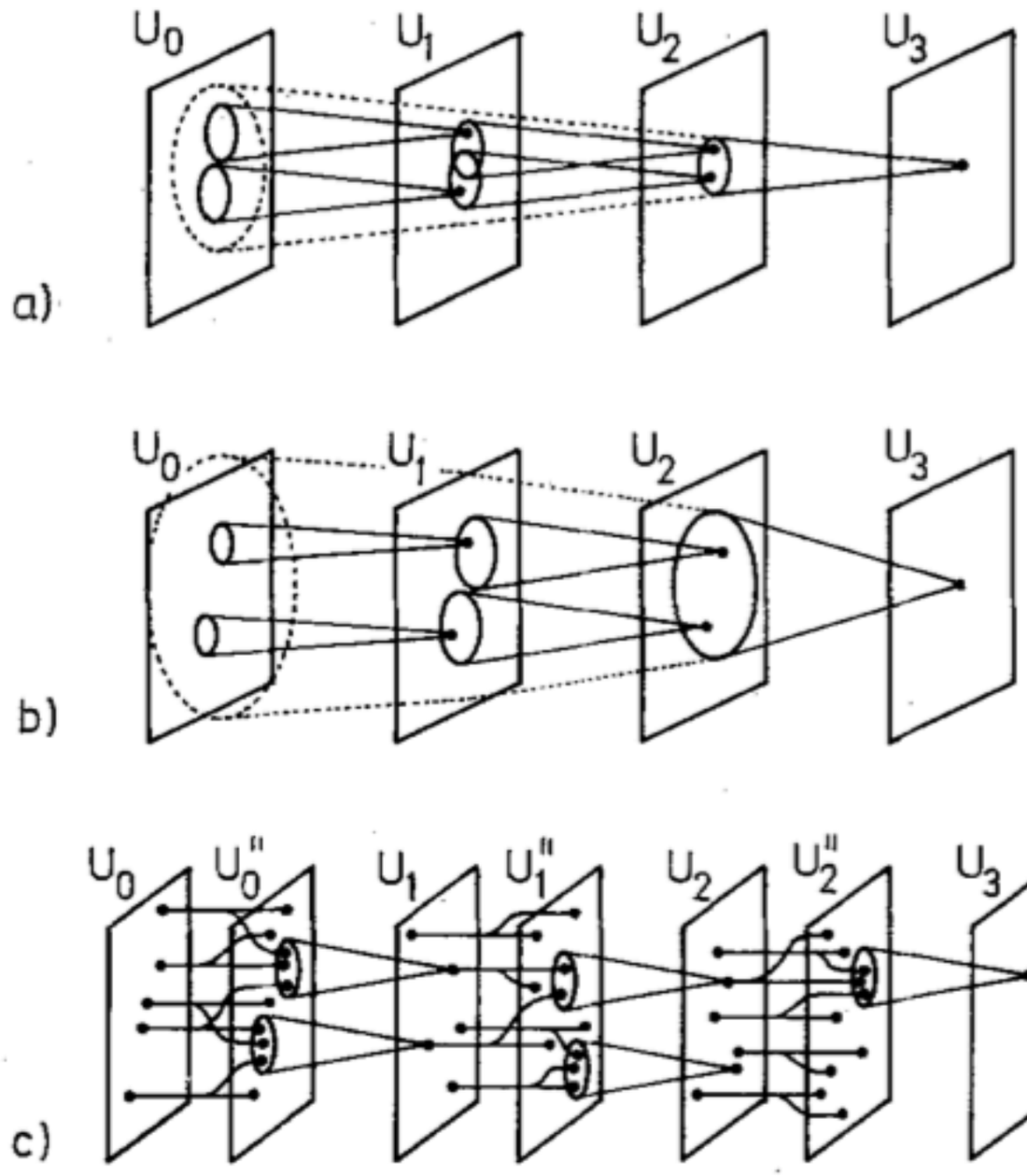


Fig. 4a–c. Three possible methods for interconnecting layers. The connectable area of each cell is differently chosen in these three methods. Method c is adopted for the cognitron discussed in this paper

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

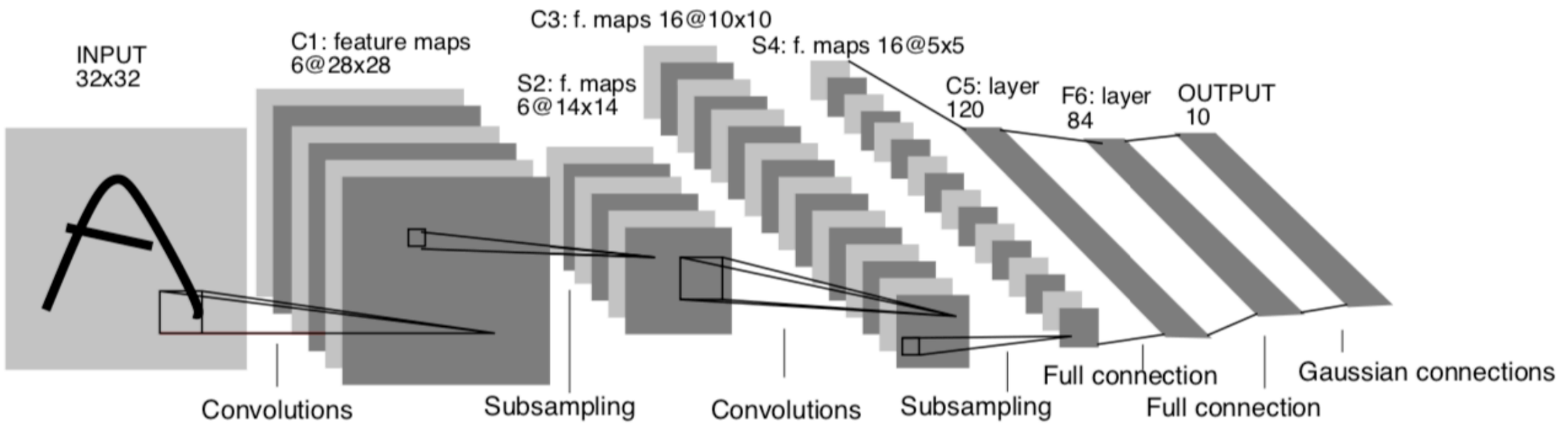
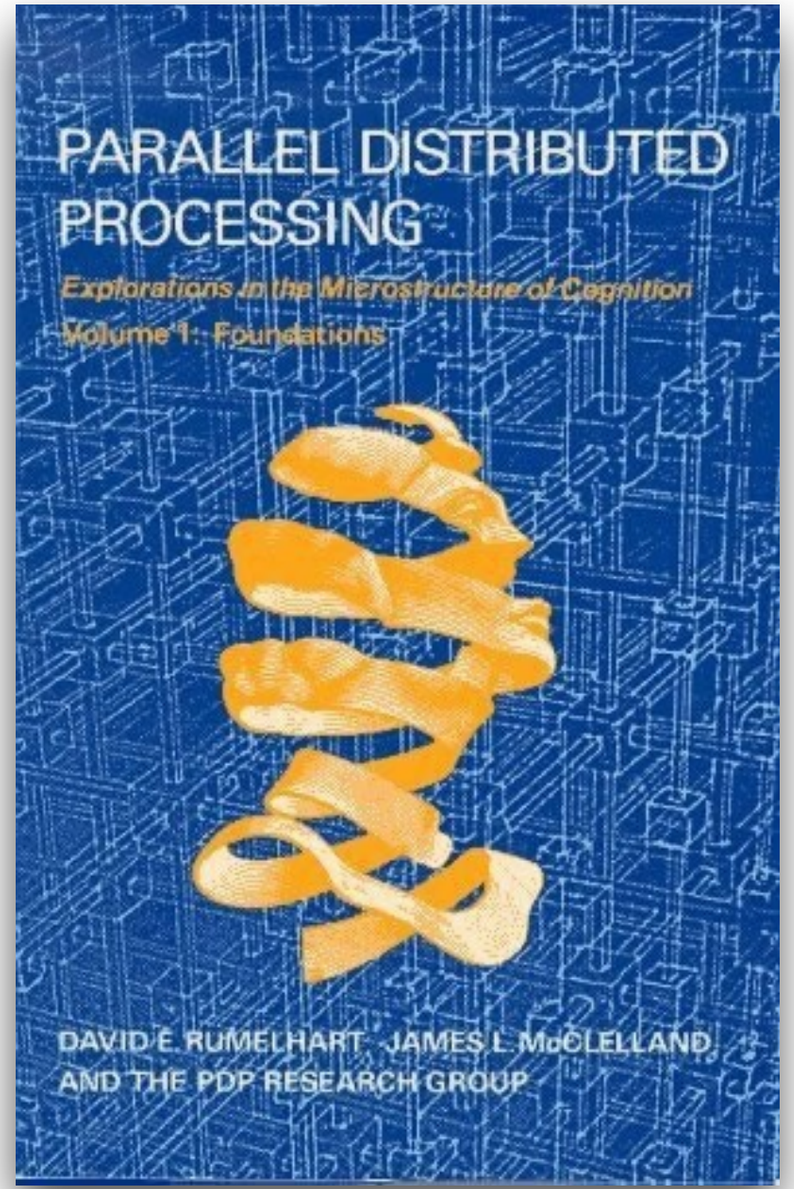
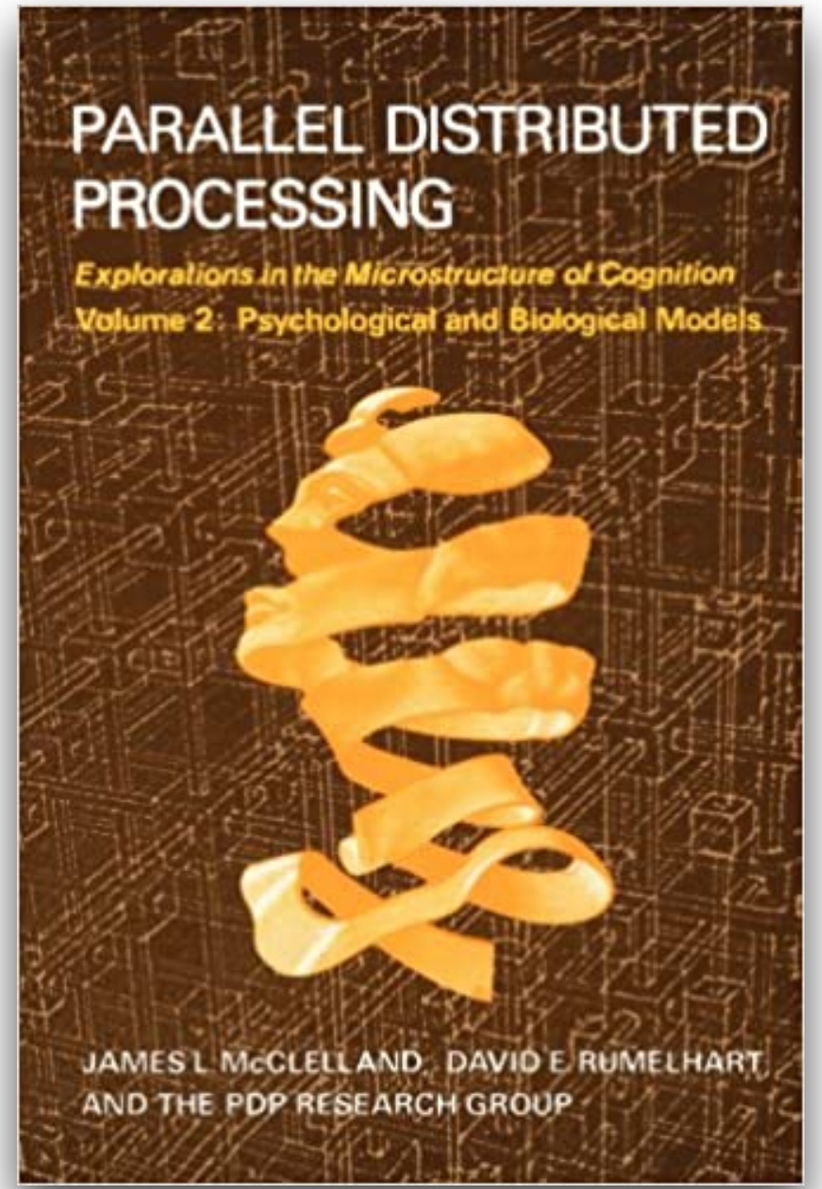


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

Learning and Relearning in Boltzmann Machines

G. E. HINTON and T. J. SEJNOWSKI

Many of the chapters in this volume make use of the ability of a parallel network to perform cooperative searches for good solutions to problems. The basic idea is simple: The weights on the connections between processing units encode knowledge about how things normally fit together in some domain and the initial states or external inputs to a subset of the units encode some fragments of a structure within the domain. These fragments constitute a problem: What is the whole structure from which they probably came? The network computes a "good solution" to the problem by repeatedly updating the states of units that represent possible other parts of the structure until the network eventually settles into a stable state of activity that represents the solution.

One field in which this style of computation seems particularly appropriate is vision (Ballard, Hinton, & Sejnowski, 1983). A visual system must be able to solve large constraint-satisfaction problems rapidly in order to interpret a two-dimensional intensity image in terms of the depths and orientations of the three-dimensional surfaces in the world that gave rise to that image. In general, the information in the

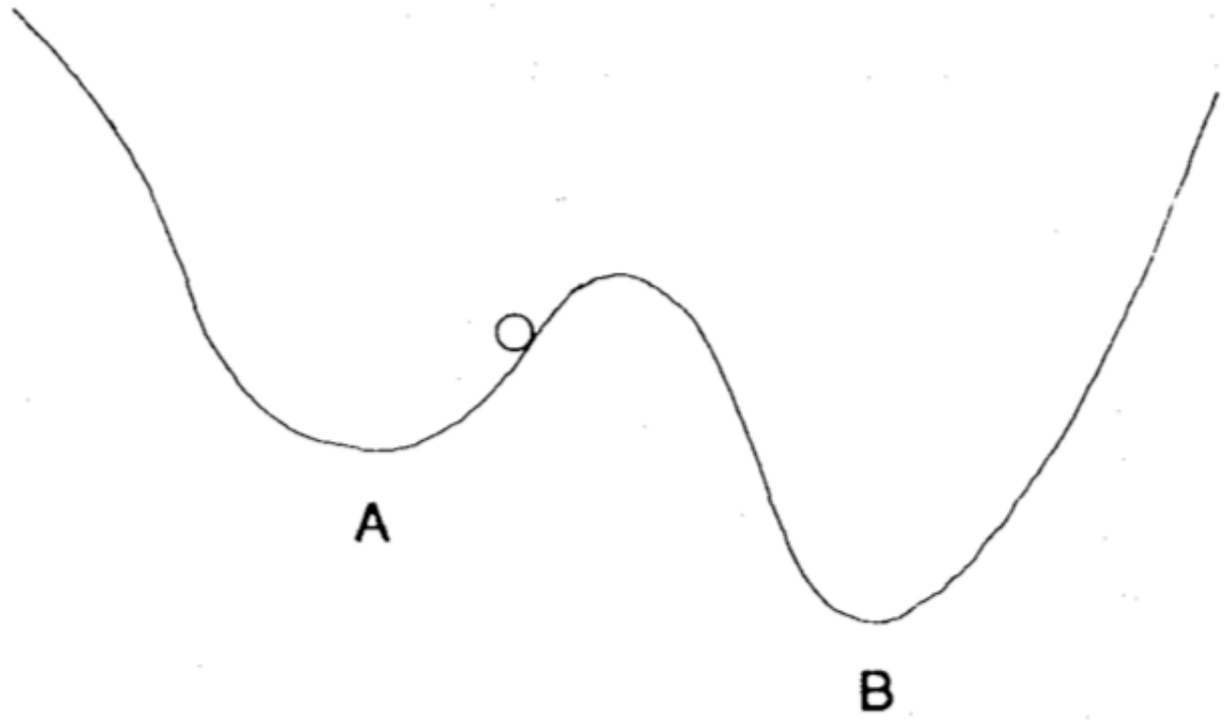


FIGURE 1. A simple energy landscape containing two local minima separated by an energy barrier. Shaking can be used to allow the state of the network (represented here by a ball-bearing) to escape from local minima.

delineating the absolute indigeneity of amino acids in fossils. As AMS techniques are refined to handle smaller samples, it may also become possible to date individual amino acid enantiomers by the ^{14}C method. If one enantiomer is entirely derived from the other by racemization during diagenesis, the individual D- and L-enantiomers for a given amino acid should have identical ^{14}C ages.

Older, more poorly preserved fossils may not always prove amenable to the determination of amino acid indigeneity by the stable isotope method, as the prospects for complete replacement of indigenous amino acids with non-indigenous amino acids increases with time. As non-indigenous amino acids undergo racemization, the enantiomers may have identical isotopic compositions and still not be related to the original organisms. Such a circumstance may, however, become easier to recognize as more information becomes available concerning the distribution and stable isotopic composition of the amino acid constituents of modern representatives of fossil organisms. Also, AMS dates on individual amino acid enantiomers may, in some cases, help to clarify indigeneity problems, in particular when stratigraphic controls can be used to estimate a general age range for the fossil in question.

Finally, the development of techniques for determining the stable isotopic composition of amino acid enantiomers may enable us to establish whether non-racemic amino acids in some carbonaceous meteorites[27] are indigenous, or result in part from terrestrial contamination.

M.H.E. thanks the NSF, Division of Earth Sciences (grant EAR-8352055) and the following contributors to his Presidential Young Investigator Award for partial support of this research: Arco, Exxon, Phillips Petroleum, Texaco Inc., The Upjohn Co. We also acknowledge the donors of the Petroleum Research Fund, administered by the American Chemical Society (grant 16144-AC2 to M.H.E., grant 14805-AC2 to S.A.M.) for support. S.A.M. acknowledges NSERC (grant A2644) for partial support.

Received 19 May; accepted 15 July 1986.

1. Bada, J. L. & Protsch, R. *Proc. natn. Acad. Sci. U.S.A.* **70,** 1331–1334 (1973).
2. Bada, J. L., Schroeder, R. A. & Carter, G. F. *Science* **184,** 791–793 (1974).
3. Boulton, G. S. *et al. Nature* **298,** 437–441 (1982).
4. Wehmiller, J. F. in *Quaternary Dating Methods* (ed. Mahaney, W. C.) 171–193 (Elsevier, Amsterdam, 1984).
5. Engel, M. H., Zumberge, J. E. & Nagy, B. *Analyt. Biochem.* **82,** 415–422 (1977).
6. Bada, J. L. *A. Rev. Earth planet. Sci.* **13,** 241–268 (1985).
7. Chisholm, B. S., Nelson, D. E. & Schwarcz, H. P. *Science* **216,** 1131–1132 (1982).
8. Ambrose, S. H. & DeNiro, M. J. *Nature* **319,** 321–324 (1986).
9. Macko, S. A., Estep, M. L. F., Hare, P. E. & Hoering, T. C. *Yb. Carnegie Instn Wash.* **82,** 404–410 (1983).
10. Hare, P. E. & Estep, M. L. F. *Yb. Carnegie Instn Wash.* **82,** 410–414 (1983).
11. Engel, M. H. & Hare, P. E. in *Chemistry and Biochemistry of the Amino Acids* (ed. Barrett, G. C.) 462–479 (Chapman and Hall, London, 1985).
12. Johnstone, R. A. W. & Rose, M. E. in *Chemistry and Biochemistry of the Amino Acids* (ed. Barrett, G. C.) 480–524 (Chapman and Hall, London, 1985).
13. Weinstein, S., Engel, M. H. & Hare, P. E. in *Practical Protein Chemistry—A Handbook* (ed. Darbre, A.) 337–344 (Wiley, New York, 1986).
14. Bada, J. L., Gillespie, R., Gowlett, J. A. J. & Hedges, R. E. M. *Nature* **312,** 442–444 (1984).
15. Mitterer, R. M. & Kriausakul, N. *Org. Geochem.* **7,** 91–98 (1984).
16. Williams, K. M. & Smith, G. G. *Origins Life* **8,** 91–144 (1977).
17. Engel, M. H. & Hare, P. E. *Yb. Carnegie Instn Wash.* **81,** 425–430 (1982).
18. Hare, P. E. *Yb. Carnegie Instn Wash.* **73,** 576–581 (1974).
19. Pillinger, C. T. *Nature* **296,** 802 (1982).
20. Neuberger, A. *Adv. Protein Chem.* **4,** 298–383 (1948).
21. Engel, M. H. & Macko, S. A. *Analyt. Chem.* **56,** 2598–2600 (1984).
22. Dungworth, G. *Chem. Geol.* **17,** 135–153 (1976).
23. Weinstein, S., Engel, M. H. & Hare, P. E. *Analyt. Biochem.* **121,** 370–377 (1982).
24. Macko, S. A., Lee, W. Y. & Parker, P. L. *J. exp. mar. Biol. Ecol.* **63,** 145–149 (1982).
25. Macko, S. A., Estep, M. L. F. & Hoering, T. C. *Yb. Carnegie Instn Wash.* **81,** 413–417 (1982).
26. Vallentyne, J. R. *Geochim. cosmochim. Acta* **28,** 157–188 (1964).
27. Engel, M. H. & Nagy, B. *Nature* **296,** 837–840 (1982).

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \quad (2)$$

† To whom correspondence should be addressed.

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 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!

Deep Learning and Computational Neuroscience

- At the same time, it is worth noting that there is an entire field of neuroscience devoted to understanding the brain using mathematical and computational models. The area is called *computational neuroscience*.

- AI and neuroscience are strictly linked and indeed understanding brain biology will lead to improvement in the design of AI systems.

- This is currently an area of intense research.

Neuron
Review

Neuroscience-Inspired Artificial Intelligence

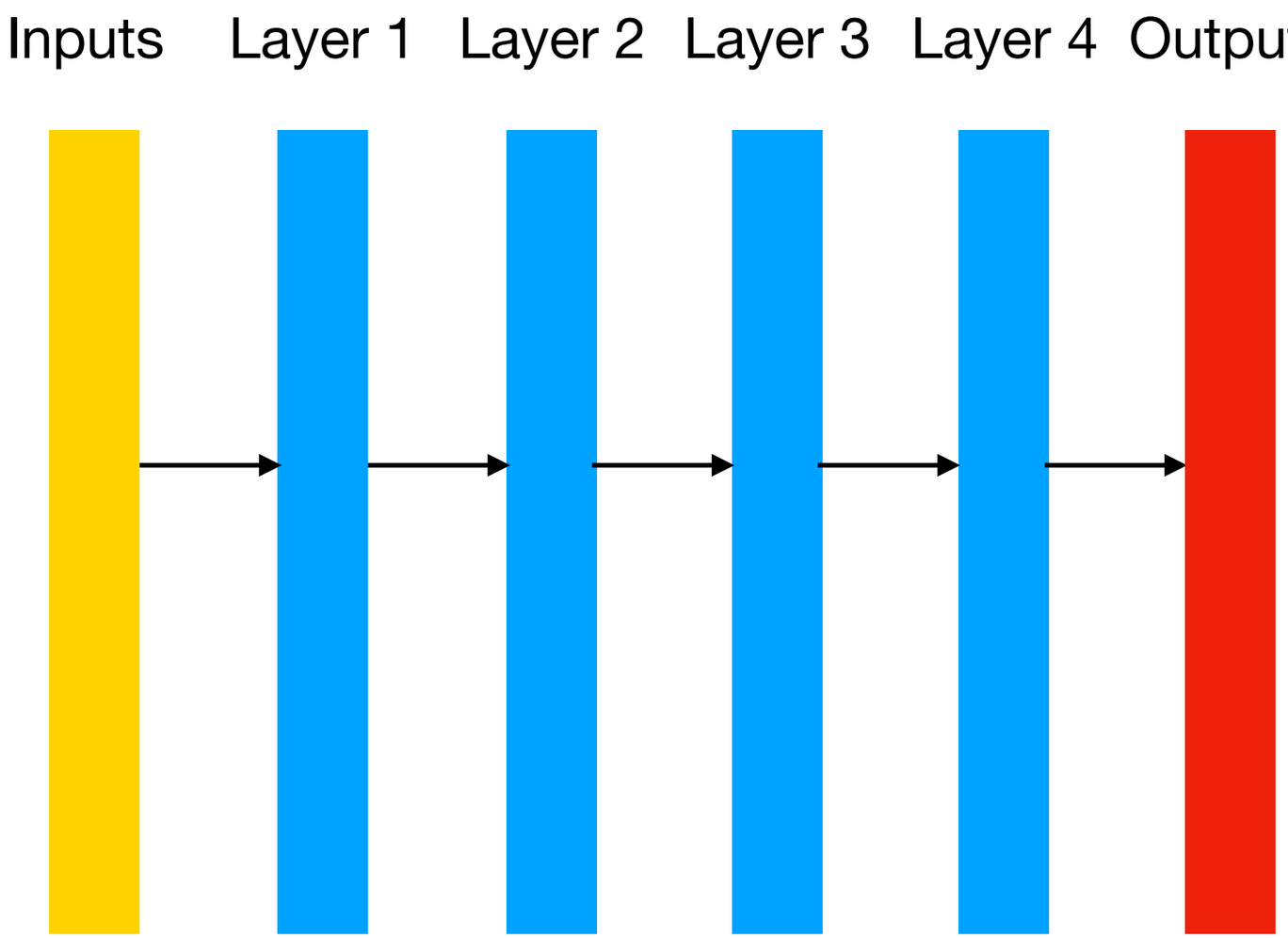
Demis Hassabis,[1,2,*] Dharshan Kumaran,[1,3] Christopher Summerfield,[1,4] and Matthew Botvinick[1,2]
[1]DeepMind, 5 New Street Square, London, UK
[2]Gatsby Computational Neuroscience Unit, 25 Howland Street, London, UK
[3]Institute of Cognitive Neuroscience, University College London, 17 Queen Square, London, UK
[4]Department of Experimental Psychology, University of Oxford, Oxford, UK
*Correspondence: dhcontact@google.com
http://dx.doi.org/10.1016/j.neuron.2017.06.011

The fields of neuroscience and artificial intelligence (AI) have a long and intertwined history. In more recent times, however, communication and collaboration between the two fields has become less commonplace. In this article, we argue that better understanding biological brains could play a vital role in building intelligent machines. We survey historical interactions between the AI and neuroscience fields and emphasize current advances in AI that have been inspired by the study of neural computation in humans and other animals. We conclude by highlighting shared themes that may be key for advancing future research in both fields.

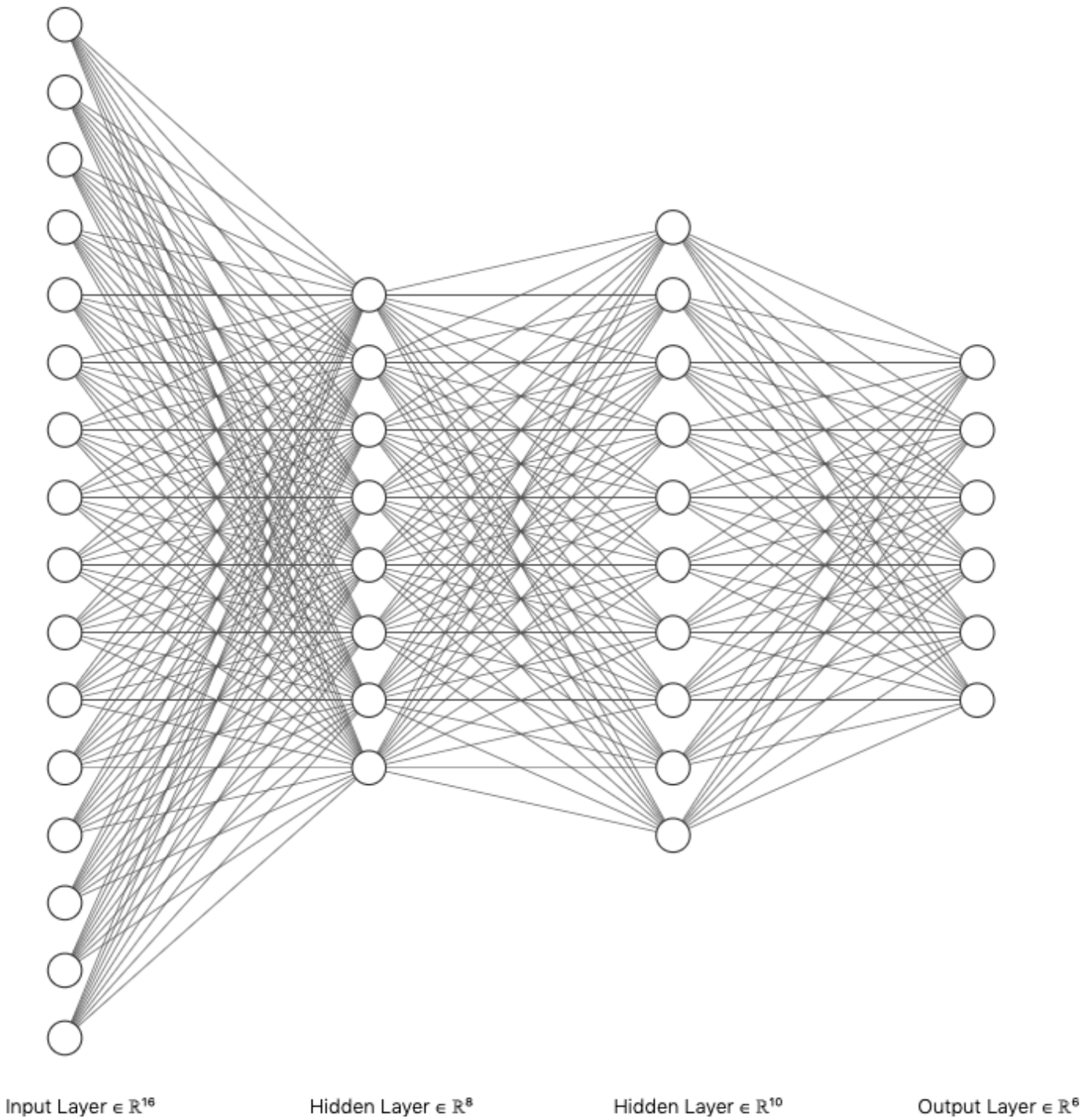
In recent years, rapid progress has been made in the related fields of neuroscience and artificial intelligence (AI). At the dawn of the computer age, work on AI was inextricably intertwined with neuroscience and psychology, and many of the early pioneers straddled both fields, with collaborations between these disciplines proving highly productive (Churchland and Sejnowski, 1988; Hebb, 1949; Hinton et al., 1986; Hopfield, 1982; McCulloch and Pitts, 1943; Turing, 1950). However, more recently, the interaction has become much less commonplace, as both subjects have grown enormously in complexity and disciplinary boundaries have solidified. In this review, we argue for the critical and ongoing importance of neuroscience in generating ideas that will accelerate and guide AI research tively. For example, if an algorithm is not quite attaining the level of performance required or expected, but we observe it is core to the functioning of the brain, then we can surmise that redoubled engineering efforts geared to making it work in artificial systems are likely to pay off.

Of course from a practical standpoint of building an AI system, we need not slavishly enforce adherence to biological plausibility. From an engineering perspective, what works is ultimately all that matters. For our purposes then, biological plausibility is a guide, not a strict requirement. What we are interested in is a systems neuroscience-level understanding of the brain, namely the algorithms, architectures, functions, and representations it utilizes. This roughly corresponds to

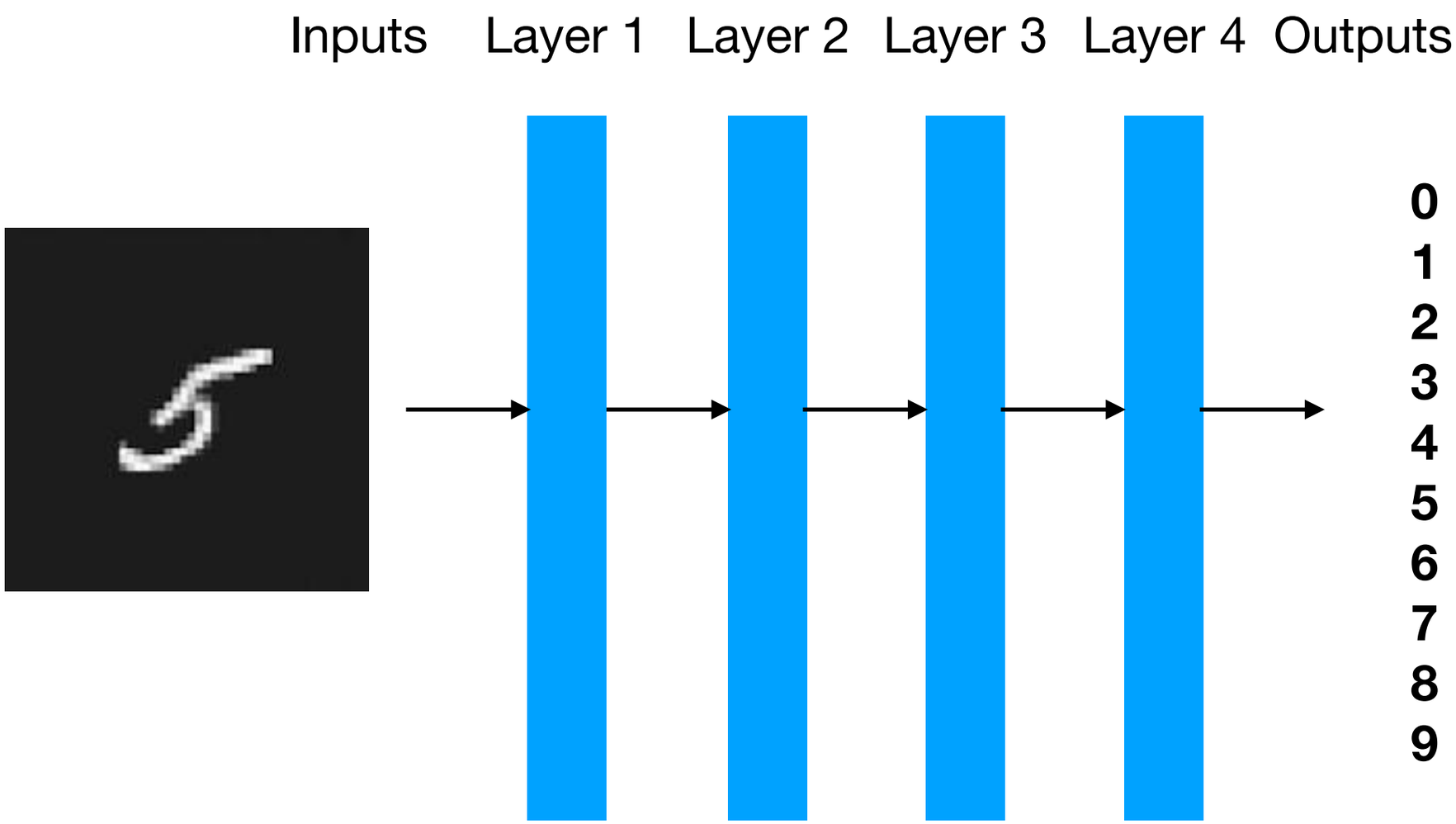
Deep Neural Networks



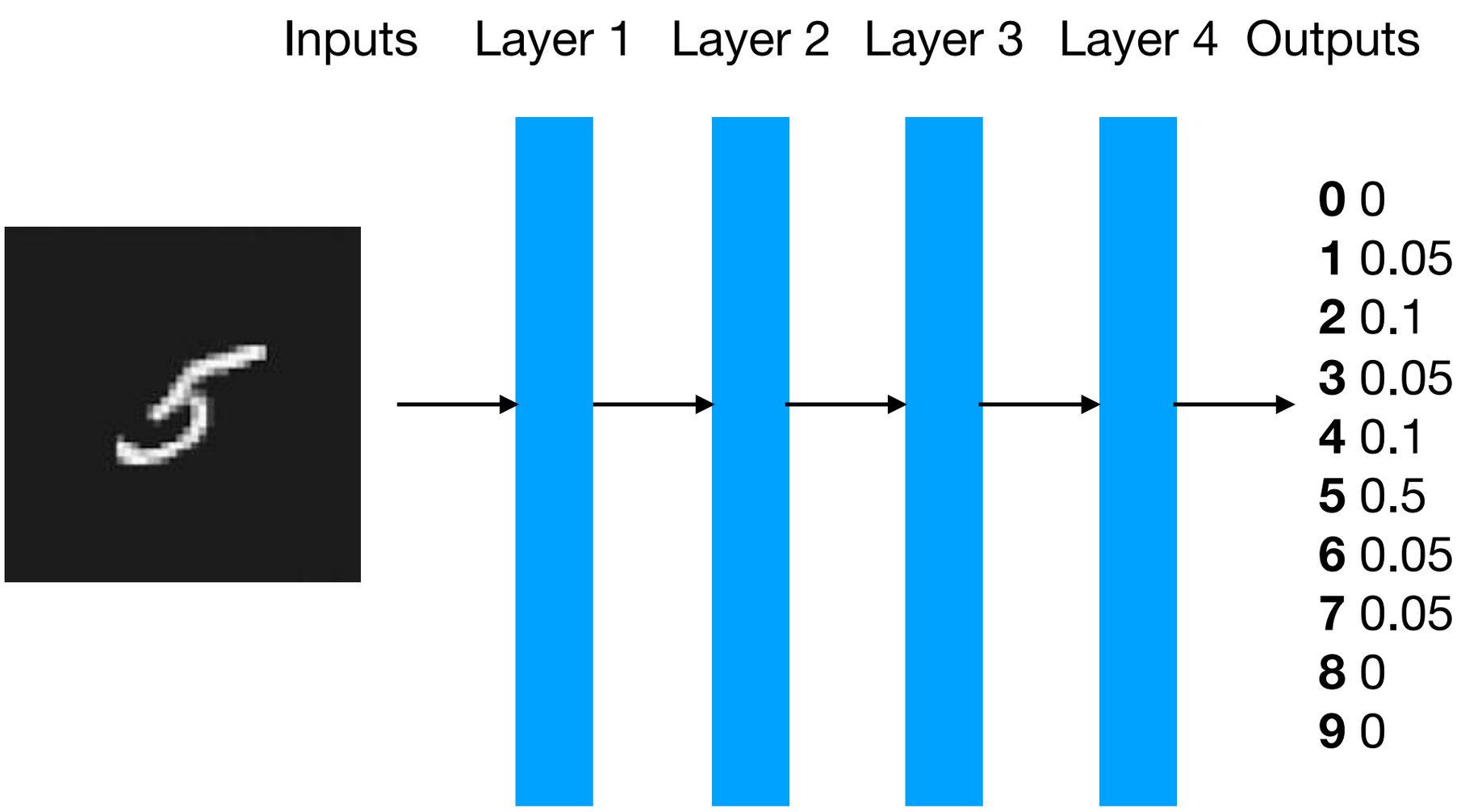
Deep Neural Networks



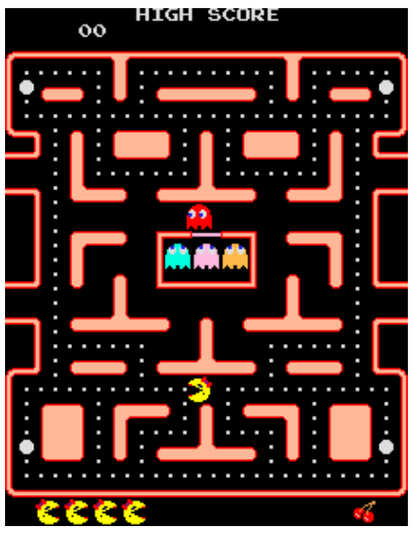
Deep Neural Networks

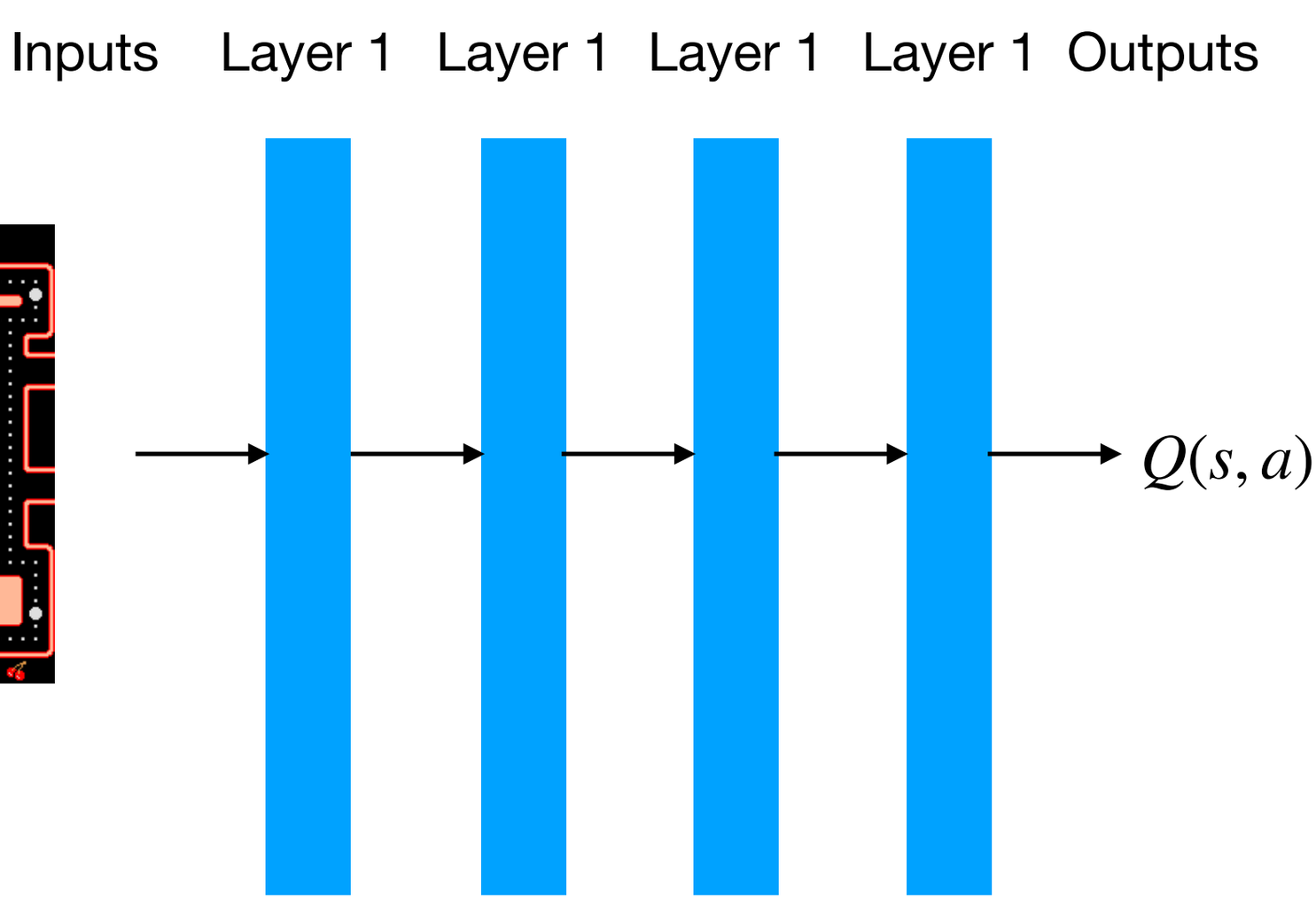


Deep Neural Networks

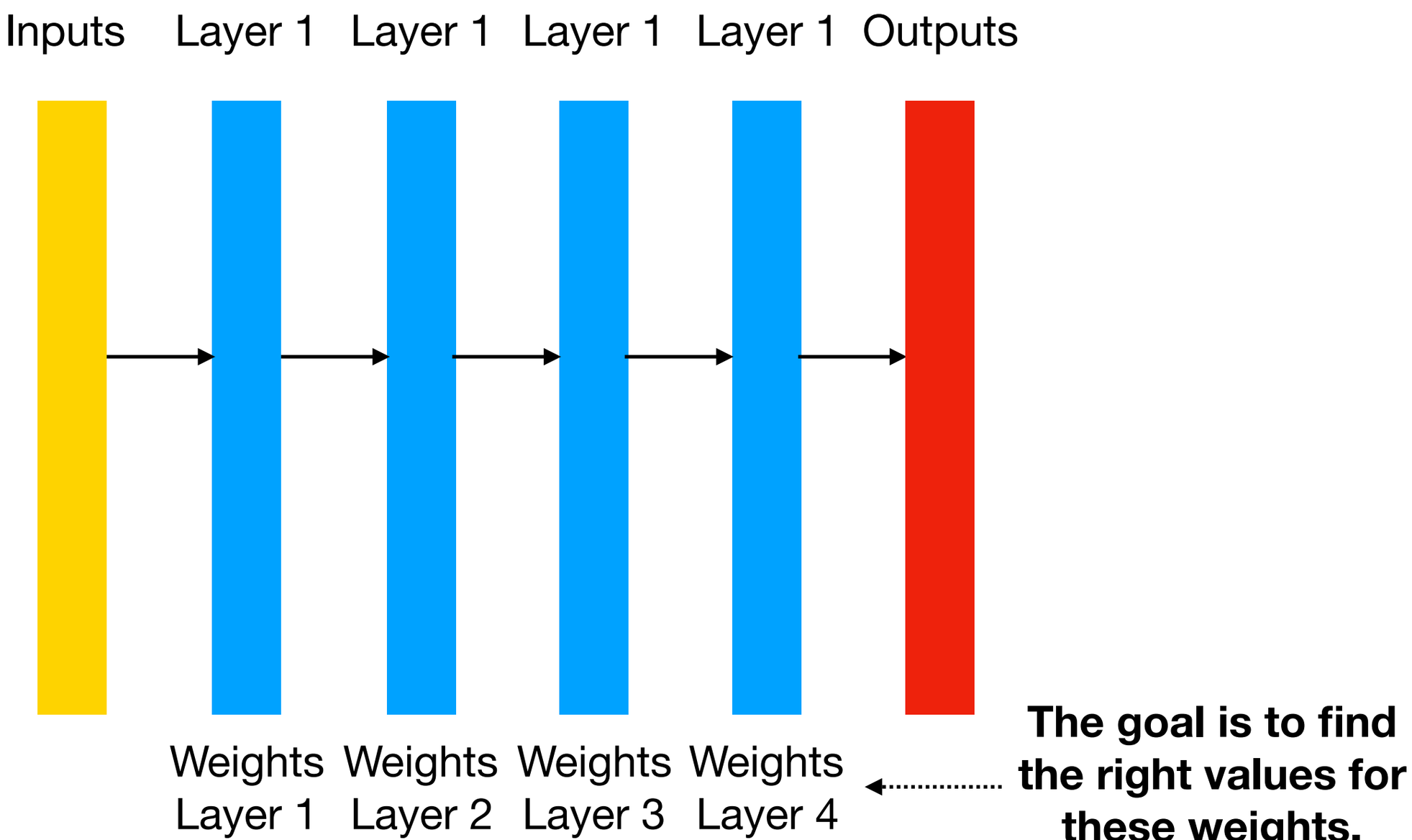


Deep Neural Networks

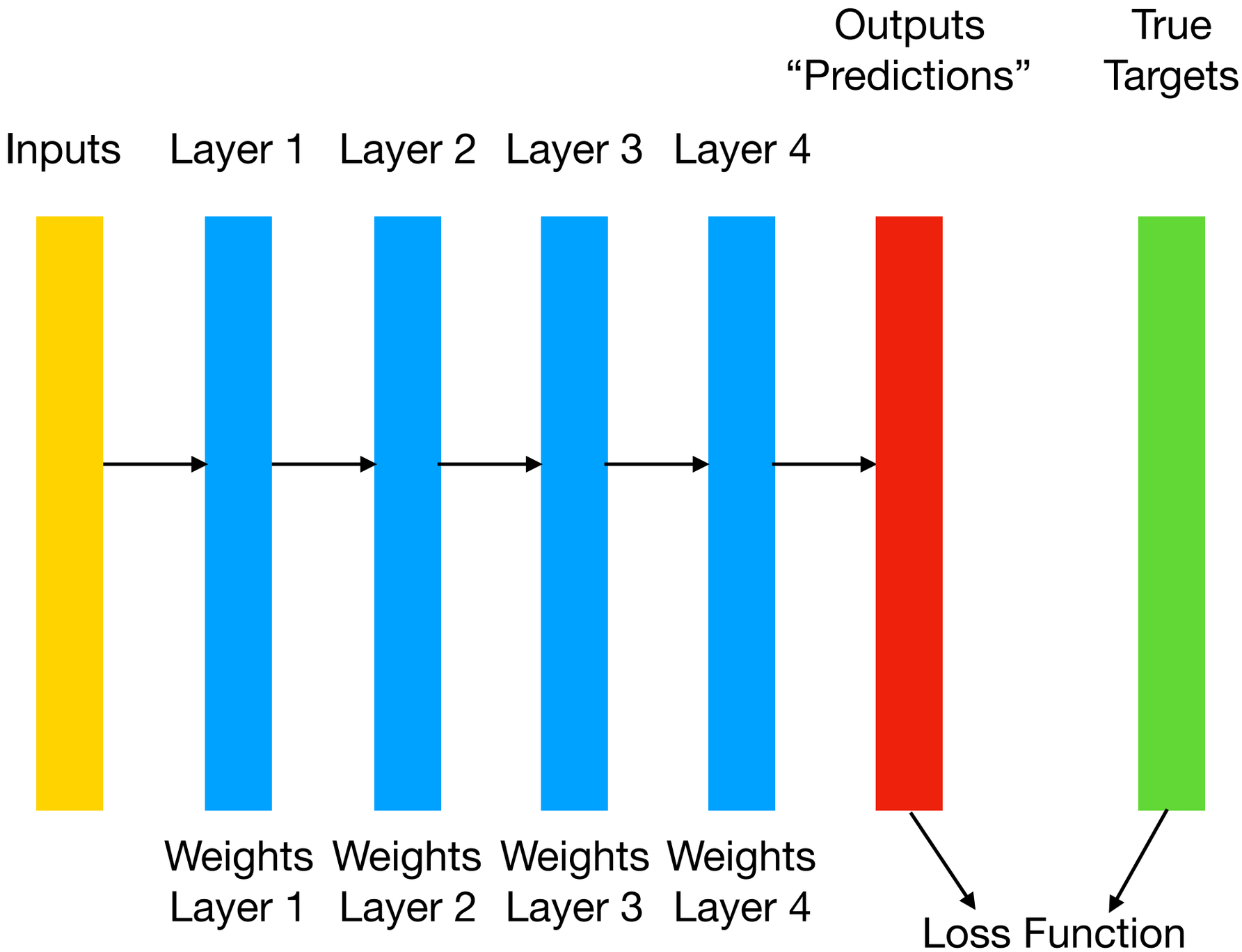




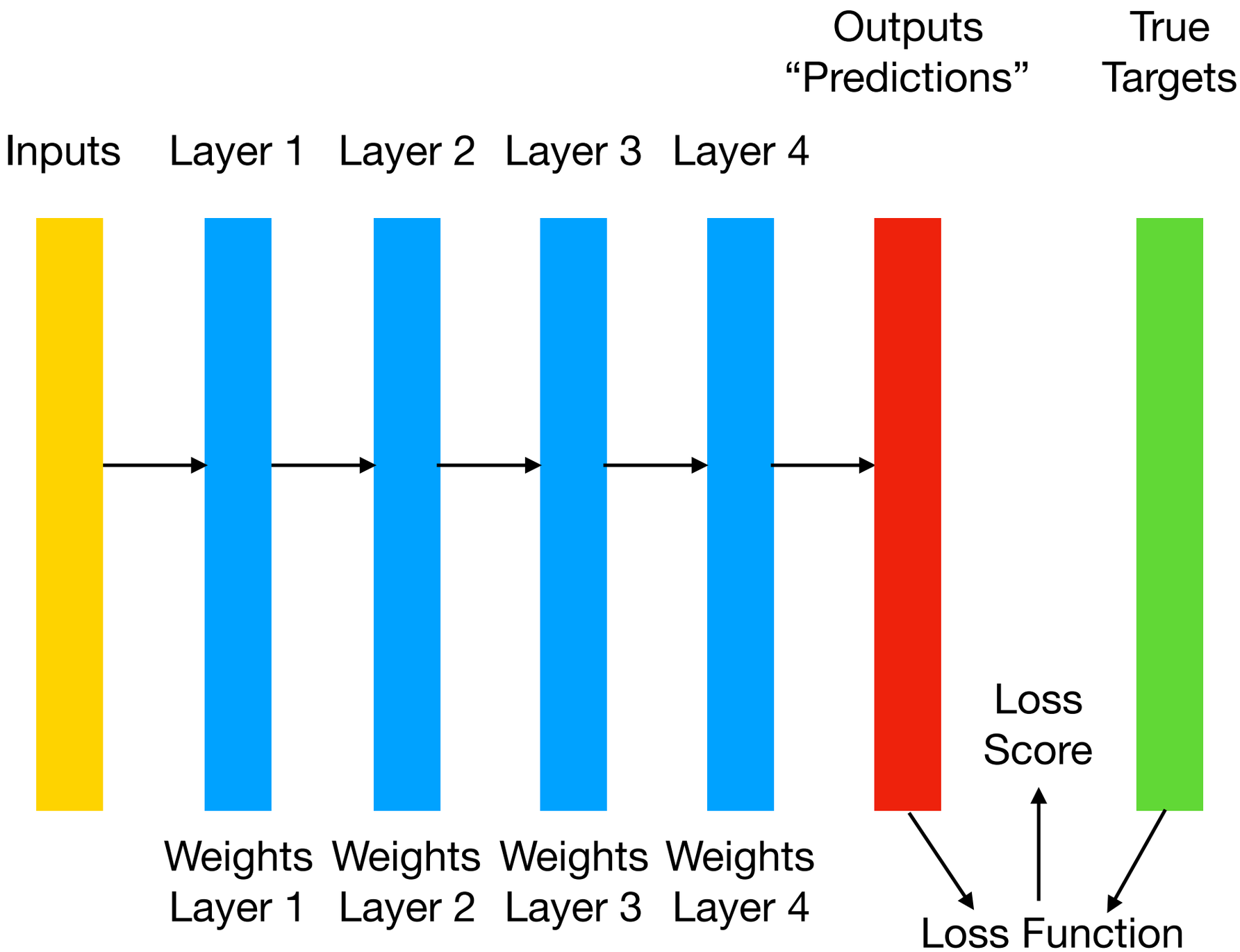
Deep Neural Networks



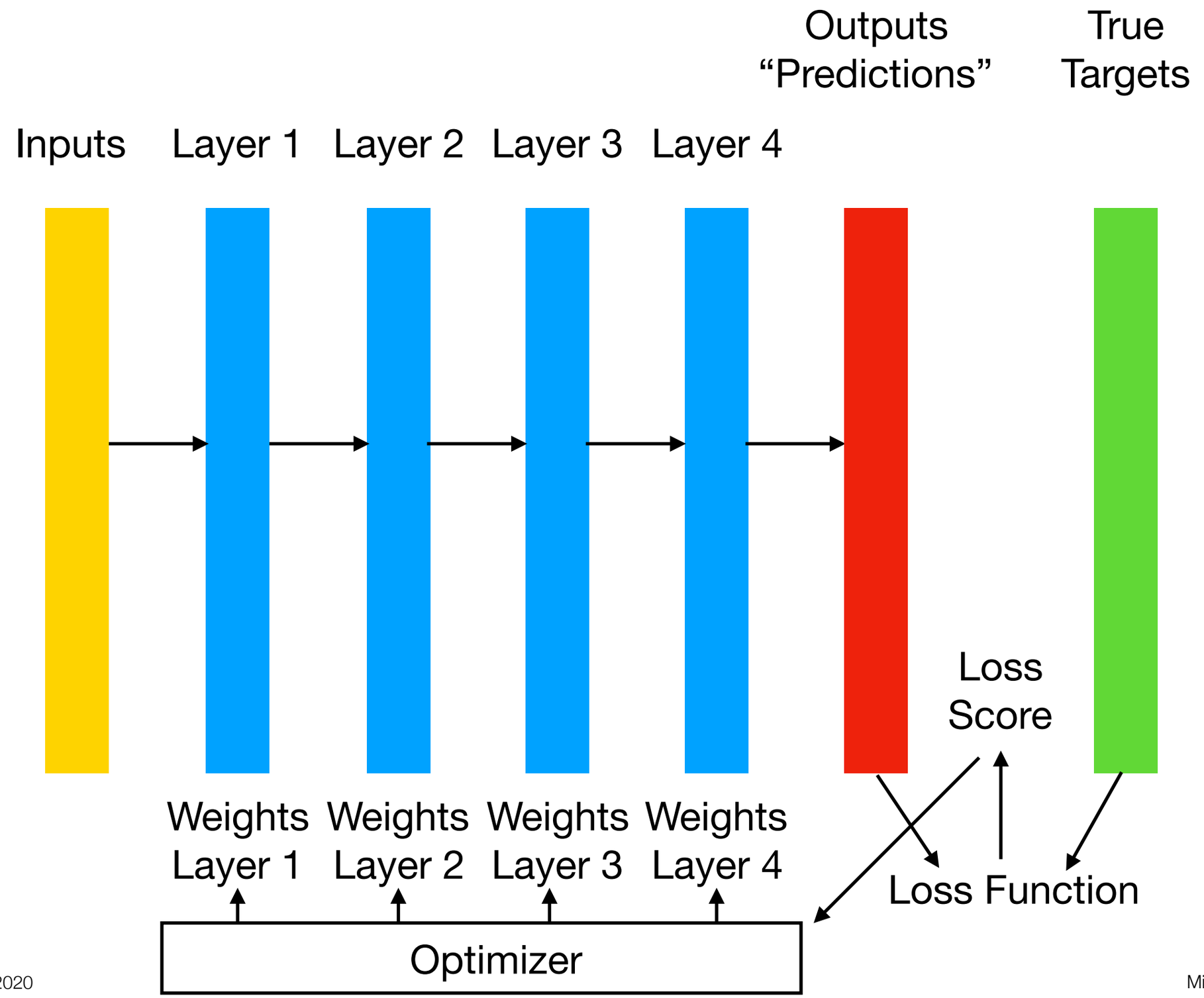
Deep Neural Networks



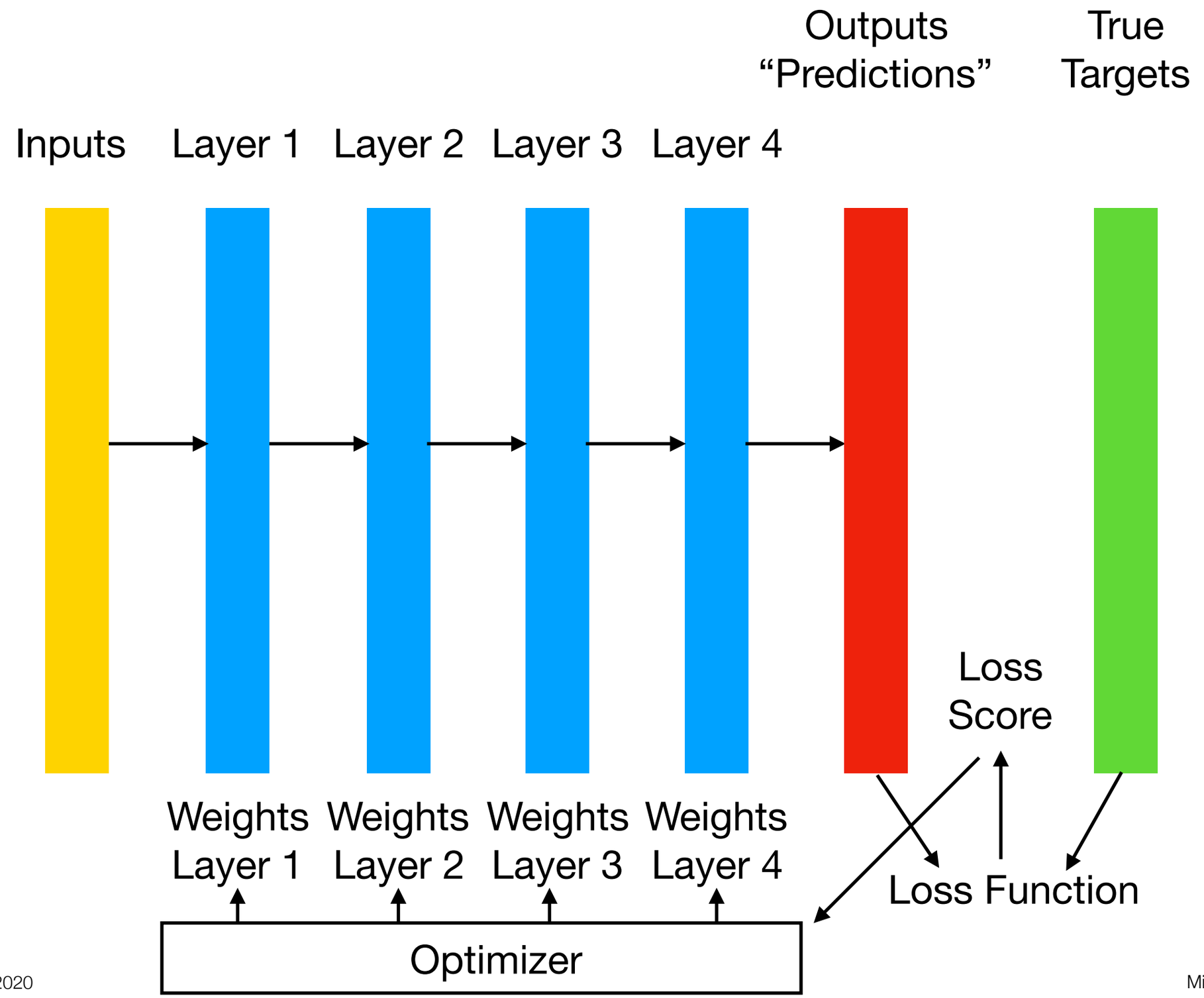
Deep Neural Networks



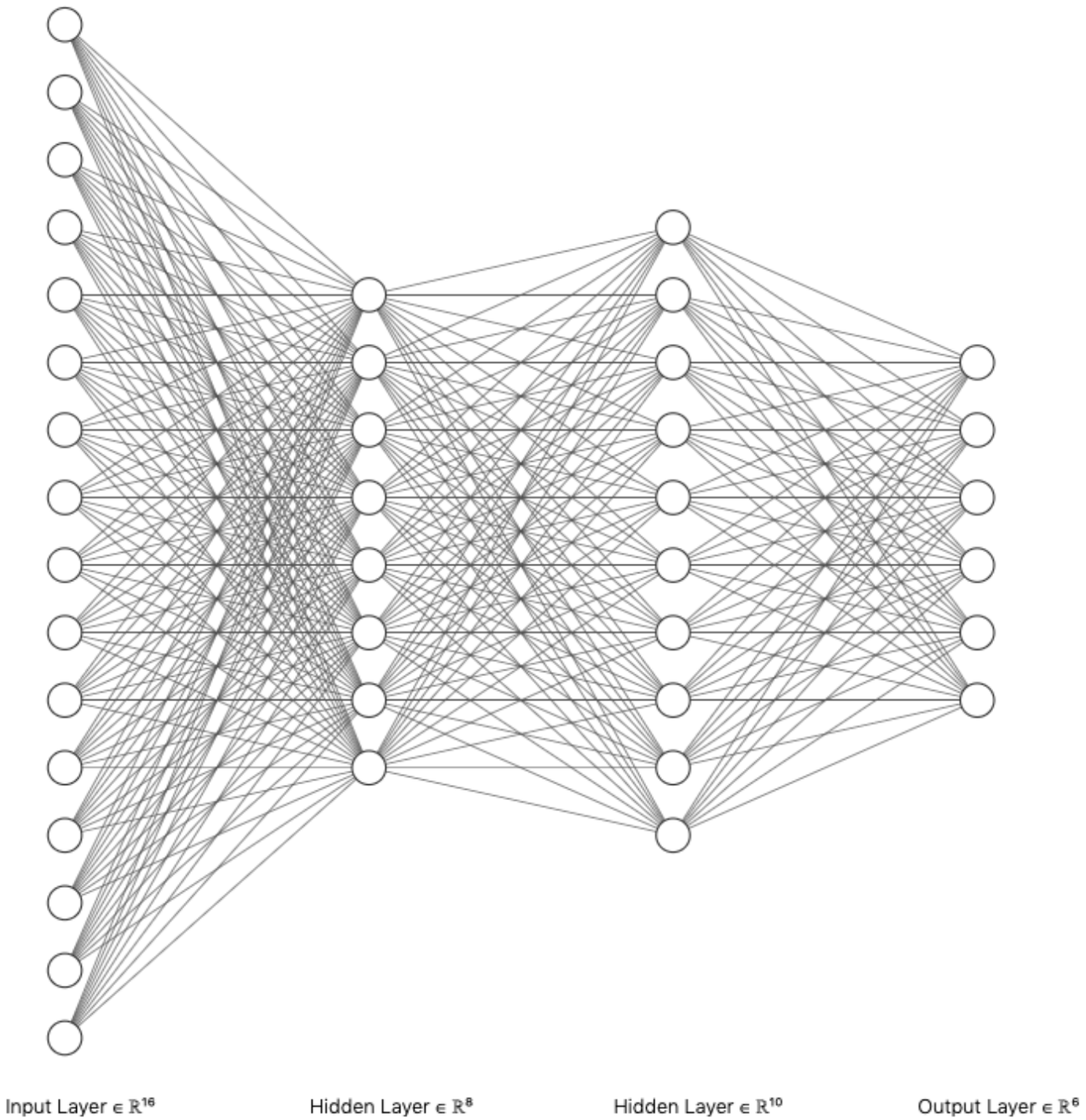
Deep Neural Networks



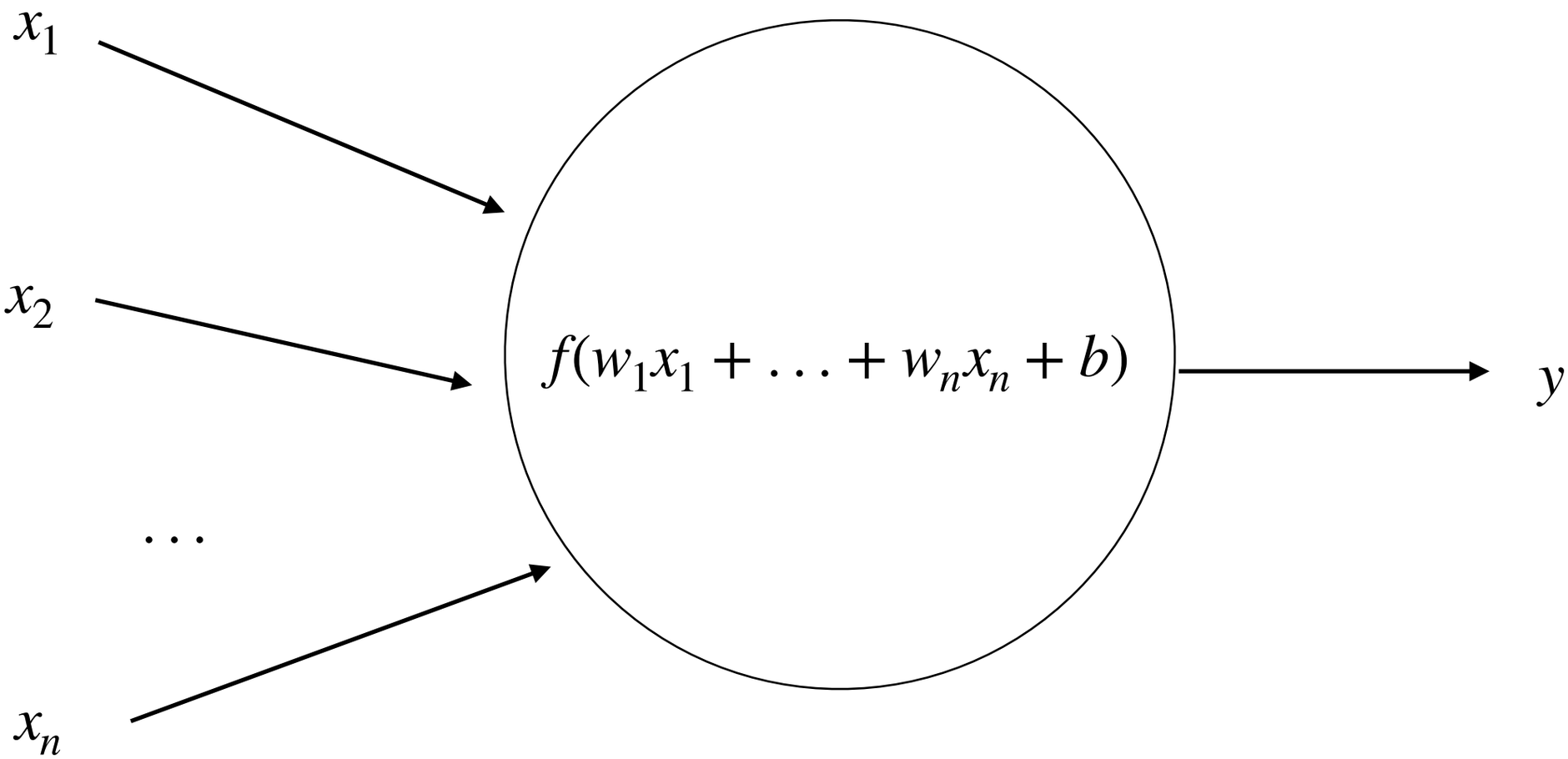
Deep Neural Networks



Deep Neural Networks



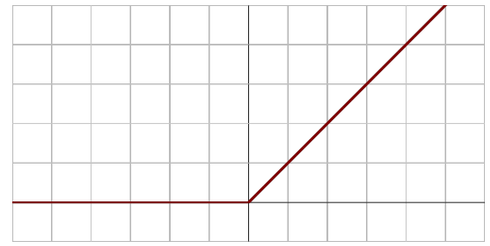
Nodes/Units/Neurons



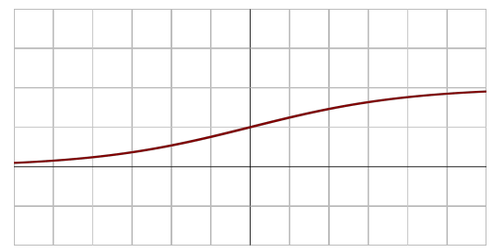
f is called the activation function, b is usually called the bias

Activations Functions

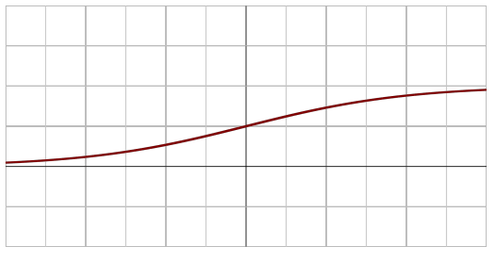
- They are generally used to add non-linearity.
- Examples:
  - *Rectified Linear Unit*: it returns the max between 0 and the value in input. In other words, given the value z in input it returns $max(0,z)$.



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

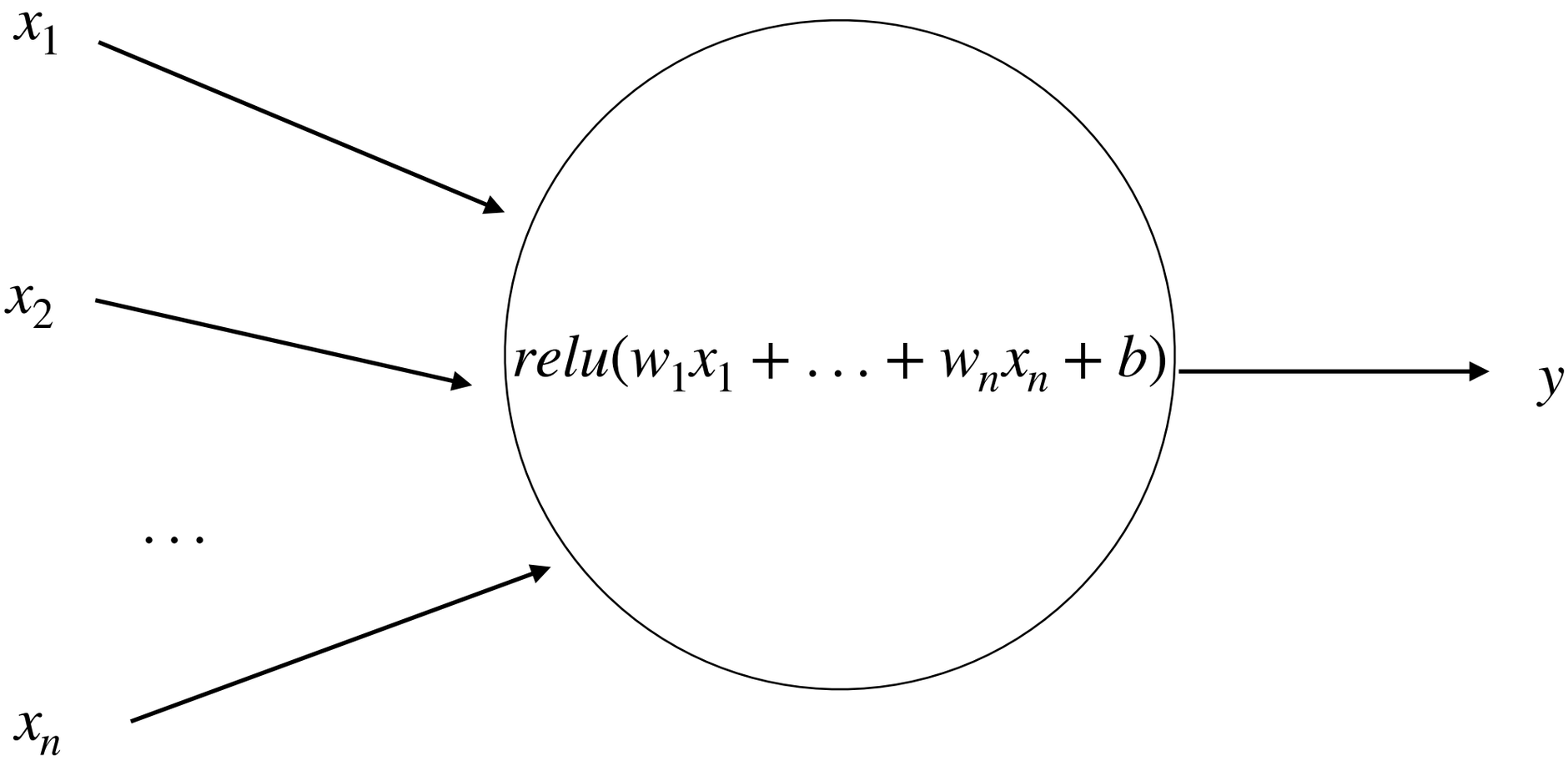


  - *Arctan*: given the value in input z, it returns $tan^{-1}(z)$.



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 in inputs 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 Optimization

- 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.

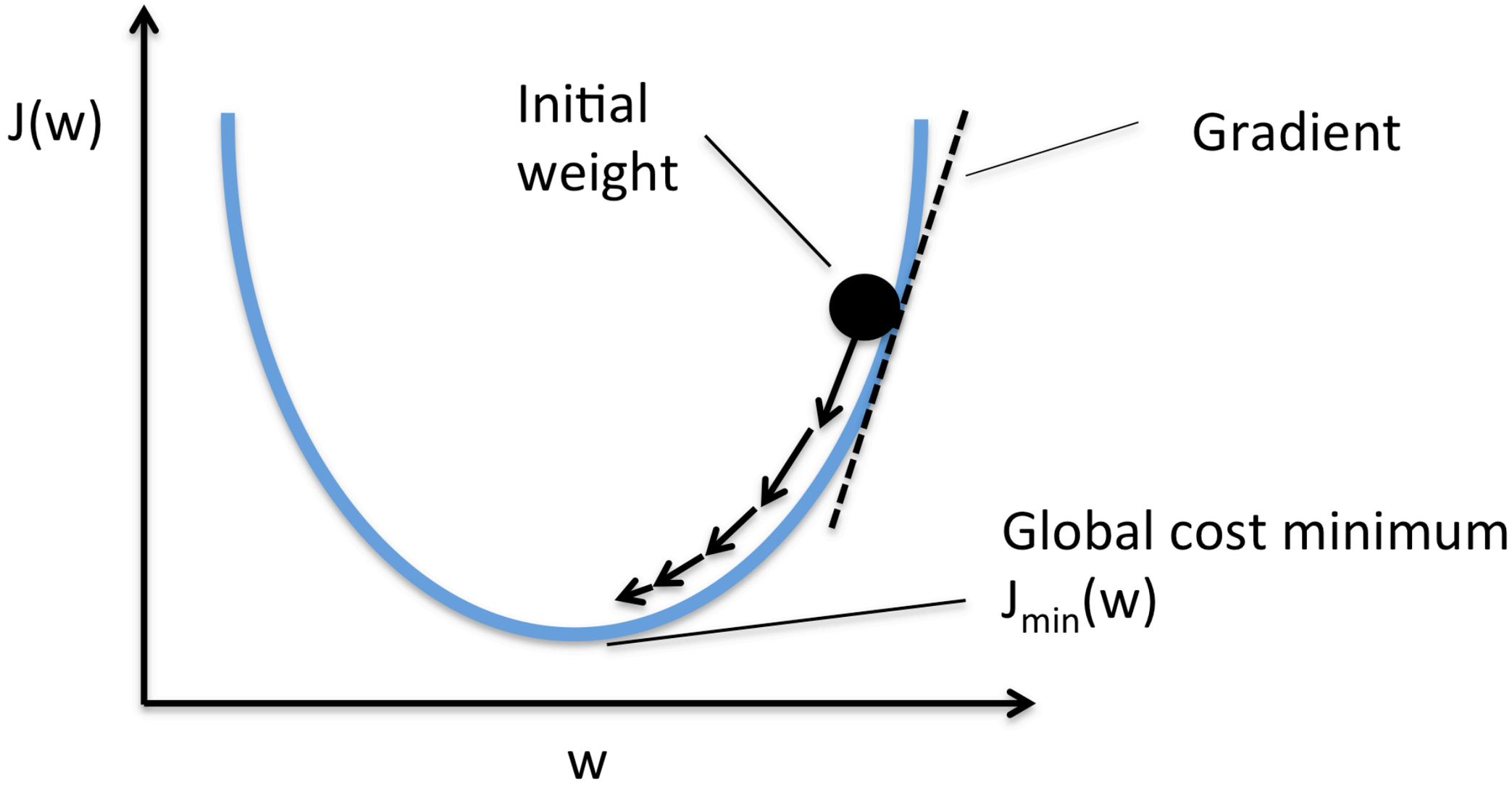
Stochastic Gradient Descent

- 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 describe above.

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

Gradient-based Optimisation



Credit: Sebastian Raschka

Stochastic Gradient Descent

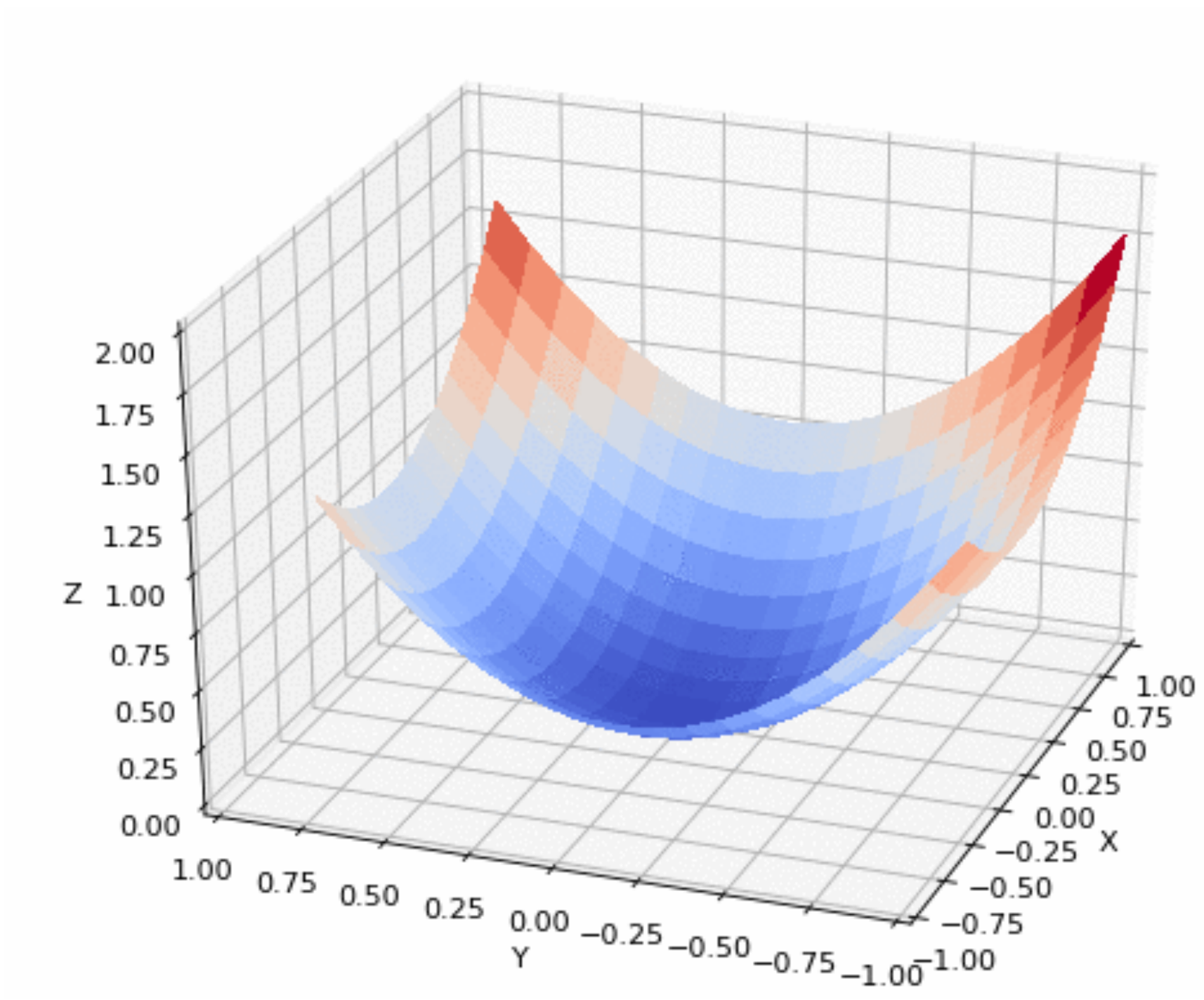
- More formally:
  - Draw a batch of training example $\mathbf{x}$ and corresponding targets $\mathbf{y}_{target}$.
  - Run the network on $\mathbf{x}$ (forward pass) to obtain predictions $\mathbf{y}_{pred}$.
  - Computer the loss of the network on the batch, a measure of the mismatch between $\mathbf{y}_{pred}$ 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:

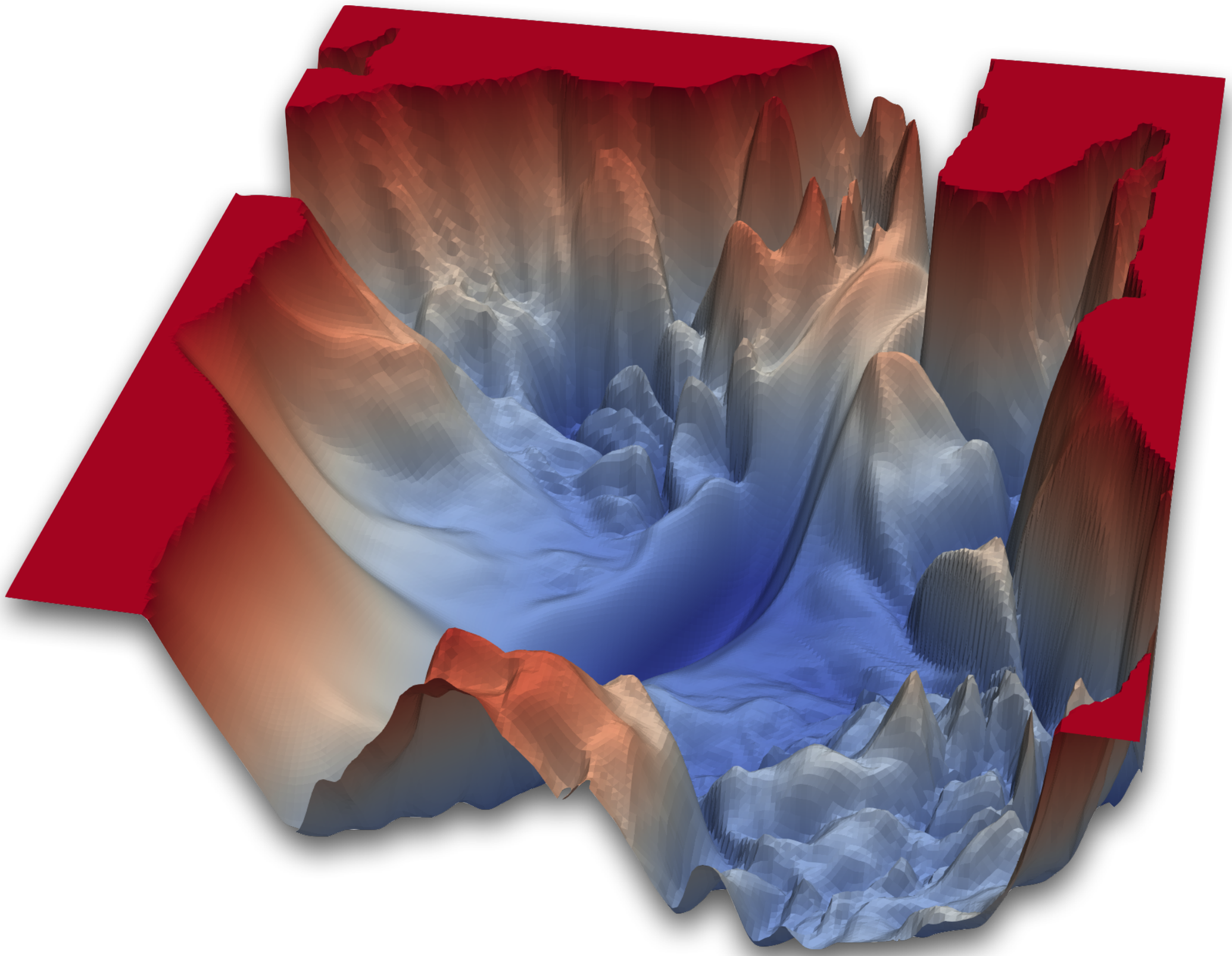
$w_j \leftarrow w_j + \Delta w_j = w_j - \eta \; average(\frac{\partial J_k}{\partial w_j})$ for all the k samples of the batch.

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 describe 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 functon:

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

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 connect 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 framework like TensorFlow that supports symbolic differentiation for the calculation of the gradient.

- TensorFlow supports 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.

References

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

- Chapter 2 of Francois Chollet. Deep Learning with Python. Manning 2018.