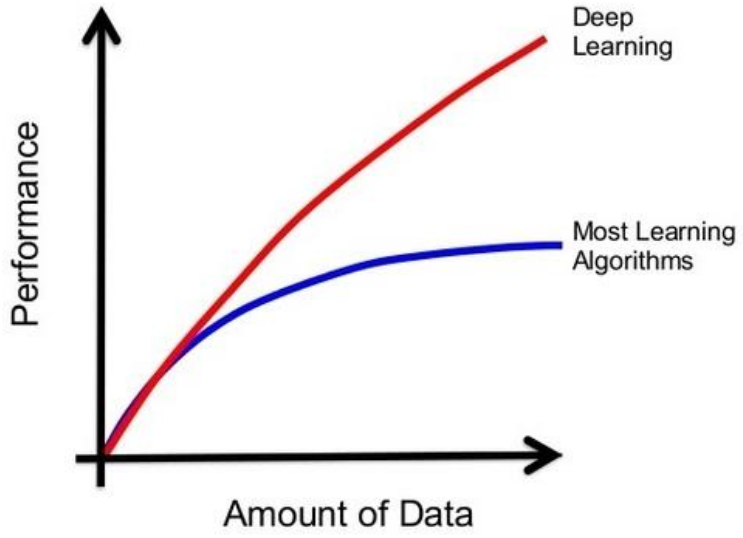
What is Deep Learning ?

Deep learning is a subset of machine learning. Usually, when people use the term deep learning, they are referring to deep artificial neural networks.

Engineered systems inspired by the structure and function of the brain.

Deep refers to the number of layers typically and so this kind of the popular term that's been adopted in the press.



As we construct larger neural networks and train them with more and more data, their performance continues to increase. This is generally different to other machine learning techniques that reach a plateau in performance.

Why now ?

- Algorithm advancements
- Powerful hardware
- Availability of large training set

Neural computation

- The study of neurons, their interconnections, and their role as the brain's elementary building blocks is one of the most dynamic and important research fields in modern biology.
- Between 1901 and 1991 approximately ten percent of the Nobel Prizes for Physiology and Medicine were awarded to scientists who contributed to the understanding of the brain.
- Artificial neural networks are an attempt at modeling the information processing capabilities of nervous systems.
- Artificial neural networks can be considered as another approach to the problem of computation.

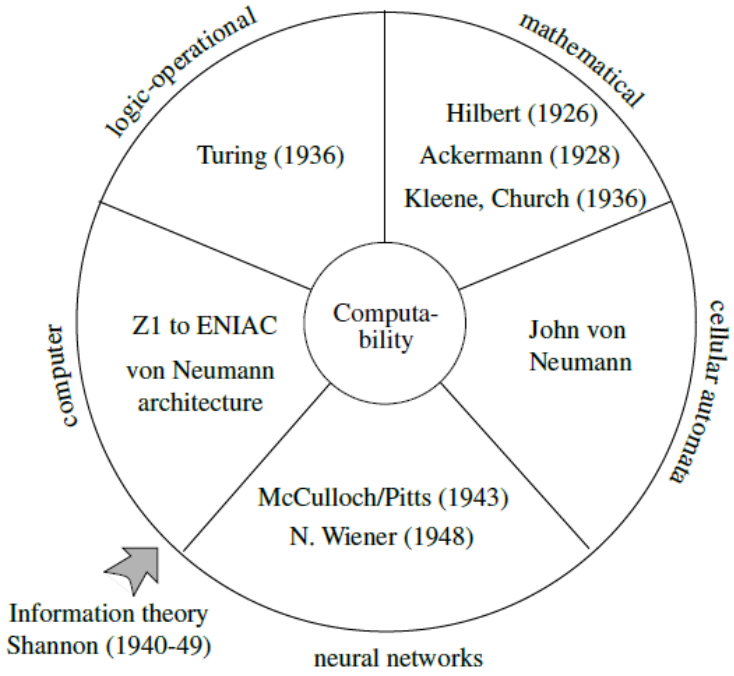


Fig. 1.1. Five models of computation

$$computation = storage + transmission + processing.$$

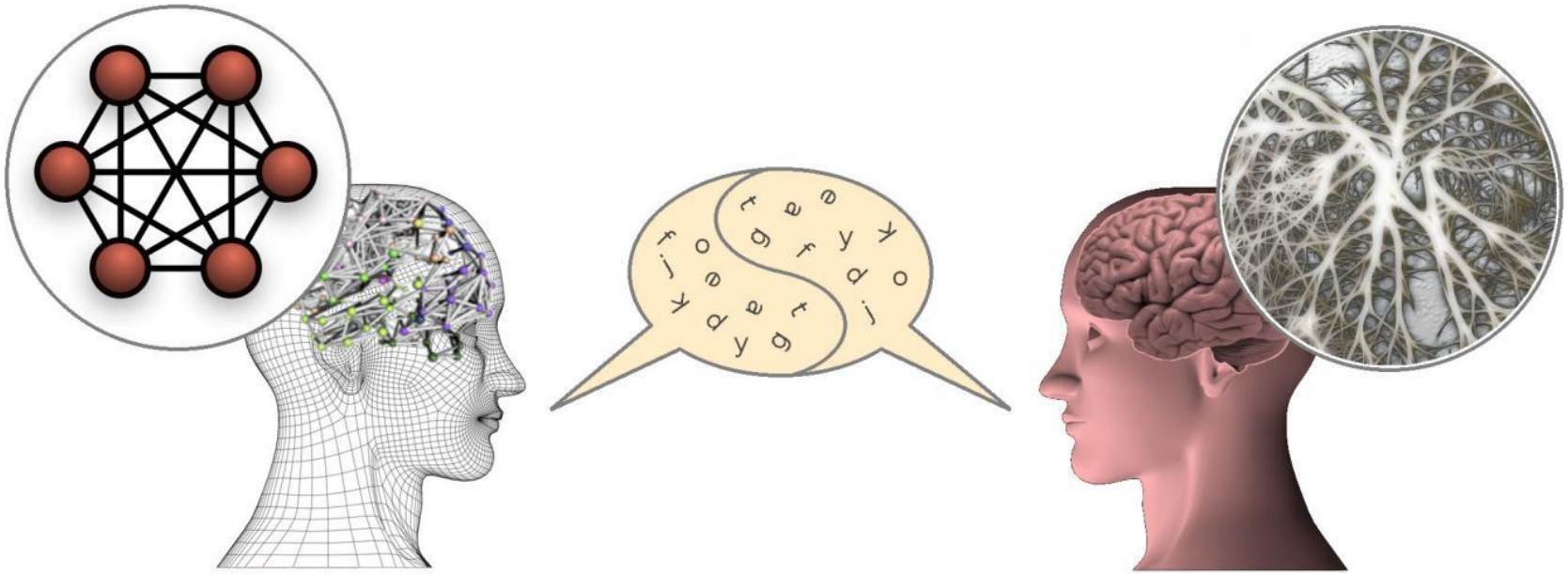
Ref: R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

What are Neural Networks used for?

There are two basic goals for neural network research:

Brain modelling: The biological goal of constructing models of how real brains work. This can potentially help us understand the nature of perception, actions, learning and memory, thought and intelligence and/or formulate medical solutions to brain damaged patients

Artificial System Construction: The engineering goal of building efficient systems for real world applications. This may make machines more powerful and intelligent, relieve humans of tedious tasks, and may even improve upon human performance.



BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

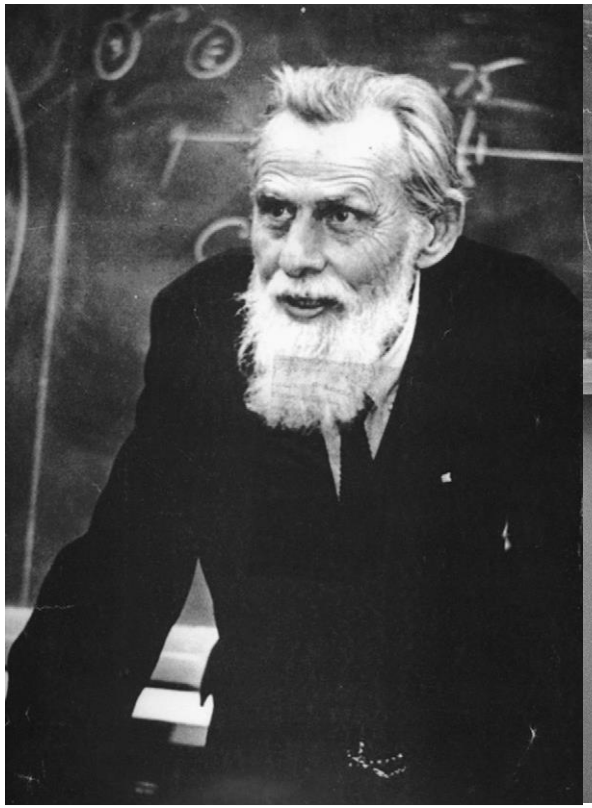
WARREN S. MCCULLOCH AND WALTER PITTS

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

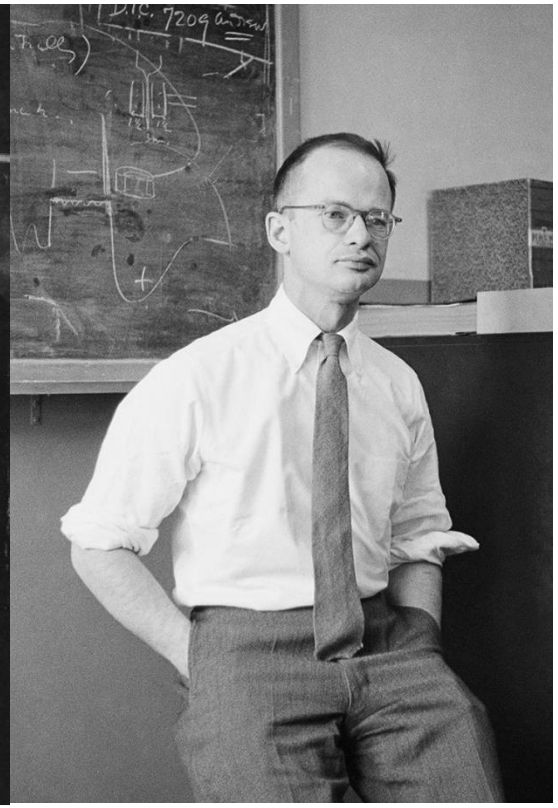
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; 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 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

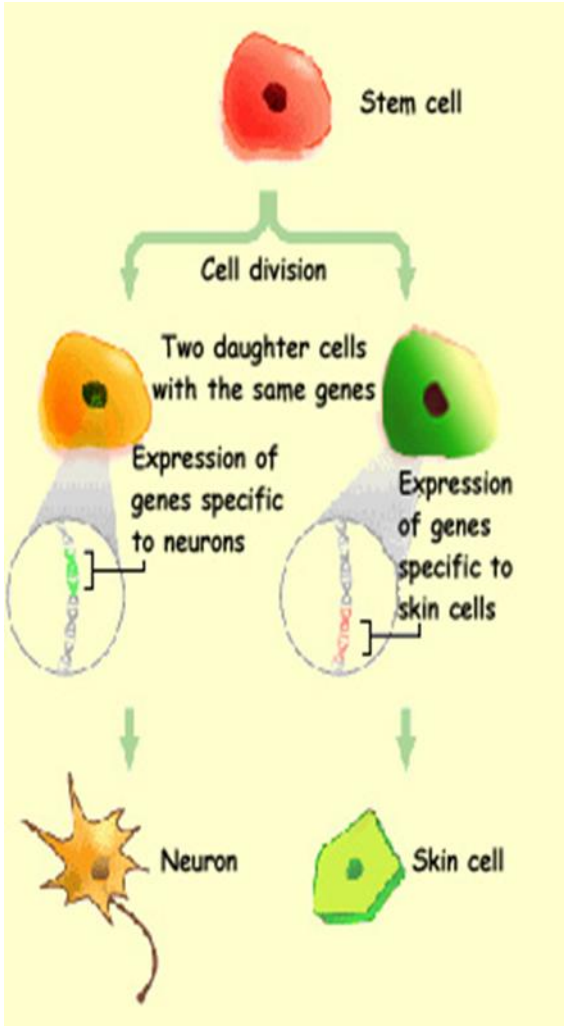


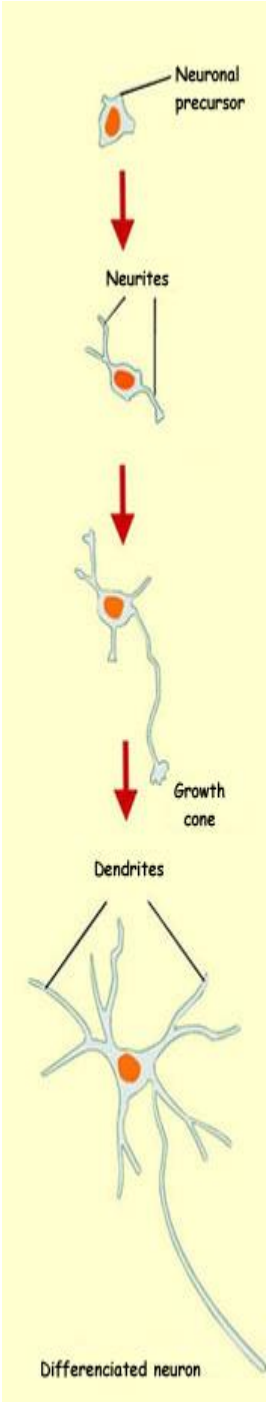
Warren McCulloch

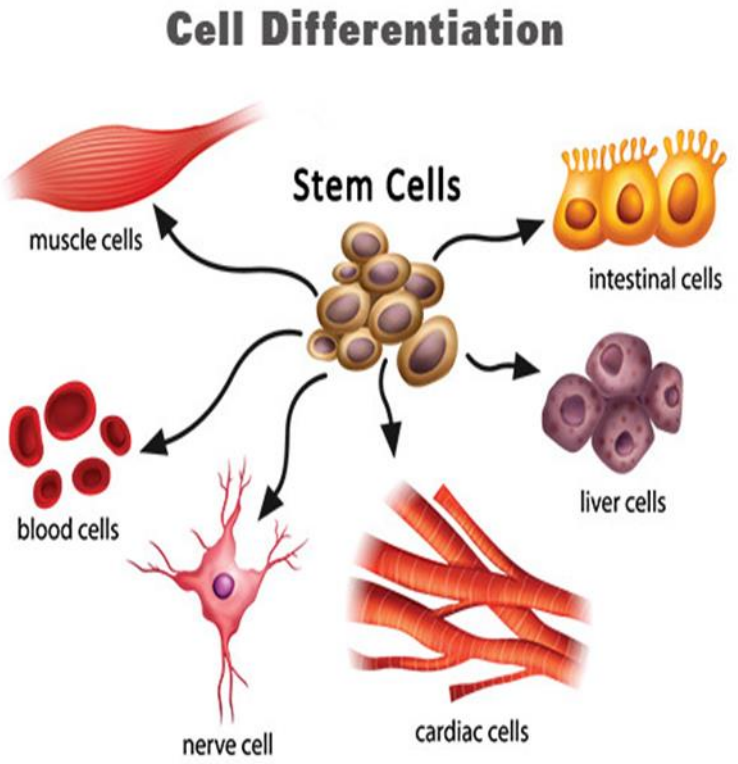


Walter Pitts

1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity". With Walter Pitts. In: *Bulletin of Mathematical Biophysics* Vol 5, pp 115–133.



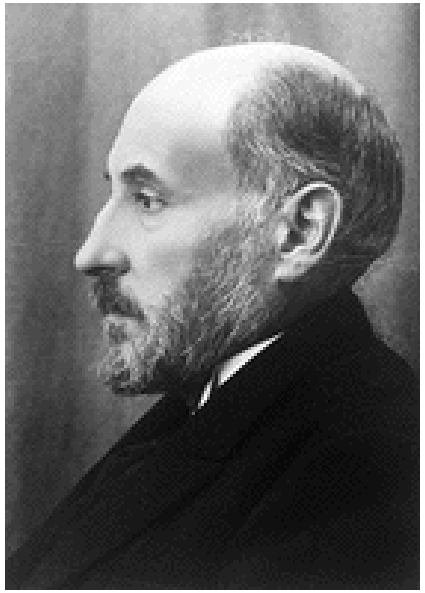




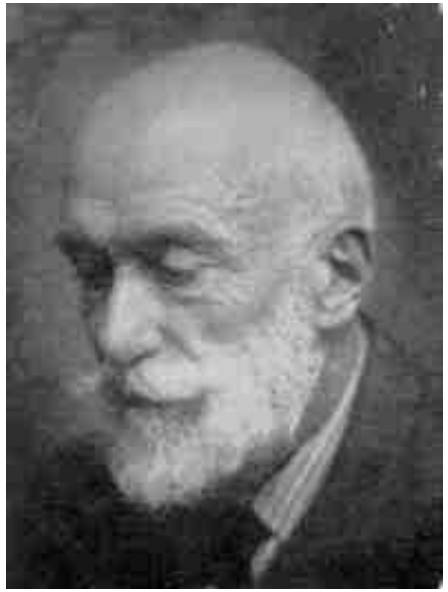
A newborn baby has about 26,000,000,000 cells.

The word neuron, as we understand it today, did not exist before 1891 (neuron doctrine)

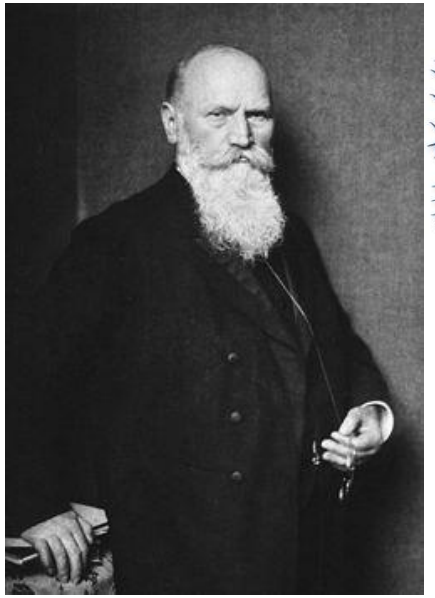
Camillo Golgi



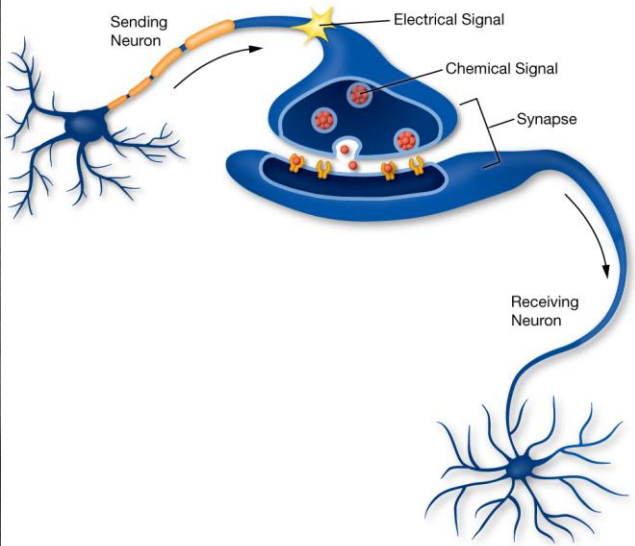
Santiago Ramón y Cajal

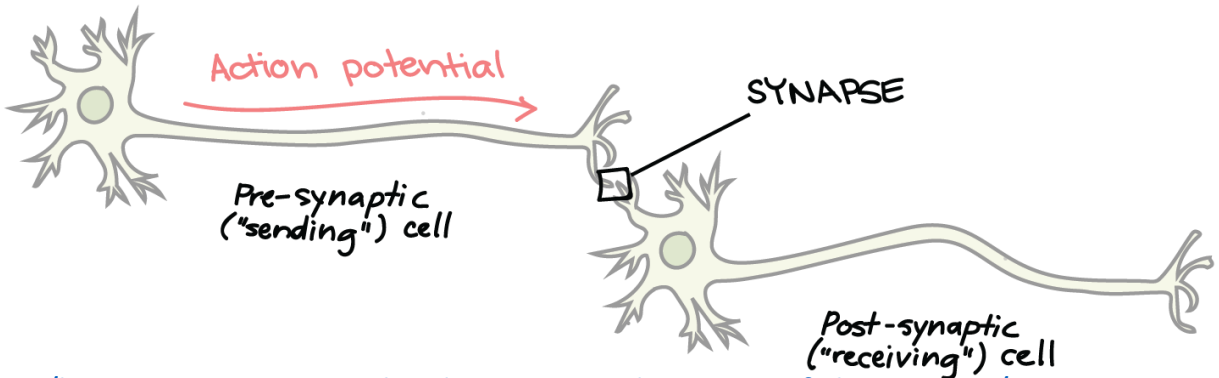


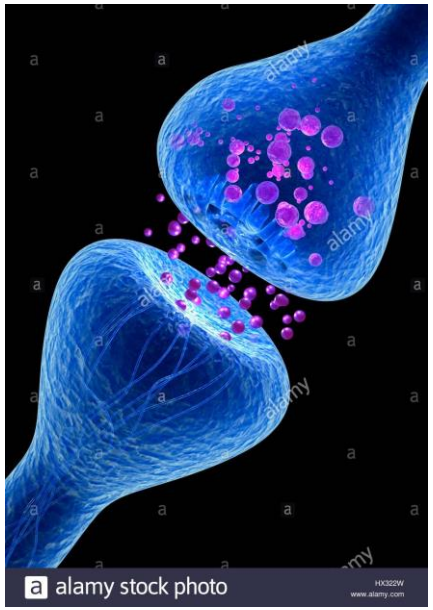
Auguste Forel



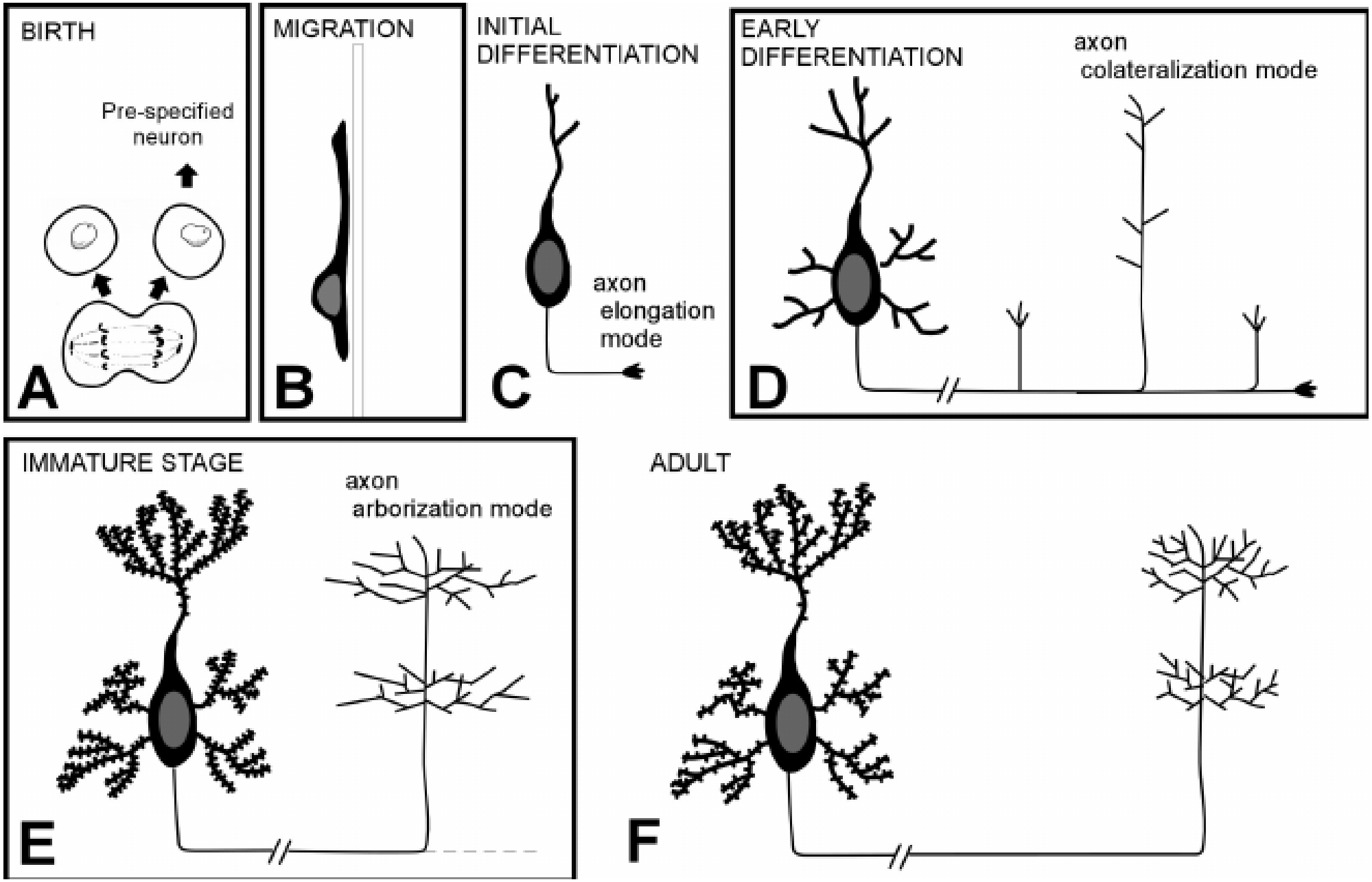
Heinrich Waldeyer







https://blogs.scientificamerican.com/brainwaves/know-your-neurons-the-discovery-and-naming-of-the-neuron/
http://cbm.msoe.edu/teacherWorkshops/ddtyResources/documents/historyOfTheNeuron2.pdf
http://www.cerebromente.org.br/n17/history/neurons3_i.htm



https://www.researchgate.net/figure/Life-history-of-a-projection-neuron-A-All-neurons-are-born-with-a-simple-round_fig1_226096790

The human body is made up of trillions of cells. Cells of the nervous system, called nerve cells or **neurons**, are specialized to carry "messages" through an electrochemical process.

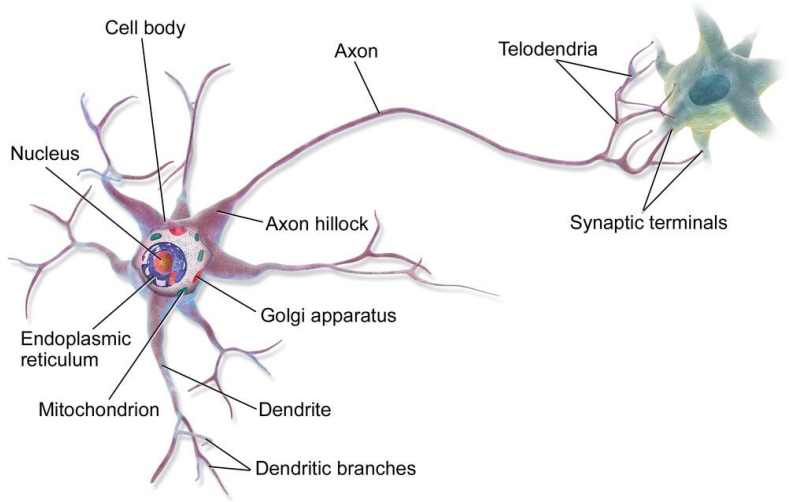
Neurons come in many different shapes and sizes. Some of the smallest neurons have cell bodies that are only 4 microns wide. Some of the biggest neurons have cell bodies that are 100 microns wide.

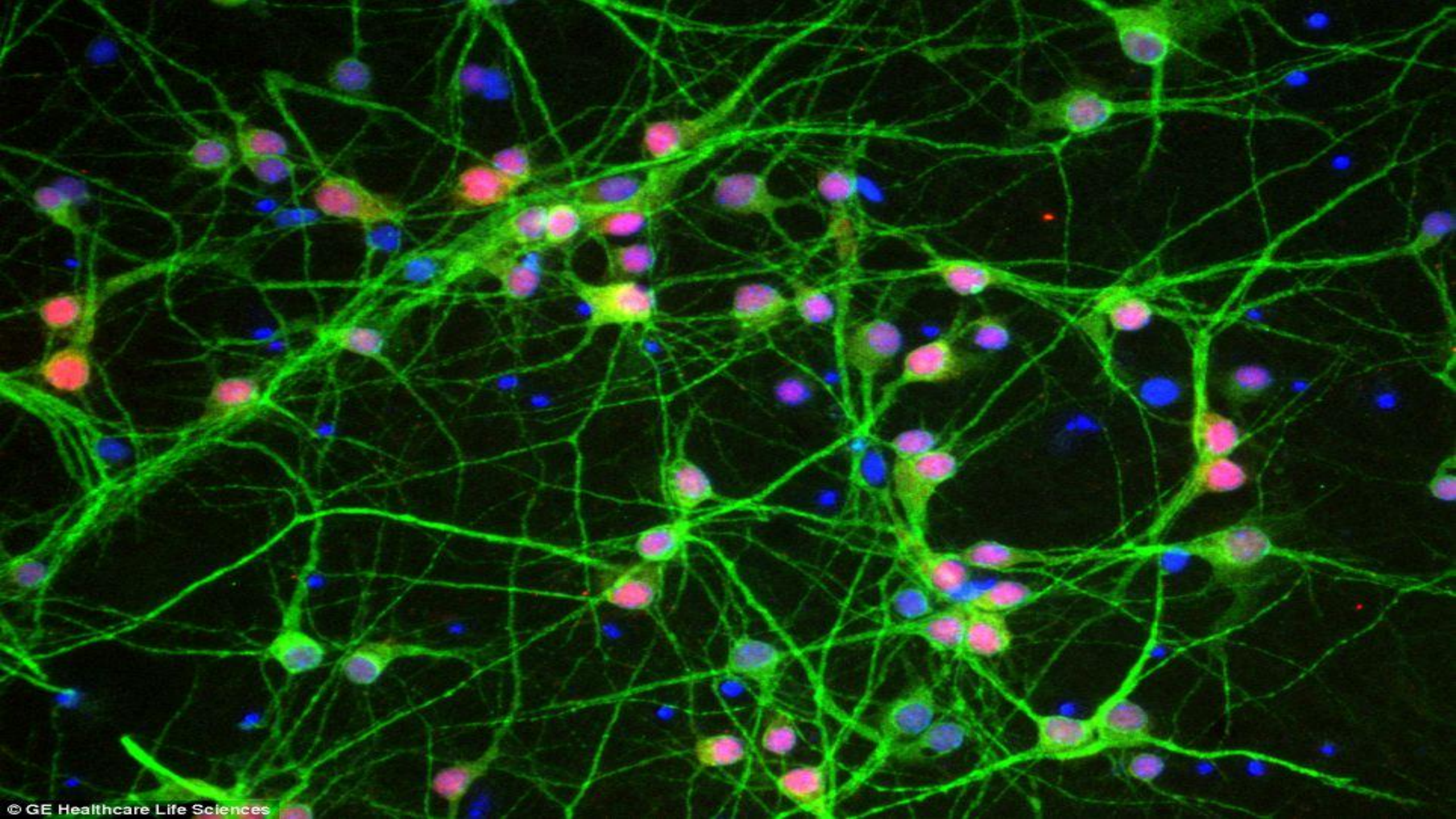
Typically, a neuron contains three important parts:

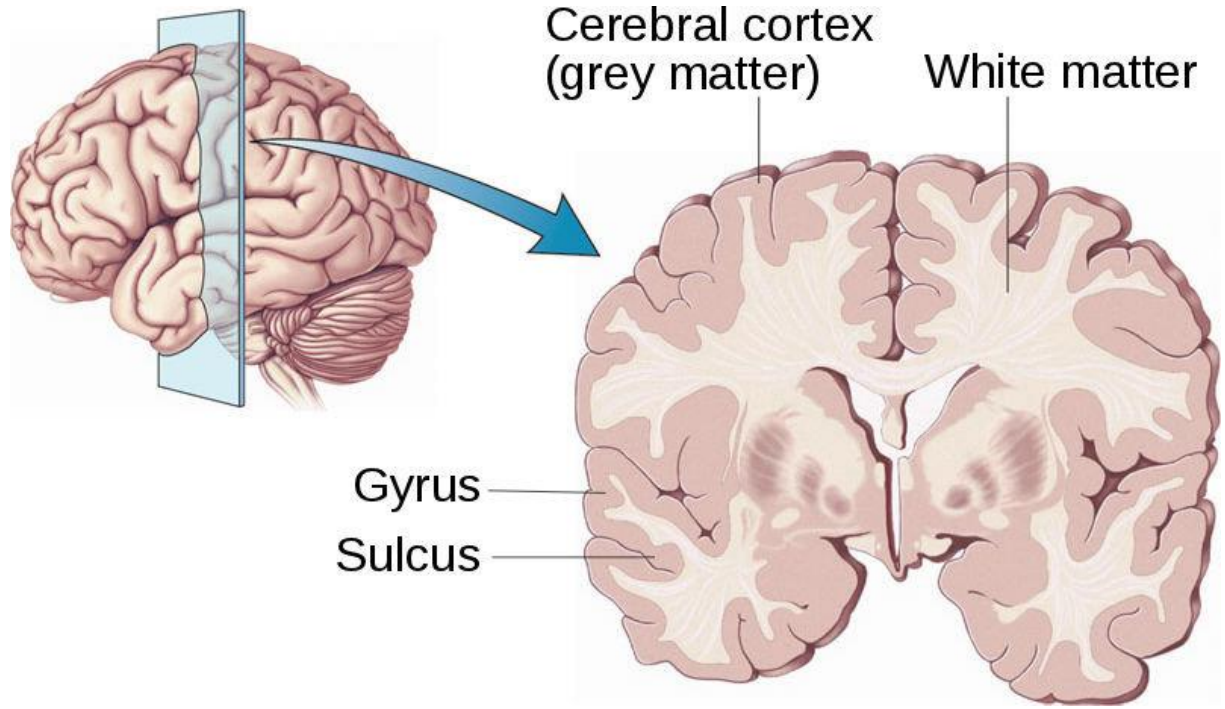
•a *cell body* that directs all activities of the neuron.

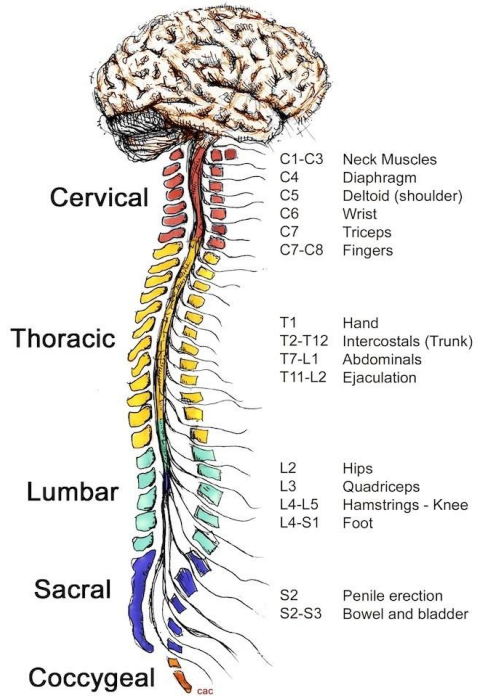
•*dendrites* (the part that looks like tree branches), which are short fibers that receive messages from other neurons and relay those messages to the cell body.

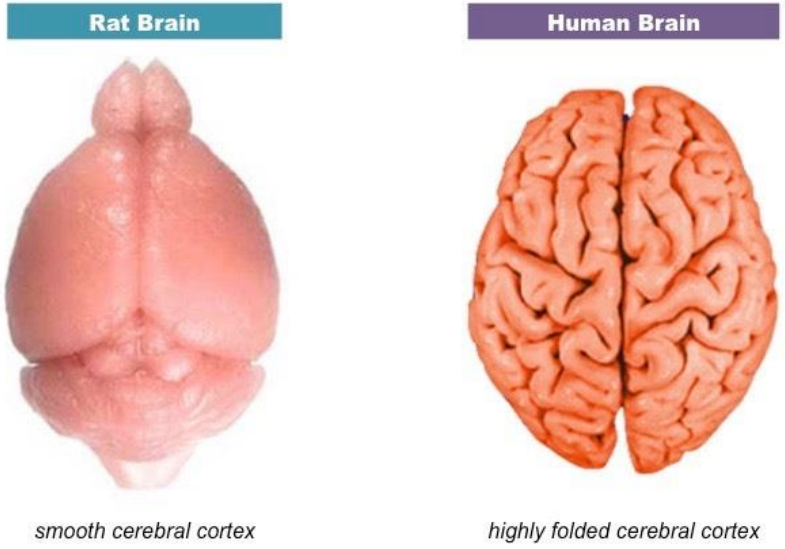
•*axon*, a long single fiber that transmits messages from the cell body to dendrites of other neurons.

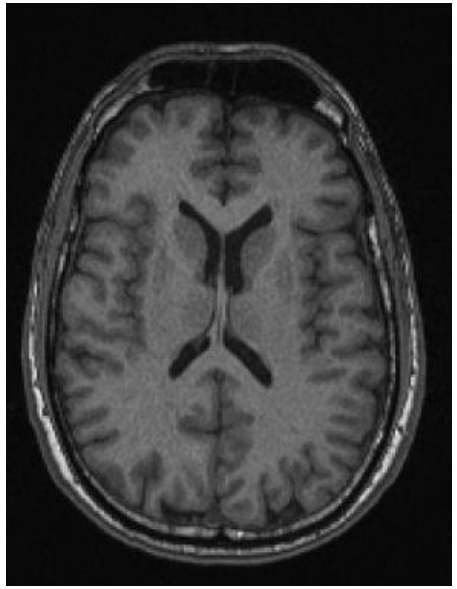


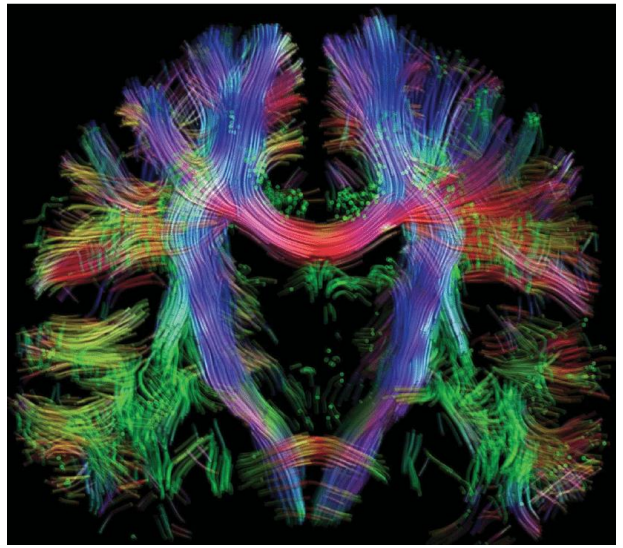

© GE Healthcare Life Sciences

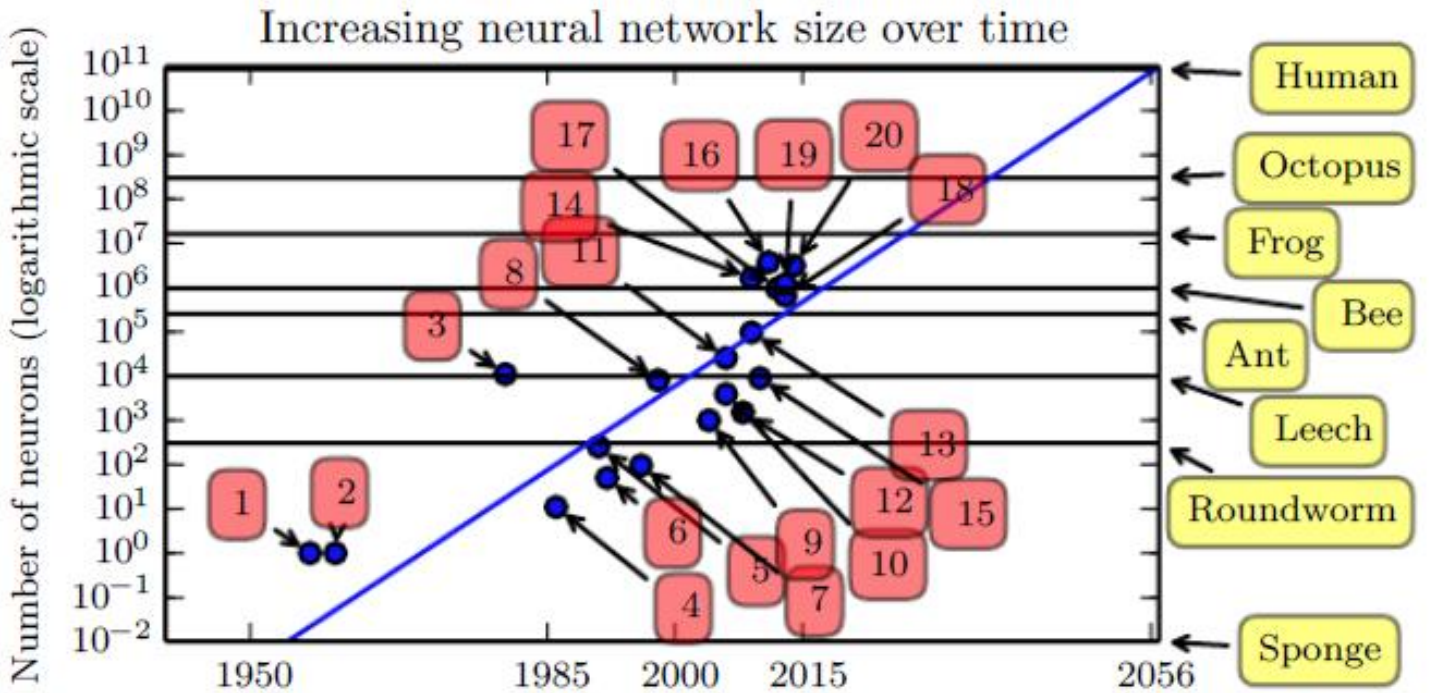

Cerebral cortex
(grey matter)
White matter
Gyrus
Sulcus


Cervical
C1-C3 Neck Muscles
C4 Diaphragm
C5 Deltoid (shoulder)
C6 Wrist
C7 Triceps
C7-C8 Fingers
Thoracic
T1 Hand
T2-T12 Intercostals (Trunk)
T7-L1 Abdominals
T11-L2 Ejaculation
Lumbar
L2 Hips
L3 Quadriceps
L4-L5 Hamstrings - Knee
L4-S1 Foot
Sacral
S2 Penile erection
S2-S3 Bowel and bladder
Coccygeal
cac

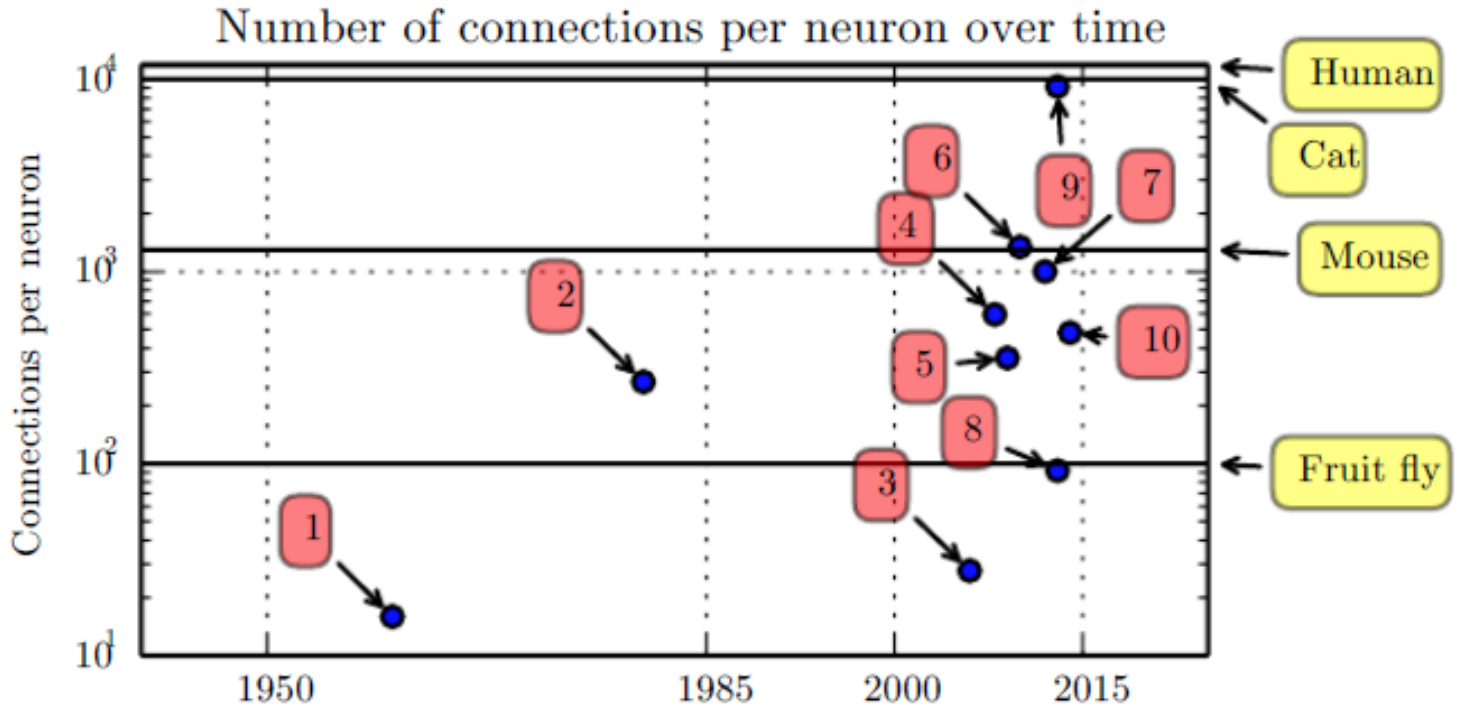

Rat Brain
Human Brain
smooth cerebral cortex
highly folded cerebral cortex







1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)



1. Adaptive linear element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)
3. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
4. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
5. Unsupervised convolutional network (Jarrett *et al.*, 2009)
6. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
7. Distributed autoencoder (Le *et al.*, 2012)
8. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
9. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
10. GoogLeNet (Szegedy *et al.*, 2014a)

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

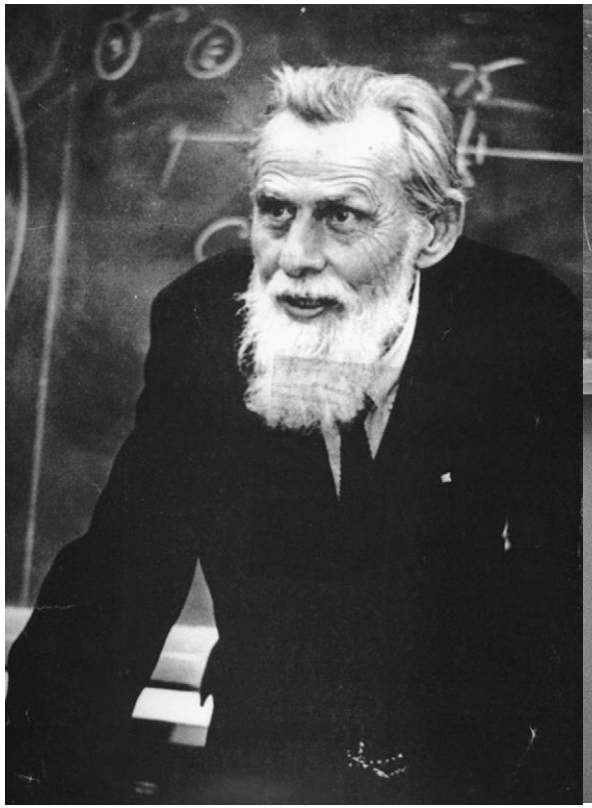
WARREN S. MCCULLOCH AND WALTER PITTS

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

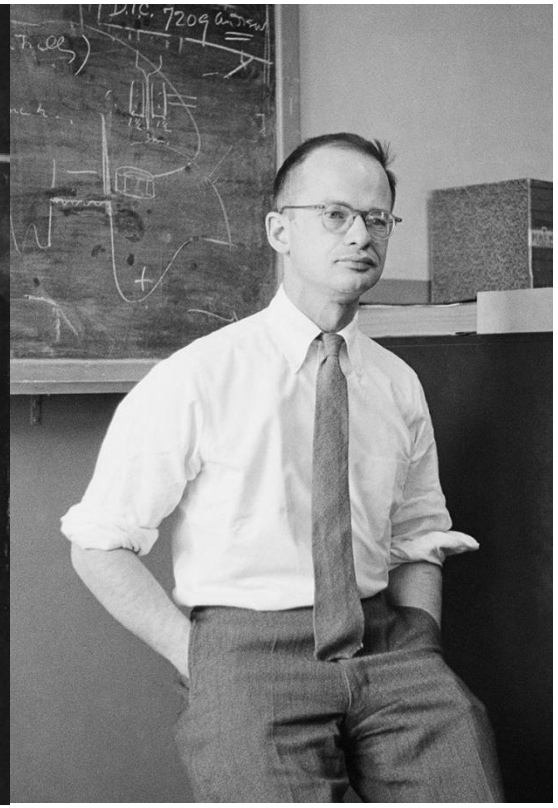
Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; 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 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



Warren McCulloch

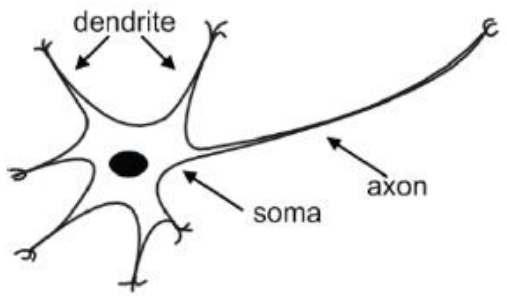


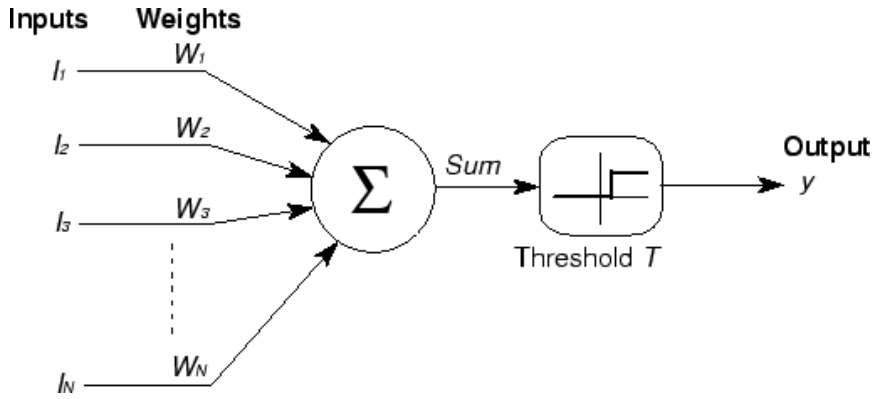
Walter Pitts

1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity". With Walter Pitts. In: *Bulletin of Mathematical Biophysics* Vol 5, pp 115–133.

The McCulloch-Pitts Model of Neuron (1942 model)

The early model of an artificial neuron is introduced by Warren McCulloch and Walter Pitts in 1943.





$$Sum = \sum_{i=1}^{N} I_i W_i$$

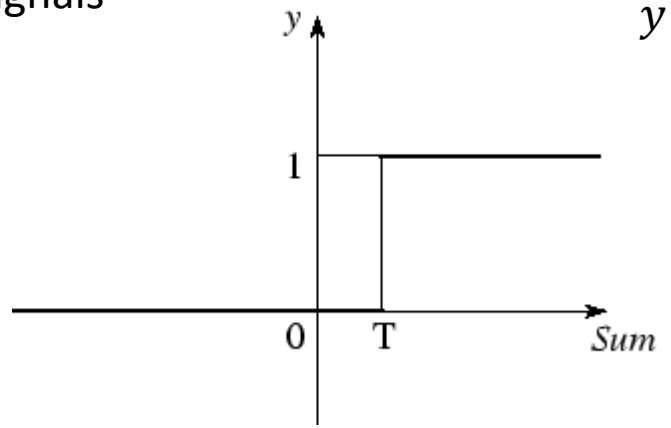
$$Output = \begin{cases} 0 & \text{if Sum} \leq T \\ 1 & \text{if Sum} > T \end{cases}$$

The main feature of their neuron model is that a weighted sum of input signals is compared to a threshold to determine the neuron output.

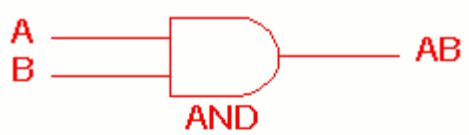
When the sum is greater than or equal to the threshold, the output is 1.

When the sum is less than the threshold, the output is 0.

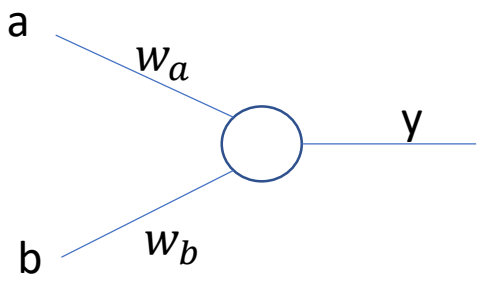
$$y = f(Sum)$$



- They demonstrated that networks of these neurons could, in principle, compute any arithmetic or logical function.
- Unlike biological networks, the parameters of their networks had to be designed, as no training method was available.
- However, the perceived connection between biology and digital computers generated a great deal of interest.



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



$$y = \begin{cases} 0 & \text{if Sum} \leq T \\ 1 & \text{if Sum} > T \end{cases}$$

$T \geq 0$

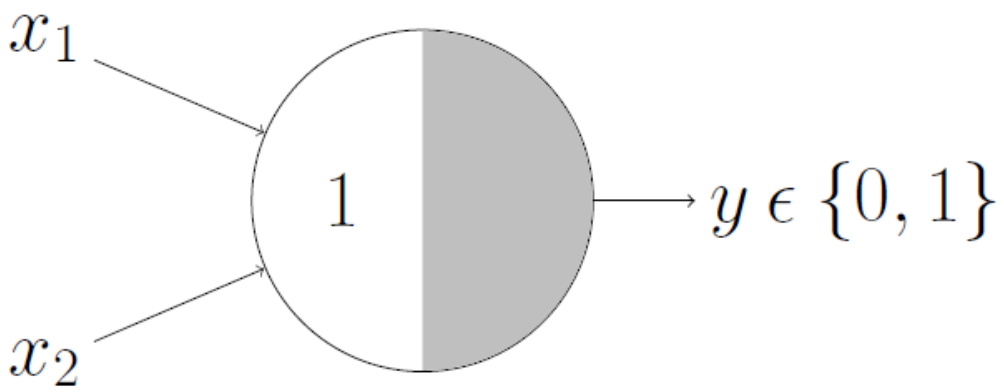
$w_b \leq T$

$w_a \leq T$

$w_a + w_b > T$

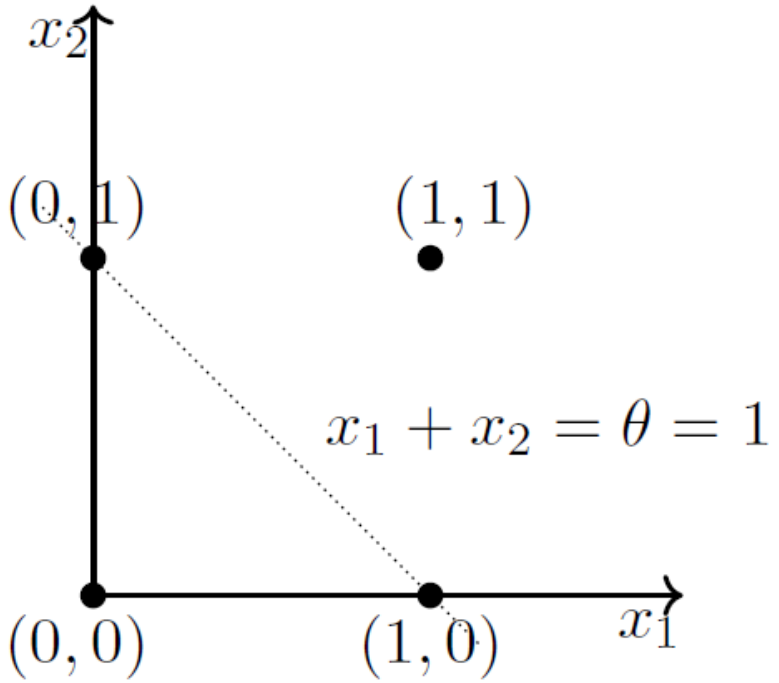
$T = 1.5, w_b = 1, w_b = 1$

Geometric Interpretation Of M-P Neuron

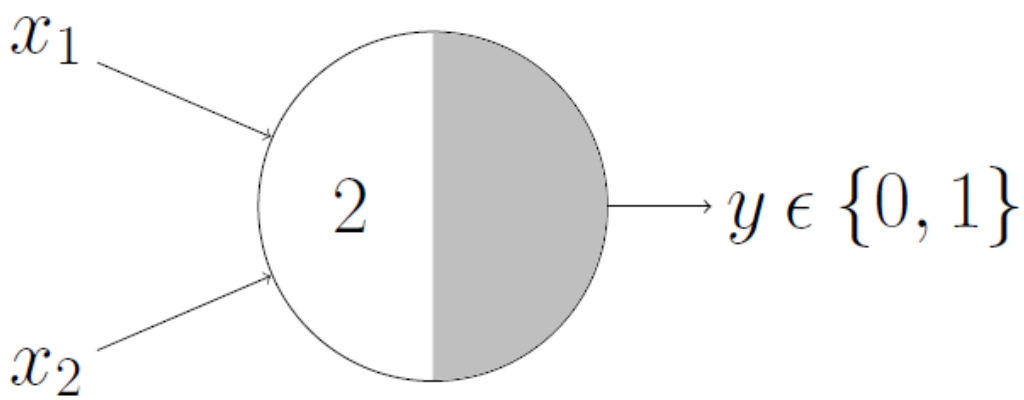


OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

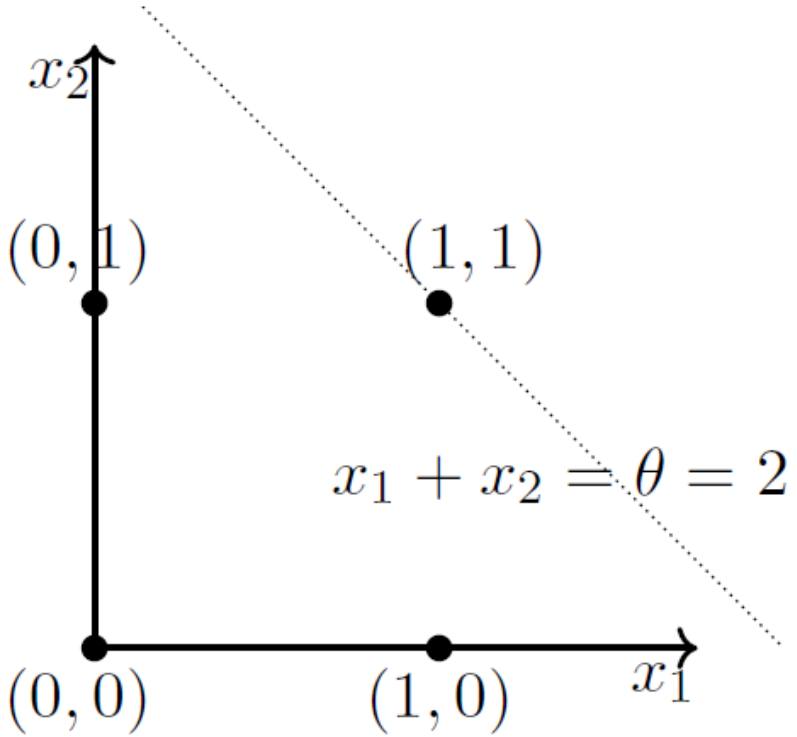


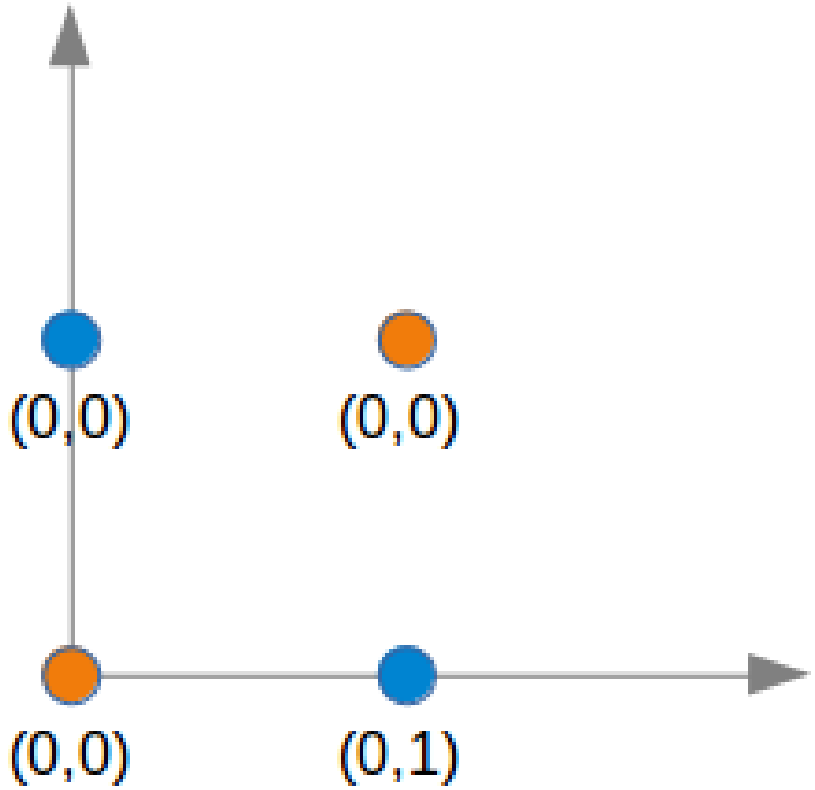
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

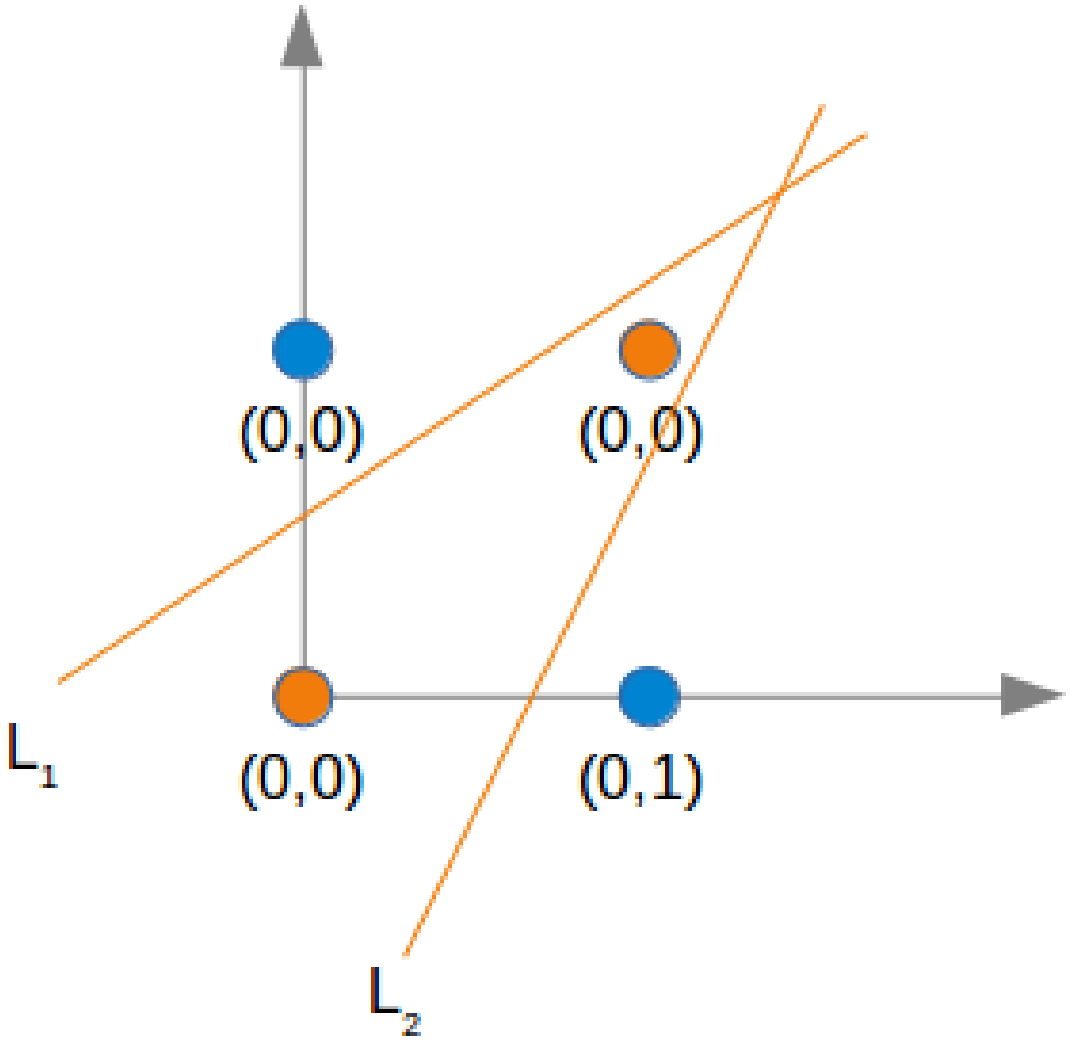


AND function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$







https://www.python-course.eu/neural_networks.php

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 recognition and behavior?

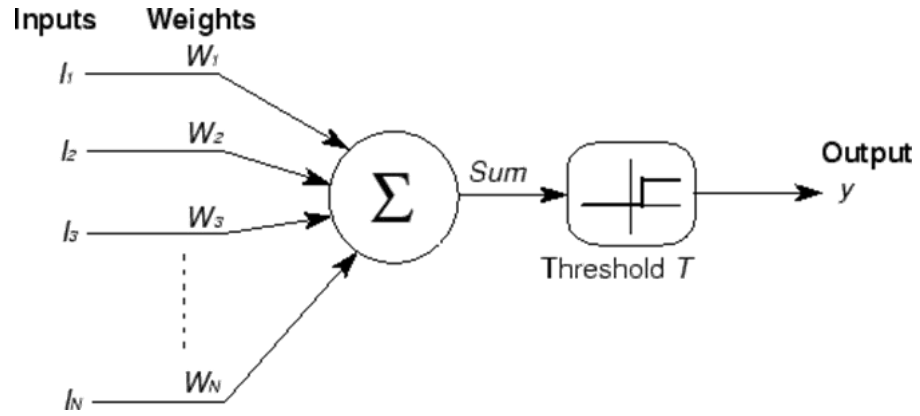
The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a vast amount of speculation, and where the few relevant facts currently supplied by neurophysiology have not yet been integrated into an acceptable theory.

With regard to the second question, two alternative positions have been maintained. The first suggests that storage of sensory information is in the form of coded representations or images, with some sort of one-to-one mapping between the sensory stimulus 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, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (**2**, **3**, **9**, **14**). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching network, where retention takes the form of new connections, or pathways, between centers of activity. In many of the more recent developments of this position (Hebb's "cell assembly," and Hull's "cortical anticipatory goal response," for example) the "responses" which are associated to stimuli may be entirely contained within the CNS itself. In this case the response represents an "idea" rather than an action. The important feature of this approach is that there is never any simple mapping of the stimulus into memory, according to some code which would permit its later reconstruction. Whatever in-

[1] The development of this theory has been carried out at the Cornell Aeronautical Laboratory, Inc., under the sponsorship of the Office of Naval Research, Contract Nonr-2381(00). This article is primarily an adaptation of material reported in Ref. 15, which constitutes the first full report on the program.

Frank Rosenblatt

Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408.



- The neurons in these networks were similar to those of McCulloch and Pitts.
- Rosenblatt's key contribution was the **introduction of a learning rule** for training perceptron networks to solve pattern recognition problems
- In addition to the variable weight values, the perceptron model added an extra input that represents *bias*. Thus, the modified equation from is now as follows:

$$Sum = \sum_{i=1}^{N} I_i W_i + b$$

$$\text{bias}, b = -t$$

$$Output = \begin{cases} 0 & \text{if Sum} \leq 0 \\ 1 & \text{if Sum} > 0 \end{cases}$$

Bias is a measure of how easy it is to get the perceptron to output 1.

Perceptron learning rule

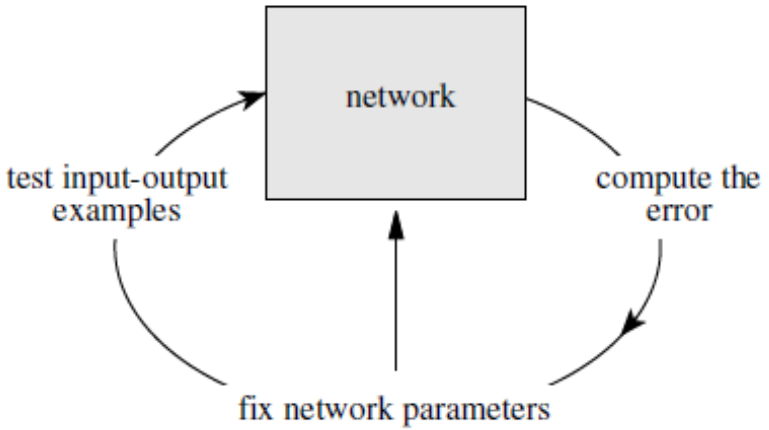
$$w(i+1) = w(i) + \Delta w$$

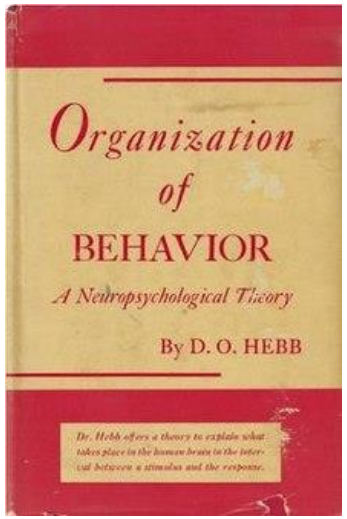
$$\Delta w = \eta . Error . x$$

$$Error = t - y$$

$$b(i+1) = b + \eta E$$

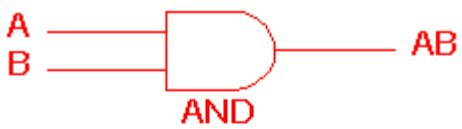
η is the learning rate



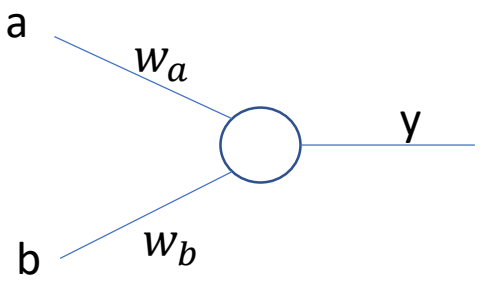


The Perceptron convergence theorem states that for any data set which is linearly separable the Perceptron learning rule is guaranteed to find a solution in a finite number of steps.

Hebb, D.O. (1949). The Organization of Behavior. New York: Wiley & Sons.



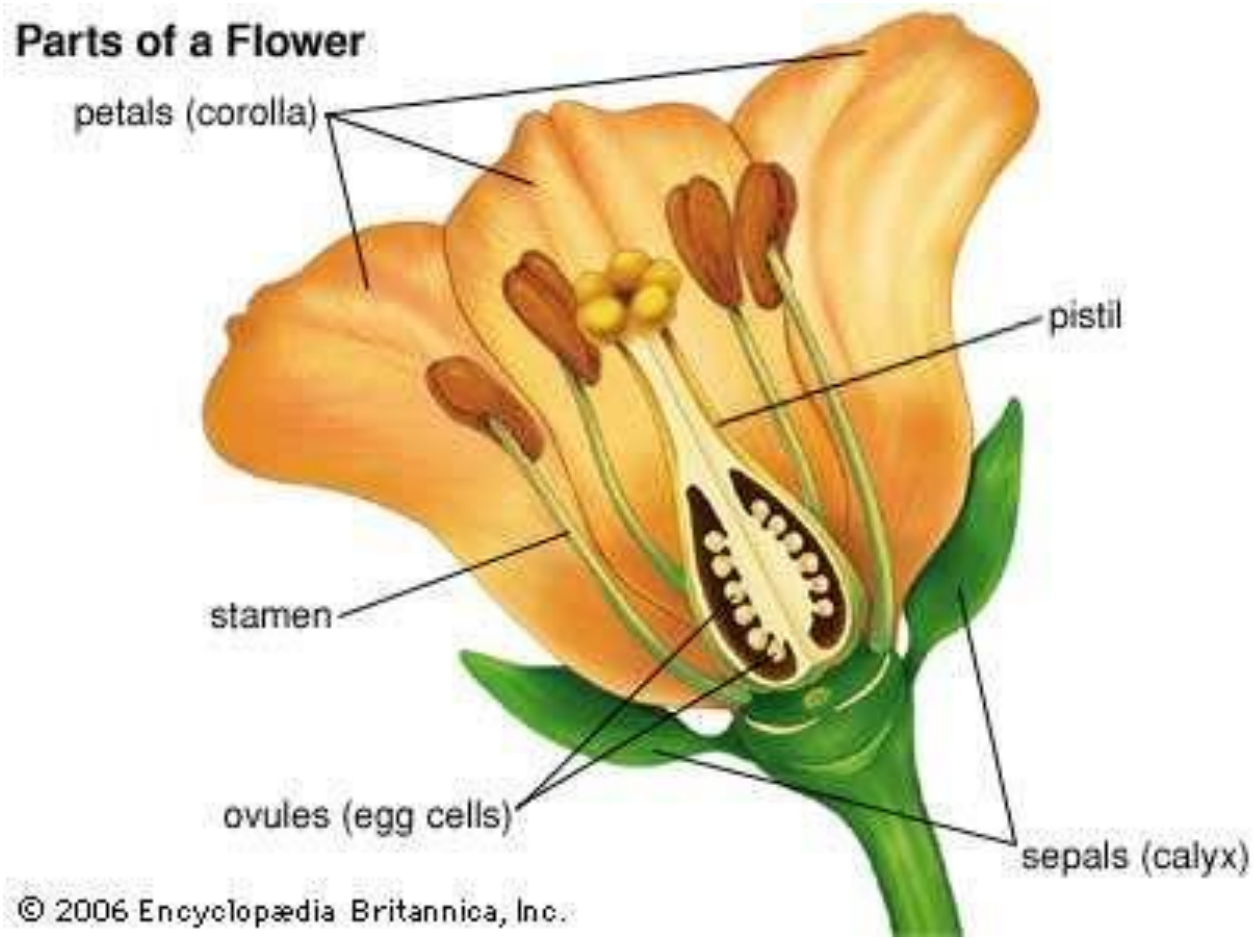
2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

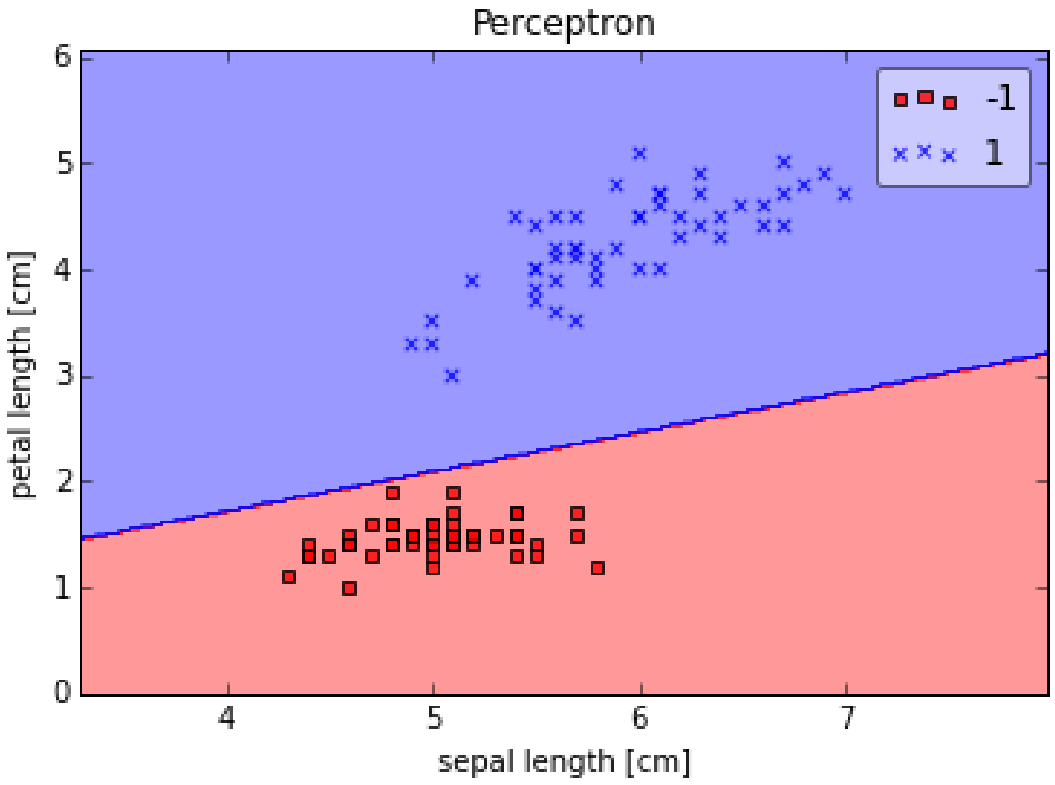


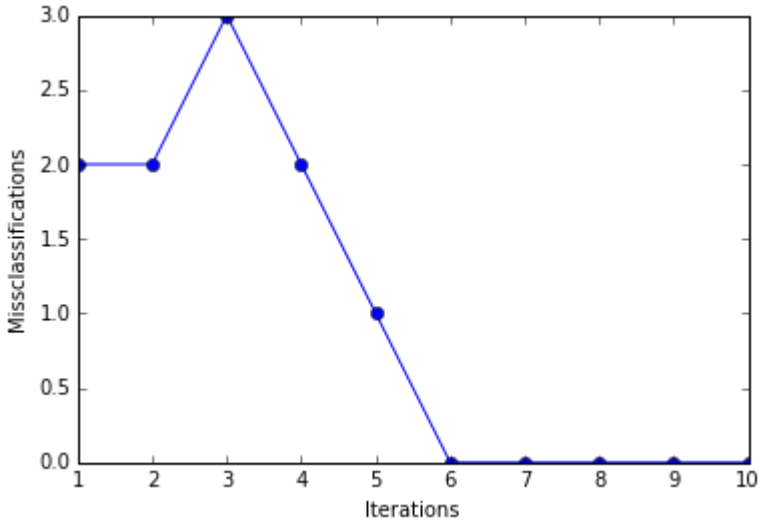
$$w_j = w_j + \eta \frac{1}{m} \sum_{i=1}^{m} (t - y)\, x_j$$

Iris versicolor

Iris setosa

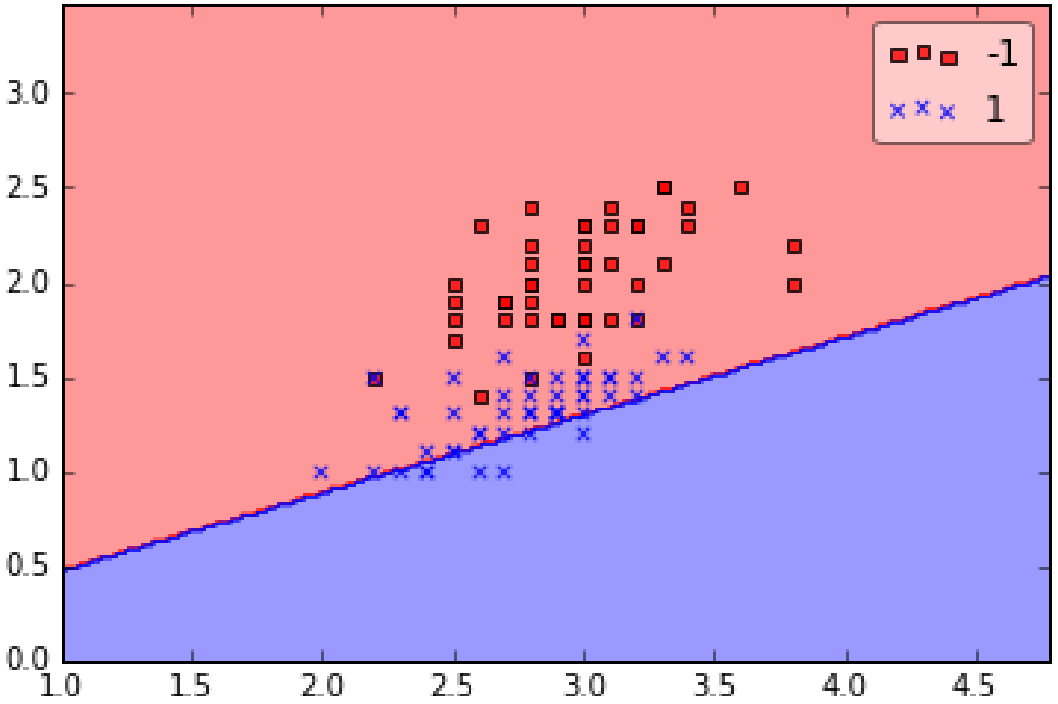

Parts of a Flower
petals (corolla)
pistil
stamen
ovules (egg cells)
sepals (calyx)
© 2006 Encyclopædia Britannica, Inc.

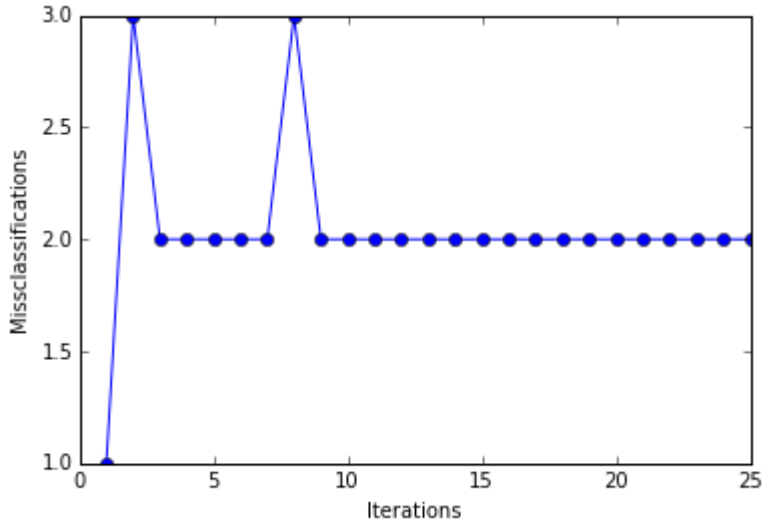

Perceptron
-1
1
petal length [cm]
sepal length [cm]
0
1
2
3
4
5
6
4
5
6
7

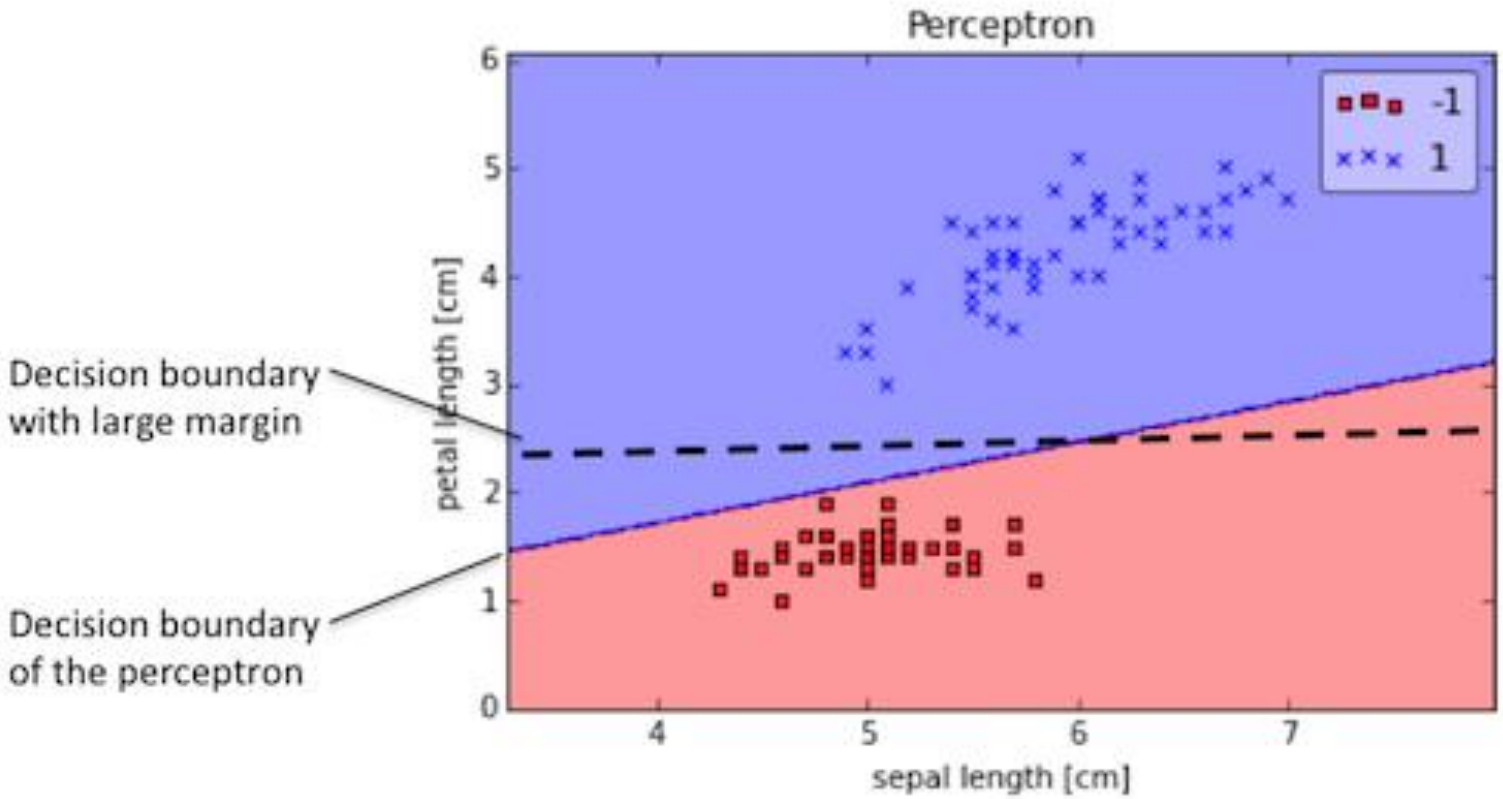

Missclassifications
Iterations
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1
2
3
4
5
6
7
8
9
10

Iris virginica

Iris versicolor


-1
1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5


Missclassifications
Iterations
3.0
2.5
2.0
1.5
1.0
0
5
10
15
20
25



Example of a large-margin decision boundary.

ADAPTIVE SWITCHING CIRCUITS

Bernard Widrow
and
Marcian E. Hoff

A. Introduction

The modern science of switching theory began with work by Shannon[1] in 1938. The field has developed rapidly since then, and at present a wealth of literature exists[2] concerning the analysis and synthesis of logical networks which might range from simple interlock systems to telephone switching systems to large-scale digital computing systems.

An example illustrating the use of switching theory is that of the design of an interlock system for the control of traffic in a railroad switch yard. The first step is the preparation of a "truth table", an exhaustive listing of all input possibilities (the positions of all incoming and outgoing trains), and what the desired system output should be (what the desired control signals should be) for each input situation. The next step is the construction of a Boolean function, and the following steps are algebraic reduction and design of the logical control system.

The design of the traffic control system is an example wherein the truth table must be followed precisely and reliably. Errors would be destructive. The design of the arithmetic element of a digital computer is another example wherein the truth table must be followed precisely.

There are other situations in which some errors are inevitable, however, and here errors are usually costly but not catastrophic. These situations call for statistically optimum switching circuits. A common performance objective is the minimization of the average number of errors. An example is that of prediction of the next bit in a correlated stochastic binary number sequence. The predictor output is to be a logical combination of a finite number of previous input sequence bits. An optimum system is a sequential switching circuit that predicts with a minimum number of errors.

Suppose that a record of the binary sequence is printed on tape and cut up into pieces (with indication of the positive direction of time preserved), say 25 bits long. Place all pieces where the most recent event is ONE in one pile, and the remainder in another pile. Delete the most recent bit on each piece of tape. If the statistical scheme could be discovered by which the pieces of tape are classified, this would lead to a prediction scheme. It is apparent that prediction is a certain kind of classification.

Assuming statistical regularity, a reasonable way to proceed might be to form a truth table, and let the data from each piece of tape be an entry in the table. It might be expected that with the data of 100 pieces of tape, a fairly good predictor could be developed. The truth table would have only 100 entries however, out of a total of 2^{24}. The "best" way to fill in the remainder of the truth table depends upon the nature of the sequence statistics and the error cost criteria. Filling in the table is a difficult and a crucial part of the problem. Even if the truth table were filled in, however, the designer would have the difficult task of realizing a logical network to satisfy a truth table with 2^{24} entries.

An approach to such problems is taken in this paper which does not require an explicit use of the truth table. The design objective is the minimization of the average number of errors, rather than a minimization of the number of logical components used. The nature of the logical elements is quite unconventional. The system design procedure is adaptive, and is based upon an iterative search process. Performance feedback is used to achieve automatic system synthesis, i.e., the selection of the "best" system from a restricted but useful class of possibilities. The designer "trains" the system to give the correct responses by "showing" it examples of inputs and respective desired outputs. The more examples "seen", the better is the system performance. System competence will be directly and quantitatively related to amount of experience.

B. A Neuron Element

In Fig. 1, a combinatorial logical circuit is shown which is a typical element in the adaptive switching circuits to be considered. This element bears some resemblance to a "neuron" model introduced by von Neuman[3], whence the name.

The binary input signals on the individual lines have values of +1 or -1, rather than the usual values of 1 or 0. Within the neuron, a linear combination of the input signals is formed. The weights are the gains $a_1, a_2, \ldots$, which could have both positive and negative values. The output signal is +1 if this weighted sum is greater than a certain threshold, and -1 otherwise. The threshold level is determined by the setting of a_0, whose input is permanently connected to a +1 source. Varying a_0 varies a constant added to the linear combination of input signals.

For fixed gain settings, each of the 2^5 possible input combinations would cause either a +1 or -1 output. Thus, all possible inputs are classified into two categories. The input-output relationship is determined by choice of the gains $a_0, \ldots a_5$. In the adaptive neuron, these gains are set during the "training" procedure.

In general, there are 2^{2^5} different input-output relationships or truth functions by which the five input variables can be mapped into the single output variable. Only a subset of these, the linearly separated truth functions[4], can be realized by all possible choices of the gains of the neuron of Fig. 1. Although this subset is not all-inclusive*, it

†Department of Electrical Engineering, Stanford University.

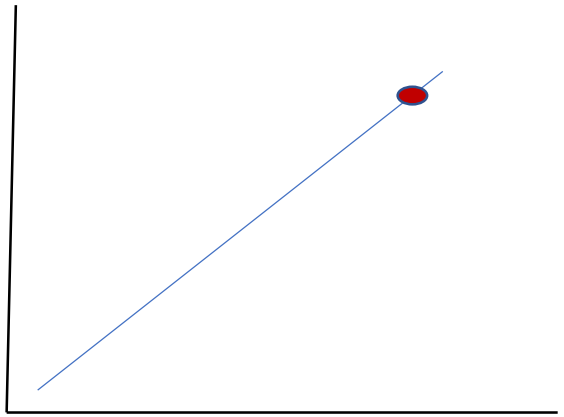
††Doctoral student in the Department of Electrical Engineering, Stanford University.

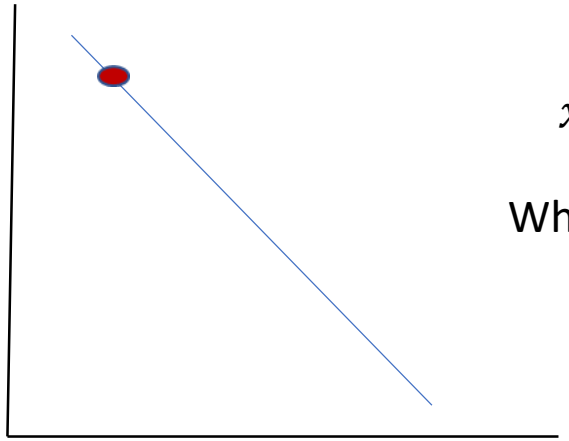
*It becomes a vanishingly small fraction of all possible switching functions as the number of inputs gets large.

Bernard Widrow

Ted Hoff

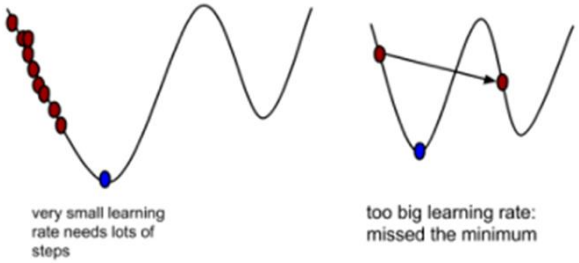
B. Widrow and M.E. Hoff, Jr., ``Adaptive Switching Circuits,'' *IRE WESCON Convention Record*, 4:96-104, August 1960.

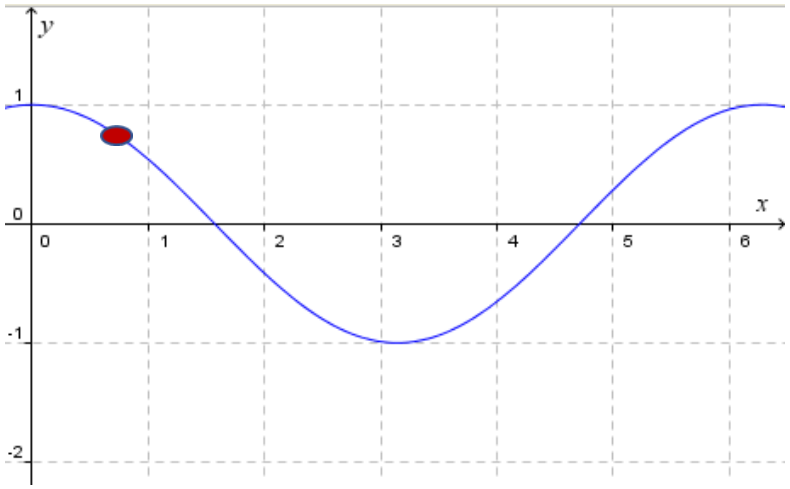




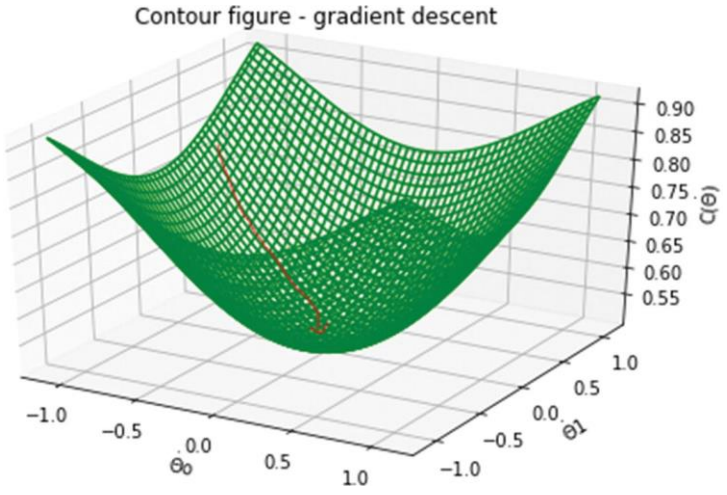
$$x = x - \alpha m$$

What will be the impact of α ?

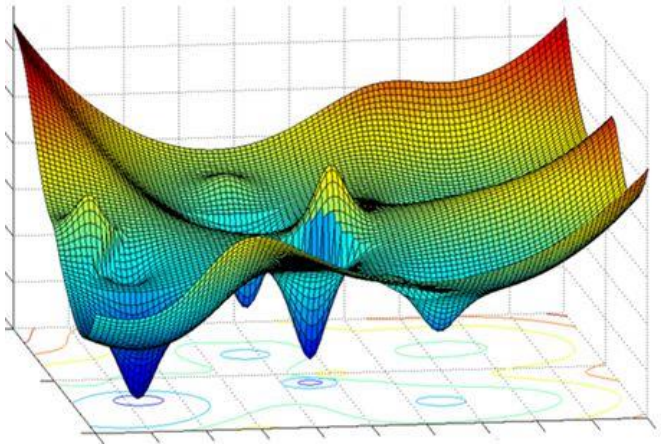




$$x = x - \alpha \frac{dy}{dx}$$

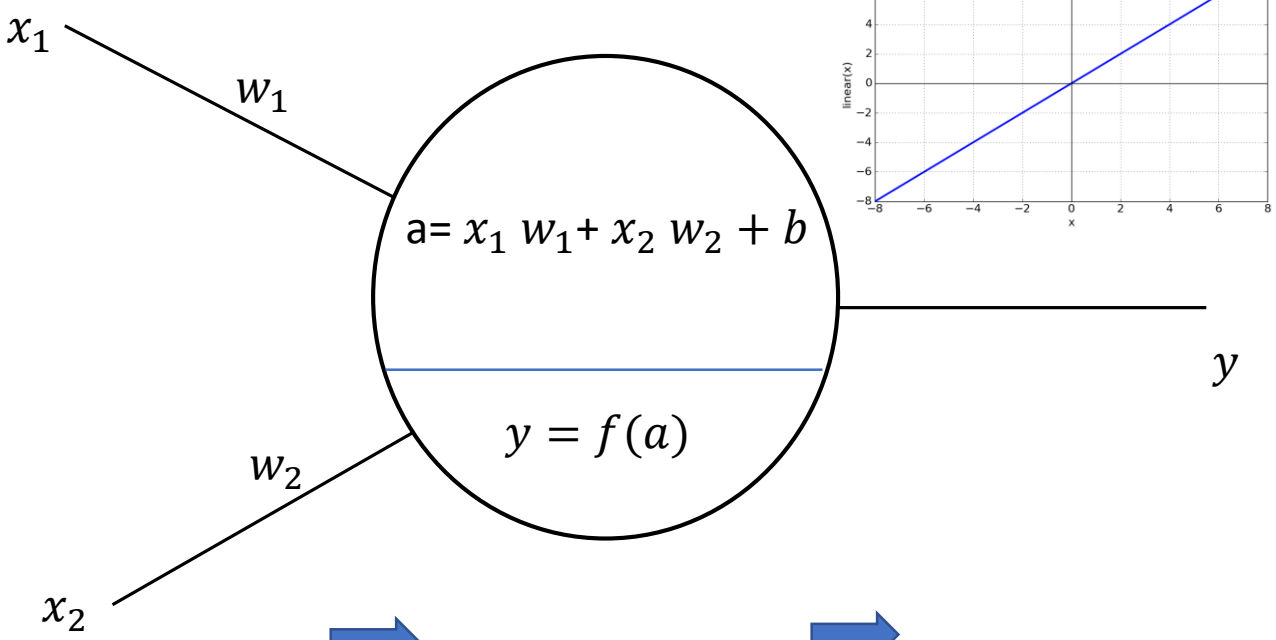


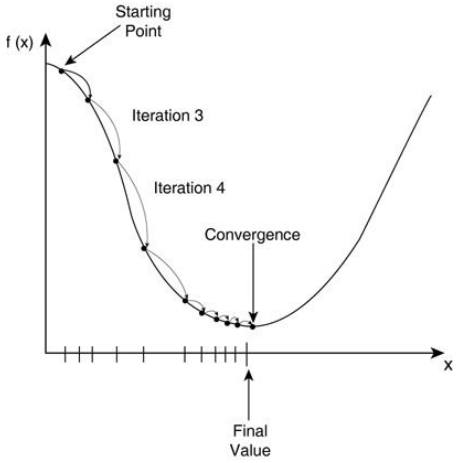
$$x_1 = x_1 - \alpha \nabla Y \qquad x_2 = x_2 - \alpha \nabla Y$$

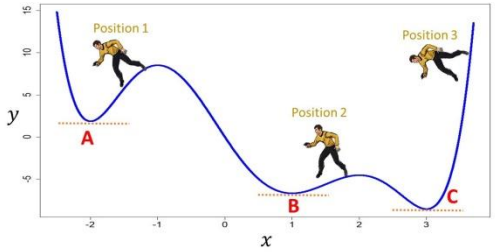


$$\nabla Y = \left\langle \frac{dy}{dx_1}, \frac{dy}{dx_2} \right\rangle$$

Single neuron with linear activation function







$$E = \frac{1}{2}(t-y)^2$$

$y = a$

$t =$ target output

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w1}$$

$$\frac{\partial E}{\partial y} = \frac{\partial(\frac{1}{2}(t-y)^2)}{\partial y} = -(t-y)$$

$$\frac{\partial y}{\partial a} = 1$$

$$\frac{\partial a}{\partial w1} = \frac{\partial(x_1 w_1 + x_2\ w_2 + b)}{\partial w1} = x_1$$

$$\frac{\partial E}{\partial w1} = -(t-y)\ x_1$$

$$w_1 = w_1 + \alpha(t-y)\ x_1$$

$$w_2 = w_2 + \alpha(t-y)\ x_2$$

$$b = b + \alpha(t-y)$$

$$w_{j,i} = w_{j,i} + \alpha\ (t_j - y_j)\ x_i$$

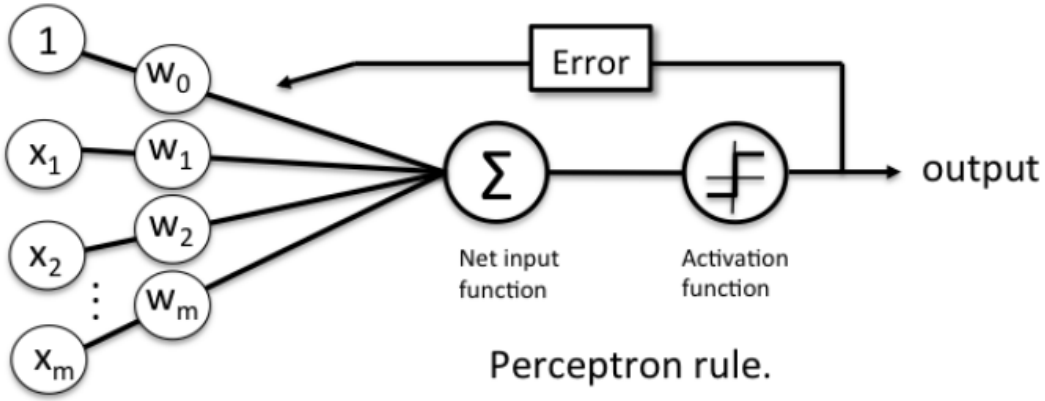
$$W = W + \alpha\ (t-y)\ X$$

$$a = XW^T$$

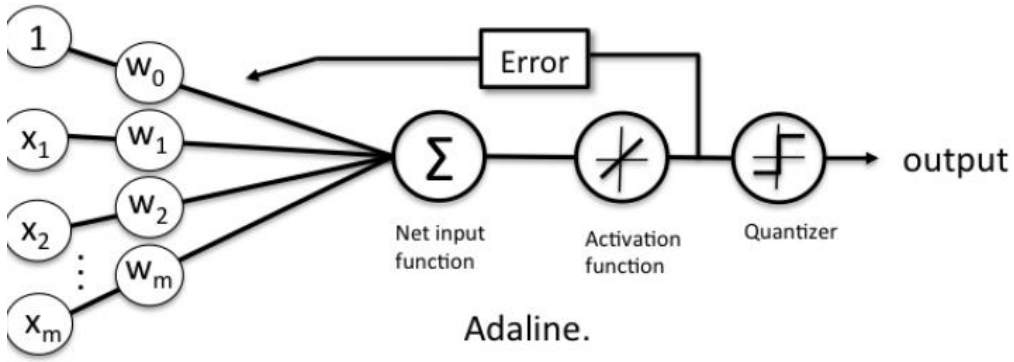
$$X = [x_1, x_2]$$

$$W = [w_1, w_2]$$

$w_{j,i}$= weight of i^{th} input of j^{th} neuron



Perceptron rule.



Adaline.

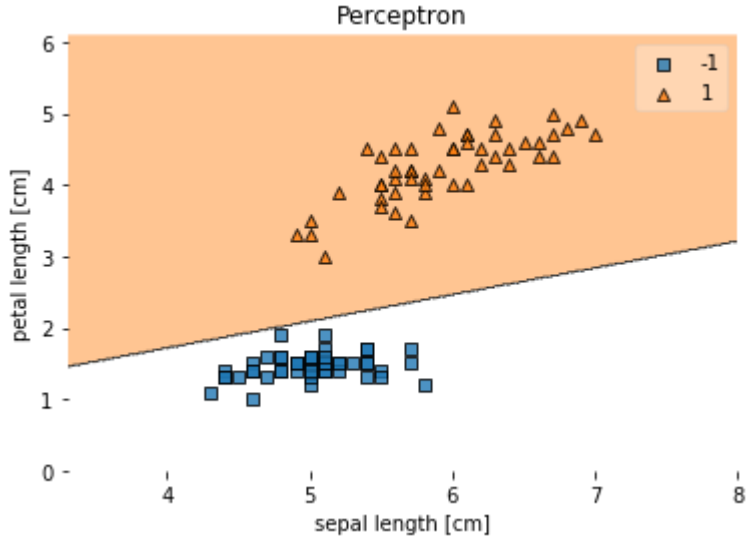
What Adaline and the Perceptron have in common

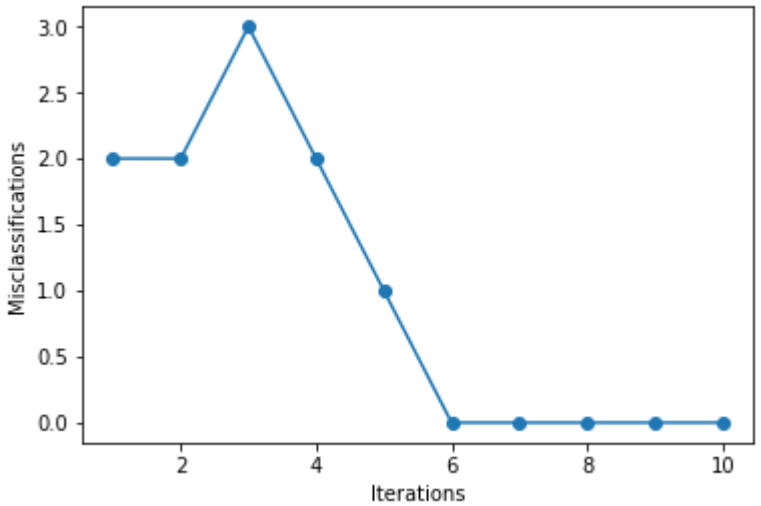
- they are classifiers for binary classification
- both have a linear decision boundary
- both can learn iteratively, sample by sample (the Perceptron naturally, and Adaline via stochastic gradient descent)
- both use a threshold function

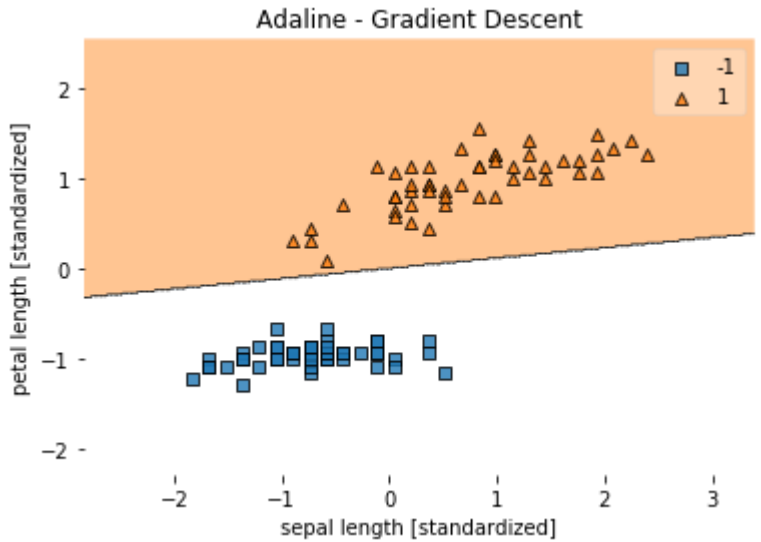
The differences between the Perceptron and Adaline

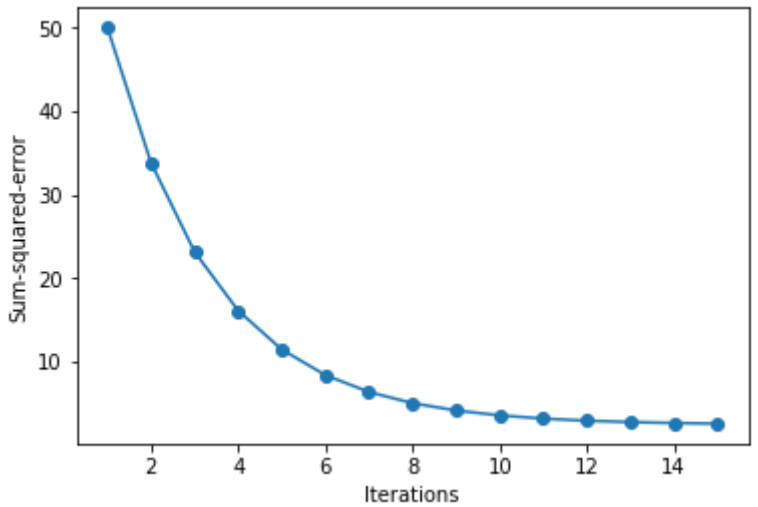
- the Perceptron uses the class labels to learn model coefficients
- Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more "powerful" since it tells us by "how much" we were right or wrong

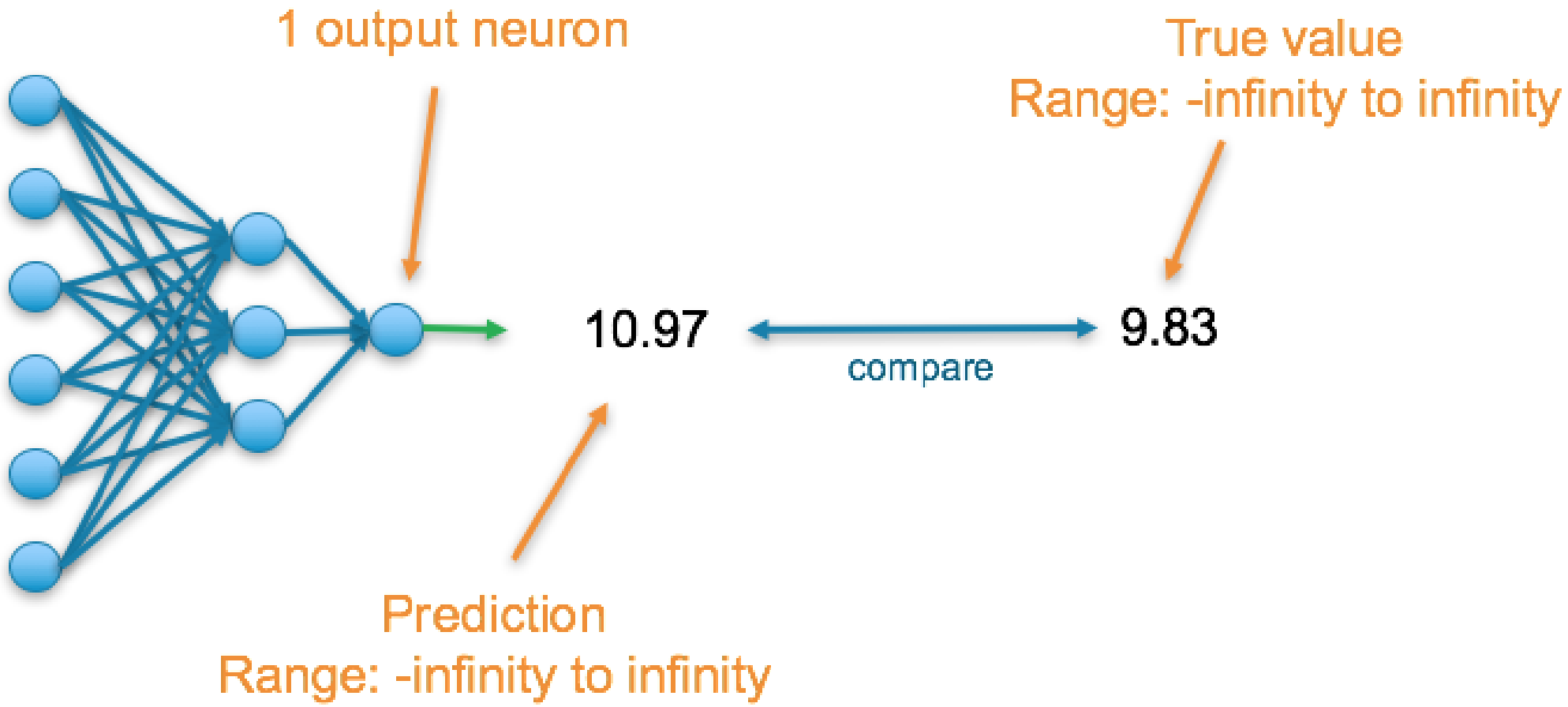
https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html


Perceptron
-1
1
petal length [cm]
sepal length [cm]


Misclassifications
Iterations

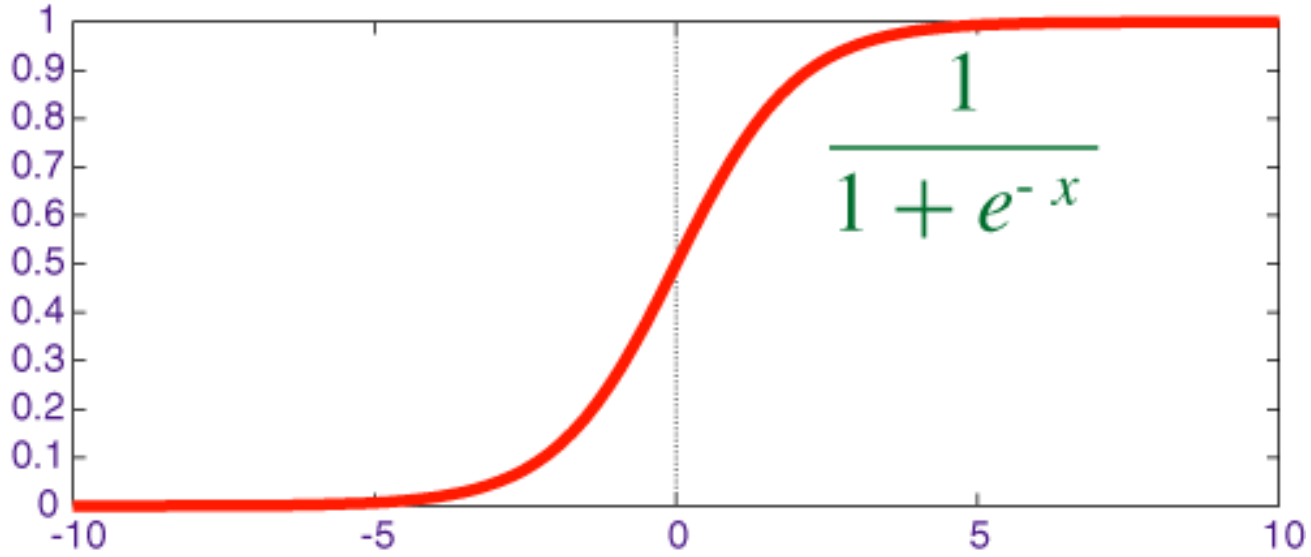

Adaline - Gradient Descent
-1
1
petal length [standardized]
sepal length [standardized]


Sum-squared-error
Iterations



$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where $\hat{y}$ is the predicted value and y is the true value



Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

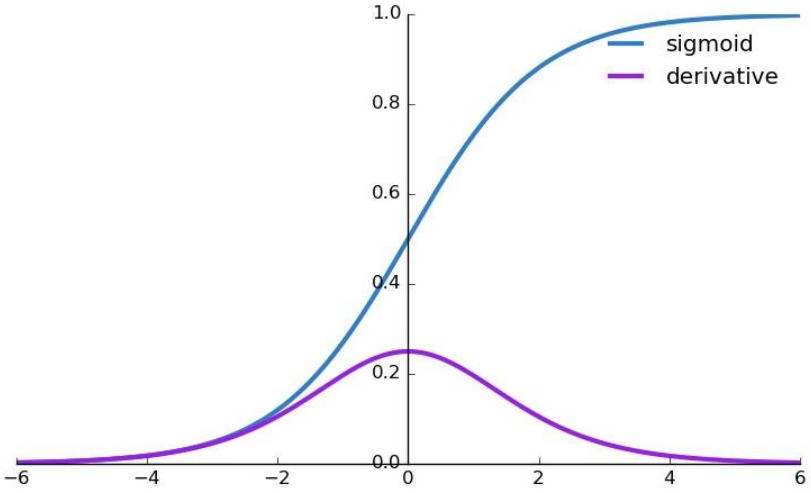
$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$

A **sigmoid function** is a mathematical function having a characteristic "S"-shaped curve or **sigmoid curve.**

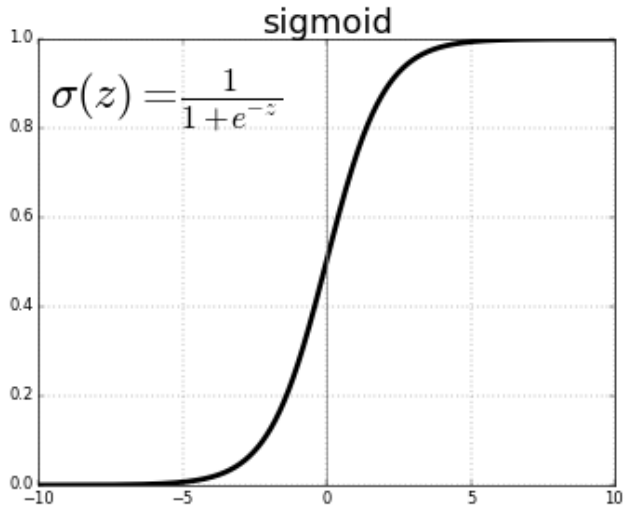
A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point.

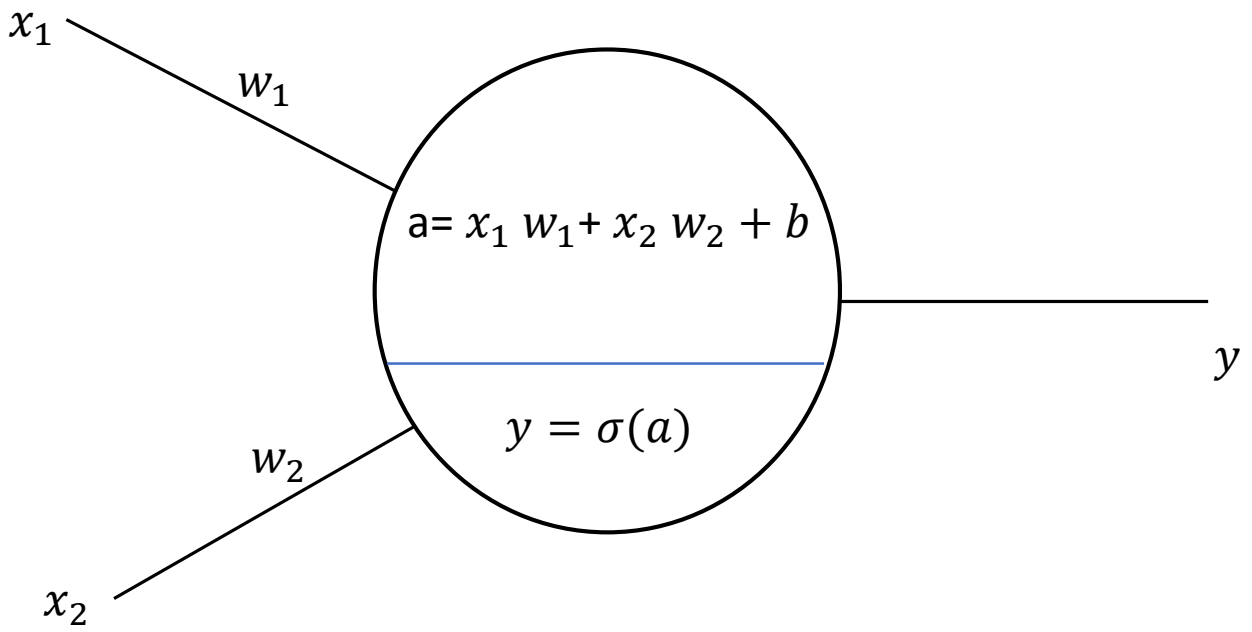
Without a ***non-linear*** activation function in the network, a NN, no matter how many layers it had, would behave just like a single-layer perceptron, because summing these layers would give you just another linear function.



https://en.wikipedia.org/wiki/Sigmoid_function
https://stackoverflow.com/questions/9782071/why-must-a-nonlinear-activation-function-be-used-in-a-backpropagation-neural-net

Sigmoid activation function





$$E = \frac{1}{2}(t-y)^2$$

$$y = \sigma(a)$$

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

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w1}$$

$$\frac{\partial E}{\partial y} = \frac{\partial(\frac{1}{2}(t-y)^2)}{\partial y} = -(t-y)$$

$$\frac{\partial y}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)\,(1-\sigma(a)) = y\,(1-y)$$

$$\frac{\partial a}{\partial w1} = \frac{\partial(x_1 w_1 + x_2 w_2 + b)}{\partial w1} = x_1$$

$$\frac{\partial E}{\partial w1} = -(t-y)y\,(1-y)\,x_1$$

$$w_1 = w_1 + \alpha\,(t-y)y\,(1-y)\,x_1$$

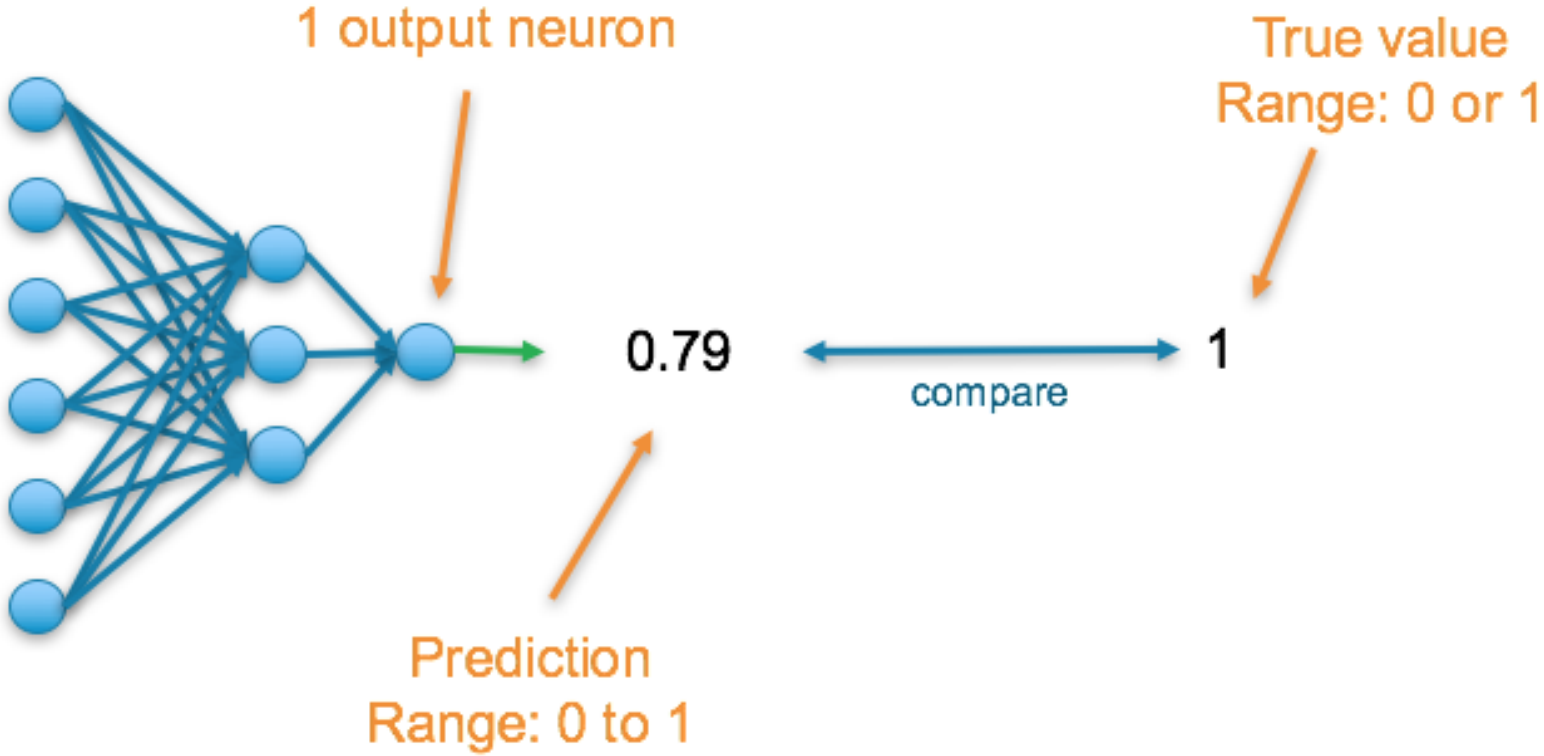
$$w_2 = w_2 + \alpha\,(t-y)y\,(1-y)\,x_2$$

$$b = b + \alpha\,(t-y)y\,(1-y)$$

$$w_{j,i} = w_{j,i} + \alpha\,(t_j - y_j)y_j(1-y_j)\,x_i$$

$$w_{j,i} = w_{j,i} + \alpha\,\delta_j\,x_i$$

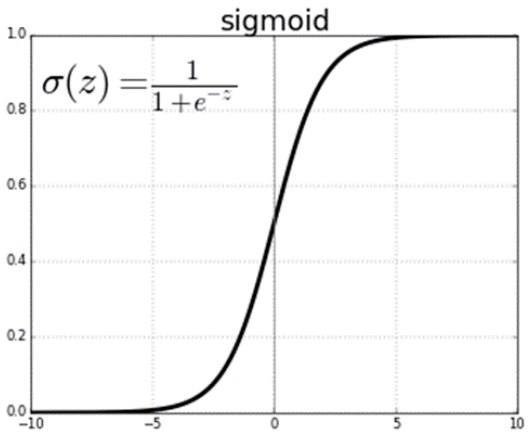
$$\delta_j = (t_j - y_j)y_j(1-y_j)$$

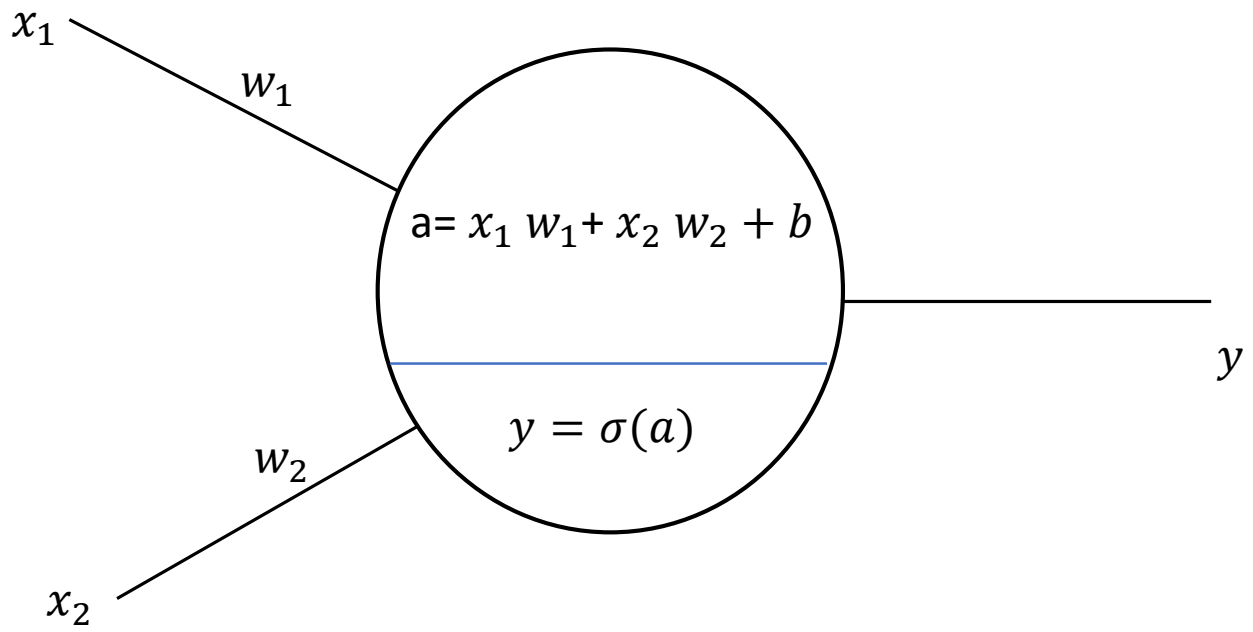


Binary cross entropy = $-(y \log(\hat{y}) + (1 - y)\log(1 - \hat{y}))$

Where $\hat{y}$ is the predicted value and y is the true value

Sigmoid activation function





$$E = -(t\log(y) + (1-t)\log(1-y))$$

$$y = \sigma(a)$$

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

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w1}$$

$$\frac{\partial E}{\partial y} = -\frac{\partial(t\log(y)+(1-t)\log(1-y))}{\partial y}$$

$$\frac{\partial E}{\partial y} = -\left(\frac{t}{y} - \frac{(1-t)}{(1-y)}\right) = \frac{y-t}{y(1-y)}$$

$$\frac{\partial y}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)\,(1-\sigma(a)) = y\,(1-y)$$

$$\frac{\partial a}{\partial w1} = \frac{\partial(x_1 w_1 + x_2 w_2 + b)}{\partial w1} = x_1$$

$$\frac{\partial E}{\partial w1} = (y-t)x_1$$

$$w_1 = w_1 + \alpha\,(t-y)x_1$$

$$w_2 = w_2 + \alpha\,(t-y)x_2$$

$$b = b + \alpha\,(t-y))$$

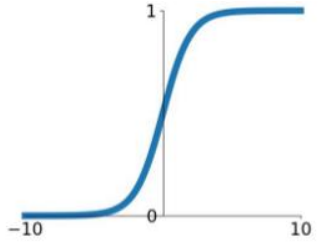
$$w_{j,i} = w_{j,i} + \alpha\,(t_j - y_j)\,x_i$$

$$w_{j,i} = w_{j,i} + \alpha\,\delta_j\,x_i$$

$$\delta_j = (t_j - y_j)$$

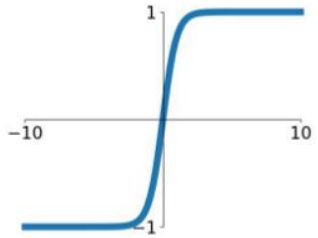
Sigmoid

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



tanh

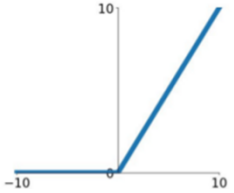
$$\tanh(x) \quad \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

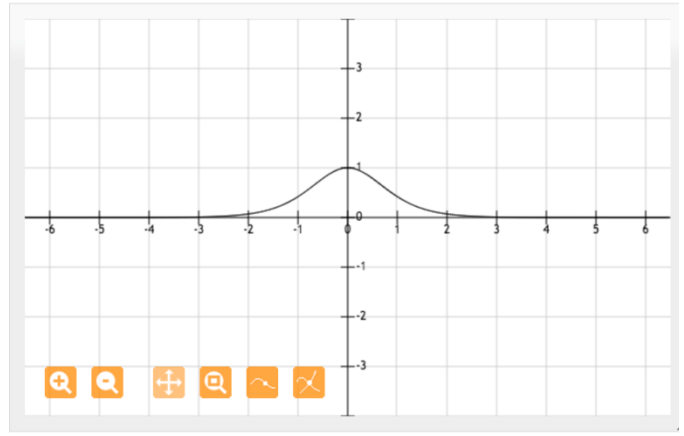


$$\tanh = 2\sigma(2x) - 1$$

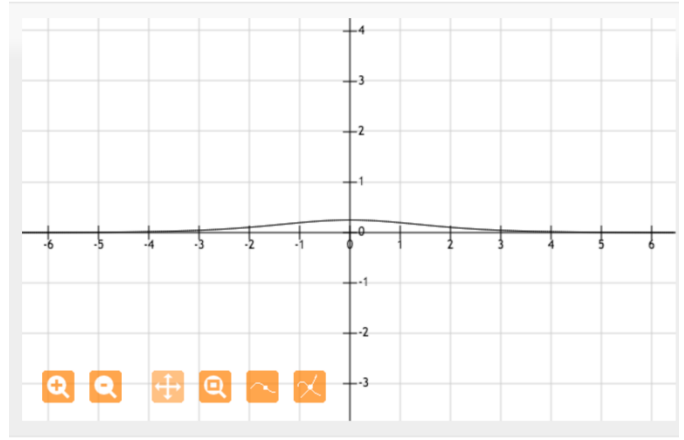
ReLU

$$\max(0, x)$$





This is the derivative of the tanh function. For input between [-1,1], we have derivative between [0.42, 1].



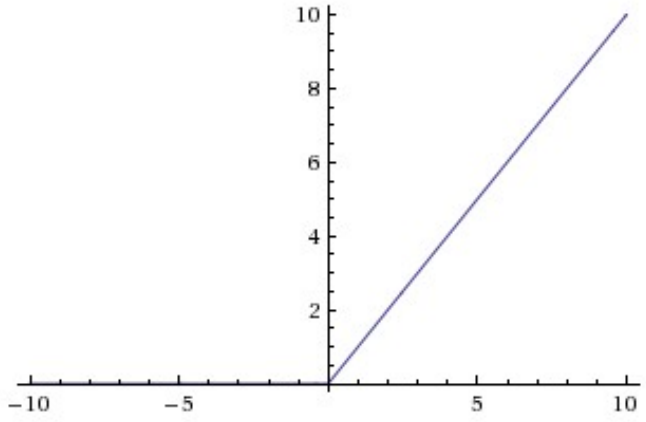
This is the derivative of the standard sigmoid function f(x)= 1/(1+exp(-x)). For input between [0,1], we have derivative between [0.20, 0.25].

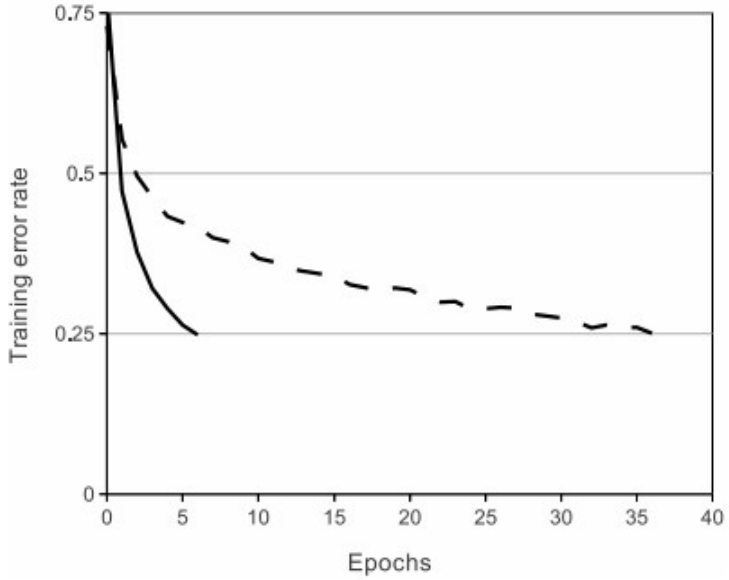
Convergence is usually faster if the average of each input variable over the training set is close to zero.

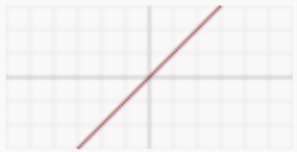
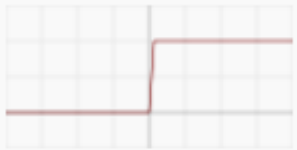
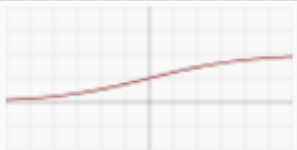
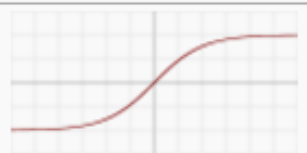
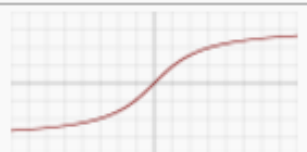
When all of the components of an input vector are positive, all of the updates of weights that feed into a node will be the same sign

The main advantage provided by the tanh function is that it produces zero centered output.

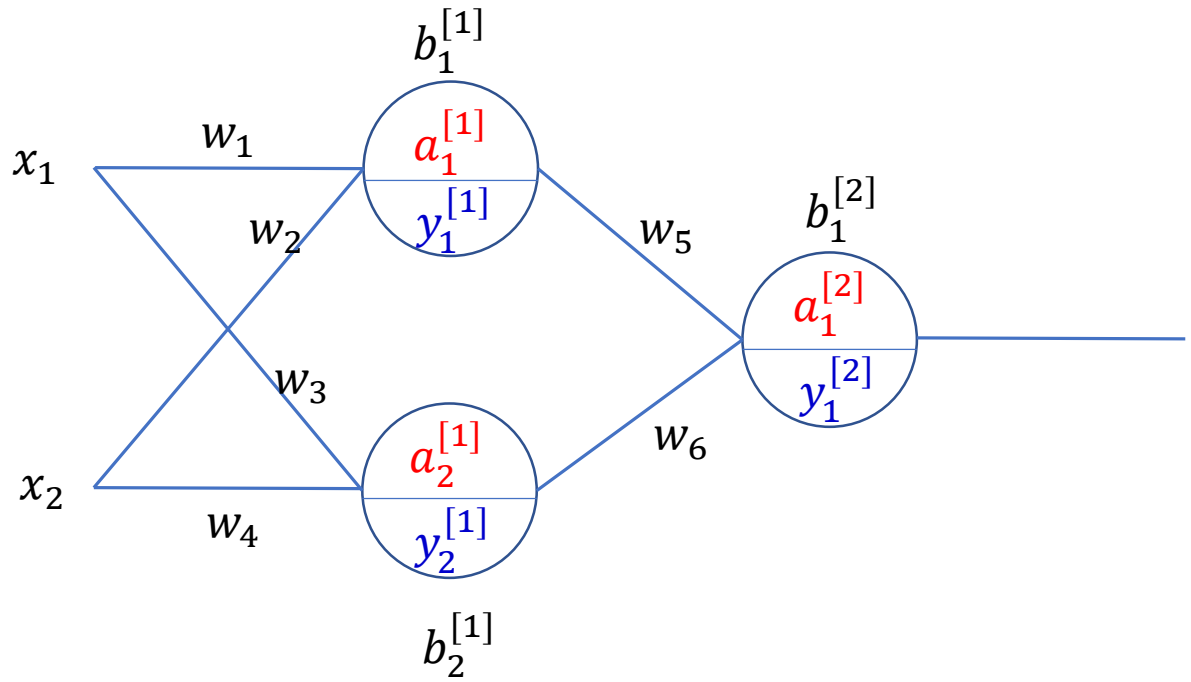
- ReLU activation function was first introduced to a dynamical network by Hahnloser et al. in 2000 with strong biological motivations and mathematical justifications.
- It has been demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely used activation functions prior to 2011.
- It’s sparsely activated.
- ReLU neurons can sometimes be pushed into states in which they become inactive for essentially all inputs.
- In this state, no gradients flow backward through the neuron, and so the neuron becomes stuck in a perpetually inactive state and "dies". This is a form of the vanishing gradient problem.





Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

ANN with multiple layers



$a_1^{[1]} = x_1\, w_1 + x_2\, w_2 + b_1^{[1]}$

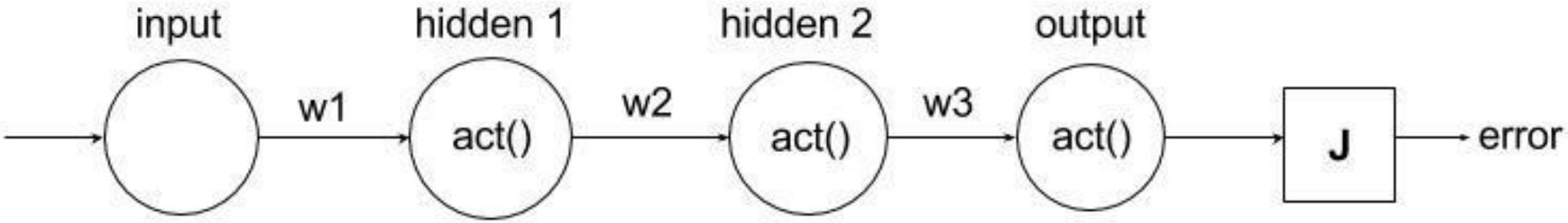
$y_1^{[1]} = \sigma(a_1^{[1]})$

$a_2^{[1]} = x_1\, w_3 + x_2\, w_4 + b_2^{[1]}$

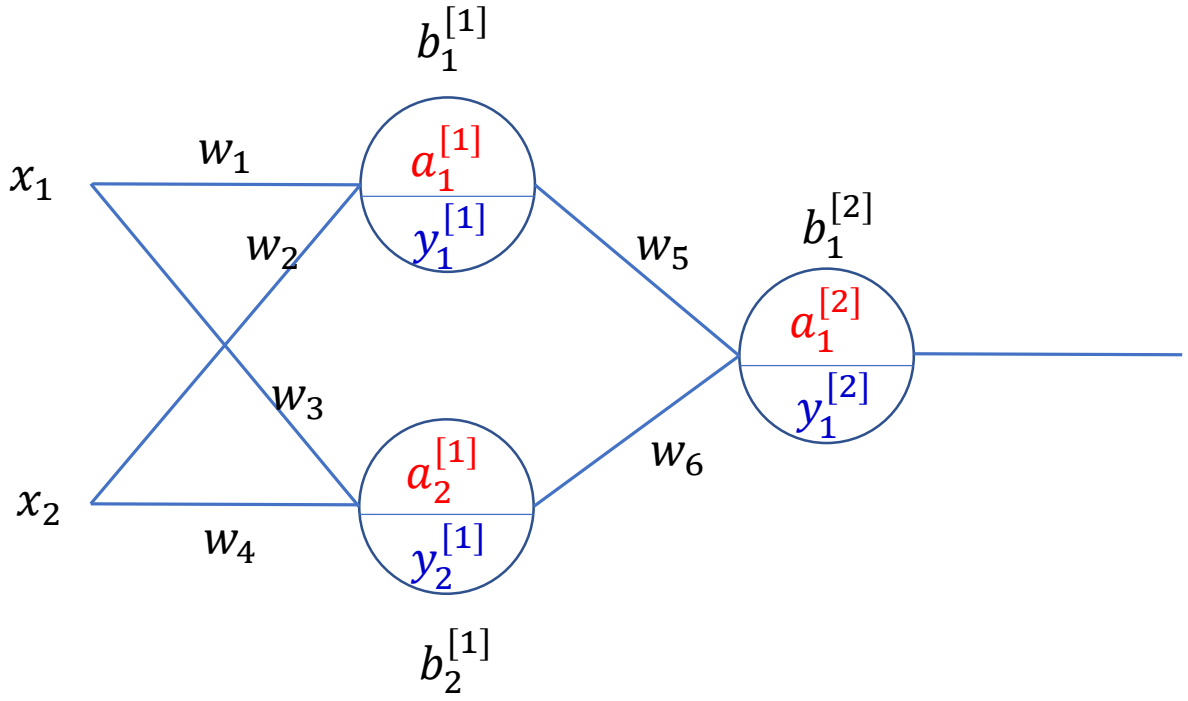
$y_2^{[1]} = \sigma(a_2^{[1]})$

$a_1^{[2]} = y_1^{[1]}\, w_5 + y_2^{[1]}\, w_6 + b_1^{[2]}$

$y_1^{[2]} = \sigma(a_1^{[2]})$



$$\frac{\partial(error)}{\partial w_1} = \frac{\partial(error)}{\partial(output)} * \frac{\partial(output)}{\partial(hidden\ 2)} * \frac{\partial(hidden\ 2)}{\partial(hidden\ 1)} * \frac{\partial(hidden\ 1)}{\partial w_1}$$



$$w_5 = w_5 - \alpha \frac{\partial E}{\partial w5}$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$E = \frac{1}{2}(t - y_1^{[2]})^2$$

$$y_1^{[2]} = \sigma(a_1^{[2]})$$

$$\frac{\partial E}{\partial w5} = \frac{\partial E}{\partial y_1^{[2]}} \frac{\partial y_1^{[2]}}{\partial a_1^{[2]}} \frac{\partial a_1^{[2]}}{\partial w5}$$

$$\frac{\partial E}{\partial y_1^{[2]}} = \frac{\partial (\frac{1}{2}(t - y_1^{[2]})^2)}{\partial y_1^{[2]}} = -(t - y_1^{[2]})$$

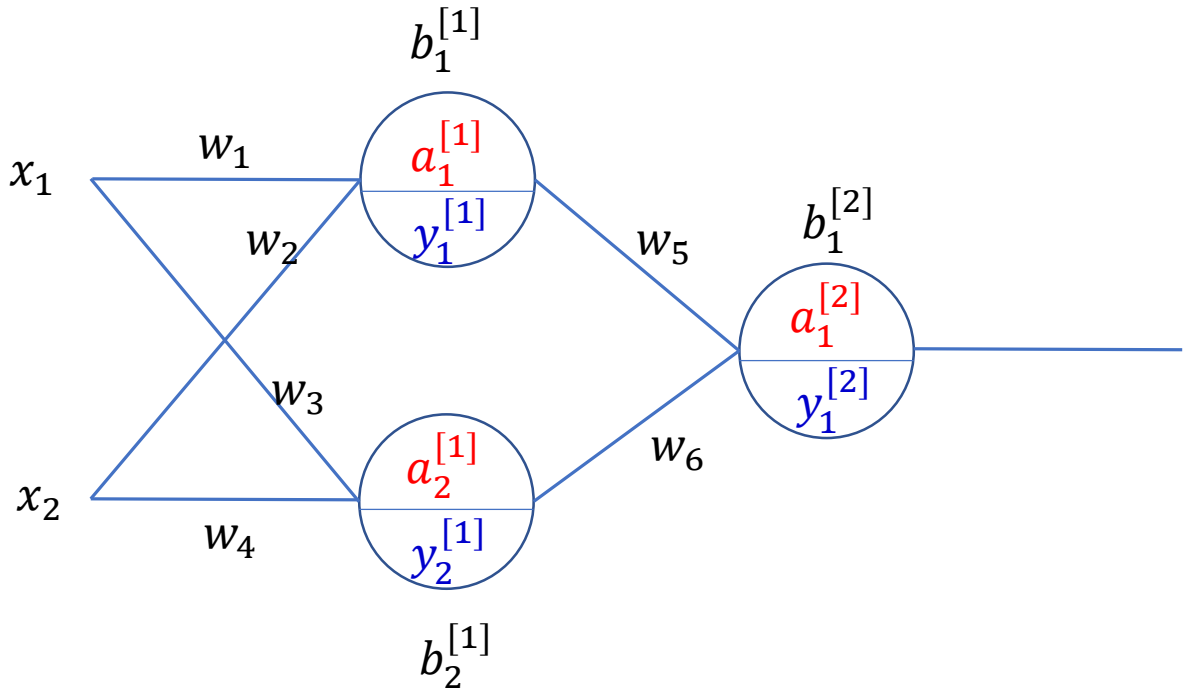
$$\frac{\partial y_1^{[2]}}{\partial a_1^{[2]}} = \frac{\partial \sigma(a_1^{[2]})}{\partial a_1^{[2]}} = \sigma(a_1^{[2]})\,(1 - \sigma(a_1^{[2]})) = y_1^{[2]}(1 - y_1^{[2]})$$

$$\frac{\partial a_1^{[2]}}{\partial w5} = \frac{\partial (y_1^{[1]} w_5 + y_2^{[1]} w_6 + b_1^{[2]})}{\partial w5} = y_1^{[1]}$$

$$\frac{\partial E}{\partial w5} = -(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]})\, y_1^{[1]}$$

$$w_5 = w_5 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]})\, y_1^{[1]}$$

$$w_5 = w_5 + \alpha\, \delta_1^{[2]} y_1^{[1]}$$



$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$E = \frac{1}{2}(t - y_1^{[2]})^2$$

$$y_1^{[2]} = \sigma(a_1^{[2]})$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y_1^{[2]}} \frac{\partial y_1^{[2]}}{\partial y_1^{[1]}} \frac{\partial y_1^{[1]}}{\partial w1}$$

$$\frac{\partial E}{\partial y_1^{[2]}} = \frac{\partial (\frac{1}{2}(t - y_1^{[2]})^2)}{\partial y_1^{[2]}} = -(t - y_1^{[2]})$$

$$\frac{\partial y_1^{[2]}}{\partial y_1^{[1]}} = \frac{\partial y_1^{[2]}}{\partial a_1^{[2]}} \frac{\partial a_1^{[2]}}{\partial y_1^{[1]}}$$

$$\frac{\partial y_1^{[2]}}{\partial a_1^{[2]}} = y_1^{[2]}(1 - y_1^{[2]})$$

$$\frac{\partial a_1^{[2]}}{\partial y_1^{[1]}} = w_5$$

$$\frac{\partial y_1^{[2]}}{\partial y_1^{[1]}} = y_1^{[2]}(1 - y_1^{[2]})\, w_5$$

$$\frac{\partial y_1^{[1]}}{\partial w1} = \frac{\partial y_1^{[1]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial w1}$$

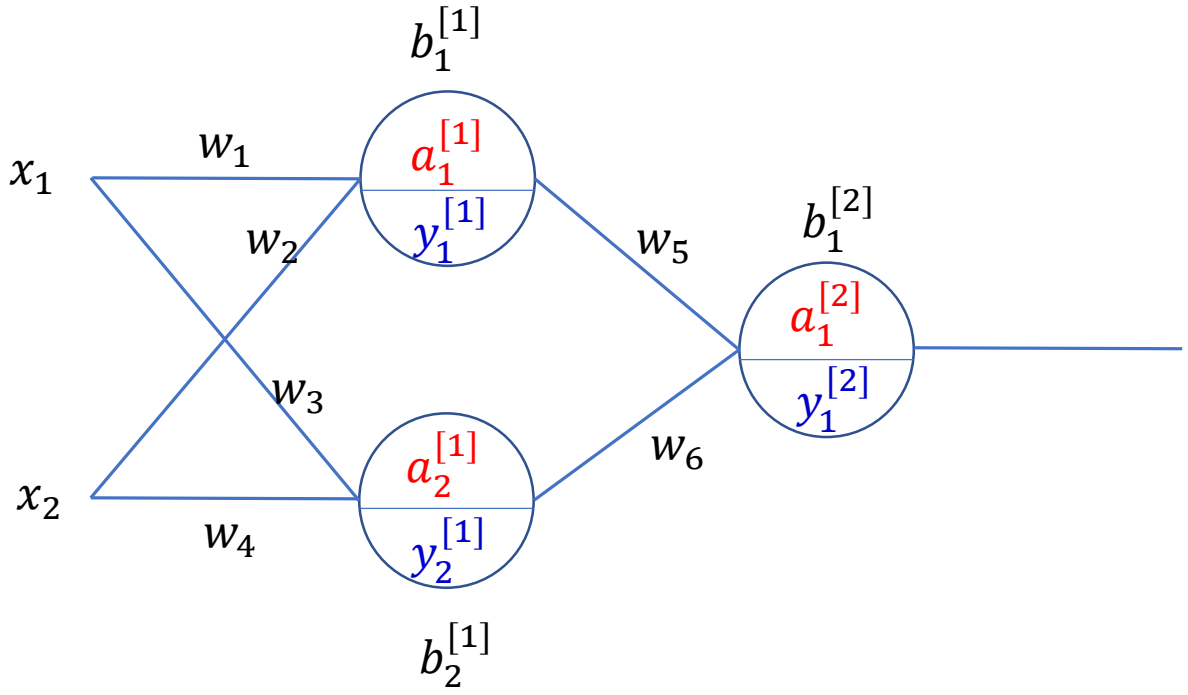
$$\frac{\partial y_1^{[1]}}{\partial a_1^{[1]}} = y_1^{[1]}(1 - y_1^{[1]})$$

$$\frac{\partial a_1^{[1]}}{\partial w1} = \frac{\partial (x_1 w_1 + x_2 w_2 + b_1^{[1]})}{\partial w1} = x_1$$

$$\frac{\partial y_1^{[1]}}{\partial w1} = y_1^{[1]}(1 - y_1^{[1]})\, x_1$$

$$\frac{\partial E}{\partial w1} = -(t - y_1^{[2]}) y_1^{[2]}(1 - y_1^{[2]}) w_5 y_1^{[1]}(1 - y_1^{[1]}) x_1$$

$$w_1 = w_1 + \alpha\,(t - y_1^{[2]}) y_1^{[2]}(1 - y_1^{[2]}) w_5 y_1^{[1]}(1 - y_1^{[1]}) x_1$$



$$w_5 = w_5 + \alpha\,\delta^{[2]} y_1^{[1]}$$

$$w_6 = w_6 + \alpha\,\delta^{[2]} y_2^{[1]}$$

$$\delta^{[2]} = (t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]})$$

$$w_1 = w_1 + \alpha\,\delta^{[1]} x_1$$

$$\delta^{[1]} = \delta^{[2]} w_5\, y_1^{[1]} (1 - y_1^{[1]})$$

$$w_{j,l} = w_{j,l} + \alpha\,\delta^{[l]} y_{l-1}$$

$$w_5 = w_5 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]})\, y_1^{[1]}$$

$$w_6 = w_6 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]})\, y_2^{[1]}$$

$$w_1 = w_1 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]}) w_5 y_1^{[1]} (1 - y_1^{[1]}) x_1$$

$$w_2 = w_2 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]}) w_5 y_1^{[1]} (1 - y_1^{[1]}) x_2$$

$$w_3 = w_3 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]}) w_6 y_2^{[1]} (1 - y_2^{[1]}) x_1$$

$$w_4 = w_4 + \alpha\,(t - y_1^{[2]}) y_1^{[2]} (1 - y_1^{[2]}) w_6 y_2^{[1]} (1 - y_2^{[1]}) x_2$$

$$\delta^{[i]} = \begin{cases} (t - y_j^{[i]}) y_j^{[i]} (1 - y_j^{[i]}) & \text{if j is an output neuron} \\ \left(\sum_{l \in L} \delta^{[l]} w_{jl} \right) y_j^{[l]} (1 - y_j^{[l]}) & \text{if j is an inner neuron} \end{cases}$$

Backpropogation algorithm

Phase 1: propagation

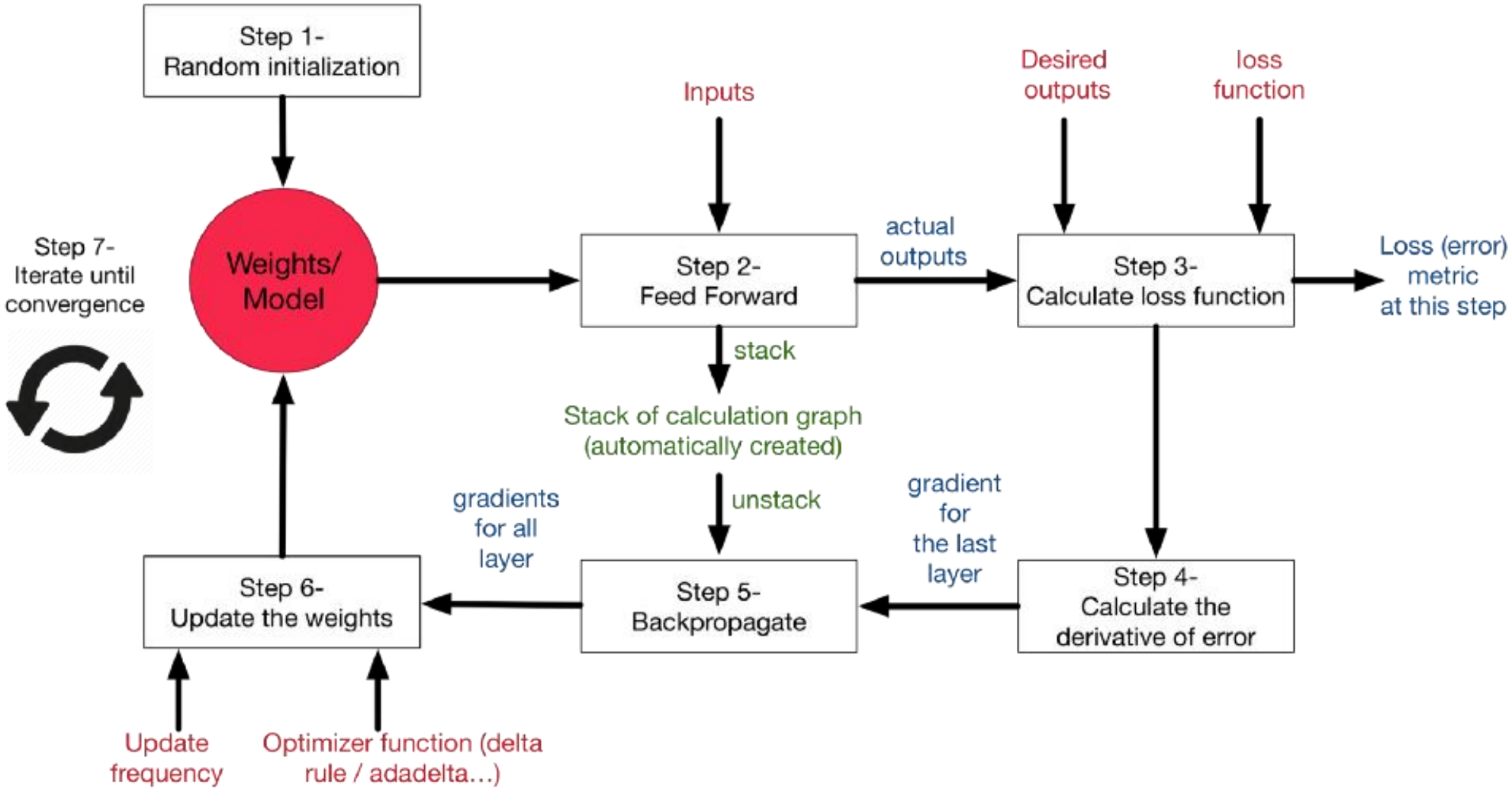
Each propagation involves the following steps:

1. Propagation forward through the network to generate the output value(s)
2. Calculation of the cost (error term)
3. Propagation of the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: weight update

For each weight, the following steps must be followed:

1. The weight's output delta and input activation are multiplied to find the gradient of the weight.
2. A ratio (percentage) of the weight's gradient is subtracted from the weight.



https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

Historical and bibliographical remarks

Modern version of BP (also called automatic differentiation) was first published in 1970 by Finnish master student Seppo Linnainmaa.

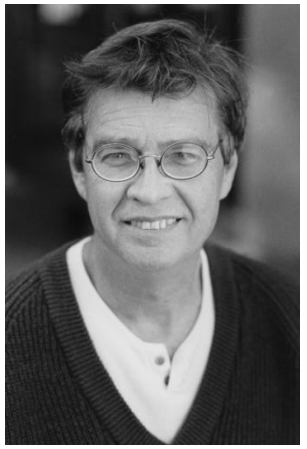
First NN-specific application of efficient BP was described by Werbos (1982).

Related work was published several years later (Parker, 1985; LeCun, 1985).

A paper of 1986 significantly contributed to the popularisation of BP for NNs (Rumelhart et al., 1986), experimentally demonstrating the emergence of useful internal representations in hidden layers.

Seppo Linnainmaa 

Paul Werbos 

David Rumelhart 

http://people.idsia.ch/~juergen/who-invented-backpropagation.html

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

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

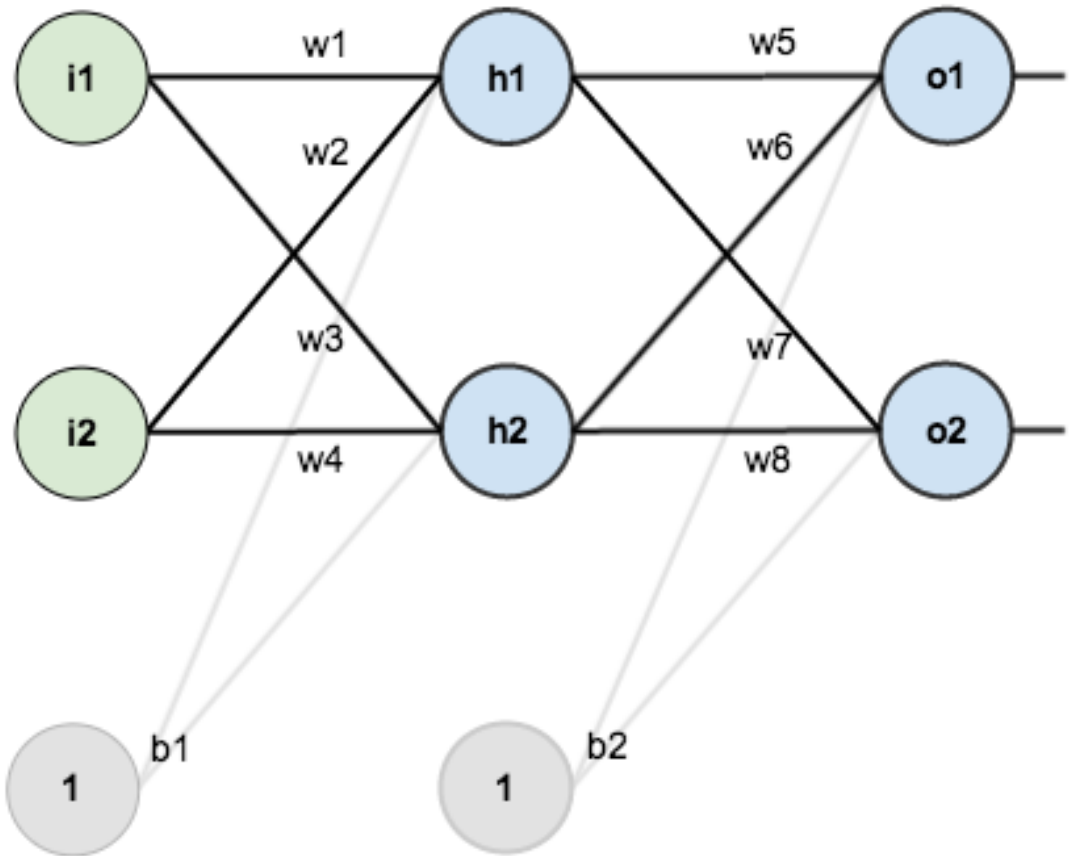
Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

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

$$y_j = \frac{1}{1+e^{-x_j}} \tag{2}$$

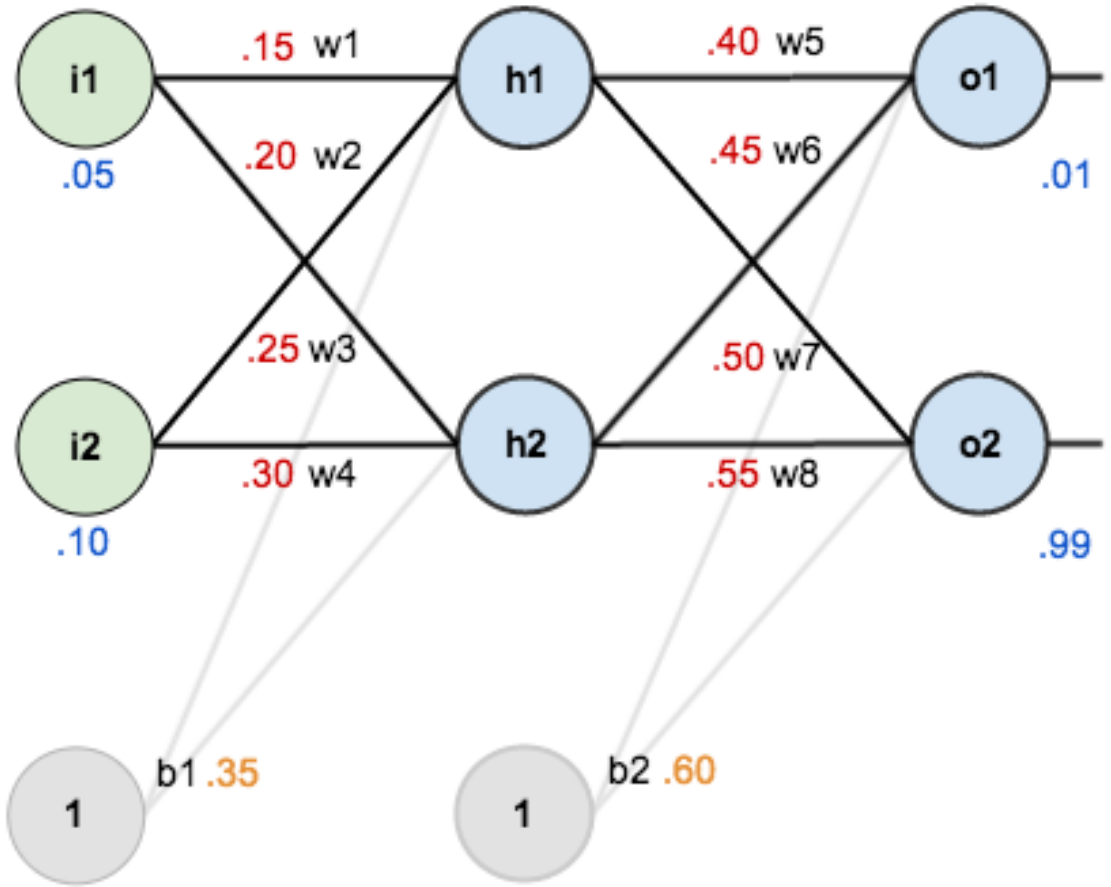
† To whom correspondence should be addressed.

Backpropagation Example



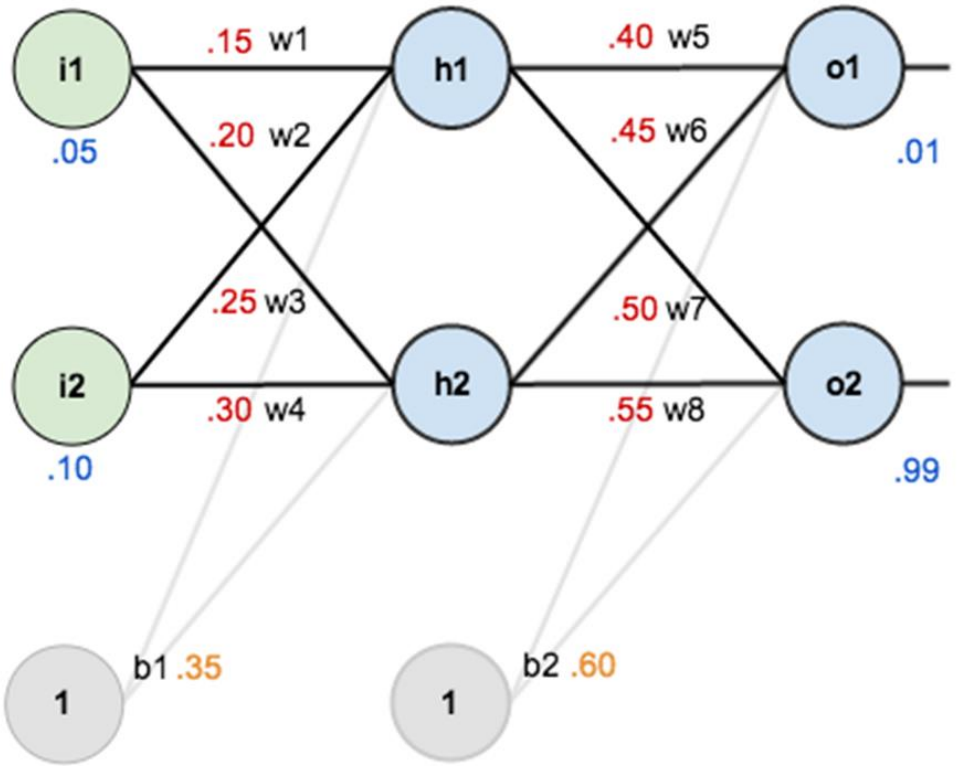
i – input layer
h – hidden layer
o – output layer
w – weights
b - bias
Activation function : sigmoid
Loss function : mse
Learning rate : 0.5

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



Here are the initial weights, the biases, and training inputs/outputs:

Given inputs are 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.



The Forward Pass

Weighted sum in h1 $net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$

$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$

Output of h1 $out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$

Output of o1 $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$

Output of o1 $out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$

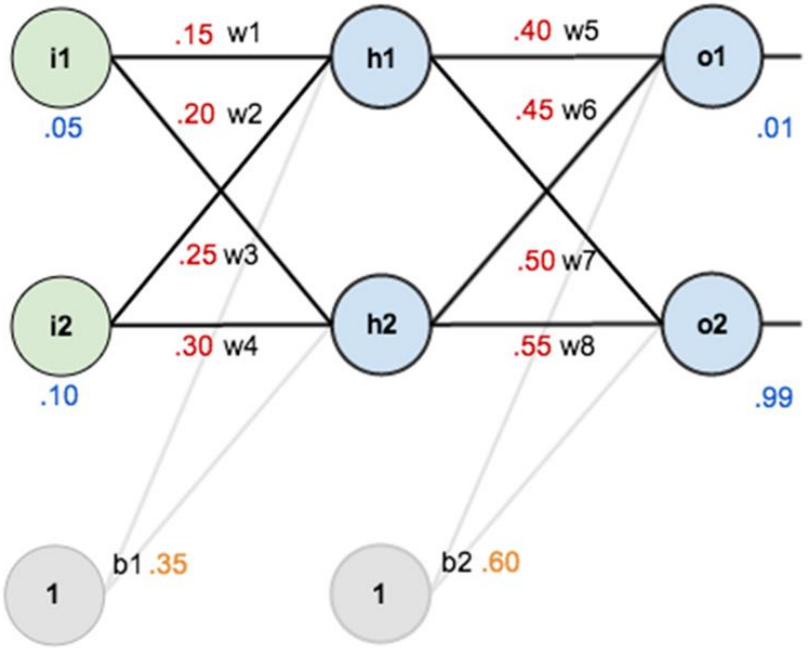
Output of o2 $out_{o2} = 0.772928465$

Error calculation : $E_{total} = \sum \frac{1}{2}(target - output)^2$

$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

$E_{o2} = 0.023560026$

$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$



The Backward Pass

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

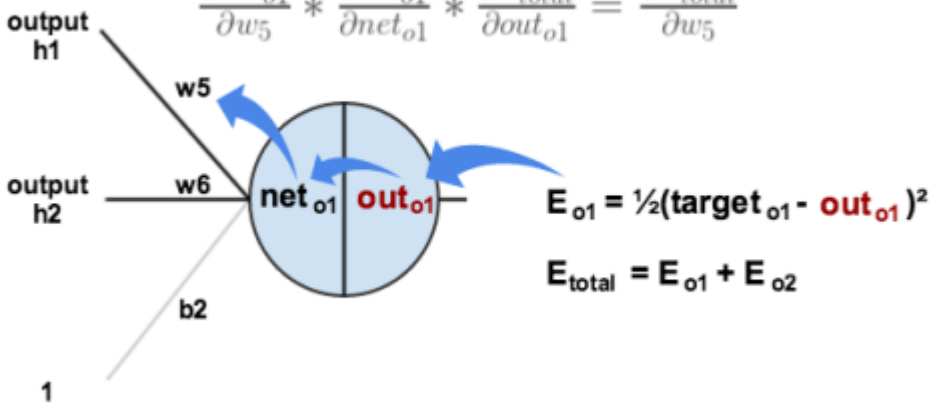
$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

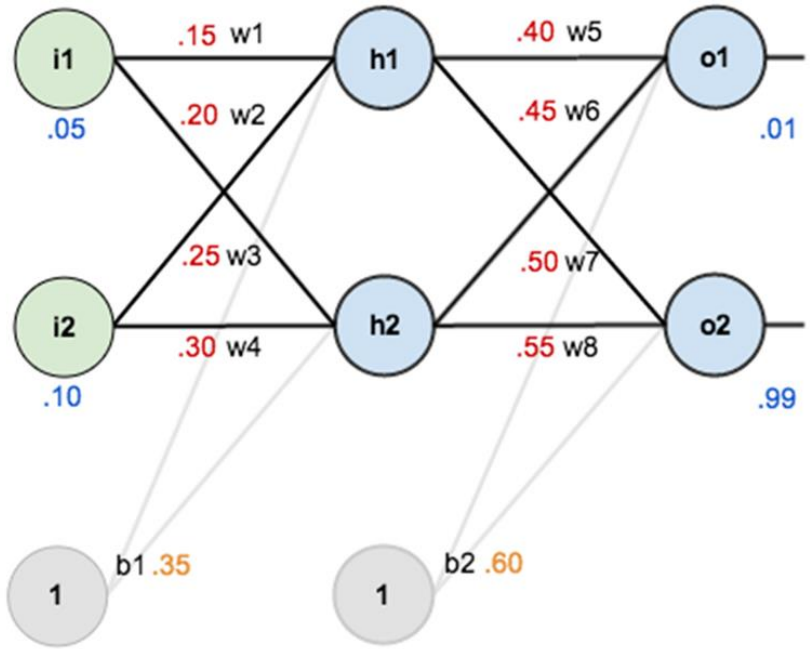
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



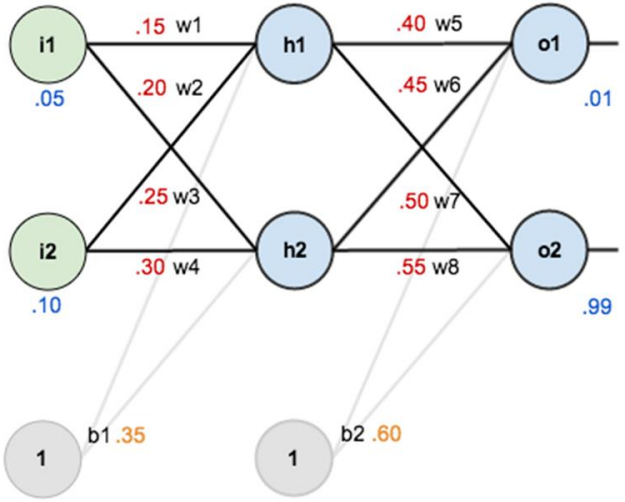


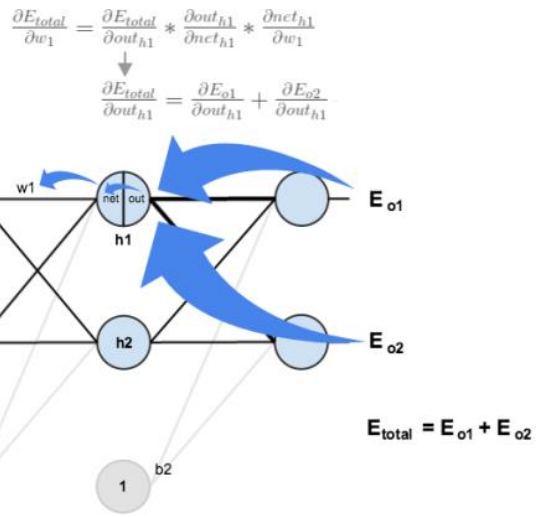
$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$





$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

$$w_2^+ = 0.19956143$$

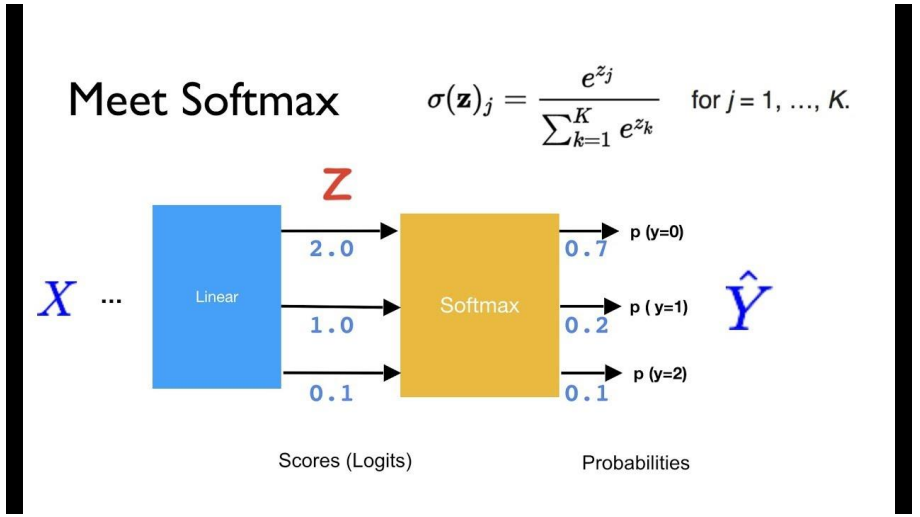
$$w_3^+ = 0.24975114$$

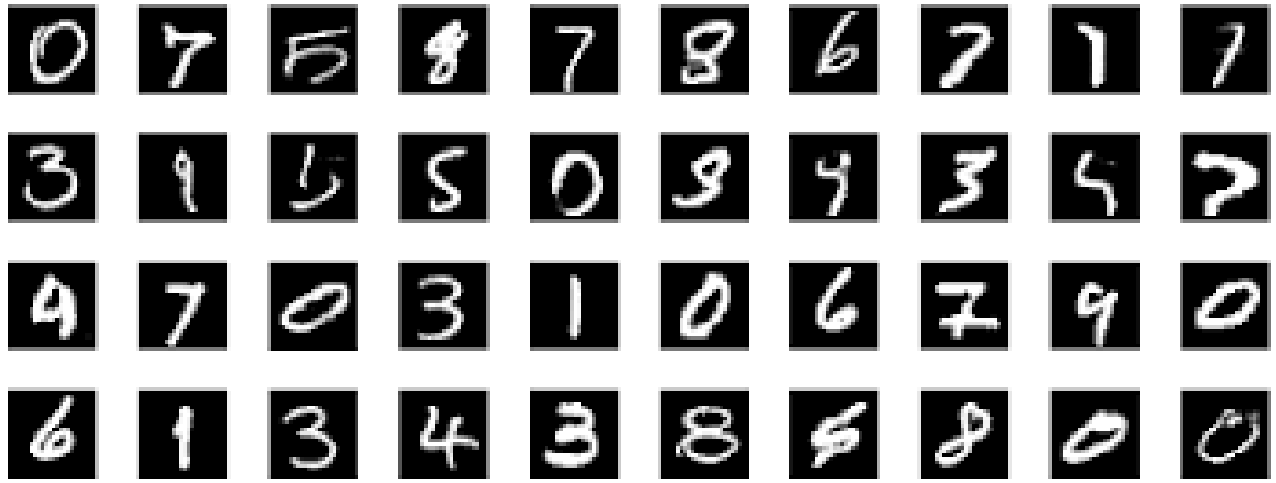
$$w_4^+ = 0.29950229$$

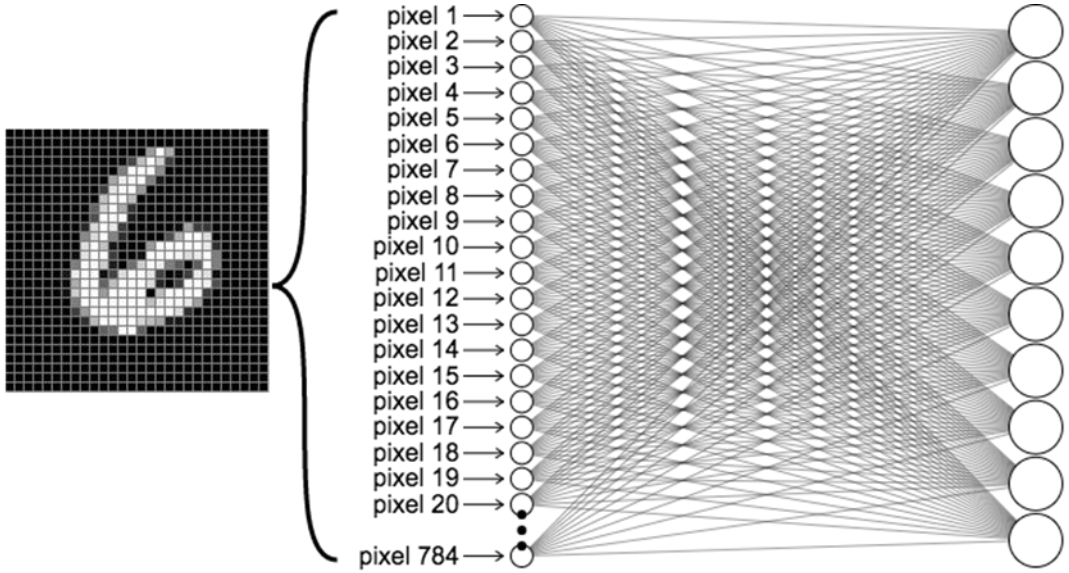
Softmax function

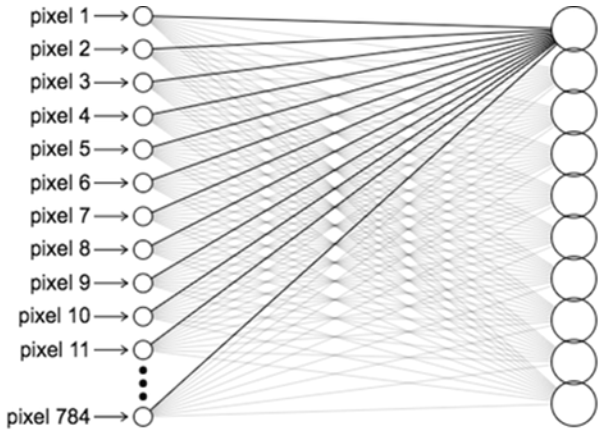
- If we wish to represent a probability distribution over a discrete variable with n possible values, we may use the softmax function.
- This can be seen as a generalization of the sigmoid function, which was used to represent a probability distribution over a binary variable.

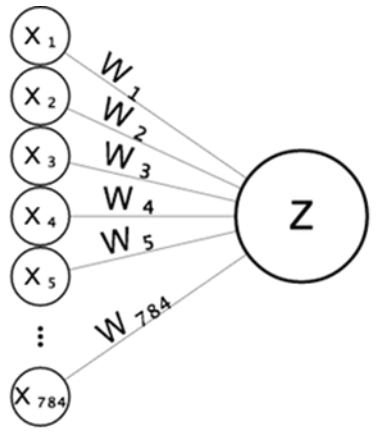
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; for \; j = 1, \ldots, k$$






pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19
pixel 20
pixel 784


pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 784

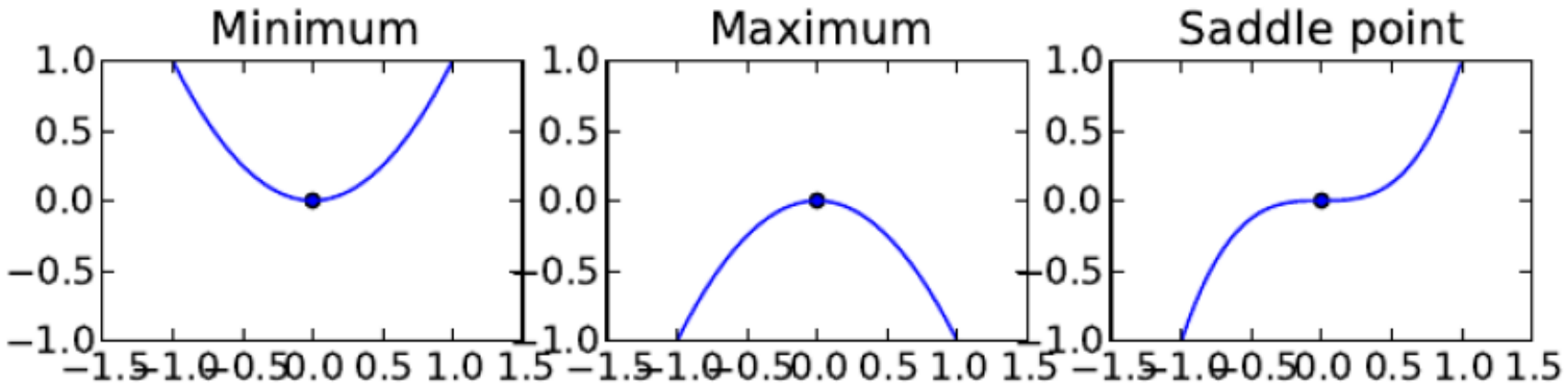

x 1
x 2
x 3
x 4
x 5
x 784
w 1
w 2
w 3
w 4
w 5
w 784
z

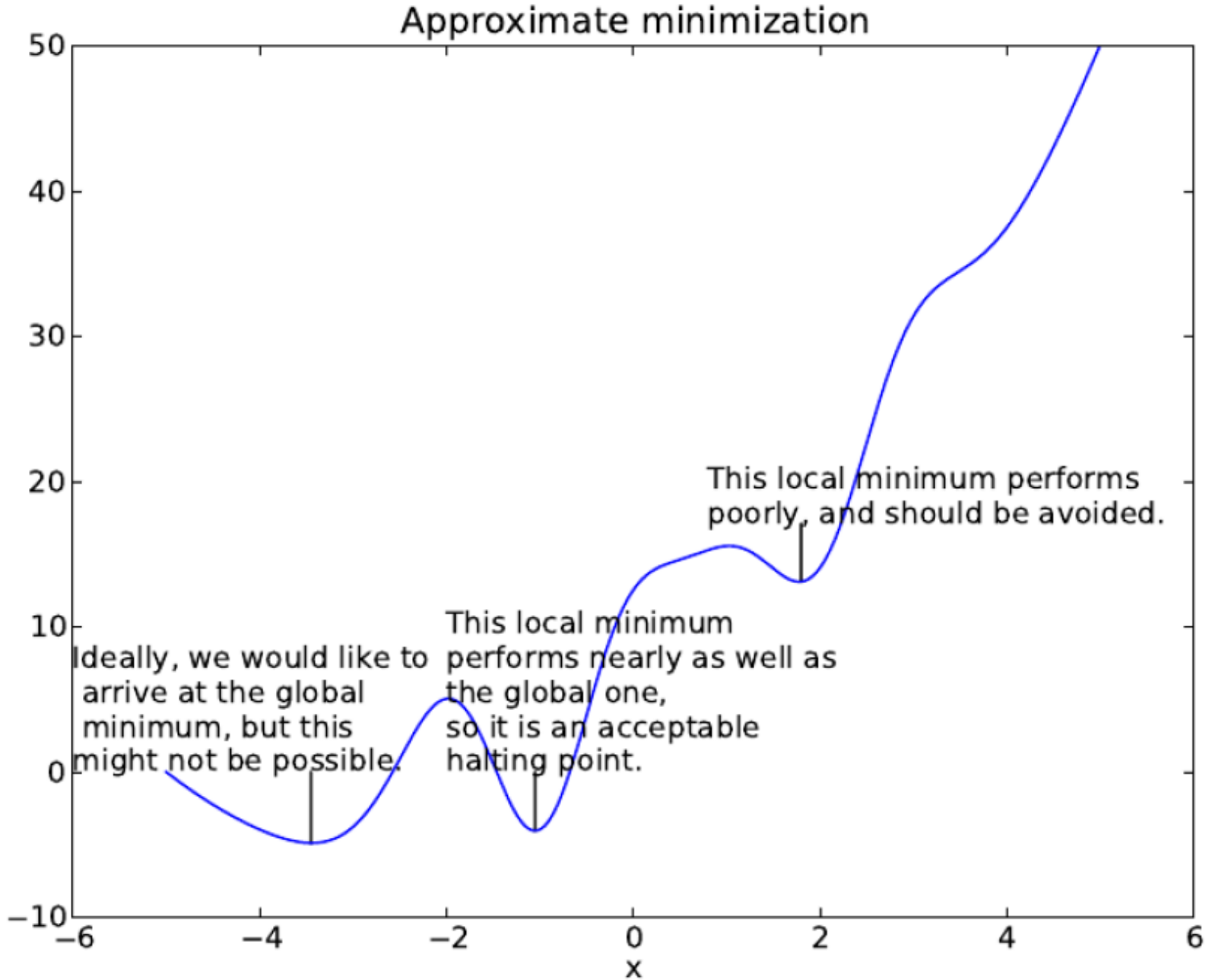
When $\frac{\partial y}{\partial x} = 0$, the derivative provides no information about which direction to move. Points where $\frac{\partial y}{\partial x} = 0$ are known as critical points or stationary points.

A *local minimum* is a point where *y* is lower than at all neighboring points, so it is no longer possible to decrease *y* by making infinitesimal steps.

A *local maximum* is a point where *y* is higher than at all neighboring points, so it is not possible to increase *y* by making infinitesimal steps.

Some critical points are neither maxima nor minima. These are known as *saddle points*.




Approximate minimization
50
40
30
20
10
0
−10
−6
−4
−2
0
2
4
6
x
This local minimum performs poorly, and should be avoided.
Ideally, we would like to arrive at the global minimum, but this might not be possible.
This local minimum performs nearly as well as the global one, so it is an acceptable halting point.