On the History and Present of Neural Networks

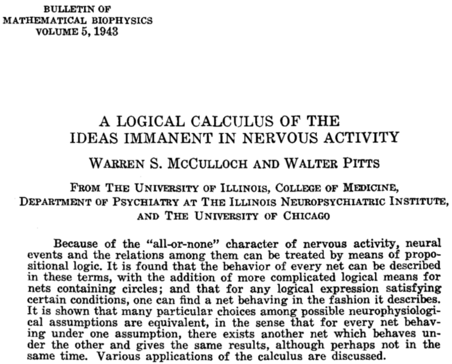
Thomas Stibor

GSI
Helmholtzzentrum für Schwerionenforschung GmbH

t.stibor@gsi.de

29th October 2024

McCulloch-Pitts Neuron



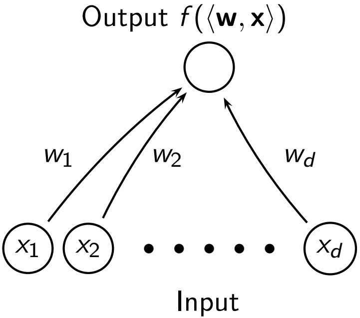
BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

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.



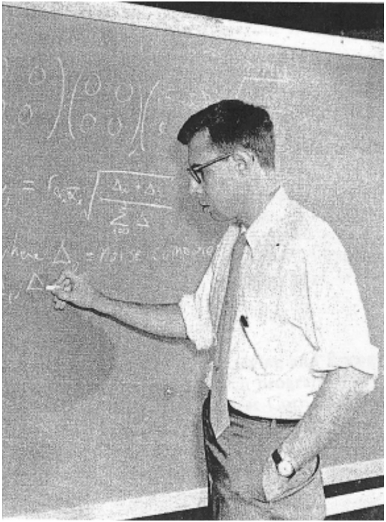
with $\mathbf{w}, \mathbf{x} \in \{0,1\}^d$

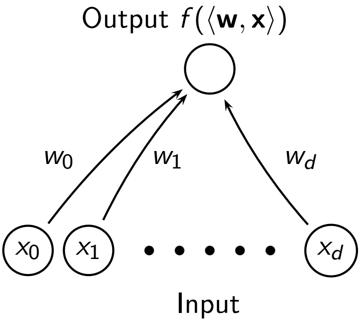
In algebraic notation: $w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \mathbf{w}^T \cdot \mathbf{x} \stackrel{def}{=} \langle \mathbf{w}, \mathbf{x} \rangle$, and threshold Θ step function,

$$f(a) = \begin{cases} 1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle \geq \Theta \\ 0 & \text{otherwise} \end{cases}$$

The McCulloch-Pitts neurons represent basic logical functions like *AND*, *OR*, and *NOT*, but doesn't have a mechanism to learn the weights $\mathbf{w}$.

Rosenblatt's Perceptron



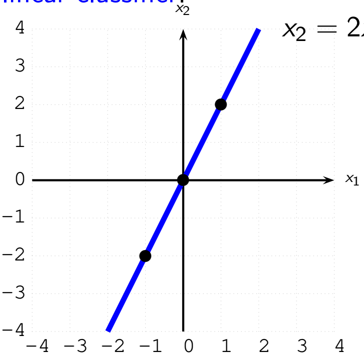


Note, $x_0 = 1$ and $\mathbf{w}, \mathbf{x} \in \mathbb{R}^{d+1}$

Proposed a *learning rule* to infer the weights values from training data.

Linear Classifier

Rosenblatt's Perceptron (also called single layer neural networks) is a linear classifier.



Observe, that $x_2 = 2x_1$ can also be expressed as

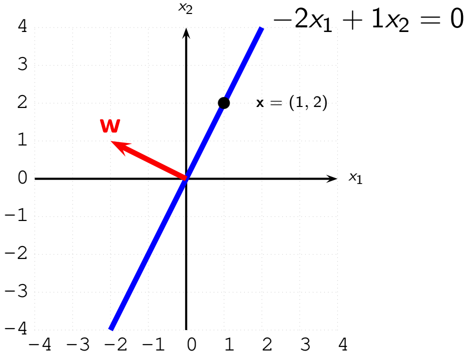
$$w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1,$$

where for instance

$$w_1 = -2,\ w_2 = 1.$$

Furthermore, observe that all points lying on the line $x_2 = 2x_1$ satisfy $w_1 x_1 + w_2 x_2 = -2x_1 + 1x_2 = 0$.

Linear Classifier & Dot Product

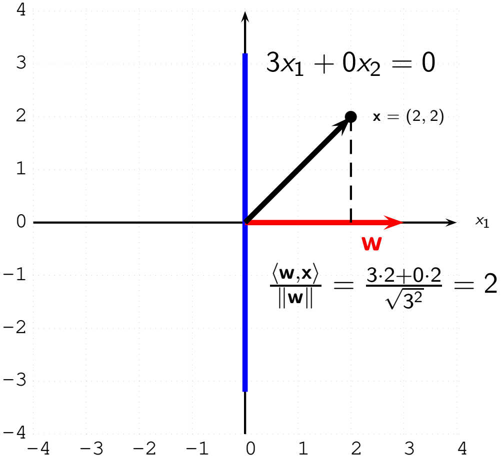


- What about the vector $\mathbf{w} = (w_1, w_2) = (-2, 1)$?
- Vector $\mathbf{w}$ is perpendicular to the line $-2x_1 + 1x_2 = 0$.
- Let us calculate the dot product of $\mathbf{w}$ and $\mathbf{x}$.

In our example $d = 2$ and we obtain $-2 \cdot 1 + 1 \cdot 2 = 0$.

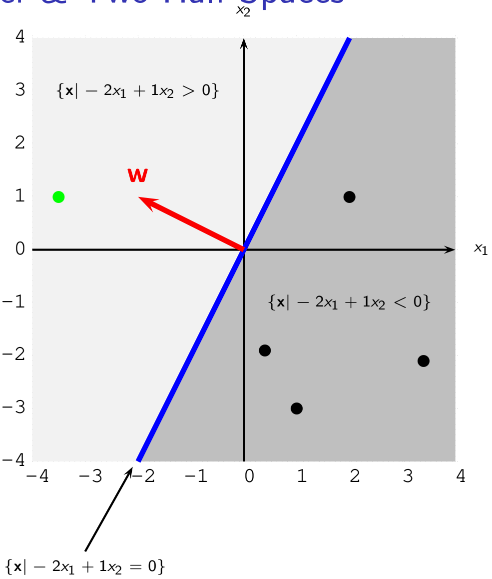
Linear Classifier & Dot Product (cont.)

Let us consider the *weight* vector $\mathbf{w} = (3, 0)$ and vector $\mathbf{x} = (2, 2)$.



Geometric interpretation of the dot product: Length of the projection of $\mathbf{x}$ onto the unit vector $\mathbf{w}/\|\mathbf{w}\|$.

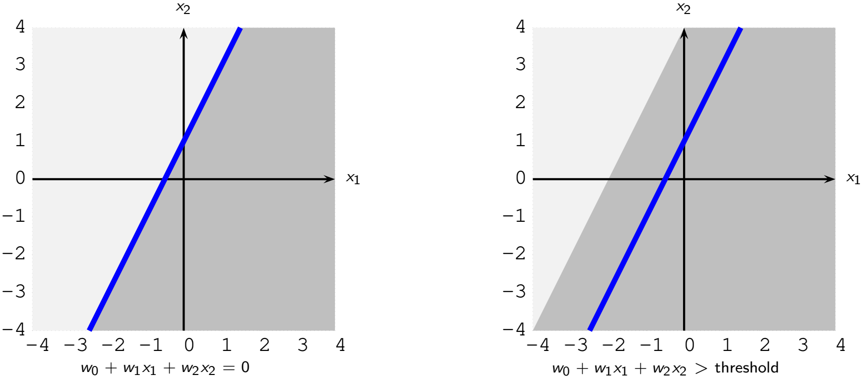
Linear Classifier & Two Half-Spaces



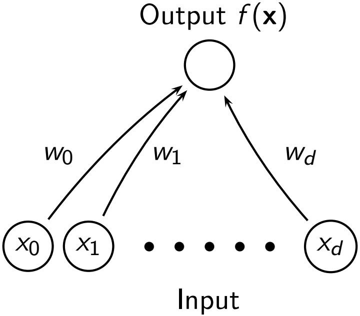
The **x**-space is separated in two half-spaces.

Linear Classifier & Dot Product (cont.)

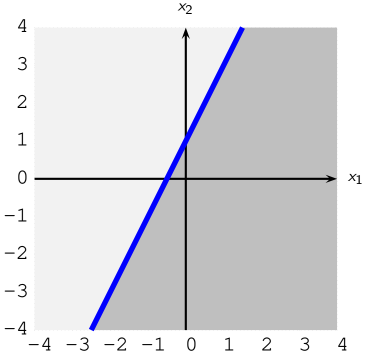
- Observe, that $w_1 x_1 + w_2 x_2 = 0$ implies, that the separating line always goes through the origin.
- By adding an offset (bias), that is $w_0 + w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \equiv y = mx + b$, one can shift the line arbitrary.



Linear Classifier & Single Layer NN



$\Leftrightarrow$



Note that $x_0 = 1$, $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$.

Given data which we want to separate, that is, a sample
$\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \in \mathbb{R}^{d+1} \times \{-1, +1\}$.

How to determine the proper values of **w** such that the "minus" and "plus" points are separated by $f(\mathbf{x})$? Infer the values of **w** from the data by some learning algorithm.

Perceptron Learning Algorithm

input : $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^{d+1} \times \{-1, +1\}, \eta \in \mathbb{R}_+$, max.epoch $\in \mathbb{N}$

output: w

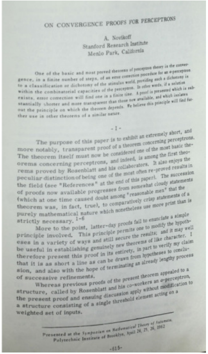
begin

    Randomly initialize **w** ;
    epoch $\leftarrow 0$;
    repeat
        for $i \leftarrow 1$ **to** N **do**
            if $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \leq 0$ **then**
                $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i y_i$
        epoch $\leftarrow$ epoch $+ 1$
    until (*epoch* $=$ *max.epoch*) *or* (*no change in* **w**);
    return w

Perceptron Convergence Theorem

How often one has to cycle through the patterns in the training set?

- If the training data is linearly separable, the perceptron learning algorithm will converge after a finite number of iterations, meaning it will find a set of weights that perfectly classify the data.
- If the data is not linearly separable, the perceptron will not converge and will continue updating its weights indefinitely.


ON CONVERGENCE PROOFS FOR PERCEPTRONS

A. Novikoff
Stanford Research Institute
Menlo Park, California

One of the basic and most proved theorems of perceptron theory is the conver-
gence, in a finite number of steps, of an error correction procedure for an α-perceptron
to a classification or dichotomy of the stimulus world, providing such a dichotomy is
within the combinatorial capacities of the perceptron. In other words, if a solution
exists, error correction will find one in a finite time. A proof is presented which is sub-
stantially shorter and more transparent than those now available, and which isolates
out the principle on which the theorem depends. We believe this principle will find fur-
ther use in other theorems of a similar nature.

- I -

The purpose of this paper is to exhibit an extremely short, and
more notably, transparent proof of a theorem concerning perceptrons.
The theorem itself must now be considered one of the most basic the-
orems concerning perceptrons, and indeed, is among the first theo-
rems proved by Rosenblatt and his collaborators. It also enjoys the
peculiar distinction of being one of the most often re-proved results in
the field (see "References" at the end of this paper). The succession
of proofs now available progresses from somewhat cloudy statements
(which at one time caused doubt among "reasonable men" that the
theorem was, in fact, true), to comparatively crisp statements of a
purely mathematical nature which nonetheless use more print than is
strictly necessary.[1-6]

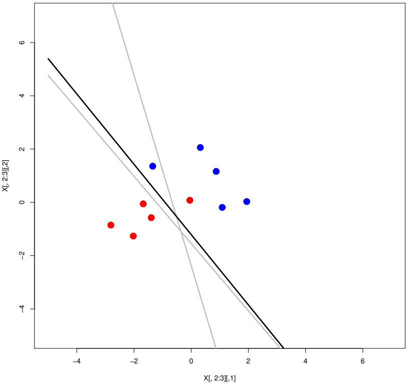
More to the point, latter-day proofs fail to enunciate a simple
principle involved. This principle permits one to modify the hypoth-
eses in a variety of ways and still secure the results; and it may well
be useful in establishing genuinely new theorems of like character. I
therefore present this proof in its entirety, in part to verify my claim
that it is as short a line as can be drawn from hypotheses to conclu-
sion, and also with the hope of terminating an already lengthy process
of successive refinements.

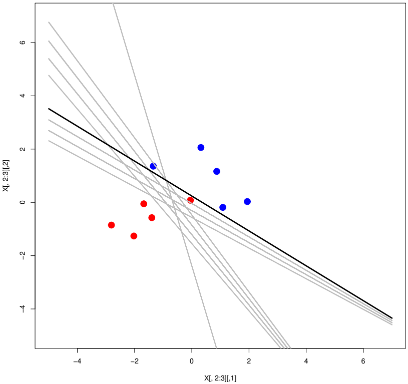
Whereas previous proofs of the present theorem appealed to a
structure, called by Rosenblatt and his co-workers an α-perceptron,
the present proof and ensuing discussion apply without modification to
a structure consisting of a single threshold element acting on a
weighted set of inputs.

Presented at the *Symposium on Mathematical Theory of Automata*,
Polytechnic Institute of Brooklyn, April 24, 25, 26, 1962

-615-

Perceptron Algorithm Visualization





One epoch

terminate if no change in **w**

From Perceptron $\text{Loss}_\ominus$ to Gradient Descent

The parameters to learn are: $(w_0, w_1, w_2) = \mathbf{w}$.

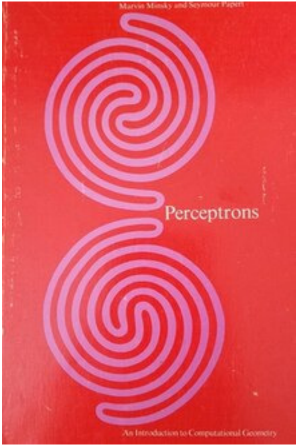
- What is our loss function $\text{Loss}_\ominus$ we would like to minimize?
- Where is term $\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \mathbf{x}\, y$ coming from?

$$\text{Loss}_\ominus \mathrel{\widehat{=}} E(\mathbf{w}) \;=\; -\sum_{m \in \mathcal{M}} \langle \mathbf{w}, \mathbf{x}_m \rangle y_m$$

where $\mathcal{M}$ denotes the set of all missclassified patterns. Moreover, $\text{Loss}_\ominus$ is *continuous* and *piecewise linear* and fits in the spirit iterative *gradient descent* method

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \nabla E(\mathbf{w}) = \mathbf{w} + \eta \mathbf{x}\, y$$

The Neural Network Winter



Perceptrons: An Introduction to Computational Geometry. Marvin Minsky and Seymour Papert, 1969.

- Analyzed the capabilities and limitations of the single-layer perceptron.
- Proved that single-layer perceptrons are fundamentally limited in their ability to solve non-linearly separable problems, such as the XOR problem.
- AI shifted their focus to other methods, particularly symbolic AI and rule-based systems.
- Funding agencies and academic institutions also deprioritized neural network research (dead-end field).

Hopfield Network

Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

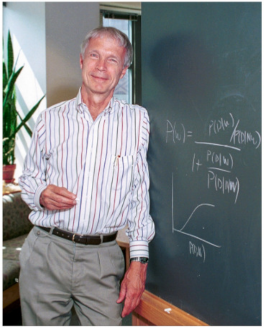
(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. HOPFIELD

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

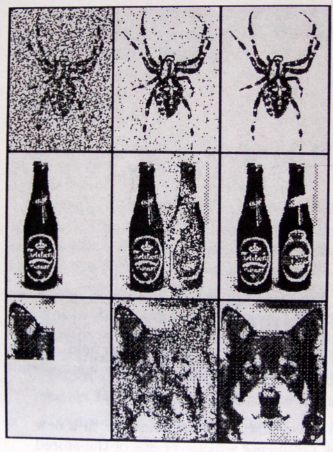
ABSTRACT Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems having a large number of simple equivalent components (or neurons). The physical meaning of content-addressable memory is described by an appropriate phase space flow of the state of a system. A model of such a system is given, based on aspects of neurobiology but readily adapted to integrated circuits. The collective properties of this model produce a content-addressable memory which correctly yields an entire memory from any subpart of sufficient size. The algorithm for the time evolution of the state of the system is based on asynchronous parallel processing. Additional emergent collective properties include some capacity for generalization, familiarity recognition, categorization, error correction, and time sequence retention. The collective properties are only weakly sensitive to details of the modeling or the failure of individual devices.



Hopfield Network Introductory Example

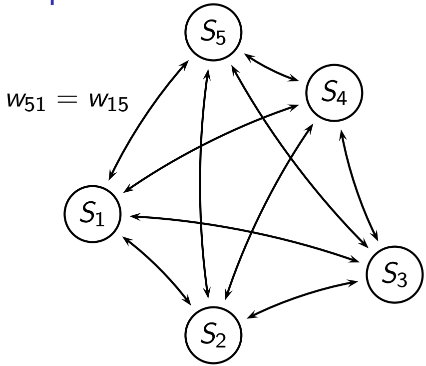
recalled by the memory

- Suppose we want to store N binary images in some memory.
- The memory should be *content-addressable* and insensitive to small errors.
- We present corrupted images to the memory (e.g. our brain) and recall the corresponding images.



presentation of corrupted images

Hopfield Network



- w_{ij} denotes weight connection from unit j to unit i
- no unit has connection with itself $w_{ii} = 0, \forall i$
- connections are symmetric $w_{ij} = w_{ji}, \forall i, j$

State of unit i can take values ± 1 and is denoted as S_i. State dynamics are governed by activity rule:

$$S_i = \text{sgn}\left(\sum_j w_{ij} S_j\right), \text{ where } \text{sgn}(a) = \begin{cases} +1 & \text{if } a \geq 0, \\ -1 & \text{if } a < 0 \end{cases}$$

Learning Rule in a Hopfield Network

Learning in Hopfield networks:

- Store a set of desired memories $\{\mathbf{x}^{(n)}\}$ in the network, where each memory is a binary pattern with $x_i \in \{-1, +1\}$.
- The weights are set using the sum of outer products

$$w_{ij} = \frac{1}{N} \sum_n x_i^{(n)} x_j^{(n)},$$

where N denotes the number of units (N can also be some positive constant, e.g. number of patterns). Given a $m \times 1$ column vector $\mathbf{a}$ and $1 \times n$ row vector $\mathbf{b}$. The outer product $\mathbf{a} \otimes \mathbf{b}$ is defined as the $m \times n$ matrix.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes [b_1 \; b_2 \; b_3] = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}, \quad m = n = 3$$

Learning in Hopfield Network (Example)

Suppose we want to store patterns $\mathbf{x}^{(1)} = [-1, +1, -1]$ and $\mathbf{x}^{(2)} = [+1, -1, +1]$.

$$\begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} \otimes [-1, +1, -1] = \begin{bmatrix} +1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & +1 \end{bmatrix}$$

$$+$$

$$\begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix} \otimes [+1, -1, +1] = \begin{bmatrix} +1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & +1 \end{bmatrix}$$

Learning in Hopfield Netw. (Example) (cont.)

$$\mathbf{W} = \frac{1}{3}\begin{bmatrix} 0 & -2 & +2 \\ -2 & 0 & -2 \\ +2 & -2 & 0 \end{bmatrix}$$

Recall: no unit has connection with itself.

The storage of patterns in the network can also be interpreted as constructing stable states. The condition for patterns to be stable is:

$$\operatorname{sgn}\left(\sum_j w_{ij} x_i\right) = x_i \,, \forall i.$$

Suppose we present pattern $\mathbf{x}^{(1)}$ to the network and want to restore the corresponding pattern.

Learning in Hopfield Netw. (Example) (cont.)

Let us assume that the network states are set as follows: $S_i = x_i, \ \forall i$. We can restore pattern $\mathbf{x}^{(1)} = [-1, +1, -1]$ as follows:

$$S_1 = \operatorname{sgn}\left(\sum_{j=1}^{3} w_{1j} S_j\right) = -1 \qquad S_2 = \operatorname{sgn}\left(\sum_{j=1}^{3} w_{2j} S_j\right) = +1$$

$$S_3 = \operatorname{sgn}\left(\sum_{j=1}^{3} w_{3j} S_j\right) = -1$$

Can we also restore the original patterns by presenting "similar" patterns which are corrupted by noise?

Updating States in a Hopfield Network

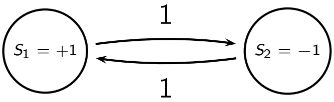
Synchronous updates:

- all units update their states $S_i = \text{sgn}\left(\sum_j w_{ij} S_j\right)$ simultaneously.

Asynchronous updates:

- one unit at a time updates its state. The sequence of selected units may be a fixed sequence or a random sequence.

Synchronously updating states can lead to oscillation (no convergence to a stable state).



Aim of a Hopfield Network

Our aim is that by presenting a corrupted pattern, and by applying iteratively the state update rule the Hopfield network will settle down in a stable state which corresponds to the desired pattern.

Hopfield network is a method for

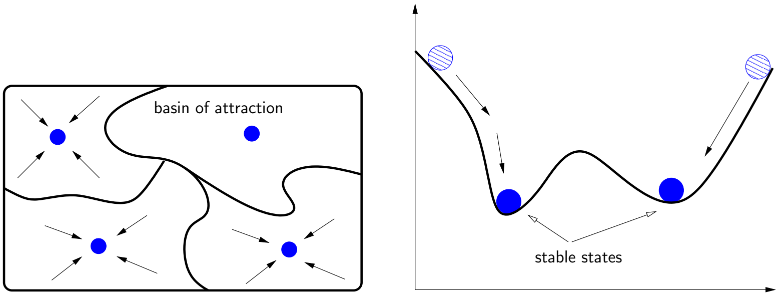
- pattern completion
- error correction.

The state of a Hopfield network can be expressed in terms of the energy function (related to Ising model and spin glass theory in Physics).

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} S_i S_j$$

Hopfield observed that if a state is a local minimum in the energy function, it is also a stable state for the network.

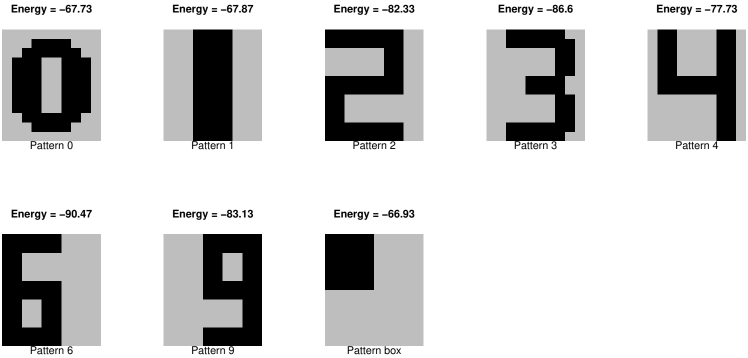
Basin of Attraction and Stable States



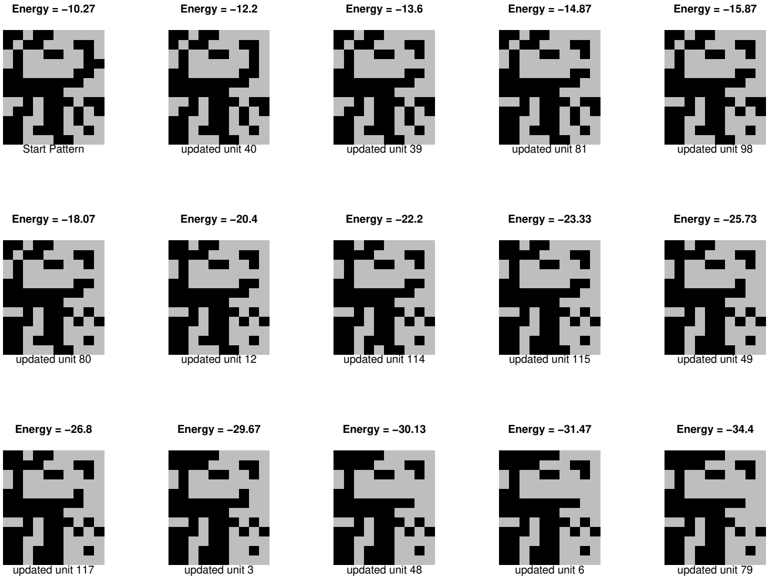
Within the space the stored patterns $\mathbf{x}^{(n)}$ are acting like attractors.

Haykin's Digit Example

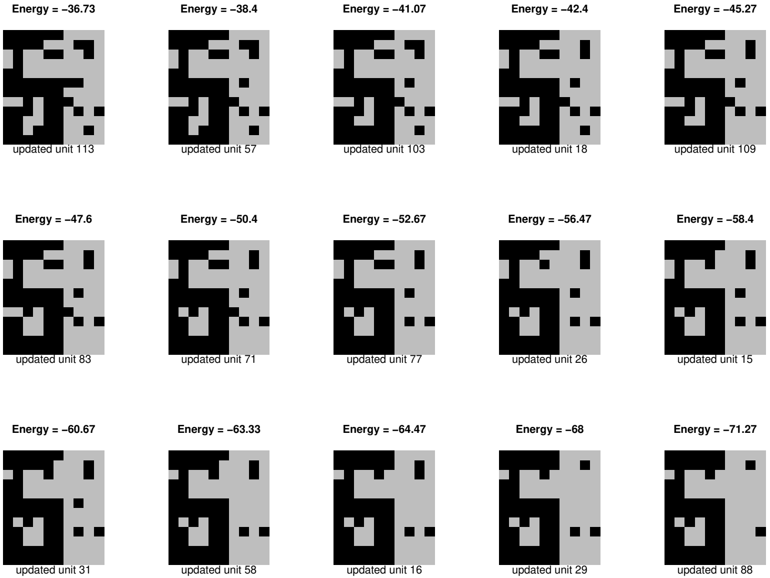
Suppose we stored the following digits in the Hopfield network:



Updated States of Corrupted Digit 6

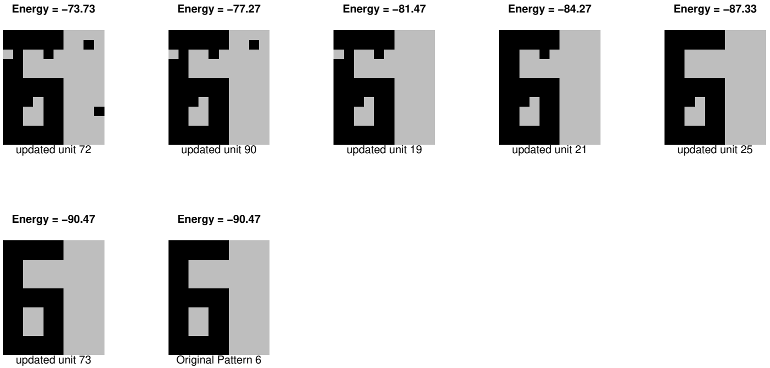


Updated States of Corrupted Digit 6 (cont.)



Updated States of Corrupted Digit 6 (cont.)

The resulting pattern (stable state with energy -90.47) matches the desired pattern.



Hopfield Networks Summary

- Learning: determine the weight matrix from the data with the outer product.
- Memory: “knowledge” is stored in the weight matrix.
- Queries to memory: apply state update rule until energy is minimized (local minimum).

John Hopfield laid the groundwork for:

- Renewed theoretical interest and connections to Physics.
- Neural network applications for optimization.
- Paved the way for recurrent networks.
- Revival of interest in multilayer networks and backpropagation.

Backpropagation the Heart of Neural Networks

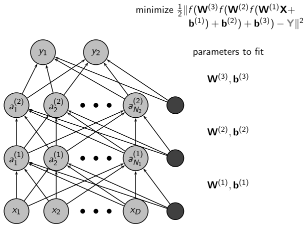
Adjust weights of connections within the network to minimize the error between the predicted and actual output.

History:

- The minimisation of errors through gradient descent (Cauchy 1847).
- ...
- Taylor Expansion of the Accumulated Rounding Error (Seppo Linnainmaa 1970 Master Thesis, backpropagation modern version).
- ...
- Learning representations by back-propagating errors (Rumelhart, Hinton and Williams, 1986).

Who Invented Backpropagation? (Excellent article by Jürgen Schmidhuber).

Learning in Neural Networks with Backpropagation



Core idea:

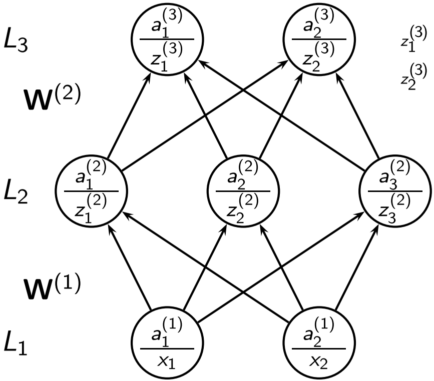
- Calculate error of loss function and change weights and biases based on output.
- These "error" measurements for each unit can be used to calculate the partial derivatives.
- Use partial derivatives with gradient descent for updating weights and biases and minimizing loss function.

Problem: At which magnitude one shall change e.g. weight $W_{ij}^{(1)}$ based on error of y_2?

Learning in Neural Networks with Backpropagation (cont.)

Input: x_1, x_2, output: $a_1^{(3)}, a_2^{(3)}$, target: y_1, y_2 and $g(\cdot)$ is activation function. NN calculates[2] $g(\mathbf{W}^{(2)} g(\mathbf{W}^{(1)}\mathbf{x}))$.

$$E(\mathbf{W}) = \frac{1}{2}\left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2\right] = \frac{1}{2}\|\mathbf{a}^{(3)} - \mathbf{y}\|^2$$



Forward pass

$$z_1^{(3)} = W_{10}^{(2)} a_0^{(2)} + W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} \qquad a_1^{(3)} = g(z_1^{(2)})$$
$$z_2^{(3)} = W_{20}^{(2)} a_0^{(2)} + W_{21}^{(2)} a_1^{(2)} + W_{22}^{(2)} a_2^{(2)} + W_{23}^{(2)} a_3^{(2)} \qquad a_2^{(3)} = g(z_2^{(2)})$$

$$\underbrace{\mathbf{z}^{(3)}}_{2\times 1} = \underbrace{\mathbf{W}^{(2)}}_{2\times 4}\underbrace{\mathbf{a}^{(2)}}_{4\times 1} \qquad \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

$$z_1^{(2)} = W_{10}^{(1)} x_0 + W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 \qquad a_1^{(2)} = g(z_1^{(2)})$$
$$z_2^{(2)} = W_{20}^{(1)} x_0 + W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 \qquad a_2^{(2)} = g(z_2^{(2)})$$
$$z_3^{(2)} = W_{30}^{(1)} x_0 + W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 \qquad a_3^{(2)} = g(z_3^{(2)})$$

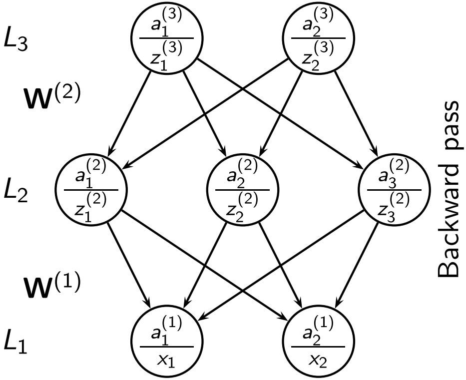
$$\underbrace{\mathbf{z}^{(2)}}_{3\times 1} = \underbrace{\mathbf{W}^{(1)}}_{3\times 3}\underbrace{\mathbf{x}}_{3\times 1} \qquad \mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

[2] Notation adapted from Andew Ng's slides.

Learning in Neural Networks with Backpropagation (cont.)

For each node we calculate $\delta_j^{(l)}$, that is, error of unit j in layer l, because $\frac{\partial}{\partial W_{ij}^{(l)}} E(\mathbf{W}) = a_j^{(l)} \delta_i^{(l+1)}$. Note $\odot$ is element wise multiplication.

$$E(\mathbf{W}) = \frac{1}{2}\left[(a_1^{(3)} - y_1)^2 + (a_2^{(3)} - y_2)^2\right] = \frac{1}{2}\|\mathbf{a}^{(3)} - \mathbf{y}\|^2$$



$$\boldsymbol{\delta}^{(3)} = (\mathbf{a}^{(3)} - \mathbf{y}) \odot g'(\mathbf{z}^{(3)})$$

$$\boldsymbol{\delta}^{(2)} = (\mathbf{W}^{(2)})^T \delta^{(3)} \odot g'(\mathbf{z}^{(2)})$$

Note $\boldsymbol{\delta}^{(1)}$ is the input, so no term.

Learning in Neural Networks with Backpropagation (cont.)

Backpropagation = forward pass & backward pass

Given labeled training data $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$.
Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j. Value Δ will be used as accumulators for computing partial derivatives.
For $n = 1$ to N

- Forward pass, compute $\mathbf{z}^{(2)}, \mathbf{a}^{(2)}, \mathbf{z}^{(3)}, \mathbf{a}^{(3)}, \ldots, \mathbf{z}^{(L)}, \mathbf{a}^{(L)}$
- Backward pass, compute $\boldsymbol{\delta}^{(L)}, \boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}$
- Accumulate partial derivate terms, $\boldsymbol{\Delta}^{(l)} := \boldsymbol{\Delta}^{(l)} + \boldsymbol{\delta}^{(l+1)}(\mathbf{a}^{(l)})^T$

Finally calculated partial derivatives for each parameter:
$\frac{\partial}{\partial W_{ij}^{(l)}} E(\mathbf{W}) = \frac{1}{N}\Delta_{ij}^{(l)}$ and use these in gradient descent.

See interactive demo.

Neural Networks vs. Kernel Methods (1995 - 2012)

Support Vector Machines and Kernel Methods were favorite methods in the field of machine learning.

Neural networks suffered from:

- Slow training time: Took usually weeks and made experimentation and tuning difficult.
- Vanishing and exploding gradient problem: Especially severe with sigmoid and tanh activation functions.
- Lack of labeled data sets.
- Lack of neural network frameworks (TensorFlow, PyTorch, MXNet, etc...): Usually Matlab code.

Era of Deep Learning Neural Networks

Geoffrey Hinton

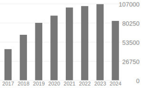
FOLGEN

Emeritus Prof. Computer Science, University of Toronto
Bestätigte E-Mail-Adresse bei cs.toronto.edu - Startseite
machine learning   psychology   artificial intelligence   cognitive science   computer science

TITEL	ZITIERT VON	JAHR
Imagenet classification with deep convolutional neural networks A Krizhevsky, I Sutskever, GE Hinton Advances in neural information processing systems 25	164596 *	2012
Deep learning Y LeCun, Y Bengio, G Hinton Nature 521 (7553), 436-44	85943	2015
Learning internal representations by error-propagation DE Rumelhart, GE Hinton, RJ Williams Parallel Distributed Processing: Explorations in the Microstructure of ...	55130 *	1986
Dropout: a simple way to prevent neural networks from overfitting N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov The journal of machine learning research 15 (1), 1929-1958	53355	2014
Visualizing data using t-SNE L van der Maaten, G Hinton Journal of Machine Learning Research 9 (Nov), 2579-2605	48419	2008

Zitiert von   ALLE ANZEIGEN

	Alle	Seit 2019
Zitate	861599	567502
h-index	187	137
i10-index	484	369



Öffentlicher Zugriff   ALLE ANZEIGEN

1 Artikel   8 Artikel
nicht verfügbar   verfügbar

Basierend auf Fördermandaten

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

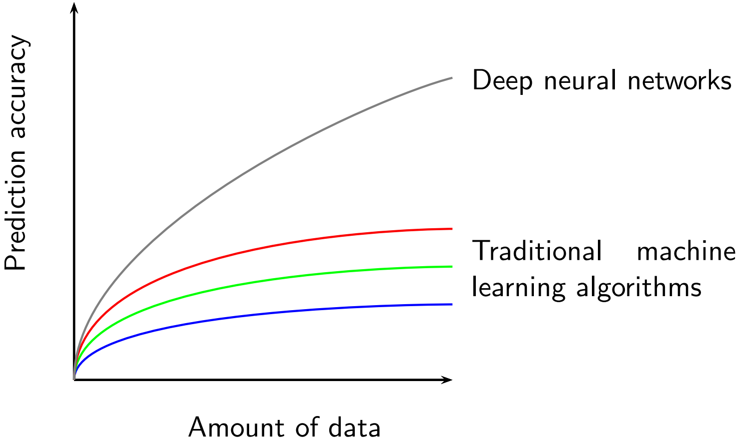
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Why are Deep Neural Networks so successful?



Deep Neural Networks (Backpropagation) are *universal*, that is, applicable to a large class of problems: Vision, speech, text, . . . and *scale* with data. Backpropagation (forward + backward pass) is intrinsically linked to matrix multiplication (GPU's, TPU's).