Introduction to Machine Learning.
CSCI-UA 9473, Zoom 1.

Augustin Cosse

Ecole Normale Supérieure, DMA & NYU

2020

What have we seen so far?

- Linear regression
  - Solution through gradient descent
  - Solution through Normal equations
  - Regularization (Ridge, Lasso, Subset Selection)
- Linear classification
  - Separating hyperplane
  - Discriminative vs Generative classifiers
    - Logistic regression
    - Linear/Gaussian discriminant Analysis (GDA)
- Non parametric regression/classification
  - Kernel methods (use in gradient descent)
  - Support vector machines

This week

- Neural Networks
    - Current applications
    - History
    - Universal Approximation Properties
    - Training/Backpropagation
    - Local mins and symmetries/ regularization

Reminders

- Linear regression = linear combination of fixed (possibly non linear) basis functions

$$Y = \beta_0 + \sum_{k=1}^{d} \beta_k X_k$$

$$Y = \beta_0 + \sum_{k=1}^{d} \beta_k \phi_k(X)$$

- Linearity in the parameters leads to interesting properties such as closed form solution, computational tractability,...

Reminders

- The difficulty stems from the fact that basis functions $\phi_i(X)$ are fixed before training
- For advanced models, the number of such basis functions grows rapidly with the dimension of the space
- The model must be reset each time a new point is being added to the training set

Reminders

- One solution was to use non parametric models such as SVMs
- .. But those grow in complexity with the size of the training set. In good frameworks, there are few support vectors, but in the worst case, the number of support vectors is the number of training samples
- In NLP for example, SVM classifiers with 10,000 support vectors is not uncommon

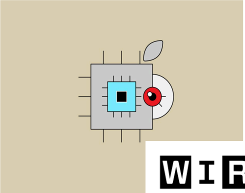
DarwinAI raises $3 million for AI that optin neural networks

KYLE WIGGERS @KYLE_L_WIGGERS SEPTEMB 18 7:55 AM

VentureBeat

TOM SIMONITE BUSINESS 09.18.18 03:01 AM

HOW APPLE MAKES THE AI CHIP POWERING THE IPHONE'S FANCY TRICKS



WIRED

FINANCIAL TIMES

Special Report Artificial intelligence + Add to myFT

Neural networks allow us to 'read faces' in a new way

Facial analysis software is being used to predict sexuality and security risks

Deeep neural networks are making facial recognition software significantly more accurate © Getty

lways Learning, Always Growing: How Neural works Do The Hard Work

Billionaires Innovation Leadership Money Consumer Industry Lifestyle Featured BrandVoice

Forbes insights

Insights Team Insights Contributor

FORBES INSIGHTS With Intel AI

Neural Networks: The biological inspiration

(E. Roberts, Stanford, C. Stergiou & D. Siganos, Imperial College)

- Much is still unknown about how the brain train itself to process information

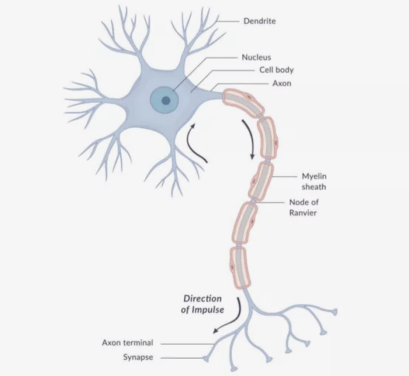
- A biological neuron collects signals from other neurons through fine structures called dendrites

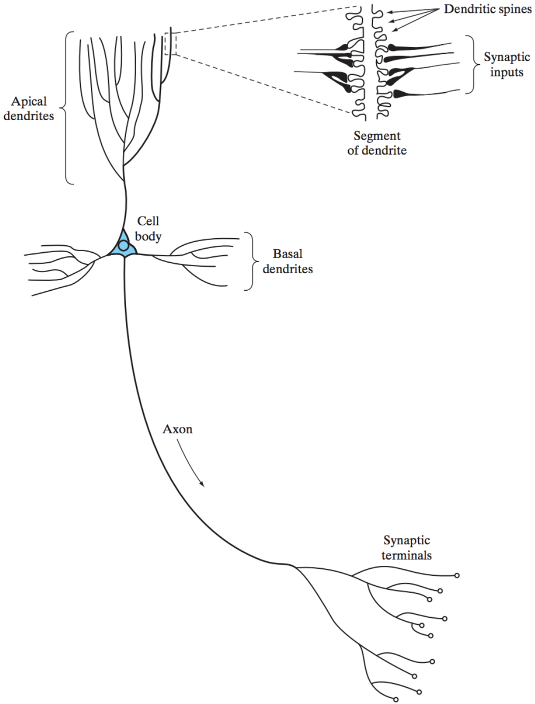
- The neuron then sends spikes of electrical activity through a long stand named axon which splits into thousands of branches

- At the end of each branch, a structure called synapse converts the activity from the axon into electrical effects that inhibit or excite acitivity in the connected neuron

Neural Networks: The biological inspiration

- When a neuron receives excitatory input that is sufficiently large compared to its inhibitory inputs, it sends a spike of electrical activity down its axon
- Learning results from changes in the strength of the synapse (e.g. past patterns of use)

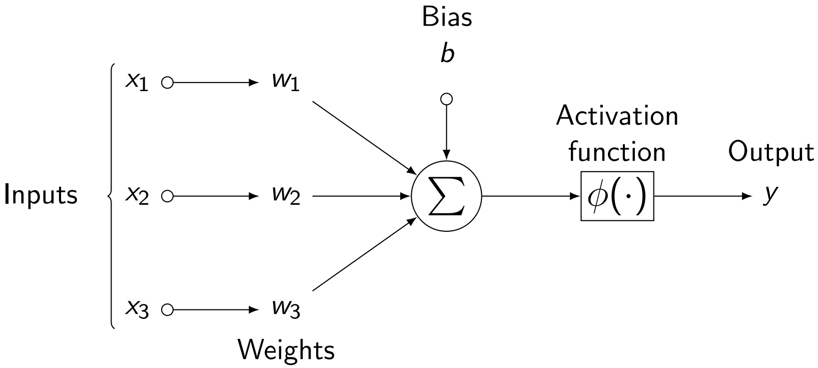




Haykin,Neural Networks
Learning Machines

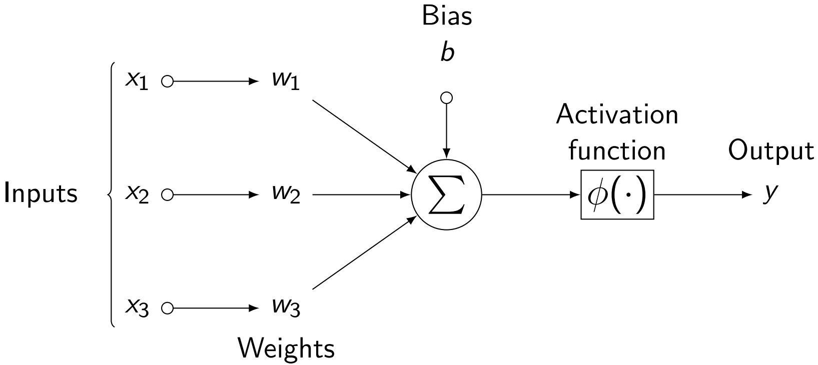
Neural Networks: From Biology to Neural Nets

- The original idea is to extract the original features of neurons and their interconnections. An artificial neuron is a device with many inputs and one output

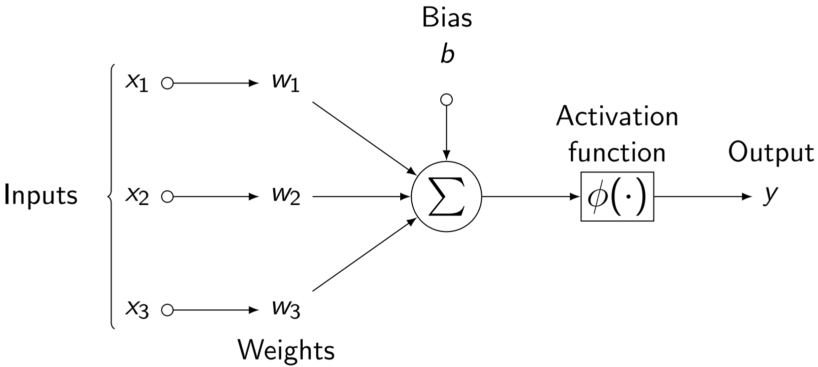


Neural Networks: From Biology to Neural Nets

- Just as other ML algorithms, the artificial neuron has two modes of operation: a training mode and a test mode
- In training mode, the neuron learns to fire or not for specific input patterns. In the test mode, the firing is controled by the firing rule which was learned at training



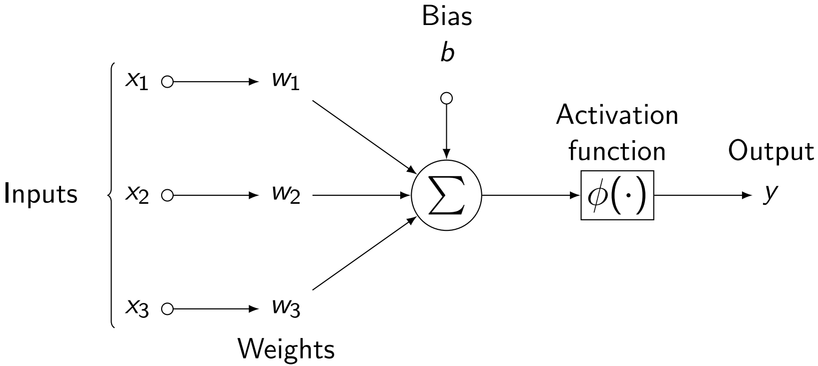
Neural Networks: From Biology to Neural Nets



(Single Unit) $$y = \phi\left(\sum_{j=1}^{3} w_j x_j + b\right)$$

Neural Networks: From Biology to Neural Nets

- The function $\phi(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$ is called Ridge function and it varies only in the direction defined by $\boldsymbol{w}$
- The general regression model $y = \sum_{m=1}^{M} \phi_m(\boldsymbol{w}_m^T \boldsymbol{x})$ is known as Projection pursuit Regression (PPR) as the input to ϕ is the projection of $\boldsymbol{x}$ onto $\boldsymbol{w}$

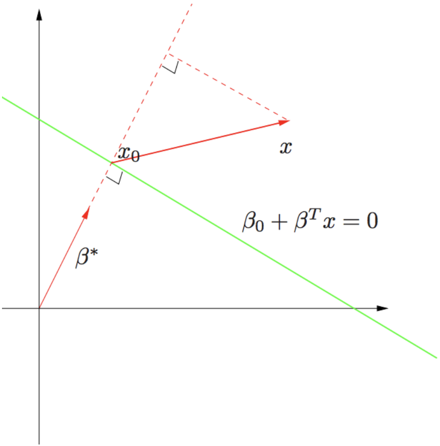


Interlude: The Perceptron

- Consider the separating hyperplane $\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}$
- $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ belong to the plane if they satisfy

$$\boxed{\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_1 = \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_2}$$

- We thus have $\boldsymbol{\beta}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2$ in the plane
- $\boldsymbol{\beta}$ ($\boldsymbol{\beta}^* = \boldsymbol{\beta}/\|\boldsymbol{\beta}\|$) is the vector normal to the hyperplane



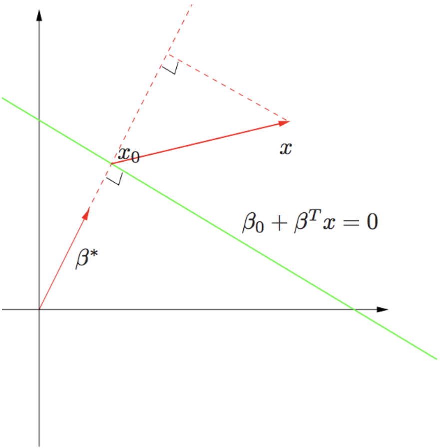
H,T,F, Elem. of Stat. Learn.

- The signed distance of a point $\boldsymbol{x}$ to the hyperplane is defined as

$$
\begin{aligned}
&(\boldsymbol{\beta}^{*})^{T}(\boldsymbol{x}-\boldsymbol{x}_{0}) \\
&= \frac{1}{\|\boldsymbol{\beta}\|}(\boldsymbol{\beta}^{T}\boldsymbol{x}+\beta_{0})
\end{aligned}
$$

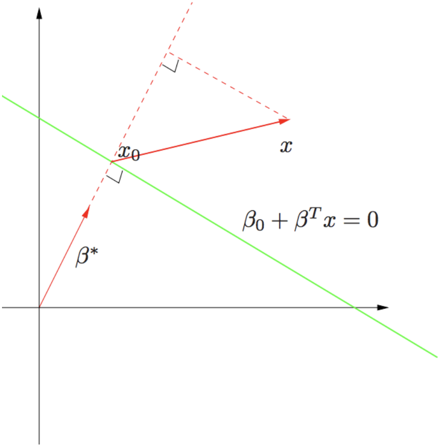
- Points that are located above thus lead to positive values $\beta^{T}\boldsymbol{x}+\beta_{0}>0$

- Points that are located below lead to negative values $\beta^{T}\boldsymbol{x}+\beta_{0}<0$



H,T,F, Elem. of Stat. Learn.

- A separating plane thus gives a natural way to associate positive or negative labels to points
- For a two class classification problem, we can look for the plane that gives positive labels to one class and negative labels to the other
- This idea leads to the perceptron algorithm of Rosenblatt



H,T,F, Elem. of Stat. Learn.

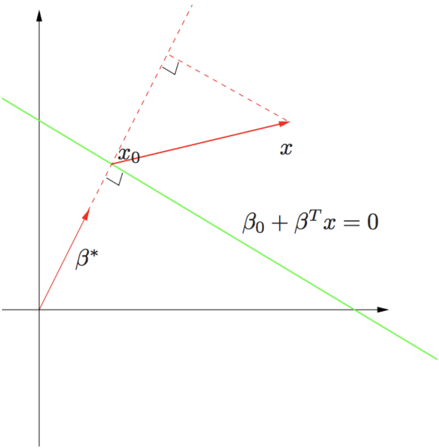
- The perceptron thus simply reads as

$$\boxed{y(\boldsymbol{x}) = f(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x})}$$

Where

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- During training, we associate $+1$ labels to points in cluster $\mathcal{C}_1$ and -1 labels to points in cluster $\mathcal{C}_2$



H,T,F, Elem. of Stat. Learn.

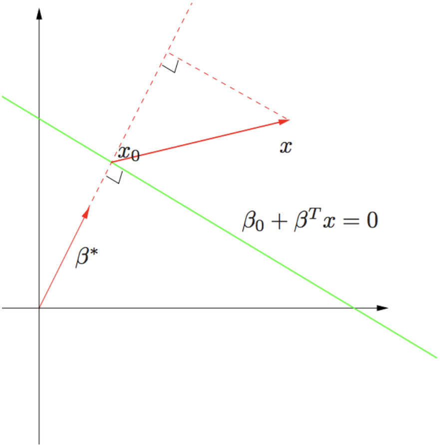
- In the perceptron, a point in $\mathcal{C}_1$ ($y_i = +1$) is thus misclassified if $\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0 < 0$
- Generally we would like all points to satisfy

$$\boxed{y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) > 0}$$

- so we minimize

$$\boxed{-\sum_{i \in \mathcal{M}} y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0)}$$

(contributions should be ≥ 0)



H,T,F, Elem. of Stat. Learn.

Perceptron

$$\text{(Perceptron)} \qquad D(\boldsymbol{\beta}, \beta_0) = -\sum_{i\in\mathcal{M}} y_i(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i)$$

- How do we train the perceptron?
- One way is to use stochastic gradient descent (we will come back to that idea later)
- General idea (perceptron learning algorithm)
  - Choose initial vector of prefactors $\boldsymbol{\beta}$
  - Then for each misclassified points $\boldsymbol{x}_n$, do

$$\begin{bmatrix} \boldsymbol{\beta}^{k+1} \\ \beta_0^{k+1} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{\beta}^{k} \\ \beta_0^{k} \end{bmatrix} - \eta \nabla D^n(\boldsymbol{\beta}, \beta_0), \quad D^n = y_n(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_n)$$

Perceptron: intuition

Perceptron w/
gen. features $\phi(X)$

$$D(\boldsymbol{\beta}, \beta_0) = -\sum_{i \in \mathcal{M}} y_i(\beta_0 + \boldsymbol{\beta}^T \phi_i)$$

$$\begin{bmatrix} \boldsymbol{\beta}^{k+1} \\ \beta_0^{k+1} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{\beta}^{k} \\ \beta_0^{k} \end{bmatrix} - \eta \nabla D^n(\boldsymbol{\beta}, \beta_0), \quad D^n = -y_n(\beta_0 + \boldsymbol{\beta}^T \phi(\boldsymbol{x}_n))$$

if no intercept $\quad \boldsymbol{\beta}^{k+1} \leftarrow \boldsymbol{\beta}^k + \eta \boldsymbol{\phi}_n y_n$

- Perceptron convergence Theorem: If there exists an exact solution (data is linearly separable), then the perceptron algorithm is guaranteed to find an exact solution in a finite number of steps.

Perceptron: more intuition

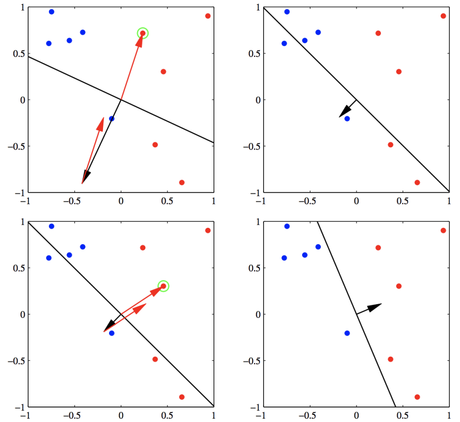


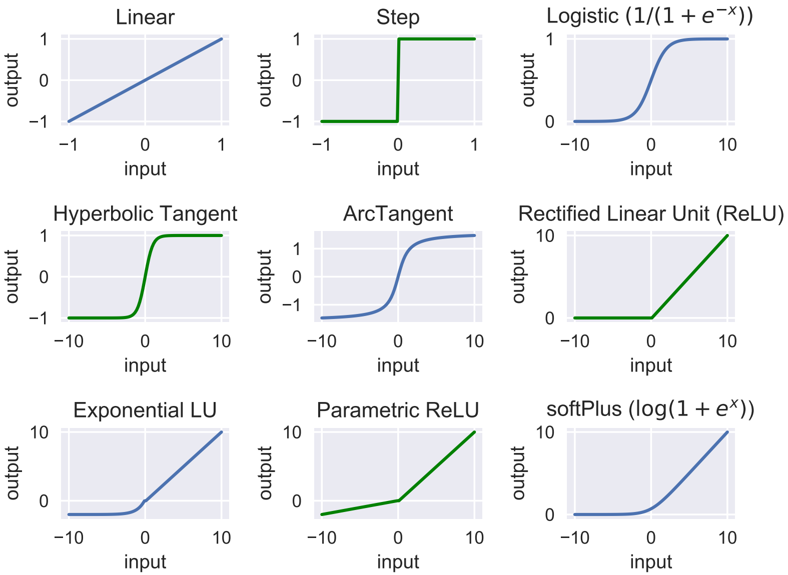
Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space (ϕ_1, ϕ_2). The top left plot shows the initial parameter vector $\mathbf{w}$ shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.

Bishop, Pattern Recogn. and ML.

- Let us assume no intercept (data has been centered)
- For each misclassified points $\boldsymbol{x}_n$ (resp. $\phi(X_n)$), the algorithm adds the pattern of the misclassified point to the weight vector $\boldsymbol{\beta}$

$$
\begin{aligned}
& -(\boldsymbol{\beta}^{k+1})^T \phi_n y_n \\
= & -(\boldsymbol{\beta}^{k})^T \phi_n y_n \\
& -(\phi_n y_n)^T \phi_n y_n \\
< & -(\beta^{k})^T \phi_n y_n
\end{aligned}
$$

Neural Networks: activation functions

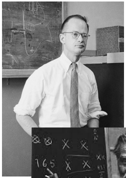
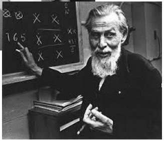


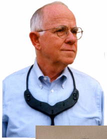
How to choose the activation function?

- A good choice is the Relu
- If the network suffers from dead neurons during training, then you can switch to leaky ReLu or Maxout

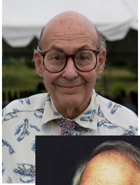
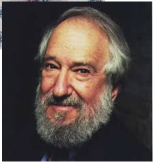
Progression **Degression**

1943	1958	1962	1969
McCulloch and Pitts	Rosenblatt	Bernard Widrow	Marvin Minsky


Marcian Hoff

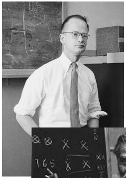
Seymour Papert

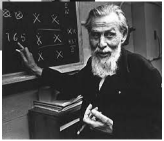
- 1943. In order to describe how neurons in the brain might work, McCulloch and Pitts model a simple neuron using electrical circuits (thresholded logic unit)
- 1958. Rosenblatt develops the perceptron (first precursor to modern neural nets)

Progression **Degression**

1943 1958 1962 1969

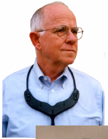
McCulloch and Pitts





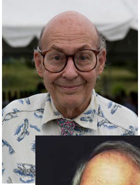
Rosenblatt

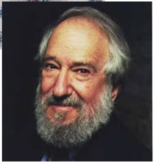
Bernard Widrow



Marcian Hoff

Marvin Minsky





Seymour Papert

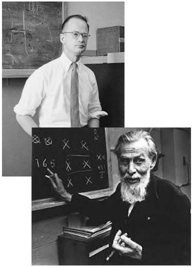
- 1958. Together with Rosenblatt's perceptron come the learning rule and the convergence Theorem (1962).

  "[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Progression **Degression**

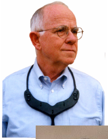
1943 1958 1962 1969

McCulloch and Pitts



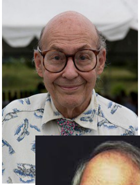
Rosenblatt

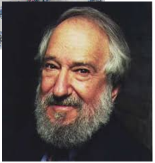
Bernard Widrow



Marcian Hoff

Marvin Minsky





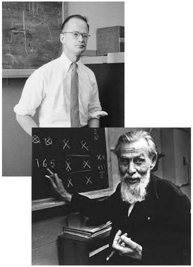
Seymour Papert

- 1959-1962. Widrow and Hoff develop models called ADALINE and MADALINE ((Multiple ADAptive LINear Elements)) to recognize binary patterns. The system is still in commercial use.

Progression **Degression**

1943 | 1958 | 1962 | 1969

McCulloch and Pitts

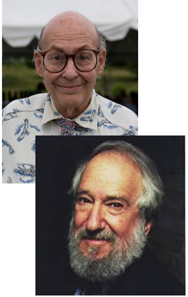


Rosenblatt

Bernard Widrow

Marcian Hoff

Marvin Minsky



Seymour Papert

- Adaline = Perceptron trained on continuous inputs,

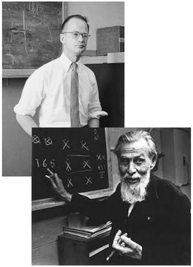
$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t - \sigma(\mathbf{w}^T\phi(\mathbf{x})))\phi(\mathbf{x}) \quad \text{Perceptron}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t - (\mathbf{w}^T\phi(\mathbf{x})))\phi(\mathbf{x}) \quad \text{ADALINE}$$

Progression **Degression**

1943 1958 1962 1969

McCulloch and Pitts

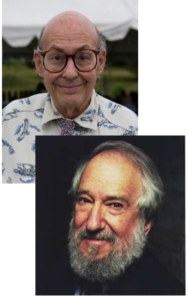


Rosenblatt

Bernard Widrow

Marcian Hoff

Marvin Minsky



Seymour Papert

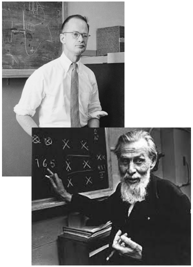
- 1969. Marvin Minsky questions the ability of the percetron

  [...] I started to worry about what such a machine could not do. [...] it could tell 'E's from 'F's, and '5's from '6's. But when there were disturbing stimuli near these figures that weren't correlated with them the recognition was destroyed.

Progression **Degression**

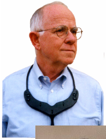
1943 1958 1962 1969

McCulloch and Pitts



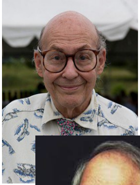
Rosenblatt

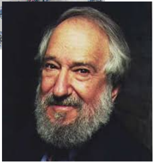
Bernard Widrow



Marcian Hoff

Marvin Minsky





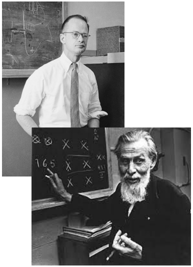
Seymour Papert

- 1969 (cont.). Together with Seymour Papert, Minsky writes the book "Perceptrons" that kills the perceptron. They prove that the perceptron is unable to learn the XOR function.
- Not clear yet how to train Multi-layers perceptrons.
- Research and funding go down.

Progression **Degression**

1943 1958 1962 1969

McCulloch and Pitts

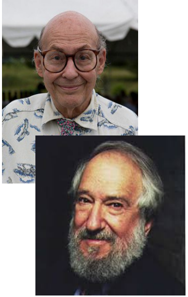


Rosenblatt

Bernard Widrow

Marcian Hoff

Marvin Minsky

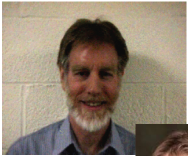


Seymour Papert

- (1963). In parallel to those more difficult times, the idea of backpropagation starts to appear (through the work of Arthur Bryson) but does not receive a lot of attention at the time.

Progression **Degression**

1986 1995 1998



Ronald Williams
David Rumelhart
Geoffrey Hinton

V.Vapnik
C.Cortes

Y. LeCun

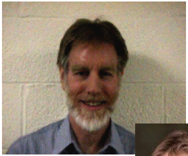
- 1986. The idea of backpropagation reappears through a paper

  Learning representations by back-propagation errors.

  published in Nature by Rumelhart, Williams and Hinton. Neural Networks with many hidden layers can be effectively trained by a relatively simple procedure. New extension to the perceptron (which had no ability to learn non linear functions)

Progression **Degression**

1986 1995 1998



Ronald Williams
David Rumelhart
Geoffrey Hinton

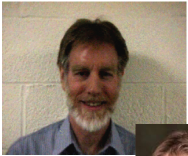
V.Vapnik
C.Cortes

Y. LeCun

- 1986. *Around the same time*, it is shown that neural networks have the ability to learn any function (Universal Approximation Theorem)
- Neural nets get back on track
- But there are still many open questions: Overfitting? Optimal structure (Number of neurons, layers) Bad local mins?

Progression **Degression**

1986 1995 1998



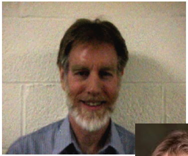
Ronald Williams
David Rumelhart
Geoffrey Hinton

V.Vapnik
C.Cortes

Y. LeCun

- (1995). Support Vector Machines are introduced by V. Vapnik and C. Cortes. SVMs have shallow architectures.
- Graphical models are becoming increasingly popular
- Together Graphical models and SVMs almost kill research on Artificial Neural Networks

Progression 1986 — **Degression** 1995 — 1998



Ronald Williams
David Rumelhart
Geoffrey Hinton

V.Vapnik
C.Cortes

Y. LeCun

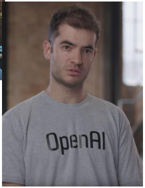
- Training deeper networks give poor results..
- (1998) LeCun introduces deep convolutional neural networks.

Progression

2006 2012

Y. Bengio
Ian Goodfellow

Alex Krizhevsky
Geoffrey Hinton
Ilya Sutskever



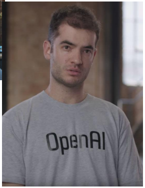
- (2006). Deep Learning appears as a rebranding of ANN
- (2006). Deep Belief Networks (Hinton et al.)
- (2007) Deep Autoencoders (Bengio et al.)

Progression

2006 2012

Y. Bengio
Ian Goodfellow

Alex Krizhevsky
Geoffrey Hinton
Ilya Sutskever

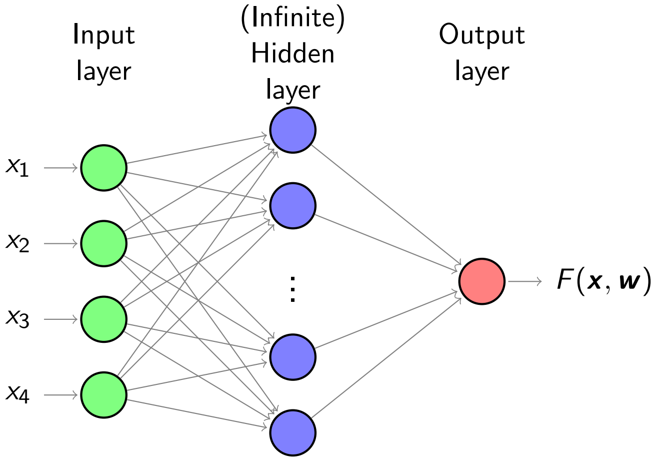


- Neural networks become increasingly popular following massive usage of GPUs
- (2012). This trend is illustrated by the use of AlexNet for image classification (Krizhevsky, Sutskever and Hinton)

Universal approximation

- For M sufficiently large, The simple Projection Pursuit Regression model (PPR) can approximate any function in $\mathbb{R}^p$.
- This result is known as the Universal Approximation Theorem
- The combination "non linear activation function" + "linear function of the inputs" is part of a class of functions called universal approximators

Universal approximation



Universal Approximation Theorem (Haykin 1994)

- Let $\phi(\cdot)$ denote a nonconstant, bounded and monotone-increasing continuous function.
- Let I_{m_0} denote the m_0 dimensional unit hypercube $[0,1]^{m_0}$.
- Let $\mathcal{C}(I_{m_0})$ denote the space of continuous functions on I_{m_0}.

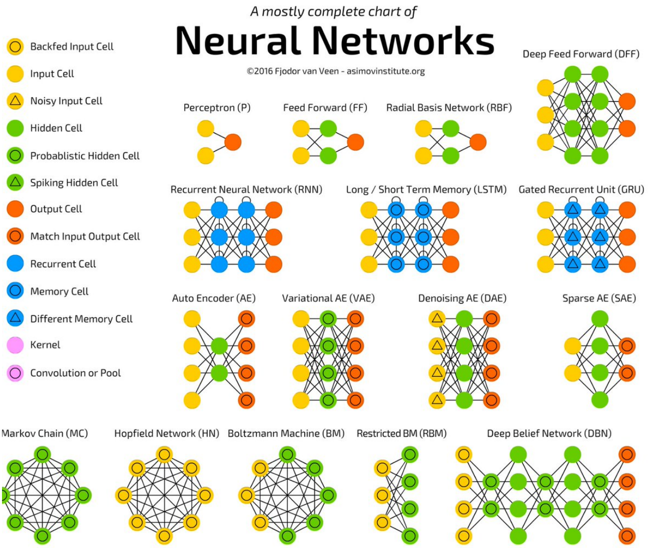
Then for any function $f \in \mathcal{C}(I_{m_0})$ and $\varepsilon > 0$, there exists an integer $\overline{M}$ and sets of real constants α_i, b_i and w_{ij} where $i = 1, \ldots, M$ and $J = 1, \ldots, d$ such that if we define

$$F(x_1, \ldots, x_d) = \sum_{i=1}^{\overline{M}} \alpha_i \phi \left(\sum_{j=1}^{d} w_{ij} x_j + b_i \right)$$

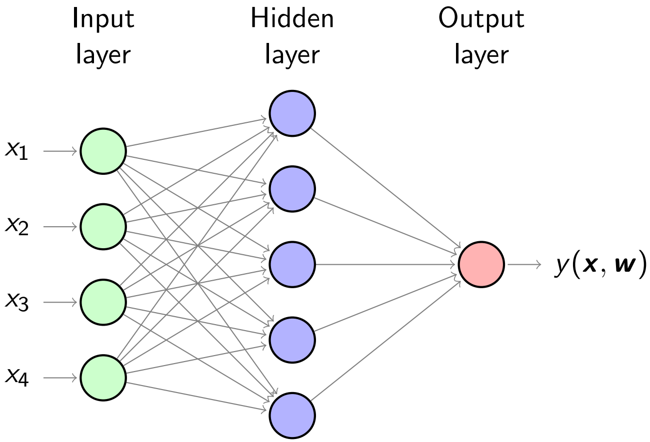
we have $$|F(x_1, \ldots, x_d) - f(x_1, \ldots, x_d)| < \varepsilon$$

for all $x_1, x_2, \ldots, x_d$ that lie in the input space.

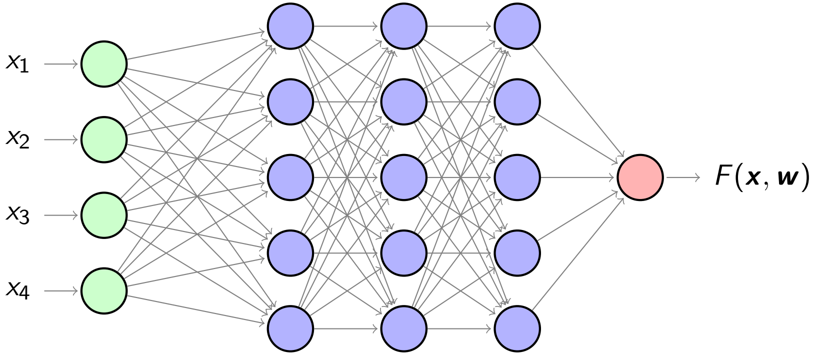
Many possible architectures

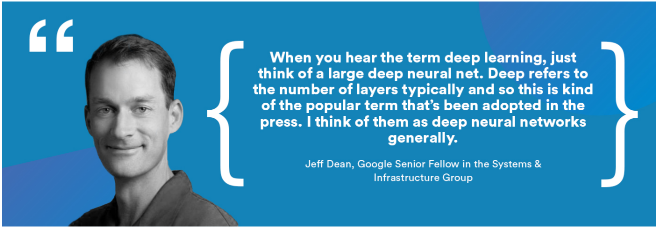


One (hidden) layer



Deep neural network





How do we train? (I)

- To train the network, we minimize the empirical risk function. For a given training set $\{\boldsymbol{x}_i, y_i\}$ and a network with weights $\boldsymbol{w}$, the loss/Empirical risk reads as (as usual there is a statistical intuition for that loss)

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N} \|y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i\|^2$$

- The general approach at minimizing functions such as $\ell(\boldsymbol{w})$ is to start from some initial value $\boldsymbol{w}$ and then follow the gradient to minimize E.

$$\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^{(k)} - \eta \nabla E(\boldsymbol{w}^{(k)})$$

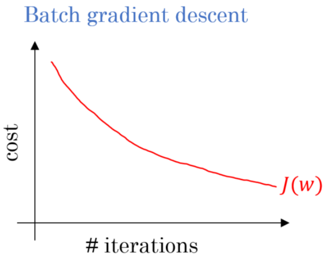
How do we train? (II)

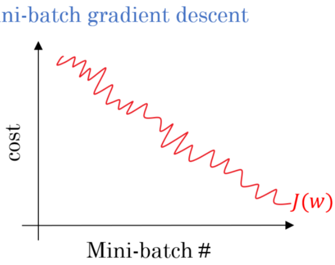
- Minimizing the empirical risk directly is often expensive because the *training* set of input-output pairs can be very large
- When dealing with practical problems, we will in general not apply gradient descent directly on those function.
- An alternative known as stochastic gradient descent or sequential gradient descent (due to LeCun) relies on the independence of the samples and view the empirical risk as a sum of N independent contributions.
- This approach then optimizes each of those terms sequentially rather than jointly resulting in iterations of the form

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla E_n(\boldsymbol{w}^{(k)}), \quad n = 1, \ldots, N.$$

How do we train? some vocabulary

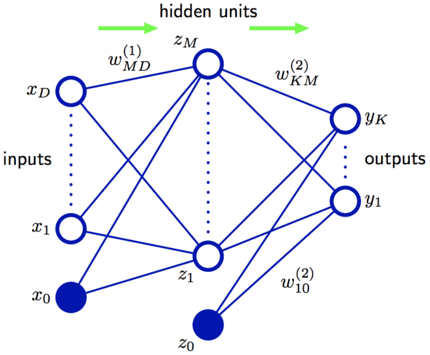
- Batch gradient descent = use all the data at once
- Minibatch = use subsets
- Epoch = one pass over the full training data





How do we train? Backpropagation

Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.



Bishop, Pattern Recognition and ML

- Consider the simple two layers neural net

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = h\left(\sum_{j=1}^{N} w_{k,j}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

How do we train? Backpropagation

- Computing the gradient of a complex nested function involving a large number of layers is painful.
- In practice, optimization relies on an idea called backpropagation. In backpropagation, the information is propagated through the network first forward and then backwards in order to update the weights.
- The method proceeds in two steps,
  - During the first step, the error vector containing the residuals is propagated backwards in the network to evaluate the derivatives
  - During the second step, the derivatives that were computed in the first step are used to update the weights.

How do we train? Backpropagation

- For an empirical risk function which reads as a sum of M independent contributions,

$$E = \sum_{m=1}^{M} E_m,$$

- In the sequential framework, we can focus on a single E_m. In a NN, each unit computes a weighted sum s_j of the inputs,

$$a_j = \sum_i w_{ji} z_i$$

  The sum is then transformed through the activation function h.

- Applying the chain rule, we get

$$\boxed{\frac{\partial E_m}{\partial w_{ji}} = \frac{\partial E_m}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}}$$

How do we train? Backpropagation

- Note that

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

- If we let E_m to denote the minbatch empirical risk function

$$E_m = \frac{1}{2} \sum_k (y_{n,k}(\boldsymbol{x}, \boldsymbol{w}) - t_{n,k})^2$$

- The gradient w.r.t the weights appearing in the last layer can thus read as

$$\frac{\partial E_m}{\partial a_k} = (y_{n,k} - t_{n,k}) = \delta_{n,k}$$

How do we train? Backpropagation

- Moreover, all the other derivatives w.r.t the $a_{n,\ell}$ (of layer ℓ) can be computed using the chain rule

$$\frac{\partial E_m}{\partial a_{n-1,\ell}} = \sum_{j=1}^{J} \frac{\partial E_m}{\partial a_{n,j}} \frac{\partial a_{n,j}}{\partial a_{n-1,\ell}}$$

- The relation between the inputs $a_{n,j}$ from the n^{th} layer and the inputs $a_{n-1,j}$ from the previous $(n-1)$ layer reads as

$$a_{n,k} = \langle \boldsymbol{w}, h(\boldsymbol{a}_{n-1}\rangle) = \sum_{j=1}^{J} w_{k,j}^{(n-1)} h\left(a_{n-1,j}\right)$$

How do we train? Backpropagation

- The relation between the inputs $a_{n,j}$ from the n^{th} layer and the inputs $a_{n-1,j}$ from the previous layer reads as

$$a_{n,k} = \langle \boldsymbol{w}, h(\boldsymbol{a}_{n-1}\rangle) = \sum_{j=1}^{J} w_{k,j}^{(n-1)} h\left(a_{n-1,j}\right)$$

- From this, we get the equation

$$\frac{\partial a_{n,k}}{\partial a_{n-1,j}} = \sum_{j=1}^{J} w_{k,j}^{(n-1)} h'(a_{n-1,j})$$

- Which we can substitute in the gradient $\partial_{a_{n-1,\ell}} E_m$

$$\delta_{n-1,\ell} = \frac{\partial E_m}{\partial a_{n-1,\ell}} = \sum_{j=1}^{J} \frac{\partial E_m}{\partial a_{n,j}} \frac{\partial a_{n,j}}{\partial a_{n-1,\ell}} = \sum_{j=1}^{J} h'(a_{n-1,j}) w_{k,j}^{(n-1)} \delta_{n,j}$$

Backpropagation (summary)

- Propagate the feature vectors from the training set forward and compute all the outputs to the activation functions $a_{\ell,j}$ as well as the derivatives $h'(a_j)$.
- Evaluate the output $\delta_{n,k} = (t_k - y_k)$
- Backpropagate those $\delta_{n,k}$ trough the chain rule
- Once you have the $\delta_{\ell,j}$ for all layers ℓ and indices j, compute the derivatives of the empirical risk by using

$$\frac{\partial E_m}{\partial w_{ij}} = \delta_j h(a_{n-1,i})$$