Machine learning from a pure mathematician's viewpoint

Roberto Rubio

Universitat Autònoma de Barcelona

Mathematical Foundations of Machine Learning: PDEs, Probability, and Dynamics

Centre de Recerca Matemàtica

8th January 2026

DISCLAIMER

Most of the characters and stories of this talk have appeared in the literature before. This talk recounts my own way of making sense of them. They are based on the supervision of final-year projects at the Autonomous University of Barcelona and especially on the graduate course 'Mathematics of machine learnings and machine learning for mathematics' that I am currently teaching while visiting the Weizmann Institute.

UAB
Universitat Autònoma
de Barcelona

WEIZMANN
INSTITUTE
OF SCIENCE

DISCLAIMER

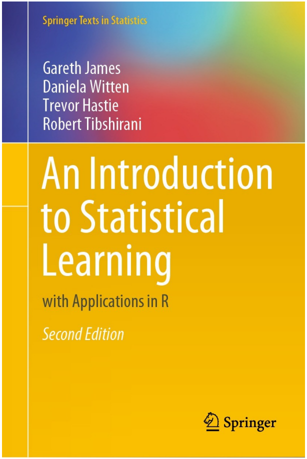
Most of the characters and stories of this talk have appeared in the literature before. This talk recounts my own way of making sense of them. They are based on the supervision of final-year projects at the Autonomous University of Barcelona and especially on the graduate course 'Mathematics of machine learnings and machine learning for mathematics' that I am currently teaching while visiting the Weizmann Institute.

UAB
Universitat Autònoma
de Barcelona

WEIZMANN
INSTITUTE
OF SCIENCE

Take this talk as a naive and biased overview.

Before the story, let me open a book...



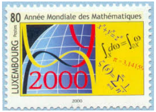
The Advertising data set consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper.

The input variables are typically denoted using the output variable symbol X, with a subscript to distinguish them. So X_1 might be the **TV budget**, X_2 the **radio budget**, and X_3 the **newspaper budget**.

The output variable—in this case, **sales**—

...and review 2000–2021

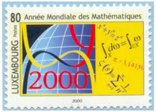
...and review 2000–2021



WIKIPEDIA
Die freie Enzyklopädie

Weltjahr der Mathematik 2000

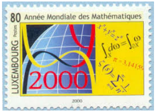
...and review 2000–2021

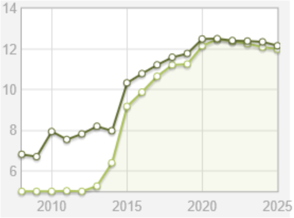


Wikipedia
Die freie Enzyklopädie
Weltjahr der Mathematik 2000

GPA 5/10 to study maths

...and review 2000–2021

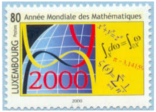




WIKIPEDIA
Die freie Enzyklopädie
Weltjahr der Mathematik 2000

GPA 5/10 to study maths

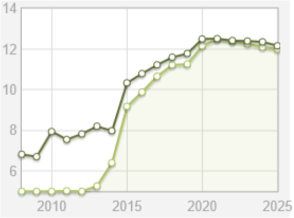
...and review 2000–2021



WIKIPEDIA
Die freie Enzyklopädie

Weltjahr der Mathematik 2000

GPA 5/10 to study maths



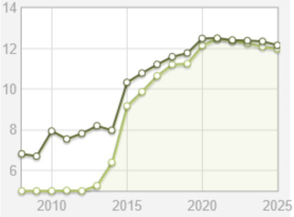
No interest in predicting sales.

...and review 2000–2021

WIKIPEDIA
Die freie Enzyklopädie

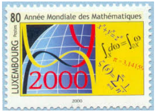
Weltjahr der Mathematik 2000

GPA 5/10 to study maths



No interest in predicting sales.
Nor machine learning until...

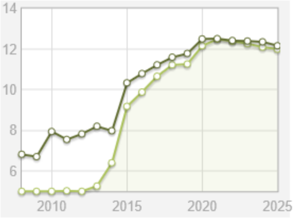
...and review 2000–2021



WIKIPEDIA
Die freie Enzyklopädie

Weltjahr der Mathematik 2000

GPA 5/10 to study maths



No interest in predicting sales.
Nor machine learning until... 2021

A subtle difference in the viewpoint matters

In the book we opened:

investment in ads lives in $X = \mathbb{R}^3$,

sales output lives in $Y = \mathbb{R}$,

we would like to learn a **distribution** $p(X \times Y)$ or $p(Y|X)$.

A subtle difference in the viewpoint matters

In the book we opened:

investment in ads lives in $X = \mathbb{R}^3$,

sales output lives in $Y = \mathbb{R}$,

we would like to learn a **distribution** $p(X \times Y)$ or $p(Y|X)$.

Instead, tell a pure mathematician:

YOU: it is better to talk about distributions, but let us say for today that we have samples $(x, g(x))$ of a **function** $g : X \to Y$ and want to learn g.

A subtle difference in the viewpoint matters

In the book we opened:

investment in ads lives in $X = \mathbb{R}^3$,

sales output lives in $Y = \mathbb{R}$,

we would like to learn a **distribution** $p(X \times Y)$ or $p(Y|X)$.

Instead, tell a pure mathematician:

YOU: it is better to talk about distributions, but let us say for today that we have samples $(x, g(x))$ of a **function** $g : X \to Y$ and want to learn g.
THE MATHEMATICIAN: do you mean interpolating the samples?
YOU: rather approximating, let us use neural networks.
For computability and access to tools, regard $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$ and learn

$$g : \mathbb{R}^n \to \mathbb{R}^m.$$

A subtle difference in the viewpoint matters

In the book we opened:

investment in ads lives in $X = \mathbb{R}^3$,

sales output lives in $Y = \mathbb{R}$,

we would like to learn a **distribution** $p(X \times Y)$ or $p(Y|X)$.

Instead, tell a pure mathematician:

YOU: it is better to talk about distributions, but let us say for today that we have samples $(x, g(x))$ of a **function** $g : X \to Y$ and want to learn g.
THE MATHEMATICIAN: do you mean interpolating the samples?
YOU: rather approximating, let us use neural networks.
For computability and access to tools, regard $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$ and learn

$$g : \mathbb{R}^n \to \mathbb{R}^m.$$

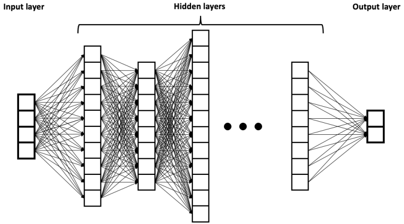
THE MATHEMATICIAN: what is a neural network?

What is a neural network

"*A neural network is a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use.*" (Haykin'94)

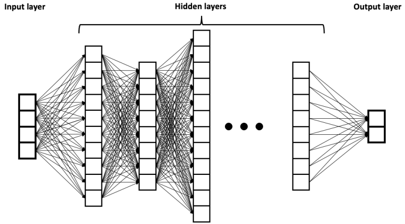
What is a neural network

"*A neural network is a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use.*" (Haykin'94)



What is a neural network for a mathematician

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use." (Haykin'94)



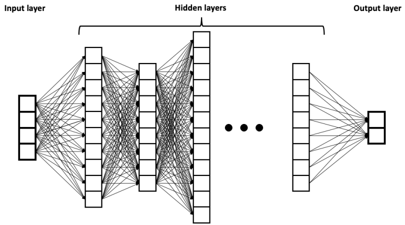
Definition

A neural network with activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is an alternating composition of affine functions h_i and component-wise σ.

$$\mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^{n_2} \xrightarrow{\sigma} \dots \xrightarrow{h_{L-1}} \mathbb{R}^{n_{L-1}} \xrightarrow{\sigma} \mathbb{R}^{n_{L-1}} \xrightarrow{h_L} \mathbb{R}^m$$

What is a neural network for a mathematician

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use." (Haykin'94)



Definition

A neural network with activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is an alternating composition of affine functions h_i and component-wise σ.

$$\mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^{n_2} \xrightarrow{\sigma} \dots \xrightarrow{h_{L-1}} \mathbb{R}^{n_{L-1}} \xrightarrow{\sigma} \mathbb{R}^{n_{L-1}} \xrightarrow{h_L} \mathbb{R}^m$$

Once σ is fixed, it depends on $P := \sum_{i=1}^{L}(n_{i-1}+1)n_i$ parameters.

How does σ look?

How does σ look?

How well does the class of neural networks approximate?

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), \| \ \|_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), || \, ||_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), || \, ||_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

Cybenko'83: dense for σ continuous, $\lim_{t \to -\infty} \sigma(t) = 0$, $\lim_{t \to +\infty} \sigma(t) = 1$.

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), \|\ \|_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

Cybenko'83: dense for σ continuous, $\lim_{t\to-\infty} \sigma(t) = 0$, $\lim_{t\to+\infty} \sigma(t) = 1$.

Hornik'91: dense for σ **continuous**, bounded and non-constant.

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), \|\ \|_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

Cybenko'83: dense for σ continuous, $\lim_{t\to-\infty} \sigma(t) = 0$, $\lim_{t\to+\infty} \sigma(t) = 1$.

Hornik'91: dense for σ **continuous**, bounded and non-constant.

Theorem (Leshno-Lin-Pinkus-Shockem'93)

For $\sigma \in L^\infty_{loc}$ with null set of discontinuities, neural networks are dense in $\mathcal{C}(K)$ **if and only if** *σ is a non-polynomial function almost everywhere.*

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), \| \|_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

Cybenko'83: dense for σ continuous, $\lim_{t\to-\infty} \sigma(t) = 0$, $\lim_{t\to+\infty} \sigma(t) = 1$.

Hornik'91: dense for σ **continuous**, bounded and non-constant.

Theorem (Leshno-Lin-Pinkus-Shockem'93)

For $\sigma \in L^\infty_{loc}$ with null set of discontinuities, neural networks are dense in $\mathcal{C}(K)$ **if and only if** *σ is a non-polynomial function almost everywhere.*

But precisely polynomials can approximate any function (Weierstrass)!

How well does the class of neural networks approximate?

We focus on $(\mathcal{C}(K), \|\ \|_\infty)$ for compact $K \subset \mathbb{R}^n$ and specify σ.

Two layers, $f : \mathbb{R}^n \xrightarrow{h_1} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{h_2} \mathbb{R}^m$, is enough!

Cybenko'83: dense for σ continuous, $\lim_{t\to-\infty} \sigma(t) = 0$, $\lim_{t\to+\infty} \sigma(t) = 1$.

Hornik'91: dense for σ **continuous**, bounded and non-constant.

Theorem (Leshno-Lin-Pinkus-Shockem'93)

For $\sigma \in L^\infty_{loc}$ with null set of discontinuities, neural networks are dense in $\mathcal{C}(K)$ **if and only if** *σ is a non-polynomial function almost everywhere.*

But precisely polynomials can approximate any function (Weierstrass)!

Remarks: beautiful proof, also results for $L^p(\mu)$, $\textit{citations} \cdot \textit{generality} = k$.

Why deep learning?

Pinkus'99 (who, incidentally, cites Corominas, Sunyer i Balaguer'54):

> Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood. Some authors see little theoretical gain in considering more than one hidden layer since a single hidden layer model suffices for density. Most authors, however, do allow for the possibility of certain other benefits to be gained from using more than one hidden layer. (See de Villiers and Barnard (1992) for a comparison of these two models.)

Why deep learning?

Pinkus'99 (who, incidentally, cites Corominas, Sunyer i Balaguer'54):

> Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood. Some authors see little theoretical gain in considering more than one hidden layer since a single hidden layer model suffices for density. Most authors, however, do allow for the possibility of certain other benefits to be gained from using more than one hidden layer. (See de Villiers and Barnard (1992) for a comparison of these two models.)

Daniely'17: "with more layers we need fewer neurons".

Why deep learning?

Pinkus'99 (who, incidentally, cites Corominas, Sunyer i Balaguer'54):

> Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood. Some authors see little theoretical gain in considering more than one hidden layer since a single hidden layer model suffices for density. Most authors, however, do allow for the possibility of certain other benefits to be gained from using more than one hidden layer. (See de Villiers and Barnard (1992) for a comparison of these two models.)

Daniely'17: "with more layers we need fewer neurons".

Theorem (Kidgers, Lyon'20)

Neural networks with $n_i \leq n + m + 2$ and σ a nonaffine continuous function which is continuously differentiable with nonzero derivative at at least one point are dense in $(\mathcal{C}(K), \|\ \|_\infty)$.

How to approximate? Let us start with $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$.

How to approximate? Let us start with $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$.

$f_w : \mathbb{R}^n \to \mathbb{R}^m$ depending on $w \in \mathbb{R}^P$, set

$$f_x : \mathbb{R}^P \to \mathbb{R}^m$$
$$f_x(w) := f_w(x).$$

How to approximate? Let us start with $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$.

$f_w : \mathbb{R}^n \to \mathbb{R}^m$ depending on $w \in \mathbb{R}^P$, set

$$\begin{aligned} f_x &: \mathbb{R}^P \to \mathbb{R}^m \\ f_x(w) &:= f_w(x). \end{aligned}$$

Measure how well we do by taking a semidistance to y, $d_y : \mathbb{R}^m \to \mathbb{R}$.

Goal: adjust $w \in \mathbb{R}^P$ to minimize the loss $\ell := d_y \circ f_x : \mathbb{R}^P \to \mathbb{R}$.

How to approximate? Let us start with $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$.

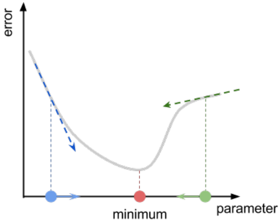
$f_w : \mathbb{R}^n \to \mathbb{R}^m$ depending on $w \in \mathbb{R}^P$, set

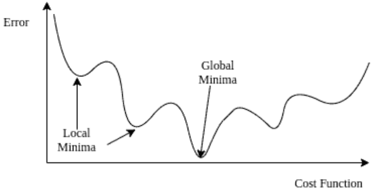
$$f_x : \mathbb{R}^P \to \mathbb{R}^m$$
$$f_x(w) := f_w(x).$$

Measure how well we do by taking a semidistance to y, $d_y : \mathbb{R}^m \to \mathbb{R}$.

Goal: adjust $w \in \mathbb{R}^P$ to minimize the loss $\ell := d_y \circ f_x : \mathbb{R}^P \to \mathbb{R}$.

How? Gradient descent, $w' \leftarrow w - step \cdot \nabla \ell(w)$.





$(\nabla \ell)(w) = d_y'(f_x(w)) \cdot (\mathrm{Jac}\, f_x)(w)$, so let us compute $(\mathrm{Jac}\, f_x)(w)$

"Backpropagation is the chain rule"

Wikipedia:

Given an input–output pair (x, y), the loss is:

$$C(y, f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2 f^1(W^1 x)) \cdots)))$$

To compute this, one starts with the input x and works forward; denote the weighted input of each hidden layer as z^l and the output of hidden layer l as the activation a^l. For backpropagation, the activation a^l as well as the derivatives $(f^l)'$ (evaluated at z^l) must be cached for use during the backwards pass.

The derivative of the loss in terms of the inputs is given by the chain rule; note that each term is a total derivative, evaluated at the value of the network (at each node) on the input x:

$$\frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdot \ldots \cdot \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x},$$

where $\dfrac{da^L}{dz^L}$ is a diagonal matrix.

These terms are: the derivative of the loss function;[d] the derivatives of the activation functions;[e] and the matrices of weights:[f]

$$\frac{dC}{da^L} \circ (f^L)' \cdot W^L \circ (f^{L-1})' \cdot W^{L-1} \circ \cdots \circ (f^1)' \cdot W^1.$$

The gradient ∇ is the transpose of the derivative of the output in terms of the input, so the matrices are transposed and the order of multiplication is reversed, but the entries are the same:

$$\nabla_x C = (W^1)^T \cdot (f^1)' \circ \ldots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C.$$

Backpropagation, an example

$$f_x(a, b, p, q, r, s) = r\sigma(p\sigma(ax + b) + q) + s,$$

Denote f_x by f:

$$\begin{aligned}
\frac{\partial f}{\partial s} &= 1, & \frac{\partial f}{\partial r} &= \sigma(p\sigma(ax + b) + q)) \\
\frac{\partial f}{\partial q} &= r\sigma'(p\sigma(ax + b)) + q), & \frac{\partial f}{\partial p} &= r\sigma'(p\sigma(ax + b) + q))\sigma(ax + b) \\
\frac{\partial f}{\partial b} &= r\sigma'(p\sigma(ax + b)) + q)p\sigma'(ax + b), & \frac{\partial f}{\partial a} &= r\sigma'(p\sigma(ax + b) + q))p\sigma'(ax + b)x
\end{aligned}$$

Backpropagation, an example

$$f_x(a, b, p, q, r, s) = r\sigma(p\sigma(ax + b) + q) + s,$$

Denote f_x by f:

$$\frac{\partial f}{\partial s} = 1, \qquad \frac{\partial f}{\partial r} = \frac{\partial f}{\partial s}\sigma(p\sigma(ax + b) + q))$$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial s} r\sigma'(p\sigma(ax + b)) + q), \qquad \frac{\partial f}{\partial p} = \frac{\partial f}{\partial q}\sigma(ax + b)$$

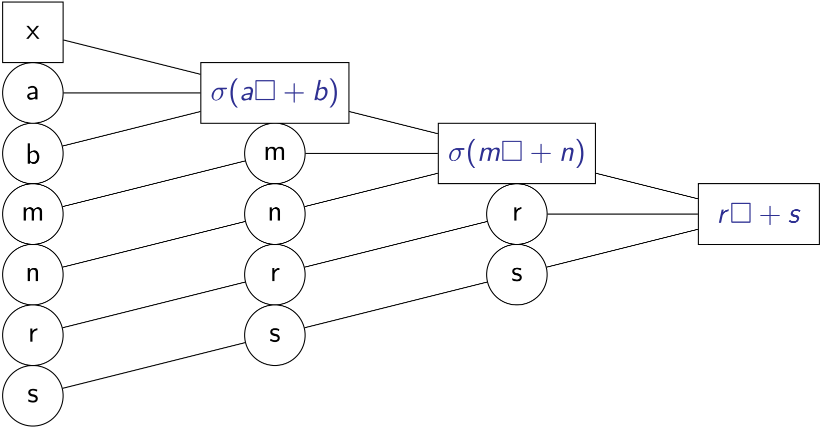
$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial q} p\sigma'(ax + b), \qquad \frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}x.$$

Backpropagation is the chain rule...

...but of $f_x(a, b, p, q, r, s)$ and making use of affine functions.

Backpropagation is the chain rule...

...but of $f_x(a, b, p, q, r, s)$ and making use of affine functions.



$(\mathrm{Jac}\, f_x)(w)$ is an $m \times P$ matrix

Understanding Machine Learning: From Theory to Algorithms by Shalev-Shwartz and Ben-David:

$$g_t(W_{t-1}) = \ell_t(\mathbf{o}_t) = \ell_t(\boldsymbol{\sigma}(\mathbf{a}_t)) = \ell_t(\boldsymbol{\sigma}(W_{t-1}\mathbf{o}_{t-1})).$$

It would be convenient to rewrite this as follows. Let $\mathbf{w}_{t-1} \in \mathbb{R}^{k_{t-1}k_t}$ be the column vector obtained by concatenating the rows of W_{t-1} and then taking the transpose of the resulting long vector. Define by O_{t-1} the $k_t \times (k_{t-1}k_t)$ matrix

$$O_{t-1} \;=\; \begin{pmatrix} \mathbf{o}_{t-1}^\top & 0 & \cdots & 0 \\ 0 & \mathbf{o}_{t-1}^\top & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{o}_{t-1}^\top \end{pmatrix}. \tag{20.2}$$

Then, $W_{t-1}\mathbf{o}_{t-1} = O_{t-1}\mathbf{w}_{t-1}$, so we can also write

$$g_t(\mathbf{w}_{t-1}) \;=\; \ell_t(\boldsymbol{\sigma}(O_{t-1}\,\mathbf{w}_{t-1})).$$

Therefore, applying the chain rule, we obtain that

$$J_{\mathbf{w}_{t-1}}(g_t) \;=\; J_{\boldsymbol{\sigma}(O_{t-1}\mathbf{w}_{t-1})}(\ell_t)\,\mathrm{diag}(\boldsymbol{\sigma}'(O_{t-1}\mathbf{w}_{t-1}))\,O_{t-1}.$$

Using our notation we have $\mathbf{o}_t = \boldsymbol{\sigma}(O_{t-1}\mathbf{w}_{t-1})$ and $\mathbf{a}_t = O_{t-1}\mathbf{w}_{t-1}$, which yields

$$J_{\mathbf{w}_{t-1}}(g_t) = J_{\mathbf{o}_t}(\ell_t)\,\mathrm{diag}(\boldsymbol{\sigma}'(\mathbf{a}_t))\,O_{t-1}.$$

Let us also denote $\boldsymbol{\delta}_t = J_{\mathbf{o}_t}(\ell_t)$. Then, we can further rewrite the preceding as

$$J_{\mathbf{w}_{t-1}}(g_t) = \left(\delta_{t,1}\,\sigma'(a_{t,1})\,\mathbf{o}_{t-1}^\top\;,\;\ldots\;,\;\delta_{t,k_t}\,\sigma'(a_{t,k_t})\,\mathbf{o}_{t-1}^\top\right). \tag{20.3}$$

Backpropagation, we had

$$f_x(a, b, p, q, r, s) = r\sigma(p\sigma(ax + b) + q) + s,$$

Denote f_x by f:

$$\frac{\partial f}{\partial s} = 1, \qquad \frac{\partial f}{\partial r} = \frac{\partial f}{\partial s}\sigma(p\sigma(ax + b) + q))$$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial s} r\sigma'(p\sigma(ax + b)) + q), \qquad \frac{\partial f}{\partial p} = \frac{\partial f}{\partial q}\sigma(ax + b)$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial q} p\sigma'(ax + b), \qquad \frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}x.$$

Backpropagation, we have

$$f_x(A, B, P, Q, R, S) = R\sigma(P\sigma(Ax + B) + Q) + S$$

Denote f_x by f, now A, B, P, Q, R, S matrices:

$$\begin{aligned}
\mathrm{Jac}_S\, f &= \mathrm{Id}, & \mathrm{Jac}_R\, f &= \mathrm{Jac}_S\, f \cdot diag_m \sigma(P\sigma(Ax + B) + Q)) \\
\mathrm{Jac}_Q\, f &= \mathrm{Jac}_S\, f \cdot R \cdot \sigma'(P\sigma(Ax + B)) + Q), & \mathrm{Jac}_P\, f &= \mathrm{Jac}_Q\, f \cdot diag_{n_2}(\sigma(Ax + B)), \\
\mathrm{Jac}_B\, f &= \mathrm{Jac}_Q\, f \cdot P \cdot \sigma'(Ax + B), & \mathrm{Jac}_A\, f &= Jac_B f \cdot diag_{n_1}(x).
\end{aligned}$$

Backpropagation: embrace tensors

$$f_x(A, B, P, Q, R, S) = R\sigma(P\sigma(Ax + B) + Q) + S$$

Denote f_x by f, now A, B, P, Q, R, S matrices:

$$\begin{aligned}
\mathrm{Jac}_S\, f &= \mathrm{Id}, & \mathrm{Jac}_R\, f &= \mathrm{Jac}_S\, f \cdot diag_m \sigma(P\sigma(Ax+B)+Q)) \\
\mathrm{Jac}_Q\, f &= \mathrm{Jac}_S\, f \cdot R \cdot \sigma'(P\sigma(Ax+B)) + Q), & \mathrm{Jac}_P\, f &= \mathrm{Jac}_Q\, f \cdot diag_{n_2}(\sigma(Ax+B)), \\
\mathrm{Jac}_B\, f &= \mathrm{Jac}_Q\, f \cdot P \cdot \sigma'(Ax+B), & \mathrm{Jac}_A\, f &= Jac_B f \cdot diag_{n_1}(x).
\end{aligned}$$

$\mathrm{Jac}_R\, f$ is a $m \times m \times n_2$ tensor, product of $m \times m$ and $m \times m \times n_2$ tensors.

Backpropagation: embrace tensors

$$f_x(A, B, P, Q, R, S) = R\sigma(P\sigma(Ax + B) + Q) + S$$

Denote f_x by f, now A, B, P, Q, R, S matrices:

$$\begin{aligned}
&\mathrm{Jac}_S\, f = \mathrm{Id}, && \mathrm{Jac}_R\, f = \mathrm{Jac}_S\, f \cdot diag_m \sigma(P\sigma(Ax + B) + Q)) \\
&\mathrm{Jac}_Q\, f = \mathrm{Jac}_S\, f \cdot R \cdot \sigma'(P\sigma(Ax + B)) + Q), && \mathrm{Jac}_P\, f = \mathrm{Jac}_Q\, f \cdot diag_{n_2}(\sigma(Ax + B)), \\
&\mathrm{Jac}_B\, f = \mathrm{Jac}_Q\, f \cdot P \cdot \sigma'(Ax + B), && \mathrm{Jac}_A\, f = Jac_B f \cdot diag_{n_1}(x).
\end{aligned}$$

$\mathrm{Jac}_R\, f$ is a $m \times m \times n_2$ tensor, product of $m \times m$ and $m \times m \times n_2$ tensors.

Two reasons: mathematically sound and matrix multiplication algorithms

YOU: Now you can do this at many points (stochastically) and it works.
THE MATHEMATICIAN: The function only matches the sample, not even

YOU: Now you can do this at many points (stochastically) and it works.
THE MATHEMATICIAN: The function only matches the sample, not even

YOU: Learn just 80 % of your sample, test the other 20%.
THE MATHEMATICIAN: Still, the function is hell, what do I get from it?

$$f : \mathbb{R}^n \to \mathbb{R}$$

YOU: Now you can do this at many points (stochastically) and it works.
THE MATHEMATICIAN: The function only matches the sample, not even

YOU: Learn just 80 % of your sample, test the other 20%.
THE MATHEMATICIAN: Still, the function is hell, what do I get from it?

$$f : \mathbb{R}^n \to \mathbb{R}$$

n features, what is their relevance? If f were linear, look at the coefficients.

Attribution: understanding supervised learning

YOU: Now you can do this at many points (stochastically) and it works.
THE MATHEMATICIAN: The function only matches the sample, not even

YOU: Learn just 80 % of your sample, test the other 20%.
THE MATHEMATICIAN: Still, the function is hell, what do I get from it?

$$f : \mathbb{R}^n \to \mathbb{R}$$

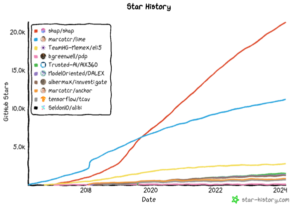
n features, what is their relevance? If f were linear, look at the coefficients.

Attribution methods (quite new!).

2013 Saliency for images
(Gradient-based methods)

2016 LIME (Local Interpretability
Model-Agnostic Explainability)

2017 SHAP (Shapley Additive exPlanation)



Shapley values

Based on cooperative games for a set of players $N = \{1, \ldots, n\}$,

$$v : 2^N \to \mathbb{R}.$$

How to distribute the worth $v(N)$? With a value function $\varphi(v) : N \to \mathbb{R}$

Shapley values

Based on cooperative games for a set of players $N = \{1, \ldots, n\}$,

$$v : 2^N \to \mathbb{R}.$$

How to distribute the worth $v(N)$? With a value function $\varphi(v) : N \to \mathbb{R}$

Shapley wanted $\varphi : \mathit{Games}(N) \to \mathbb{R}^N$ such that (with $\varphi_i(v) := \varphi(v)(i)$)

- $\sum_{i \in N} \varphi_i(v) = v(N)$ (efficiency),
- if $v(S \cup \{i\}) = v(S \cup \{j\})$ for all $S \subseteq N \setminus \{i, j\}$, then $\varphi_i(v) = \varphi_j(v)$ (symmetry),
- if $v(S) = v(S \cup \{i\})$ for every $S \subseteq N \setminus \{i\}$, then $\varphi_i(v) = 0$ (null),
- $\varphi(u + v) = \varphi(u) + \varphi(v)$ for any $u, v \in \mathit{Games}(N)$ (additivity).

Theorem (Shapley'53)

There is only one such function.

Denote the permutations of the set N by $\Pi(N)$ and define

$$N_i^\pi := \{j \in N \mid \pi(j) < \pi(i).\}$$

The only solution is

$$\varphi_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi(N)} (v(N_i^\pi \cup \{i\}) - v(N_i^\pi)).$$

Denote the permutations of the set N by $\Pi(N)$ and define

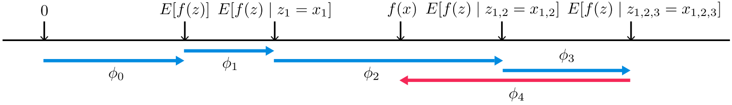
$$N_i^\pi := \{j \in N \mid \pi(j) < \pi(i).\}$$

The only solution is

$$\varphi_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi(N)} (v(N_i^\pi \cup \{i\}) - v(N_i^\pi)).$$

What game do we play? Just at a point:

$$v_x(S) := \mathbb{E}[f(x_S)],$$



Denote the permutations of the set N by $\Pi(N)$ and define

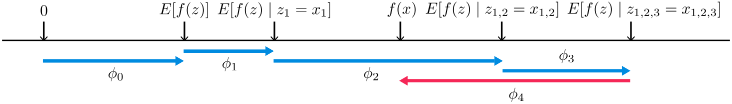
$$N_i^\pi := \{j \in N \mid \pi(j) < \pi(i).\}$$

The only solution is

$$\varphi_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi(N)} (v(N_i^\pi \cup \{i\}) - v(N_i^\pi)).$$

What game do we play? Just at a point:

$$v_x(S) := \mathbb{E}[f(x_S)],$$



Additivity can be replaced by strong monotonicity (Young'85),

$$v(S \cup \{i\}) - v(S) \geq w(S \cup \{i\}) - w(S) \text{ for all } S \Rightarrow \varphi_i(v) \geq \varphi_i(w),$$

Better? Determined by the **game potential** (Hart,Mas-Colell'88)

But everyone uses reinforcement learning!

But everyone uses reinforcement learning!

That is just:

- a set $\mathcal{S}$, of states
- a set $\mathcal{A}$, of actions
- for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, a probability distribution on $\mathcal{S} \times \mathbb{R}$, of new state and (bounded) reward.

$$p(s', r|s, a).$$

But everyone uses reinforcement learning!

That is just:

- a set $\mathcal{S}$, of states
- a set $\mathcal{A}$, of actions
- for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, a probability distribution on $\mathcal{S} \times \mathbb{R}$, of new state and (bounded) reward.

$$p(s', r|s, a).$$

PROBLEM: choose a probability distribution on $\mathcal{A}$ for each $\mathcal{S}$

$$\pi(a|s),$$

such that if the A_i follow π

$$s \xrightarrow{A_0} R_1, S_1 \xrightarrow{A_1} R_2, S_2 \xrightarrow{A_3} \dots R_k, S_k \xrightarrow{A_k} R_{k+1}, S_{k+1} \xrightarrow{A_{k+1}} \dots$$

has maximum expected return (discount rate $0 < \gamma < 1$), for every s,

$$\mathbb{E}\left(\sum_{i=1}^{+\infty} \gamma^i R_i\right).$$

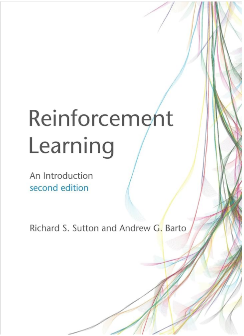
Theory and opening another book

Assume $\mathcal{S}$ and $\mathcal{A}$ to be finite. Define

$$v_\pi(s) := \mathbb{E}\left(\sum_{i=1}^{+\infty} \gamma^i R_i S_0 = s\right).$$

We aim for π^* such that, for every π and any s,

$$v_{\pi^*}(s) \geq v_\pi(s).$$



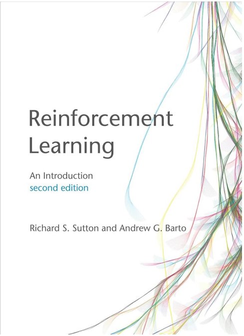
Theory and opening another book

Assume $\mathcal{S}$ and $\mathcal{A}$ to be finite. Define

$$v_\pi(s) := \mathbb{E}\left(\sum_{i=1}^{+\infty} \gamma^i R_i S_0 = s\right).$$

We aim for π^* such that, for every π and any s,

$$v_{\pi^*}(s) \geq v_\pi(s).$$



There is always at least one policy that is better than or equal to all other policies (page 62 of 524).

The existence and uniqueness of v are guaranteed as long as either < 1 or eventual termination is guaranteed from all states under the policy (page 74).

Essentially, only two theorems in 524 pages.

For a mathematician

Two-line deduction: Bellman's equation.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')).$$

It is a linear system of equations that computes the values of a policy.

For a mathematician

Two-line deduction: Bellman's equation.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')).$$

It is a linear system of equations that computes the values of a policy.

If the optimal policy exists, it must satisfy Bellman's optimality equation

$$v_*(s) = \max_{a \in \mathcal{A}} \left(\sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')) \right).$$

For a mathematician

Two-line deduction: Bellman's equation.

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')).$$

It is a linear system of equations that computes the values of a policy.

If the optimal policy exists, it must satisfy Bellman's optimality equation

$$v_*(s) = \max_{a\in\mathcal{A}} \left(\sum_{s',r} p(s', r|s, a)(r + \gamma v_*(s')) \right).$$

For an arbitrary 'state-value' function $f : \mathcal{S} \to \mathbb{R}$, define the operator:

$$(\mathcal{T} f)(s) = \max_{a\in\mathcal{A}} \sum_{s',r} p(s', r|s, a)(r + \gamma f(s')).$$

For a mathematician

Two-line deduction: Bellman's equation.

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s',r} p(s',r|s,a)(r+\gamma v_\pi(s')).$$

It is a linear system of equations that computes the values of a policy.

If the optimal policy exists, it must satisfy Bellman's optimality equation

$$v_*(s) = \max_{a\in\mathcal{A}} \left(\sum_{s',r} p(s',r|s,a)(r+\gamma v_*(s')) \right).$$

For an arbitrary 'state-value' function $f : \mathcal{S} \to \mathbb{R}$, define the operator:

$$(\mathcal{T}f)(s) = \max_{a\in\mathcal{A}} \sum_{s',r} p(s',r|s,a)(r+\gamma f(s')).$$

Theorem (Banach's fixed point theorem)

A contractive operator on a complete metric space has a fixed point, which is unique and can be computed iteratively.

But a neural network may be able to learn the policy!

WARNING: The optimal policy is actually deterministic (go to highest-valued state) but π allows us to explore options.

But a neural network may be able to learn the policy!

WARNING: The optimal policy is actually deterministic (go to highest-valued state) but π allows us to explore options.

So we also want to learn a function

$$\mathcal{S} \to \mathit{Dist}(\mathcal{A})$$

YOU: I told you that the probabilistic approach is better!

But a neural network may be able to learn the policy!

WARNING: The optimal policy is actually deterministic (go to highest-valued state) but π allows us to explore options.

So we also want to learn a function

$$\mathcal{S} \to \mathit{Dist}(\mathcal{A})$$

YOU: I told you that the probabilistic approach is better!

TM: Regard $\mathit{Dist}(\mathcal{A})$ as almost $\mathbb{R}^{|A|}$ via $\mathrm{softmax} : \mathbb{R}^m \to \mathit{Dist}(\{1, \ldots, m\})$

$$\mathrm{softmax}(y_1, \ldots, y_m) = \frac{1}{\sum_{i=1}^{n} e^{y_i}} \left(e^{y_1}, \ldots, e^{y_m}\right).$$

and I can understand reinforcement learning as supervised learning where the samples are evolving while playing the game.

But a neural network may be able to learn the policy!

WARNING: The optimal policy is actually deterministic (go to highest-valued state) but π allows us to explore options.

So we also want to learn a function

$$\mathcal{S} \to \mathit{Dist}(\mathcal{A})$$

YOU: I told you that the probabilistic approach is better!

TM: Regard $\mathit{Dist}(\mathcal{A})$ as almost $\mathbb{R}^{|A|}$ via $\mathrm{softmax} : \mathbb{R}^m \to \mathit{Dist}(\{1, \ldots, m\})$

$$\mathrm{softmax}(y_1, \ldots, y_m) = \frac{1}{\sum_{i=1}^n e^{y_i}} \left(e^{y_1}, \ldots, e^{y_m}\right).$$

and I can understand reinforcement learning as supervised learning where the samples are evolving while playing the game.

YOU: you'd better use cross-entropy here...

Cross-entropy $<<$ 101

Say $(1, 0) \in Dist(Y)$ is approximated by $(p, 1-p)$.

The Mean-Squared Error gives

$$\frac{1}{2}((1-p)^2 + (0-(1-p))^2) = (1-p)^2.$$

Cross-entropy $<<$ 101

Say $(1,0) \in \mathit{Dist}(Y)$ is approximated by $(p, 1-p)$.

The Mean-Squared Error gives

$$\frac{1}{2}((1-p)^2 + (0-(1-p))^2) = (1-p)^2.$$

Instead do

$$-1 \cdot \log p - 0 \log(1-p) = -\log p.$$

Cross-entropy $<<$ 101

Say $(1,0) \in \mathit{Dist}(Y)$ is approximated by $(p, 1-p)$.

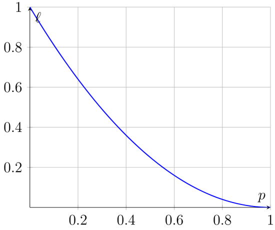
The Mean-Squared Error gives

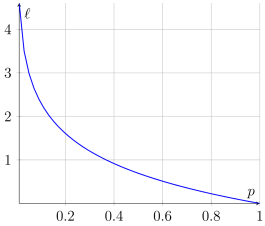
$$\frac{1}{2}((1-p)^2 + (0-(1-p))^2) = (1-p)^2.$$

Instead do

$$-1 \cdot \log p - 0 \log(1-p) = -\log p.$$

This way we punish confidently wrong predictions:





Cross-entropy << 101

Say $(1, 0) \in \mathit{Dist}(Y)$ is approximated by $(p, 1-p)$.

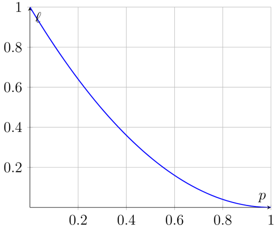
The Mean-Squared Error gives

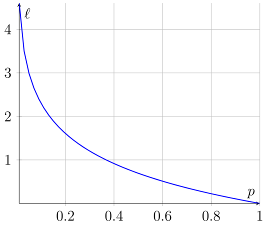
$$\frac{1}{2}((1-p)^2 + (0-(1-p))^2) = (1-p)^2.$$

Instead do

$$-1 \cdot \log p - 0 \log(1-p) = -\log p.$$

This way we punish confidently wrong predictions:





Simplifications and toy models help us understand each other.

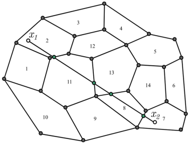
Mathematical understanding as a bridge to join forces

Mathematical understanding as a bridge to join forces

b=M2L

about 2026 2025 2024 2023 our library sign up

colloquia'26



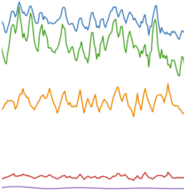
adi shamir

Weizmann Institute of Science

will talk about his personal take on mathematics and machine learning

Date: **16 Feb, 14:00 CET**

Click here for abstract and +info.



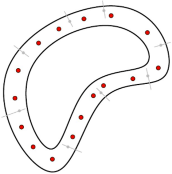
carlos simpson

CNRS

will talk about the interaction between proof assistants and reinforcement learning

Date: **March**

Click here for abstract and +info.



charles fefferman

Princeton University

will talk about 'Personal encounters with machine learning'

Date: **27 Apr, 15:00 CET**

Click here for abstract and +info.

Join us at mat.uab.cat/bM2L

Thank you for your attention!

MINISTERIO DE CIENCIA E INNOVACIÓN

Cofinanciado por la Unión Europea

AGENCIA ESTATAL DE INVESTIGACIÓN

RYC2020-030114-I

PID2022-137667NA-I00

CNS2024-154695

Slides may become available at
mat.uab.cat/~rubio