Deep Learning & Applied AI

Multi-layer perceptron and back-propagation

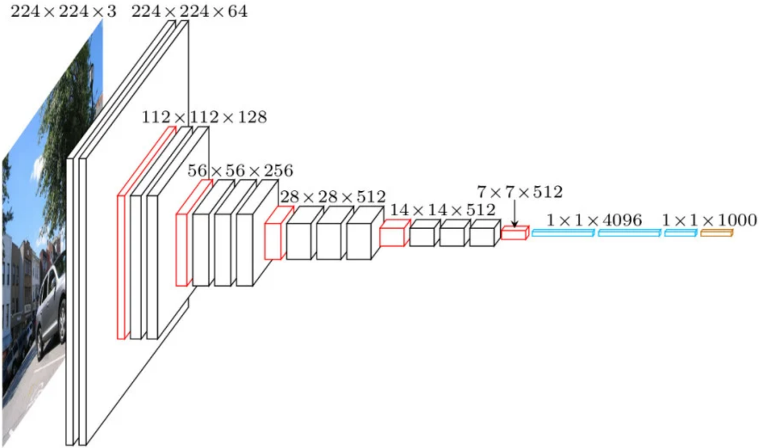
Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2022/2023 · March 29, 2023

A glimpse into neural networks

In deep learning, we deal with highly parametrized models called deep neural networks:



Deep composition

The simplest example of a nonlinear parametric model:

$$f \circ f(\mathbf{x})$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\underbrace{f}_{\text{linear}} \circ \underbrace{f}_{\text{linear}}(\mathbf{x})$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\underbrace{f \circ f}_{\text{linear}}(\mathbf{x})$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is the logistic function, we have the logistic regression model.

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is the logistic function, we have the logistic regression model.

Consider multiple layers of logistic regression models:

$$\underbrace{(\sigma \circ f)}_{\text{layer}} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{\text{layer}}(\mathbf{x})$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is the logistic function, we have the logistic regression model.

Consider multiple layers of logistic regression models:

$$\text{output} \leftarrow \underbrace{(\sigma \circ f)}_{\text{layer } n} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{\text{layer } 1}(\mathbf{x}) \leftarrow \text{input}$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is the logistic function, we have the logistic regression model.

Consider multiple layers of logistic regression models:

$$\text{output} \leftarrow \underbrace{(\sigma \circ f)}_{\text{output layer}} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{\text{input layer}} (\mathbf{x}) \leftarrow \text{input}$$

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is the logistic function, we have the logistic regression model.

Consider multiple layers of logistic regression models:

$$\text{output} \leftarrow \underbrace{(\sigma \circ f)}_{\text{output layer}} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{\text{input layer}} (\mathbf{x}) \leftarrow \text{input}$$

(Function composition is associative, so parentheses are not necessary.)

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

quindi una linear map seguita da una funzione non lineare (**activation function**)

If σ is the logistic function, we have the logistic regression model.

Consider multiple layers of logistic regression models:

$$\text{output} \leftarrow \underbrace{(\sigma \circ f)}_{\text{output layer}} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{\text{input layer}} (\mathbf{x}) \leftarrow \text{input}$$

(Function composition is associative, so parentheses are not necessary.)

More in general, consider other activation functions than logistic:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma(x) = \max\{0, x\}$$

ReLu

continuous

discontinuous gradient

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$(\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

a multi-layer perceptron (MLP) or deep feed-forward neural network.

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$g_{\boldsymbol{\Theta}}(\mathbf{x}) = (\sigma \circ f_{\boldsymbol{\Theta}_n}) \circ (\sigma \circ f_{\boldsymbol{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\boldsymbol{\Theta}_1})(\mathbf{x})$$

a multi-layer perceptron (MLP) or deep feed-forward neural network.

The parameters or weights of the MLP are scattered across the layers.

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$g_{\mathbf{\Theta}}(\mathbf{x}) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x})$$

a multi-layer perceptron (MLP) or deep feed-forward neural network.

The parameters or weights of the MLP are scattered across the layers.

Each layer outputs an intermediate hidden representation:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell)$$

where we encode the weights at layer ℓ in the matrix $\mathbf{W}_\ell$

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$g_{\mathbf{\Theta}}(\mathbf{x}) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x})$$

a multi-layer perceptron (MLP) or deep feed-forward neural network.

The parameters or weights of the MLP are scattered across the layers.

Each layer outputs an intermediate hidden representation:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell + \mathbf{b}_\ell)$$

where we encode the weights at layer ℓ in the matrix $\mathbf{W}_\ell$ and bias $\mathbf{b}_\ell$.

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$g_{\boldsymbol{\Theta}}(\mathbf{x}) = (\sigma \circ f_{\boldsymbol{\Theta}_n}) \circ (\sigma \circ f_{\boldsymbol{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\boldsymbol{\Theta}_1})(\mathbf{x})$$

i numeri indicano
i livelli della rete

a multi-layer perceptron (MLP) or deep feed-forward neural network.

The parameters or weights of the MLP are scattered across the layers.

Each layer outputs an intermediate hidden representation:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell + \mathbf{b}_\ell)$$

where we encode the weights at layer ℓ in the matrix $\mathbf{W}_\ell$ and bias $\mathbf{b}_\ell$.

Remark: The bias can be included in the weight matrix by writing:

$$\mathbf{W} \mapsto \begin{pmatrix} \mathbf{W} & \mathbf{b} \end{pmatrix}, \quad \mathbf{x} \mapsto \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix},$$

viene aggiunta una nuova
dimensione (parametro)

because each f is linear in the parameters just like in linear regression.

Hidden units

At each hidden layer we have:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell)$$

Hidden units

At each hidden layer we have:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell)$$

Each row of the weight matrix is called a neuron or hidden unit:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{— unit —} \\ \vdots \\ \text{— unit —} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

Hidden units

At each hidden layer we have:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell)$$

Each row of the weight matrix is called a neuron or hidden unit:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{— unit —} \\ \vdots \\ \text{— unit —} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

We have two interpretations:

1. Each layer is a vector-to-vector function $\mathbb{R}^p \to \mathbb{R}^q$.

Hidden units

At each hidden layer we have:

$$\mathbf{x}_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell)$$

Each row of the weight matrix is called a neuron or hidden unit:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{— unit —} \\ \vdots \\ \text{— unit —} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

We have two interpretations:

1. Each layer is a vector-to-vector function $\mathbb{R}^p \to \mathbb{R}^q$.
2. Each layer has q units acting in parallel. perché le unità sono indipendenti tra di loro
   Each unit acts as a scalar function $\mathbb{R}^p \to \mathbb{R}$.

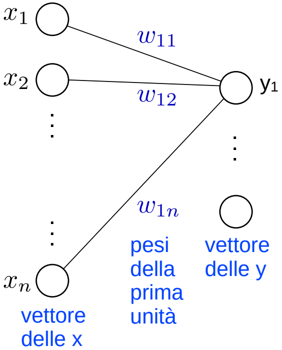
Single layer illustration

$$\sigma(\mathbf{W}\mathbf{x}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \ldots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

Single layer illustration

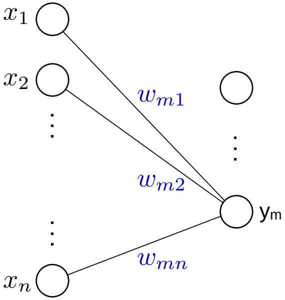
$$\sigma(\mathbf{Wx}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \cdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

questo è un modo di scrivere le moltiplicazioni tra vettori di sopra (spoiler: è la struttura del MLP)



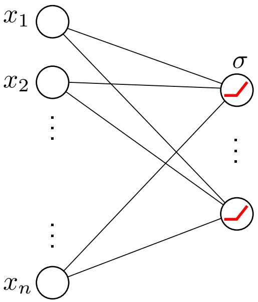
Single layer illustration

$$\sigma(\mathbf{W}\mathbf{x}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \dots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$



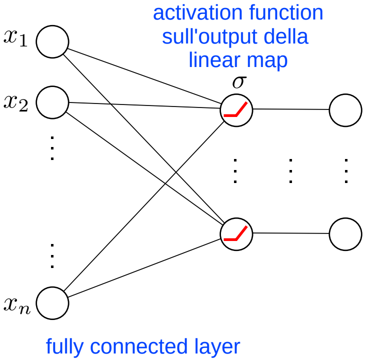
Single layer illustration

$$\sigma(\mathbf{W}\mathbf{x}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \dots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$



Single layer illustration

$$\sigma(\mathbf{W}\mathbf{x}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \cdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$



Output layer

The output layer determines the co-domain of the network:

$$\mathbf{y} = (\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

Output layer

The output layer determines the co-domain of the network:

$$\mathbf{y} = (\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

If σ is the logistic sigmoid, then the entire network will map:

$$\mathbb{R}^p \to (0, 1)^q$$

Output layer

The output layer determines the co-domain of the network:

$$\mathbf{y} = (\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

If σ is the logistic sigmoid, then the entire network will map:

$$\mathbb{R}^p \to (0,1)^q$$

indica un vettore di dimensione q dove ogni dimensione può assumere valore (0, 1) con 0 e 1 esclusi (classification)

For generality, it is common to have a linear layer at the output:

$$\mathbf{y} = f \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

tranne nel caso di classificazione binaria

mapping:

$$\mathbb{R}^p \to \mathbb{R}^q$$

Deep ReLU networks

Adding a linear layer at the output:

$$\mathbf{y} = f \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

Deep ReLU networks

Adding a linear layer at the output:

$$\mathbf{y} = f \circ \sigma(\cdots)(\mathbf{x})$$

expresses $\mathbf{y}$ as a combination of "ridge functions" $\sigma(\cdots)$.

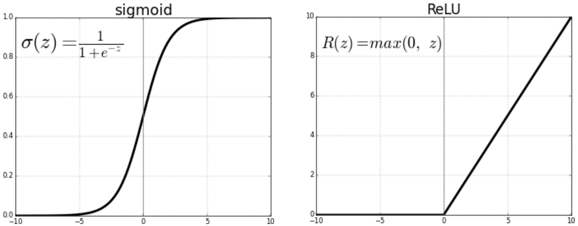
Deep ReLU networks

Adding a linear layer at the output:

$$\mathbf{y} = f \circ \sigma(\cdots)(\mathbf{x})$$

expresses $\mathbf{y}$ as a combination of “ridge functions” $\sigma(\cdots)$.

For a 2-layer network with activation $\sigma(x) = \max\{0, x\}$ (rectifier), we get a piecewise-linear function:



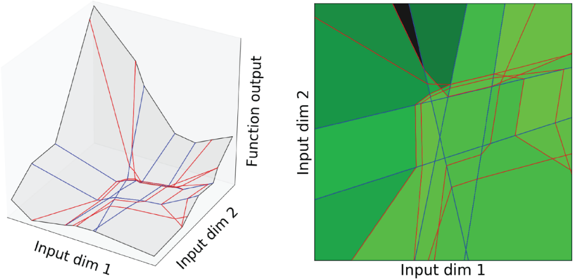
Deep ReLU networks

Adding a linear layer at the output:

$$\mathbf{y} = f \circ \sigma(\cdots)(\mathbf{x})$$

expresses $\mathbf{y}$ as a combination of "ridge functions" $\sigma(\cdots)$.

For a 2-layer network with activation $\sigma(x) = \max\{0, x\}$ (rectifier), we get a piecewise-linear function:



The blue and red edges are produced by the first and second layer.

Universality

What class of functions can we represent with a MLP?

Universality

What class of functions can we represent with a MLP?

If σ is sigmoidal, we have the following:

Universal Approximation Theorem For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}+\mathbf{b})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence.

Cybenko, "Approximations by superpositions of sigmoidal functions", 1989

Universality

What class of functions can we represent with a MLP?

If σ is sigmoidal, we have the following:

Universal Approximation Theorem For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}+\mathbf{b})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence. Thus, for any continuous function f and any $\epsilon > 0$, there exists $q \in \mathbb{N}$ and weights s.t.:

$$|f(\mathbf{x}) - \sum_{k=1}^{q} u_k \phi(\mathbf{x})| \leq \epsilon \qquad \text{for all } \mathbf{x} \in \Omega$$

Cybenko, "Approximations by superpositions of sigmoidal functions", 1989

Universality

What class of functions can we represent with a MLP?

If σ is sigmoidal, we have the following:

Universal Approximation Theorem For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}+\mathbf{b})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence. Thus, for any continuous function f and any $\epsilon > 0$, there exists $q \in \mathbb{N}$ and weights s.t.:

$$|f(\mathbf{x}) - \sum_{k=1}^{q} u_k \phi(\mathbf{x})| \leq \epsilon \qquad \text{for all } \mathbf{x} \in \Omega$$

The network in the theorem has just one hidden layer.

Cybenko, "Approximations by superpositions of sigmoidal functions", 1989

Universality

What class of functions can we represent with a MLP?

If σ is sigmoidal, we have the following:

> **Universal Approximation Theorem** For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x}+\mathbf{b})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence. Thus, for any continuous function f and any $\epsilon > 0$, there exists $q \in \mathbb{N}$ and weights s.t.:
>
> $$|f(\mathbf{x}) - \sum_{k=1}^{q} u_k \phi(\mathbf{x})| \leq \epsilon \text{ for all } \mathbf{x} \in \Omega$$

The network in the theorem has just one hidden layer.

For large enough q, the training error can be made arbitrarily small.

Cybenko, "Approximations by superpositions of sigmoidal functions", 1989

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\mathbf{\Theta}}(\mathbf{x}_i) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\mathbf{\Theta}}(\mathbf{x}_i) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

$$\ell_{\mathbf{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - g_{\mathbf{\Theta}}(\mathbf{x}_i)\|_2^2$$

Solving for the weights $\mathbf{\Theta}$ is referred to as training.

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\mathbf{\Theta}}(\mathbf{x}_i) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

$$\ell_{\mathbf{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - g_{\mathbf{\Theta}}(\mathbf{x}_i)\|_2^2$$

Solving for the weights $\mathbf{\Theta}$ is referred to as training.

In general, the loss is not convex w.r.t. $\mathbf{\Theta}$.

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\mathbf{\Theta}}(\mathbf{x}_i) = (\sigma \circ f_{\mathbf{\Theta}_n}) \circ (\sigma \circ f_{\mathbf{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\mathbf{\Theta}_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

$$\ell_{\mathbf{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - g_{\mathbf{\Theta}}(\mathbf{x}_i)\|_2^2$$

Solving for the weights $\mathbf{\Theta}$ is referred to as training.

In general, the loss is not convex w.r.t. $\mathbf{\Theta}$.

As we have seen, the following special cases are convex:

- One layer, no activation, MSE loss ($\Rightarrow$ linear regression).

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\boldsymbol{\Theta}}(\mathbf{x}_i) = (\sigma \circ f_{\boldsymbol{\Theta}_n}) \circ (\sigma \circ f_{\boldsymbol{\Theta}_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\boldsymbol{\Theta}_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

$$\ell_{\boldsymbol{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - g_{\boldsymbol{\Theta}}(\mathbf{x}_i)\|_2^2$$

Solving for the weights $\boldsymbol{\Theta}$ is referred to as training.

In general, the loss is not convex w.r.t. $\boldsymbol{\Theta}$.

As we have seen, the following special cases are convex:

- One layer, no activation, MSE loss ($\Rightarrow$ linear regression).
- One layer, sigmoid activation, logistic loss ($\Rightarrow$ logistic regression).

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\boldsymbol{\Theta}}$.

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\boldsymbol{\Theta}}$.

For the basic MSE, this means:

$$\nabla \ell_{\boldsymbol{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\Theta}} \|\mathbf{y}_i - g_{\boldsymbol{\Theta}}(\mathbf{x}_i)\|_2^2$$

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\mathbf{\Theta}}$.

For the basic MSE, this means:

$$\nabla \ell_{\mathbf{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{\Theta}} \|(\mathbf{y}_i - (\sigma(f_{\mathbf{\Theta}_n}((\sigma(f_{\mathbf{\Theta}_{n-1}}(\cdots(\sigma(f_{\mathbf{\Theta}_1}(\mathbf{x}_i))\cdots))))\|_2^2$$

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\mathbf{\Theta}}$.

For the basic MSE, this means:

$$\nabla \ell_{\mathbf{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{\Theta}} \|(\mathbf{y}_i - (\sigma(f_{\mathbf{\Theta}_n}((\sigma(f_{\mathbf{\Theta}_{n-1}}(\cdots(\sigma(f_{\mathbf{\Theta}_1}(\mathbf{x}_i))\cdots))))\|_2^2$$

- Computing the gradients by hand is infeasible.

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\boldsymbol{\Theta}}$.

For the basic MSE, this means:

$$\nabla \ell_{\boldsymbol{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\Theta}} \|(\mathbf{y}_i - (\sigma(f_{\boldsymbol{\Theta}_n}((\sigma(f_{\boldsymbol{\Theta}_{n-1}}(\cdots(\sigma(f_{\boldsymbol{\Theta}_1}(\mathbf{x}_i))\cdots))))\|_2^2$$

- Computing the gradients by hand is infeasible.
- Finite differences require $O(\#\text{weights})$ evaluations of $\ell_{\boldsymbol{\Theta}}$.

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla \ell_{\boldsymbol{\Theta}}$.

For the basic MSE, this means:

$$\nabla \ell_{\boldsymbol{\Theta}}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\Theta}} \|(\mathbf{y}_i - (\sigma(f_{\boldsymbol{\Theta}_n}((\sigma(f_{\boldsymbol{\Theta}_{n-1}}(\cdots(\sigma(f_{\boldsymbol{\Theta}_1}(\mathbf{x}_i))\cdots))))\|_2^2$$

- Computing the gradients by hand is infeasible.
- Finite differences require $O(\#\text{weights})$ evaluations of $\ell_{\boldsymbol{\Theta}}$.
- Using the chain rule is sub-optimal.

Training

We train using gradient descent-like algorithms.

Bottleneck: Computation of gradients $\nabla\ell_{\boldsymbol{\Theta}}$.

For the basic MSE, this means:

$$\nabla\ell_{\boldsymbol{\Theta}}(\{\mathbf{x}_i,\mathbf{y}_i\}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\Theta}}\|(\mathbf{y}_i - (\sigma(f_{\boldsymbol{\Theta}_n}((\sigma(f_{\boldsymbol{\Theta}_{n-1}}(\cdots(\sigma(f_{\boldsymbol{\Theta}_1}(\mathbf{x}_i))\cdots))))\|_2^2$$

- Computing the gradients by hand is infeasible.
- Finite differences require $O(\#\text{weights})$ evaluations of $\ell_{\boldsymbol{\Theta}}$.
- Using the chain rule is sub-optimal.

We want to automatize this computational step efficiently.

Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Example:

$$f(x) = \log x + \sqrt{\log x}$$

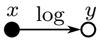
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Example:

$$f(x) = \log x + \sqrt{\log x}$$



Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Example:

$$f(x) = \log x + \sqrt{\log x}$$

$$x \xrightarrow{\log} y \xrightarrow{\sqrt{\ }} z$$

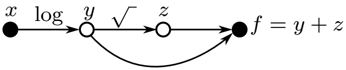
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Example:

$$f(x) = \log x + \sqrt{\log x}$$



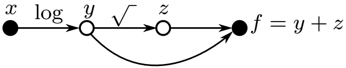
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$

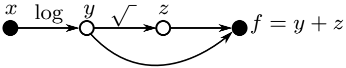
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

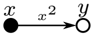
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



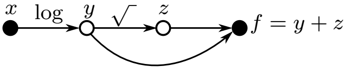
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

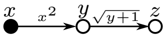
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2 + 1})}{x^2} - \frac{\log^3(x + \sqrt{x^2 + 1})}{\sqrt{x^2 + 1}}$$



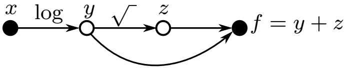
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

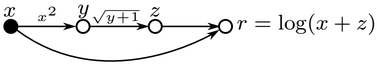
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



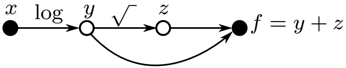
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

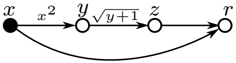
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



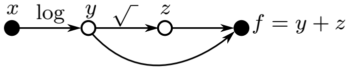
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

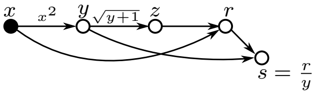
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



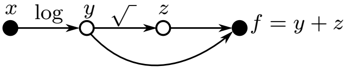
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

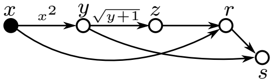
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



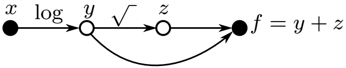
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

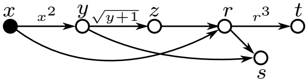
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



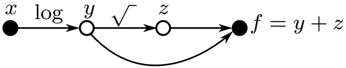
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

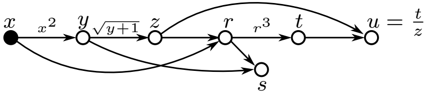
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2 + 1})}{x^2} - \frac{\log^3(x + \sqrt{x^2 + 1})}{\sqrt{x^2 + 1}}$$



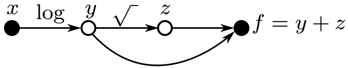
Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

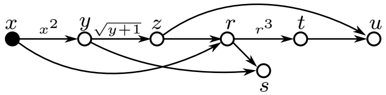
Example:

$$f(x) = \log x + \sqrt{\log x}$$



Example:

$$f(x) = \frac{\log(x + \sqrt{x^2 + 1})}{x^2} - \frac{\log^3(x + \sqrt{x^2 + 1})}{\sqrt{x^2 + 1}}$$



Computational graphs

Consider a generic function $f : \mathbb{R} \to \mathbb{R}$.

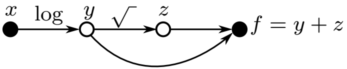
A computational graph is a directed acyclic graph representing the computation of $f(x)$ with intermediate variables.

le variabili intermedie sono i pallini bianchi, quelli neri sono l'input e l'output

Example:

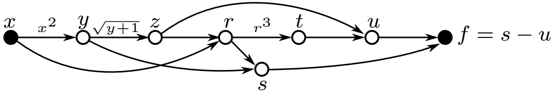
$$f(x) = \log x + \sqrt{\log x}$$

le frecce indicano che l'operazione è salvata nella variabile successiva



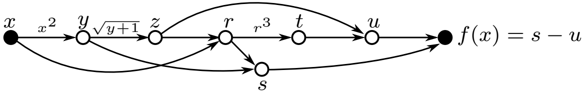
Example:

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}}$$



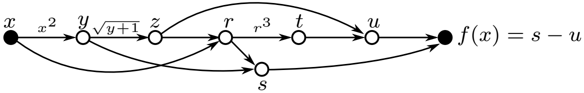
Computational graphs

The evaluation of $f(x)$ corresponds to a forward traversal of the graph:



Computational graphs

The evaluation of $f(x)$ corresponds to a forward traversal of the graph:

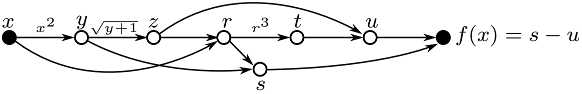


The graph is constructed programmaticaly, for example:

z = sqrt(sum(square(x), 1));

Computational graphs

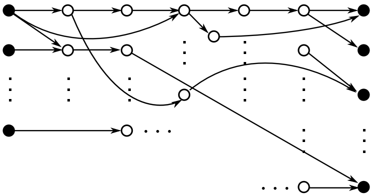
The evaluation of $f(x)$ corresponds to a forward traversal of the graph:



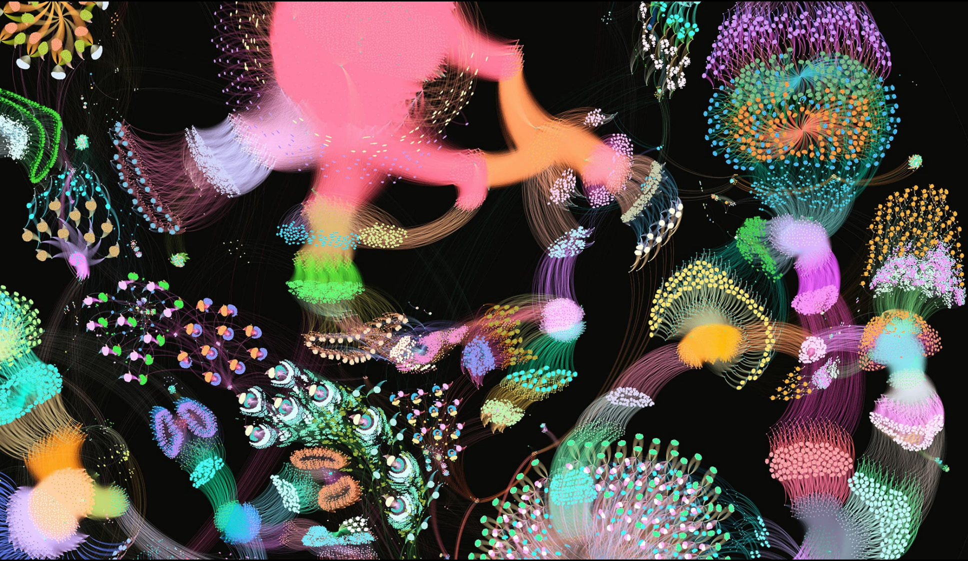
The graph is constructed programmaticaly, for example:

z = sqrt(sum(square(x), 1));

For high-dimensional input/output, the graph may be more complex:



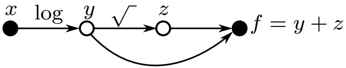
The computational graph gets big quickly.



Poplar visualization, see `https://www.graphcore.ai/products/poplar`

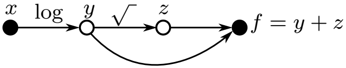
Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$



Automatic differentiation: Forward mode

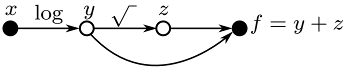
$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial x}{\partial x} = 1$$

Automatic differentiation: Forward mode

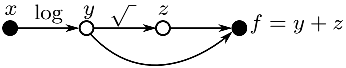
$$f(x) = \log x + \sqrt{\log x}$$



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

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$

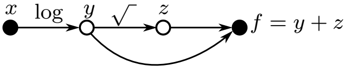


$$\frac{\partial x}{\partial x} = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x}$$

Automatic differentiation: Forward mode

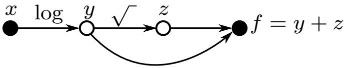
$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial x}{\partial x} = 1$$
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$



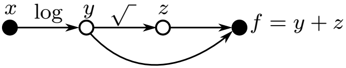
$$\frac{\partial x}{\partial x} = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial x}{\partial x} = 1$$
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x}$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$

$$\begin{tikzcd} x \arrow[r, "\log"] & y \arrow[r, "\sqrt{\ }"] \arrow[rr, bend right] & z \arrow[r] & f = y + z \end{tikzcd}$$

$$\frac{\partial x}{\partial x} = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x}$$

Automatic differentiation: Forward mode

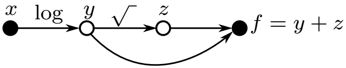
$$f(x) = \log x + \sqrt{\log x}$$

$$\begin{tikzcd} x \arrow[r, "\log"] & y \arrow[r, "\sqrt{\ }"] \arrow[rr, bend right] & z \arrow[r] & f = y + z \end{tikzcd}$$

$$\begin{aligned}
\frac{\partial x}{\partial x} &= 1 \\
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x} \\
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x} \\
\frac{\partial f}{\partial x} &=
\end{aligned}$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial x}{\partial x} = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial x}$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x}$$

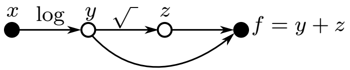
$$\begin{tikzcd} x \arrow[r, "\log"] & y \arrow[r, "\sqrt{}"] \arrow[rr, bend right] & z \arrow[r] & f = y + z \end{tikzcd}$$

$$\begin{aligned}
\frac{\partial x}{\partial x} &= 1 \\
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x} \\
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x} \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial (y+z)}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial (y+z)}{\partial z}\frac{\partial z}{\partial x}
\end{aligned}$$

Automatic differentiation: Forward mode

PRIMO METODO PER FARE
AUTOMATIC DIFFERENTATION

$$f(x) = \log x + \sqrt{\log x}$$



in pratica, calcola la derivata parziale
per ogni nodo (variabile intermedia e output)
SULLA BASE DI X (l'input) applicando la
chain rule e poi somma tutto. Questo è il
significato di "forward mode"

$$\frac{\partial x}{\partial x} = 1$$

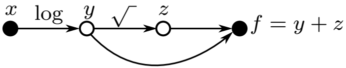
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x}\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \sqrt{y}}{\partial y}\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial (y+z)}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial (y+z)}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} + \frac{\partial z}{\partial x}$$

Automatic differentiation: Forward mode

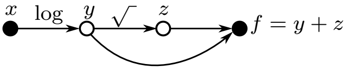
$$f(x) = \log x + \sqrt{\log x} \qquad \frac{\partial f}{\partial x} = \frac{1}{2\sqrt{y}}\frac{1}{x} + \frac{1}{x}$$



Assumption: Each partial derivative is a "primitive" accessible in closed form and can be computed on the fly.

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x} \qquad \frac{\partial f}{\partial x} = \frac{1}{2\sqrt{y}}\frac{1}{x} + \frac{1}{x}$$



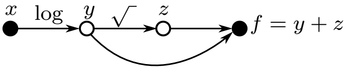
Assumption: Each partial derivative is a “primitive” accessible in closed form and can be computed on the fly.

$$\text{cost of computing } \tfrac{\partial f}{\partial x}(x) \;\; = \;\; \text{cost of computing } f(x)$$

Automatic differentiation: Forward mode

$$f(x) = \log x + \sqrt{\log x} \qquad \frac{\partial f}{\partial x} = \frac{1}{2\sqrt{y}}\frac{1}{x} + \frac{1}{x}$$

era il risultato
della slide precedente



Assumption: Each partial derivative is a "primitive" accessible in closed form and can be computed on the fly.

$$\text{cost of computing } \tfrac{\partial f}{\partial x}(x) \;=\; \text{cost of computing } f(x)$$

However, if the input is high-dimensional, i.e. $f : \mathbb{R}^p \to \mathbb{R}$:

$$\text{cost of computing } \nabla f(\mathbf{x}) \;=\; p \times \text{cost of computing } f(\mathbf{x})$$

since partial derivatives must be computed w.r.t. each input dimension.

In pratica, è efficiente per funzioni con un piccolo numero di input

Automatic differentiation: Forward mode

The forward mode computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \ldots$ with respect to the input x.

Straightforward application of the chain rule.

Automatic differentiation: Forward mode

The forward mode computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \ldots$ with respect to the input x.

Straightforward application of the chain rule.

$$\begin{array}{ccc} \text{Automatic differentiation} & \neq & \text{Symbolic differentiation} \\ \text{(e.g. autograd)} & & \text{(e.g. Mathematica)} \end{array}$$

We accumulate values during code execution to generate numerical derivative evaluations rather than derivative expressions.

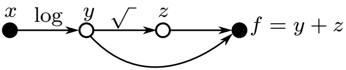
Automatic differentiation: Forward mode

The forward mode computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \ldots$ with respect to the input x.

Straightforward application of the chain rule.

Automatic differentiation	$\neq$	Symbolic differentiation
(e.g. autograd)		(e.g. Mathematica)

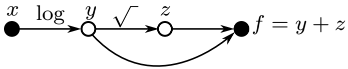
We accumulate values during code execution to generate numerical derivative evaluations rather than derivative expressions.



Reverse mode: compute all the partial derivatives $\frac{\partial f}{\partial z}, \ldots, \frac{\partial f}{\partial x}$ with respect to the inner nodes.

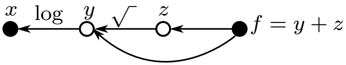
Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



Automatic differentiation: Reverse mode

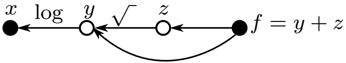
$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

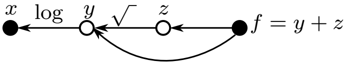
$$\frac{\partial f}{\partial z} =$$

$$\frac{\partial f}{\partial y} =$$

$$\frac{\partial f}{\partial x} =$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

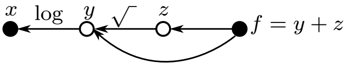
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z}$$

$$\frac{\partial f}{\partial y} =$$

$$\frac{\partial f}{\partial x} =$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

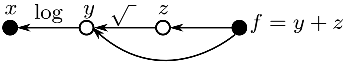
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z}$$

$$\frac{\partial f}{\partial y} =$$

$$\frac{\partial f}{\partial x} =$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

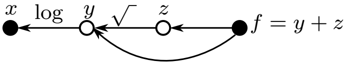
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial y} =$$

$$\frac{\partial f}{\partial x} =$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$
\begin{aligned}
\frac{\partial f}{\partial f} &= 1 \\
\frac{\partial f}{\partial z} &= \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f} \\
\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} \\
\frac{\partial f}{\partial x} &=
\end{aligned}
$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$

$$\begin{tikzcd} x & y \arrow[l, "\log"] & z \arrow[l, "\sqrt{\ }"] & f = y + z \arrow[l] \arrow[ll, bend left] \end{tikzcd}$$

$$\frac{\partial f}{\partial f} = 1$$

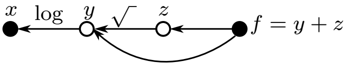
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial y}$$

$$\frac{\partial f}{\partial x} =$$

Automatic differentiation: Reverse mode

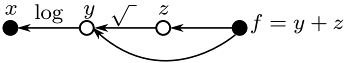
$$f(x) = \log x + \sqrt{\log x}$$



$$
\begin{aligned}
\frac{\partial f}{\partial f} &= 1 \\
\frac{\partial f}{\partial z} &= \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f} \\
\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial y} = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y}} + \frac{\partial f}{\partial f} \\
\frac{\partial f}{\partial x} &=
\end{aligned}
$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$



$$\frac{\partial f}{\partial f} = 1$$

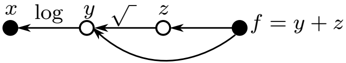
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial y} = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y}} + \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}$$

Automatic differentiation: Reverse mode

$$f(x) = \log x + \sqrt{\log x}$$

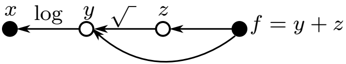


$$
\begin{aligned}
\frac{\partial f}{\partial f} &= 1 \\
\frac{\partial f}{\partial z} &= \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f} \\
\frac{\partial f}{\partial y} &= \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial y} = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y}} + \frac{\partial f}{\partial f} \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial \log x}{\partial x}
\end{aligned}
$$

Automatic differentiation: Reverse mode

SECONDO METODO PER FARE
AUTOMATIC DIFFERENTATION

$$f(x) = \log x + \sqrt{\log x}$$



viene ripetuto lo stesso procedimento di prima, ma inversamente (sulla base dei nodi intermedi partendo da f)

$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial z}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial y} = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y}} + \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial \log x}{\partial x} = \frac{\partial f}{\partial y}\frac{1}{x}$$

Automatic differentiation: Reverse mode

Reverse mode requires computing the values of the internal nodes first:

$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\,\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\,\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\,\frac{\partial (y+z)}{\partial f} = \frac{\partial f}{\partial z}\,\frac{1}{2\sqrt{\mathbf{y}}} + \frac{\partial f}{\partial f}$$

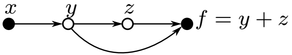
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\,\frac{\partial \log x}{\partial x} = \frac{\partial f}{\partial y}\,\frac{1}{\mathbf{x}}$$

Automatic differentiation: Reverse mode

Reverse mode requires computing the values of the internal nodes first:

$$
\begin{aligned}
&\frac{\partial f}{\partial f} = 1 \\
&\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f} \\
&\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\frac{\partial (y+z)}{\partial f} = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{\mathbf{y}}} + \frac{\partial f}{\partial f} \\
&\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial \log x}{\partial x} = \frac{\partial f}{\partial y}\frac{1}{\mathbf{x}}
\end{aligned}
$$

1. Forward pass to evaluate all the interior nodes $y, z, \ldots$.



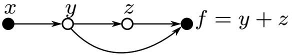
Remark: This is not forward-mode autodiff, since we are only computing function values.

Automatic differentiation: Reverse mode

Reverse mode requires computing the values of the internal nodes first:

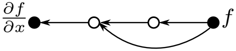
$$
\begin{aligned}
&\frac{\partial f}{\partial f} = 1 \\
&\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\,\frac{\partial (y+z)}{\partial z} = \frac{\partial f}{\partial f} \\
&\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\,\frac{\partial \sqrt{y}}{\partial y} + \frac{\partial f}{\partial f}\,\frac{\partial (y+z)}{\partial f} = \frac{\partial f}{\partial z}\,\frac{1}{2\sqrt{\mathbf{y}}} + \frac{\partial f}{\partial f} \\
&\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\,\frac{\partial \log x}{\partial x} = \frac{\partial f}{\partial y}\,\frac{1}{\mathbf{x}}
\end{aligned}
$$

1. Forward pass to evaluate all the interior nodes $y, z, \ldots$.



Remark: This is not forward-mode autodiff, since we are only computing function values.

2. Backward pass to compute the derivatives.



La reverse mode è più efficiente per funzioni con un grande numero di parametri (come nelle reti neurali)

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

il codominio di una loss function
sarà sempre unidimensionale

where $p \gg 1$ is the number of weights.

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

matrici di gradienti

Instead of simple derivatives we must compute gradients and Jacobians.

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

$$\ell = \epsilon(\sigma \circ f \circ \sigma \circ f \circ \cdots \circ f)$$

ϵ computes the actual scalar error for the loss.

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

$$\ell = \epsilon(f_{t-1} \circ f_{t-2} \circ \cdots \circ f_2 \circ f_1)$$

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

$$\ell = \epsilon(f_{t-1} \circ f_{t-2} \circ \cdots \circ f_2 \circ f_1)$$

Denote by $\mathbf{J}_k$ the Jacobian at layer k.

- Forward-mode autodiff:

$$\nabla \ell = \mathbf{J}_{t-1}(\mathbf{J}_{t-2}(\cdots(\mathbf{J}_3(\mathbf{J}_2\mathbf{J}_1))))$$

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

$$\ell = \epsilon(f_{t-1} \circ f_{t-2} \circ \cdots \circ f_2 \circ f_1)$$

Denote by $\mathbf{J}_k$ the Jacobian at layer k.

- Forward-mode autodiff:

$$\nabla \ell = \mathbf{J}_{t-1}(\mathbf{J}_{t-2}(\cdots(\mathbf{J}_3(\mathbf{J}_2\mathbf{J}_1))))$$

- Reverse-mode autodiff:

$$\nabla \ell = ((((\mathbf{J}_{t-1}\mathbf{J}_{t-2})\mathbf{J}_{t-3})\cdots)\mathbf{J}_2)\mathbf{J}_1$$

Back-propagation

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^p \to \mathbb{R}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

$$\ell = \epsilon(f_{t-1} \circ f_{t-2} \circ \cdots \circ f_2 \circ f_1)$$

Denote by $\mathbf{J}_k$ the Jacobian at layer k.

- Forward-mode autodiff:

$$\nabla \ell = \mathbf{J}_{t-1}(\mathbf{J}_{t-2}(\cdots(\mathbf{J}_3(\mathbf{J}_2\mathbf{J}_1)))) \qquad \#\text{ ops: } \; p\sum_{k=2}^{t-1} d_k d_{k+1}$$

- Reverse-mode autodiff:

$$\nabla \ell = ((((\mathbf{J}_{t-1}\mathbf{J}_{t-2})\mathbf{J}_{t-3})\cdots)\mathbf{J}_2)\mathbf{J}_1$$

Back-propagation ridenominazione della reverse mode

When training neural nets, we compute the gradient of a loss

$$\ell : \mathbb{R}^{p} \to \mathbb{R}^{1}$$

where $p \gg 1$ is the number of weights.

Instead of simple derivatives we must compute gradients and Jacobians.

le derivate allo step k

$$\ell = \epsilon(f_{t-1} \circ f_{t-2} \circ \cdots \circ f_2 \circ f_1)$$

Denote by $\mathbf{J}_k$ the Jacobian at layer k.

- Forward-mode autodiff:

$$\nabla \ell = \mathbf{J}_{t-1}(\mathbf{J}_{t-2}(\cdots(\mathbf{J}_3(\mathbf{J}_2\mathbf{J}_1)))) \qquad \#\text{ ops: } \; p\sum_{k=2}^{t-1} d_k d_{k+1}$$

- Reverse-mode autodiff:

$$\nabla \ell = ((((\mathbf{J}_{t-1}\mathbf{J}_{t-2})\mathbf{J}_{t-3})\cdots)\mathbf{J}_2)\mathbf{J}_1 \qquad \#\text{ ops: } \; 1\sum_{k=1}^{t-2} d_k d_{k+1}$$

le parentesi indicano che
il risultato di quella chain rule
è salvato in qualche modo

Back-propagation

We call back-propagation the reverse mode automatic differentiation applied to deep neural networks.

Evaluating $\nabla \ell$ with backprop is as fast as evaluating ℓ.

Back-propagation

We call back-propagation the reverse mode automatic differentiation applied to deep neural networks.

Evaluating $\nabla \ell$ with backprop is as fast as evaluating ℓ.

Back-propagation is not just the chain rule.

Back-propagation

We call back-propagation the reverse mode automatic differentiation applied to deep neural networks.

Evaluating $\nabla\ell$ with backprop is as fast as evaluating ℓ.

Back-propagation is not just the chain rule.

In fact, not even the costly forward mode is just the chain rule. There are intermediate variables. Backprop is a computational technique.

Back-propagation

We call back-propagation the reverse mode automatic differentiation applied to deep neural networks.

Evaluating $\nabla\ell$ with backprop is as fast as evaluating ℓ.

Back-propagation is not just the chain rule.

In fact, not even the costly forward mode is just the chain rule. There are intermediate variables. Backprop is a computational technique.

Backprop through computational graph of the loss

$$\approx$$

Backprop "through the network"

Suggested reading

Nice, accessible survey on automatic differentiation:
https://arxiv.org/pdf/1502.05767