

Deep Learning & Applied AI

Multi-layer perceptron and
back-propagation

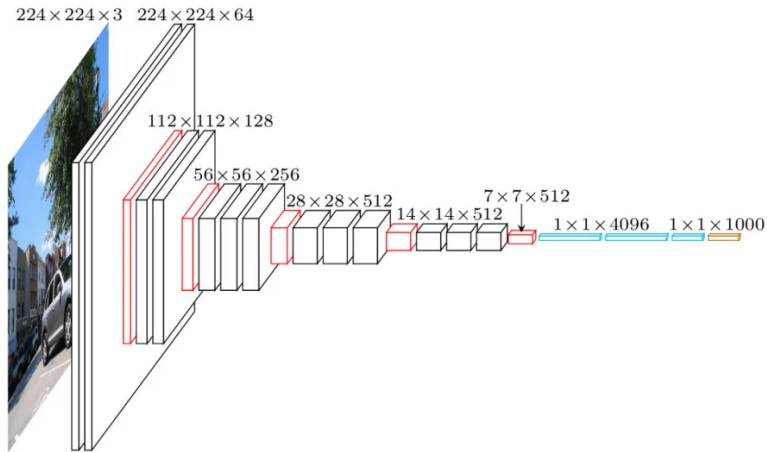
Emanuele Rodolà
rodola@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

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 **logistic regression**.

Deep composition

The simplest example of a nonlinear parametric model:

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

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

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 **logistic regression**.

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 **logistic regression**.

Consider multiple **layers** of logistic regression models:

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

Deep composition

The simplest example of a nonlinear parametric model:

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

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

Consider multiple **layers** of logistic regression models:

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

More in general, consider other **activation functions**:

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

continuous

$$\sigma(x) = \max\{0, x\}$$

discontinuous
gradient

Multi-layer perceptron

We call the composition:

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

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

Multi-layer perceptron

We call the composition:

$$g_{\Theta}(\mathbf{x}) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x})$$

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

The weights of the MLP are scattered across the layers.

Multi-layer perceptron

We call the composition:

$$g_{\Theta}(\mathbf{x}) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x})$$

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

The 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 in matrix \mathbf{W}_{ℓ} and bias \mathbf{b}_{ℓ} .

Multi-layer perceptron

We call the composition:

$$g_{\Theta}(\mathbf{x}) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x})$$

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

The **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 in matrix \mathbf{W}_{ℓ} and bias \mathbf{b}_{ℓ} .

The hidden representations are sometimes called the **activations** at layer $\ell + 1$.

Hidden units

At each hidden layer we have:

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

Hidden units

At each **hidden layer** we have:

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

Each row is called a **hidden unit** or **neuron**:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{--- neuron ---} \\ \vdots \\ \text{--- neuron ---} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

Hidden units

At each **hidden layer** we have:

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

Each row is called a **hidden unit** or **neuron**:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{--- neuron ---} \\ \vdots \\ \text{--- neuron ---} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

We have two interpretations:

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

Hidden units

At each **hidden layer** we have:

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

Each row is called a **hidden unit** or **neuron**:

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} \text{--- neuron ---} \\ \vdots \\ \text{--- neuron ---} \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \end{pmatrix}$$

We have two interpretations:

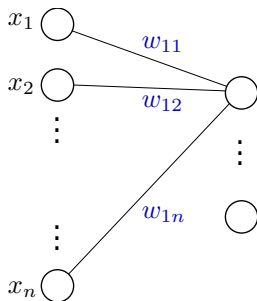
- ① Each layer is a vector-to-vector function $\mathbb{R}^p \rightarrow \mathbb{R}^q$.
- ② Each layer has q neurons acting **in parallel**.
Each neuron acts as a scalar function $\mathbb{R}^p \rightarrow \mathbb{R}$.

Single layer illustration

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

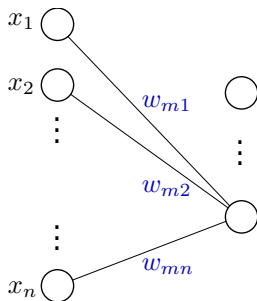
Single layer illustration

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



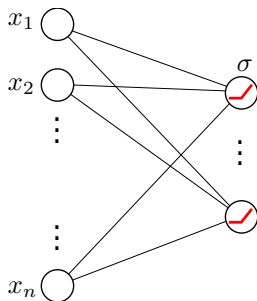
Single layer illustration

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



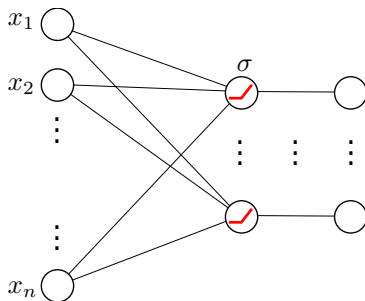
Single layer illustration

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



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, the entire network will map:

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

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, the entire network will map:

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

It is common to have a **linear** layer at the output:

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

which maps:

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

Deep ReLU networks

Let's add a linear layer at the output:

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

Deep ReLU networks

Let's add a linear layer at the output:

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

Then, \mathbf{y} is a combination of "ridge functions" $\sigma(\cdots)$.

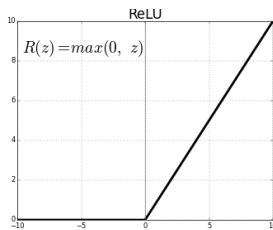
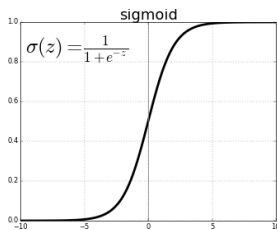
Deep ReLU networks

Let's add a linear layer at the output:

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

Then, \mathbf{y} is 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:



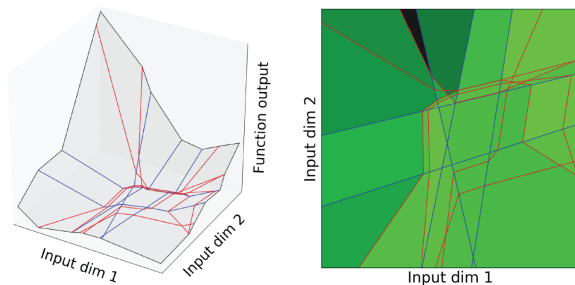
Deep ReLU networks

Let's add a linear layer at the output:

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

Then, \mathbf{y} is 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.

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.:

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

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.:

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

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.:

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

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 $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_{\Theta}(\mathbf{x}_i) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Training

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

$$g_{\Theta}(\mathbf{x}_i) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

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

Solving for the weights Θ is referred to as **training**.

Training

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

$$g_{\Theta}(\mathbf{x}_i) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

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

Solving for the weights Θ is referred to as **training**.

The loss is **not convex** w.r.t. Θ

Training

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

$$g_{\Theta}(\mathbf{x}_i) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \dots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

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

Solving for the weights Θ is referred to as **training**.

The loss is **not convex** w.r.t. Θ

Only some **special cases** are convex:

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

Training

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

$$g_{\Theta}(\mathbf{x}_i) = (\sigma \circ f_{\Theta_n}) \circ (\sigma \circ f_{\Theta_{n-1}}) \circ \cdots \circ (\sigma \circ f_{\Theta_1})(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

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

Solving for the weights Θ is referred to as **training**.

The loss is **not convex** w.r.t. Θ

Only some **special cases** are convex:

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

Training

We train using gradient descent-like algorithms.

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

Training

We train using gradient descent-like algorithms.

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

For the basic MSE, this means:

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

Training

We train using gradient descent-like algorithms.

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

For the basic MSE, this means:

$$\nabla \ell_{\Theta}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\Theta} \|(\mathbf{y}_i - (\sigma(f_{\Theta_n}((\sigma(f_{\Theta_{n-1}}(\cdots(\sigma(f_{\Theta_1}(\mathbf{x}_i))\cdots))))))\|_2^2$$

Training

We train using gradient descent-like algorithms.

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

For the basic MSE, this means:

$$\nabla \ell_{\Theta}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\Theta} \|(\mathbf{y}_i - (\sigma(f_{\Theta_n}((\sigma(f_{\Theta_{n-1}}(\cdots(\sigma(f_{\Theta_1}(\mathbf{x}_i))\cdots))))))\|_2^2$$

- Computing the gradients **by hand** is infeasible.

Training

We train using gradient descent-like algorithms.

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

For the basic MSE, this means:

$$\nabla \ell_{\Theta}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\Theta} \|(\mathbf{y}_i - (\sigma(f_{\Theta_n}((\sigma(f_{\Theta_{n-1}}(\cdots(\sigma(f_{\Theta_1}(\mathbf{x}_i))\cdots))))))\|_2^2$$

- Computing the gradients **by hand** is infeasible.
- **Finite differences** need $O(\text{\#weights})$ evaluations of ℓ_{Θ} .

Training

We train using gradient descent-like algorithms.

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

For the basic MSE, this means:

$$\nabla \ell_{\Theta}(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\Theta} \|(\mathbf{y}_i - (\sigma(f_{\Theta_n}((\sigma(f_{\Theta_{n-1}}(\cdots(\sigma(f_{\Theta_1}(\mathbf{x}_i))\cdots))))))\|_2^2$$

- Computing the gradients **by hand** is infeasible.
- **Finite differences** need $O(\#\text{weights})$ evaluations of ℓ_{Θ} .

We want to automatize this **computational step** efficiently.

Computational graphs

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

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

Computational graphs

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

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
●

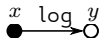
Computational graphs

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

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

Example:

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



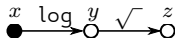
Computational graphs

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

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

Example:

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



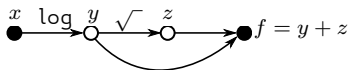
Computational graphs

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

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

Example:

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



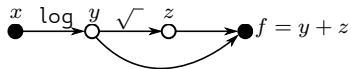
Computational graphs

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

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:

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

x
●

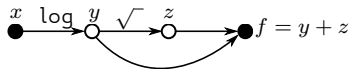
Computational graphs

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

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:

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



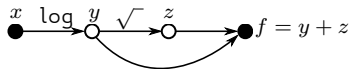
Computational graphs

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

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

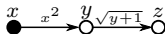
Example:

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



Example:

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



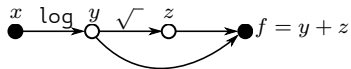
Computational graphs

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

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

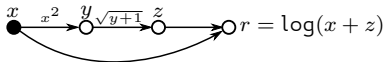
Example:

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



Example:

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



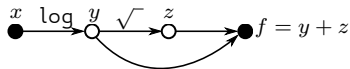
Computational graphs

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

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

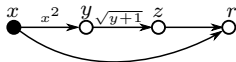
Example:

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



Example:

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



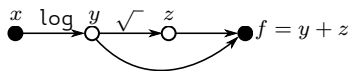
Computational graphs

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

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

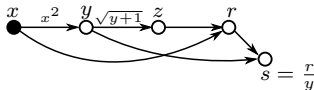
Example:

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



Example:

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



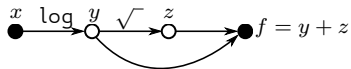
Computational graphs

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

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

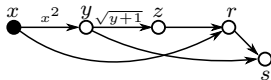
Example:

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



Example:

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



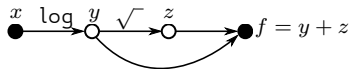
Computational graphs

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

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

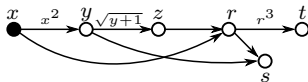
Example:

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



Example:

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



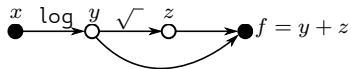
Computational graphs

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

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

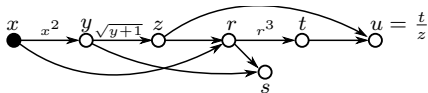
Example:

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



Example:

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



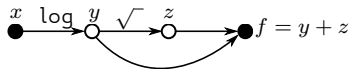
Computational graphs

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

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

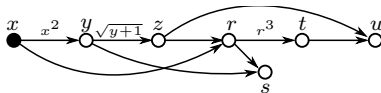
Example:

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



Example:

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



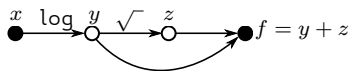
Computational graphs

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

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

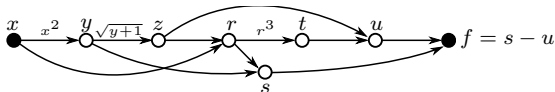
Example:

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



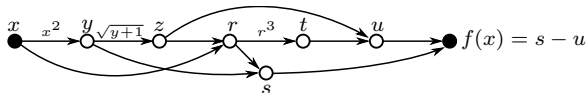
Example:

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



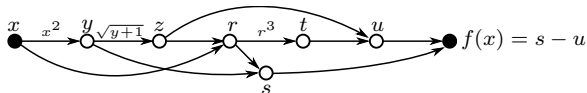
Computational graphs

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:

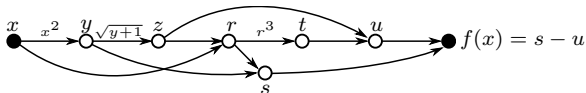


The graph is constructed programmatically, for example:

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


Computational graphs

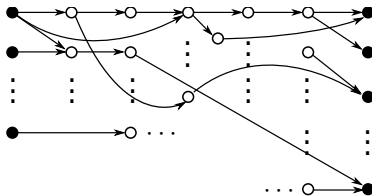
The evaluation of $f(x)$ corresponds to a **forward traversal** of the graph:



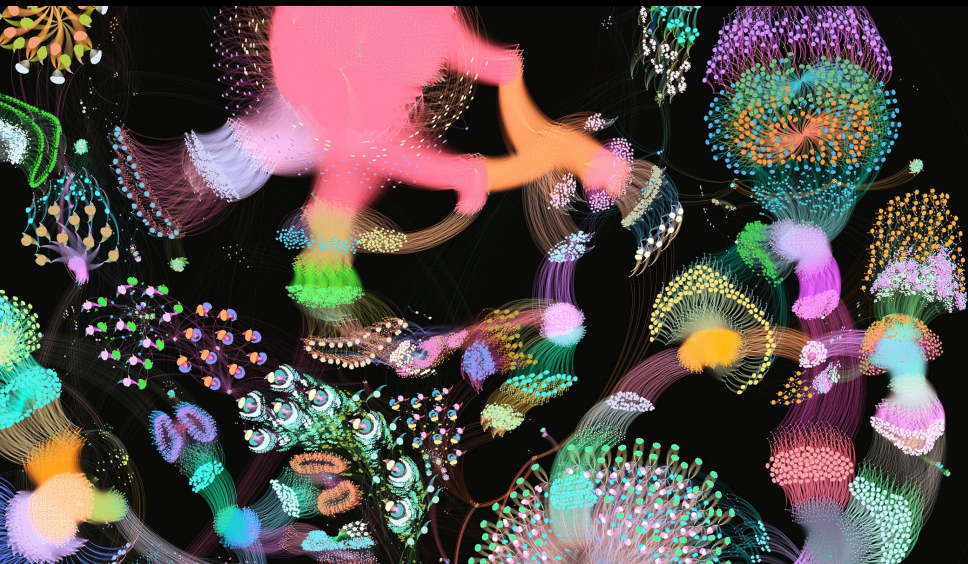
The graph is constructed programmatically, for example:

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

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



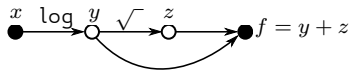
The computational graph gets big quickly.



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

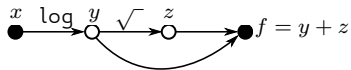
Automatic differentiation: Forward mode

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



Automatic differentiation: Forward mode

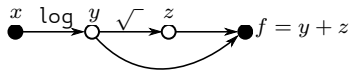
$$f(x) = \log x + \sqrt{\log x}$$



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

Automatic differentiation: Forward mode

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

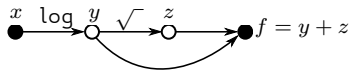


$$\frac{\partial x}{\partial x} = 1$$

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

Automatic differentiation: Forward mode

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

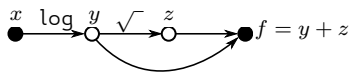


$$\frac{\partial x}{\partial x} = 1$$

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

Automatic differentiation: Forward mode

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

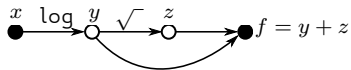


$$\frac{\partial x}{\partial x} = 1$$

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

Automatic differentiation: Forward mode

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



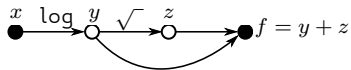
$$\frac{\partial x}{\partial x} = 1$$

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

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

Automatic differentiation: Forward mode

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



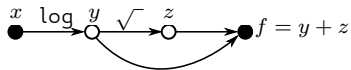
$$\frac{\partial x}{\partial x} = 1$$

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

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

Automatic differentiation: Forward mode

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



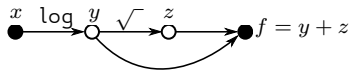
$$\frac{\partial x}{\partial x} = 1$$

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

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

Automatic differentiation: Forward mode

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



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

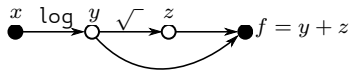
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x} \frac{\partial x}{\partial x} = \frac{1}{x} \frac{\partial x}{\partial x}$$

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

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

Automatic differentiation: Forward mode

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



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

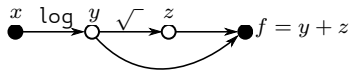
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x} \frac{\partial x}{\partial x} = \frac{1}{x} \frac{\partial x}{\partial x}$$

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

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

Automatic differentiation: Forward mode

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



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

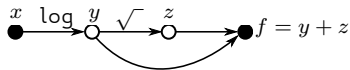
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x} \frac{\partial x}{\partial x} = \frac{1}{x} \frac{\partial x}{\partial x}$$

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

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

Automatic differentiation: Forward mode

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



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

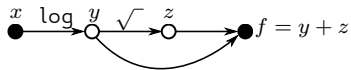
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \frac{\partial x}{\partial x} = \frac{\partial \log x}{\partial x} \frac{\partial x}{\partial x} = \frac{1}{x} \frac{\partial x}{\partial x}$$

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

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

Automatic differentiation: Forward mode

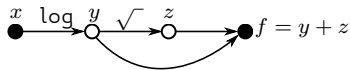
$$f(x) = \log x + \sqrt{\log x} \qquad \frac{\partial f}{\partial x} = \frac{1}{2\sqrt{y}} \frac{1}{x} + \frac{1}{x}$$



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

Automatic differentiation: Forward mode

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

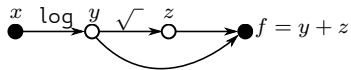


Assumption: Each partial derivative is a "primitive" accessible in **closed form** and can be computed on the fly.

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

Automatic differentiation: Forward mode

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



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

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

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

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

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

Automatic differentiation: Forward mode

Computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \dots$ with respect to the **input** x .

Automatic differentiation: Forward mode

Computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \dots$ with respect to the **input** x .

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

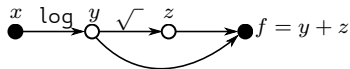
We accumulate values during code execution, to get numerical **evaluations** rather than **expressions** for the derivative.

Automatic differentiation: Forward mode

Computes all the partial derivatives $\frac{\partial y}{\partial x}, \frac{\partial z}{\partial x}, \dots$ with respect to the **input** x .

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

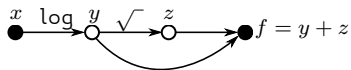
We accumulate values during code execution, to get numerical **evaluations** rather than **expressions** for the derivative.



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

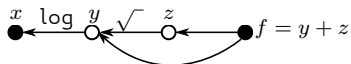
Automatic differentiation: Reverse mode

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



Automatic differentiation: Reverse mode

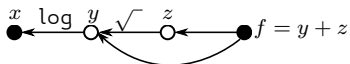
$$f(x) = \log x + \sqrt{\log x}$$



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

Automatic differentiation: Reverse mode

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



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

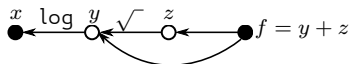
$$\frac{\partial f}{\partial z} =$$

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

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

Automatic differentiation: Reverse mode

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



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

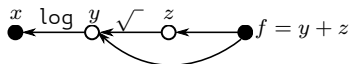
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z}$$

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

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

Automatic differentiation: Reverse mode

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



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

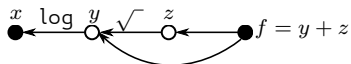
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z}$$

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

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

Automatic differentiation: Reverse mode

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



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

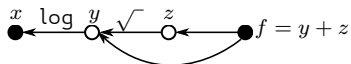
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

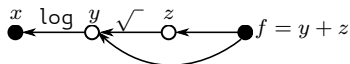
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

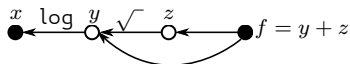
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

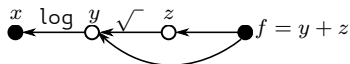
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

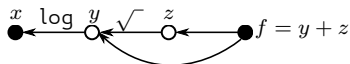
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

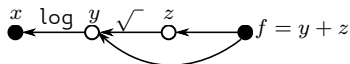
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \frac{\partial (y + z)}{\partial z} = \frac{\partial f}{\partial f}$$

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

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

Automatic differentiation: Reverse mode

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



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

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

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

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

Automatic differentiation: Reverse mode

Before computing the derivatives, we must compute the values of the **internal nodes** first:

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

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

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

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

Automatic differentiation: Reverse mode

Before computing the derivatives, we must compute the values of the **internal nodes** first:

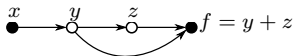
$$\frac{\partial f}{\partial f} = 1$$

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

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

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

① **Forward pass** to evaluate all the interior nodes y, z, \dots



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

Automatic differentiation: Reverse mode

Before computing the derivatives, we must compute the values of the **internal nodes** first:

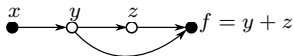
$$\frac{\partial f}{\partial f} = 1$$

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

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

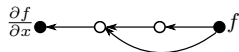
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \frac{\partial \log x}{\partial x} = \frac{\partial f}{\partial y} \frac{1}{x}$$

- ① **Forward pass** to evaluate all the interior nodes y, z, \dots



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

- ② **Backward pass** to compute the **derivatives**.



Back-propagation

When training NNs, we compute the gradient of a loss

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

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

Back-propagation

When training NNs, we compute the gradient of a loss

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

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

Instead of simple derivatives we must compute **gradients** and **Jacobians**.

Back-propagation

When training NNs, we compute the gradient of a loss

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

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

Instead of simple derivatives we must compute **gradients** and **Jacobians**.

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

ϵ computes the actual scalar error for the loss.

Back-propagation

When training NNs, we compute the gradient of a loss

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

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 NNs, we compute the gradient of a loss

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

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 NNs, we compute the gradient of a loss

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

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 NNs, we compute the gradient of a loss

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

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:

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

- **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

When training NNs, we compute the gradient of a loss

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

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:

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

- **Reverse**-mode autodiff:

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

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/abs/1502.05767>