# lecture 6: differentiation

## deep learning for vision

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, Nov. 2018 – Jan. 2019

# logistics

- audio of lectures available: link shared via piazza
- related courses including coding assignments linked on course website
- oral presentations: selection of papers by Sunday Dec 16; graduate student to attend presentations on Monday Jan 21
- material marked as XXXX*: some material like examples or details skipped during the lectures is to be studied at home; other material citing recent methods is really optional

# logistics

- audio of lectures available: link shared via piazza
- related courses including coding assignments linked on course website
- oral presentations: selection of papers by Sunday Dec 16; graduate student to attend presentations on Monday Jan 21
- material marked as XXXX*: some material like examples or details skipped during the lectures is to be studied at home; other material citing recent methods is really optional

# logistics

- audio of lectures available: link shared via piazza
- related courses including coding assignments linked on course website
- oral presentations: selection of papers by Sunday Dec 16; graduate student to attend presentations on Monday Jan 21
- material marked as XXXX*: some material like examples or details skipped during the lectures is to be studied at home; other material citing recent methods is really optional

# outline

gradient descent
gradient computation
automatic differentiation: units
automatic differentiation: functions

# gradient descent

# gradient descent

- a first-order Taylor approximation of $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x}_0$ is

$$f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) := f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^\top \nabla f(\mathbf{x}_0)$$

*i.e.* , the gradient points in the direction of the greatest increase rate

- a second-order approximation needs the Hessian matrix $Hf$

$$f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) := f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top (Hf(\mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0)$$

- assuming $f$ is locally convex with isotropic $Hf(\mathbf{x}_0) = \frac{1}{\epsilon} I$, the gradient of $f^{(2)}_{\mathbf{x}_0}$ is

$$\nabla f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \nabla f(\mathbf{x}_0) + \frac{1}{\epsilon}(\mathbf{x} - \mathbf{x}_0)$$

- so if we were to minimize this approximation instead of $f$, we would let this gradient vanish and solve for $\mathbf{x}$

$$\arg\min_{\mathbf{x}} f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \mathbf{x}_0 - \epsilon \nabla f(\mathbf{x}_0)$$

# gradient descent

- a first-order Taylor approximation of $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x}_0$ is

$$f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) := f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^\top \nabla f(\mathbf{x}_0)$$

  *i.e.* , the gradient points in the direction of the greatest increase rate
- a second-order approximation needs the Hessian matrix $Hf$

$$f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) := f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top (Hf(\mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0)$$

- assuming $f$ is locally convex with isotropic $Hf(\mathbf{x}_0) = \frac{1}{\epsilon} I$, the gradient of $f^{(2)}_{\mathbf{x}_0}$ is

$$\nabla f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \nabla f(\mathbf{x}_0) + \frac{1}{\epsilon}(\mathbf{x} - \mathbf{x}_0)$$

- so if we were to minimize this approximation instead of $f$, we would let this gradient vanish and solve for $\mathbf{x}$

$$\arg\min_{\mathbf{x}} f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \mathbf{x}_0 - \epsilon \nabla f(\mathbf{x}_0)$$

# gradient descent

- a first-order Taylor approximation of $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x}_0$ is

$$f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) := f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^\top \nabla f(\mathbf{x}_0)$$

  *i.e.* , the gradient points in the direction of the greatest increase rate
- a second-order approximation needs the Hessian matrix $Hf$

$$f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) := f^{(1)}_{\mathbf{x}_0}(\mathbf{x}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top (Hf(\mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0)$$

- assuming $f$ is locally convex with isotropic $Hf(\mathbf{x}_0) = \frac{1}{\epsilon} I$, the gradient of $f^{(2)}_{\mathbf{x}_0}$ is

$$\nabla f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \nabla f(\mathbf{x}_0) + \frac{1}{\epsilon}(\mathbf{x} - \mathbf{x}_0)$$

- so if we were to minimize this approximation instead of $f$, we would let this gradient vanish and solve for $\mathbf{x}$

$$\arg\min_{\mathbf{x}} f^{(2)}_{\mathbf{x}_0}(\mathbf{x}) = \mathbf{x}_0 - \epsilon \nabla f(\mathbf{x}_0)$$

# gradient descent

- this yields the update rule

$$\mathbf{x}^{(\tau+1)} = \mathbf{x}^{(\tau)} - \epsilon \nabla f(\mathbf{x}^{(\tau)})$$

*i.e.* , we are moving in the direction of the greatest decrease rate such that locally (depending on $\epsilon$)

$$\begin{aligned} f^{(1)}_{\mathbf{x}^{(\tau)}}(\mathbf{x}^{(\tau+1)}) &= f(\mathbf{x}^{(\tau)}) + (\mathbf{x}^{(\tau+1)} - \mathbf{x}^{(\tau)})^\top \nabla f(\mathbf{x}^{(\tau)}) \\ &= f(\mathbf{x}^{(\tau)}) - \epsilon \nabla f(\mathbf{x}^{(\tau)})^\top \nabla f(\mathbf{x}^{(\tau)}) \\ &\leq f(\mathbf{x}^{(\tau)}) \end{aligned}$$

- the step size $\epsilon$ is inversely proportional to the curvature we assume for $f$ at the local minimum

# gradient descent

- this yields the update rule

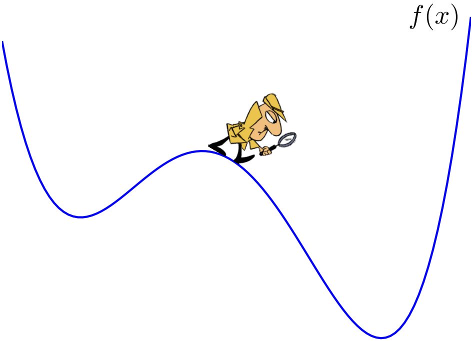
$$\mathbf{x}^{(\tau+1)} = \mathbf{x}^{(\tau)} - \epsilon \nabla f(\mathbf{x}^{(\tau)})$$

*i.e.* , we are moving in the direction of the greatest decrease rate such that locally (depending on $\epsilon$)

$$\begin{aligned} f^{(1)}_{\mathbf{x}^{(\tau)}}(\mathbf{x}^{(\tau+1)}) &= f(\mathbf{x}^{(\tau)}) + (\mathbf{x}^{(\tau+1)} - \mathbf{x}^{(\tau)})^\top \nabla f(\mathbf{x}^{(\tau)}) \\ &= f(\mathbf{x}^{(\tau)}) - \epsilon \nabla f(\mathbf{x}^{(\tau)})^\top \nabla f(\mathbf{x}^{(\tau)}) \\ &\leq f(\mathbf{x}^{(\tau)}) \end{aligned}$$

- the step size $\epsilon$ is inversely proportional to the curvature we assume for $f$ at the local minimum

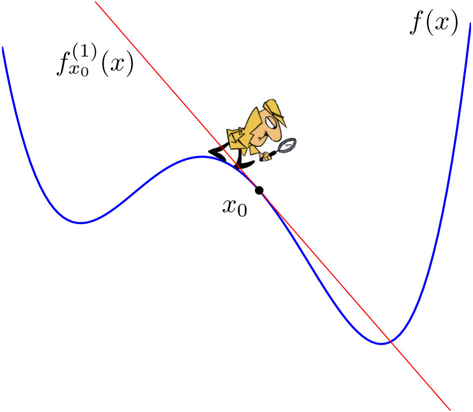
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

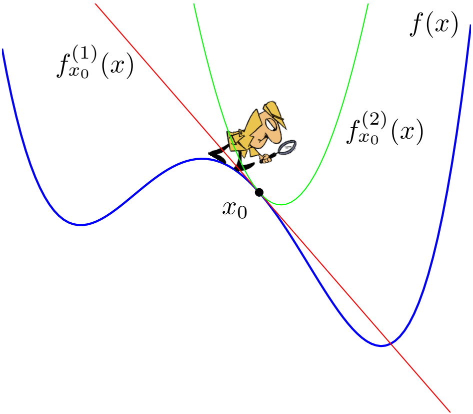
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

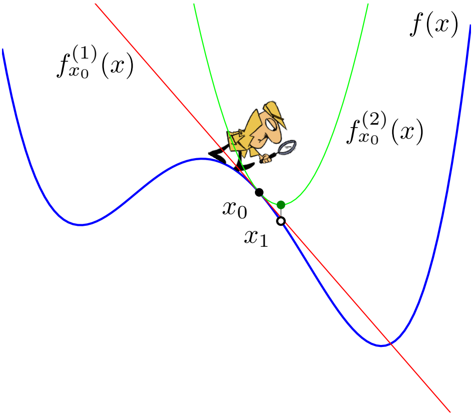
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

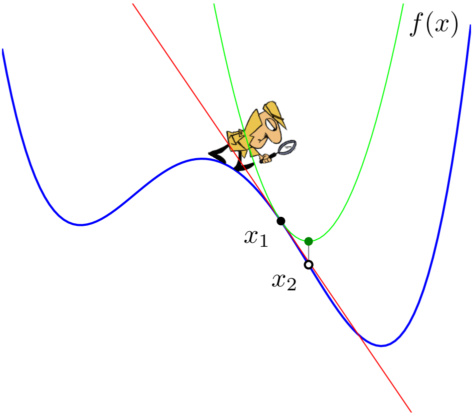
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

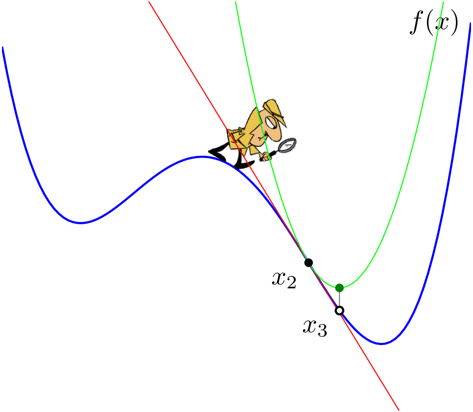
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

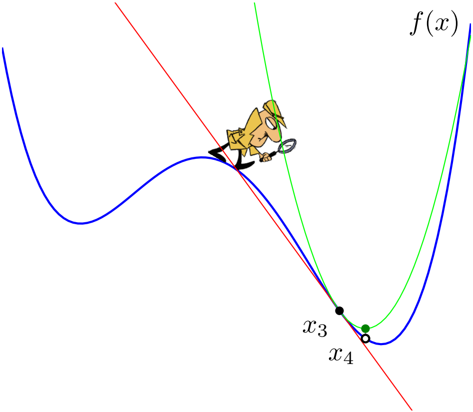
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

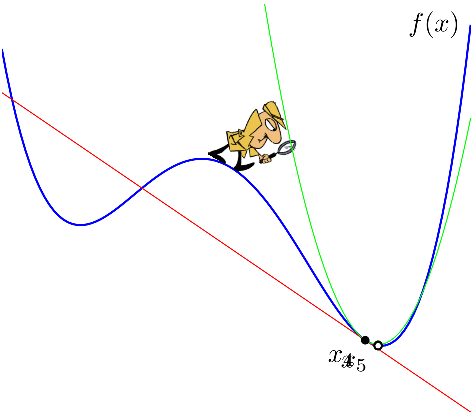
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

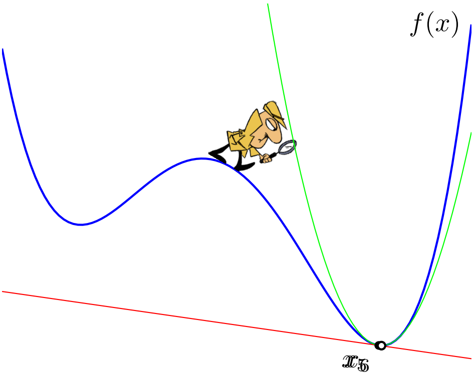
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

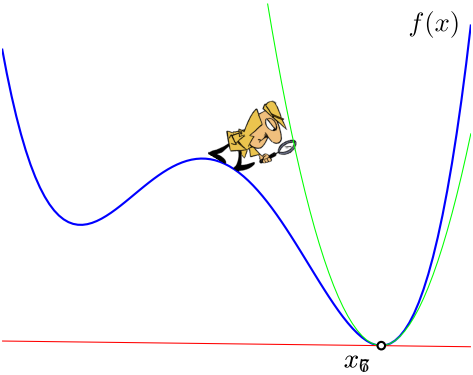
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

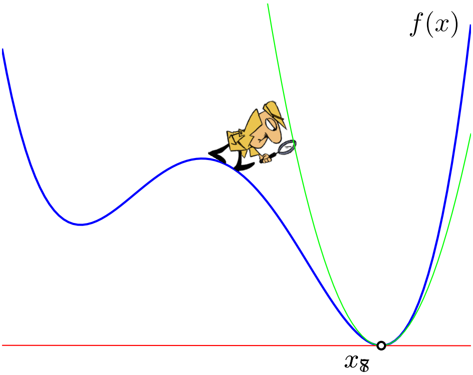
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

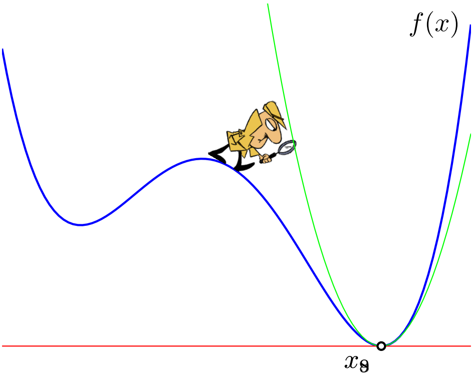
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

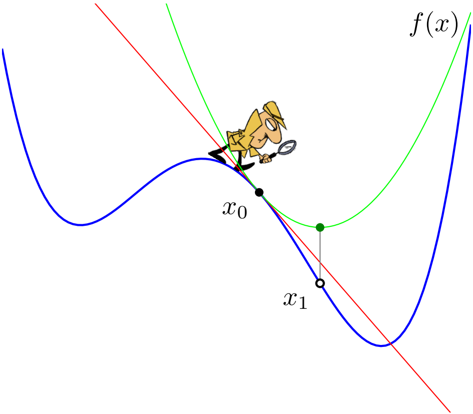
# gradient descent in one dimension




- $\epsilon = 0.05$: converges to local minimum

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

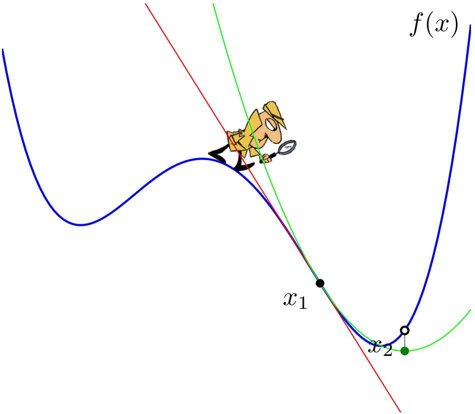
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

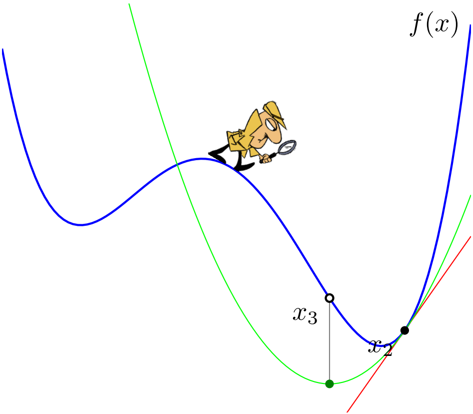
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

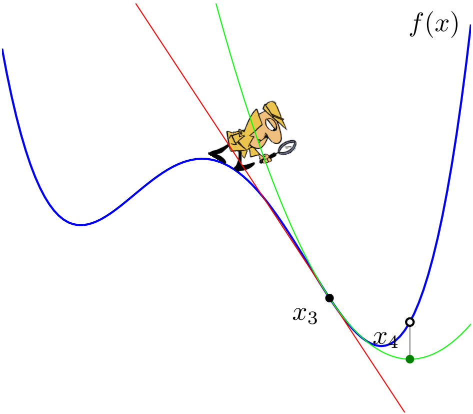
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

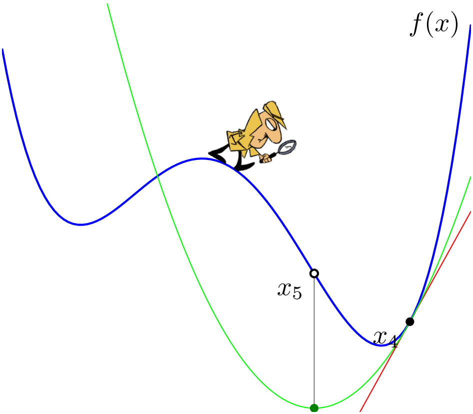
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

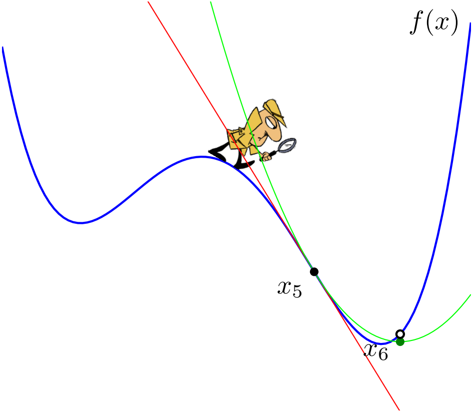
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

Artwork credit: `https://the-fox-after-dark.deviantart.com/`

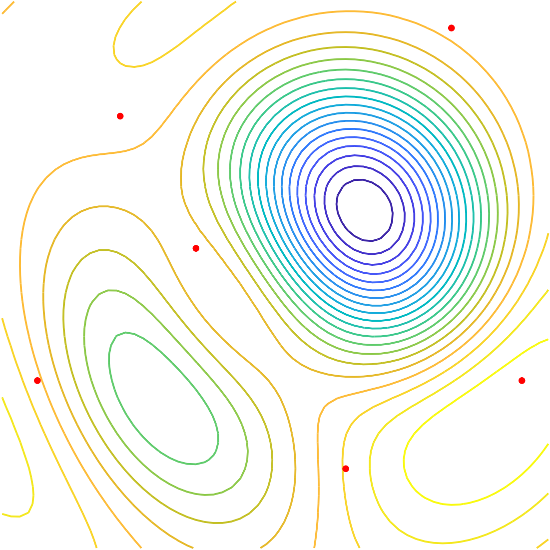
# gradient descent in one dimension




- $\epsilon = 0.14$: $1/\epsilon$ less than actual curvature, does not converge

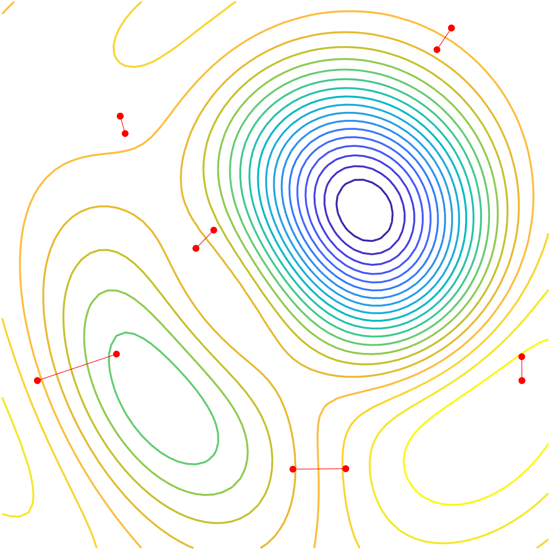
Artwork credit: `https://the-fox-after-dark.deviantart.com/`

# gradient descent in two dimensions



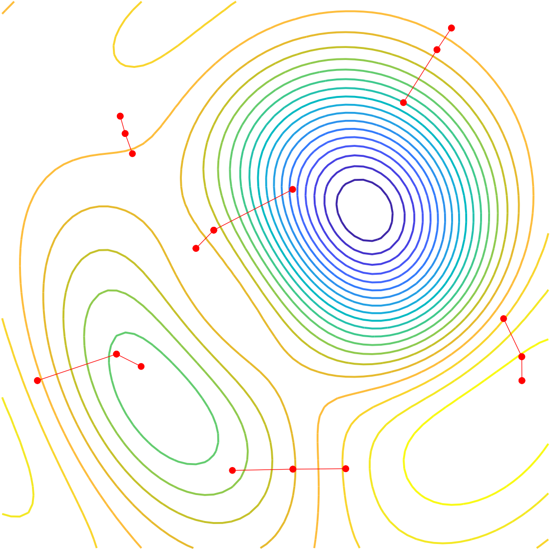
$\epsilon = 0.14$, iteration $0$

# gradient descent in two dimensions



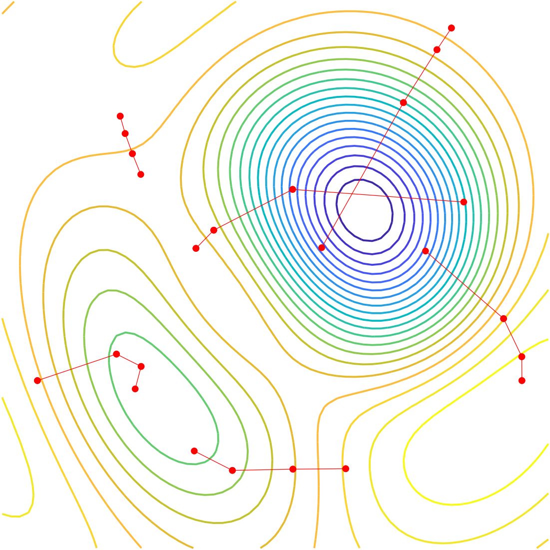
$\epsilon = 0.14$, iteration 1

# gradient descent in two dimensions



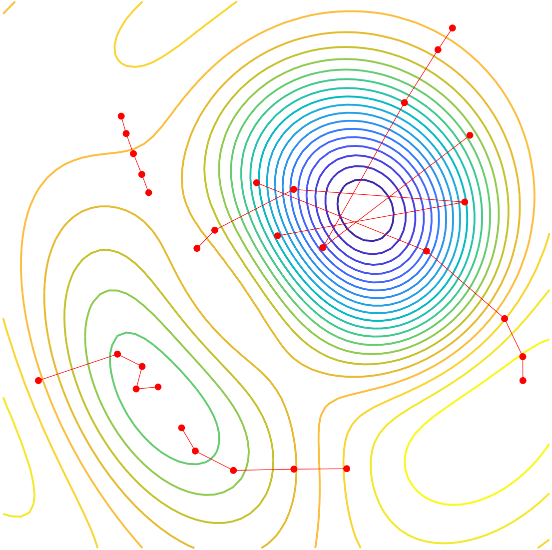
$\epsilon = 0.14$, iteration 2

# gradient descent in two dimensions



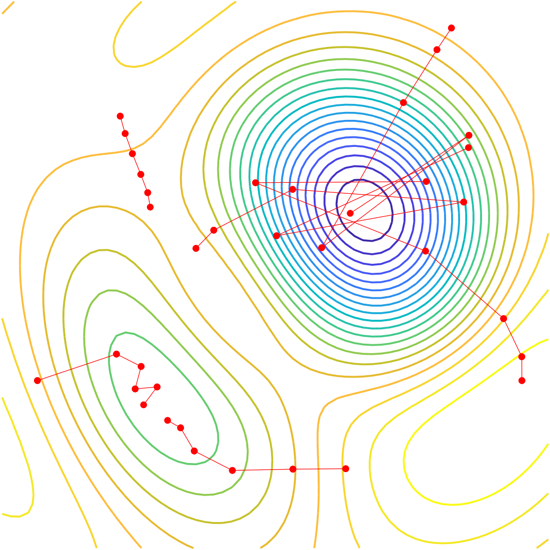
$\epsilon = 0.14$, iteration 3

# gradient descent in two dimensions



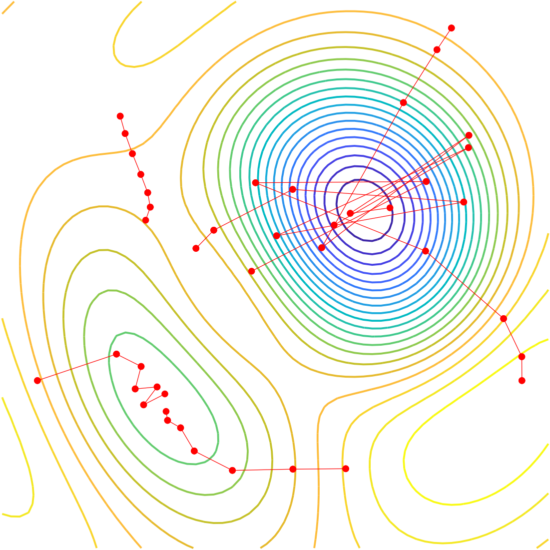
$\epsilon = 0.14$, iteration $4$

# gradient descent in two dimensions



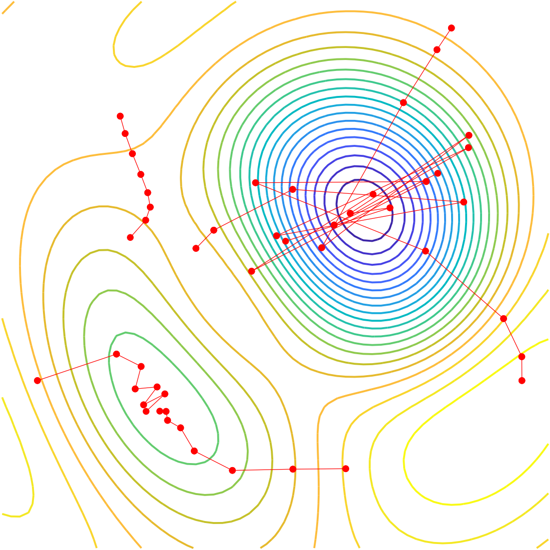
$\epsilon = 0.14$, iteration 5

# gradient descent in two dimensions



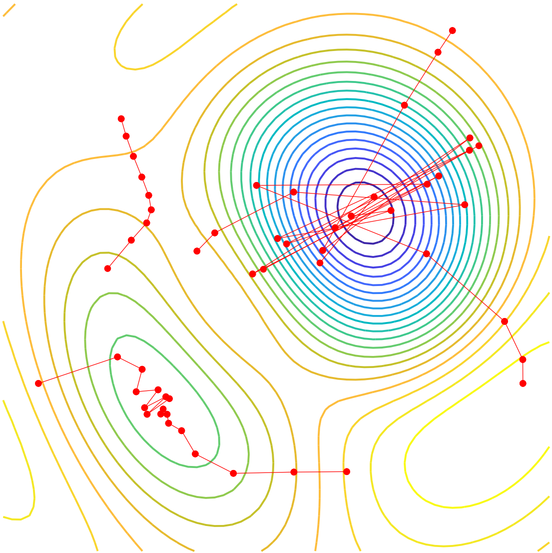
$\epsilon = 0.14$, iteration 6

# gradient descent in two dimensions



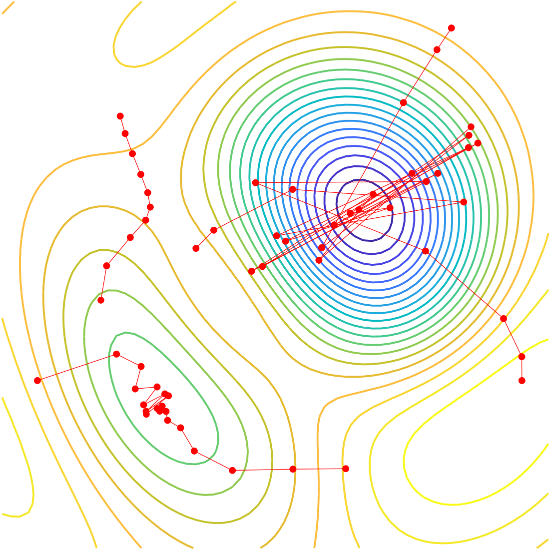
$\epsilon = 0.14$, iteration 7

# gradient descent in two dimensions



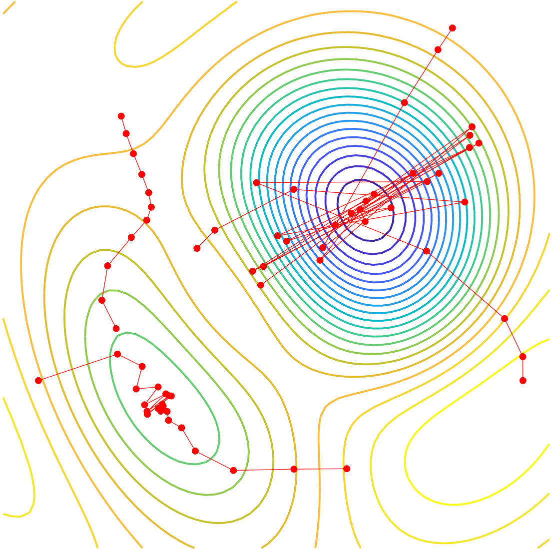
$\epsilon = 0.14$, iteration 8

# gradient descent in two dimensions



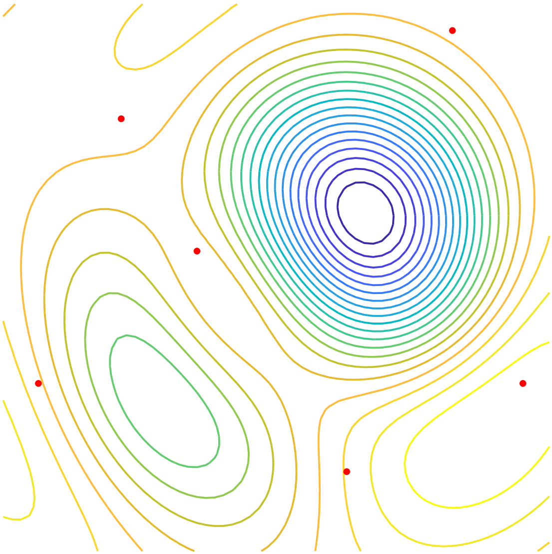
$\epsilon = 0.14$, iteration $9$

# gradient descent in two dimensions



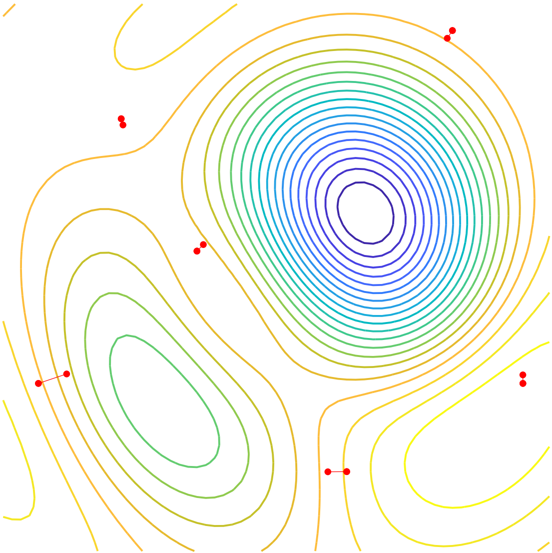
$\epsilon = 0.14$, iteration 10

# gradient descent in two dimensions



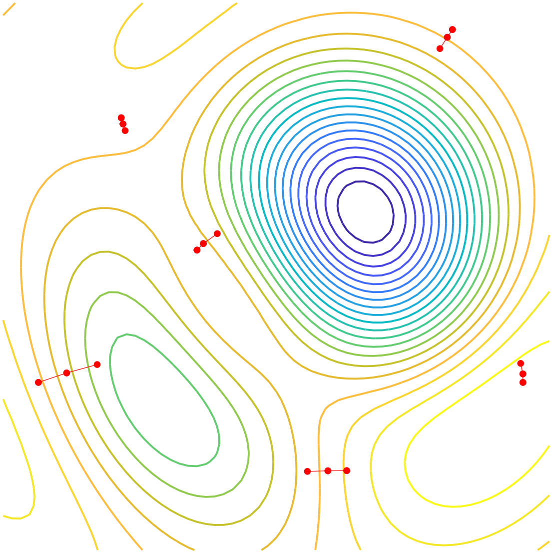
$\epsilon = 0.05$, iteration $0$

# gradient descent in two dimensions



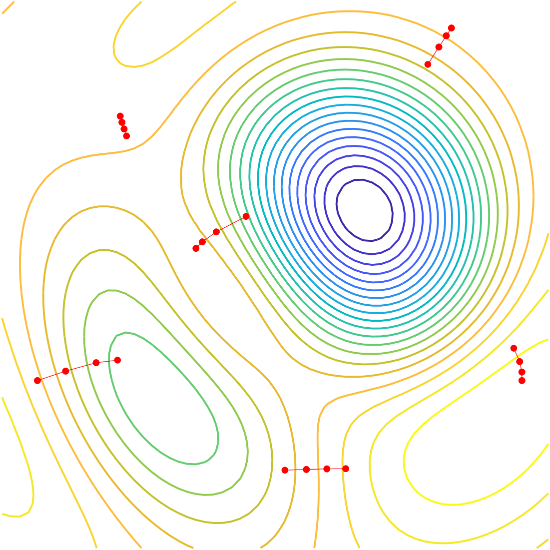
$\epsilon = 0.05$, iteration 1

# gradient descent in two dimensions



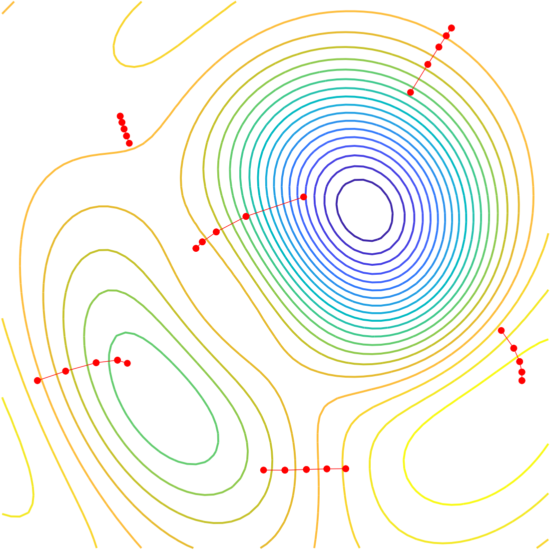
$\epsilon = 0.05$, iteration 2

# gradient descent in two dimensions



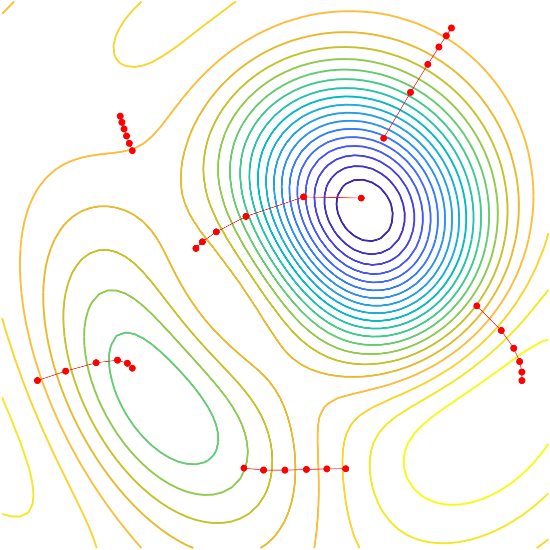
$\epsilon = 0.05$, iteration 3

# gradient descent in two dimensions



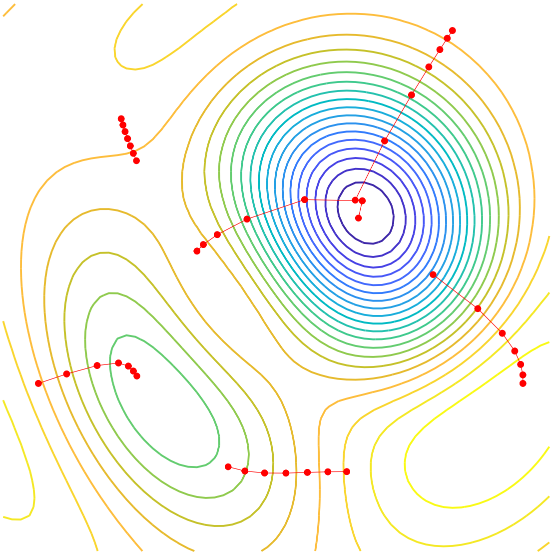
$\epsilon = 0.05$, iteration 4

# gradient descent in two dimensions



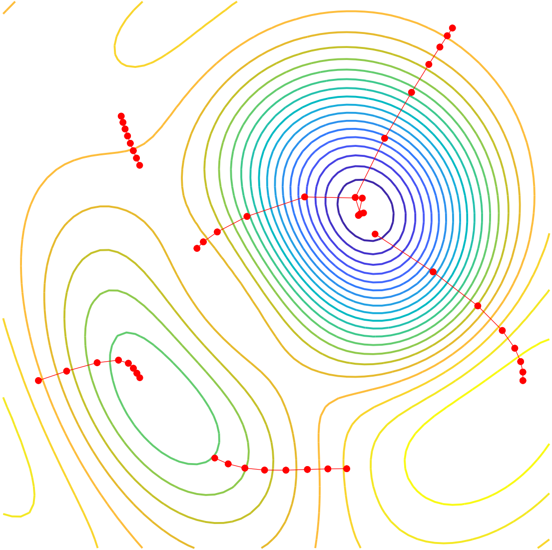
$\epsilon = 0.05$, iteration 5

# gradient descent in two dimensions



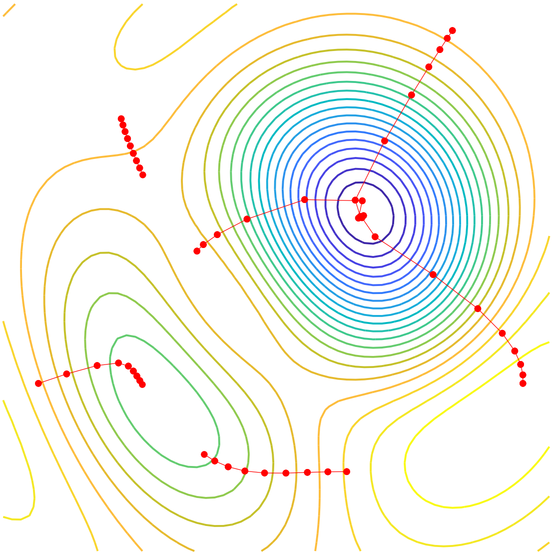
$\epsilon = 0.05$, iteration 6

# gradient descent in two dimensions



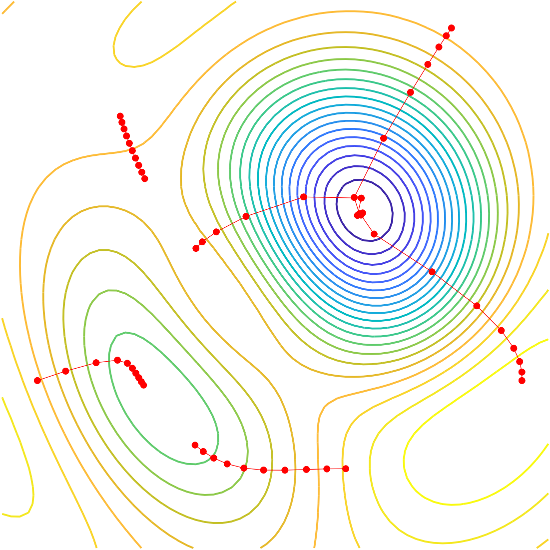
$\epsilon = 0.05$, iteration 7

# gradient descent in two dimensions



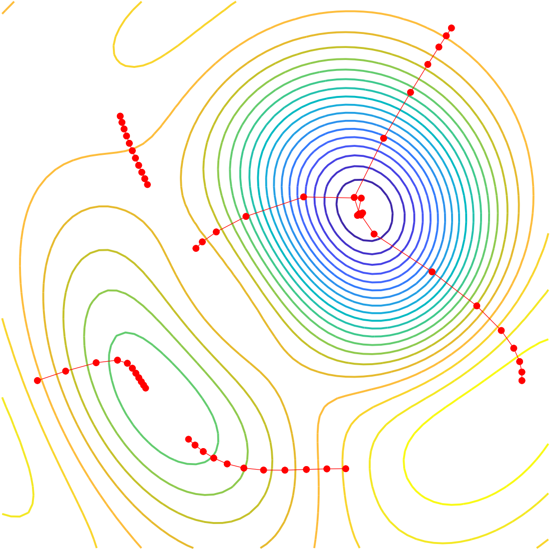
$\epsilon = 0.05$, iteration $8$

# gradient descent in two dimensions



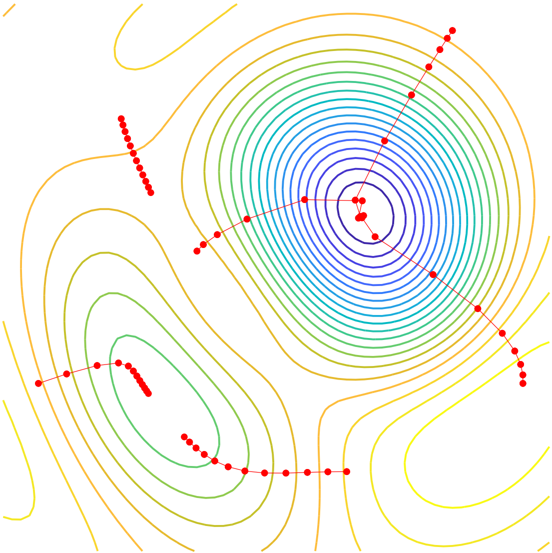
$\epsilon = 0.05$, iteration 9

# gradient descent in two dimensions



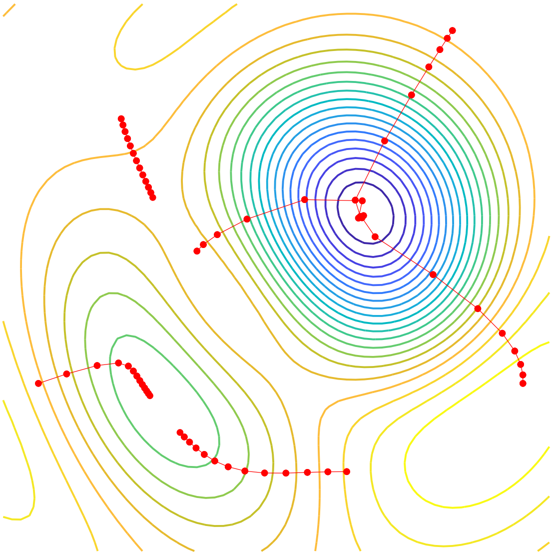
$\epsilon = 0.05$, iteration 10

# gradient descent in two dimensions



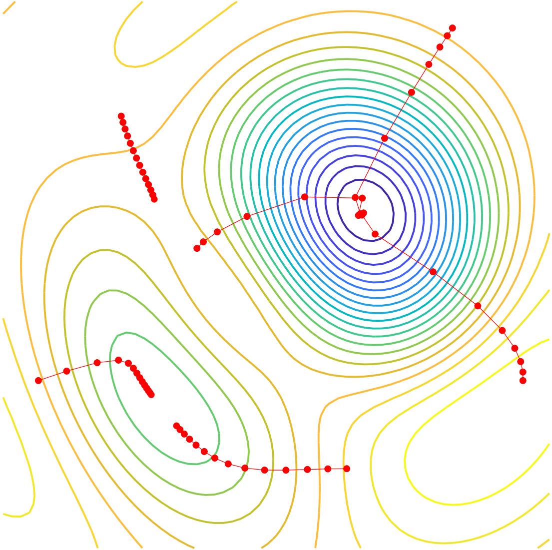
$\epsilon = 0.05$, iteration 11

# gradient descent in two dimensions



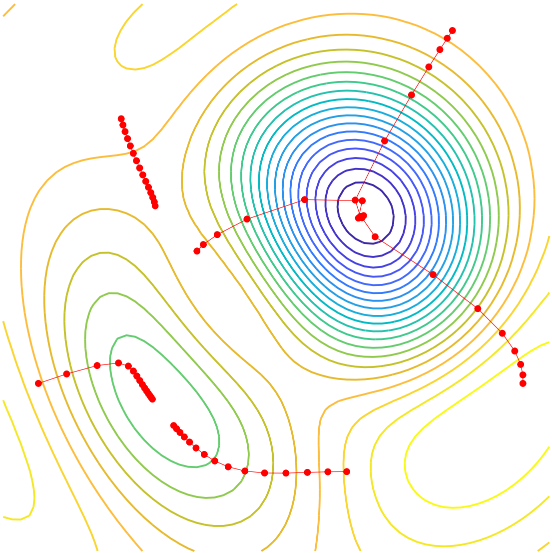
$\epsilon = 0.05$, iteration 12

# gradient descent in two dimensions



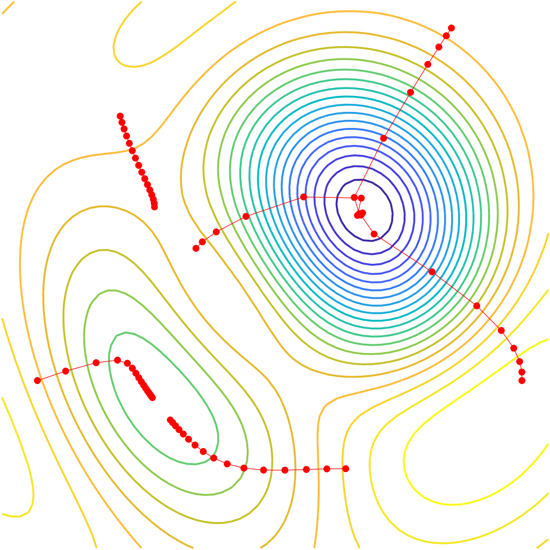
$\epsilon = 0.05$, iteration 13

# gradient descent in two dimensions



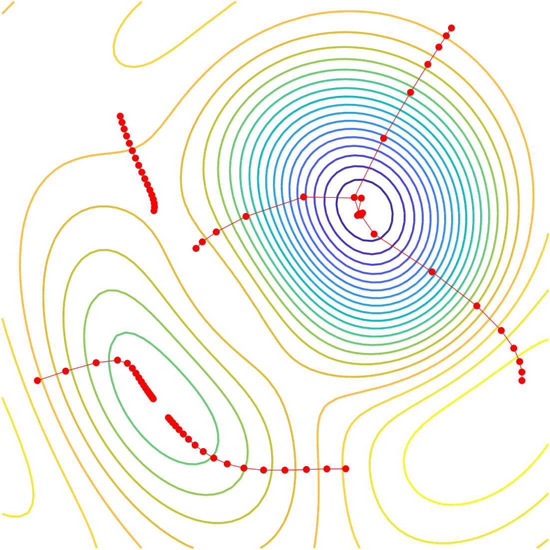
$\epsilon = 0.05$, iteration 14

# gradient descent in two dimensions



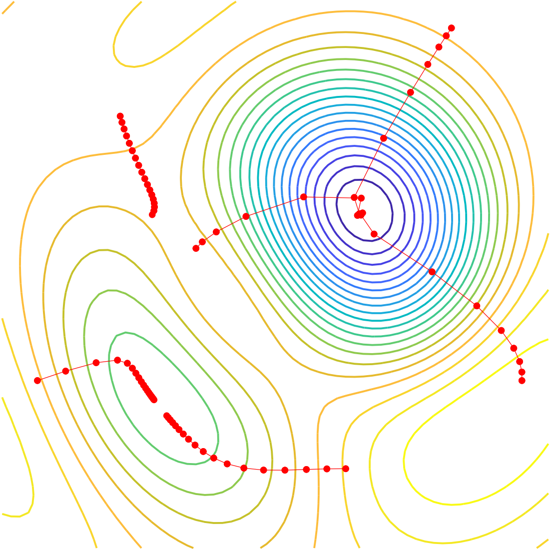
$\epsilon = 0.05$, iteration 15

# gradient descent in two dimensions



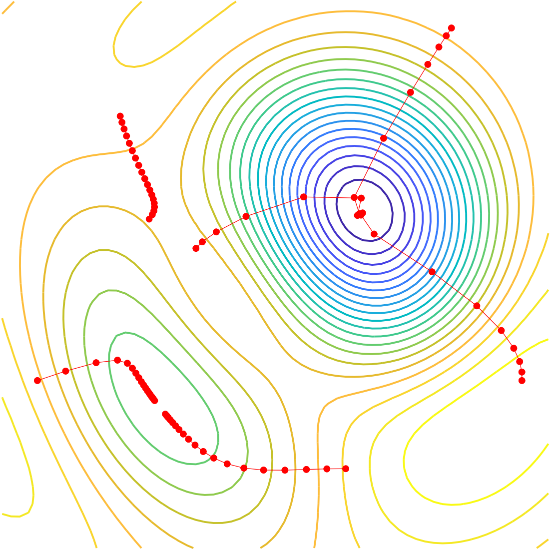
$\epsilon = 0.05$, iteration 16

# gradient descent in two dimensions



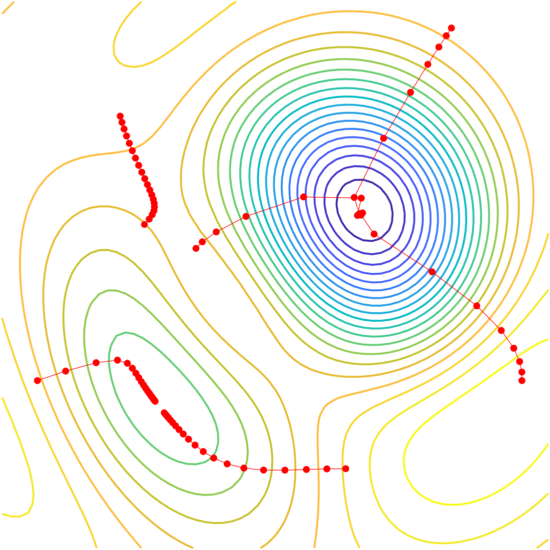
$\epsilon = 0.05$, iteration 17

# gradient descent in two dimensions



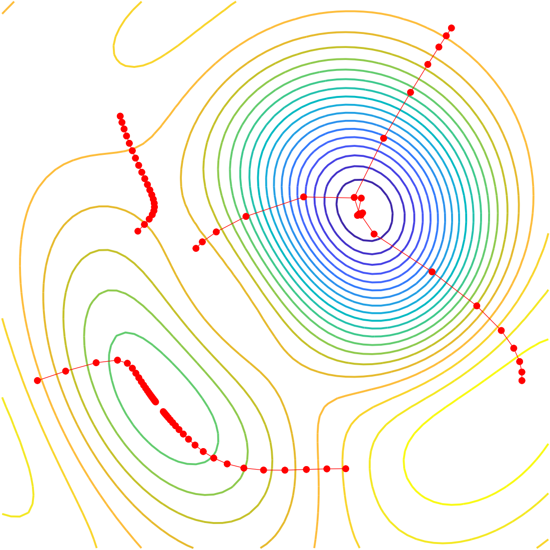
$\epsilon = 0.05$, iteration 18

# gradient descent in two dimensions



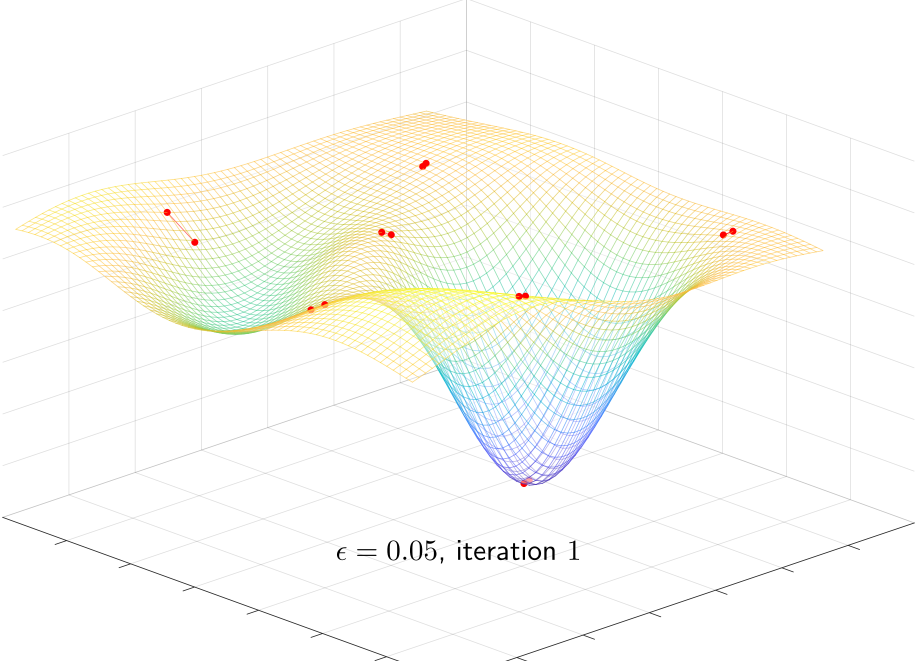
$\epsilon = 0.05$, iteration 19

# gradient descent in two dimensions



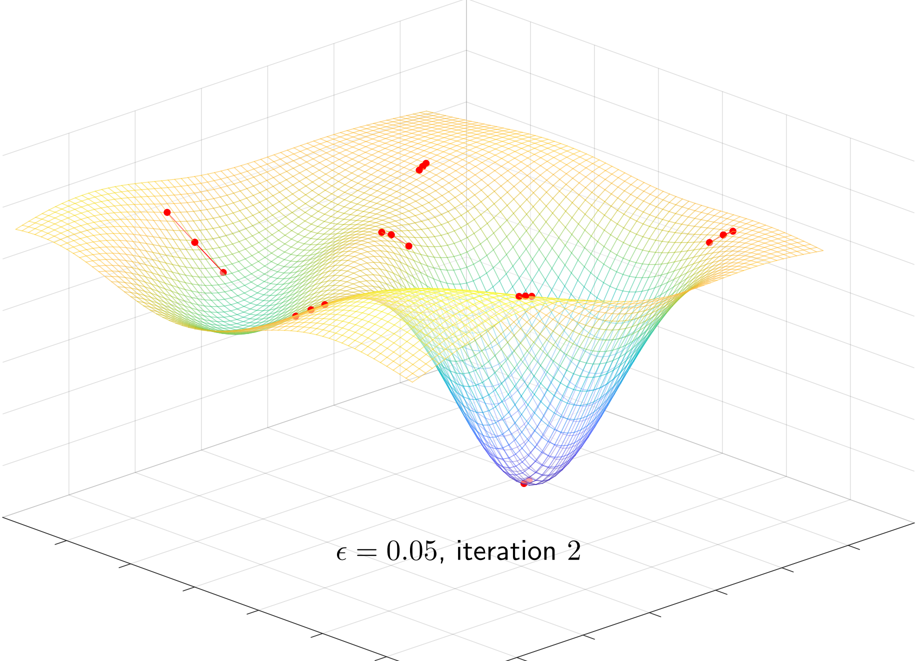
$\epsilon = 0.05$, iteration 20

# gradient descent in two dimensions

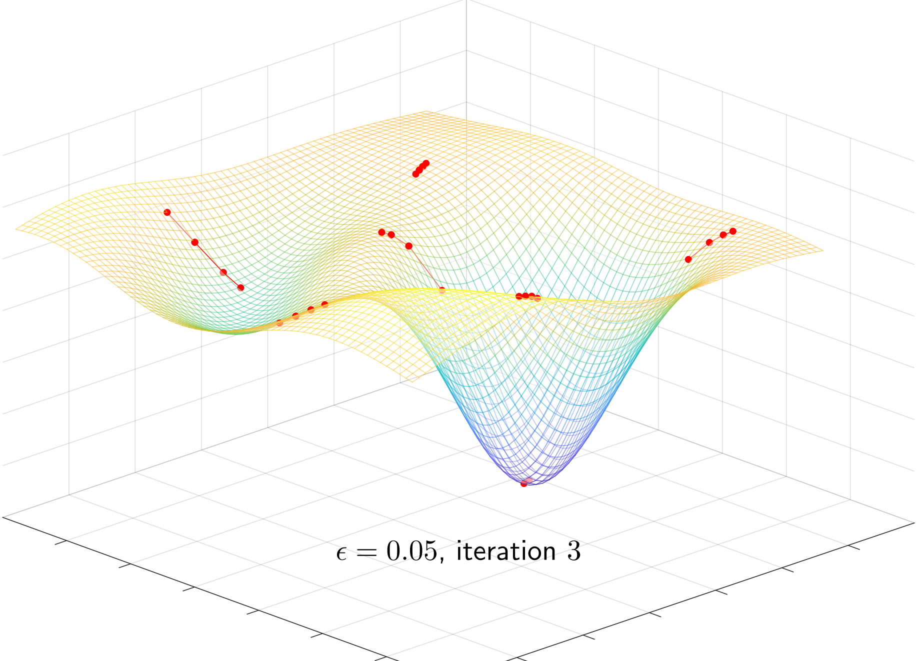


$\epsilon = 0.05$, iteration 1

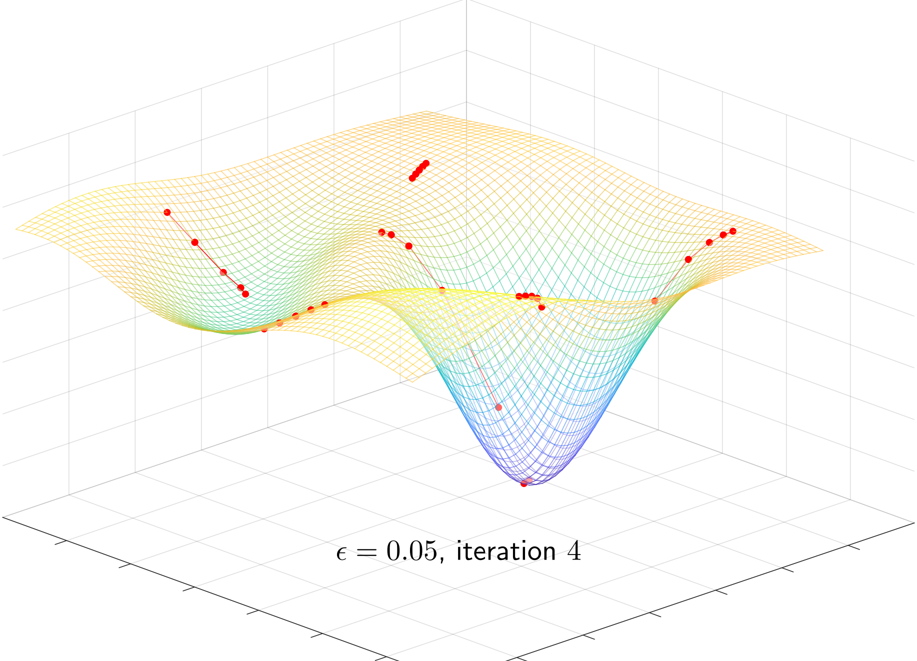
# gradient descent in two dimensions



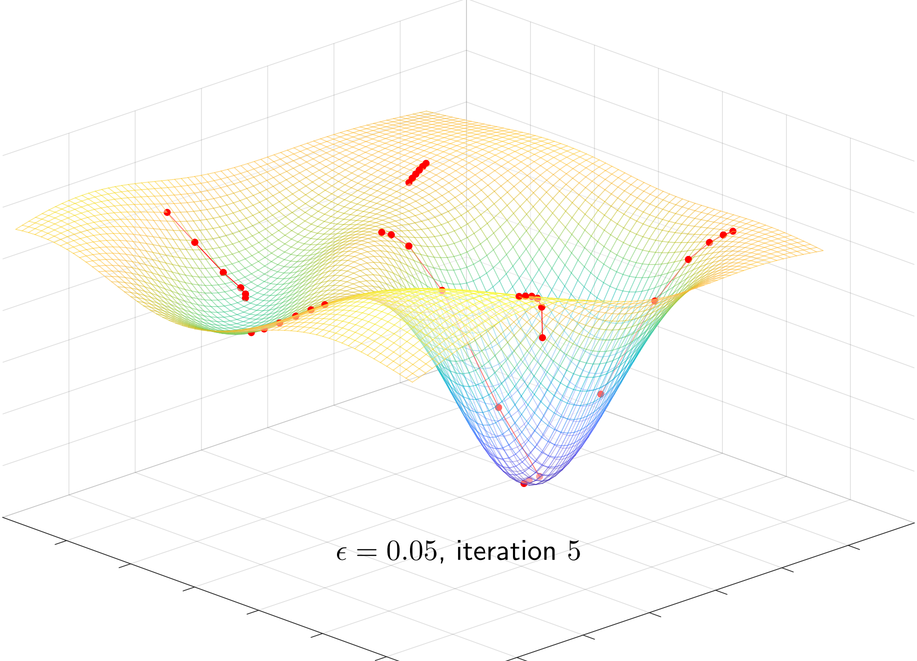
# gradient descent in two dimensions



# gradient descent in two dimensions

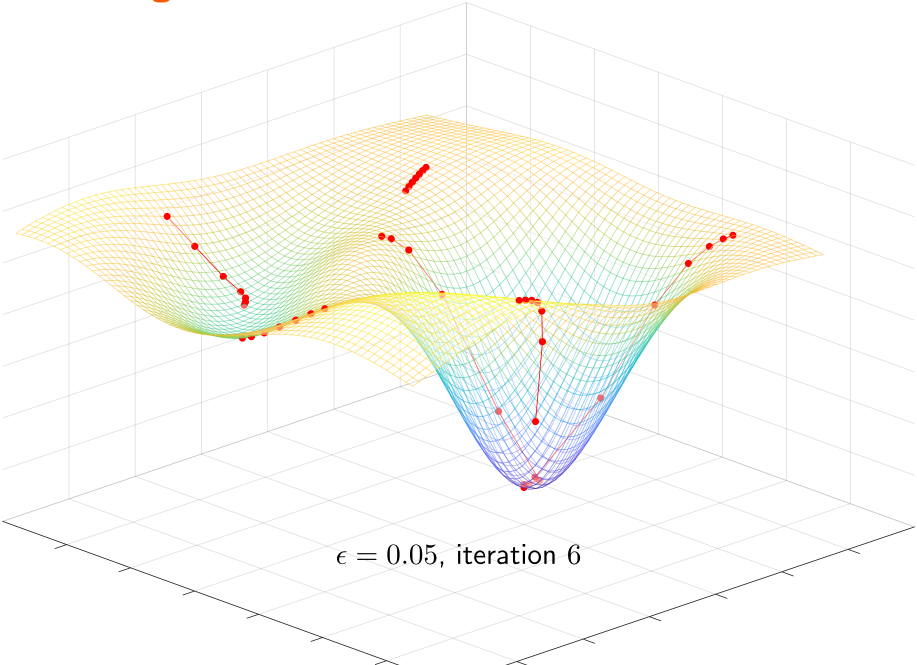


# gradient descent in two dimensions



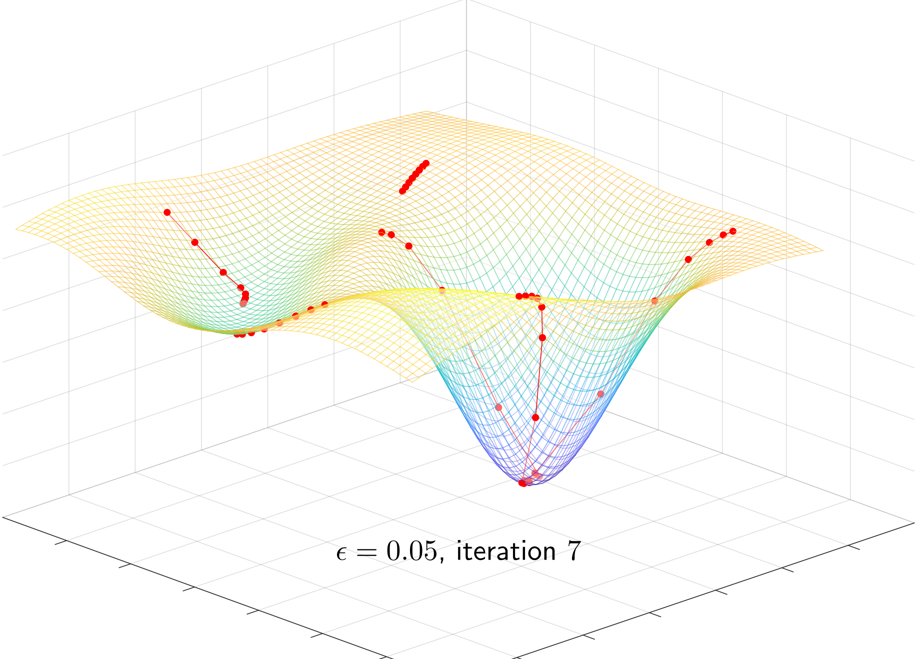
$\epsilon = 0.05$, iteration 5

# gradient descent in two dimensions



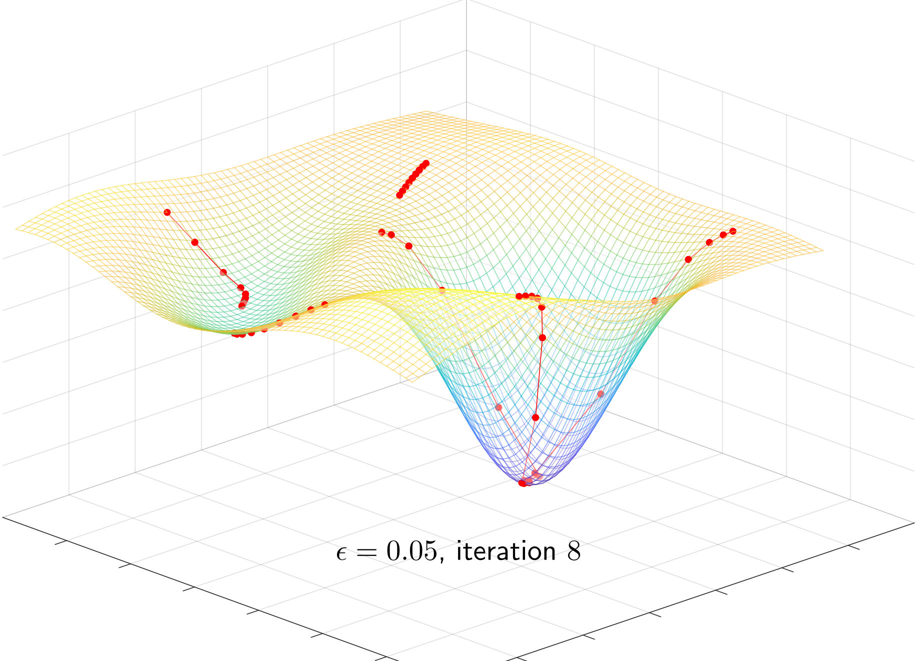
$\epsilon = 0.05$, iteration 6

# gradient descent in two dimensions

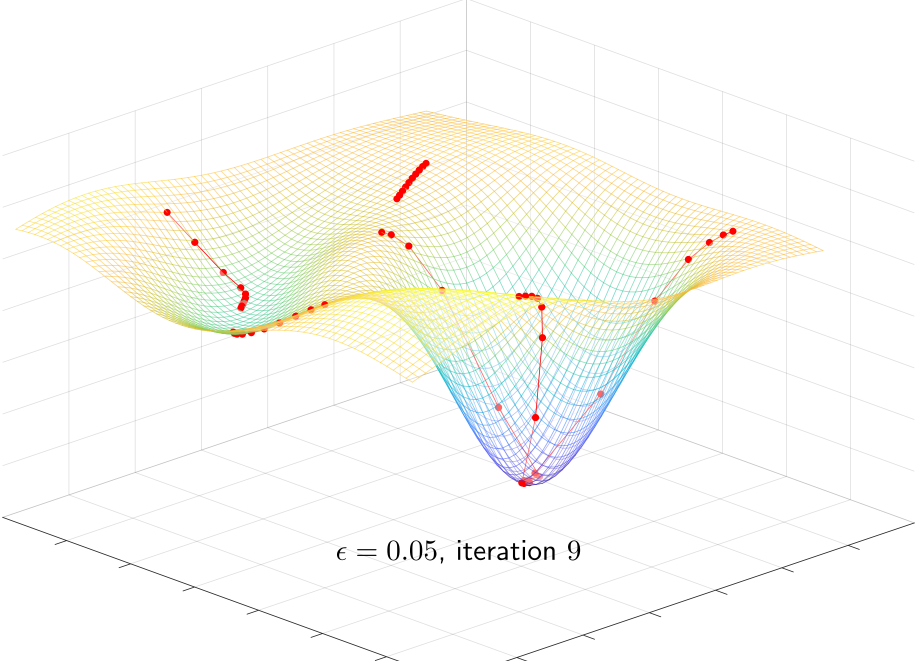


$\epsilon = 0.05$, iteration 7

# gradient descent in two dimensions

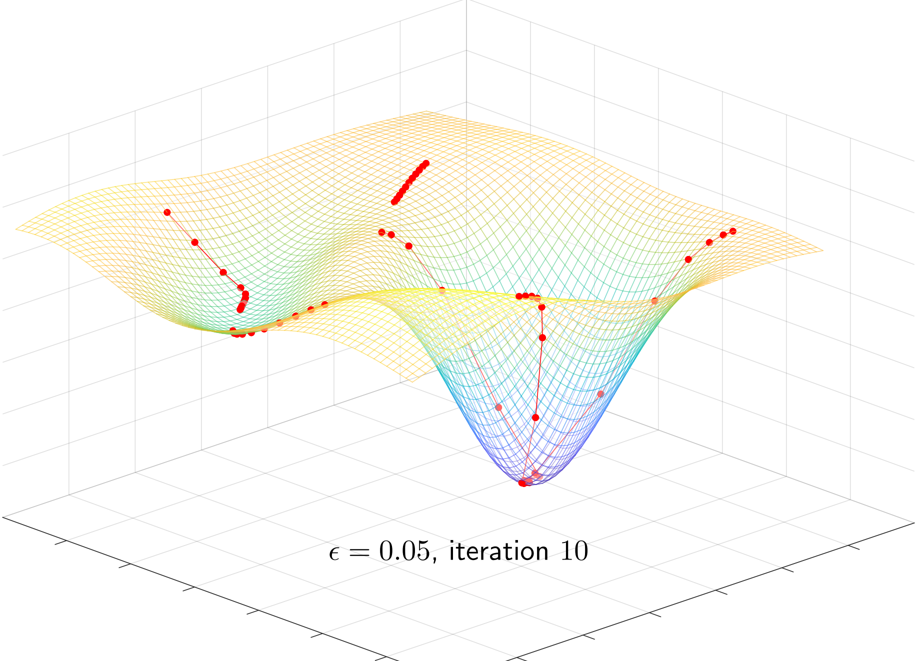


# gradient descent in two dimensions



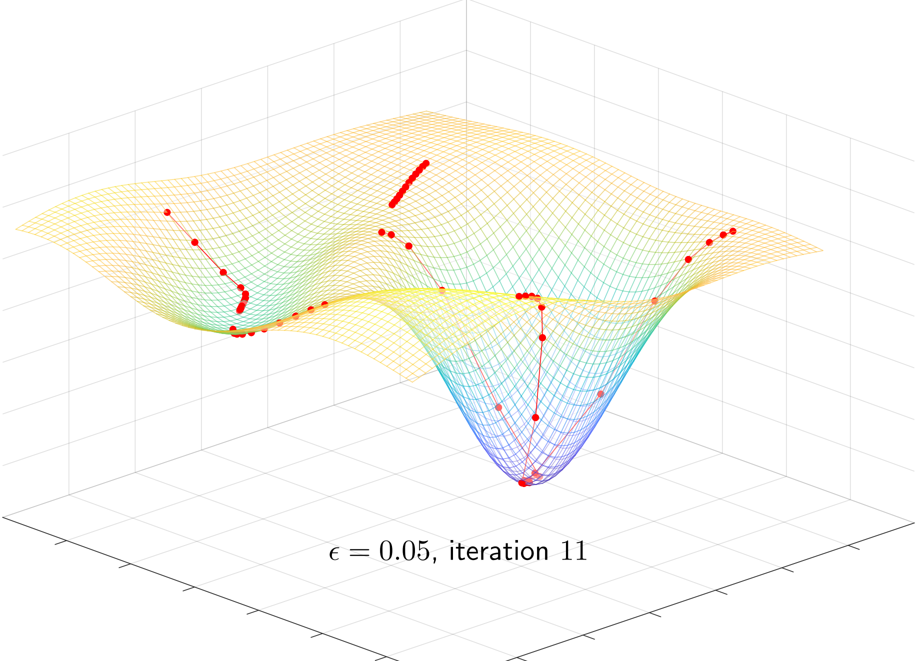
$\epsilon = 0.05$, iteration 9

# gradient descent in two dimensions



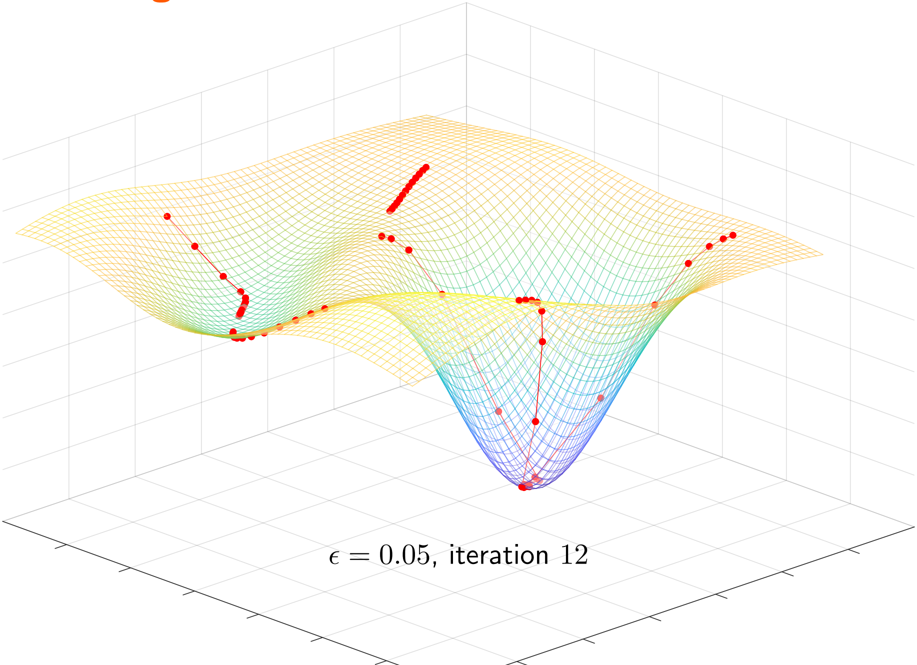
$\epsilon = 0.05$, iteration 10

# gradient descent in two dimensions



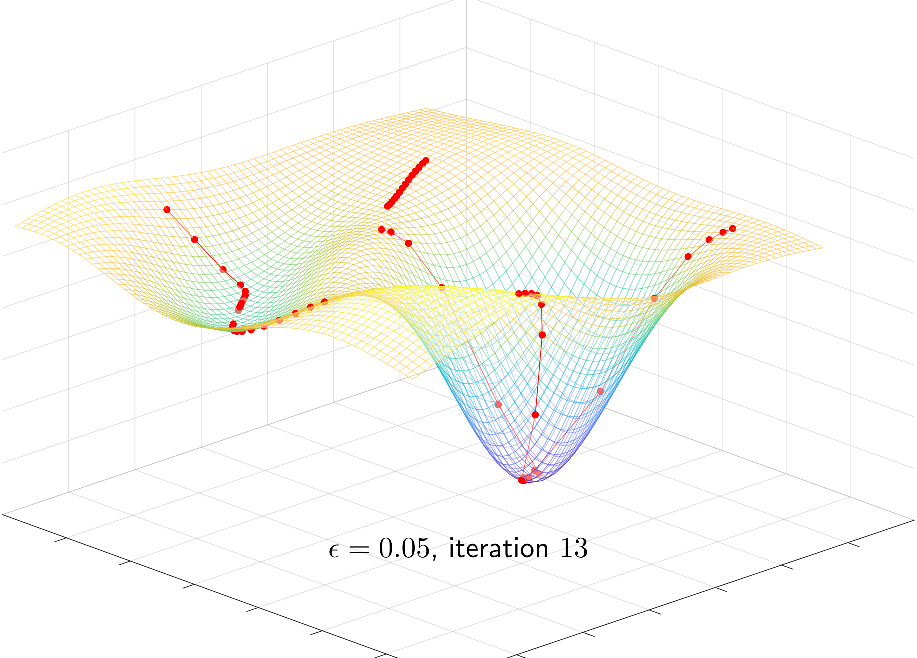
$\epsilon = 0.05$, iteration 11

# gradient descent in two dimensions



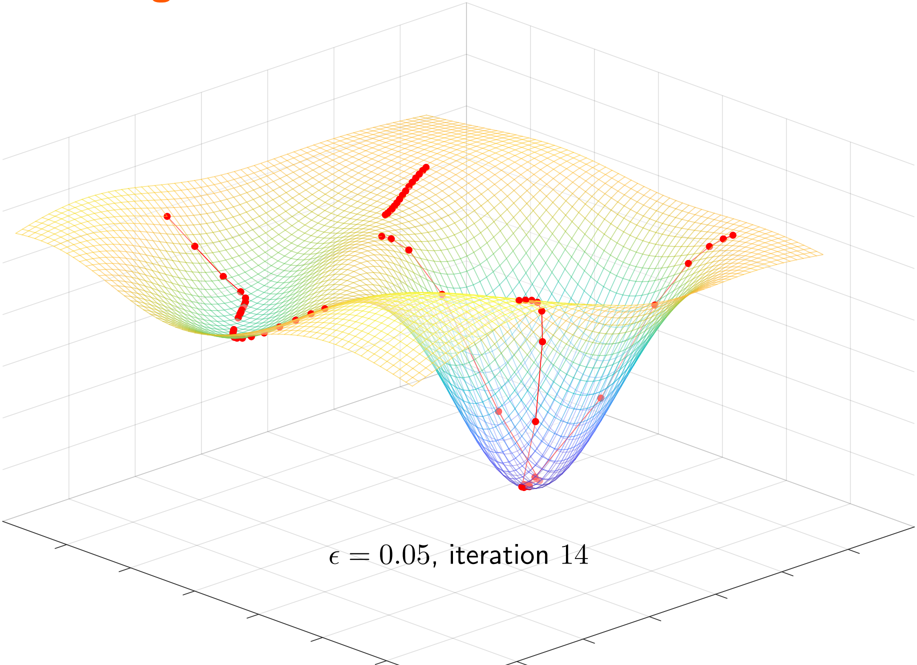
$\epsilon = 0.05$, iteration 12

# gradient descent in two dimensions



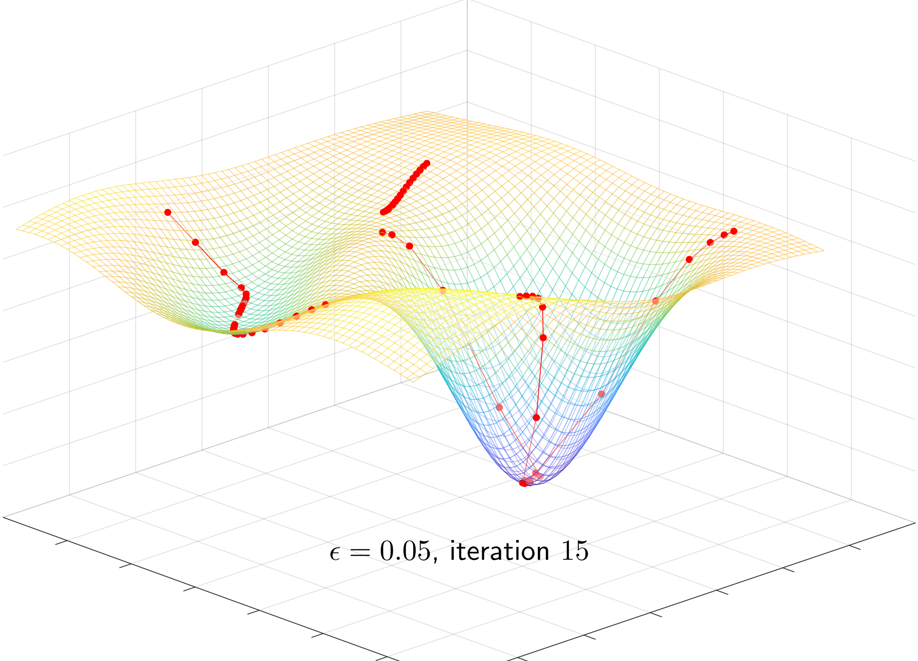
$\epsilon = 0.05$, iteration 13

# gradient descent in two dimensions



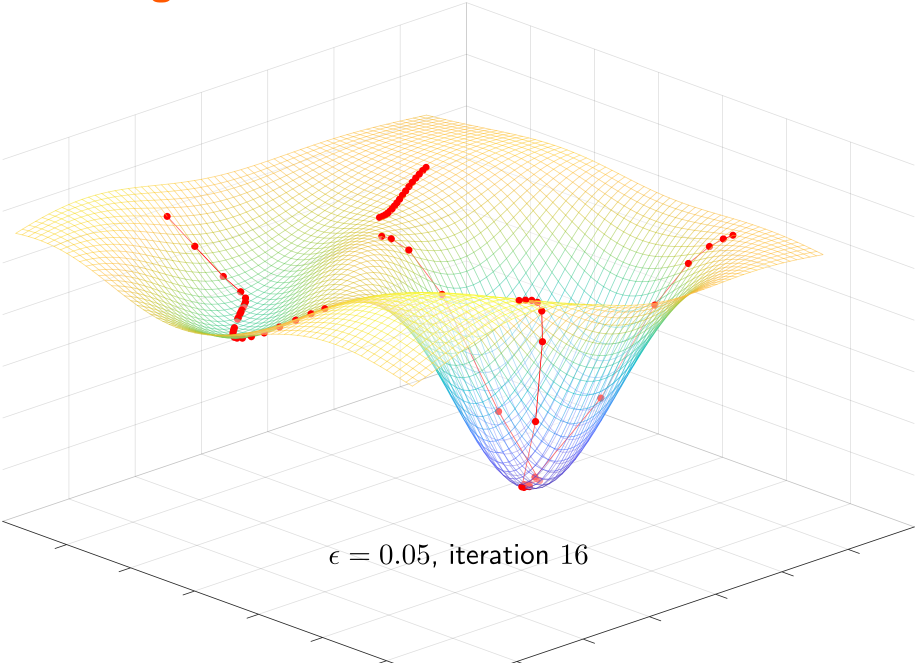
$\epsilon = 0.05$, iteration 14

# gradient descent in two dimensions



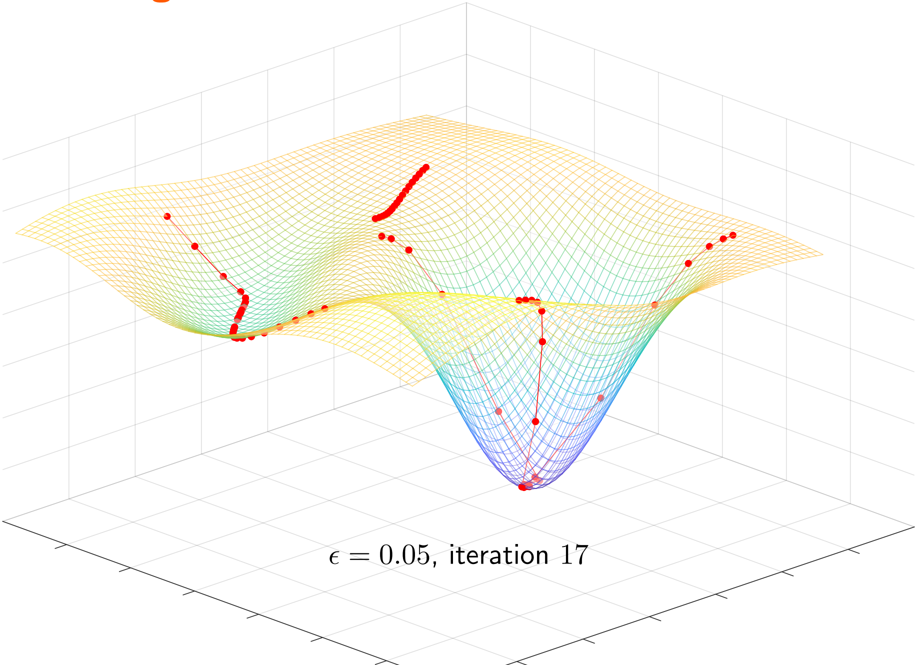
$\epsilon = 0.05$, iteration 15

# gradient descent in two dimensions



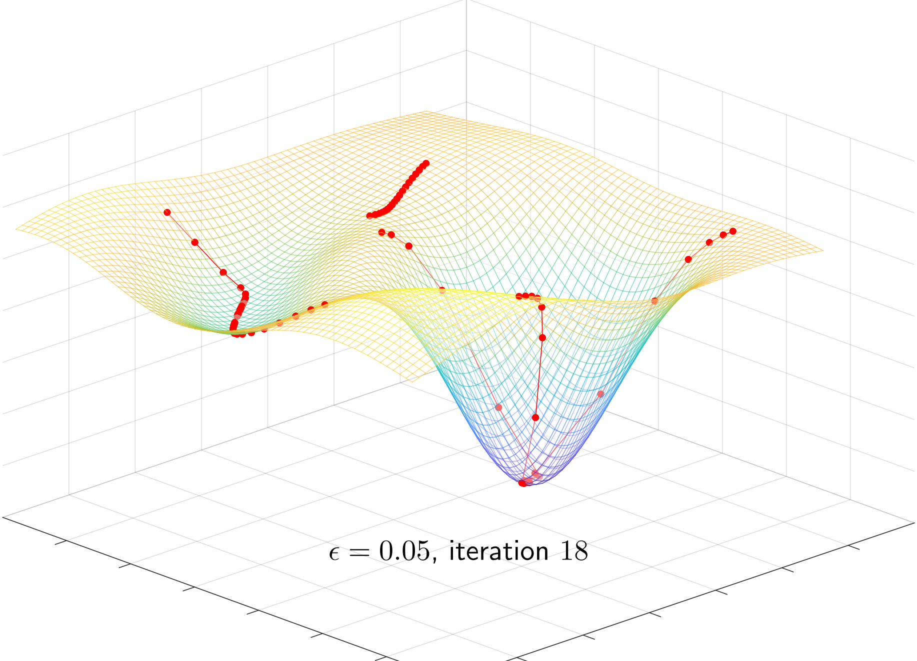
$\epsilon = 0.05$, iteration 16

# gradient descent in two dimensions



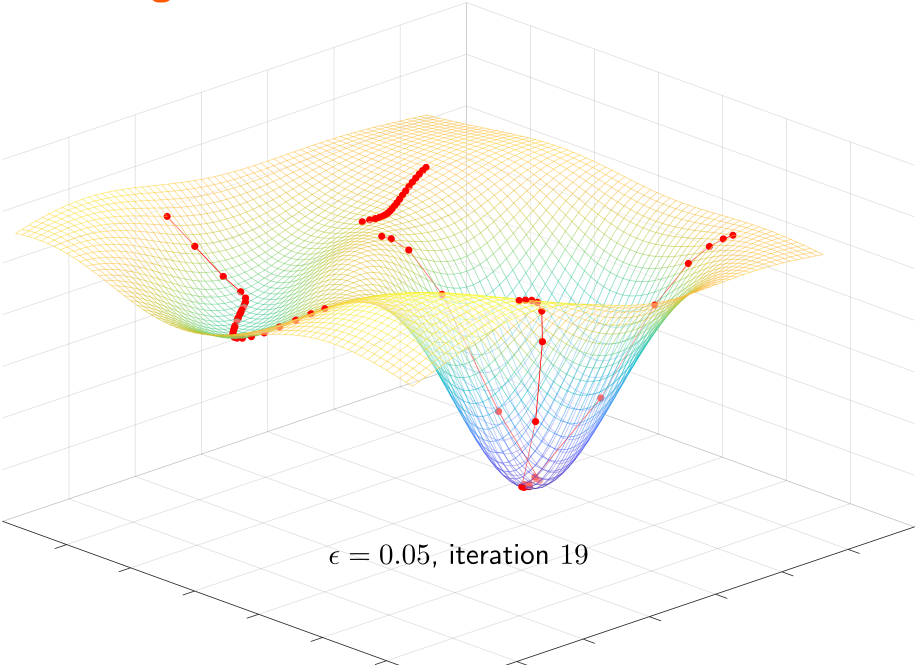
$\epsilon = 0.05$, iteration 17

# gradient descent in two dimensions



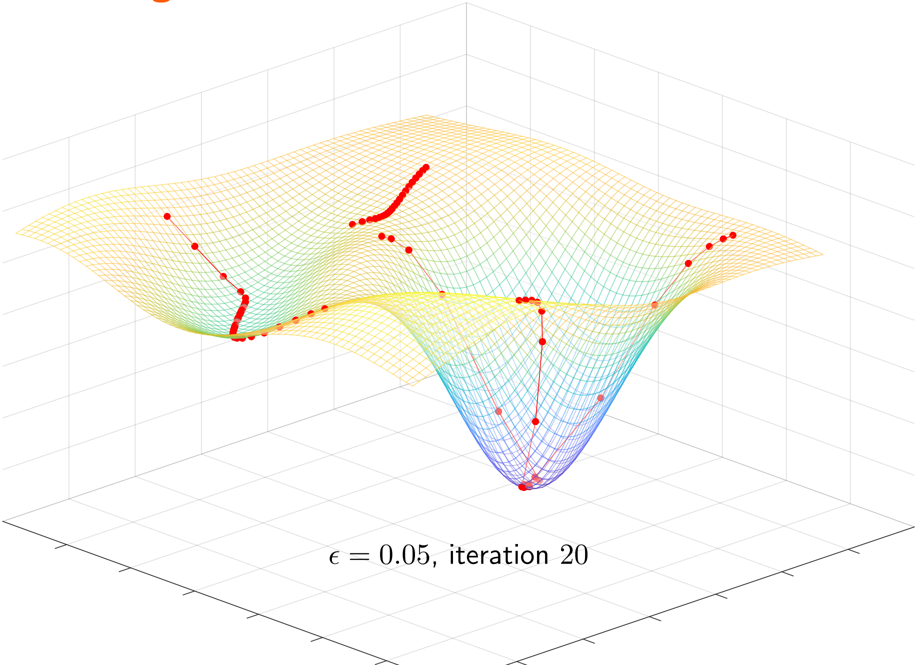
$\epsilon = 0.05$, iteration 18

# gradient descent in two dimensions



$\epsilon = 0.05$, iteration 19

# gradient descent in two dimensions



$\epsilon = 0.05$, iteration 20

# problems

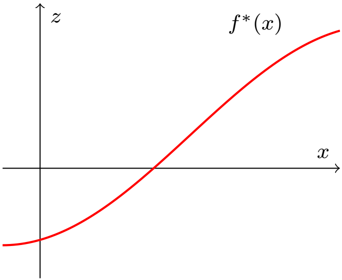
- $f$ non-convex: local minima
- $d \times d$ Hessian matrix too expensive ($d$ can be millions): unknown curvature
- high condition number: elongated regions
- plateaus, saddle points: no progress
- $\nabla f = \sum_{i=1}^{n} \nabla f_i$ itself too expensive ($n$ can also be millions)

# problems

- $f$ non-convex: local minima
- $d \times d$ Hessian matrix too expensive ($d$ can be millions): unknown curvature
- high condition number: elongated regions
- plateaus, saddle points: no progress
- $\nabla f = \sum_{i=1}^{n} \nabla f_i$ itself too expensive ($n$ can also be millions)

# sequential estimation

[Robbins and Monro 1951]




- suppose $f^*$ is the expectation of random variable $z$ conditional on $x$, and $f$ is its empirical estimate on $n$ samples

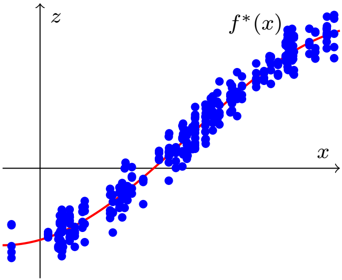
$$f^*(x) := \mathbb{E}[z|x] \qquad f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

- we would like to estimate a root $x^*$ of $f$ where $f(x^*) = 0$

Robbins and Monro. AMS 1951. A Stochastic Approximation Method.

# sequential estimation

**[Robbins and Monro 1951]**




- suppose $f^*$ is the expectation of random variable $z$ conditional on $x$, and $f$ is its empirical estimate on $n$ samples

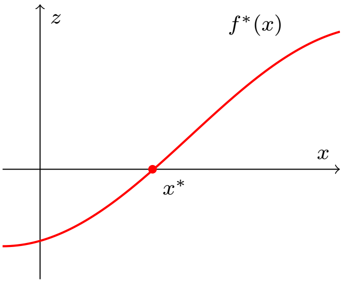
$$f^*(x) := \mathbb{E}[z|x] \qquad f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

- we would like to estimate a root $x^*$ of $f$ where $f(x^*) = 0$

Robbins and Monro. AMS 1951. A Stochastic Approximation Method.

# sequential estimation

[Robbins and Monro 1951]




- suppose $f^*$ is the expectation of random variable $z$ conditional on $x$, and $f$ is its empirical estimate on $n$ samples

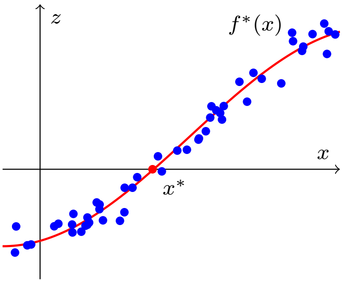
$$f^*(x) := \mathbb{E}[z|x] \qquad f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

- we would like to estimate a root $x^*$ of $f$ where $f(x^*) = 0$

Robbins and Monro. AMS 1951. A Stochastic Approximation Method.

# sequential estimation

[Robbins and Monro 1951]




- then we can estimate $x^*$ sequentially

$$x^{(\tau+1)} = x^{(\tau)} - \epsilon_\tau z(x^{(\tau)}) = x^{(\tau)} - \epsilon_\tau f_i(x^{(\tau)})$$

where $z(x^{(\tau)})$ is an observation of $z$ when $x = x^{(\tau)}$ and $i$ is a random index in $\{1, \ldots, n\}$

Robbins and Monro. AMS 1951. A Stochastic Approximation Method.

# sufficient conditions for convergence

- successive corrections decrease in magnitude

$$\lim_{\tau \to \infty} \epsilon_\tau = 0$$

- the algorithm does not converge short of the root

$$\sum_{\tau=1}^{\infty} \epsilon_\tau = \infty$$

- the accumulated "noise" has finite variance

$$\sum_{\tau=1}^{\infty} \epsilon_\tau^2 < \infty$$

Robbins and Monro. AMS 1951. A Stochastic Approximation Method.

# online gradient descent

- now, replace $x$ by the parameters $\boldsymbol{\theta}$ of our model, and $f$ by $\nabla E$, the gradient of our empirical risk
- the update rule becomes

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon_\tau \nabla E_i(\boldsymbol{\theta}^{(\tau)})$$

- and, under the same conditions, it converges to a root of

$$\nabla E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla E_i(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla L(f(\mathbf{x}_i; \boldsymbol{\theta}), t_i)$$

  that is, to a local minimum of $E$
- mini-batch gradient descent is similar but with less "stochastic noise"

# online gradient descent

- now, replace $x$ by the parameters $\boldsymbol{\theta}$ of our model, and $f$ by $\nabla E$, the gradient of our empirical risk
- the update rule becomes

$$\boldsymbol{\theta}^{(\tau+1)} \leftarrow \boldsymbol{\theta}^{(\tau)} - \epsilon_\tau \nabla E_i(\boldsymbol{\theta}^{(\tau)})$$

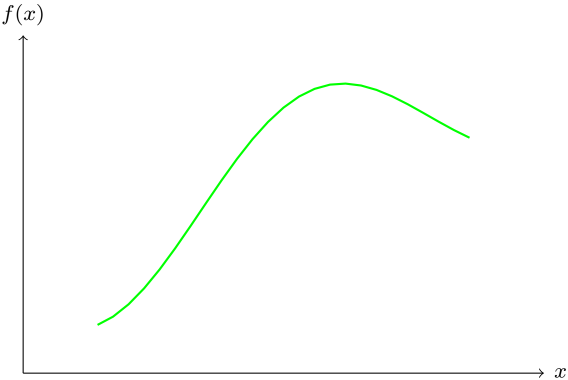
- and, under the same conditions, it converges to a root of

$$\nabla E(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla E_i(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla L(f(\mathbf{x}_i; \boldsymbol{\theta}), t_i)$$

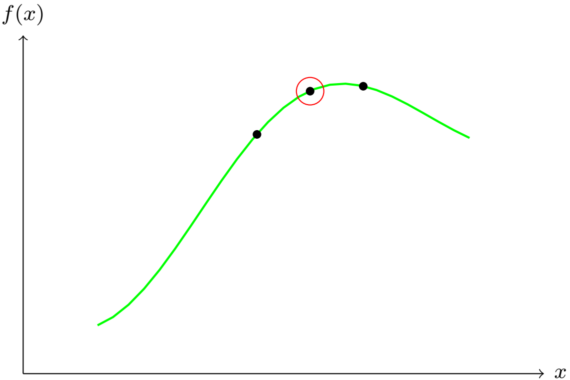
  that is, to a local minimum of $E$
- mini-batch gradient descent is similar but with less “stochastic noise”

# gradient computation

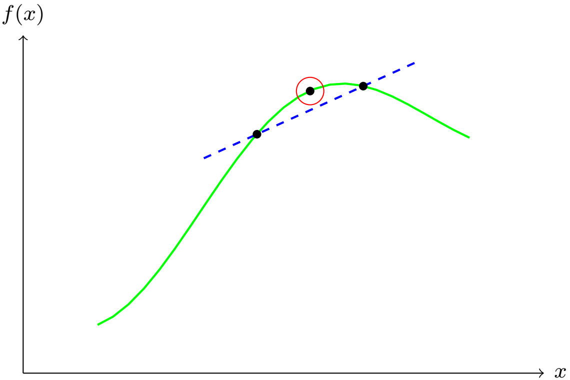
# numerical approximation



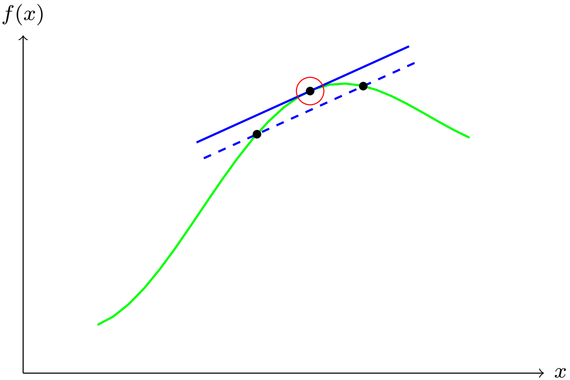
# numerical approximation



# numerical approximation



# numerical approximation




$$\frac{df}{dx}(x) \approx \frac{f(x+\delta) - f(x-\delta)}{2\delta}$$

# numerical approximation

- given $f : \mathbb{R}^p \to \mathbb{R}$, its gradient is the vector function

$$\nabla f := \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_p} \right)$$

- each partial derivative $\frac{\partial f}{\partial x_i}$ can be approximated at $\mathbf{x}$ by the symmetric difference formula

$$\Delta_i f(\mathbf{x}; \delta) := \frac{f(\mathbf{x} + \delta \mathbf{e}_i) - f(\mathbf{x} - \delta \mathbf{e}_i)}{2\delta}$$

  for small $\delta > 0$, where $\mathbf{e}_i$ is the $i$ standard basis vector of $\mathbb{R}^m$

- in practice, the smallest $\delta$ should be used that does not cause numerical issues, *e.g.* $\delta \in [10^{-10}, 10^{-5}]$ for double-precision arithmetic

# numerical approximation

- given $f : \mathbb{R}^p \to \mathbb{R}$, its gradient is the vector function

$$\nabla f := \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_p} \right)$$

- each partial derivative $\frac{\partial f}{\partial x_i}$ can be approximated at $\mathbf{x}$ by the symmetric difference formula

$$\Delta_i f(\mathbf{x}; \delta) := \frac{f(\mathbf{x} + \delta \mathbf{e}_i) - f(\mathbf{x} - \delta \mathbf{e}_i)}{2\delta}$$

  for small $\delta > 0$, where $\mathbf{e}_i$ is the $i$ standard basis vector of $\mathbb{R}^m$

- in practice, the smallest $\delta$ should be used that does not cause numerical issues, *e.g.* $\delta \in [10^{-10}, 10^{-5}]$ for double-precision arithmetic

# numerical approximation

- given $f : \mathbb{R}^p \to \mathbb{R}$, its gradient is the vector function

$$\nabla f := \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_p} \right)$$

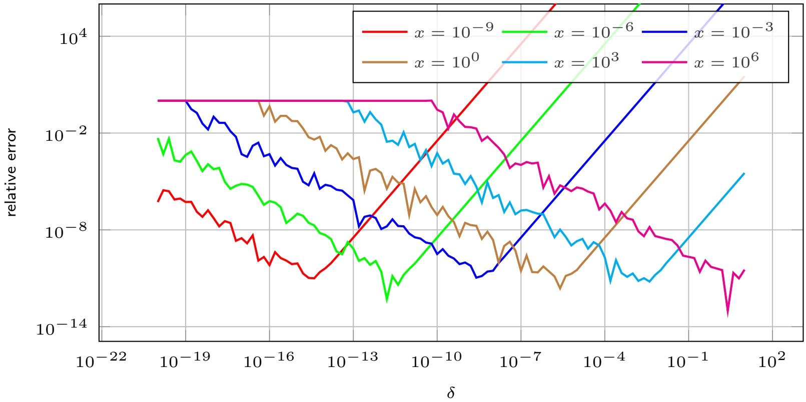
- each partial derivative $\frac{\partial f}{\partial x_i}$ can be approximated at $\mathbf{x}$ by the symmetric difference formula

$$\Delta_i f(\mathbf{x}; \delta) := \frac{f(\mathbf{x} + \delta \mathbf{e}_i) - f(\mathbf{x} - \delta \mathbf{e}_i)}{2\delta}$$

  for small $\delta > 0$, where $\mathbf{e}_i$ is the $i$ standard basis vector of $\mathbb{R}^m$

- in practice, the smallest $\delta$ should be used that does not cause numerical issues, *e.g.* $\delta \in [10^{-10}, 10^{-5}]$ for double-precision arithmetic

# example




- relative error for $f(x) = x^3$, $\nabla f(x) = 3x^2$

$$\frac{|\Delta f(x;\delta) - \nabla f(x)|}{\nabla f(x)}$$

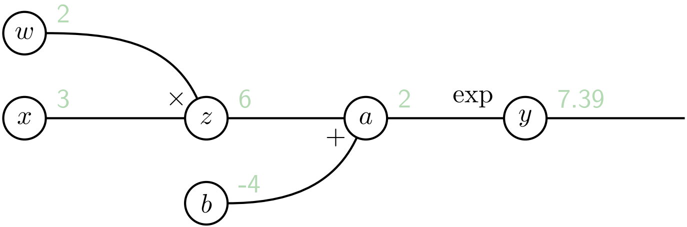
# numerical vs. analytical

- apart from accuracy issues, the numerical approximation is impractical in high dimensions: one evaluation of $\Delta f$ requires $2p$ evaluations of $f$, and dimension $p$ is easily in the order of millions
- we turn to analytical computation of the gradient, which costs roughly as much as one evaluation of $f$
- but the numerical approximation always remains useful for double-checking

# analytical computation

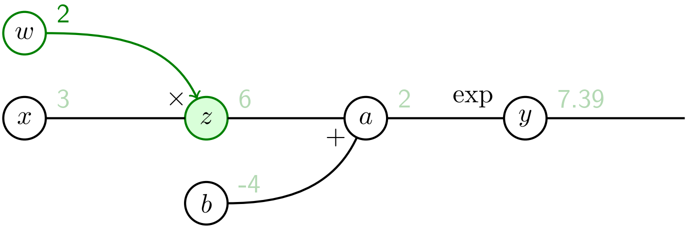
- all derivatives we care about are the derivatives of the error function with respect to the model parameters: the error function is scalar and we need its gradient
- we are going to write the error function as a composition of simpler functions, and use the chain rule to compute the gradient efficiently
- the error function can be as complex as a program with control flow statements
- each component function, called a unit, is assumed to be at least piecewise differentiable with a known formula for its derivative
- a unit may be a vector function, so we need Jacobian matrices in general, not just gradients

# example



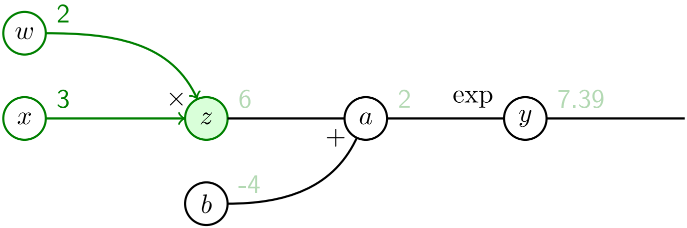

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



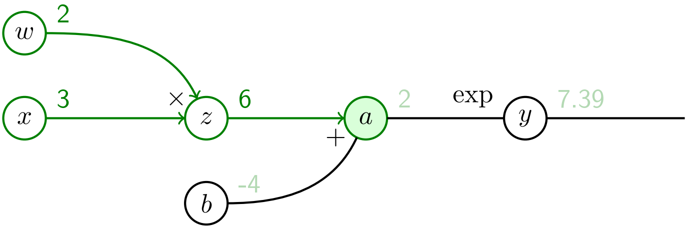

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z}w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z}x = 22.17$

# example



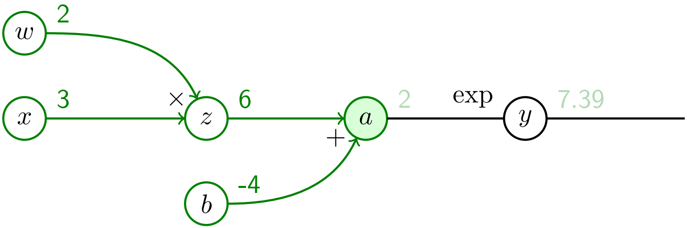

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



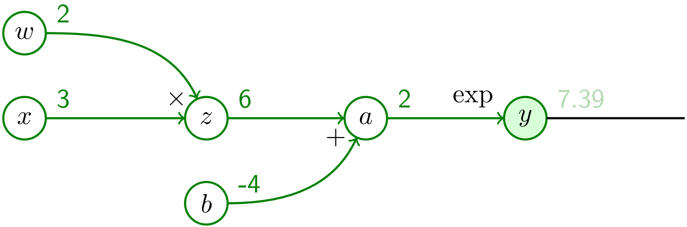

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



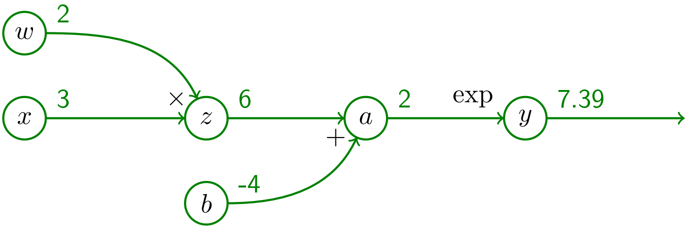

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



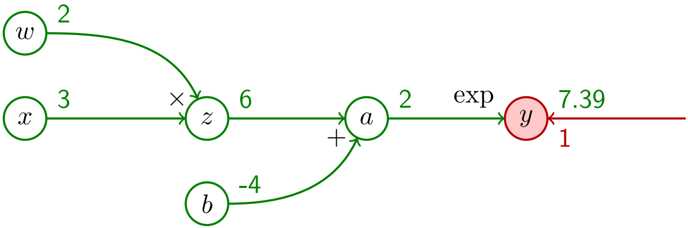

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



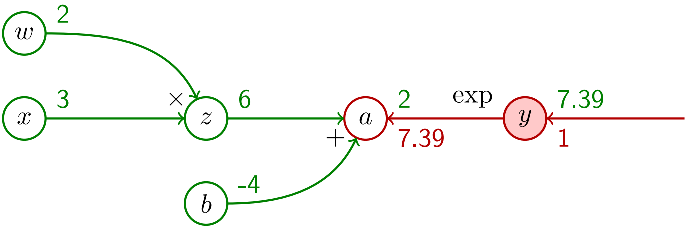

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



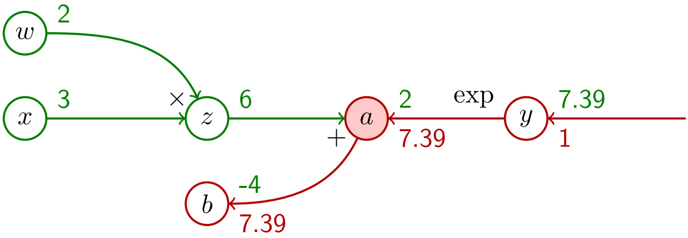

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



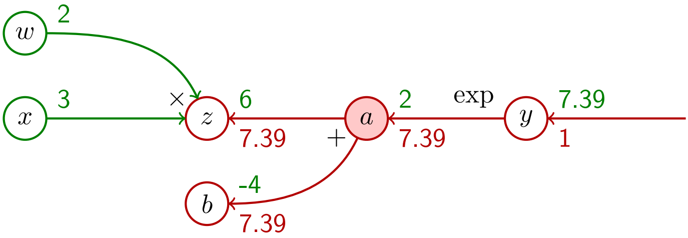

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



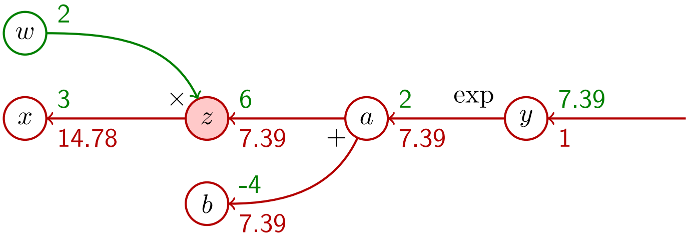

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example



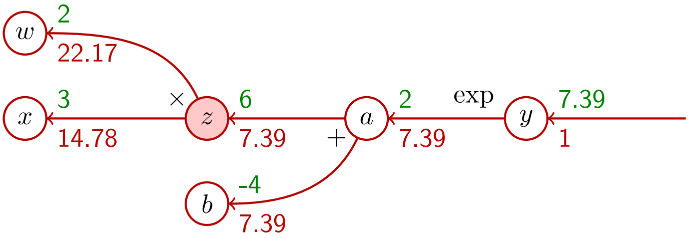

- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$, $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example




- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# example




- $y = e^{wx+b}$ is broken down into $y = e^a$, $a = z + b$, $z = wx$
- we seek $\frac{\partial y}{\partial w}$, $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial b}$ for $(w, x, b) = (2, 3, -4)$
- $\frac{\partial y}{\partial y} = 1$, $\frac{\partial y}{\partial a} = e^a = 7.39$, $\frac{\partial y}{\partial b} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial b} = \frac{\partial y}{\partial a} = 7.39$,
  $\frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial y}{\partial a} = 7.39$, $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial y}{\partial z} w = 14.78$,
  $\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial y}{\partial z} x = 22.17$

# vector functions: derivative

- a function $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ if there is a $q \times p$ matrix $A$ such that

$$\frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - A \cdot \mathbf{h}}{|\mathbf{h}|} \to \mathbf{0}$$

  as $\mathbf{h} \to \mathbf{0}$; matrix $A$ is the derivative of $f$ at $\mathbf{x}$, denoted as $Df(\mathbf{x})$

- if

$$f(\mathbf{x}) = A\mathbf{x}$$

  then

$$Df(\mathbf{x}) = A$$

# vector functions: derivative

- a function $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ if there is a $q \times p$ matrix $A$ such that
$$\frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - A \cdot \mathbf{h}}{|\mathbf{h}|} \to \mathbf{0}$$
as $\mathbf{h} \to \mathbf{0}$; matrix $A$ is the derivative of $f$ at $\mathbf{x}$, denoted as $Df(\mathbf{x})$
- if
$$f(\mathbf{x}) = A\mathbf{x}$$
then
$$Df(\mathbf{x}) = A$$

# vector functions: derivative vs. Jacobian

- given $f = (f_1, \dots, f_q) : \mathbb{R}^p \to \mathbb{R}^q$ whose its partial derivatives exist at $\mathbf{x}$, and $\mathbf{y} = f(\mathbf{x})$, its Jacobian matrix at $\mathbf{x}$ can be written as

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} := \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_q}{\partial x_1} & \cdots & \frac{\partial f_q}{\partial x_p} \end{pmatrix}$$

- if $f$ is differentiable at $\mathbf{x}$, its derivative at $\mathbf{x}$ is

$$Df(\mathbf{x}) := \begin{pmatrix} D_1 f_1 & \dots & D_p f_1 \\ \vdots & \ddots & \vdots \\ D_1 f_q & \dots & D_p f_q \end{pmatrix} (\mathbf{x})$$

- if $f$ is differentiable at $\mathbf{x}$, the derivative $Df(\mathbf{x})$ equals the Jacobian at $\mathbf{x}$; but the Jacobian may exist without any derivative defined

# vector functions: derivative vs. Jacobian

- given $f = (f_1, \dots, f_q) : \mathbb{R}^p \to \mathbb{R}^q$ whose its partial derivatives exist at $\mathbf{x}$, and $\mathbf{y} = f(\mathbf{x})$, its Jacobian matrix at $\mathbf{x}$ can be written as

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} := \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_q}{\partial x_1} & \cdots & \frac{\partial f_q}{\partial x_p} \end{pmatrix}$$

- if $f$ is differentiable at $\mathbf{x}$, its derivative at $\mathbf{x}$ is

$$Df(\mathbf{x}) := \begin{pmatrix} D_1 f_1 & \dots & D_p f_1 \\ \vdots & \ddots & \vdots \\ D_1 f_q & \dots & D_p f_q \end{pmatrix} (\mathbf{x})$$

- if $f$ is differentiable at $\mathbf{x}$, the derivative $Df(\mathbf{x})$ equals the Jacobian at $\mathbf{x}$; but the Jacobian may exist without any derivative defined

# scalar functions: derivative vs. gradient

- the gradient of a scalar $f : \mathbb{R}^p \to \mathbb{R}$ with respect to an input vector $\mathbf{x}$ is a column vector in $\mathbb{R}^p$, the same size as $\mathbf{x}$

$$\nabla f := \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_p} \right)$$

- in contrast, the derivative is an $1 \times p$ row vector

$$Df(\mathbf{x}) := \begin{pmatrix} \frac{\partial f}{\partial x_1} & \ldots & \frac{\partial f}{\partial x_p} \end{pmatrix} = (\nabla f)^\top$$

- the following analysis uses derivatives/Jacobians, so we will transpose them to make them compatible with $\mathbf{x}$

# chain rule

- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

# chain rule

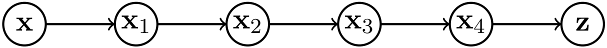
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$



# chain rule

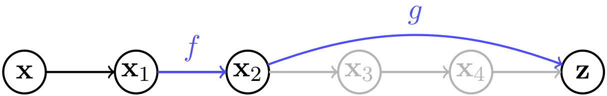
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$



# chain rule

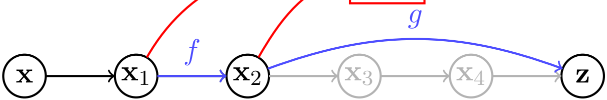
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$



# chain rule

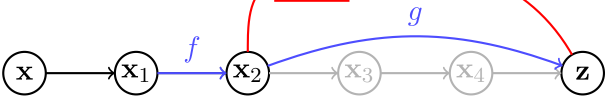
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$



# chain rule

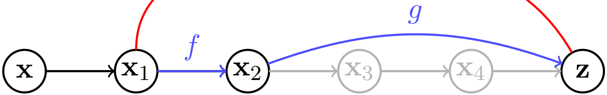
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$



# chain rule

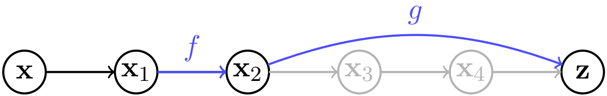
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$




- now, for all $i$, let us call the partial derivatives

$$d\mathbf{x}_i^\top := \frac{\partial \mathbf{z}}{\partial \mathbf{x}_i}$$

# chain rule

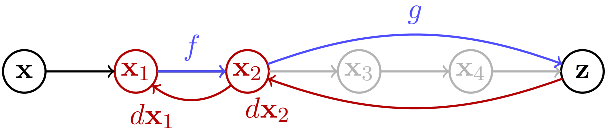
- if $f : \mathbb{R}^p \to \mathbb{R}^q$ is differentiable at $\mathbf{x}$ and $g : \mathbb{R}^q \to \mathbb{R}^r$ is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f : \mathbb{R}^p \to \mathbb{R}^r$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

  where $\cdot$ denotes matrix multiplication

- how to use it:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

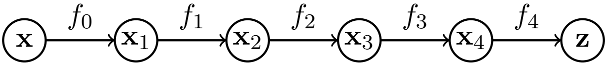



- then, we are back-propagating from $d\mathbf{x}_2$ to $d\mathbf{x}_1$

$$d\mathbf{x}_1^\top = d\mathbf{x}_2^\top \cdot Df(\mathbf{x}_1)$$

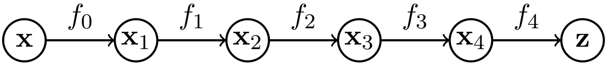
# chaining

- let $f = f_4 \circ f_3 \circ f_2 \circ f_1 \circ f_0$ and $\mathbf{z} = f(\mathbf{x})$



# chaining

- let $f = f_4 \circ f_3 \circ f_2 \circ f_1 \circ f_0$ and $\mathbf{z} = f(\mathbf{x})$

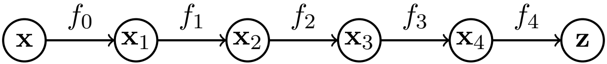



- we apply the chain rule

$$
\begin{aligned}
\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = Df(\mathbf{x}) &= D(f_4 \circ f_3 \circ f_2 \circ f_1)(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= D(f_4 \circ f_3 \circ f_2)(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= D(f_4 \circ f_3)(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= Df_4(\mathbf{x}_4) \cdot Df_3(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_4^\top \cdot Df_3(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_3^\top \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_2^\top \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_1^\top \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}^\top
\end{aligned}
$$

# chaining

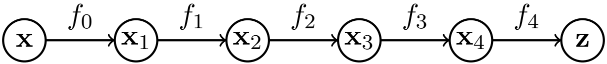
- let $f = f_4 \circ f_3 \circ f_2 \circ f_1 \circ f_0$ and $\mathbf{z} = f(\mathbf{x})$




- we apply the chain rule, then collect back into factors $d\mathbf{x}_i$

$$
\begin{aligned}
\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = Df(\mathbf{x}) &= D(f_4 \circ f_3 \circ f_2 \circ f_1)(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= D(f_4 \circ f_3 \circ f_2)(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= D(f_4 \circ f_3)(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= Df_4(\mathbf{x}_4) \cdot Df_3(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_4^\top \cdot Df_3(\mathbf{x}_3) \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_3^\top \cdot Df_2(\mathbf{x}_2) \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_2^\top \cdot Df_1(\mathbf{x}_1) \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}_1^\top \cdot Df_0(\mathbf{x}) \\
&= d\mathbf{x}^\top
\end{aligned}
$$

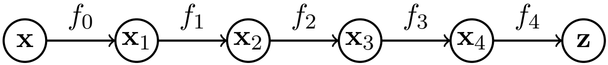
# back-propagation




## forward pass

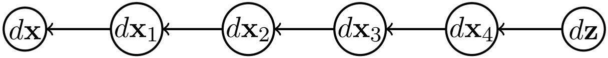
$$\mathbf{x}_1 = f_0(\mathbf{x}) \quad \mathbf{x}_2 = f_1(\mathbf{x}_1) \quad \mathbf{x}_3 = f_2(\mathbf{x}_2) \quad \mathbf{x}_4 = f_3(\mathbf{x}_3) \quad \mathbf{z} = f_4(\mathbf{x}_4)$$

# back-propagation




## forward pass

$$\mathbf{x}_1 = f_0(\mathbf{x}) \quad \mathbf{x}_2 = f_1(\mathbf{x}_1) \quad \mathbf{x}_3 = f_2(\mathbf{x}_2) \quad \mathbf{x}_4 = f_3(\mathbf{x}_3) \quad \mathbf{z} = f_4(\mathbf{x}_4)$$




## backward pass

$$d\mathbf{z}^\top = I \qquad d\mathbf{x}_4^\top = d\mathbf{z}^\top \cdot Df_4(\mathbf{x}_4) \qquad d\mathbf{x}_3^\top = d\mathbf{x}_4^\top \cdot Df_3(\mathbf{x}_3)$$

$$d\mathbf{x}_2^\top = d\mathbf{x}_3^\top \cdot Df_3(\mathbf{x}_3) \qquad d\mathbf{x}_1^\top = d\mathbf{x}_2^\top \cdot Df_1(\mathbf{x}_1) \qquad d\mathbf{x}^\top = d\mathbf{x}_1^\top \cdot Df_0(\mathbf{x})$$

# back-propagation is dynamic programming

- we need to store all the $\mathbf{x}_i$ that we compute in the forward pass before the backward pass begins
- the $d\mathbf{x}_i$ can be computed on the fly in reverse order on a chain, but may need to be all stored on a general network structure
- that's exactly what we do in dynamic programming: break the problem down into a collection of smaller, overlapping subproblems, store their solutions and save computation time at the expense of a (hopefully) modest expenditure in storage space
- as in all dynamic programming problems, there is a bottom-up approach that we have just described, and a top-down approach coming out of the recursive formulation through memoization; this can be useful if we are looking for the derivative with respect to only few parameters

# back-propagation is dynamic programming

- we need to store all the $\mathbf{x}_i$ that we compute in the forward pass before the backward pass begins
- the $d\mathbf{x}_i$ can be computed on the fly in reverse order on a chain, but may need to be all stored on a general network structure
- that's exactly what we do in dynamic programming: break the problem down into a collection of smaller, overlapping subproblems, store their solutions and save computation time at the expense of a (hopefully) modest expenditure in storage space
- as in all dynamic programming problems, there is a bottom-up approach that we have just described, and a top-down approach coming out of the recursive formulation through memoization; this can be useful if we are looking for the derivative with respect to only few parameters

# partial derivatives

- in the following, for any vector $\mathbf{x}$ appearing in our function, we will use the symbol

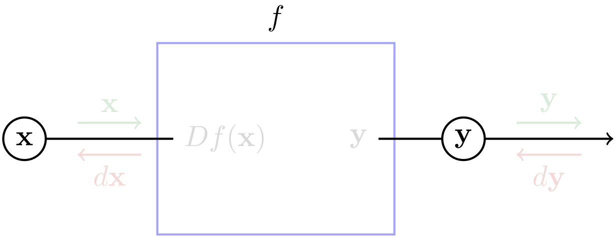
$$d\mathbf{x}^\top := \frac{\partial}{\partial \mathbf{x}}$$

  for the partial derivative operator of any quantity with respect to $\mathbf{x}$
- in practice, we will apply this to the quantity we want to optimize, *i.e.* the error
- the error gradient will consist of the partial derivatives with respect to the model parameters, but we still need to compute partial derivatives with respect to all variables appearing in back-propagation

# partial derivatives

- in the following, for any vector $\mathbf{x}$ appearing in our function, we will use the symbol
$$d\mathbf{x}^\top := \frac{\partial}{\partial \mathbf{x}}$$
for the partial derivative operator of any quantity with respect to $\mathbf{x}$
- in practice, we will apply this to the quantity we want to optimize, *i.e.* the error
- the error gradient will consist of the partial derivatives with respect to the model parameters, but we still need to compute partial derivatives with respect to all variables appearing in back-propagation

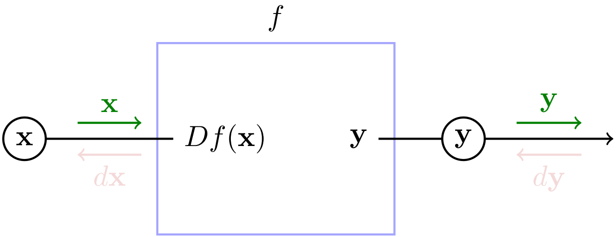
# nodes




- to every variable $\mathbf{y}$ is associated a node with the function $f$ that produces it, from input variable $\mathbf{x}$
- given $\mathbf{x}$, derivative $Df(\mathbf{x})$ is "stored", and output $\mathbf{y}$ is computed and flows forward
- given $d\mathbf{y}$, partial derivative $d\mathbf{x}$ is computed and flows backward

$$d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

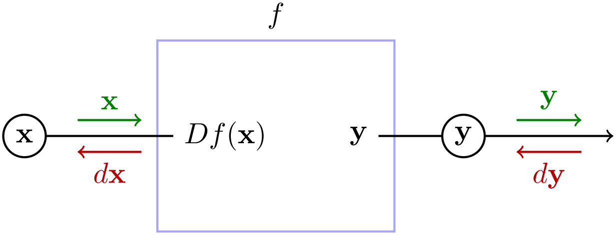
# nodes




- to every variable $\mathbf{y}$ is associated a node with the function $f$ that produces it, from input variable $\mathbf{x}$
- given $\mathbf{x}$, derivative $Df(\mathbf{x})$ is "stored", and output $\mathbf{y}$ is computed and flows forward
- given $d\mathbf{y}$, partial derivative $d\mathbf{x}$ is computed and flows backward

$$d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

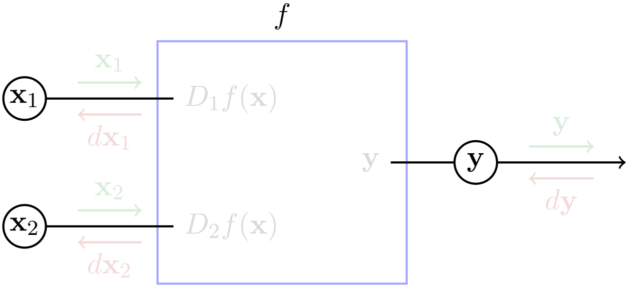
# nodes




- to every variable $\mathbf{y}$ is associated a node with the function $f$ that produces it, from input variable $\mathbf{x}$
- given $\mathbf{x}$, derivative $Df(\mathbf{x})$ is "stored", and output $\mathbf{y}$ is computed and flows forward
- given $d\mathbf{y}$, partial derivative $d\mathbf{x}$ is computed and flows backward

$$d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

# splitting the input



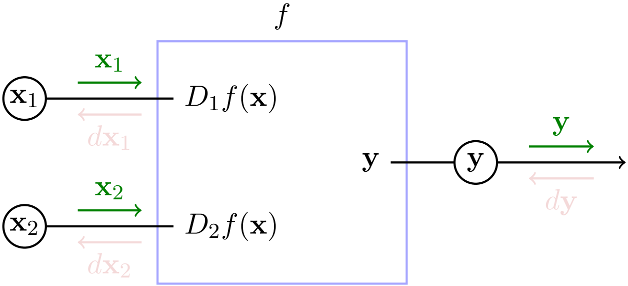

- we split input vector $\mathbf{x}$ into subvectors as $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$
- then, the derivative consists of blocks stacked horizontally

$$Df(\mathbf{x}) = (D_1 f \;\; D_2 f)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{y}}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}}{\partial \mathbf{x}_2} \end{pmatrix}$$

- $d\mathbf{x}$ is also split as $d\mathbf{x} = (d\mathbf{x}_1, d\mathbf{x}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

$$d\mathbf{x}_i^\top = d\mathbf{y}^\top \cdot D_i f(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}_i}$$

# splitting the input



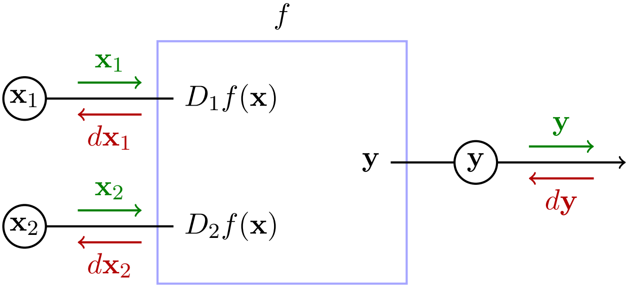

- we split input vector $\mathbf{x}$ into subvectors as $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$
- then, the derivative consists of blocks stacked horizontally

$$Df(\mathbf{x}) = (D_1 f \;\; D_2 f)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{y}}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}}{\partial \mathbf{x}_2} \end{pmatrix}$$

- $d\mathbf{x}$ is also split as $d\mathbf{x} = (d\mathbf{x}_1, d\mathbf{x}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

$$d\mathbf{x}_i^\top = d\mathbf{y}^\top \cdot D_i f(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}_i}$$

# splitting the input



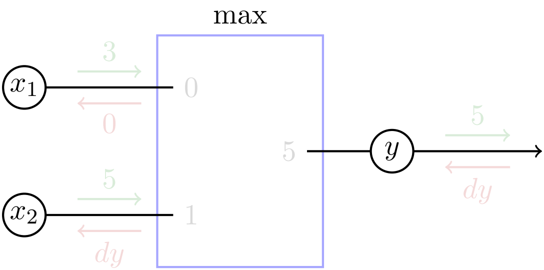

- we split input vector $\mathbf{x}$ into subvectors as $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$
- then, the derivative consists of blocks stacked horizontally

$$Df(\mathbf{x}) = (D_1 f \;\; D_2 f)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}_1} \;\; \frac{\partial \mathbf{y}}{\partial \mathbf{x}_2} \right)$$

- $d\mathbf{x}$ is also split as $d\mathbf{x} = (d\mathbf{x}_1, d\mathbf{x}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

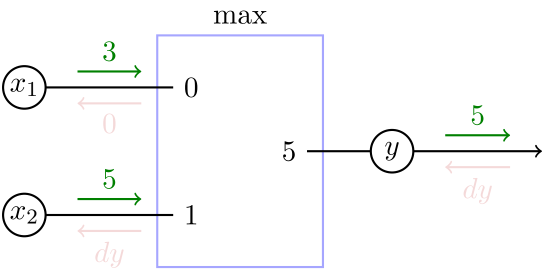
$$d\mathbf{x}_i^\top = d\mathbf{y}^\top \cdot D_i f(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}_i}$$

# example: maximum



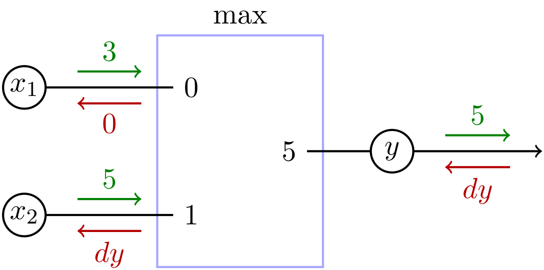

- if $f(x_1, x_2) = \max(x_1, x_2)$, then $D_i f(x_1, x_2) = \mathbb{1}[x_i = \max(x_1, x_2)]$
- and $dy$ is routed into the branch of the maximum input

# example: maximum



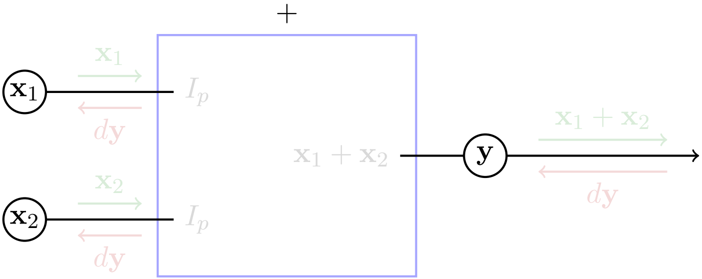

- if $f(x_1, x_2) = \max(x_1, x_2)$, then $D_i f(x_1, x_2) = \mathbb{1}[x_i = \max(x_1, x_2)]$
- and $dy$ is routed into the branch of the maximum input

# example: maximum



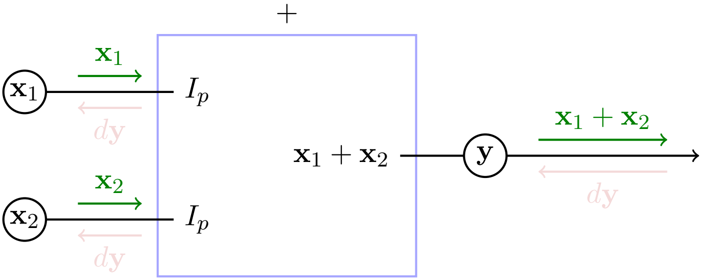

- if $f(x_1, x_2) = \max(x_1, x_2)$, then $D_i f(x_1, x_2) = \mathbb{1}[x_i = \max(x_1, x_2)]$
- and $dy$ is routed into the branch of the maximum input

# example: sum



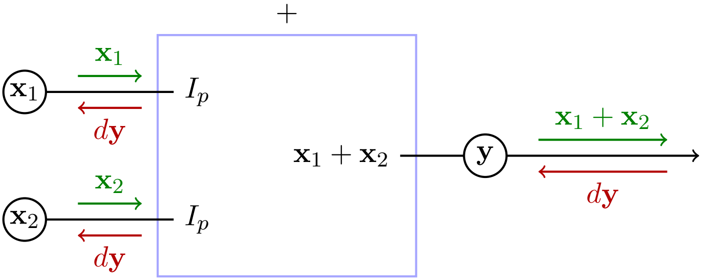

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 + \mathbf{x}_2$ and $\mathbf{x}_i \in \mathbb{R}^p$, then $D_i f(\mathbf{x}_1, \mathbf{x}_2) = I_p$
- and $d\mathbf{y}$ is distributed to both branches

# example: sum



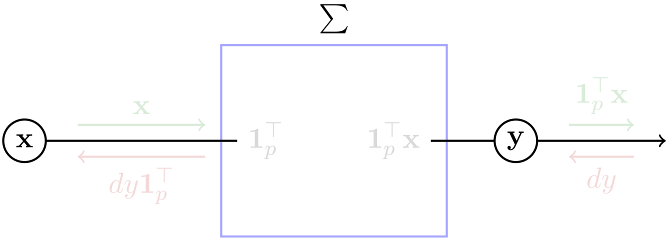

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 + \mathbf{x}_2$ and $\mathbf{x}_i \in \mathbb{R}^p$, then $D_i f(\mathbf{x}_1, \mathbf{x}_2) = I_p$
- and $d\mathbf{y}$ is distributed to both branches

# example: sum



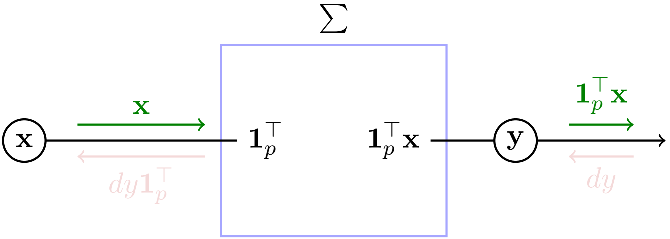

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 + \mathbf{x}_2$ and $\mathbf{x}_i \in \mathbb{R}^p$, then $D_i f(\mathbf{x}_1, \mathbf{x}_2) = I_p$
- and $d\mathbf{y}$ is distributed to both branches

# example: vector sum*



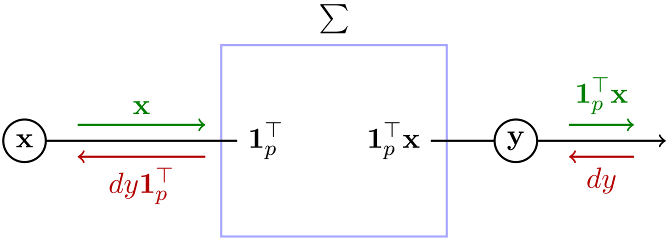

- if $f(\mathbf{x}) = \mathbf{1}_p^\top \mathbf{x} = \sum_{i=1}^{p} x_i$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = \mathbf{1}_p^\top$
- and $dy$ is distributed to every element

# example: vector sum*



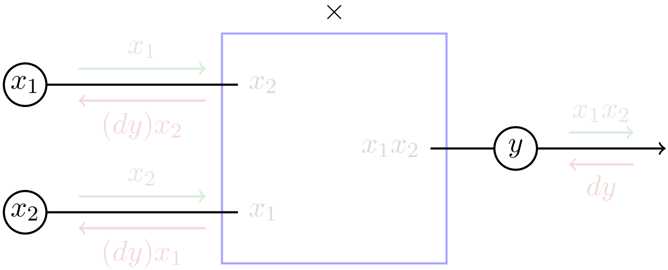

- if $f(\mathbf{x}) = \mathbf{1}_p^\top \mathbf{x} = \sum_{i=1}^p x_i$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = \mathbf{1}_p^\top$
- and $dy$ is distributed to every element

# example: vector sum*



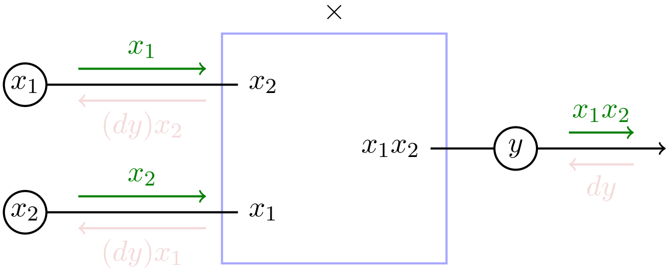

- if $f(\mathbf{x}) = \mathbf{1}_p^\top \mathbf{x} = \sum_{i=1}^p x_i$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = \mathbf{1}_p^\top$
- and $dy$ is distributed to every element

# example: product*



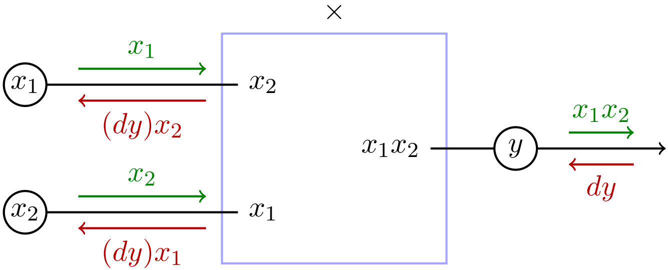

- if $f(x_1, x_2) = x_1x_2$, then $D_1f(x_1, x_2) = x_2$ and $D_2f(x_1, x_2) = x_1$
- the derivative on each branch is multiplied by the input of the other

# example: product*



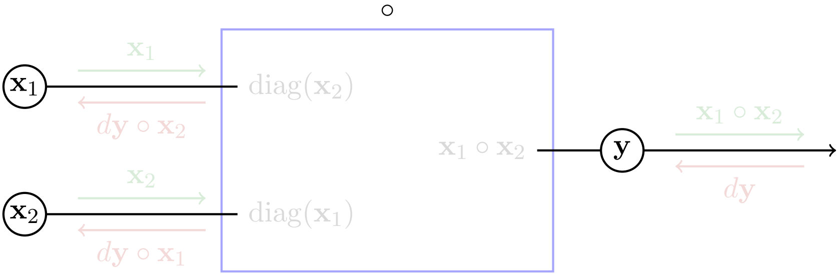

- if $f(x_1, x_2) = x_1x_2$, then $D_1 f(x_1, x_2) = x_2$ and $D_2 f(x_1, x_2) = x_1$
- the derivative on each branch is multiplied by the input of the other

# example: product*



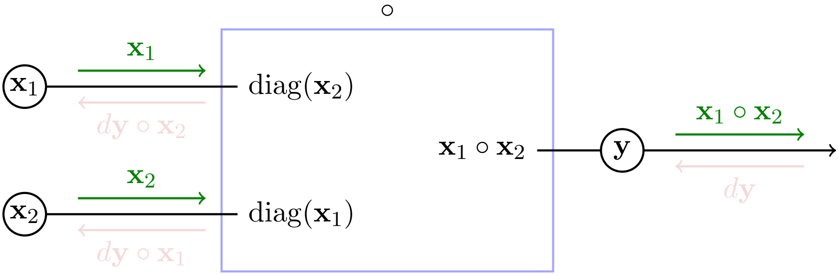

- if $f(x_1, x_2) = x_1x_2$, then $D_1f(x_1, x_2) = x_2$ and $D_2f(x_1, x_2) = x_1$
- the derivative on each branch is multiplied by the input of the other

# example: Hadamard (element-wise) product*



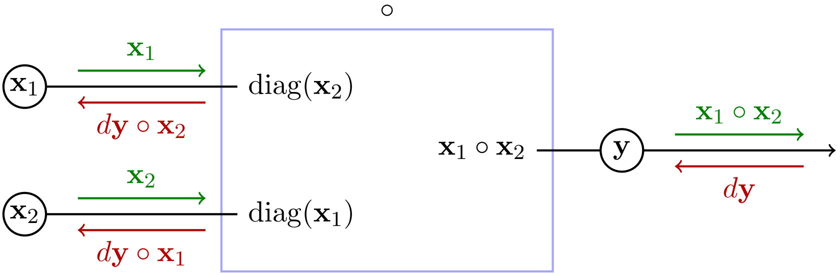

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \circ \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_2)$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_1)$
- the derivative on each branch is element-wise multiplied by the input of the other

# example: Hadamard (element-wise) product*



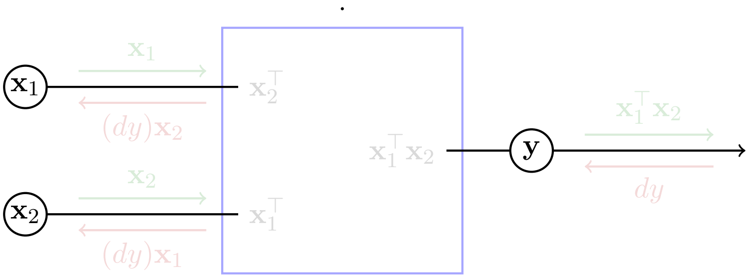

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \circ \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_2)$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_1)$
- the derivative on each branch is element-wise multiplied by the input of the other

# example: Hadamard (element-wise) product*



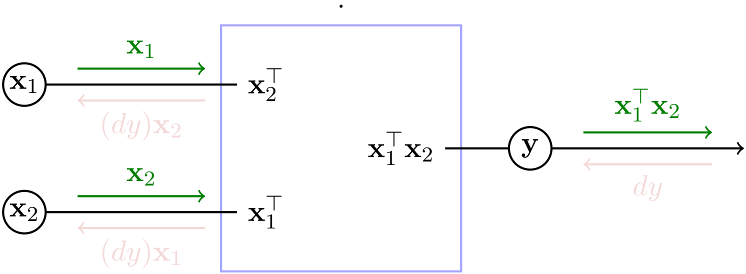

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \circ \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_2)$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \text{diag}(\mathbf{x}_1)$
- the derivative on each branch is element-wise multiplied by the input of the other

# example: dot product*



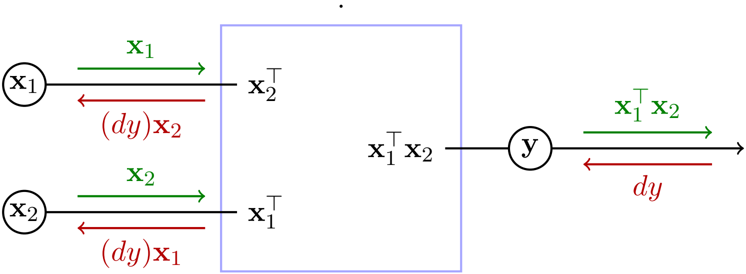

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1^\top \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1$
- the derivative on each branch is multiplied by the input of the other; this can be seen by composing an element-wise product with a vector sum

# example: dot product*



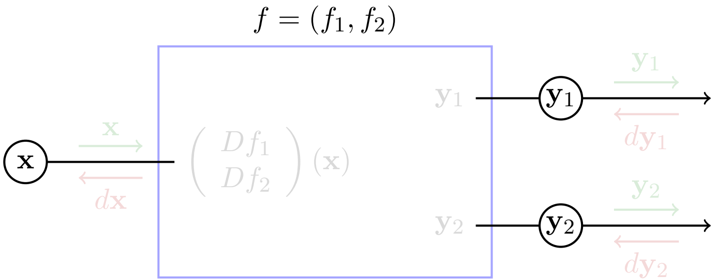

- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1^\top \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1$
- the derivative on each branch is multiplied by the input of the other; this can be seen by composing an element-wise product with a vector sum

# example: dot product*




- if $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1^\top \mathbf{x}_2$, then $D_1 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_2$ and $D_2 f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1$
- the derivative on each branch is multiplied by the input of the other; this can be seen by composing an element-wise product with a vector sum

# splitting the output



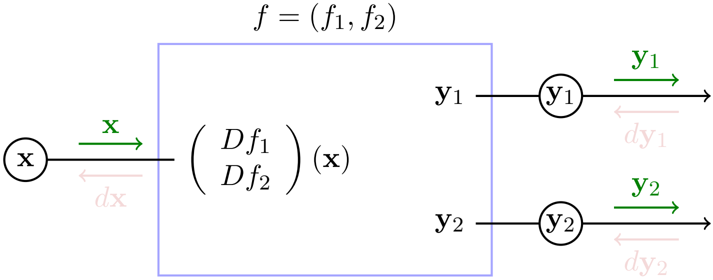

- we split output $\mathbf{y}$ into subvectors as $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2) = (f_1(\mathbf{x}), f_2(\mathbf{x}))$
- then, the derivative consists of blocks stacked vertically

$$Df(\mathbf{x}) = (Df_1; Df_2)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left( \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}}; \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} \right)$$

- $d\mathbf{y}$ is also split as $d\mathbf{y} = (d\mathbf{y}_1, d\mathbf{y}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

$$d\mathbf{x}^\top = \sum_i d\mathbf{y}_i^\top \cdot Df_i(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \sum_i \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}}$$

# splitting the output



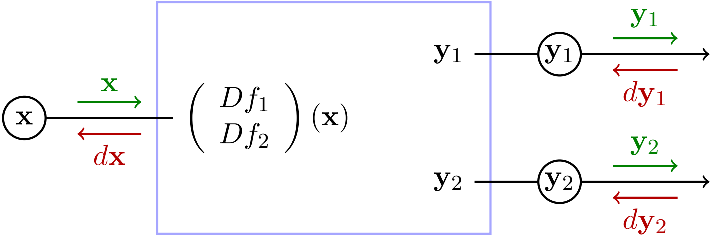

- we split output $\mathbf{y}$ into subvectors as $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2) = (f_1(\mathbf{x}), f_2(\mathbf{x}))$
- then, the derivative consists of blocks stacked vertically

$$Df(\mathbf{x}) = (Df_1; Df_2)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left( \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}}; \ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} \right)$$

- $d\mathbf{y}$ is also split as $d\mathbf{y} = (d\mathbf{y}_1, d\mathbf{y}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

$$d\mathbf{x}^\top = \sum_i d\mathbf{y}_i^\top \cdot Df_i(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \sum_i \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}}$$

# splitting the output



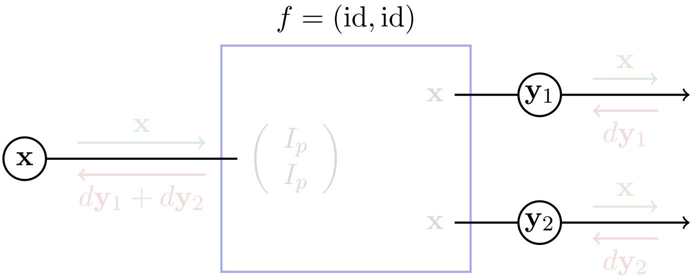

- we split output $\mathbf{y}$ into subvectors as $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2) = (f_1(\mathbf{x}), f_2(\mathbf{x}))$
- then, the derivative consists of blocks stacked vertically

$$Df(\mathbf{x}) = (Df_1; Df_2)(\mathbf{x}) \quad \text{or} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left( \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}}; \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} \right)$$

- $d\mathbf{y}$ is also split as $d\mathbf{y} = (d\mathbf{y}_1, d\mathbf{y}_2)$ and $d\mathbf{x}^\top = d\mathbf{y}^\top \cdot Df(\mathbf{x})$ becomes

$$d\mathbf{x}^\top = \sum_i d\mathbf{y}_i^\top \cdot Df_i(\mathbf{x}) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \sum_i \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}}$$

# example: splitter (sharing)

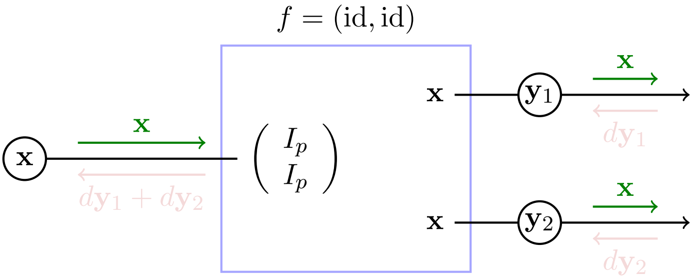



- if $f(\mathbf{x}) = (\mathbf{x}, \mathbf{x})$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = (I_p; I_p)$
- and the node behaves like sum backwards

$$d\mathbf{x} = d\mathbf{y}_1 + d\mathbf{y}_2 \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}_1} + \frac{\partial}{\partial \mathbf{y}_2}$$

- whenever a variable is shared (used more than once), we need to sum the gradients flowing from all paths where it appears

# example: splitter (sharing)

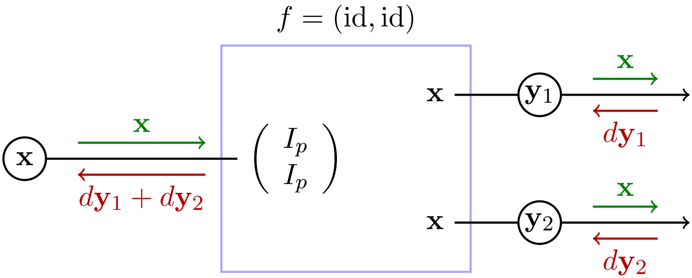



- if $f(\mathbf{x}) = (\mathbf{x}, \mathbf{x})$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = (I_p; I_p)$
- and the node behaves like sum backwards

$$d\mathbf{x} = d\mathbf{y}_1 + d\mathbf{y}_2 \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}_1} + \frac{\partial}{\partial \mathbf{y}_2}$$

- whenever a variable is shared (used more than once), we need to sum the gradients flowing from all paths where it appears

# example: splitter (sharing)

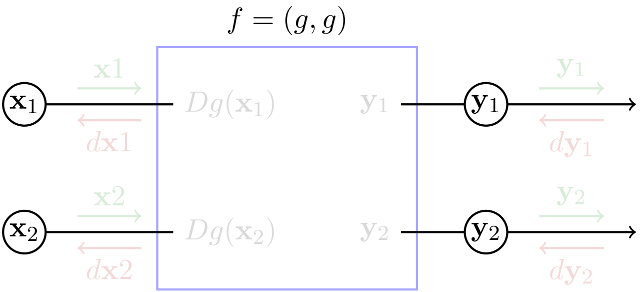



- if $f(\mathbf{x}) = (\mathbf{x}, \mathbf{x})$ and $\mathbf{x} \in \mathbb{R}^p$, then $Df(\mathbf{x}) = (I_p; I_p)$
- and the node behaves like sum backwards

$$d\mathbf{x} = d\mathbf{y}_1 + d\mathbf{y}_2 \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{y}_1} + \frac{\partial}{\partial \mathbf{y}_2}$$

- whenever a variable is shared (used more than once), we need to sum the gradients flowing from all paths where it appears

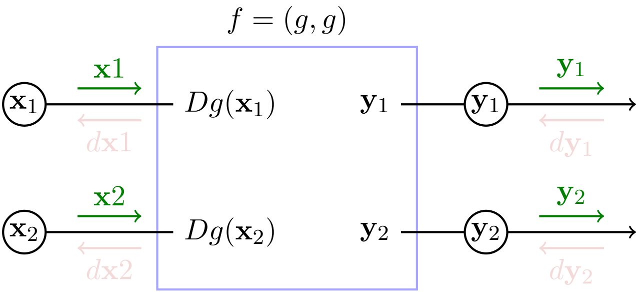
# example: tuples*




- if $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and $f = (g, g)$, then $Df(\mathbf{x})$ is block-wise diagonal: $\mathrm{diag}(Dg(\mathbf{x}_1), Dg(\mathbf{x}_2))$
- and the backward paths flow independently like the forward

$$d\mathbf{x}_i^\top = d\mathbf{y}_i^\top \cdot Dg(\mathbf{x}_i) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i}$$

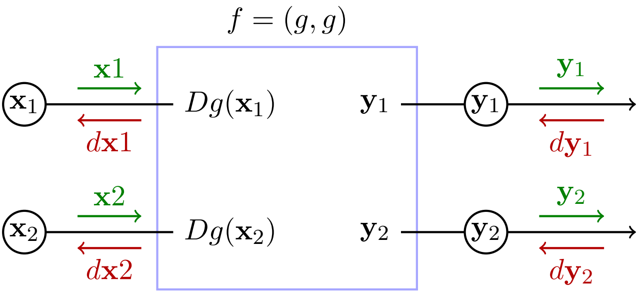
# example: tuples*




- if $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and $f = (g, g)$, then $Df(\mathbf{x})$ is block-wise diagonal: $\operatorname{diag}(Dg(\mathbf{x}_1), Dg(\mathbf{x}_2))$
- and the backward paths flow independently like the forward

$$d\mathbf{x}_i^\top = d\mathbf{y}_i^\top \cdot Dg(\mathbf{x}_i) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i}$$

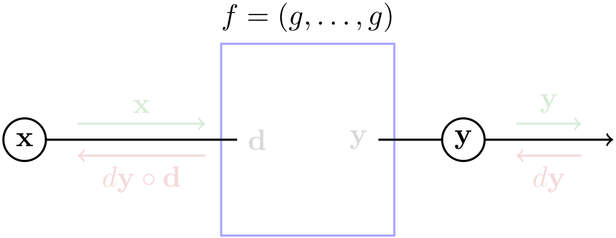
# example: tuples*




- if $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ and $f = (g, g)$, then $Df(\mathbf{x})$ is block-wise diagonal: $\mathrm{diag}(Dg(\mathbf{x}_1), Dg(\mathbf{x}_2))$
- and the backward paths flow independently like the forward

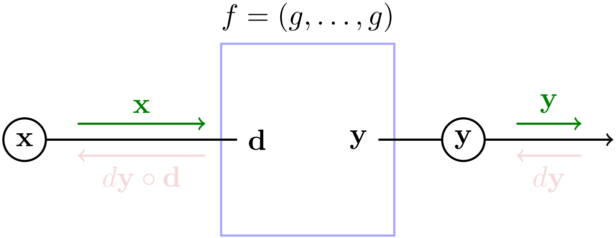
$$d\mathbf{x}_i^\top = d\mathbf{y}_i^\top \cdot Dg(\mathbf{x}_i) \quad \text{or} \quad \frac{\partial}{\partial \mathbf{x}_i} = \frac{\partial}{\partial \mathbf{y}_i} \cdot \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i}$$

# example: element-wise functions



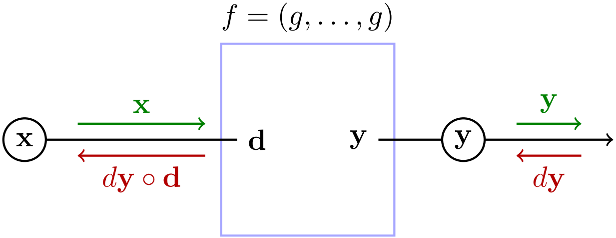

- if $\mathbf{x} \in \mathbb{R}^p$ and $f$ is element-wise with $f(\mathbf{x}) = (g(x_1), \ldots, g(x_p))$ where $g : \mathbb{R} \to \mathbb{R}$, then $Df(\mathbf{x}) = \operatorname{diag} \mathbf{d}$ is diagonal, where $\mathbf{d} = (Dg(x_1), \ldots, Dg(x_p))$
- and the partial derivatives are element-wise multiplied

# example: element-wise functions



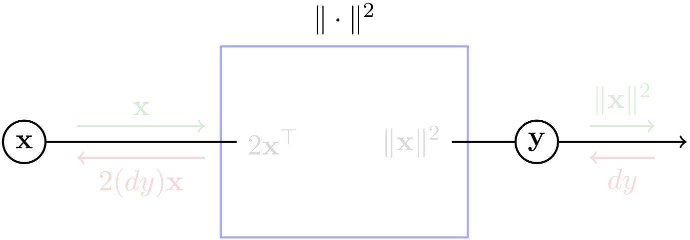

- if $\mathbf{x} \in \mathbb{R}^p$ and $f$ is element-wise with $f(\mathbf{x}) = (g(x_1), \dots, g(x_p))$ where $g : \mathbb{R} \to \mathbb{R}$, then $Df(\mathbf{x}) = \operatorname{diag} \mathbf{d}$ is diagonal, where $\mathbf{d} = (Dg(x_1), \dots, Dg(x_p))$
- and the partial derivatives are element-wise multiplied

# example: element-wise functions



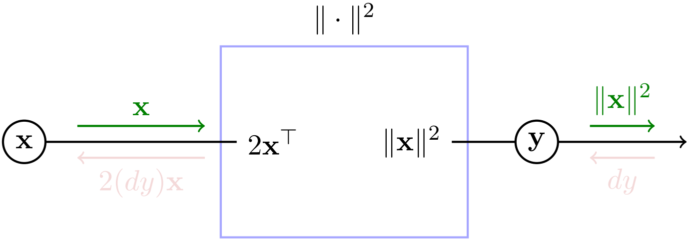

- if $\mathbf{x} \in \mathbb{R}^p$ and $f$ is element-wise with $f(\mathbf{x}) = (g(x_1), \dots, g(x_p))$ where $g : \mathbb{R} \to \mathbb{R}$, then $Df(\mathbf{x}) = \operatorname{diag} \mathbf{d}$ is diagonal, where $\mathbf{d} = (Dg(x_1), \dots, Dg(x_p))$
- and the partial derivatives are element-wise multiplied

# example: squared norm*



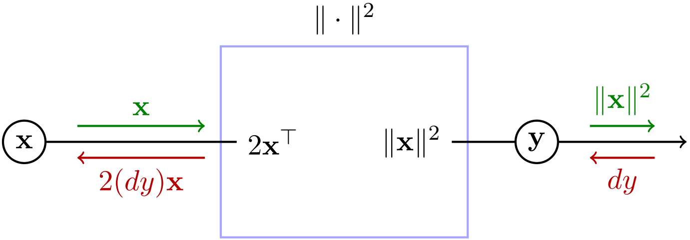

- if $f(\mathbf{x}) = \|\mathbf{x}\|^2$ then $Df(\mathbf{x}) = 2\mathbf{x}^\top$
- and $dy$ is multiplied by $2\mathbf{x}^\top$; this can be seen by composing a splitter (factor 2) with a dot product (factor $\mathbf{x}^\top$)

# example: squared norm*




- if $f(\mathbf{x}) = \|\mathbf{x}\|^2$ then $Df(\mathbf{x}) = 2\mathbf{x}^\top$
- and $dy$ is multiplied by $2\mathbf{x}^\top$; this can be seen by composing a splitter (factor 2) with a dot product (factor $\mathbf{x}^\top$)

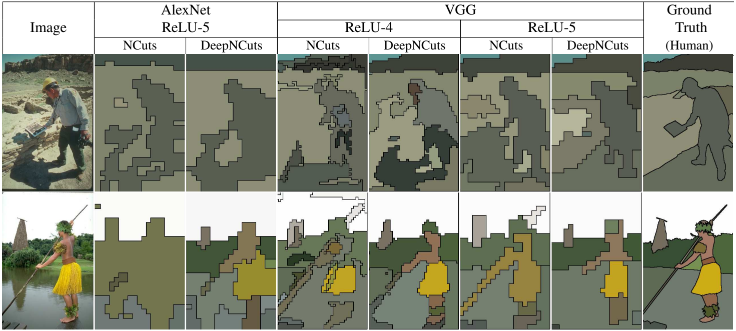
# example: squared norm*




- if $f(\mathbf{x}) = \|\mathbf{x}\|^2$ then $Df(\mathbf{x}) = 2\mathbf{x}^\top$
- and $dy$ is multiplied by $2\mathbf{x}^\top$; this can be seen by composing a splitter (factor 2) with a dot product (factor $\mathbf{x}^\top$)

# matrix derivatives*

[Ionescu et al. 2015]




- derivatives for
  - SVD decomposition $A = U\Sigma V^\top$
  - eigenvalue decomposition $A = U\Sigma U^\top$
  - nonlinear matrix functions $f(A) = Uf(\Sigma)U^\top$
- application to spectral methods for image segmentation

Ionescu, Vantzos, Sminchisescu. ICCV 2015. Matrix Backpropagation for Deep Networks with Structured Layers.

# matrix calculus*

- results like these, and many more

$$\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A$$

$$\frac{\partial \mathbf{x}^\top A\mathbf{x}}{\partial \mathbf{x}} = \mathbf{x}^\top (A + A^\top)$$

$$\frac{\partial \mathrm{vec}(\mathbf{x}^\top A\mathbf{x})}{\partial \mathrm{vec}\, A} = \mathbf{x}^\top \otimes \mathbf{x}^\top$$

$$\frac{\partial AXB}{\partial X} = B^\top \otimes A$$

$$\frac{dA^{-1}}{dA} = -(A^{-\top} \otimes A^{-1})$$

$$\frac{d \ln |A|}{dA} = \mathrm{vec}(A^{-\top})^\top$$

$$\frac{\partial \mathrm{tr}(AX)}{\partial X} = \mathrm{vec}(A^\top)^\top$$

Fackler 2005. Notes on Matrix Calculus. `http://www4.ncsu.edu/~pfackler/MatCalc.pdf`

# in general

- apparently, we do not need to store the Jacobian matrix $Df(\mathbf{x})$, which may be huge, but only what is needed to compute the partial derivatives in the backward pass
- our function can be decomposed into a directed acyclic graph (DAG) of nodes, called a computational graph
- each time we call the function in the forward pass, a new graph may be constructed if our program contains control flow statements like conditionals and loops; methods supporting this operation are called dynamic

# automatic differentiation

[Wengert 1964]

- is the more general set of methods used to automatically evaluate the derivative of a given function at a given input; it is not numerical and not symbolic
- what we call back-propagation here is known as the reverse accumulation mode in this context and makes sense because we compute the gradient of a single scalar quantity with respect to maybe millions of parameters
- forward accumulation makes sense when we need the derivative of many variables with respect to few parameters
- we will use the term automatic differentiation to refer to the process of generating a computer program for the derivatives given the program for the original function and the input variables

Wengert. CACM 1964. A simple automatic derivative evaluation program.

# aside: higher-order derivatives*

- the Hessian was assumed fixed and isotropic in gradient descent; if we knew it, we could use the Newton method instead and solve all curvature-related problems
- given $f : \mathbb{R}^p \to \mathbb{R}$, its Hessian matrix at $\mathbf{x}$ is

$$Hf(\mathbf{x}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_p \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_p^2} \end{pmatrix} (\mathbf{x}) = \nabla(Df)(\mathbf{x})$$

- unfortunately, this is a $p \times p$ matrix and with $p$ in the order of millions, it is impractical even to store it, let alone compute it

# aside: multiplication by Hessian*

[Pearlmutter 1994]

- fortunately, in many cases what we need is only the product of the Hessian with a given vector $\mathbf{v}$, which is just a vector in $\mathbb{R}^p$

$$\mathbf{v}^\top \cdot Hf(\mathbf{x}) = \mathbf{v}^\top \cdot \nabla(Df)(\mathbf{x}) = \nabla_\mathbf{v}(Df)(\mathbf{x})$$

- here $\nabla_\mathbf{v}$ is the directional derivative operator

$$\nabla_\mathbf{v}(f) := \mathbf{v}^\top \cdot \nabla f$$

- remember that in back-propagation, for each variable $\mathbf{x}$, we defined a vector $d\mathbf{x}$, which was computed in the backward pass
- so all we need to do is allocate another vector $\nabla_\mathbf{v}(\mathbf{x})$ for the forward pass and another $\nabla_\mathbf{v}(d\mathbf{x})$ for the backward, and compute them by applying the chain rule in both passes!

Pearlmutter. NC 1994. Fast Exact Multiplication By the Hessian.

# aside: multiplication by Hessian*

**[Pearlmutter 1994]**

- fortunately, in many cases what we need is only the product of the Hessian with a given vector $\mathbf{v}$, which is just a vector in $\mathbb{R}^p$

$$\mathbf{v}^\top \cdot Hf(\mathbf{x}) = \mathbf{v}^\top \cdot \nabla(Df)(\mathbf{x}) = \nabla_{\mathbf{v}}(Df)(\mathbf{x})$$

- here $\nabla_{\mathbf{v}}$ is the directional derivative operator

$$\nabla_{\mathbf{v}}(f) := \mathbf{v}^\top \cdot \nabla f$$

- remember that in back-propagation, for each variable $\mathbf{x}$, we defined a vector $d\mathbf{x}$, which was computed in the backward pass
- so all we need to do is allocate another vector $\nabla_{\mathbf{v}}(\mathbf{x})$ for the forward pass and another $\nabla_{\mathbf{v}}(d\mathbf{x})$ for the backward, and compute them by applying the chain rule in both passes!

Pearlmutter. NC 1994. Fast Exact Multiplication By the Hessian.

# automatic differentiation: units

# automatic differentiation

## forward

- evaluation is carried out by units, one calling another
- when invoked, each unit generates a node object
- each node holds the gradient with respect to its unit's inputs, including parameters
- it also holds any information needed for the backward pass

## backward

- all gradients are set to zero, except for the gradient with respect to the scalar quantity that is to be optimized (the error), which is set to one
- the $\mathrm{back}()$ method is invoked on the node of this quantity
- this, in turn, triggers the same method on all units that have participated in the forward pass

# automatic differentiation

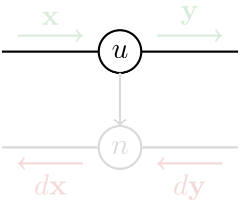
## forward

- evaluation is carried out by units, one calling another
- when invoked, each unit generates a node object
- each node holds the gradient with respect to its unit's inputs, including parameters
- it also holds any information needed for the backward pass

## backward

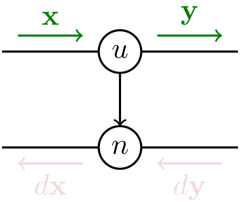
- all gradients are set to zero, except for the gradient with respect to the scalar quantity that is to be optimized (the error), which is set to one
- the $\mathrm{back}()$ method is invoked on the node of this quantity
- this, in turn, triggers the same method on all units that have participated in the forward pass

# units and nodes



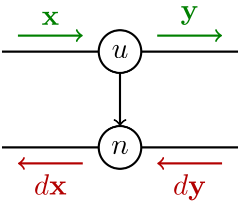

- unit $u$ manually generates node $n$

# units and nodes




- unit $u$ manually generates node $n$

# units and nodes




- unit $u$ manually generates node $n$

# units and nodes

- given a function $f$ with derivative $Df$, a *unit* is a function of the form

$$
\begin{aligned}
&\textbf{def } \text{forward}(\mathbf{x}_1, \dots, \mathbf{x}_n): \\
&\quad \mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\quad \textbf{def } \text{back}(d\mathbf{y}, d\mathbf{x}_1, \dots, d\mathbf{x}_n): \\
&\qquad d\mathbf{x}_1^\top \mathrel{+}= d\mathbf{y}^\top \cdot D_1 f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\qquad\quad \vdots \\
&\qquad d\mathbf{x}_n^\top \mathrel{+}= d\mathbf{y}^\top \cdot D_n f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\quad \textbf{return } \mathit{node}(\mathbf{y}, \text{back})
\end{aligned}
$$

- a *node* object:
  - holds $\mathbf{y}$ and an associated derivative $d\mathbf{y}$ of the same shape
  - exposes a method $\text{back}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ where $\mathbf{x}_i$ can be *node*s
  - automatically adds its own $d\mathbf{y}$ as first argument
  - if an input $\mathbf{x}_i$ is a *node*, extracts the derivative part $d\mathbf{x}_i$
  - otherwise, $d\mathbf{x}_i$ is an object for which operation $\mathrel{+}=$ is ignored

# units and nodes

- given a function $f$ with derivative $Df$, a *unit* is a function of the form

$$
\begin{aligned}
&\textbf{def } \mathrm{forward}(\mathbf{x}_1, \dots, \mathbf{x}_n): \\
&\quad \mathbf{y} = f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\quad \textbf{def } \mathrm{back}(d\mathbf{y}, d\mathbf{x}_1, \dots, d\mathbf{x}_n): \\
&\qquad d\mathbf{x}_1^\top \mathrel{+}= d\mathbf{y}^\top \cdot D_1 f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\qquad \vdots \\
&\qquad d\mathbf{x}_n^\top \mathrel{+}= d\mathbf{y}^\top \cdot D_n f(\mathbf{x}_1, \dots, \mathbf{x}_n) \\
&\quad \textbf{return } \mathit{node}(\mathbf{y}, \mathrm{back})
\end{aligned}
$$

- a *node* object:
  - holds $\mathbf{y}$ and an associated derivative $d\mathbf{y}$ of the same shape
  - exposes a method $\mathrm{back}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ where $\mathbf{x}_i$ can be *node*s
  - automatically adds its own $d\mathbf{y}$ as first argument
  - if an input $\mathbf{x}_i$ is a *node*, extracts the derivative part $d\mathbf{x}_i$
  - otherwise, $d\mathbf{x}_i$ is an object for which operation $+=$ is ignored

# the affine unit

- input vectors are represented as rows of $m \times p$ input matrix $X$ where $m$ is the mini-batch size and $p$ the input dimension
- parameters are represented by $p \times q$ weight matrix $W$ and $1 \times q$ bias vector $\mathbf{b}$ where $q$ is the output dimension
- the unit is defined as

**def** $\text{affine}(X, (W, \mathbf{b}))$:
  $A = \text{dot}(X, W) + \mathbf{b}$
  **def** $\text{back}(dA, dX, (dW, d\mathbf{b}))$:
    $dW \mathrel{+}= \text{dot}(X^\top, dA)$
    $d\mathbf{b} \mathrel{+}= \text{sum}_0(dA)$
    $dX \mathrel{+}= \text{dot}(dA, W^\top)$
  **return** $node(A, \text{back})$

# the affine unit

- input vectors are represented as rows of $m \times p$ input matrix $X$ where $m$ is the mini-batch size and $p$ the input dimension
- parameters are represented by $p \times q$ weight matrix $W$ and $1 \times q$ bias vector $\mathbf{b}$ where $q$ is the output dimension
- the unit is defined as

$$
\begin{array}{l}
\textbf{def } \mathrm{affine}(X, (W, \mathbf{b})): \\
\quad A = \mathrm{dot}(X, W) + \mathbf{b} \\
\quad \textbf{def } \mathrm{back}(dA, dX, (dW, d\mathbf{b})): \\
\quad\quad dW \mathrel{+}= \mathrm{dot}(X^\top, dA) \\
\quad\quad d\mathbf{b} \mathrel{+}= \mathrm{sum}_0(dA) \\
\quad\quad dX \mathrel{+}= \mathrm{dot}(dA, W^\top) \\
\quad \textbf{return } \mathit{node}(A, \mathrm{back})
\end{array}
$$

# the affine unit in math*

## forward

- input $X \in \mathbb{R}^{m \times p}$, $W \in \mathbb{R}^{p \times q}$, $\mathbf{b} \in \mathbb{R}^q$, output $A \in \mathbb{R}^{m \times q}$

$$A = f(X; W, \mathbf{b}) := XW + \mathbf{1}_m \mathbf{b}^\top$$

observe that in the code, addition of $\mathbf{b}$ is handled by broadcasting

## backward

- if $\mathbf{a}_i$, $\mathbf{w}_i$ is the $i$-th column of $A$, $W$,

$$\frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_i} = \frac{\partial (X\mathbf{w}_i)}{\partial \mathbf{w}_i} = X$$

and there are no other dependencies, so by the chain rule

$$d\mathbf{w}_i^\top := \frac{\partial}{\partial \mathbf{w}_i} = \frac{\partial}{\partial \mathbf{a}_i} \cdot \frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_i} = d\mathbf{a}_i^\top \cdot X$$

- finally, the partial derivative with respect to $W$

$$dW = (dA^\top X)^\top = X^\top dA$$

# the affine unit in math*

## forward

- input $X \in \mathbb{R}^{m \times p}$, $W \in \mathbb{R}^{p \times q}$, $\mathbf{b} \in \mathbb{R}^q$, output $A \in \mathbb{R}^{m \times q}$

$$A = f(X; W, \mathbf{b}) := XW + \mathbf{1}_m \mathbf{b}^\top$$

observe that in the code, addition of $\mathbf{b}$ is handled by broadcasting

## backward

- if $\mathbf{a}_i$, $\mathbf{w}_i$ is the $i$-th column of $A$, $W$,

$$\frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_i} = \frac{\partial (X\mathbf{w}_i)}{\partial \mathbf{w}_i} = X$$

and there are no other dependencies, so by the chain rule

$$d\mathbf{w}_i^\top := \frac{\partial}{\partial \mathbf{w}_i} = \frac{\partial}{\partial \mathbf{a}_i} \cdot \frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_i} = d\mathbf{a}_i^\top \cdot X$$

- finally, the partial derivative with respect to $W$

$$dW = (dA^\top X)^\top = X^\top dA$$

# the affine unit in math*

- by symmetry, writing $A^\top = W^\top X^\top + \mathbf{b}\mathbf{1}_m^\top$ and using the previous result for $dW$, we find $dX^\top = (W^\top)^\top dA^\top$ or

$$dX = (dA)W^\top$$

- again, by replacing $X$ and $W$ by $\mathbf{1}_m$ and $\mathbf{b}^\top$ respectively in the previous result for $dW$,

$$d\mathbf{b}^\top = (dA^\top \mathbf{1}_m)^\top = \mathbf{1}_m^\top dA$$

- observe that distributing $\mathbf{b}$ in the forward yields a sum in the backward

# the affine unit in math*

- by symmetry, writing $A^\top = W^\top X^\top + \mathbf{b}\mathbf{1}_m^\top$ and using the previous result for $dW$, we find $dX^\top = (W^\top)^\top dA^\top$ or

$$dX = (dA)W^\top$$

- again, by replacing $X$ and $W$ by $\mathbf{1}_m$ and $\mathbf{b}^\top$ respectively in the previous result for $dW$,

$$d\mathbf{b}^\top = (dA^\top \mathbf{1}_m)^\top = \mathbf{1}_m^\top dA$$

- observe that distributing $\mathbf{b}$ in the forward yields a sum in the backward

# the logistic unit

- the input is an $m \times q$ activation matrix $A$ and the $m \times k$ one-of-$k$ encoded target matrix, where $k$ is the number of classes
- there are no parameters
- the unit integrates softmax with average cross-entropy loss

**def** $\mathrm{logistic}(A, T)$:
  $E = \exp(A)$
  $Y = E / \mathrm{sum}_1(E)$
  $C = -\mathrm{sum}_1(T * \log(Y))$
  $D = \mathrm{sum}_0(C)/m$
  **def** $\mathrm{back}(dD, dA, \_)$:
    $dA \mathrel{+}= dD * (Y - T)/m$
  **return** $node(D, \mathrm{back})$

# the logistic unit

- the input is an $m \times q$ activation matrix $A$ and the $m \times k$ one-of-$k$ encoded target matrix, where $k$ is the number of classes
- there are no parameters
- the unit integrates $\mathrm{softmax}$ with average cross-entropy loss

$$
\begin{aligned}
&\textbf{def } \mathrm{logistic}(A, T)\text{:} \\
&\quad E = \exp(A) \\
&\quad Y = E/\mathrm{sum}_1(E) \\
&\quad C = -\mathrm{sum}_1(T * \log(Y)) \\
&\quad D = \mathrm{sum}_0(C)/m \\
&\quad \textbf{def } \mathrm{back}(dD, dA, \_)\text{:} \\
&\qquad dA \mathrel{+}= dD * (Y - T)/m \\
&\quad \textbf{return } node(D, \mathrm{back})
\end{aligned}
$$

# the logistic unit in math*

## forward

- $E$ is given element-wise as $e_{ij} = \exp(a_{ij})$, and $m \times q$ matrix $Y$ is row-normalized as

$$Y = (\mathrm{diag}(E\mathbf{1}_k))^{-1}E$$

- the $i$-th row of $Y$ is the $\mathrm{softmax}$ output of the $i$-th input sample representing the $k$ posterior class probabilities
- $C$ is actually a $m \times 1$ column vector and its $i$-th element represents the cross-entropy loss of the $i$-th input sample

$$c_i = -\sum_{j=1}^{k} t_{ij} \log(y_{ij})$$

- finally, $D = \frac{1}{m}\sum_{i=1}^{m} c_i$ is a scalar and represents the average cross-entropy (data) error over the mini-batch

# the logistic unit in math*

## forward

- $E$ is given element-wise as $e_{ij} = \exp(a_{ij})$, and $m \times q$ matrix $Y$ is row-normalized as

$$Y = (\operatorname{diag}(E\mathbf{1}_k))^{-1}E$$

- the $i$-th row of $Y$ is the $\operatorname{softmax}$ output of the $i$-th input sample representing the $k$ posterior class probabilities
- $C$ is actually a $m \times 1$ column vector and its $i$-th element represents the cross-entropy loss of the $i$-th input sample

$$c_i = -\sum_{j=1}^{k} t_{ij} \log(y_{ij})$$

- finally, $D = \frac{1}{m}\sum_{i=1}^{m} c_i$ is a scalar and represents the average cross-entropy (data) error over the mini-batch

# the logic unit in math*

## backward

- if $\mathbf{a}_i^\top$, $\mathbf{y}_i^\top$, $\mathbf{t}_i^\top$ is the $i$-th row of $A$, $Y$, $T$, the derivative of the cross-entropy loss is, according to what we have seen,

$$\frac{\partial c_i}{\partial \mathbf{a}_i}(\mathbf{a}_i, \mathbf{t}_i) = (\boldsymbol{\sigma}(\mathbf{a}_i) - \mathbf{t}_i)^\top = (\mathbf{y}_i - \mathbf{t}_i)^\top$$

- since $D$ is the average of the individual sample losses $c_i$, the derivative of the total error, which is $1$ by default, is distributed over the samples with a factor of $\frac{1}{m}$

$$dA^\top = \frac{1}{m}(Y - T) \cdot dD$$

# why integrate $\mathrm{softmax}$ with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# why integrate softmax with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# why integrate softmax with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# why integrate $\mathrm{softmax}$ with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# why integrate softmax with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# why integrate softmax with cross-entropy?

- the simplified formula is faster compared to blind application of back-propagation at the level of elementary functions
- if this is not convincing, try evaluating the binary cross-entropy loss

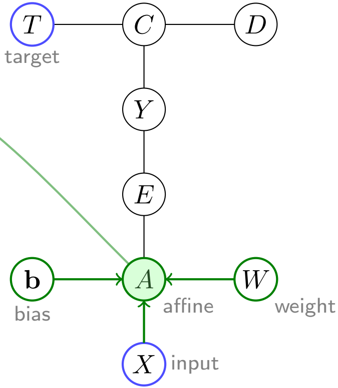
$$\ell(x) := \ln(1 + e^{-x})$$

- $\ell(-1) = 1.3133$
- $\ell(-2) = 2.1269$
- $\ell(-5) = 5.0067$
- $\ell(-10) = 10.0000$
- $\ell(-20) = 20.0000$
- $\ell(-50) = 50.0000$
- $\ell(-100) = 100.0000$
- $\ell(-200) = 200.0000$
- $\ell(-500) = 500.0000$
- $\ell(-1000) = \infty$
- $\ell(-2000) = \infty$

# back-propagation

**forward**

$A = \text{dot}(X, W) + \mathbf{b}$



# back-propagation

**forward**

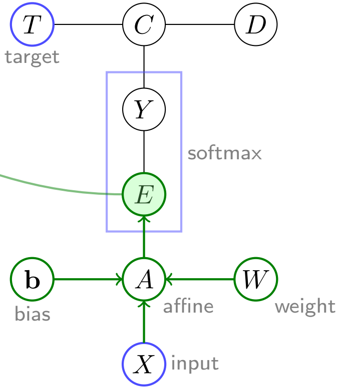
$A = \text{dot}(X, W) + \mathbf{b}$

$E = \exp(A)$

$Y = E/\text{sum}_1(E)$

$C = -\text{sum}_1(T * \log(Y))$

$D = \text{sum}_0(C)/m$



# back-propagation

**forward**

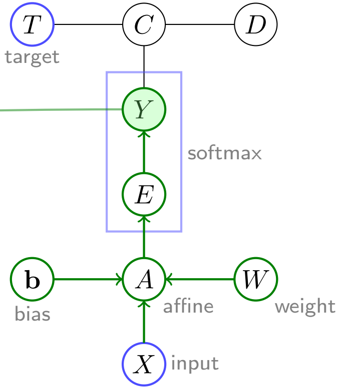
$A = \text{dot}(X, W) + \mathbf{b}$

$E = \exp(A)$

$Y = E / \text{sum}_1(E)$

$C = -\text{sum}_1(T * \log(Y))$

$D = \text{sum}_0(C)/m$



# back-propagation

**forward**

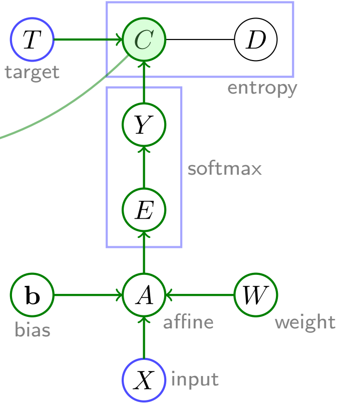
$A = \text{dot}(X, W) + \mathbf{b}$

$E = \exp(A)$

$Y = E / \text{sum}_1(E)$

$C = -\text{sum}_1(T * \log(Y))$

$D = \text{sum}_0(C)/m$



# back-propagation

**forward**

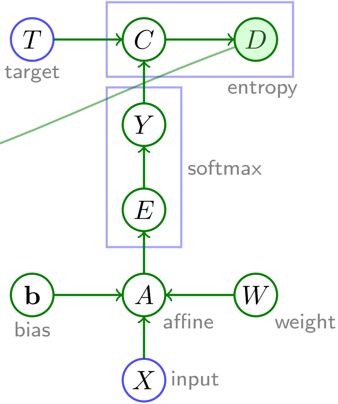
$A = \mathrm{dot}(X, W) + \mathbf{b}$

$E = \exp(A)$

$Y = E/\mathrm{sum}_1(E)$

$C = -\mathrm{sum}_1(T * \log(Y))$

$D = \mathrm{sum}_0(C)/m$



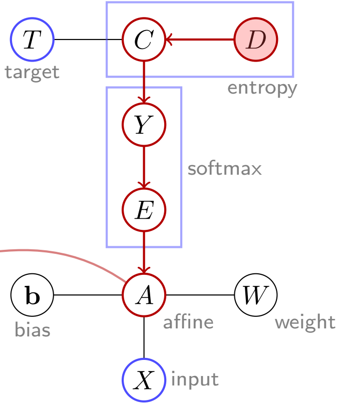
# back-propagation

**forward**

$A = \text{dot}(X, W) + \mathbf{b}$
$E = \exp(A)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$



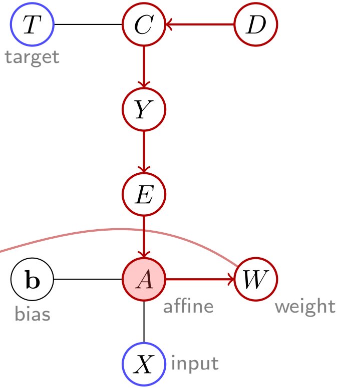
# back-propagation

**forward**

$A = \text{dot}(X, W) + \mathbf{b}$
$E = \exp(A)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$
$dW \mathrel{+}= \text{dot}(X^\top, dA)$
$d\mathbf{b} = \text{sum}_0(dA)$



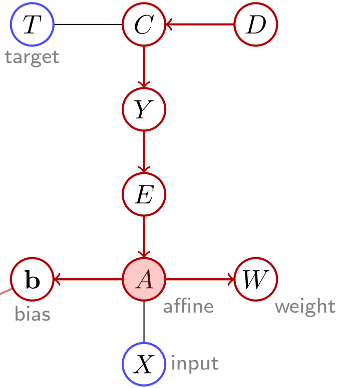
# back-propagation

**forward**

$A = \text{dot}(X, W) + \mathbf{b}$
$E = \exp(A)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$
$dW += \text{dot}(X^\top, dA)$
$d\mathbf{b} = \text{sum}_0(dA)$



# automatic differentiation

**forward**

$$A = \mathrm{dot}(X, W) + \mathbf{b}$$
$$E = \exp(A)$$
$$Y = E/\mathrm{sum}_1(E)$$
$$C = -\mathrm{sum}_1(T * \log(Y))$$
$$D = \mathrm{sum}_0(C)/m$$

**backward**

$$dA = dD * (Y - T)/m$$
$$dW \mathrel{+}= \mathrm{dot}(X^\top, dA)$$
$$d\mathbf{b} = \mathrm{sum}_0(dA)$$

now we organize forward and backward code into units

# automatic differentiation

**forward**

$A = \text{dot}(X, W) + \mathbf{b}$
$E = \exp(A)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$
$dW \mathrel{+}= \text{dot}(X^\top, dA)$
$d\mathbf{b} = \text{sum}_0(dA)$

**def** $\text{affine}(X, (W, \mathbf{b}))$:
    $A = \text{dot}(X, W) + \mathbf{b}$
    **def** $\text{back}(dA, dX, (dW, d\mathbf{b}))$:
        $dW \mathrel{+}= \text{dot}(X^\top, dA)$
        $d\mathbf{b} \mathrel{+}= \text{sum}_0(dA)$
        $dX \mathrel{+}= \text{dot}(dA, W^\top)$
    **return** $\mathit{node}(A, \text{back})$

# automatic differentiation

**forward**

$A = \text{affine}(X, (W, \mathbf{b}))$
$E = \exp(A)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$
$A.\text{back}(X, (W, \mathbf{b}))$

**def** $\text{affine}(X, (W, \mathbf{b}))$:
  $A = \text{dot}(X, W) + \mathbf{b}$
  **def** $\text{back}(dA, dX, (dW, d\mathbf{b}))$:
    $dW \mathrel{+}= \text{dot}(X^\top, dA)$
    $d\mathbf{b} \mathrel{+}= \text{sum}_0(dA)$
    $dX \mathrel{+}= \text{dot}(dA, W^\top)$
  **return** $\mathit{node}(A, \text{back})$

# automatic differentiation

**forward**

$A = \text{affine}(X, (W, \mathbf{b}))$
$E = \exp(A)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$

**backward**

$dA = dD * (Y - T)/m$
$A.\text{back}(X, (W, \mathbf{b}))$

**def** $\text{logistic}(A, T)$:
$E = \exp(A)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
**def** $\text{back}(dD, dA, \_)$:
$dA \mathrel{+}= dD * (Y - T)/m$
**return** $\textit{node}(D, \text{back})$

# automatic differentiation

**forward**

$A = \text{affine}(X, (W, \mathbf{b}))$

$D = \text{entropy}(A, T)$

```
def logistic(A, T):
    E = exp(A)
    Y = E / sum_1(E)
    C = -sum_1(T * log(Y))
    D = sum_0(C) / m
    def back(dD, dA, _):
        dA += dD * (Y - T) / m
    return node(D, back)
```

**backward**

$D.\text{back}(A, T)$

$A.\text{back}(X, (W, \mathbf{b}))$

# automatic differentiation: functions

# the relu unit*

- relu is an element-wise activation function; its input is activation matrix $A$ and returns matrix $Z$ of the same size
- its backward pass behaves like a switch

**def** relu($A$):
    $Z = \max(0, A)$
    **def** back($dZ, dA$):
        $dA \mathrel{+}= dZ * (Z > 0)$
    **return** $node(Z, \text{back})$

# the relu unit*

- relu is an element-wise activation function; its input is activation matrix $A$ and returns matrix $Z$ of the same size
- its backward pass behaves like a switch

$$
\begin{aligned}
&\textbf{def } \mathrm{relu}(A): \\
&\quad Z = \max(0, A) \\
&\quad \textbf{def } \mathrm{back}(dZ, dA): \\
&\quad\quad dA \mathrel{+}= dZ * (Z > 0) \\
&\quad \textbf{return } \mathit{node}(Z, \mathrm{back})
\end{aligned}
$$

# the decay unit*

- it takes as input a tuple or list $W$ of weight matrices of any size and returns the weight decay error term $\frac{\lambda}{2}\|w\|^2$ for each $w \in W$, where $\|\cdot\|_F$ is the Frobenius norm
- the backward derivative is proportional to $w$, as for the $\ell_2$ norm

```
def decay(W):
    R = λ/2 * sum(‖w‖_F^2 for w in W)
    def back(dR, dW):
        for (w, dw) in zip(W, dW):
            dw += dR * λ * w
    return node(R, back)
```

# the decay unit*

- it takes as input a tuple or list $W$ of weight matrices of any size and returns the weight decay error term $\frac{\lambda}{2}\|w\|^2$ for each $w \in W$, where $\|\cdot\|_F$ is the Frobenius norm
- the backward derivative is proportional to $w$, as for the $\ell_2$ norm

```
def decay(W):
    R = λ/2 * sum(‖w‖_F^2 for w in W)
    def back(dR, dW):
        for (w, dw) in zip(W, dW):
            dw += dR * λ * w
    return node(R, back)
```

# the add unit*

- it takes as input a tuple or list $X$ of matrices (or vectors, or scalars) of the same size and returns their sum
- its backward pass distributes the derivative to all input branches

```
def add(X):
    S = sum(X)
    def back(dS, dX):
        for dx in dX:
            dx += dS
    return node(S, back)
```

- operator $+$ is overloaded for *node*s such that $A + B$ means $\mathrm{add}((A, B))$

# the add unit*

- it takes as input a tuple or list $X$ of matrices (or vectors, or scalars) of the same size and returns their sum
- its backward pass distributes the derivative to all input branches

```
def add(X):
    S = sum(X)
    def back(dS, dX):
        for dx in dX:
            dx += dS
    return node(S, back)
```

- operator $+$ is overloaded for *node*s such that $A + B$ means $\mathrm{add}((A, B))$

# the add unit*

- it takes as input a tuple or list $X$ of matrices (or vectors, or scalars) of the same size and returns their sum
- its backward pass distributes the derivative to all input branches

```
def add(X):
    S = sum(X)
    def back(dS, dX):
        for dx in dX:
            dx += dS
    return node(S, back)
```

- operator $+$ is overloaded for *node*s such that $A + B$ means $\mathrm{add}((A, B))$

# the $\mathrm{loss}$ function

- it takes as input the activation matrix $A$, the target matrix $T$ and the weight matrix list $W$
- it calls the $\mathrm{logistic}$ unit on $(A, T)$ and the $\mathrm{decay}$ unit on $W$, and returns the sum of the two scalar terms

$$
\begin{array}{l}
\textbf{def } \mathrm{loss}(A, T, W): \\
\quad L = \mathrm{logistic}(A, T) + \mathrm{decay}(W) \\
\quad \textbf{return } \mathit{block}(L)
\end{array}
$$

- addition is handled by $\mathrm{add}$ and the error derivative flows backward to both branches

# the $\mathrm{loss}$ function

- it takes as input the activation matrix $A$, the target matrix $T$ and the weight matrix list $W$
- it calls the $\mathrm{logistic}$ unit on $(A, T)$ and the $\mathrm{decay}$ unit on $W$, and returns the sum of the two scalar terms

**def** $\mathrm{loss}(A, T, W)$:
  $L = \mathrm{logistic}(A, T) + \mathrm{decay}(W)$
  **return** $block(L)$

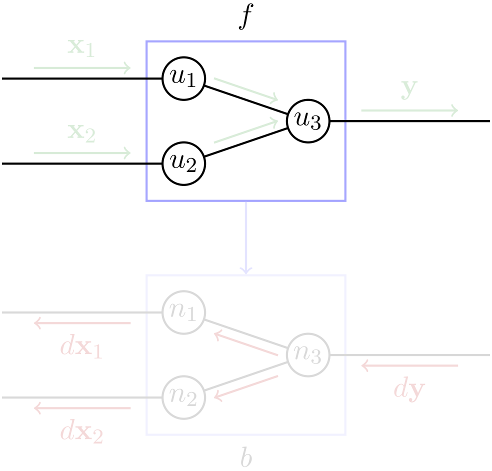
- addition is handled by $\mathrm{add}$ and the error derivative flows backward to both branches

# the model function

- this is a two-layer network model where an affine layer is followed by a $\mathrm{relu}$ activation function and another affine layer
- the parameter tuple $U_i = (W_i, \mathbf{b}_i)$ for layer $i$ contains a weight matrix $W_i$ and a bias vector $\mathbf{b}_i$
- unit calls are nested like every other function

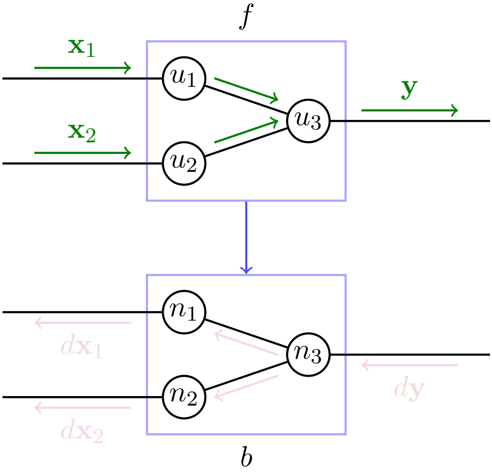
$$
\begin{array}{l}
\textbf{def } \mathrm{model}(X, (U_1, U_2)): \\
\quad A = \mathrm{affine}(\mathrm{relu}(\mathrm{affine}(X, U_1)), U_2) \\
\quad \textbf{return } \mathit{block}(A)
\end{array}
$$

# functions and blocks



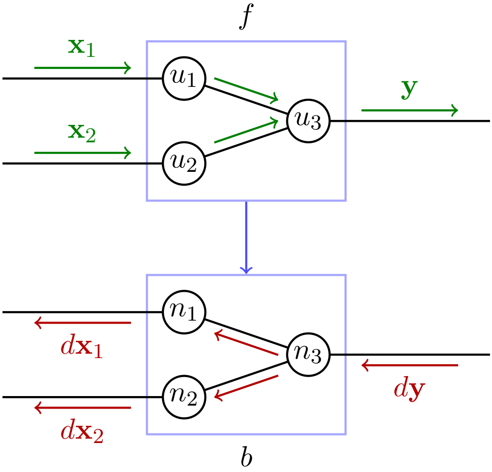

- function $f$ containing units $u_1, u_2, u_3$
- $f$ dynamically generates block $b$ containing nodes $n_1, n_2, n_3$, manually generated by $u_1, u_2, u_3$ respectively

# functions and blocks




- function $f$ containing units $u_1, u_2, u_3$
- $f$ dynamically generates block $b$ containing nodes $n_1, n_2, n_3$, manually generated by $u_1, u_2, u_3$ respectively

# functions and blocks




- function $f$ containing units $u_1, u_2, u_3$
- $f$ dynamically generates block $b$ containing nodes $n_1, n_2, n_3$, manually generated by $u_1, u_2, u_3$ respectively

# functions and blocks

- a *function* is a function of the following form, where code is arbitrary but computation takes place through calls to *unit*s or *function*s

$$
\begin{aligned}
&\textbf{def } name(\mathbf{x}_1, \ldots, \mathbf{x}_n)\text{:} \\
&\quad \langle \text{code generating the following} \rangle \\
&\quad \mathbf{r}_1 = \mathrm{call}_1(\mathbf{a}_1, \ldots, \mathbf{a}_{n_1}) \\
&\qquad \vdots \\
&\quad \mathbf{r}_s = \mathrm{call}_s(\mathbf{a}_1, \ldots, \mathbf{a}_{n_s}) \\
&\quad \textbf{return } block(\mathbf{r}_s)
\end{aligned}
$$

- all calls are recorded as a list of *unit*s or *function*s by call order, each associated with a list of arguments
- a *block* object is a *node*, but
  - its method $\mathrm{back}()$ does not add its own derivative in the arguments
  - its method $\mathrm{back}()$ is automatically generated and its body calls the recorded functions with the same arguments in reverse order

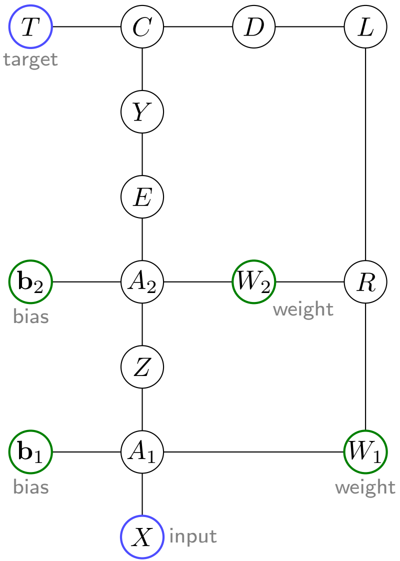
# functions and blocks

- a *function* is a function of the following form, where code is arbitrary but computation takes place through calls to *unit*s or *function*s

$$
\begin{aligned}
&\textbf{def } name(\mathbf{x}_1, \ldots, \mathbf{x}_n):\\
&\quad \langle\text{code generating the following}\rangle\\
&\quad \mathbf{r}_1 = \text{call}_1(\mathbf{a}_1, \ldots, \mathbf{a}_{n_1})\\
&\qquad \vdots\\
&\quad \mathbf{r}_s = \text{call}_s(\mathbf{a}_1, \ldots, \mathbf{a}_{n_s})\\
&\quad \textbf{return } block(\mathbf{r}_s)
\end{aligned}
$$

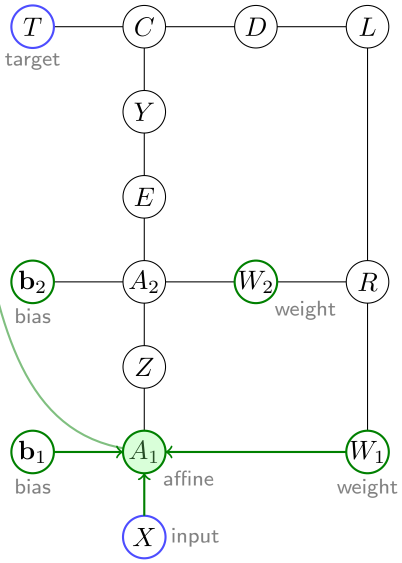
- all calls are recorded as a list of *unit*s or *function*s by call order, each associated with a list of arguments
- a *block* object is a *node*, but
  - its method back() does not add its own derivative in the arguments
  - its method back() is automatically generated and its body calls the recorded functions with the same arguments in reverse order

# back-propagation



# back-propagation

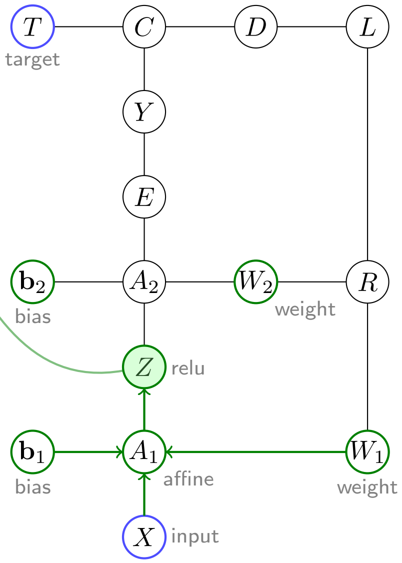
$$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$$



# back-propagation

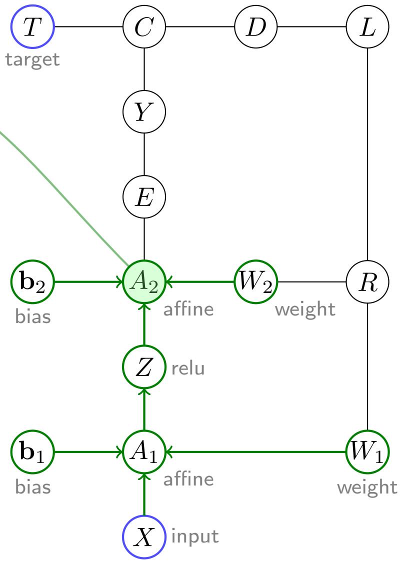
$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$

$Z = \max(0, A_1)$



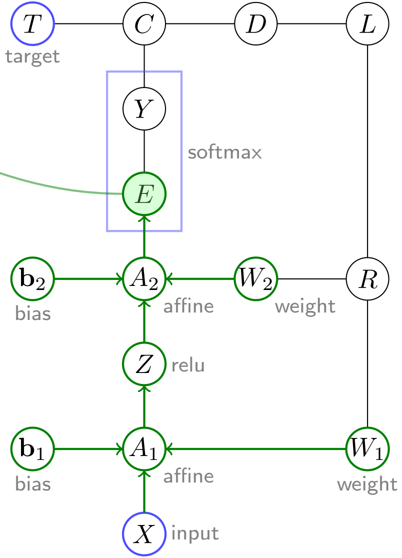
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$



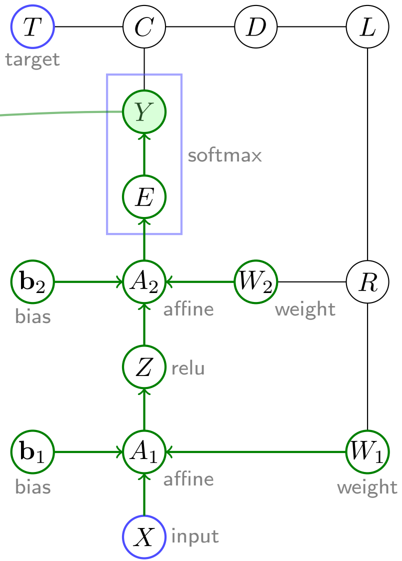
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \text{max}(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \text{exp}(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \text{log}(Y))$
$D = \text{sum}_0(C)/m$



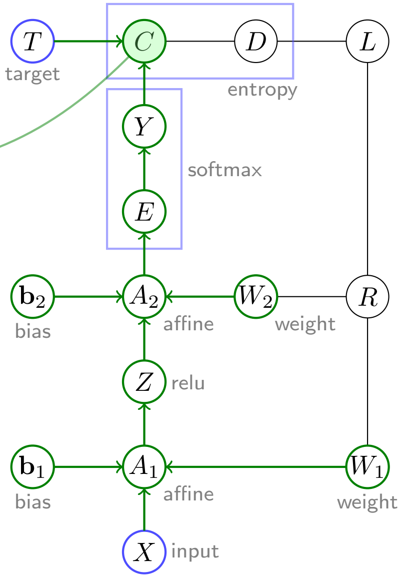
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$



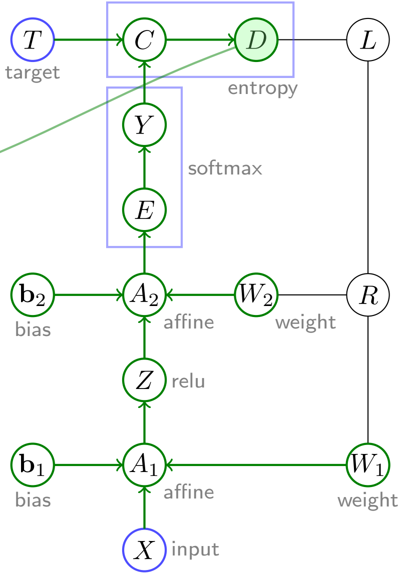
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$



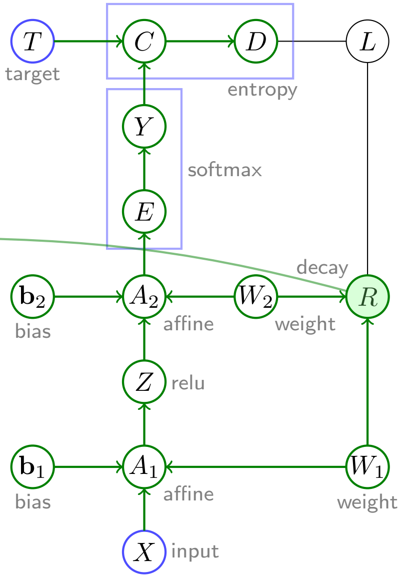
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$



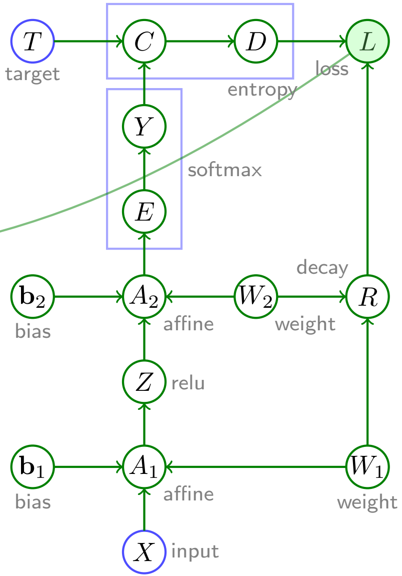
# back-propagation

$$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$$
$$E = \exp(A_2)$$
$$Y = E/\text{sum}_1(E)$$
$$C = -\text{sum}_1(T * \log(Y))$$
$$D = \text{sum}_0(C)/m$$
$$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$$



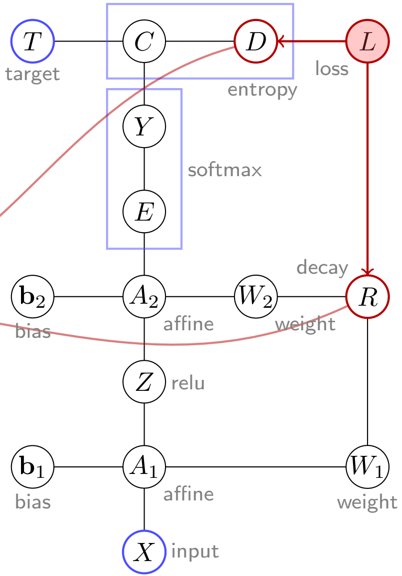
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \text{max}(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \text{exp}(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \text{log}(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$



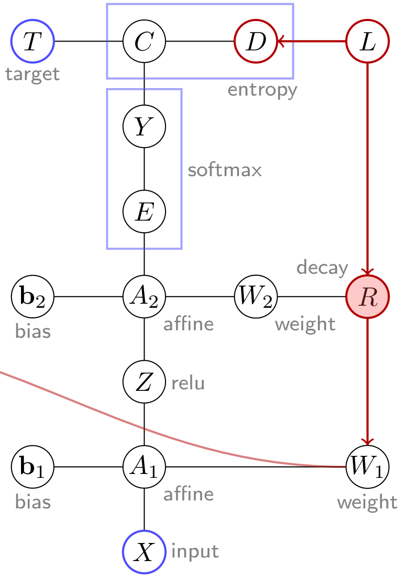
# back-propagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}}_1(E)$
$C = -\textcolor{blue}{\mathrm{sum}}_1(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$



# back-propagation

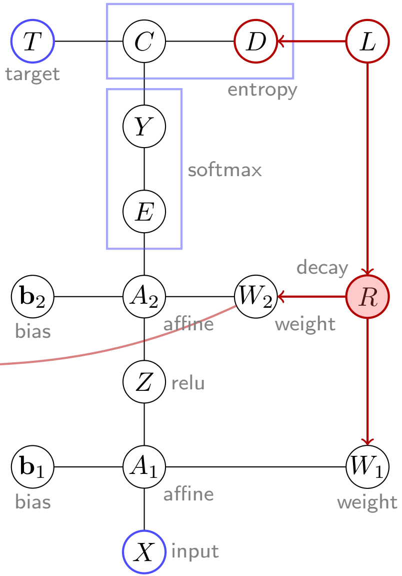
$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$



# back-propagation

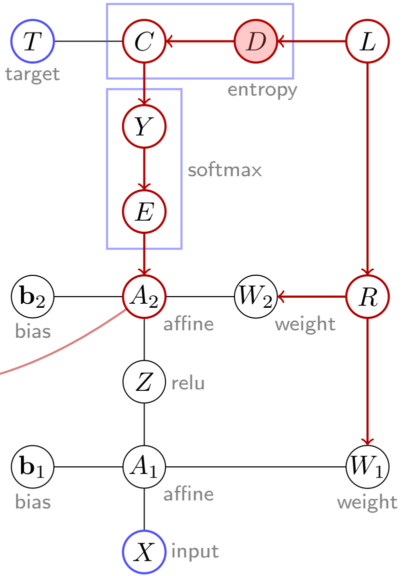
$A_1 = \textcolor{blue}{\text{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\text{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E / \textcolor{blue}{\text{sum}_1}(E)$
$C = -\textcolor{blue}{\text{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\text{sum}_0}(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$

$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$



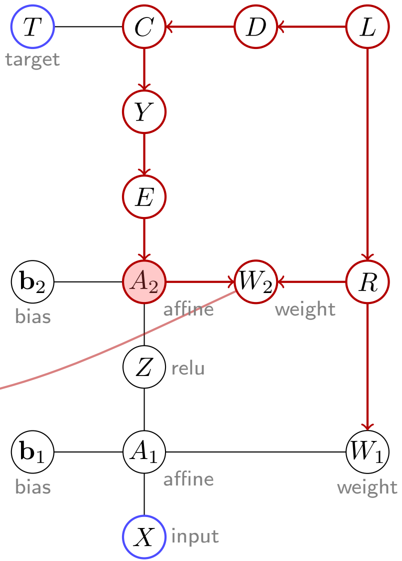
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$



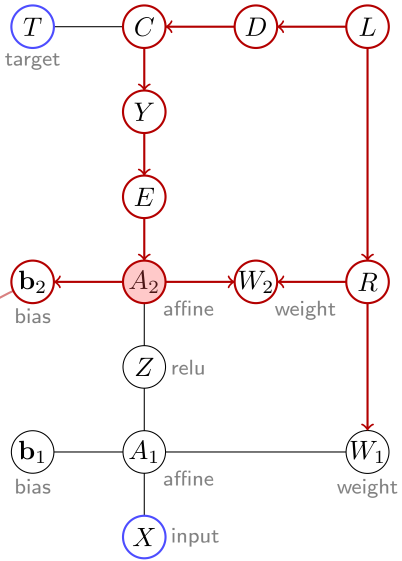
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$



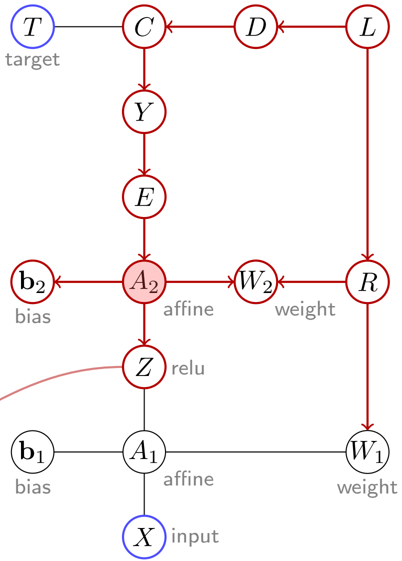
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$



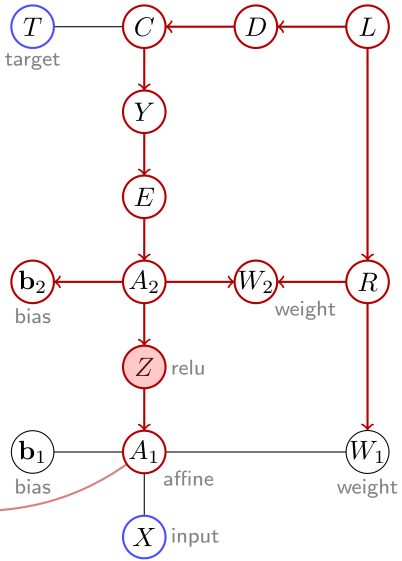
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$



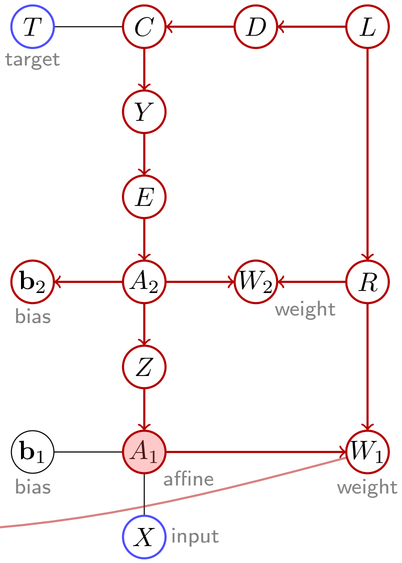
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$
$dA_1 = dZ * (Z > 0)$



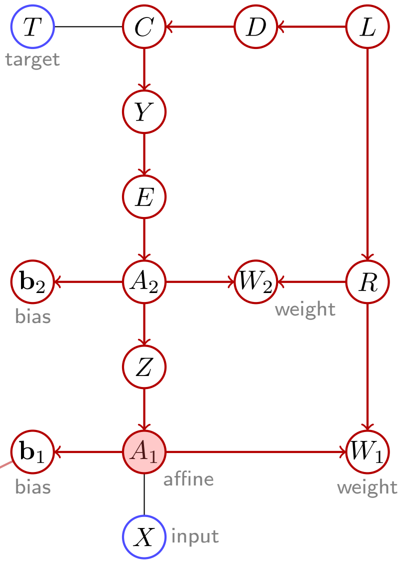
# back-propagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$
$dA_1 = dZ * (Z > 0)$
$dW_1 \mathrel{+}= \text{dot}(X^\top, dA_1)$
$d\mathbf{b}_1 = \text{sum}_0(dA_1)$



# back-propagation

$A_1 = \textcolor{blue}{\text{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\text{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\text{sum}_1}(E)$
$C = -\textcolor{blue}{\text{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\text{sum}_0}(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \textcolor{blue}{\text{dot}}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \textcolor{blue}{\text{sum}_0}(dA_2)$
$dZ = \textcolor{blue}{\text{dot}}(dA_2, W_2^\top)$
$dA_1 = dZ * (Z > 0)$
$dW_1 \mathrel{+}= \textcolor{blue}{\text{dot}}(X^\top, dA_1)$
$d\mathbf{b}_1 = \textcolor{blue}{\text{sum}_0}(dA_1)$



# automatic differentiation

$$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$$
$$E = \exp(A_2)$$
$$Y = E / \text{sum}_1(E)$$
$$C = -\text{sum}_1(T * \log(Y))$$
$$D = \text{sum}_0(C)/m$$
$$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$$
$$L = D + R$$

$$(dD, dR) = (dL, dL)$$
$$dW_1 = dR * \lambda * W_1$$
$$dW_2 = dR * \lambda * W_2$$
$$dA_2 = dD * (Y - T)/m$$
$$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$$
$$d\mathbf{b}_2 = \text{sum}_0(dA_2)$$
$$dZ = \text{dot}(dA_2, W_2^\top)$$
$$dA_1 = dZ * (Z > 0)$$
$$dW_1 \mathrel{+}= \text{dot}(X^\top, dA_1)$$
$$d\mathbf{b}_1 = \text{sum}_0(dA_1)$$

now we organize forward and backward code into units and functions

# automatic differentiation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$
$dA_1 = dZ * (Z > 0)$
$dW_1 \mathrel{+}= \text{dot}(X^\top, dA_1)$
$d\mathbf{b}_1 = \text{sum}_0(dA_1)$

**def** $\text{relu}(A)$:
  $Z = \max(0, A)$
  **def** $\text{back}(dZ, dA)$:
    $dA \mathrel{+}= dZ * (Z > 0)$
  **return** $\textit{node}(Z, \text{back})$

# automatic differentiation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \text{relu}(A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \text{sum}_0(dA_2)$
$dZ = \text{dot}(dA_2, W_2^\top)$
$Z.\text{back}(A_1)$
$dW_1 \mathrel{+}= \text{dot}(X^\top, dA_1)$
$d\mathbf{b}_1 = \text{sum}_0(dA_1)$

**def** relu($A$):
  $Z = \max(0, A)$
  **def** back($dZ, dA$):
    $dA \mathrel{+}= dZ * (Z > 0)$
  **return** $node(Z, \text{back})$

# automatic differentiation

$A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1$
$Z = \mathrm{relu}(A_1)$
$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \mathrm{sum}_1(E)$
$C = -\mathrm{sum}_1(T * \log(Y))$
$D = \mathrm{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$dW_2 \mathrel{+}= \mathrm{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \mathrm{sum}_0(dA_2)$
$dZ = \mathrm{dot}(dA_2, W_2^\top)$
$Z.\mathrm{back}(A_1)$
$dW_1 \mathrel{+}= \mathrm{dot}(X^\top, dA_1)$
$d\mathbf{b}_1 = \mathrm{sum}_0(dA_1)$

**def** $\mathrm{affine}(X, (W, \mathbf{b}))$:
    $A = \mathrm{dot}(X, W) + \mathbf{b}$
    **def** $\mathrm{back}(dA, dX, (dW, d\mathbf{b}))$:
        $dW \mathrel{+}= \mathrm{dot}(X^\top, dA)$
        $d\mathbf{b} \mathrel{+}= \mathrm{sum}_0(dA)$
        $dX \mathrel{+}= \mathrm{dot}(dA, W^\top)$
    **return** $\mathit{node}(A, \mathrm{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$E = \exp(A_2)$
$Y = E / \text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{affine}(X, (W, \mathbf{b}))$:
  $A = \text{dot}(X, W) + \mathbf{b}$
  **def** $\text{back}(dA, dX, (dW, d\mathbf{b}))$:
    $dW \mathrel{+}= \text{dot}(X^\top, dA)$
    $d\mathbf{b} \mathrel{+}= \text{sum}_0(dA)$
    $dX \mathrel{+}= \text{dot}(dA, W^\top)$
  **return** $\textit{node}(A, \text{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/m$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/m$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{logistic}(A, T)$:
  $E = \exp(A)$
  $Y = E/\text{sum}_1(E)$
  $C = -\text{sum}_1(T * \log(Y))$
  $D = \text{sum}_0(C)/m$
  **def** $\text{back}(dD, dA, \_)$:
    $dA \mathrel{+}= dD * (Y - T)/m$
  **return** $\mathit{node}(D, \text{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$

$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{logistic}(A, T)$:
  $E = \exp(A)$
  $Y = E/\text{sum}_1(E)$
  $C = -\text{sum}_1(T * \log(Y))$
  $D = \text{sum}_0(C)/m$
  **def** $\text{back}(dD, dA, \_)$:
    $dA \mathrel{+}= dD * (Y - T)/m$
  **return** $node(D, \text{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$

$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{decay}(W)$:
  $R = \frac{\lambda}{2} * \text{sum}(\|w\|_F^2$ **for** $w$ **in** $W)$
  **def** $\text{back}(dR, dW)$:
    **for** $(w, dw)$ **in** $\text{zip}(W, dW)$:
      $dw \mathrel{+}= dR * \lambda * w$
  **return** $\textit{node}(R, \text{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$

$R = \text{decay}((W_1, W_2))$
$L = D + R$
$(dD, dR) = (dL, dL)$
$R.\text{back}((W_1, W_2))$

$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

```
def decay(W):
    R = λ/2 * sum(‖w‖_F^2 for w in W)
    def back(dR, dW):
        for (w, dw) in zip(W, dW):
            dw += dR * λ * w
    return node(R, back)
```

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$

$R = \text{decay}((W_1, W_2))$
$L = D + R$
$(dD, dR) = (dL, dL)$
$R.\text{back}((W_1, W_2))$

$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

```
def add(X):
    S = sum(X)
    def back(dS, dX):
        for dx in dX:
            dx += dS
    return node(S, back)
```

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$

$R = \text{decay}((W_1, W_2))$
$L = \text{add}((D, R))$
$L.\text{back}((D, R))$
$R.\text{back}((W_1, W_2))$

$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

```
def add(X):
    S = sum(X)
    def back(dS, dX):
        for dx in dX:
            dx += dS
    return node(S, back)
```

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{logistic}(A_2, T)$
$R = \text{decay}((W_1, W_2))$
$L = \text{add}((D, R))$
$L.\,\text{back}((D, R))$
$R.\,\text{back}((W_1, W_2))$
$D.\,\text{back}(A_2, T)$
$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{loss}(A, T, W)$:
  $D = \text{logistic}(A, T)$
  $R = \text{decay}(W)$
  $L = \text{add}((D, R))$
  **def** $\text{back}(A, T, W)$:
    $L.\,\text{back}((D, R))$
    $R.\,\text{back}(W)$
    $D.\,\text{back}(A, T)$
  **return** $\mathit{block}(L, \text{back})$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$L = \text{loss}(A_2, T, (W_1, W_2))$

**def** $\text{loss}(A, T, W)$:
    $D = \text{logistic}(A, T)$
    $R = \text{decay}(W)$
    $L = \text{add}((D, R))$
    **def** $\text{back}(A, T, W)$:
        $L.\text{back}((D, R))$
        $R.\text{back}(W)$
        $D.\text{back}(A, T)$
    **return** $\mathit{block}(L, \text{back})$

$L.\text{back}(A_2, T, (W_1, W_2))$

$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$L = \text{loss}(A_2, T, (W_1, W_2))$

**def** $\text{loss}(A, T, W)$:
  $L = \text{logistic}(A, T) + \text{decay}(W)$
  **return** $\mathit{block}(L)$

**def** $\text{loss}(A, T, W)$:
  $D = \text{logistic}(A, T)$
  $R = \text{decay}(W)$
  $L = \text{add}((D, R))$
  **def** $\text{back}(A, T, W)$:
    $L.\,\text{back}((D, R))$
    $R.\,\text{back}(W)$
    $D.\,\text{back}(A, T)$
  **return** $\mathit{block}(L, \text{back})$

$L.\,\text{back}(A_2, T, (W_1, W_2))$

$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

# automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$L = \text{loss}(A_2, T, (W_1, W_2))$

$L.\,\text{back}(A_2, T, (W_1, W_2))$

$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

**def** $\text{model}(X, (U_1, U_2))$:
  $A_1 = \text{affine}(X, U_1)$
  $Z = \text{relu}(A)$
  $A_2 = \text{affine}(Z, U_2)$
  **def** $\text{back}(X, (U_1, U_2))$:
    $A_2.\,\text{back}(Z, U_2)$
    $Z.\,\text{back}(A)$
    $A_1.\,\text{back}(X, U_1)$
  **return** $\mathit{block}(A_2, \text{back})$

# automatic differentiation

$A_2 = \text{model}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

$L = \text{loss}(A_2, T, (W_1, W_2))$

$L.\text{back}(A_2, T, (W_1, W_2))$

$A_2.\text{back}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

**def** $\text{model}(X, (U_1, U_2))$:
    $A_1 = \text{affine}(X, U_1)$
    $Z = \text{relu}(A)$
    $A_2 = \text{affine}(Z, U_2)$
**def** $\text{back}(X, (U_1, U_2))$:
    $A_2.\text{back}(Z, U_2)$
    $Z.\text{back}(A)$
    $A_1.\text{back}(X, U_1)$
**return** $\mathit{block}(A_2, \text{back})$

# automatic differentiation

$A_2 = \text{model}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

$L = \text{loss}(A_2, T, (W_1, W_2))$

$L.\,\text{back}(A_2, T, (W_1, W_2))$

$A_2.\,\text{back}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

**def** $\text{model}(X, (U_1, U_2))$:
  $A = \text{affine}(\text{relu}(\text{affine}(X, U_1)), U_2)$
  **return** $\mathit{block}(A)$

**def** $\text{model}(X, (U_1, U_2))$:
  $A_1 = \text{affine}(X, U_1)$
  $Z = \text{relu}(A)$
  $A_2 = \text{affine}(Z, U_2)$
  **def** $\text{back}(X, (U_1, U_2))$:
    $A_2.\,\text{back}(Z, U_2)$
    $Z.\,\text{back}(A)$
    $A_1.\,\text{back}(X, U_1)$
  **return** $\mathit{block}(A_2, \text{back})$

# pynet

code available at https://github.com/iavr/pynet

# deep learning software

- automatically build computational graphs and compute derivatives
- run on GPU, multiple GPU, distributed
- component (unit, layer) libraries
- pre-trained models
- community

# summary

- stochastic gradient descent and its limitations
- numerical gradient approximation
- analytical computation by decomposing and applying the chain rule
- back-propagation as dynamic programming
- chaining, splitting and sharing
- common patterns between forward and backward flow
- decomposition into units (forward) and nodes (backward)
- grouping into functions (forward) and blocks (backward)