Differential Programming

Gabriel Peyré

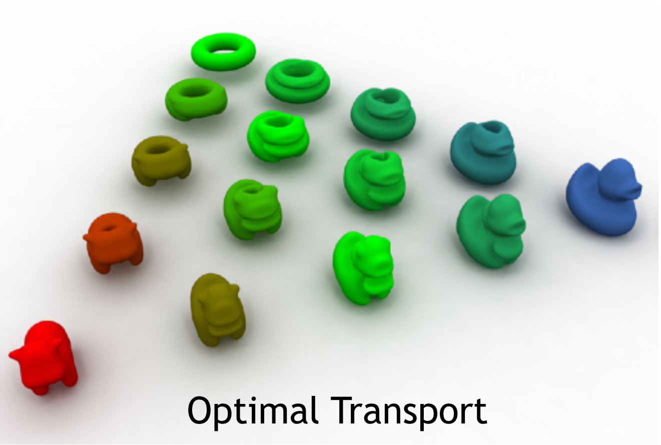
www.numerical-tours.com

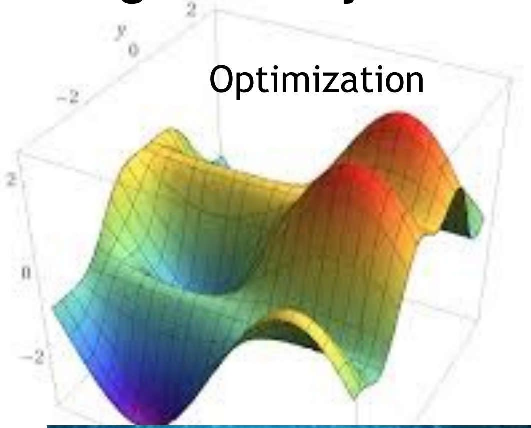
Mathematical Coffees

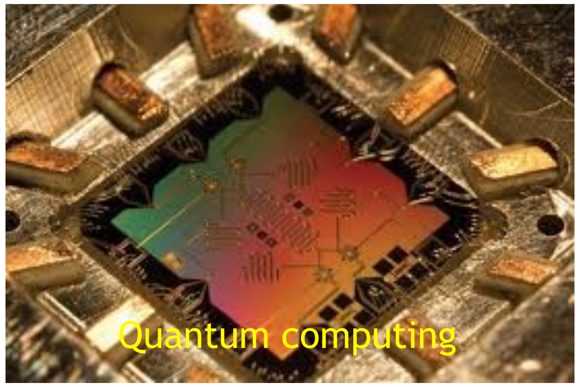
Huawei-FSMP joint seminars

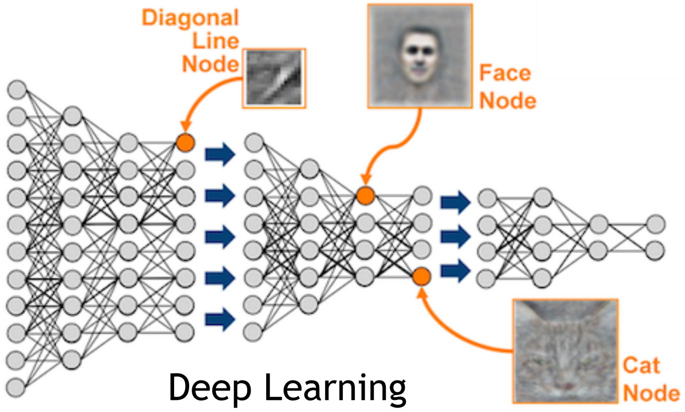
https://mathematical-coffees.github.io

Organized by: Mérouane Debbah & Gabriel Peyré

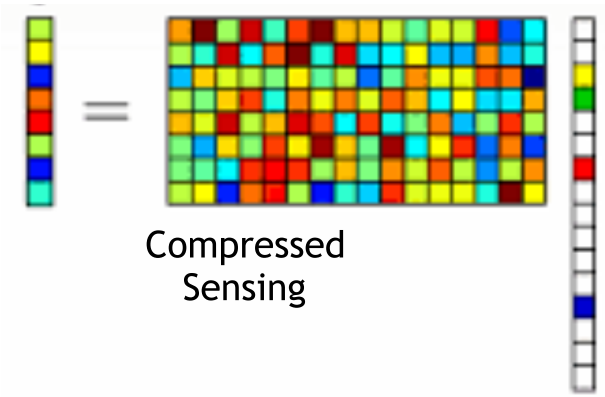

Optimal Transport


Optimization

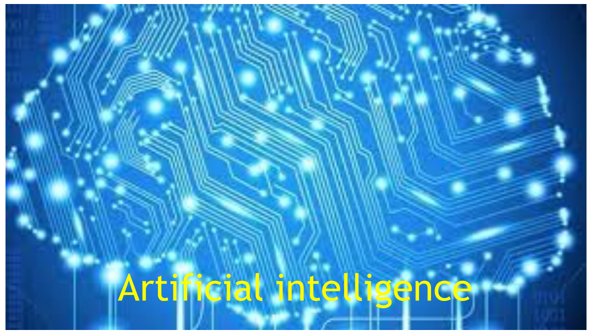

Quantum computing

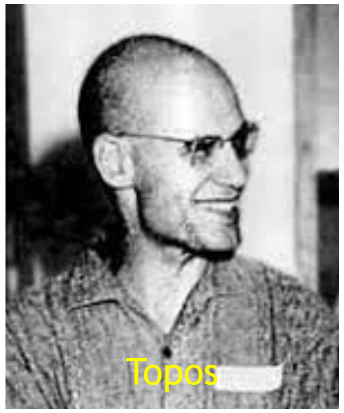


Deep Learning


Compressed Sensing

Mean field games

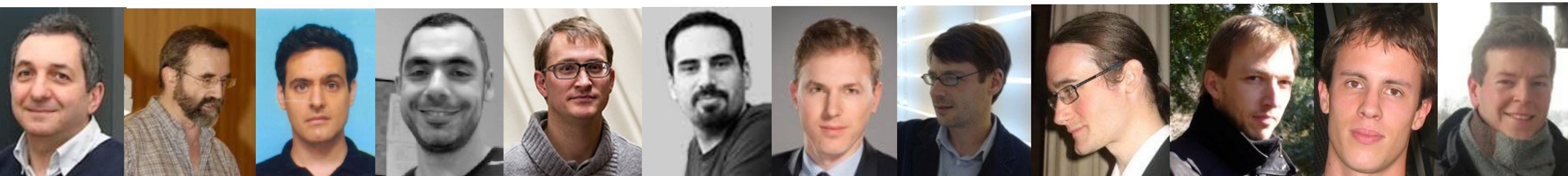

Artificial intelligence


Topos

Yves Achdou, Paris 6
Daniel Bennequin, Paris 7
Marco Cuturi, ENSAE
Jalal Fadili, ENSICaen

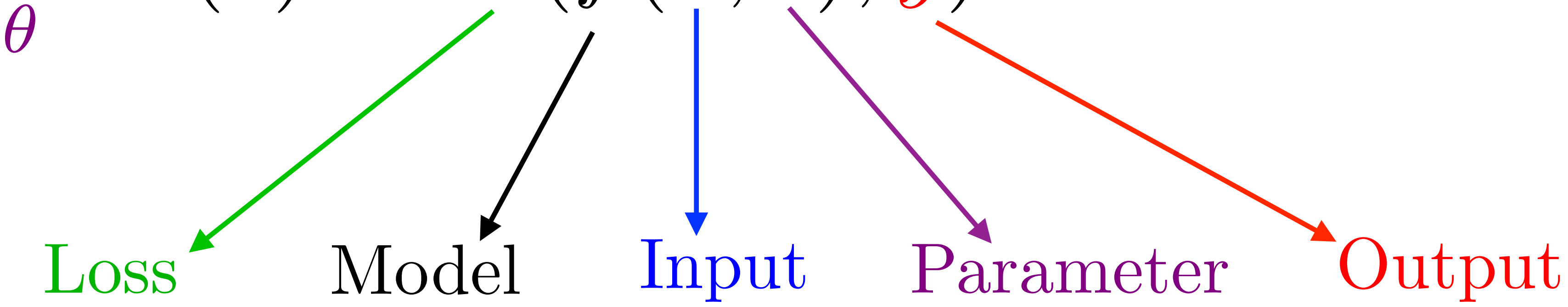
Alexandre Gramfort, INRIA
Olivier Grisel (INRIA)
Olivier Guéant, Paris 1
Iordanis Kerenidis, CNRS and Paris 7
Guillaume Lecué, CNRS and ENSAE

Frédéric Magniez, CNRS and Paris 7
Edouard Oyallon, CentraleSupelec
Gabriel Peyré, CNRS and ENS
Joris Van den Bossche (INRIA)



Model Fitting in Data Sciences

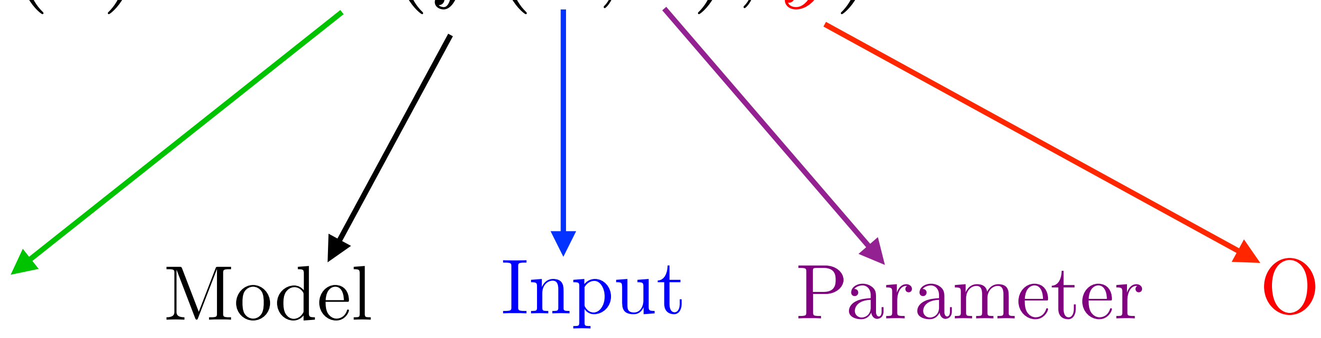
$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$



Loss Model Input Parameter Output

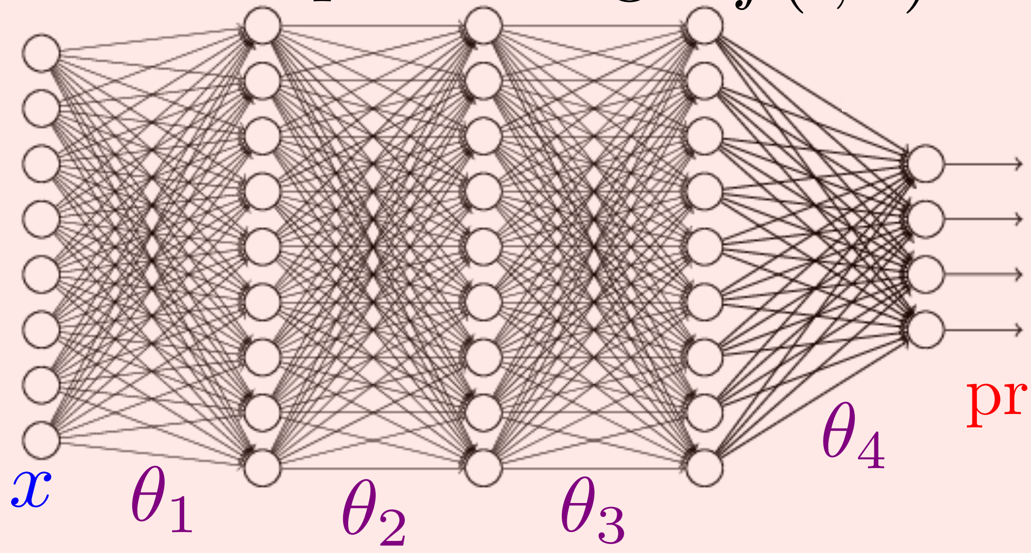
Model Fitting in Data Sciences

$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$



Loss Model Input Parameter Output

Deep-learning: $f(\cdot,\theta)$

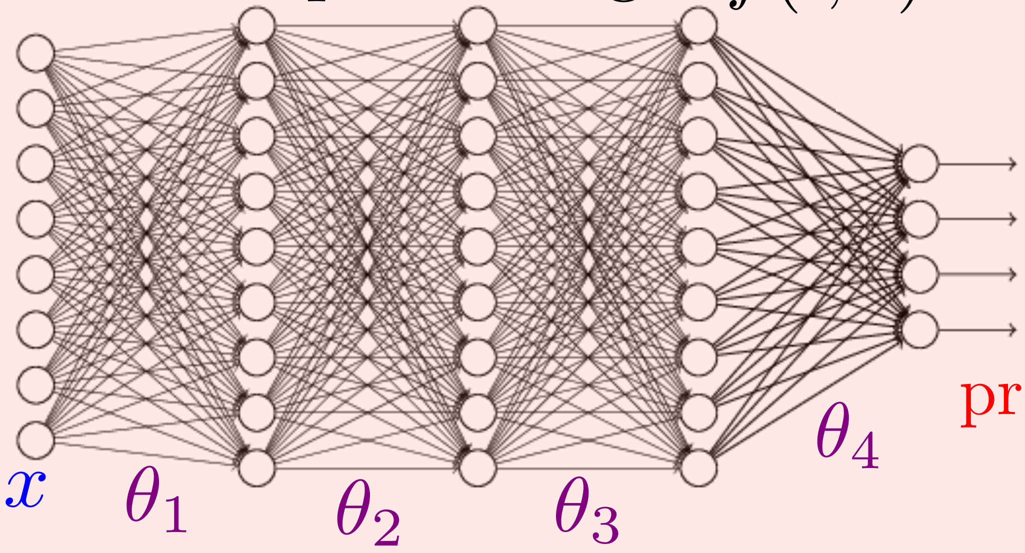


Model Fitting in Data Sciences

$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$

Loss Model Input Parameter Output

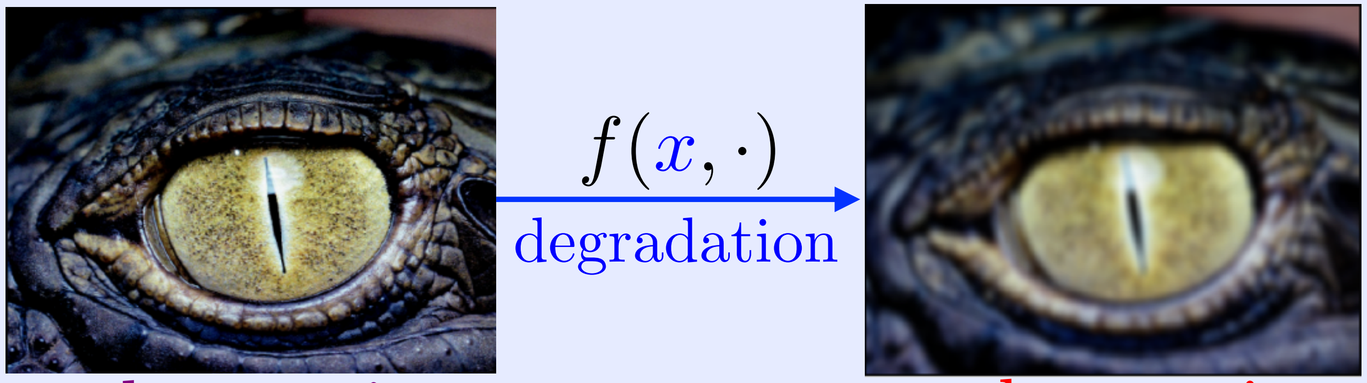
Deep-learning: $f(\cdot,\theta)$



y

class probabilities

Super-resolution:



θ unknown image

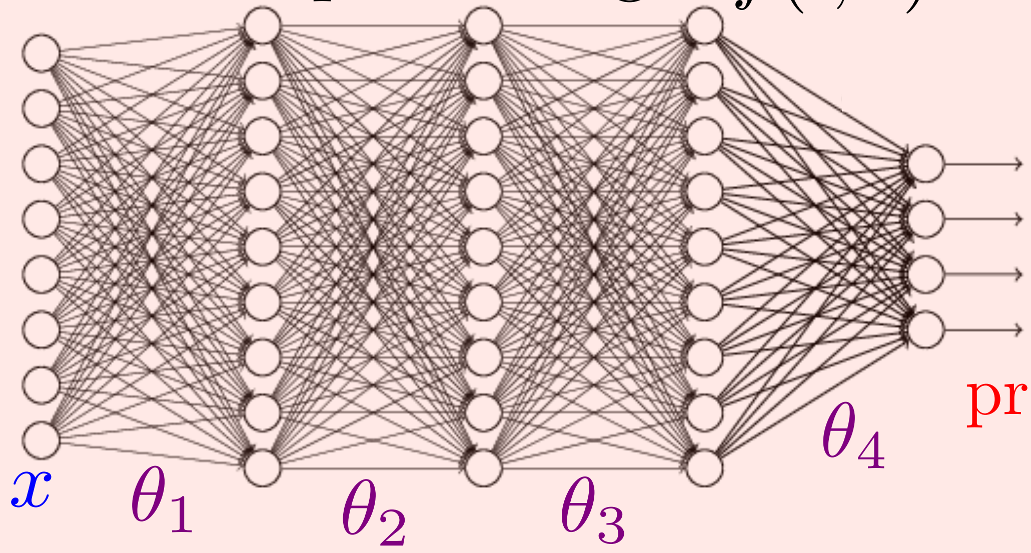
y observation

Model Fitting in Data Sciences

$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$

Loss Model Input Parameter Output

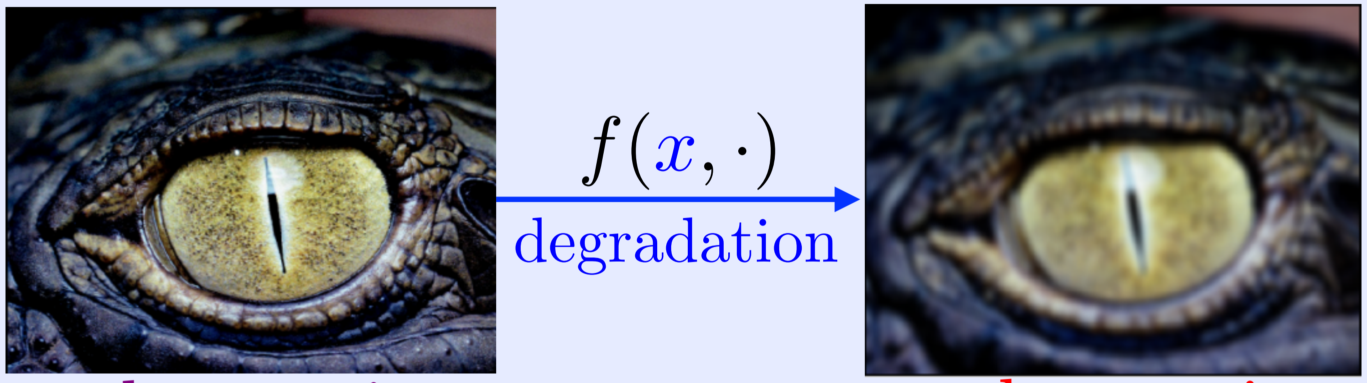
Deep-learning: $f(\cdot,\theta)$



y

class probabilities

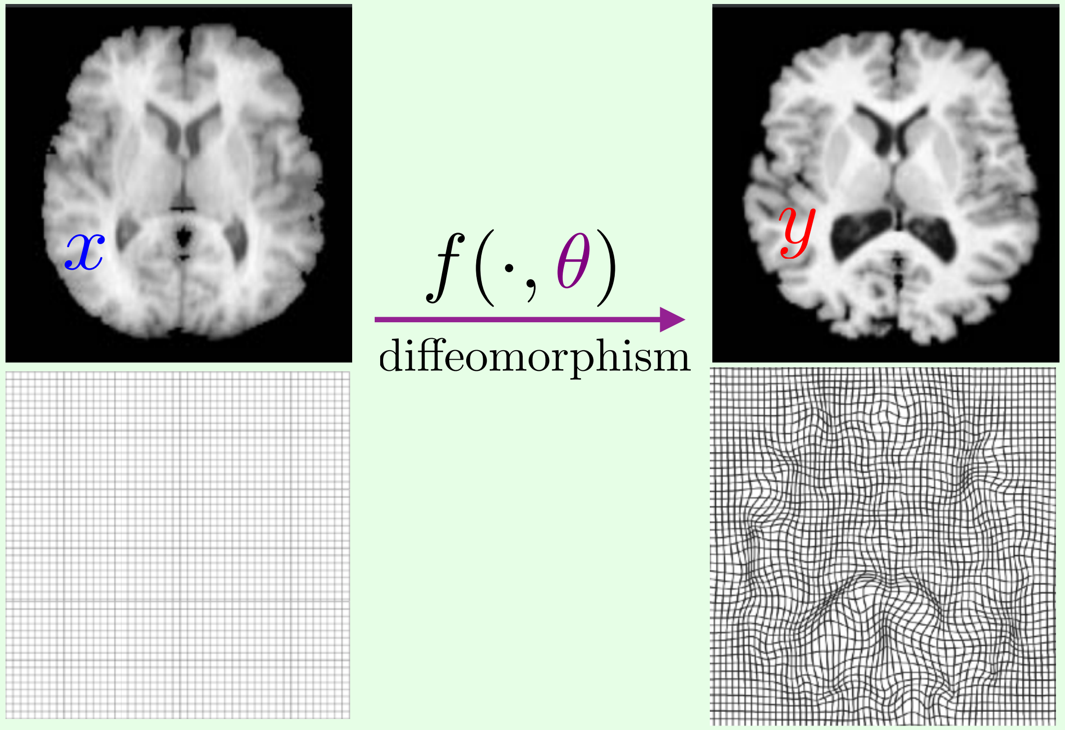
Super-resolution:



θ unknown image

y observation

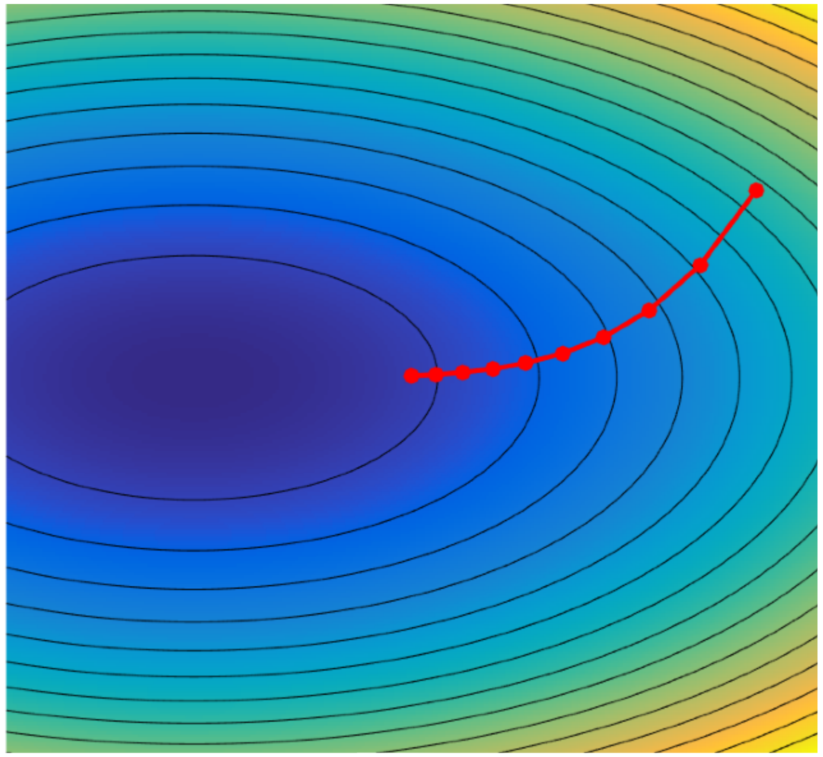
Medical imaging registration:



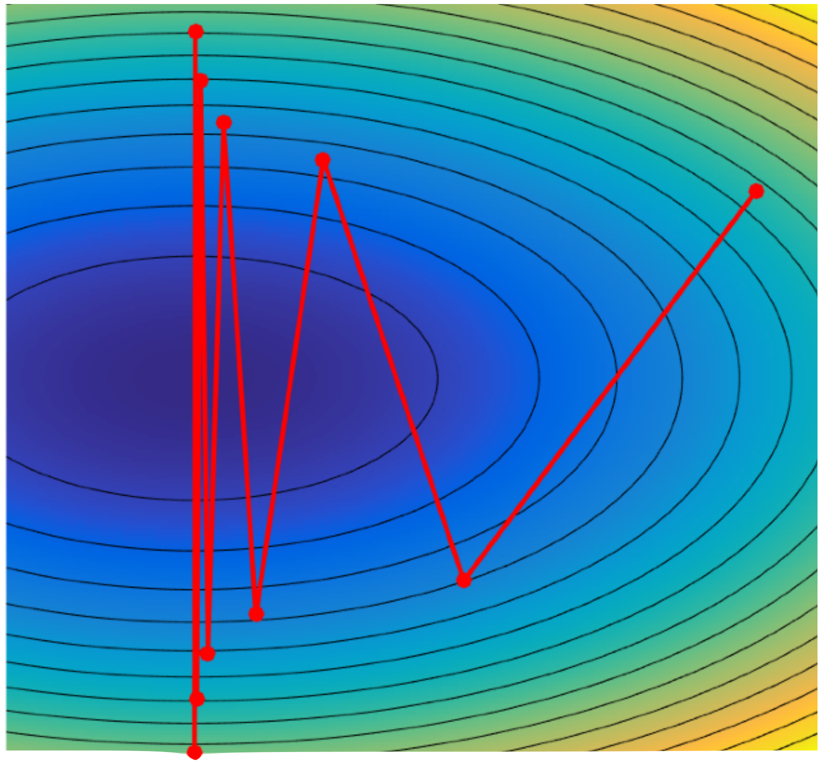
Gradient-based Methods

$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$

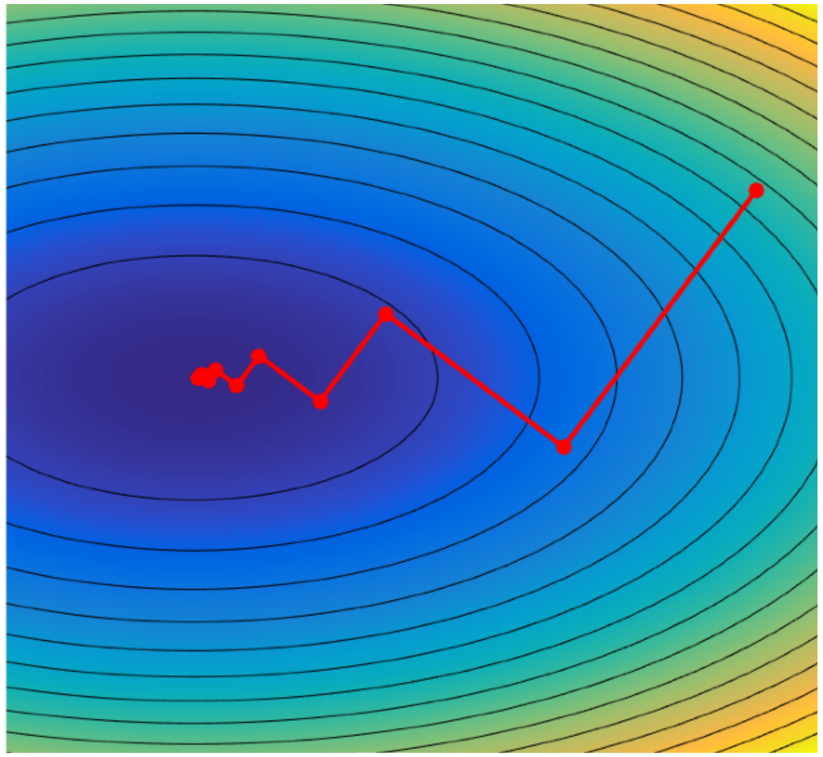
Gradient descent: $\theta_{\ell+1} = \theta_\ell - \tau_\ell \nabla \mathcal{E}(\theta_\ell)$



Small τ_ℓ



Large τ_ℓ

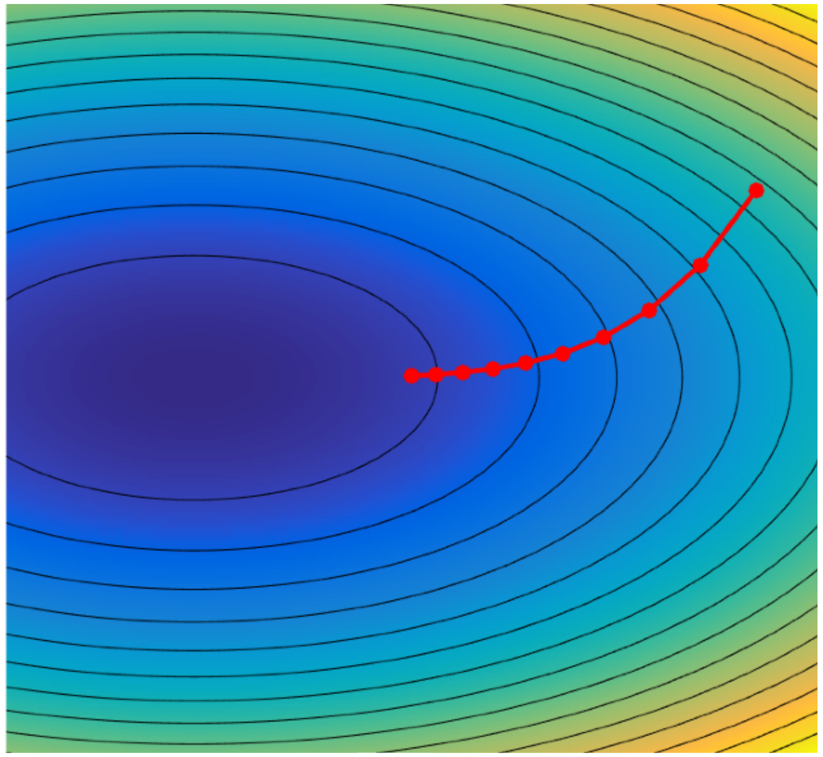


Optimal $\tau_\ell = \tau_\ell^\star$

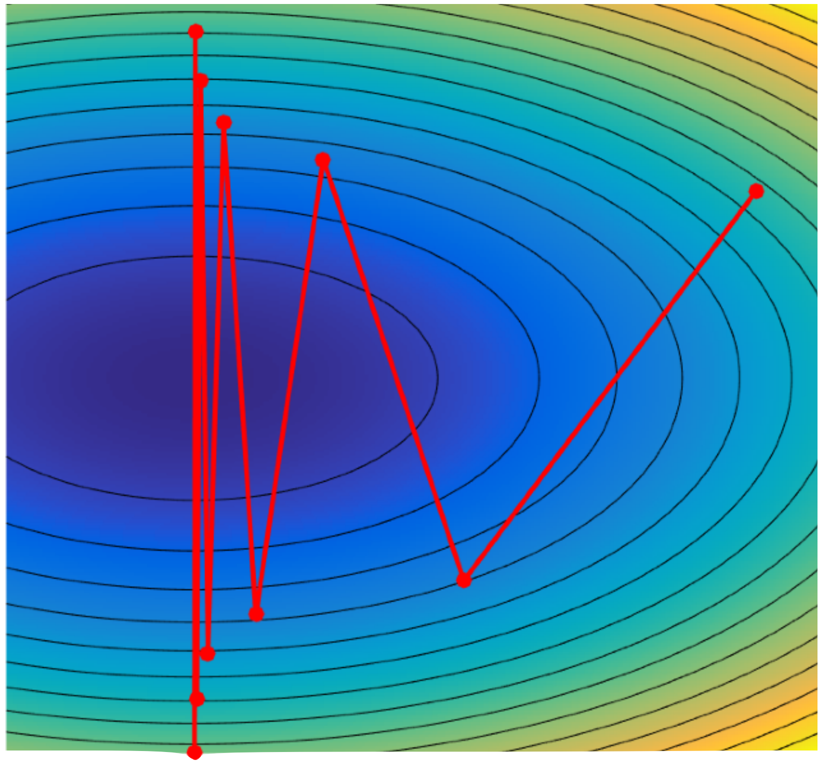
Gradient-based Methods

$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$$

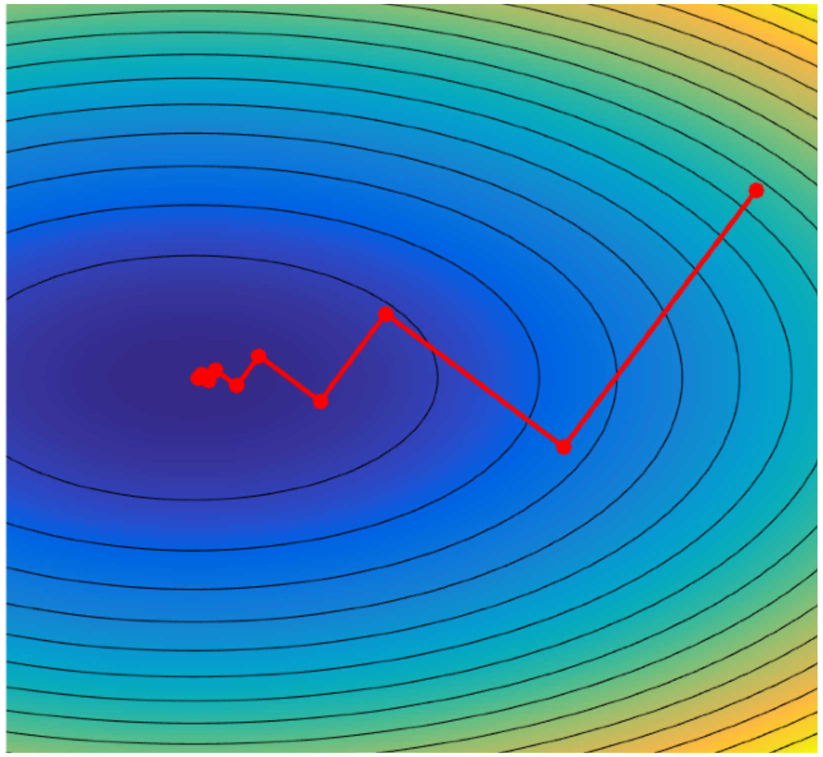
Gradient descent: $\theta_{\ell+1} = \theta_\ell - \tau_\ell \nabla \mathcal{E}(\theta_\ell)$



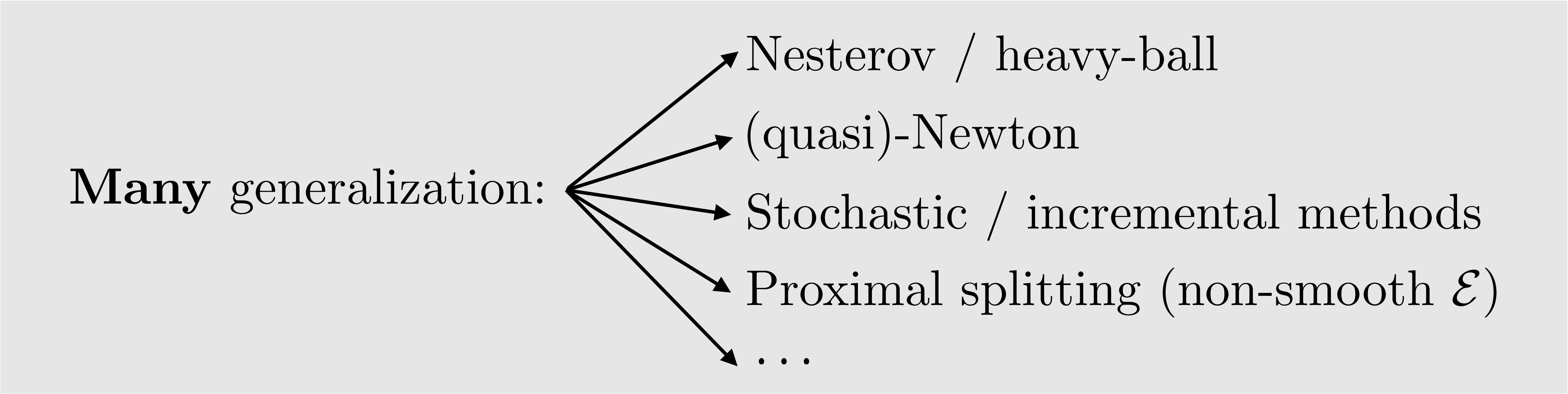
Small τ_ℓ



Large τ_ℓ



Optimal $\tau_\ell = \tau_\ell^\star$

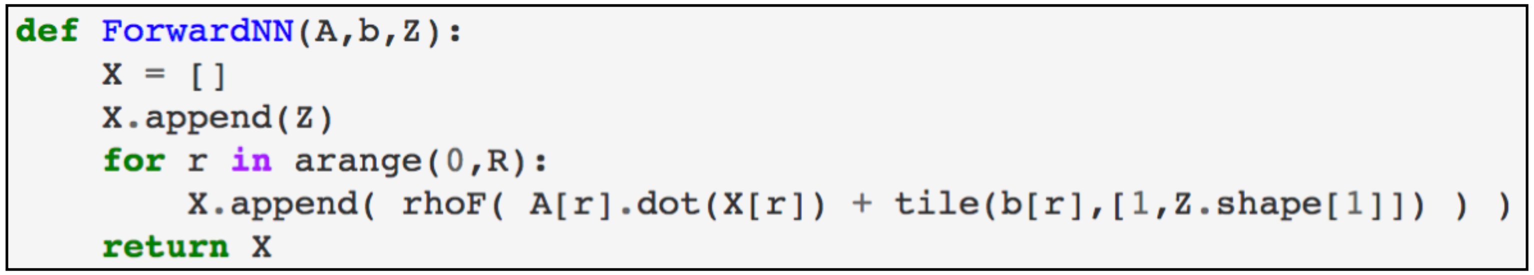


Many generalization:

- Nesterov / heavy-ball
- (quasi)-Newton
- Stochastic / incremental methods
- Proximal splitting (non-smooth $\mathcal{E}$)
- $\ldots$

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.



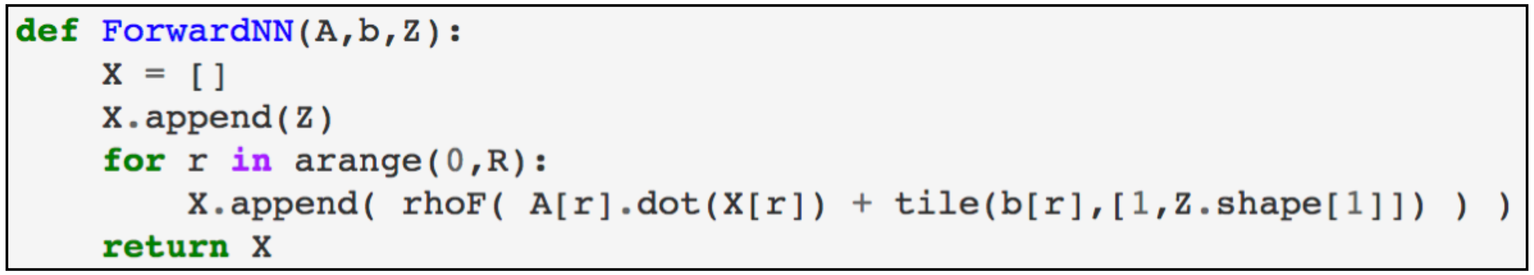
```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$

and their derivatives cost $O(1)$.

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.



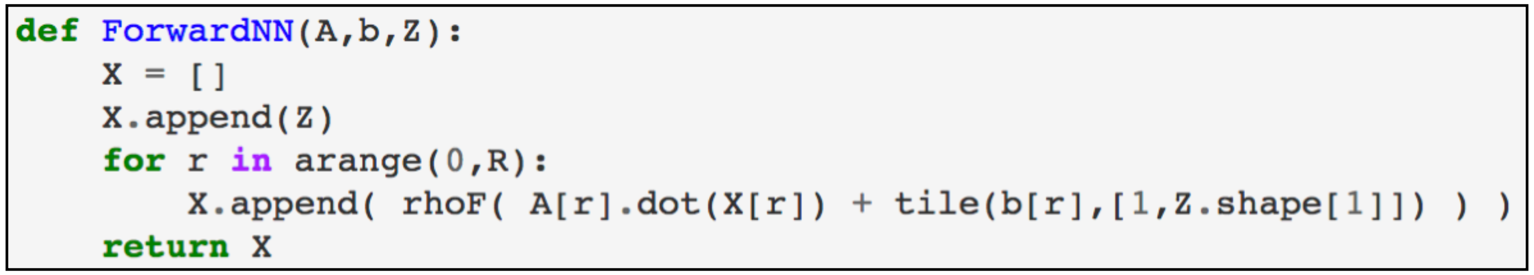
```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$
and their derivatives cost $O(1)$.

Question: What is the complexity of computing $\nabla\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^n$?

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.



```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$

and their derivatives cost $O(1)$.

Question: What is the complexity of computing $\nabla \mathcal{E} : \mathbb{R}^n \to \mathbb{R}^n$?

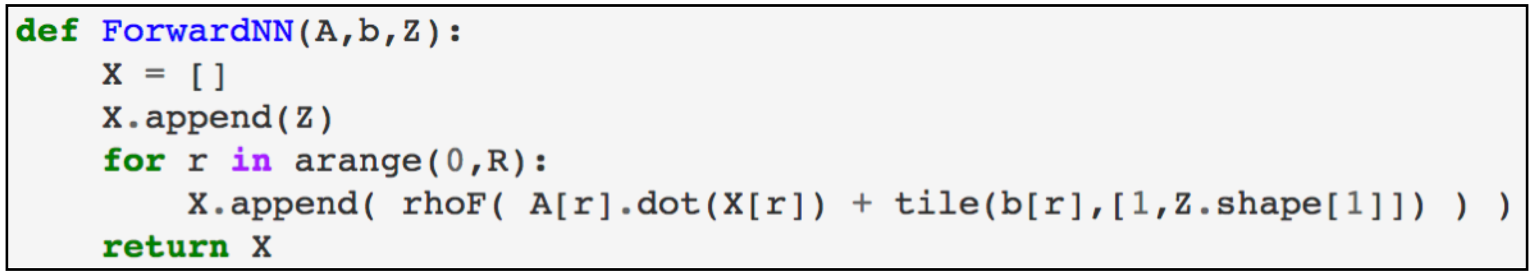
Finite differences:

$$\nabla \mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_n) - \mathcal{E}(\theta))$$

$K(n+1)$ operations, intractable for large n.

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.



```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$ and their derivatives cost $O(1)$.

Question: What is the complexity of computing $\nabla\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^n$?

Finite differences:

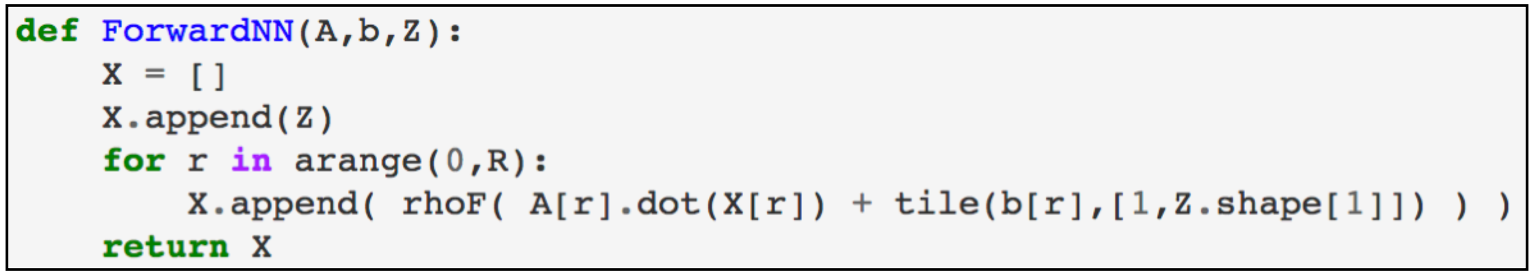
$$\nabla\mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_n) - \mathcal{E}(\theta))$$

$K(n+1)$ operations, intractable for large n.

Theorem: there is an algorithm to compute $\nabla\mathcal{E}$ in $O(K)$ operations.
[Seppo Linnainmaa, 1970]

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.



```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

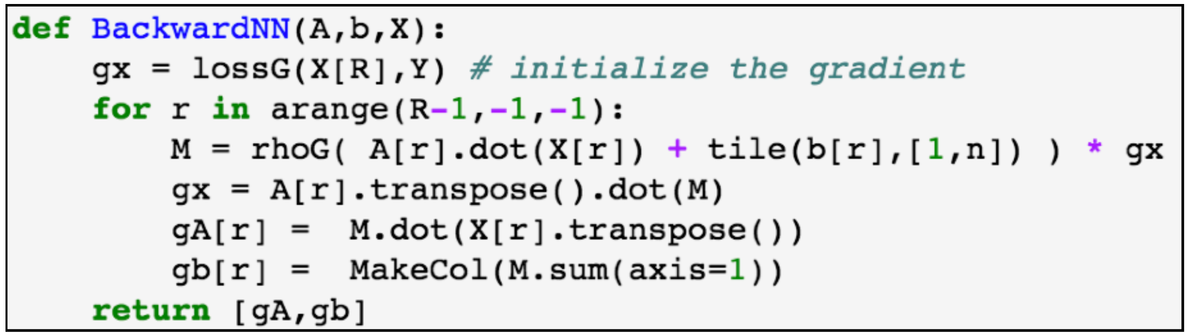
Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$ and their derivatives cost $O(1)$.

Question: What is the complexity of computing $\nabla\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^n$?

Finite differences:
$$\nabla\mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_n) - \mathcal{E}(\theta))$$
$K(n+1)$ operations, intractable for large n.

Theorem: there is an algorithm to compute $\nabla\mathcal{E}$ in $O(K)$ operations.
[Seppo Linnainmaa, 1970]

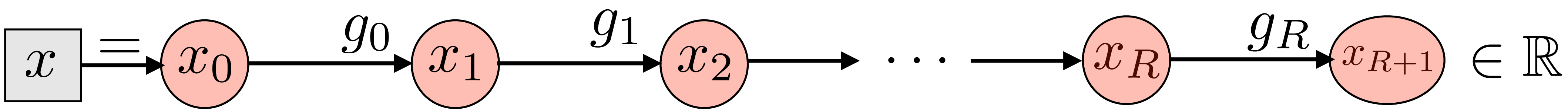
This algorithm is reverse mode automatic differentiation



```
def BackwardNN(A,b,X):
    gx = lossG(X[R],Y) # initialize the gradient
    for r in arange(R-1,-1,-1):
        M = rhoG( A[r].dot(X[r]) + tile(b[r],[1,n]) ) * gx
        gx = A[r].transpose().dot(M)
        gA[r] =  M.dot(X[r].transpose())
        gb[r] =  MakeCol(M.sum(axis=1))
    return [gA,gb]
```

Seppo Linnainmaa

Differentiating Composition of Functions

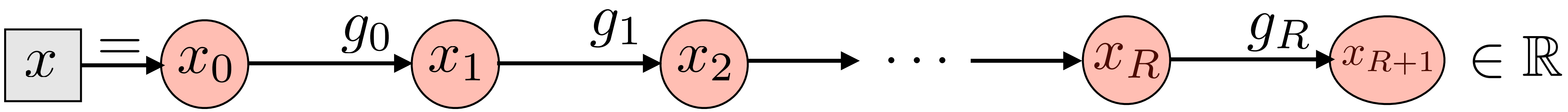


$$\partial g_r(x_r) \in \mathbb{R}^{n_{r+1}\times n_r}$$

$$x_{r+1} = g_r(x_r) \qquad g_r : \mathbb{R}^{n_r} \to \mathbb{R}^{n_{r+1}}$$

$$\nabla g_R(x_r) = [\partial g_r(x_r)]^\top \in \mathbb{R}^{n_{r+1}\times 1}$$

Differentiating Composition of Functions



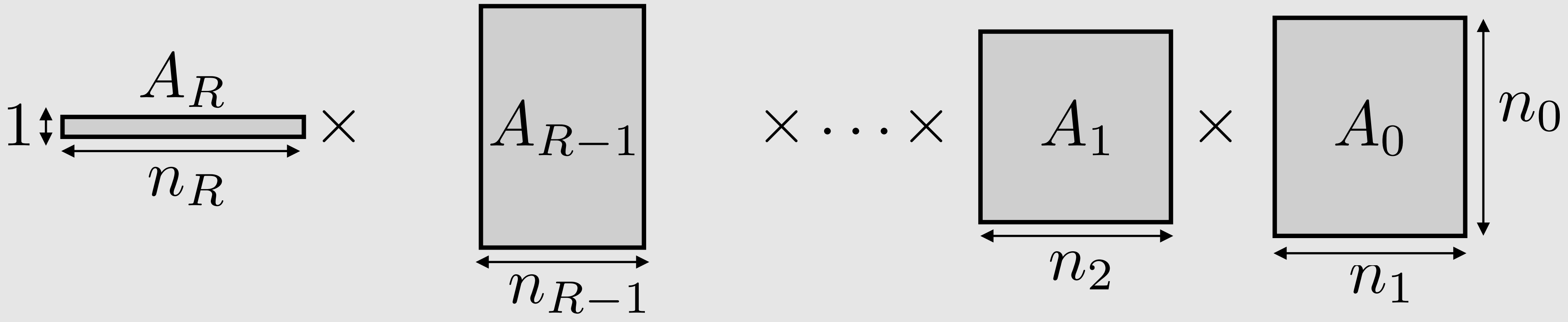
$$\partial g_r(x_r) \in \mathbb{R}^{n_{r+1} \times n_r}$$

$$x_{r+1} = g_r(x_r) \qquad g_r : \mathbb{R}^{n_r} \to \mathbb{R}^{n_{r+1}}$$

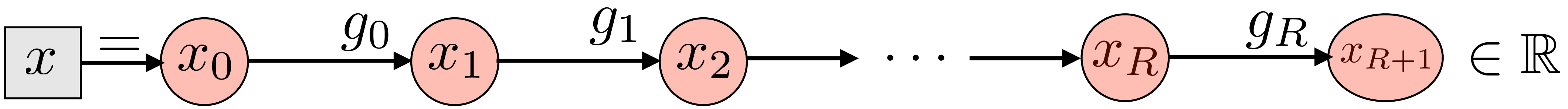
$$\nabla g_R(x_r) = [\partial g_r(x_r)]^\top \in \mathbb{R}^{n_{r+1} \times 1}$$

Chain rule:

$$\partial g(x) = \partial g_R(x_R) \times \partial g_{R-1}(x_{R-1}) \times \ldots \times \partial g_1(x_1) \times \partial g_0(x_0)$$



Differentiating Composition of Functions



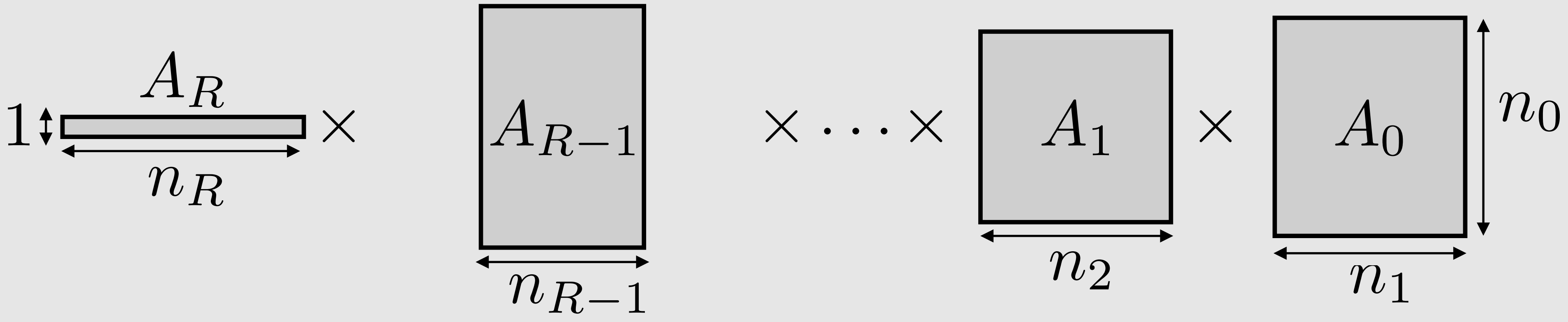
$$\partial g_r(x_r) \in \mathbb{R}^{n_{r+1} \times n_r}$$

$$x_{r+1} = g_r(x_r) \qquad g_r : \mathbb{R}^{n_r} \to \mathbb{R}^{n_{r+1}}$$

$$\nabla g_R(x_r) = [\partial g_r(x_r)]^\top \in \mathbb{R}^{n_{r+1} \times 1}$$

Chain rule:

$$\partial g(x) = \partial g_R(x_R) \times \partial g_{R-1}(x_{R-1}) \times \ldots \times \partial g_1(x_1) \times \partial g_0(x_0)$$



Forward

$O(n^3)$

$$\partial g(x) = ((\ldots(\underbrace{(\underbrace{A_0 \times A_1}_{n_0 n_1 n_2}) \times A_2}_{n_1 n_2 n_3})\ldots \times \underbrace{A_{R-2}) \times A_{R-1}}_{n_{R-2} n_{R-1} n_R}) \times A_R$$

$$\underbrace{A_{R-1}) \times A_R}_{n_{R-1} n_R}$$

Complexity: (if $n_r = 1$ for $r = 0, \ldots, R-1$) $(R-1)n^3 + n^2$

Differentiating Composition of Functions

$$\begin{tikzcd} \boxed{x} \arrow[r, "="] & x_0 \arrow[r, "g_0"] & x_1 \arrow[r, "g_1"] & x_2 \arrow[r] & \cdots \arrow[r] & x_R \arrow[r, "g_R"] & x_{R+1} \in \mathbb{R} \end{tikzcd}$$

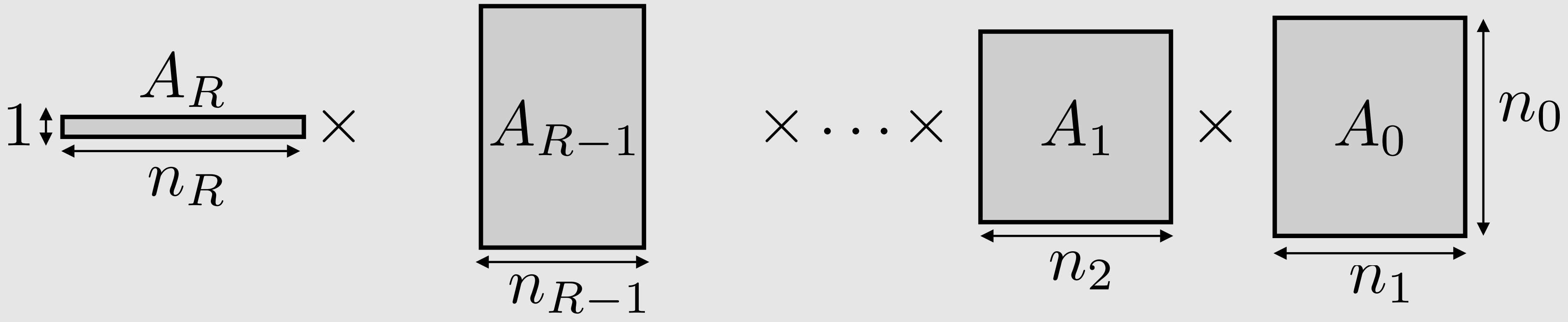
$$x_{r+1} = g_r(x_r) \qquad g_r : \mathbb{R}^{n_r} \to \mathbb{R}^{n_{r+1}}$$

$$\partial g_r(x_r) \in \mathbb{R}^{n_{r+1} \times n_r}$$

$$\nabla g_R(x_r) = [\partial g_r(x_r)]^\top \in \mathbb{R}^{n_{r+1} \times 1}$$

Chain rule:

$$\partial g(x) = \partial g_R(x_R) \times \partial g_{R-1}(x_{R-1}) \times \ldots \times \partial g_1(x_1) \times \partial g_0(x_0)$$



Forward
$O(n^3)$

$$\partial g(x) = ((\ldots(\underbrace{\underbrace{(A_0 \times A_1)}_{n_0 n_1 n_2} \times A_2)}_{n_1 n_2 n_3} \ldots \times \underbrace{\underbrace{A_{R-2}) \times A_{R-1}}_{n_{R-2} n_{R-1} n_R}) \times A_R}_{n_{R-1} n_R}$$

Complexity: (if $n_r = 1$ for $r = 0, \ldots, R-1$) $(R-1)n^3 + n^2$

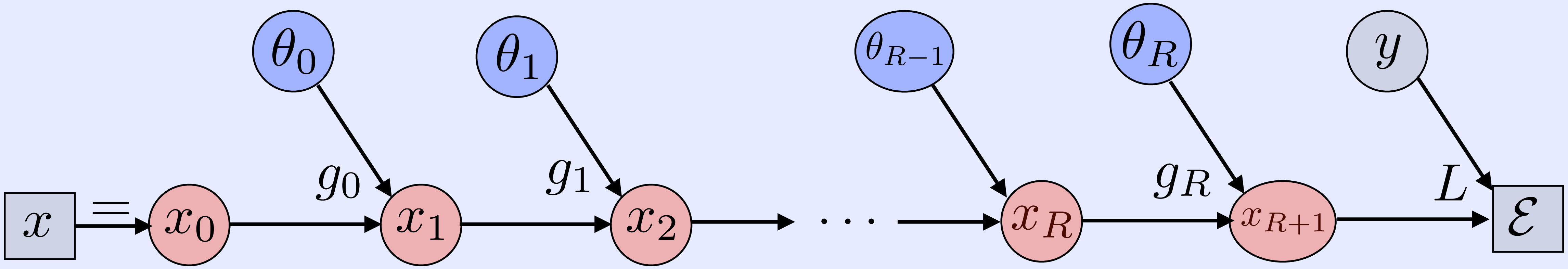
Backward
$O(n^2)$

$$\partial g(x) = \underbrace{A_0 \times (\underbrace{A_1 \times (A_2}_{n_1 n_2}}_{n_0 n_1} \times \ldots \times \underbrace{(A_{R-2} \times (\underbrace{A_{R-1} \times A_R}_{n_{R-1} n_R}}_{n_{R-2} n_{R-1}})) \ldots))$$

Complexity: Rn^2

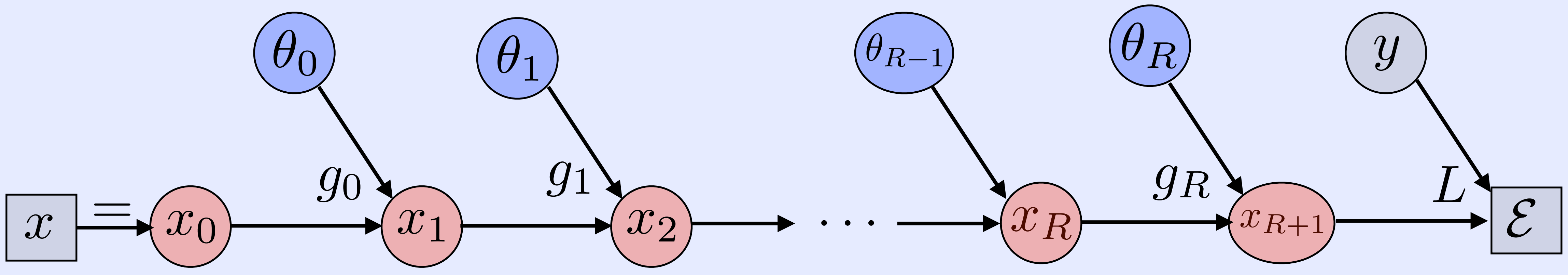
Feedfordward Computational Graphs

$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$

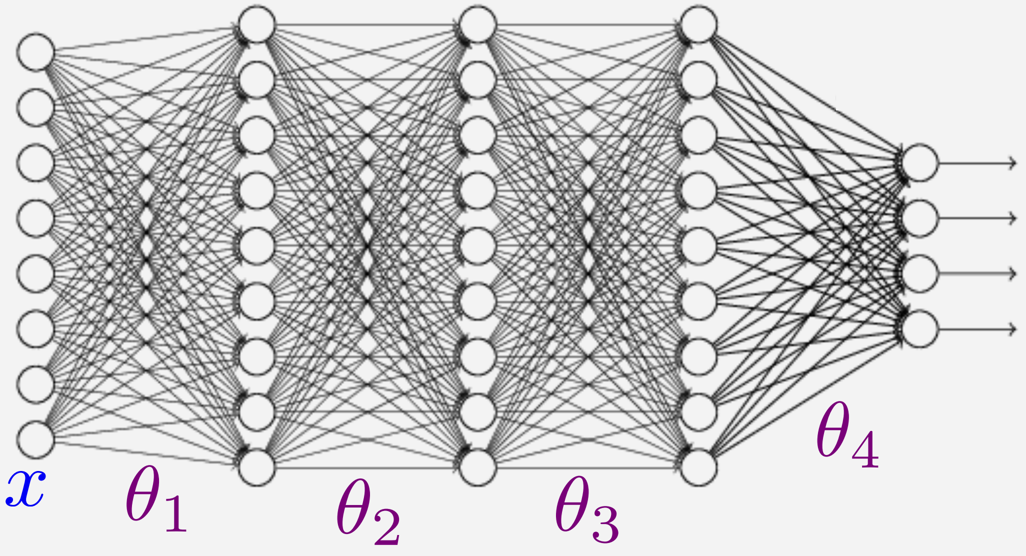


Feedforward Computational Graphs

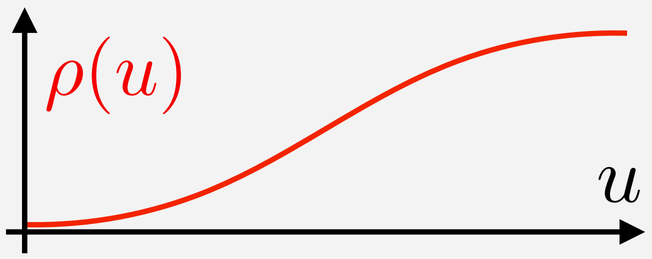
$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Example: deep neural network (here fully connected)



$$x_{r+1} = \rho(A_r x_r + b_r)$$



$$\theta_r = (A_r, b_r)$$

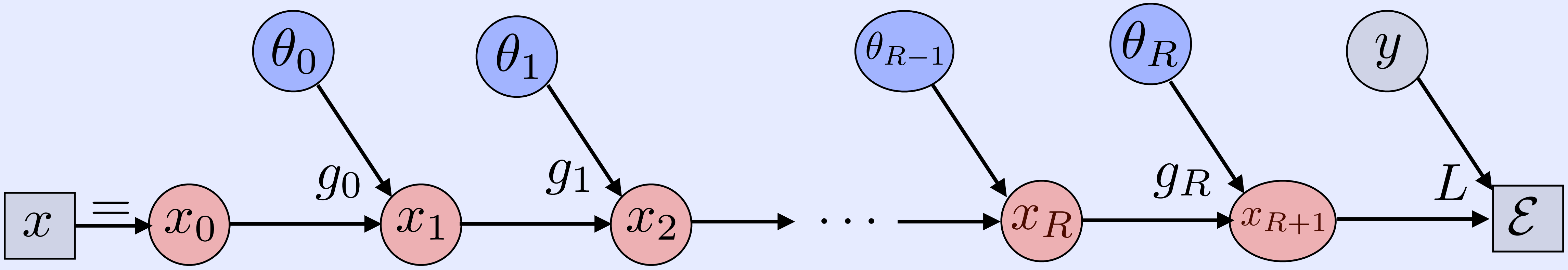
$$x_r \in \mathbb{R}^{d_r}$$

$$A_r \in \mathbb{R}^{d_{r+1} \times d_r}$$

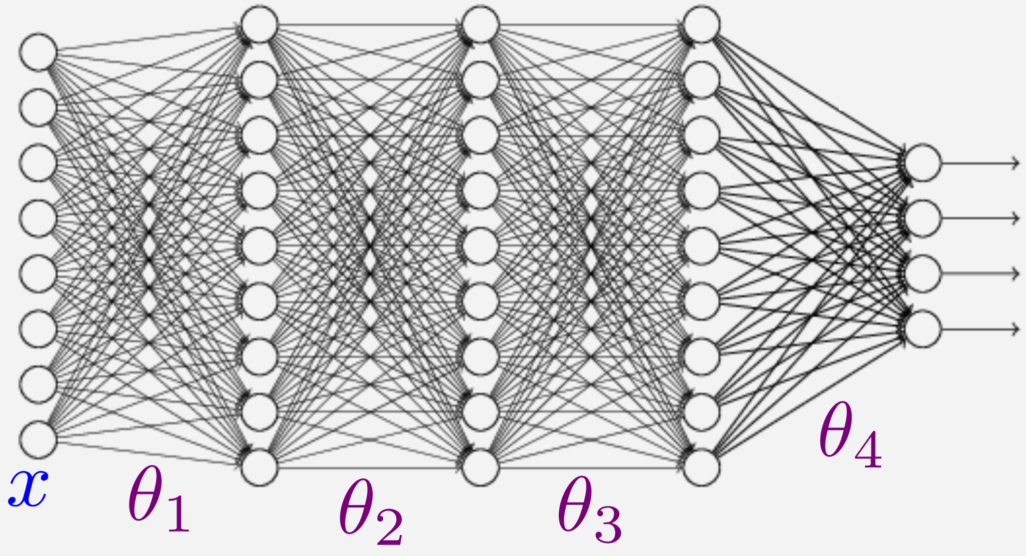
$$b_r \in \mathbb{R}^{d_{r+1}}$$

Feedfordward Computational Graphs

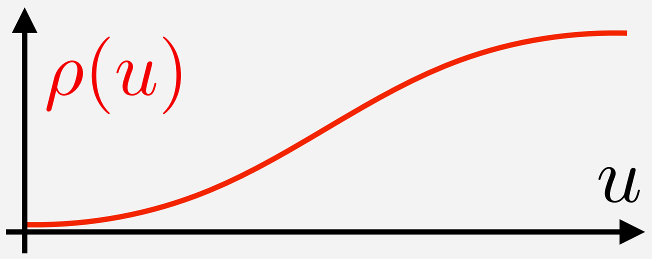
$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Example: deep neural network (here fully connected)



$$x_{r+1} = \rho(A_r x_r + b_r)$$



$$\theta_r = (A_r, b_r)$$
$$x_r \in \mathbb{R}^{d_r}$$
$$A_r \in \mathbb{R}^{d_{r+1} \times d_r}$$
$$b_r \in \mathbb{R}^{d_{r+1}}$$

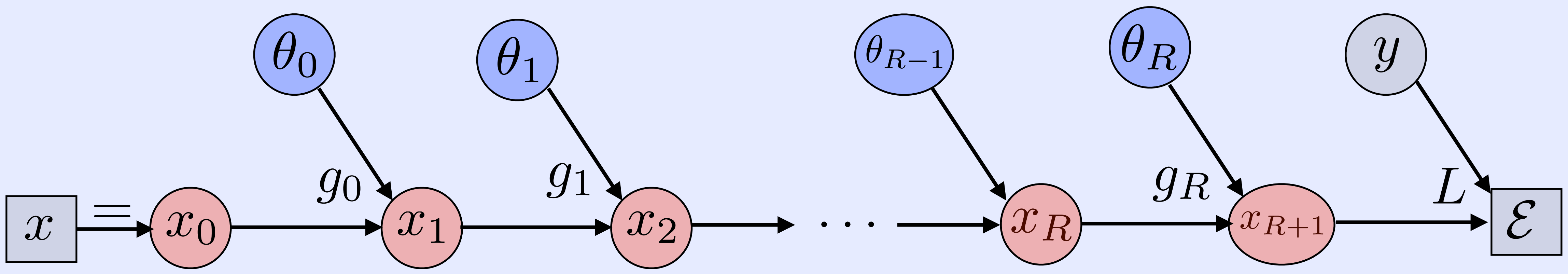
Logistic loss:
(classification)

$$L(x_{R+1}, y) \overset{\text{def.}}{=} \log \sum_i \exp(x_{R+1,i}) - x_{R+1,i} y_i$$

$$\nabla_{x_{R+1}} L(x_{R+1}, y) = \frac{e^{x_{R+1}}}{\sum_i e^{x_{R+1,i}}} - y$$

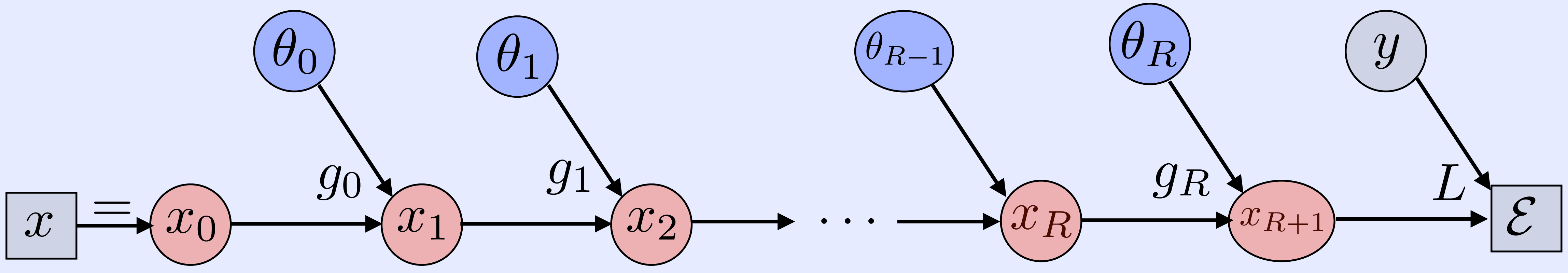
Backpropagation Algorithm

$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Backpropagation Algorithm

$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



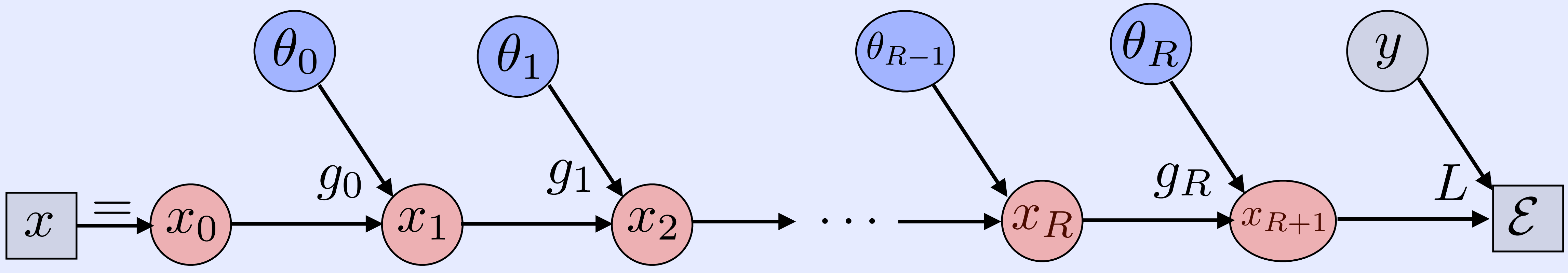
Proposition: $\forall r = R, \ldots, 0,$

$$\nabla_{x_r}\mathcal{E} = [\partial_{x_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$$

$$\nabla_{\theta_r}\mathcal{E} = [\partial_{\theta_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$$

Backpropagation Algorithm

$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$

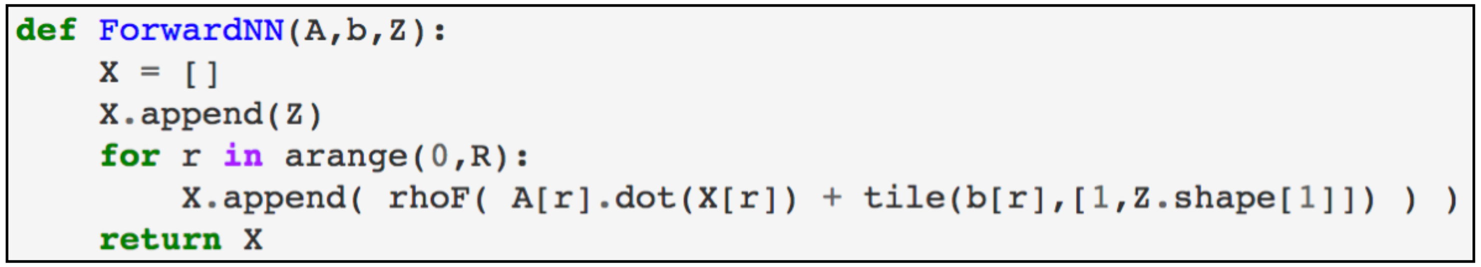


Proposition: $\forall r = R, \ldots, 0, \quad \nabla_{x_r}\mathcal{E} = [\partial_{x_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$

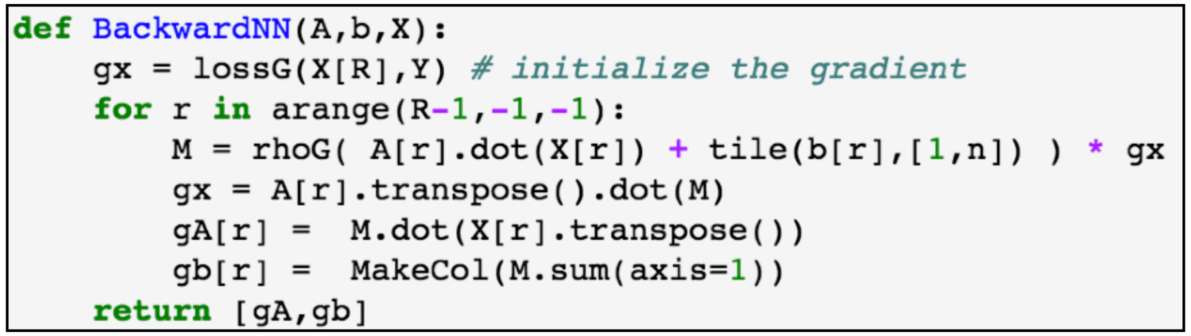
$$\nabla_{\theta_r}\mathcal{E} = [\partial_{\theta_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$$

Example: deep neural network $\quad x_{r+1} = \rho(A_r x_r + b_r)$

$$\forall r = R, \ldots, 0, \qquad \begin{aligned} \nabla_{x_r}\mathcal{E} &= A_r^\top M_r \\ \nabla_{A_r}\mathcal{E} &= M_r x_r^\top \\ \nabla_{b_r}\mathcal{E} &= M_r \mathbb{1} \end{aligned} \qquad M_r \overset{\text{def.}}{=} \rho'(A_r x_r + b_r) \odot \nabla_{x_{r+1}}\mathcal{E}$$



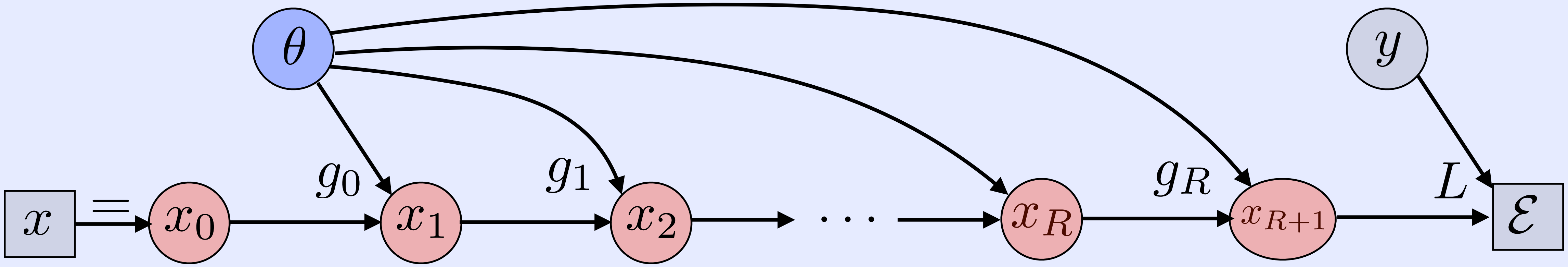
```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```



```
def BackwardNN(A,b,X):
    gx = lossG(X[R],Y) # initialize the gradient
    for r in arange(R-1,-1,-1):
        M = rhoG( A[r].dot(X[r]) + tile(b[r],[1,n]) ) * gx
        gx = A[r].transpose().dot(M)
        gA[r] =  M.dot(X[r].transpose())
        gb[r] =  MakeCol(M.sum(axis=1))
    return [gA,gb]
```

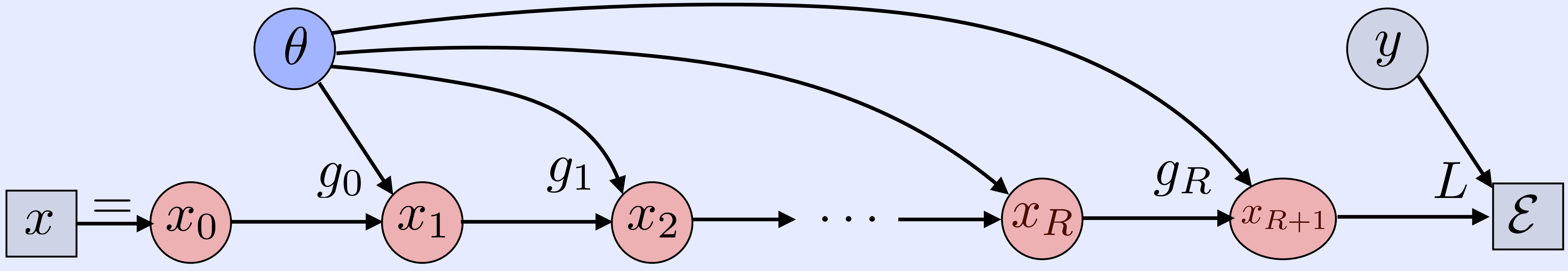
Recurrent Architectures

Shared parameters: $x_{r+1} = g_r(x_r, \theta)$

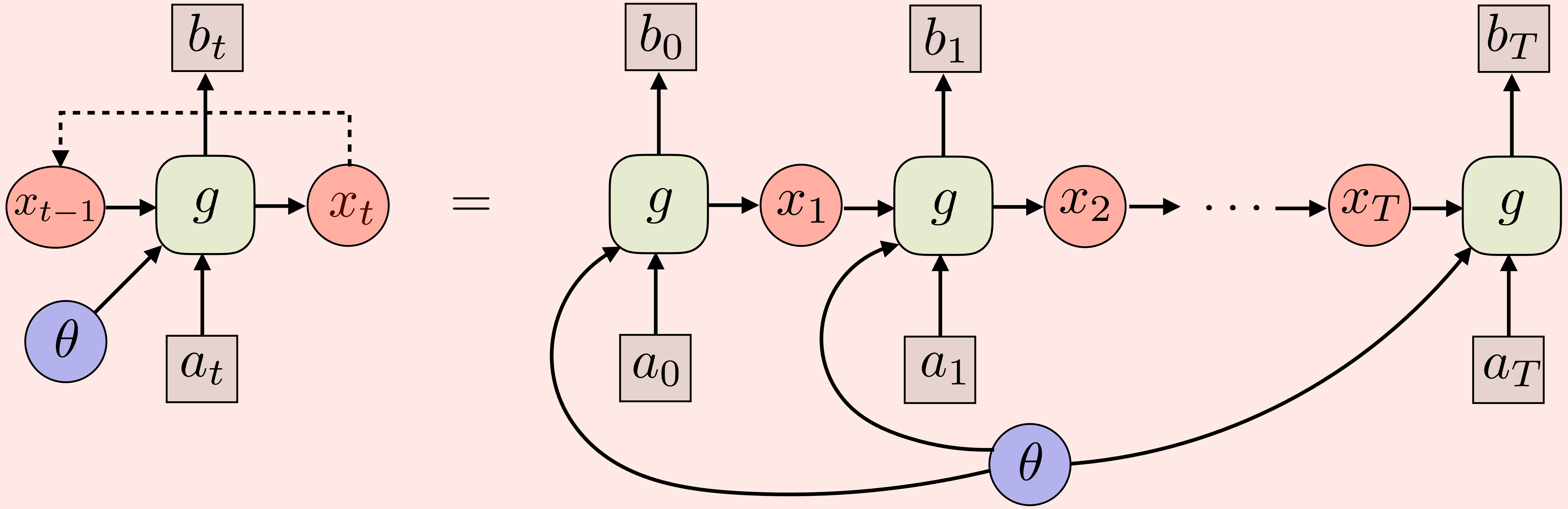


Recurrent Architectures

Shared parameters: $x_{r+1} = g_r(x_r, \theta)$

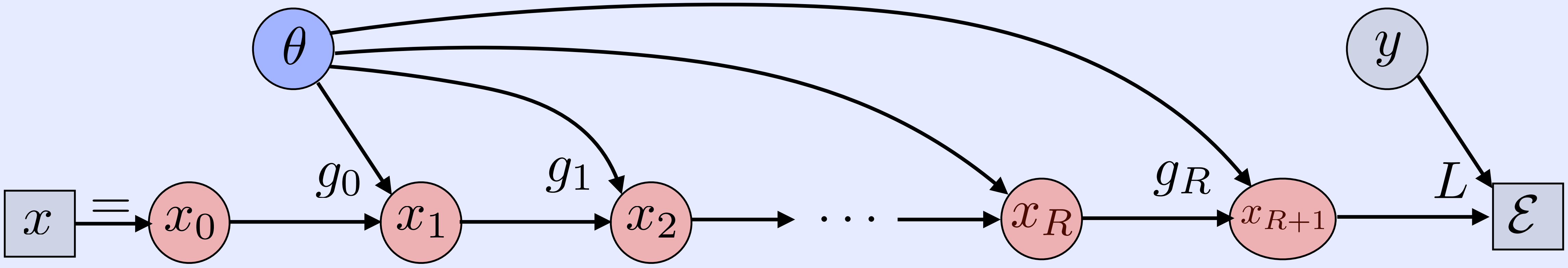


Recurrent networks for natural language processing:

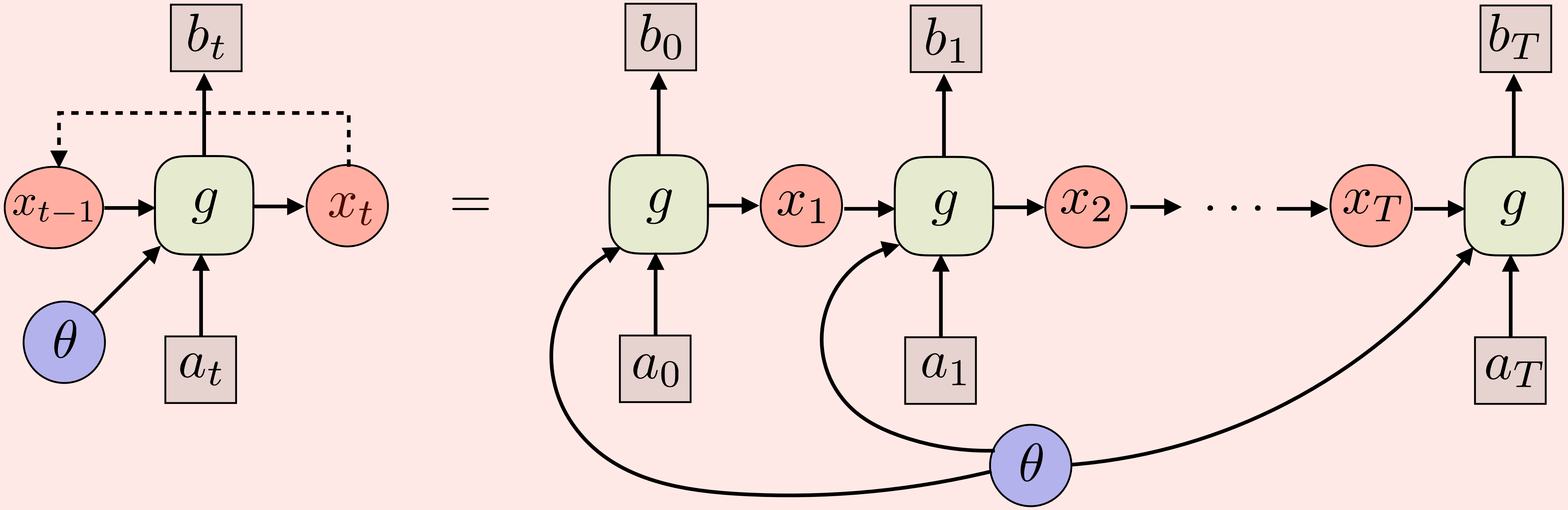


Recurrent Architectures

Shared parameters: $x_{r+1} = g_r(x_r, \theta)$



Recurrent networks for natural language processing:



Take home message: for complicated computational architectures, you do not want to do the computation/implementation by hand.

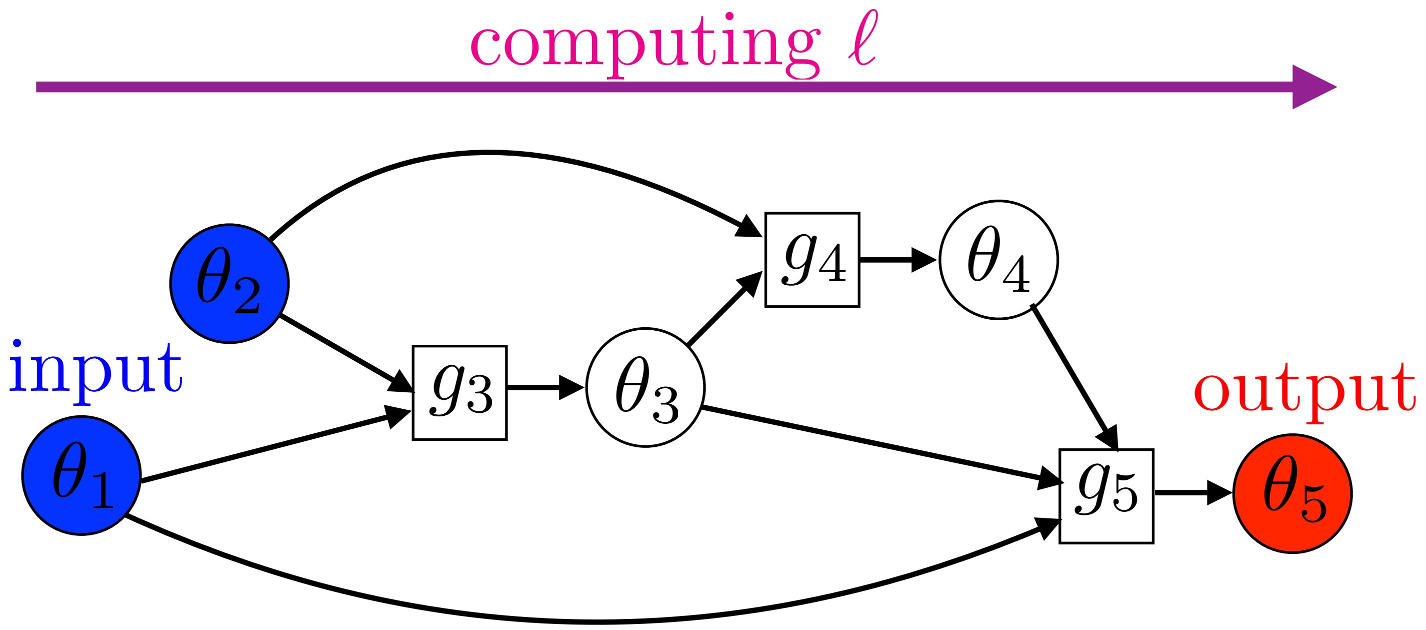
Computational Graph

Computational Graph

Computer program ⇔ directed acyclic graph ⇔ linear ordering of nodes $(\theta_r)_r$

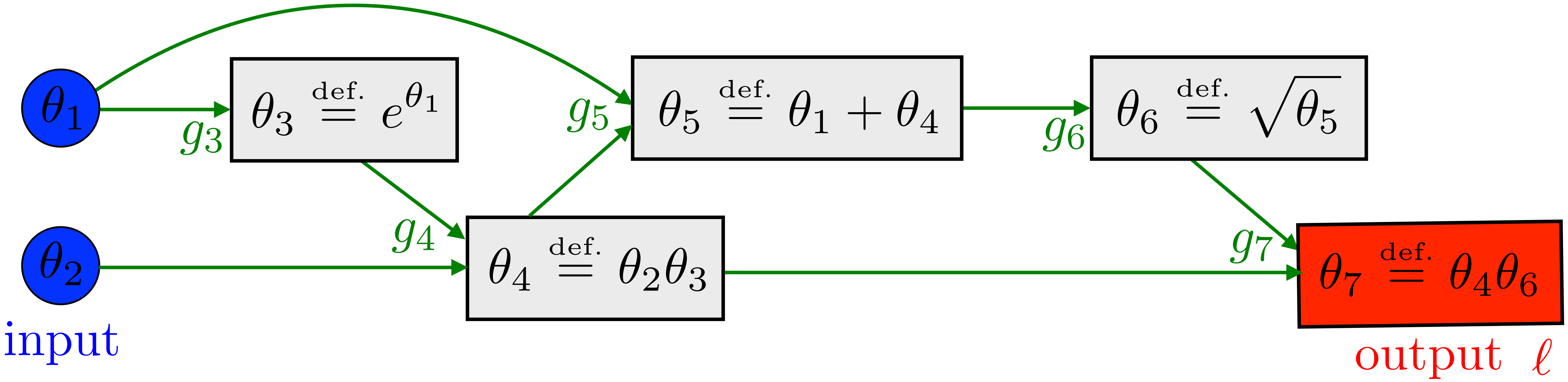
forward

```
function $\ell(\theta_1, \ldots, \theta_M)$
    for $r = M+1, \ldots, R$
        $\theta_r = g_r(\theta_{\text{Parents}(r)})$
    return $\theta_R$
```



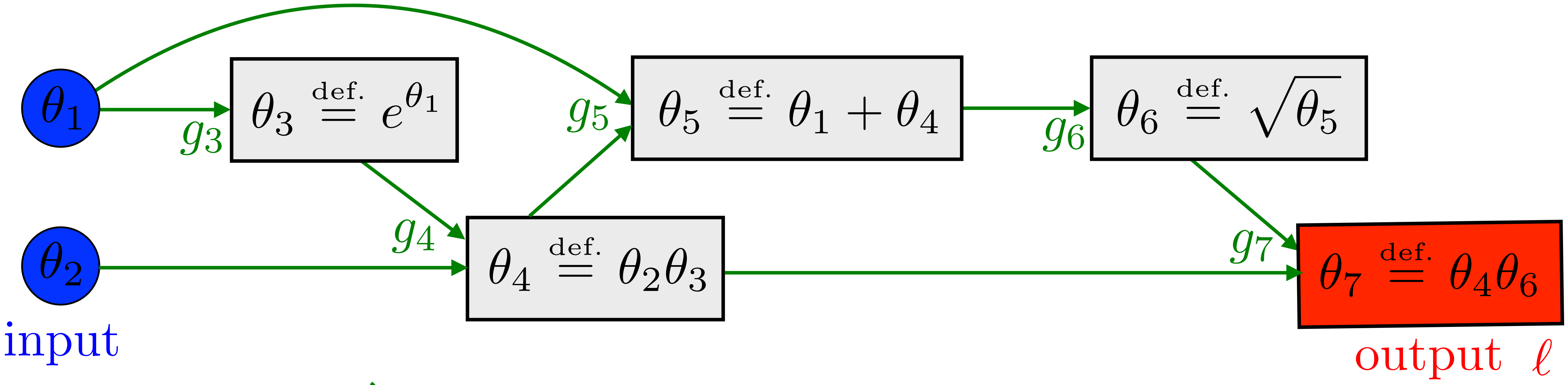
Example

$$\ell(\theta_1, \theta_2) \overset{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$

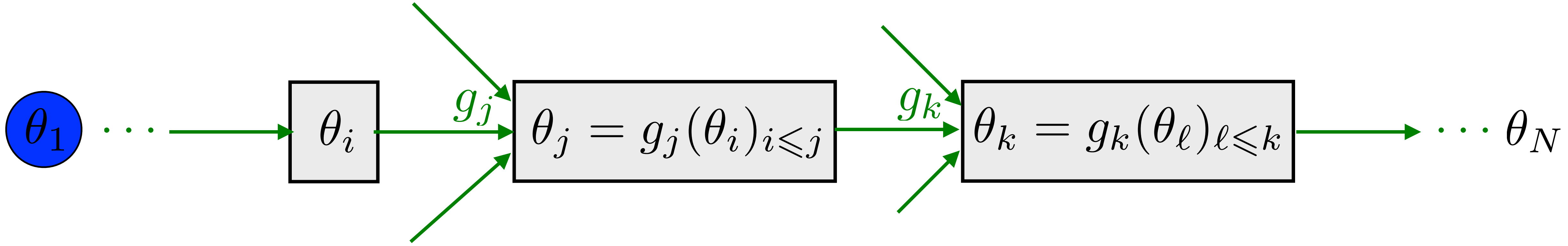

θ_1
θ_2
input
g_3
$\theta_3 \overset{\text{def.}}{=} e^{\theta_1}$
g_4
$\theta_4 \overset{\text{def.}}{=} \theta_2\theta_3$
g_5
$\theta_5 \overset{\text{def.}}{=} \theta_1 + \theta_4$
g_6
$\theta_6 \overset{\text{def.}}{=} \sqrt{\theta_5}$
g_7
$\theta_7 \overset{\text{def.}}{=} \theta_4\theta_6$
output ℓ

Example

$$\ell(\theta_1, \theta_2) \overset{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$



θ_1
g_3
$\theta_3 \overset{\text{def.}}{=} e^{\theta_1}$
g_5
$\theta_5 \overset{\text{def.}}{=} \theta_1 + \theta_4$
g_6
$\theta_6 \overset{\text{def.}}{=} \sqrt{\theta_5}$
θ_2
g_4
$\theta_4 \overset{\text{def.}}{=} \theta_2\theta_3$
g_7
$\theta_7 \overset{\text{def.}}{=} \theta_4\theta_6$
input
output ℓ



θ_1
$\cdots$
θ_i
g_j
$\theta_j = g_j(\theta_i)_{i \leqslant j}$
g_k
$\theta_k = g_k(\theta_\ell)_{\ell \leqslant k}$
$\cdots\ \theta_N$

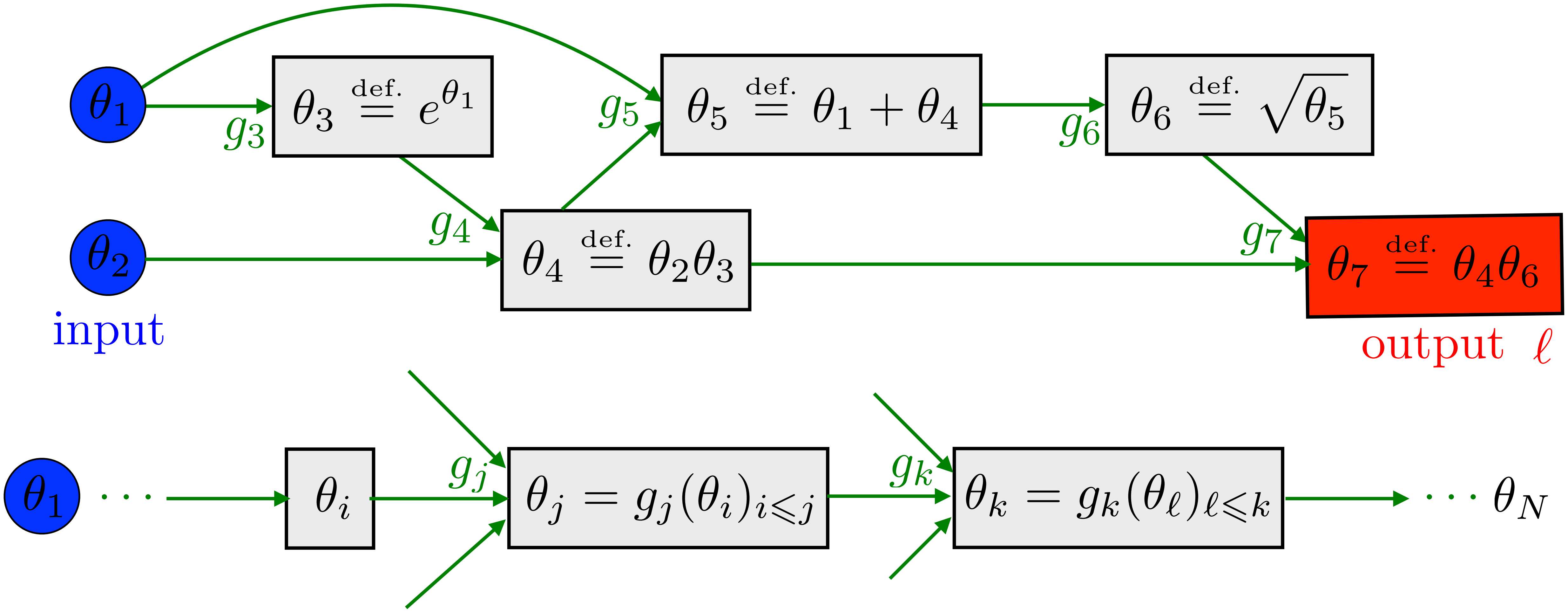
Chain rules:

$$“\ \frac{\partial\theta_j}{\partial\theta_1} = \sum_{i \in \text{Parent}(j)} \underset{\searrow\ \partial_i g_j(\theta)}{\frac{\partial\theta_j}{\partial\theta_i}} \frac{\partial\theta_i}{\partial\theta_1}\ ”$$

“Classical” evaluation: **forward**.
Complexity ∼ #inputs.

Example

$$\ell(\theta_1, \theta_2) \overset{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$


θ_1
θ_2
input
g_3
$\theta_3 \overset{\text{def.}}{=} e^{\theta_1}$
g_4
$\theta_4 \overset{\text{def.}}{=} \theta_2\theta_3$
g_5
$\theta_5 \overset{\text{def.}}{=} \theta_1 + \theta_4$
g_6
$\theta_6 \overset{\text{def.}}{=} \sqrt{\theta_5}$
g_7
$\theta_7 \overset{\text{def.}}{=} \theta_4\theta_6$
output ℓ
θ_1
$\cdots$
θ_i
g_j
$\theta_j = g_j(\theta_i)_{i \leqslant j}$
g_k
$\theta_k = g_k(\theta_\ell)_{\ell \leqslant k}$
$\cdots \theta_N$

Chain rules:

$$\text{“}\; \frac{\partial \theta_j}{\partial \theta_1} = \sum_{i \in \text{Parent}(j)} \frac{\partial \theta_j}{\partial \theta_i} \frac{\partial \theta_i}{\partial \theta_1} \;\text{”}$$

$\partial_i g_j(\theta)$

“Classical” evaluation: **forward**.
Complexity $\sim$ #inputs.

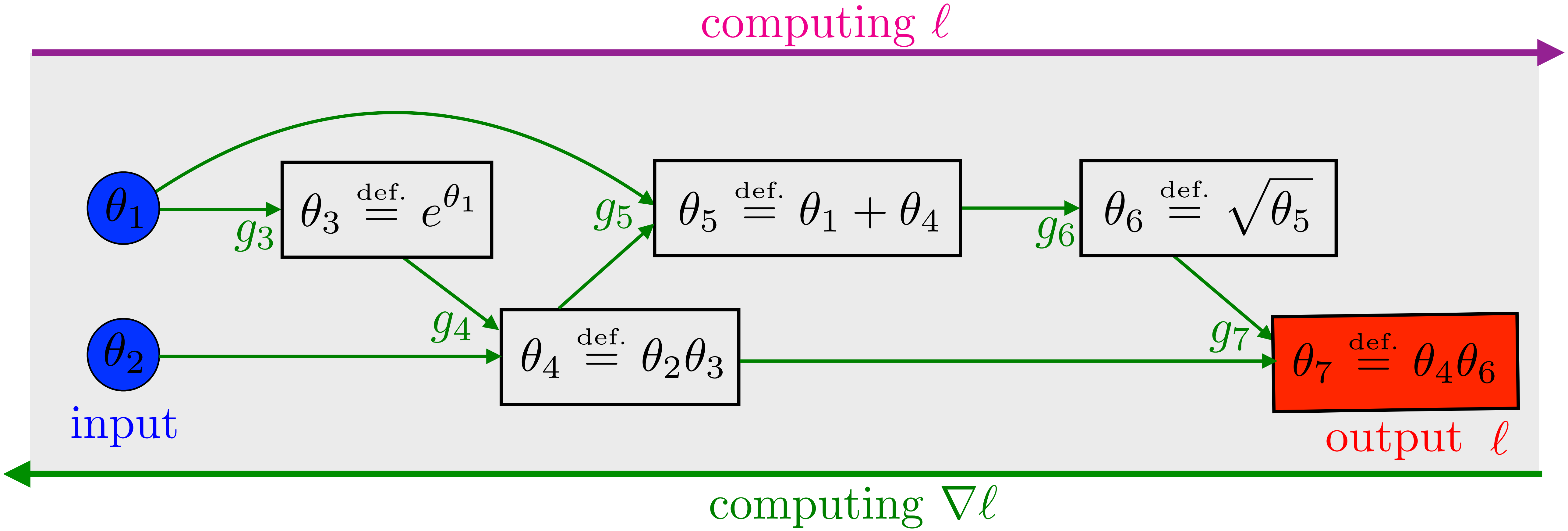
$$\text{“}\; \frac{\partial \theta_N}{\partial \theta_j} = \sum_{k \in \text{Child}(j)} \frac{\partial \theta_N}{\partial \theta_k} \frac{\partial \theta_k}{\partial \theta_j} \;\text{”}$$

$\nabla_j \ell(\theta)$ $\nabla_k \ell(\theta)$ $\partial_j g_k(\theta)$

Backward evaluation.
Complexity $\sim$ #outputs (1 for grad).

Backward Automatic Differentiation

$$\ell(\theta_1, \theta_2) \overset{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$



computing ℓ
θ_1
θ_2
g_3
$\theta_3 \overset{\text{def.}}{=} e^{\theta_1}$
g_4
$\theta_4 \overset{\text{def.}}{=} \theta_2\theta_3$
g_5
$\theta_5 \overset{\text{def.}}{=} \theta_1 + \theta_4$
g_6
$\theta_6 \overset{\text{def.}}{=} \sqrt{\theta_5}$
g_7
$\theta_7 \overset{\text{def.}}{=} \theta_4\theta_6$
input
output ℓ
computing $\nabla\ell$

forward

```
function $\ell(\theta_1, \ldots, \theta_M)$
  for $r = M+1, \ldots, R$
  |   $\theta_r = g_r(\theta_{\text{Parents}(r)})$
  return $\theta_R$
```

backward

```
function $\nabla\ell(\theta_1, \ldots, \theta_M)$
    $\nabla_R \ell = 1$
    for $r = R-1, \ldots, 1$
    |   $\nabla_r \ell = \sum_{s \in \text{Child}(r)} \partial_r g_s(\theta)\, \nabla_s \ell$
    return $(\nabla_1 \ell, \ldots, \nabla_M \ell)$
```

Softwares