Differential Programming

Gabriel Peyré

www.numerical-tours.com

Mathematical Coffees

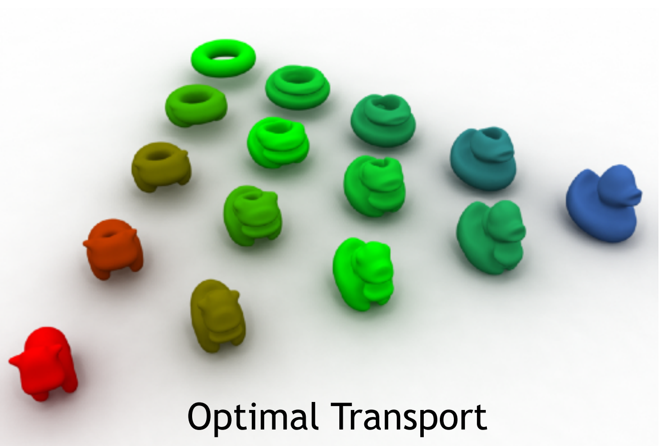
Huawei-FSMP joint seminars

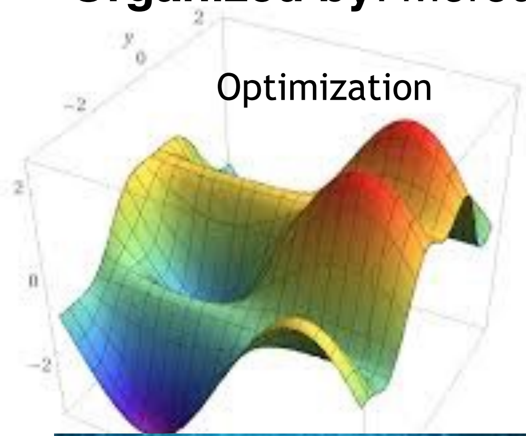
https://mathematical-coffees.github.io

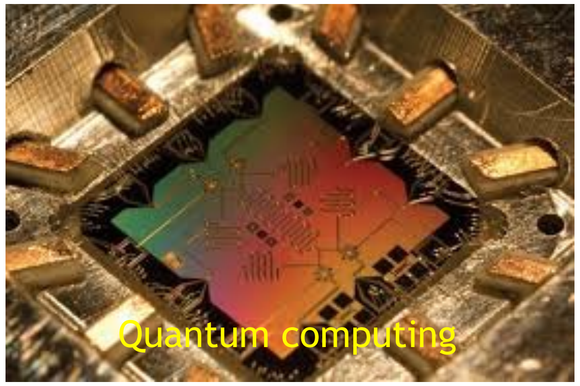
HUAWEI

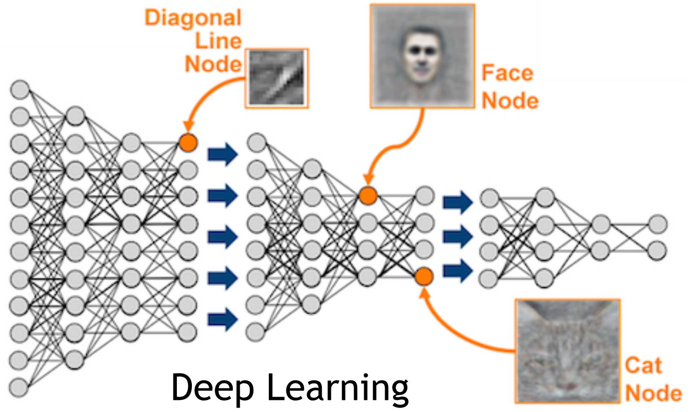
FSMP Fondation Sciences Mathématiques de Paris

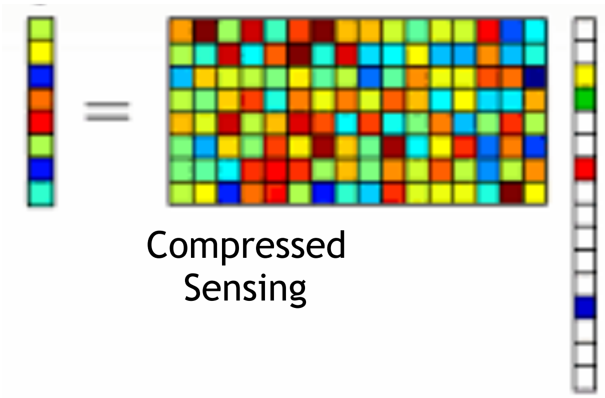
Organized by: Mérouane Debbah & Gabriel Peyré

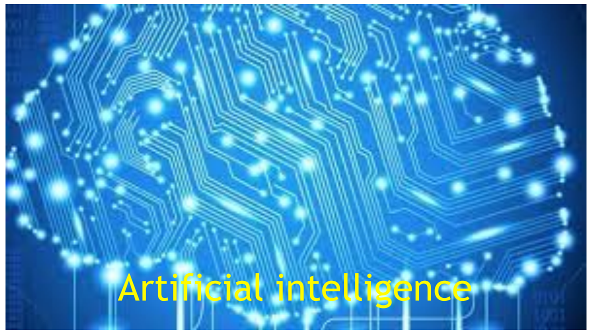


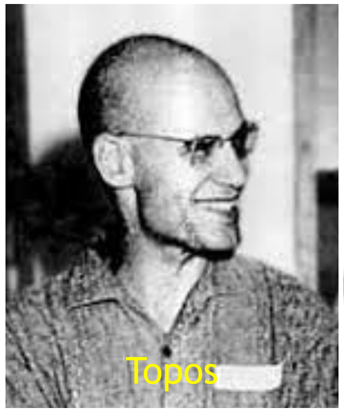










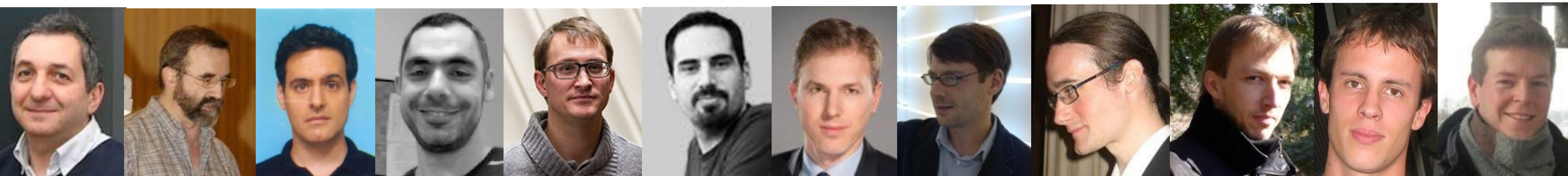


scikit learn

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

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

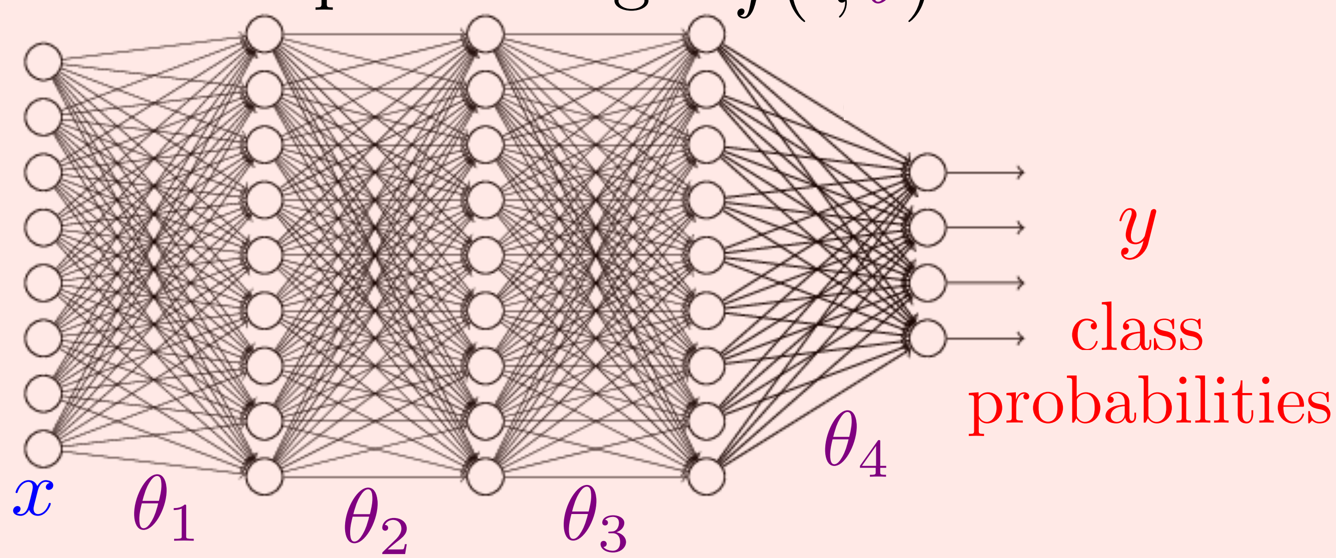
Loss Model Input Parameter Output

Model Fitting in Data Sciences

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

Loss Model Input Parameter Output

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

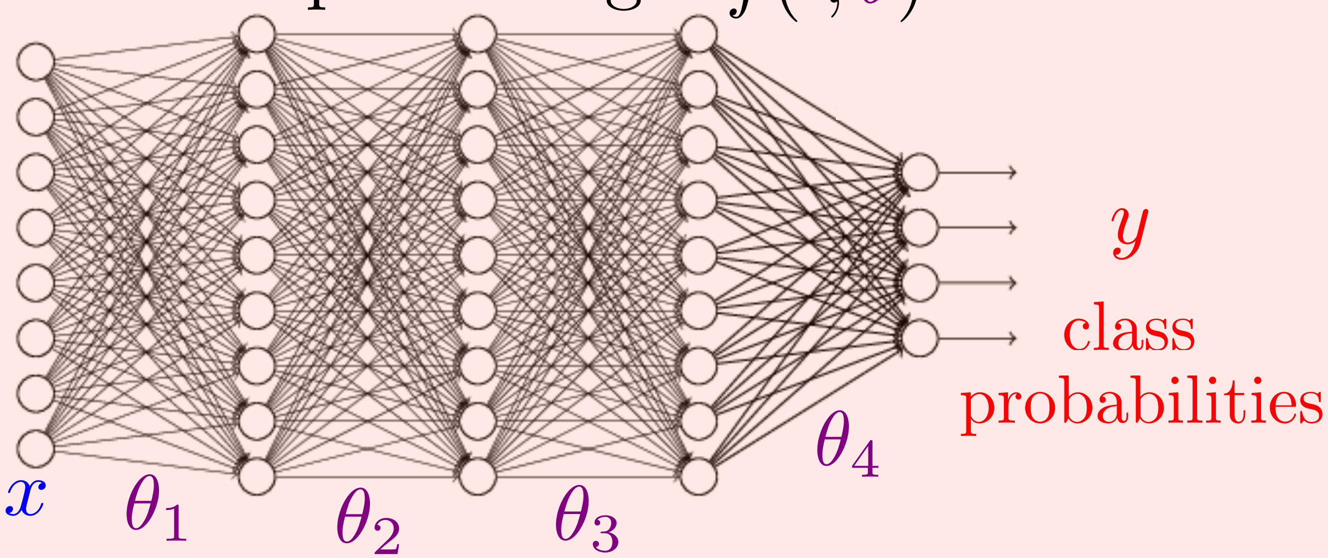


Model Fitting in Data Sciences

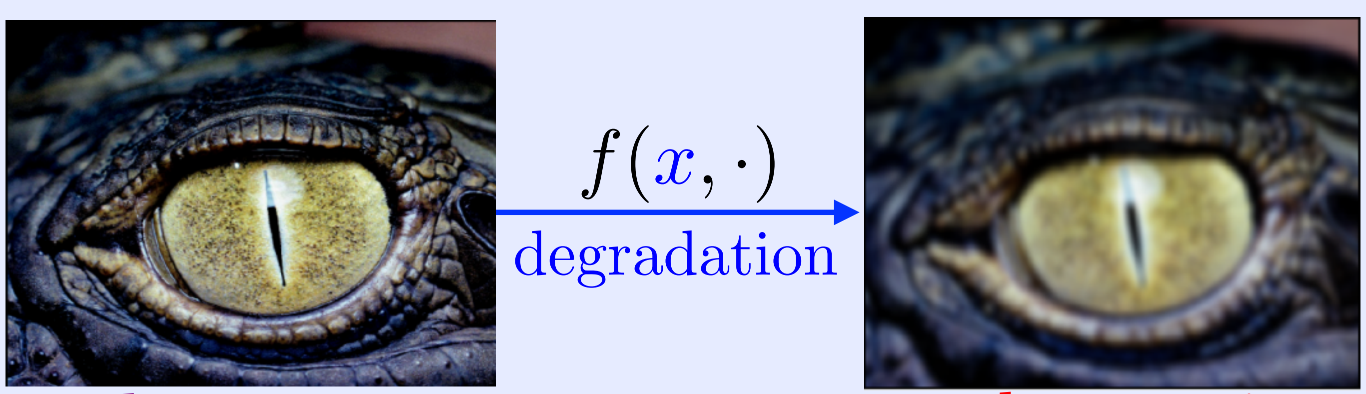
$$\min_{\theta} \mathcal{E}(\theta) \overset{\text{def.}}{=} L(f(x, \theta), y)$$

Loss — Model — Input — Parameter — Output

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



Super-resolution:



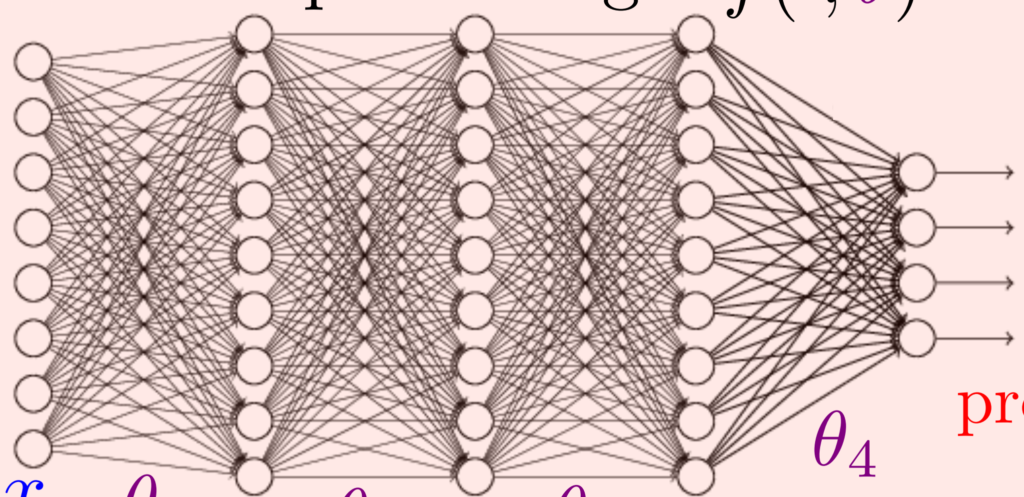
θ unknown image — y observation

Model Fitting in Data Sciences

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

Loss Model Input Parameter Output

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



x θ_1 θ_2 θ_3 θ_4

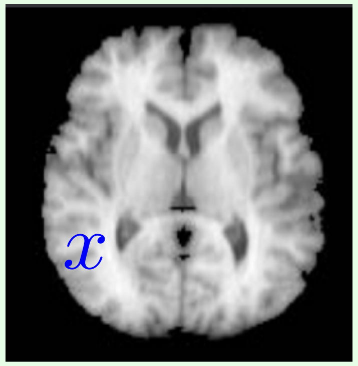
y class probabilities

Super-resolution:

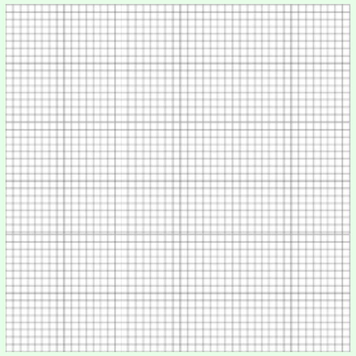
$\xrightarrow[\text{degradation}]{f(x,\cdot)}$

θ unknown image y observation

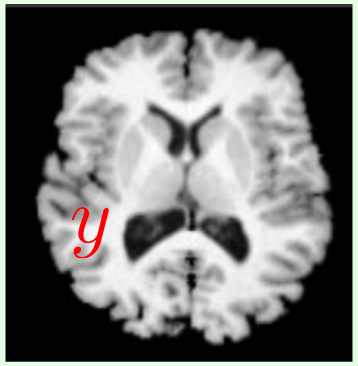
Medical imaging registration:



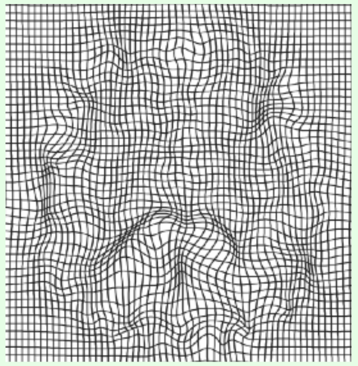
x



$\xrightarrow[\text{diffeomorphism}]{f(\cdot,\theta)}$



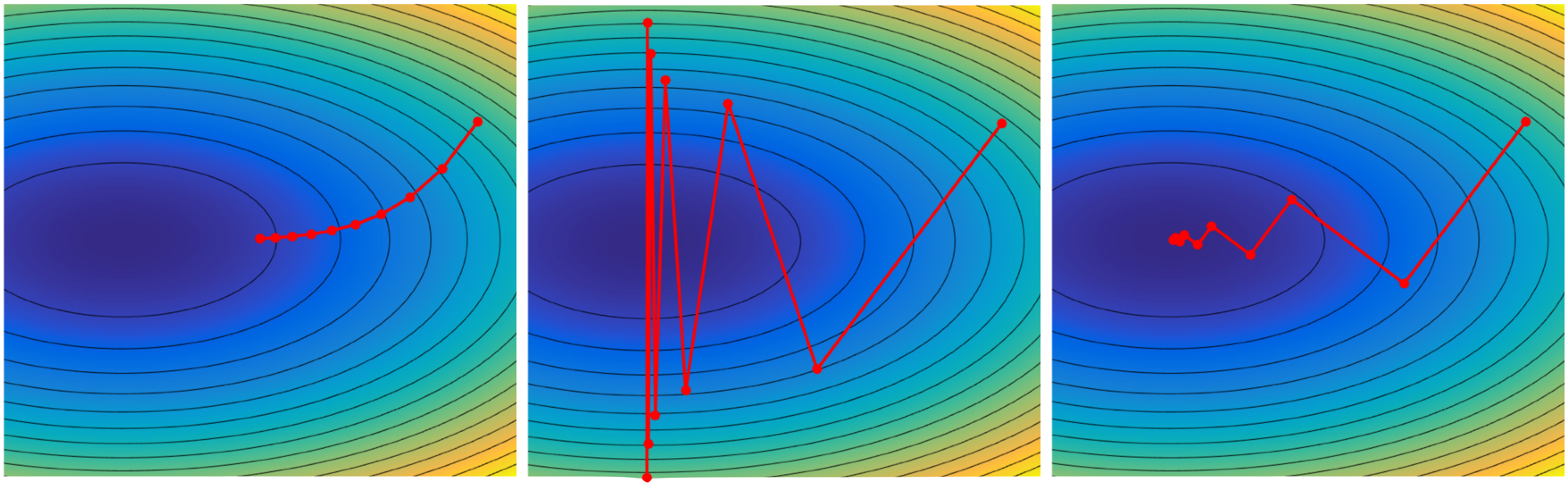
y



Gradient-based Methods

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

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

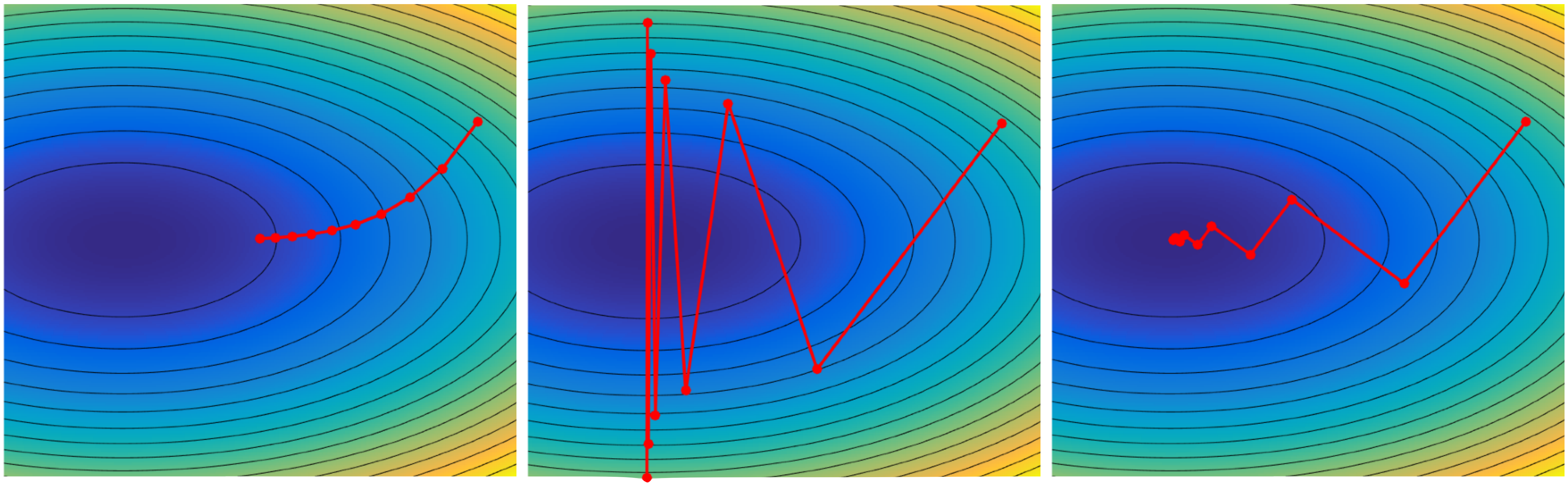


Small τ_ℓ | Large τ_ℓ | Optimal $\tau_\ell = \tau_\ell^\star$

Gradient-based Methods

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

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



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: $\mathcal{E} : \mathbb{R}^n \rightarrow \mathbb{R}$ 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:
$$\nabla\mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta))$$
$K(n+1)$ 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: $$\nabla\mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta))$$
$K(n+1)$ 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:
$$\nabla \mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta))$$
$K(n+1)$ operations, intractable for large n.

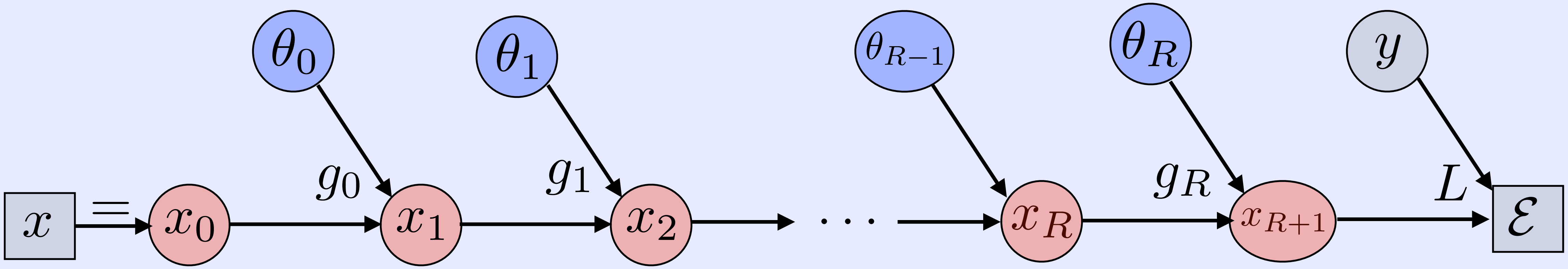
Theorem: there is an algorithm to compute $\nabla \mathcal{E}$ in $O(K)$ operations. [Seppo Linnainmaa, 1970]

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]
```

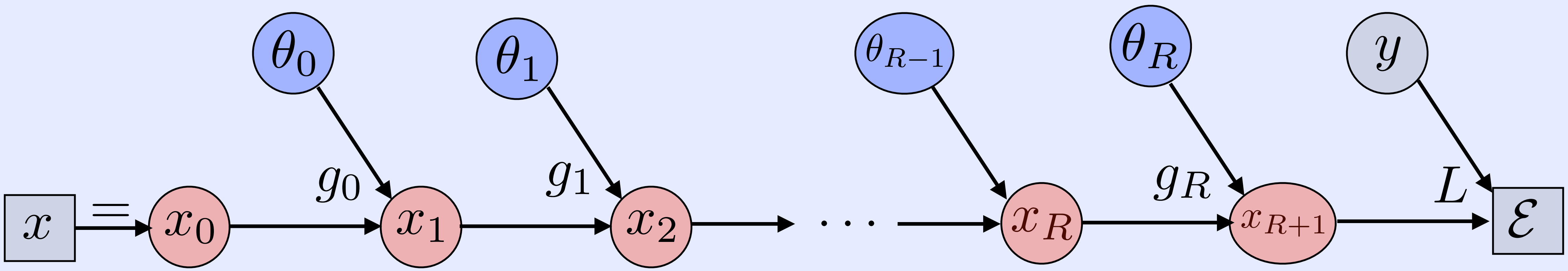
Feedfordward Computational Graphs

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

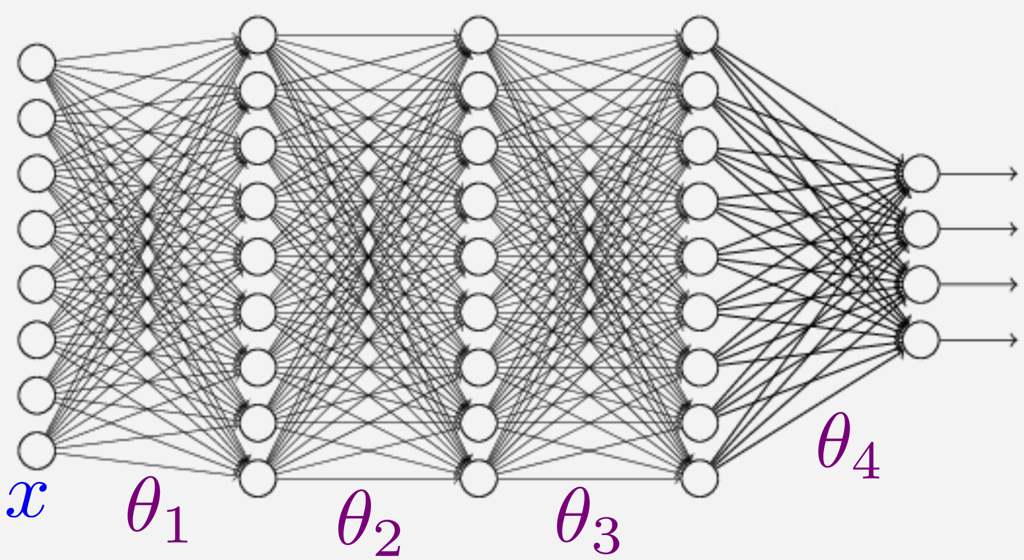


Feedforward Computational Graphs

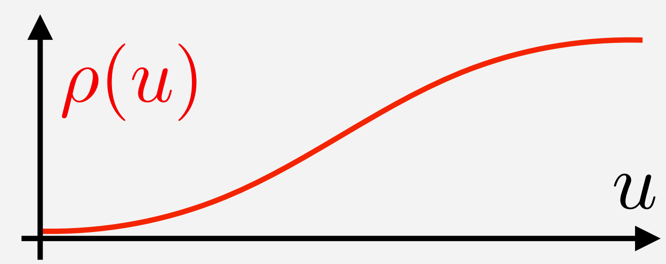
$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Example: deep neural network (here fully connected)



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



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

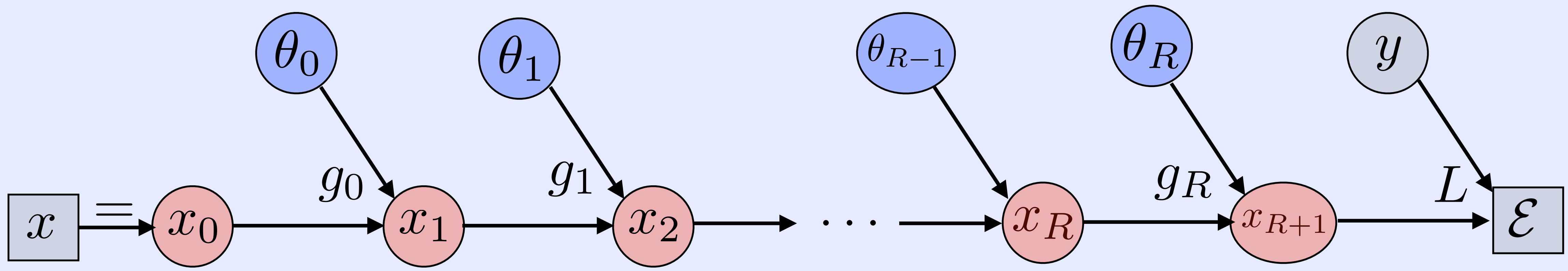
$$x_r \in \mathbb{R}^{d_r}$$

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

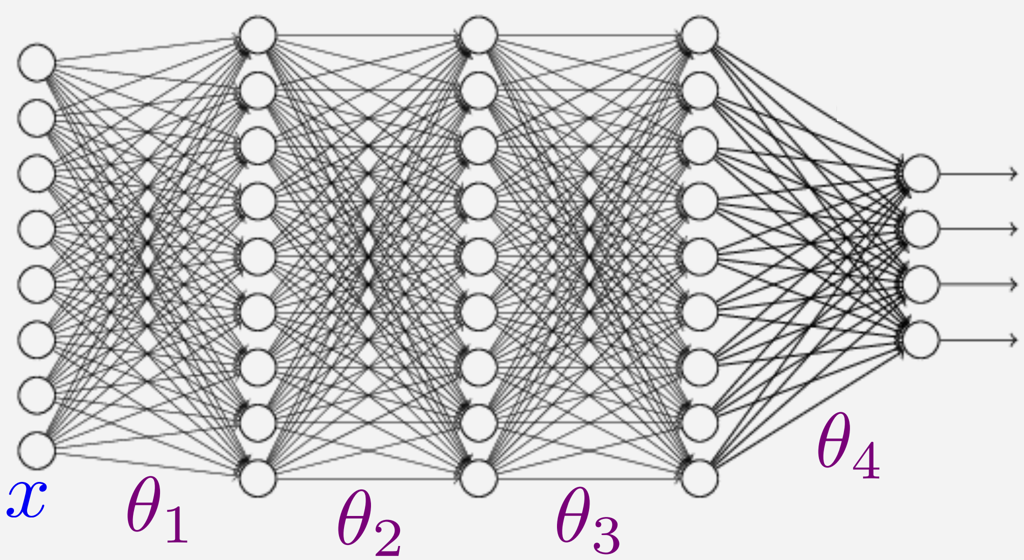
$$b_r \in \mathbb{R}^{d_{r+1}}$$

Feedfordward Computational Graphs

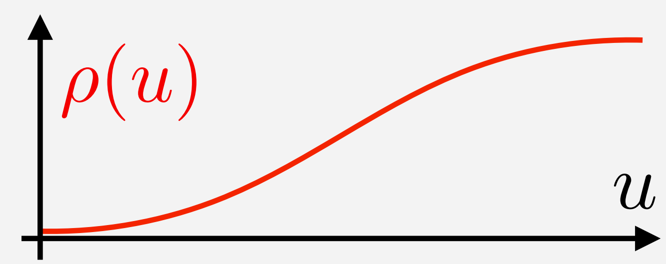
$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Example: deep neural network (here fully connected)



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



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

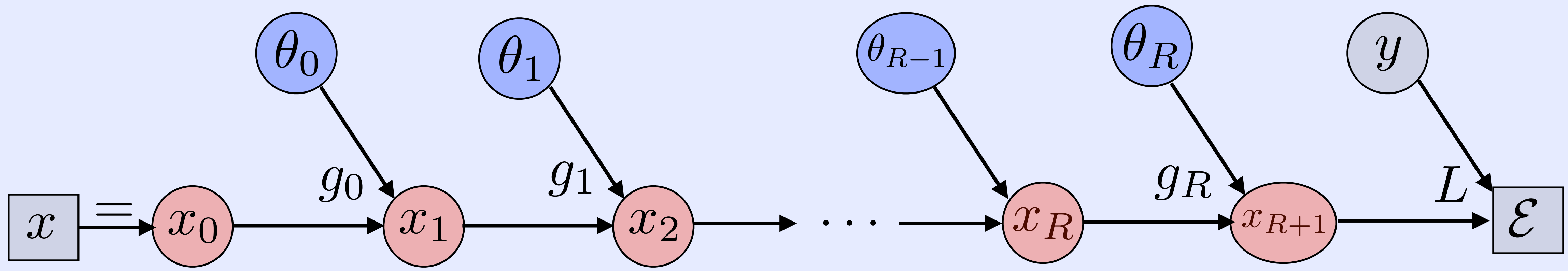
Logistic loss:
(classification)

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

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

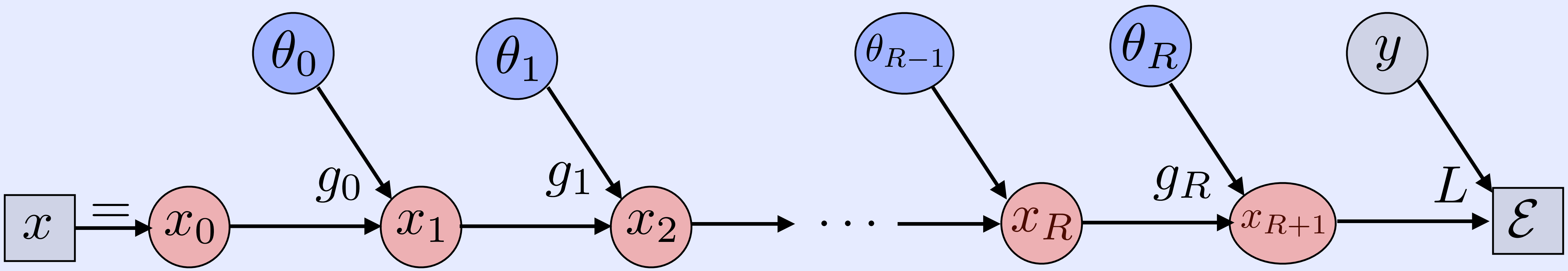
Backpropagation Algorithm

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



Backpropagation Algorithm

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



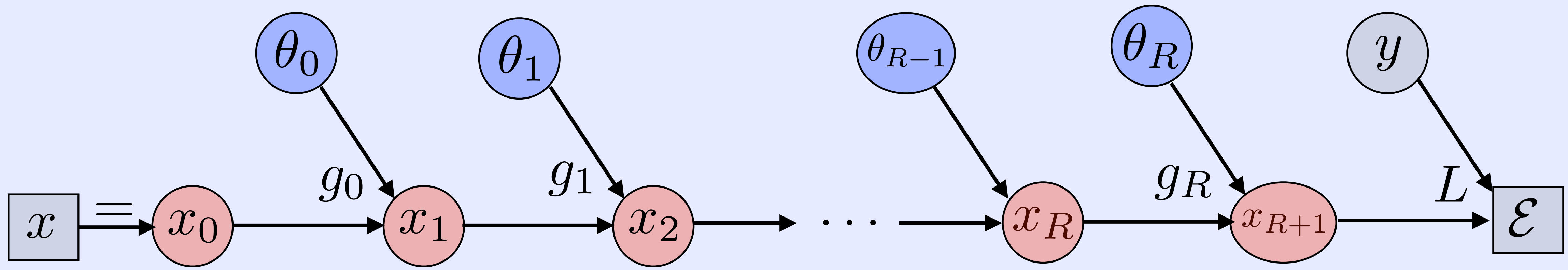
Proposition: $\forall r = R, \ldots, 0,$

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

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

Backpropagation Algorithm

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



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

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

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

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

$$\forall r = R, \ldots, 0, \qquad \nabla_{x_r}\mathcal{E} = A_r^\top M_r \qquad \nabla_{A_r}\mathcal{E} = M_r x_r^\top \qquad \nabla_{b_r}\mathcal{E} = M_r \mathbb{1}$$

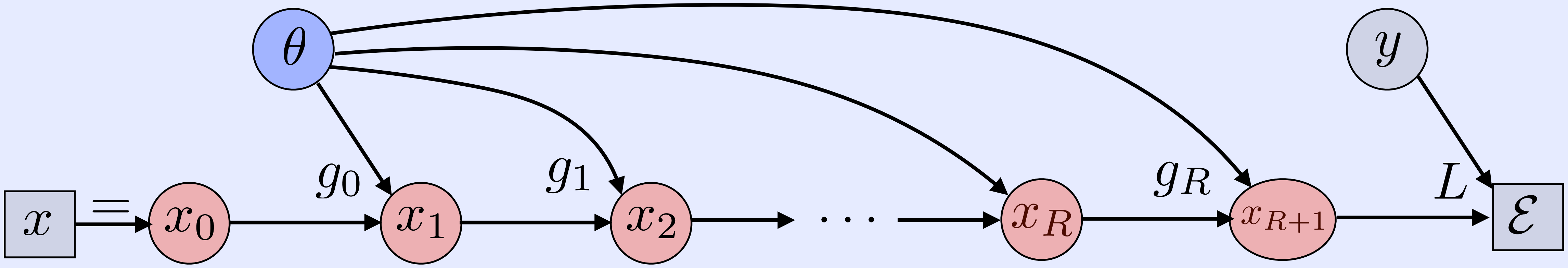
$$M_r \stackrel{\text{def.}}{=} \rho'(A_r x_r + b_r) \odot \nabla_{x_{r+1}}\mathcal{E}$$

```
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]
```

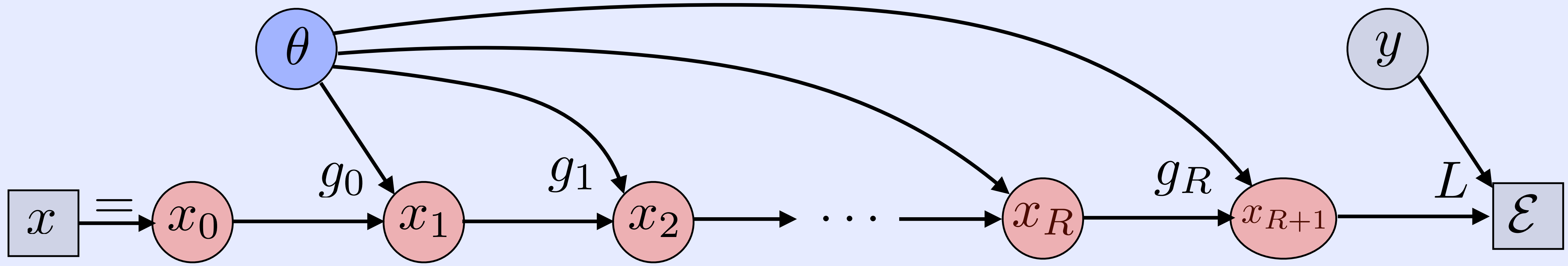
Recurrent Architectures

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

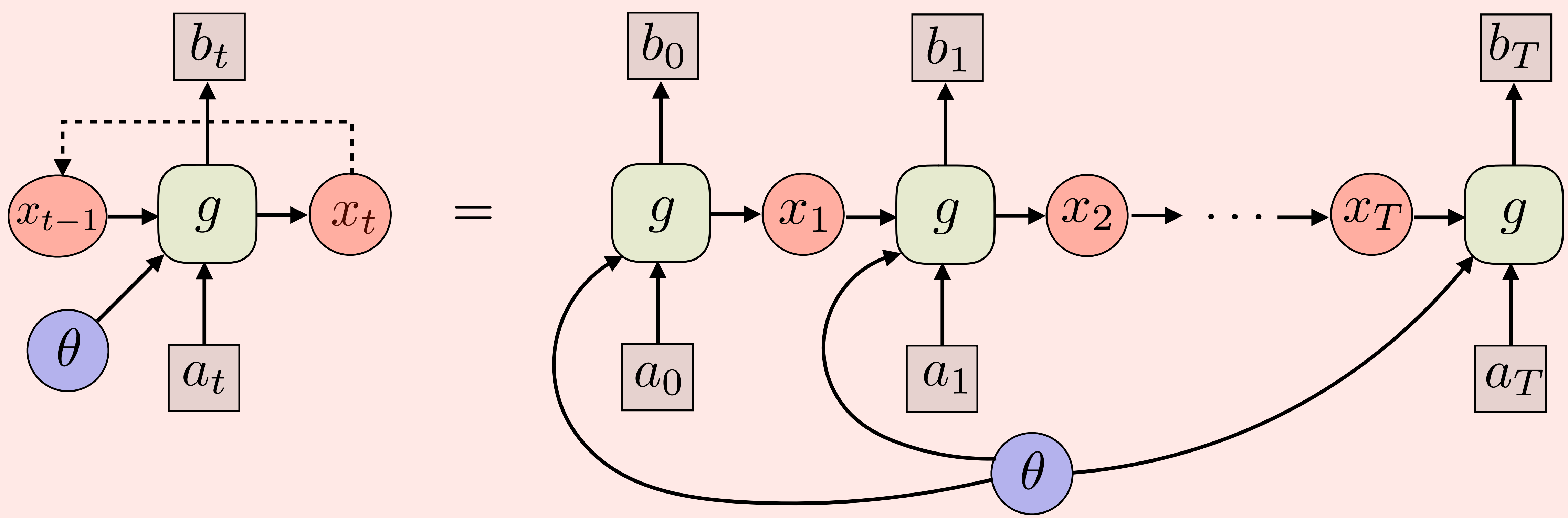


Recurrent Architectures

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

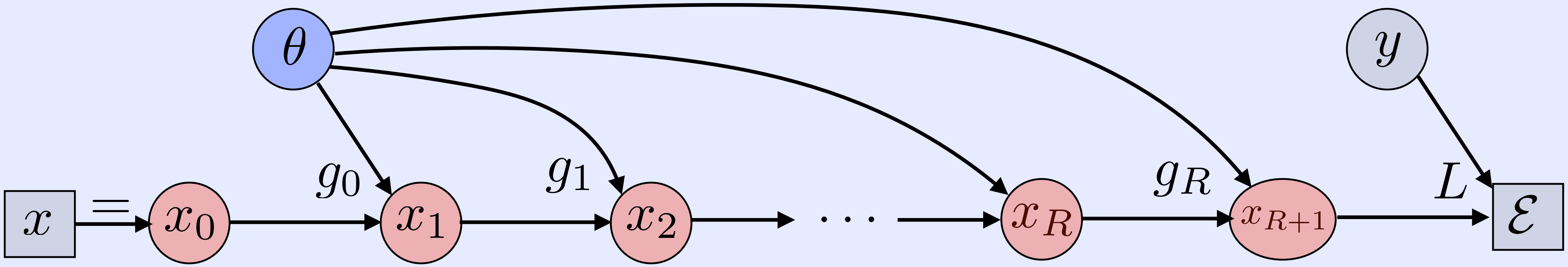


Recurrent networks for natural language processing:

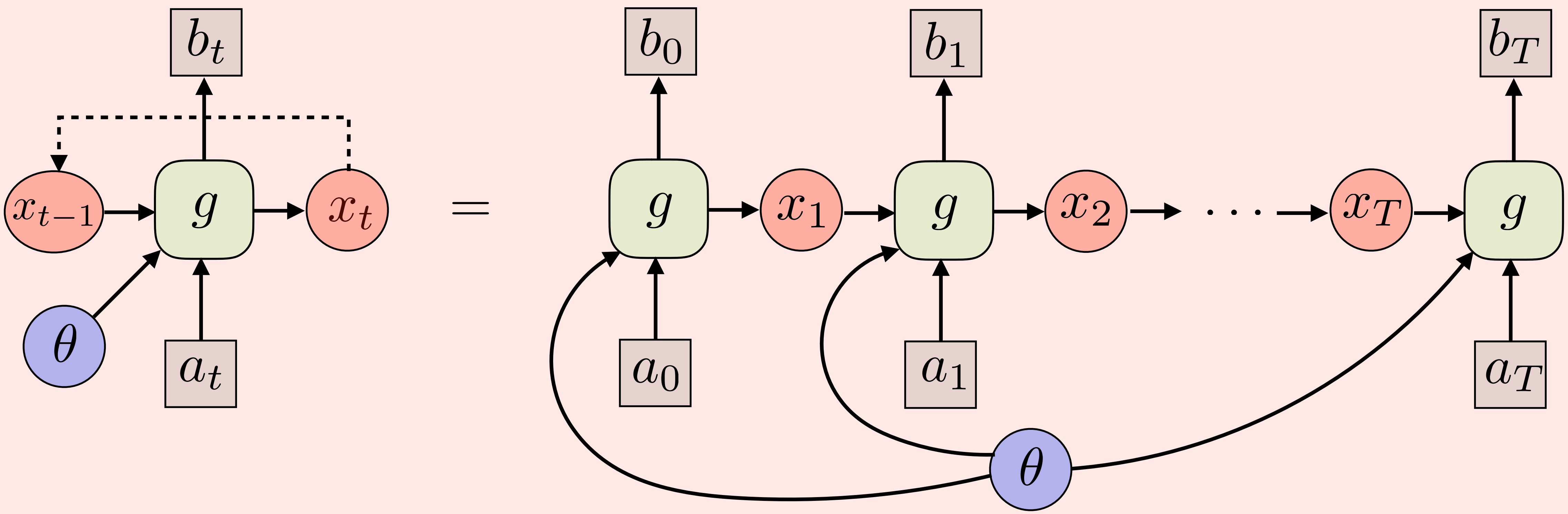


Recurrent Architectures

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



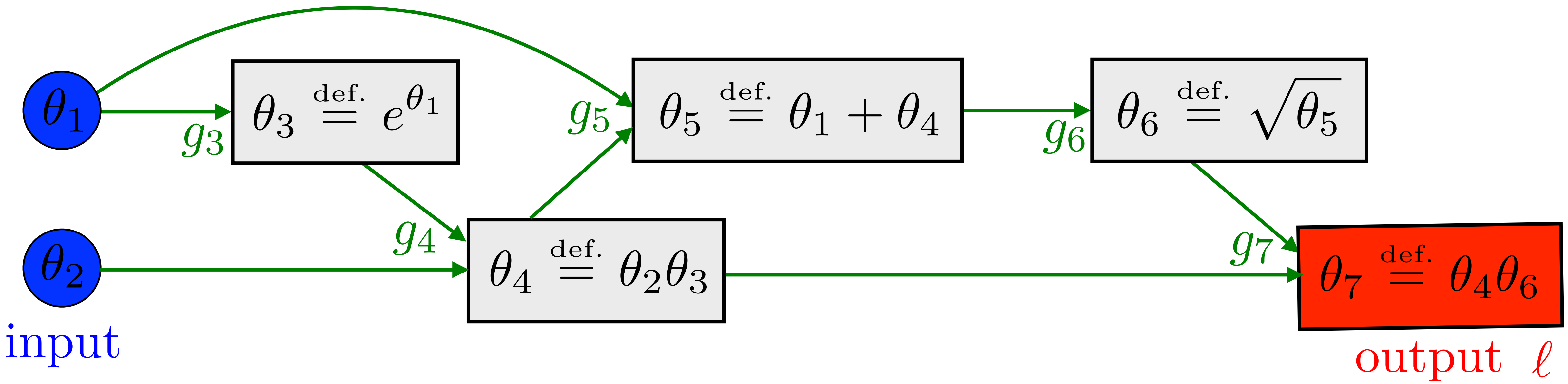
Recurrent networks for natural language processing:



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

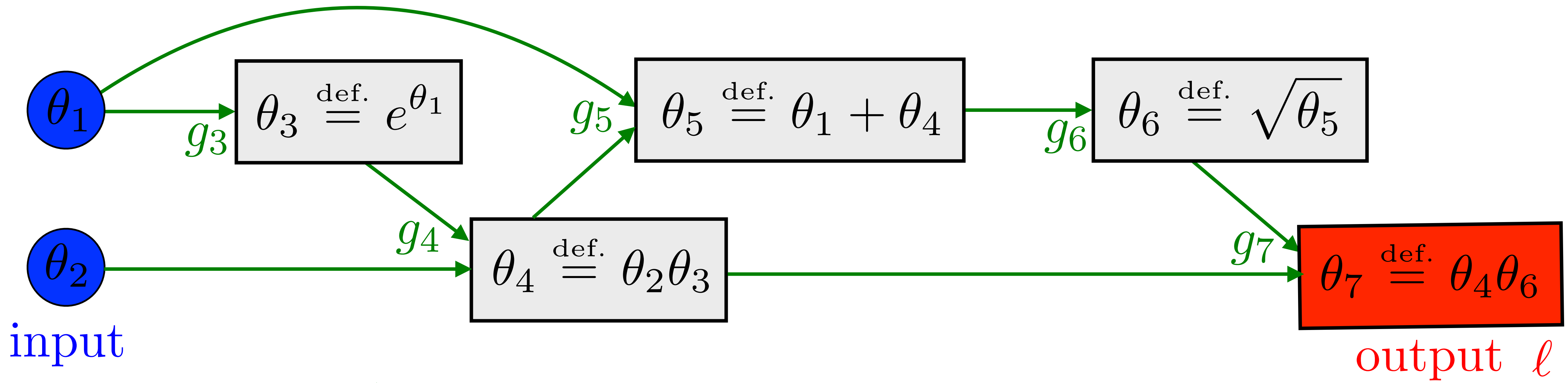
Example

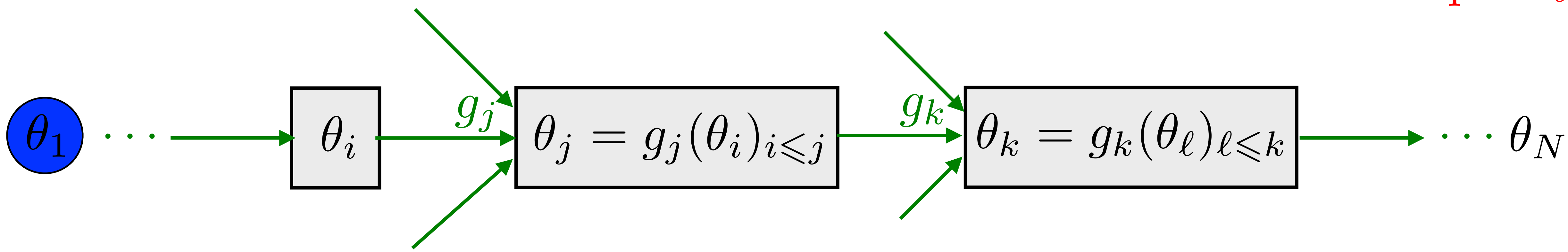
$$\ell(\theta_1, \theta_2) \overset{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$



Example

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





Chain rules:

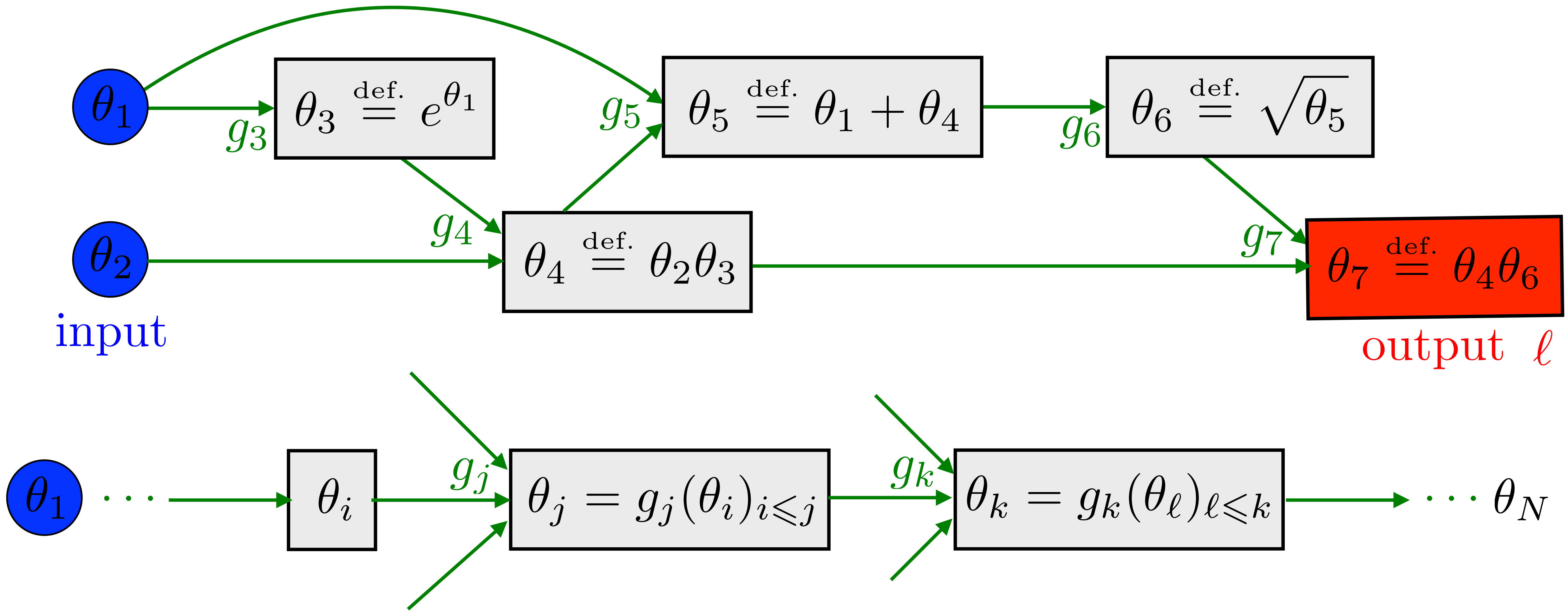
$$\text{“}\ \frac{\partial \theta_j}{\partial \theta_1} = \sum_{i \in \text{Parent}(j)} \frac{\partial \theta_j}{\partial \theta_i} \frac{\partial \theta_i}{\partial \theta_1}\ \text{”}$$

$$\frac{\partial \theta_j}{\partial \theta_i} \to \partial_i g_j(\theta)$$

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

Example

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



Chain rules:

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

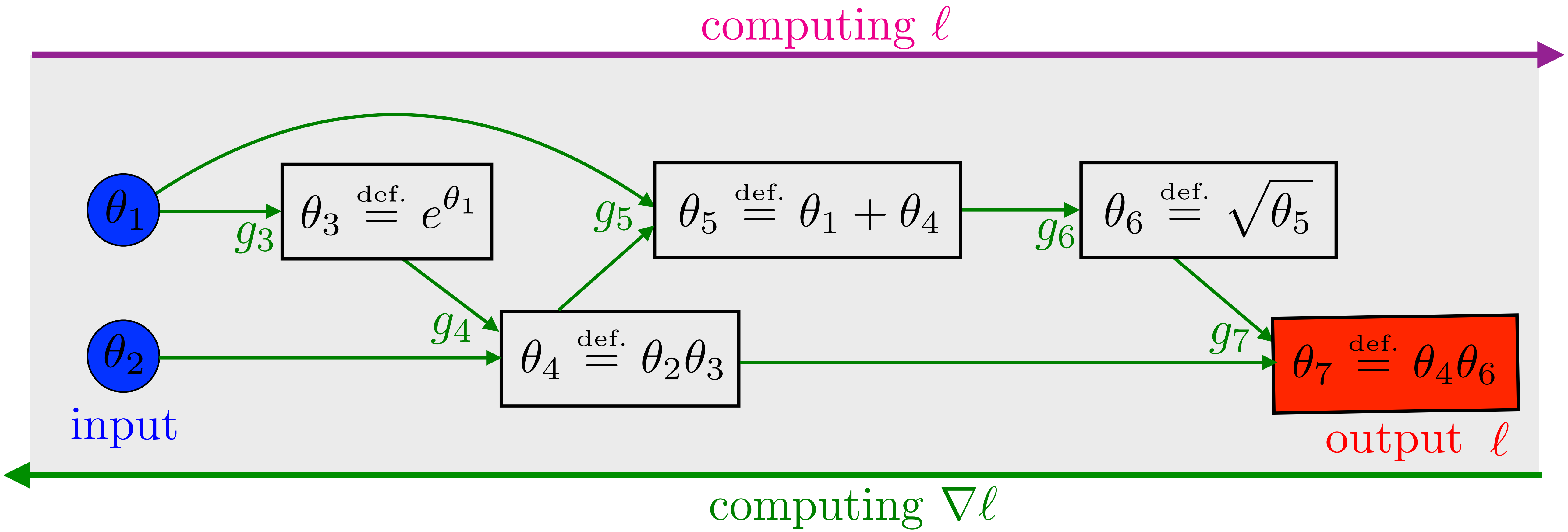
“Classical” evaluation: **forward**.
Complexity $\sim$ #inputs.

$$\text{“}\ \underbrace{\frac{\partial \theta_N}{\partial \theta_j}}_{\nabla_j \ell(\theta)} = \sum_{k \in \text{Child}(j)} \underbrace{\frac{\partial \theta_N}{\partial \theta_k}}_{\nabla_k \ell(\theta)} \underbrace{\frac{\partial \theta_k}{\partial \theta_j}}_{\partial_j g_k(\theta)}\ \text{”}$$

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

Backward Automatic Differentiation

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



forward

function $\ell(\theta_1, \ldots, \theta_M)$
  for $r = M+1, \ldots, R$
    $\theta_r = g_r(\theta_{\text{Parents}(r)})$
  return θ_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 $(\nabla_1 \ell, \ldots, \nabla_M \ell)$