The Mathematics of Neural Networks

Gabriel Peyré

www.numerical-tours.com

CNRS

Python GitHub jupyter

1794 ENS ÉCOLE NORMALE SUPÉRIEURE

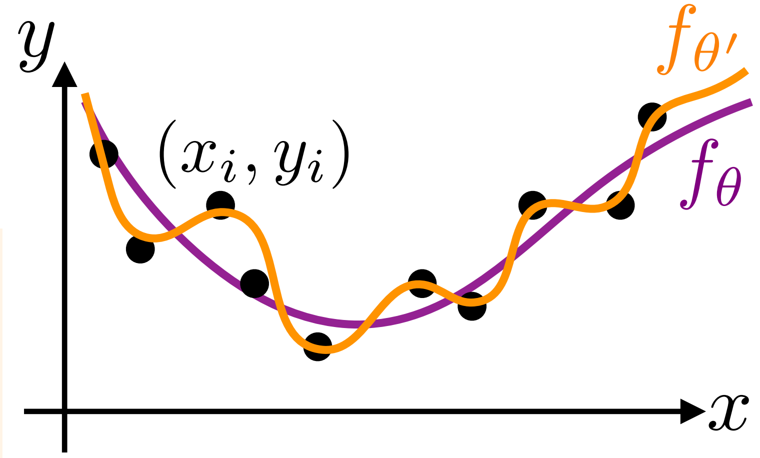
Overview

- **Empirical Risk Minimization**
- Perceptrons
- Optimization
- Convolutional Networks
- Residual Networks
- Transformers

Empirical Risk Minimization

Dataset: $(x_i, y_i)_{i=1}^n$. $\quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$

Prediction: $\quad y \approx f_\theta(x)$



Empirical risk minimization:

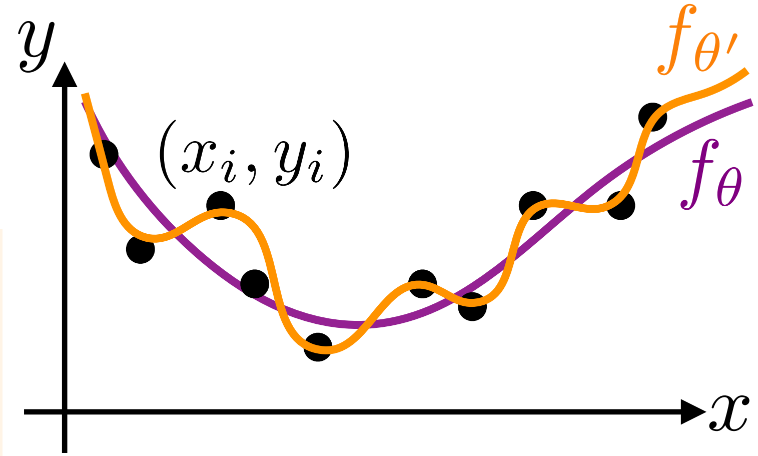
$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

Least square: $\ell(y, y') = (y - y')^2$

Empirical Risk Minimization

Dataset: $(x_i, y_i)_{i=1}^n$. $\quad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$

Prediction: $\quad y \approx f_\theta(x)$



Empirical risk minimization:

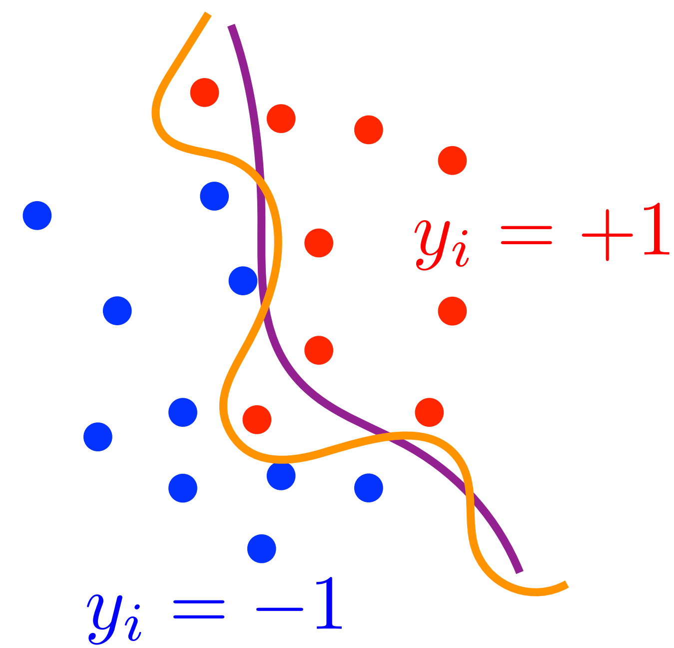
$$\min_\theta \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i)$$

Least square: $\ell(y, y') = (y - y')^2$

Classification: $\quad y_i \in \{-1, 1\}$

$$y \approx \mathrm{sign}(f_\theta(x))$$

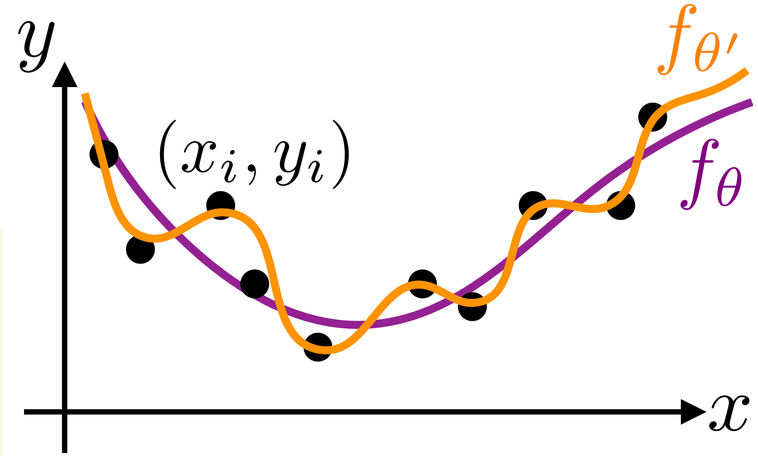
Logistic: $\ell(y, y') = \log(1 + e^{-yy'})$



Empirical Risk Minimization

Dataset: $(x_i, y_i)_{i=1}^n$. $\quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$

Prediction: $\quad y \approx f_\theta(x)$



Empirical risk minimization:

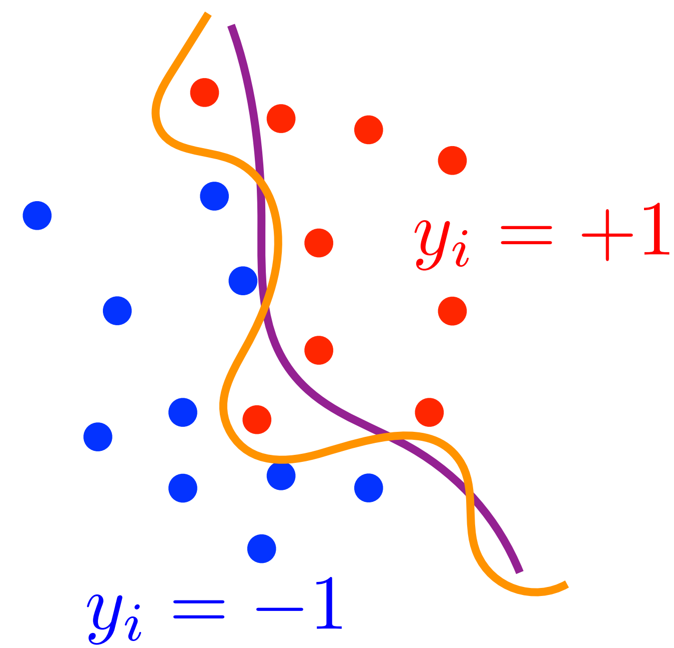
$$\min_\theta \frac{1}{n} \sum_{i=1}^n \ell(f_\theta(x_i), y_i)$$

Least square: $\ell(y, y') = (y - y')^2$

Classification: $\quad y_i \in \{-1, 1\}$

$$y \approx \operatorname{sign}(f_\theta(x))$$

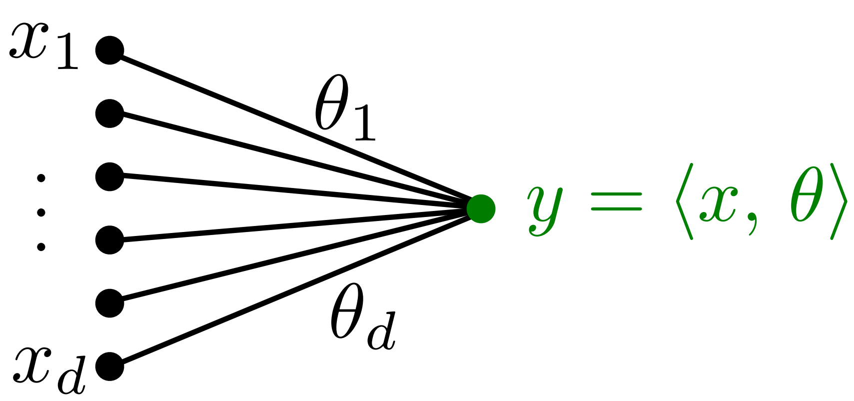
Logistic: $\ell(y, y') = \log(1 + e^{-yy'})$



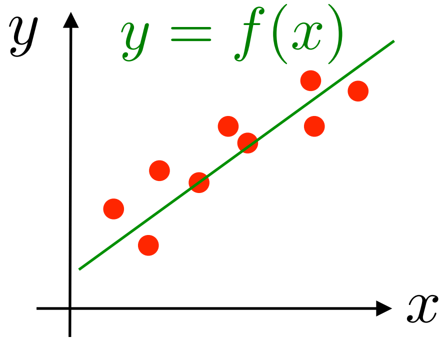
Overfitting, regularization, ...

Linear model (1 layer)

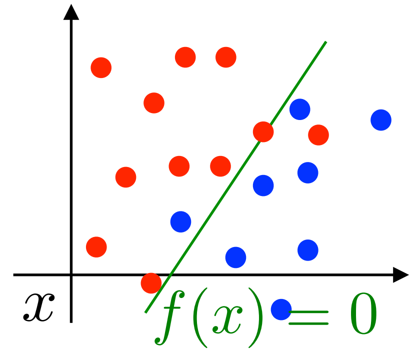
$$f_\theta(x) = \langle x, \theta \rangle = \sum_k x_k \theta_k$$



Regression

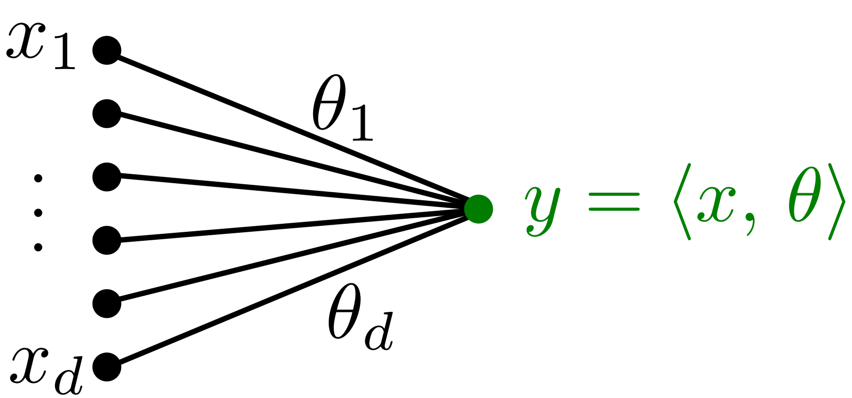


Classification:

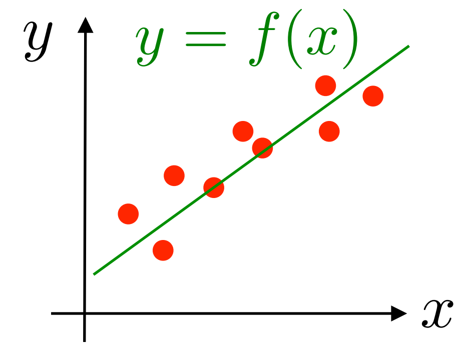


Linear model (1 layer)

$$f_\theta(x) = \langle x,\, \theta \rangle = \sum_k x_k \theta_k$$



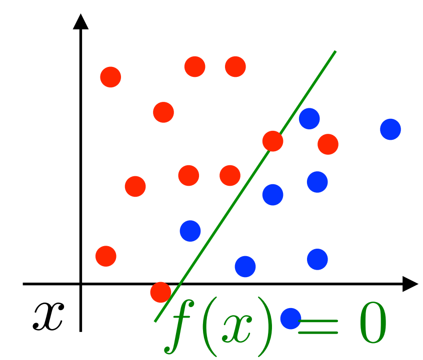
Regression



Convex optimization:

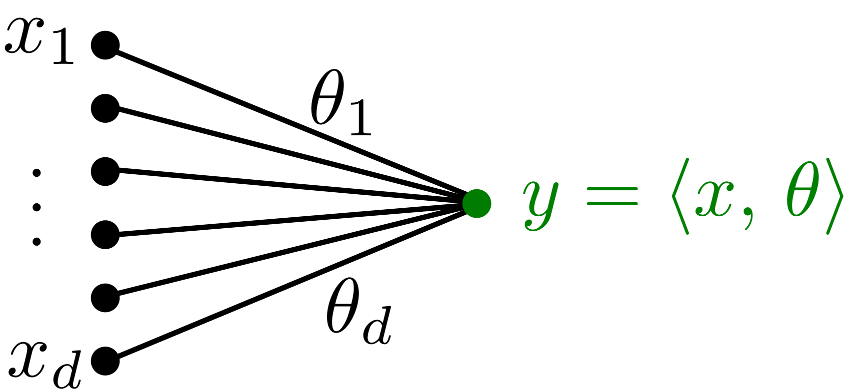
$$\min_\theta \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(\langle x_i,\, \theta_i \rangle, y_i)$$

Classification:

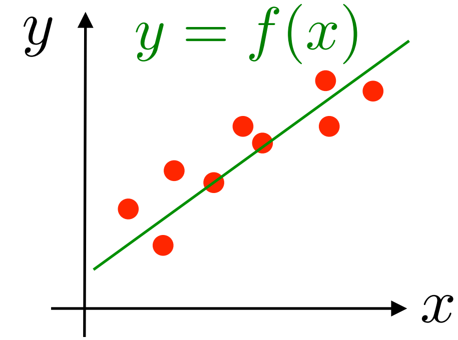


Linear model (1 layer)

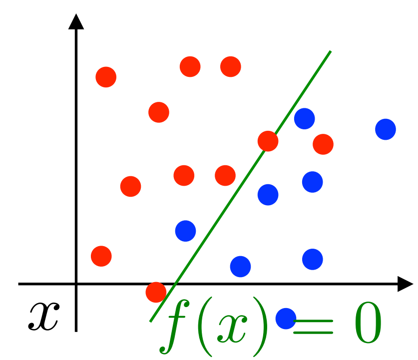
$$f_\theta(x) = \langle x,\, \theta \rangle = \sum_k x_k \theta_k$$



Regression



Classification:



Convex optimization:

$$\min_\theta \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(\langle x_i,\, \theta_i \rangle, y_i)$$

Kernel methods: replace x by $\varphi(x) \in \mathbb{R}^D$

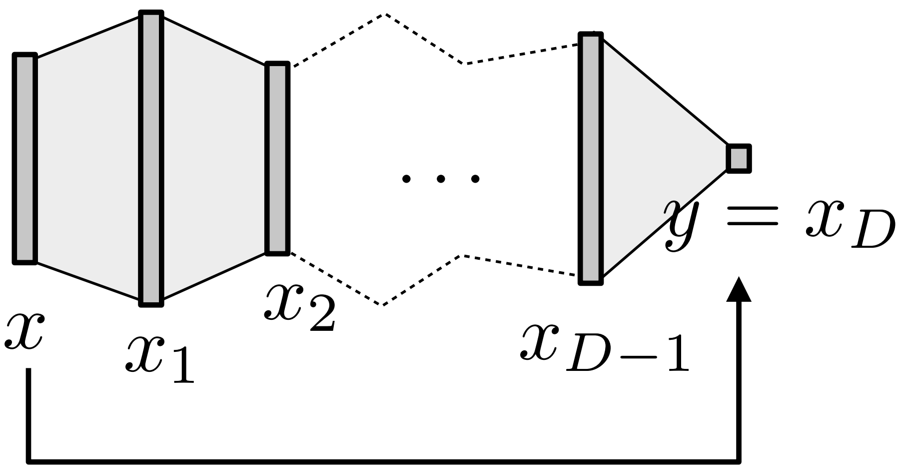
$$(D \gg d, \text{ even } D = \infty!)$$

Deep learning methods: learn $\varphi(x)$!

Overview

- Empirical Risk Minimization
- **Perceptrons**
- Optimization
- Convolutional Networks
- Residual Networks
- Transformers

Multi-layer Perceptron



$$f_\theta(x_0) = x_D$$

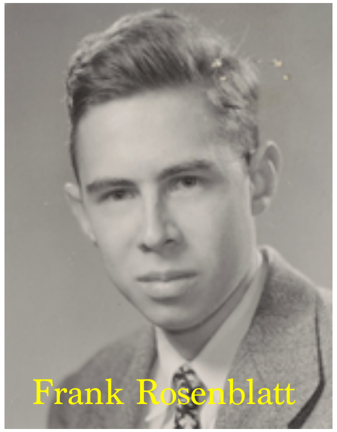
$$x_0 \leftarrow x$$

$$x_{k+1} \triangleq \sigma(W_k x_k + b_k)$$

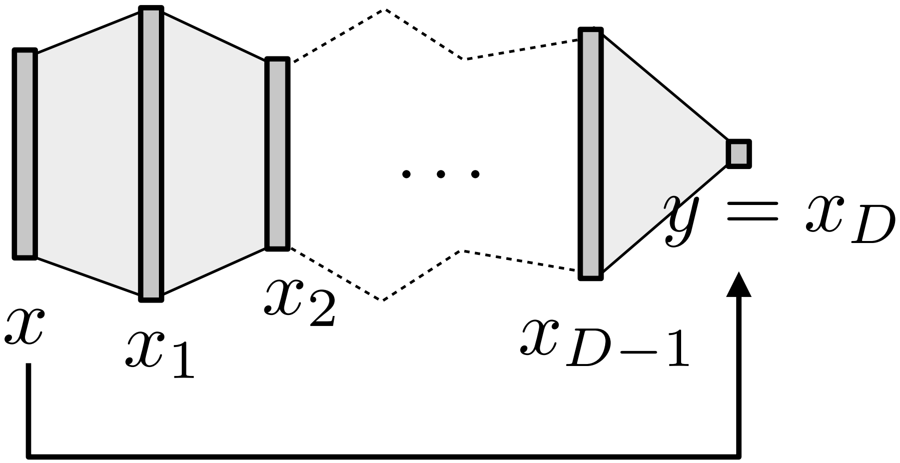
$$\theta = \{(W_k, b_k)\}_{k=0}^{D-1}$$

$$W_k \in \mathbb{R}^{d_{k+1} \times d_k}$$

$$b_k \in \mathbb{R}^{d_{k+1}}$$



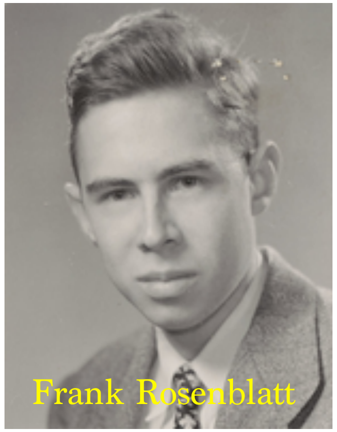
Multi-layer Perceptron



$$f_\theta(x_0) = x_D$$

$$x_0 \leftarrow x$$

$$x_{k+1} \triangleq \sigma(W_k x_k + b_k)$$

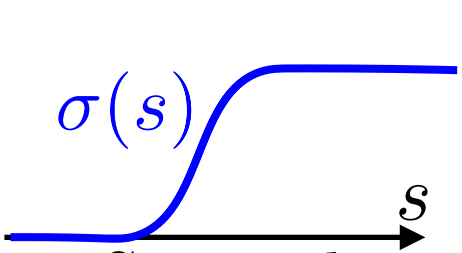


$$\theta = \{(W_k, b_k)\}_{k=0}^{D-1}$$

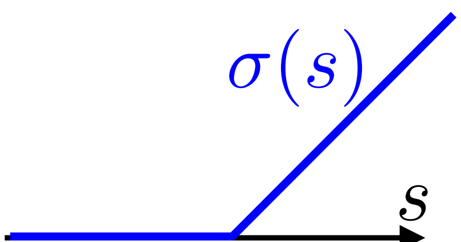
$$W_k \in \mathbb{R}^{d_{k+1} \times d_k}$$

$$b_k \in \mathbb{R}^{d_{k+1}}$$

Non-linearity: σ must be non-polynomial to increase expressivity.

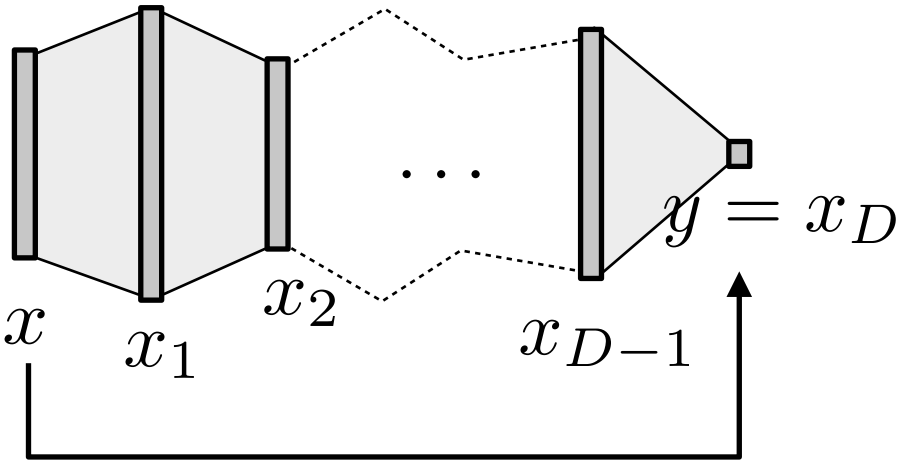


Sigmoid



ReLu

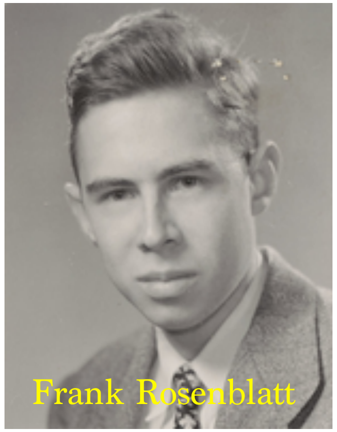
Multi-layer Perceptron



$$f_\theta(x_0) = x_D$$

$$x_0 \leftarrow x$$

$$x_{k+1} \triangleq \sigma(W_k x_k + b_k)$$

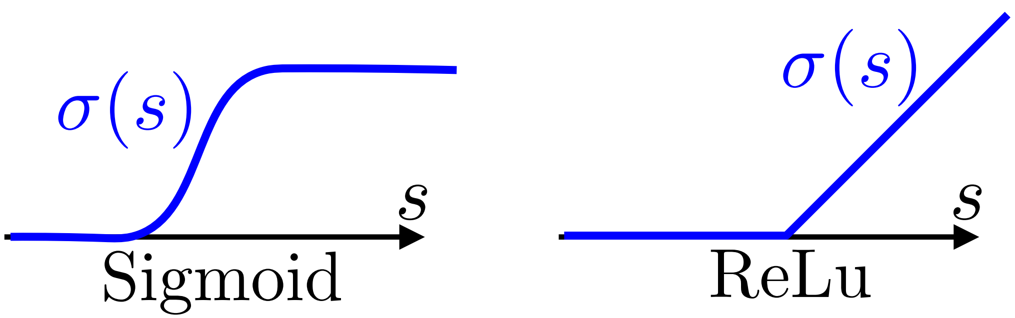


$$\theta = \{(W_k, b_k)\}_{k=0}^{D-1}$$

$$W_k \in \mathbb{R}^{d_{k+1} \times d_k}$$

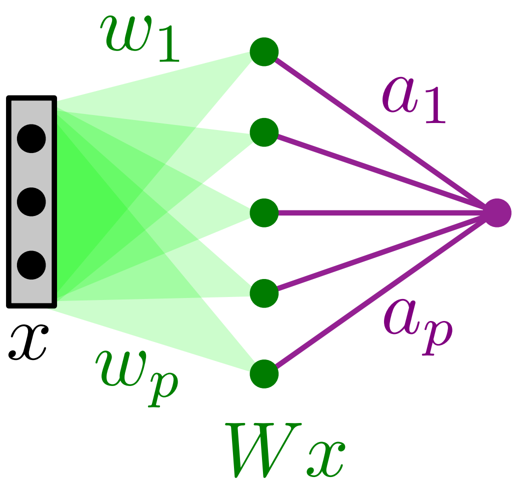
$$b_k \in \mathbb{R}^{d_{k+1}}$$

Non-linearity: σ must be non-polynomial to increase expressivity.



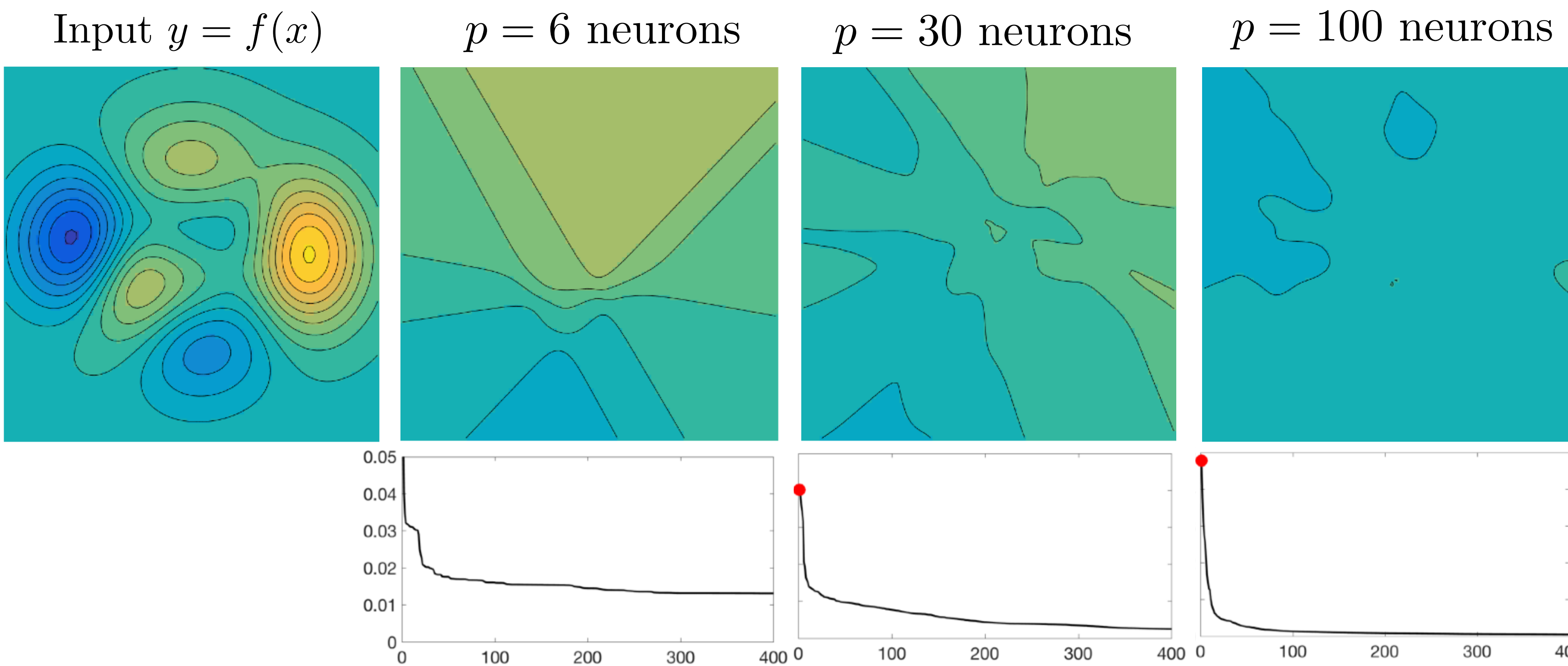
Weight matrix: needs extra constraints (e.g. convolution & sub-sampling)

Two Layers Perceptron

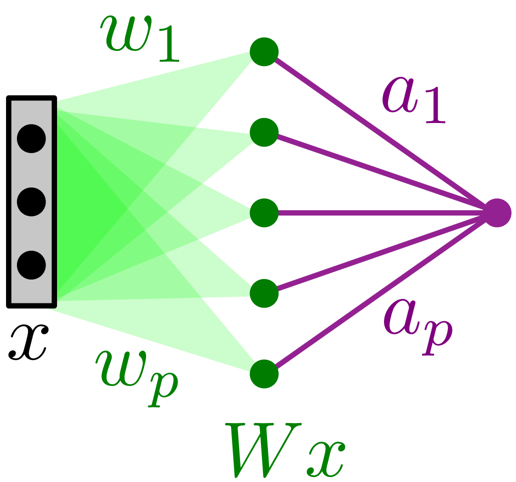


$$f_\theta(x) \triangleq \sum_{s=1}^{p} a_s \sigma(\langle x,\, w_s \rangle + b_s)$$

$\rightarrow$ sum of "ridge" functions $\sigma(\langle x,\, w \rangle + b)$



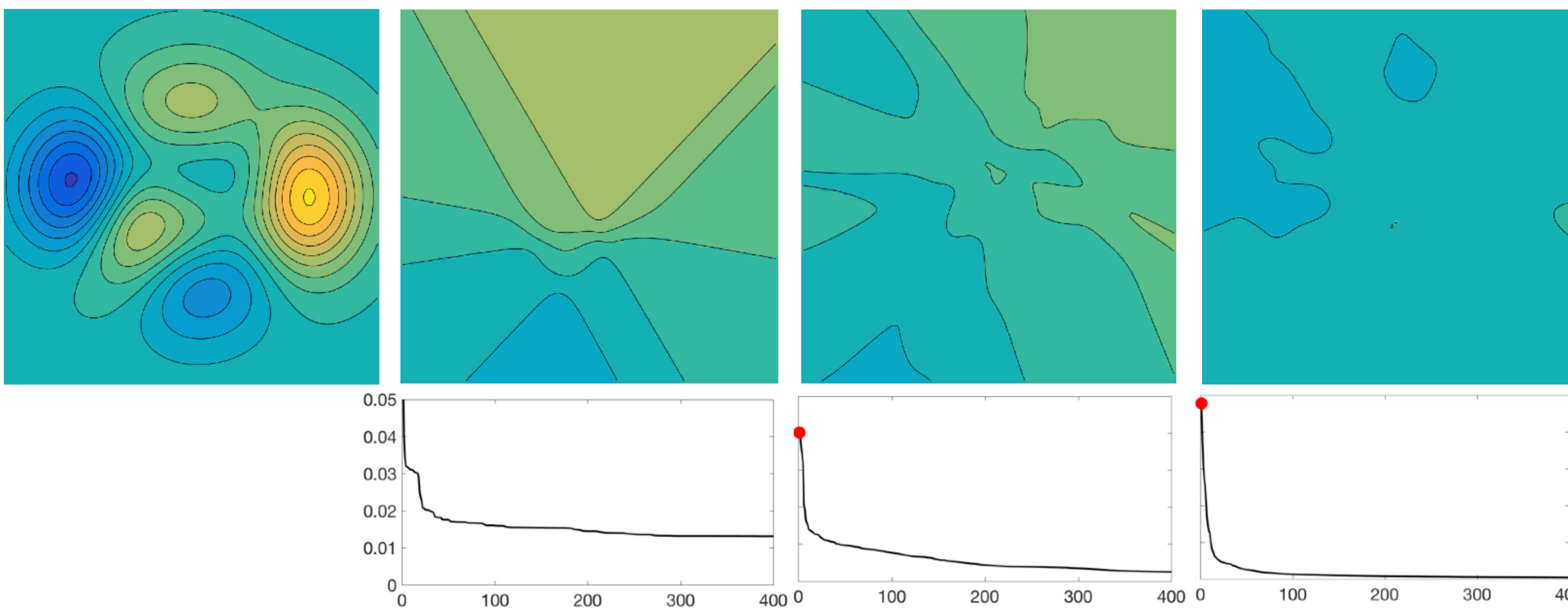
Two Layers Perceptron



$$f_\theta(x) \triangleq \sum_{s=1}^{p} a_s \sigma(\langle x,\, w_s \rangle + b_s)$$

$\rightarrow$ sum of "ridge" functions $\sigma(\langle x,\, w \rangle + b)$

Input $y = f(x)$ | $p = 6$ neurons | $p = 30$ neurons | $p = 100$ neurons



Universality of Perceptrons

Theorem: If f is continuous on a compact Ω, for all $\varepsilon > 0$ for p large enough, there exists θ such that

$$\forall\, x \in \Omega,\ |f_\theta(x) - f(x)| \leqslant \varepsilon$$

$\rightarrow$ non quantitative …no free lunch.

Universality of Perceptrons

Theorem: If f is continuous on a compact Ω, for all $\varepsilon > 0$ for p large enough, there exists θ such that

$$\forall\, x \in \Omega,\; |f_\theta(x) - f(x)| \leqslant \varepsilon$$

$\rightarrow$ non quantitative ... no free lunch.

Barron's functions: $\|f\|_B \triangleq \int_{\mathbb{R}^d} \|\omega\| |\hat{f}(\omega)| \mathrm{d}\omega < +\infty$

Theorem: for p large, there exists θ such that

$$\|f - f_\theta\|_{L^2(\Omega)} \leqslant \frac{2\mathrm{diam}(\Omega)\|f\|_B}{\sqrt{p}}$$

Universality of Perceptrons

Theorem: If f is continuous on a compact Ω, for all $\varepsilon > 0$ for p large enough, there exists θ such that

$$\forall\, x \in \Omega,\ |f_\theta(x) - f(x)| \leqslant \varepsilon$$

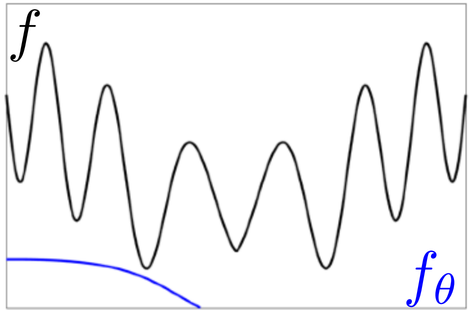
$\rightarrow$ non quantitative … no free lunch.

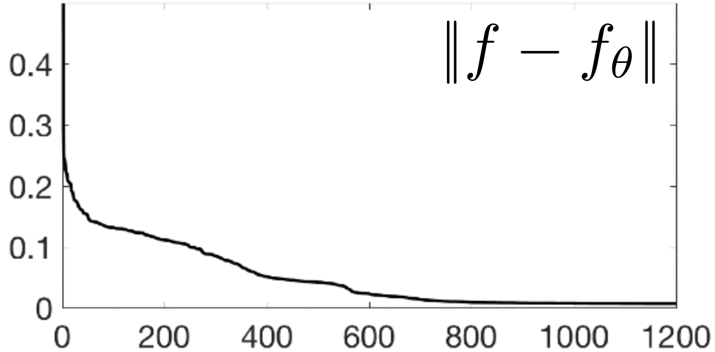
Barron's functions: $\|f\|_B \triangleq \int_{\mathbb{R}^d} \|\omega\| |\hat{f}(\omega)| \mathrm{d}\omega < +\infty$

Theorem: for p large, there exists θ such that

$$\|f - f_\theta\|_{L^2(\Omega)} \leqslant \frac{2\mathrm{diam}(\Omega)\|f\|_B}{\sqrt{p}}$$

$\rightarrow$ non-constructive.





$\rightarrow$ for p "large enough" gradient descent works [Chizat-Bach 2018]

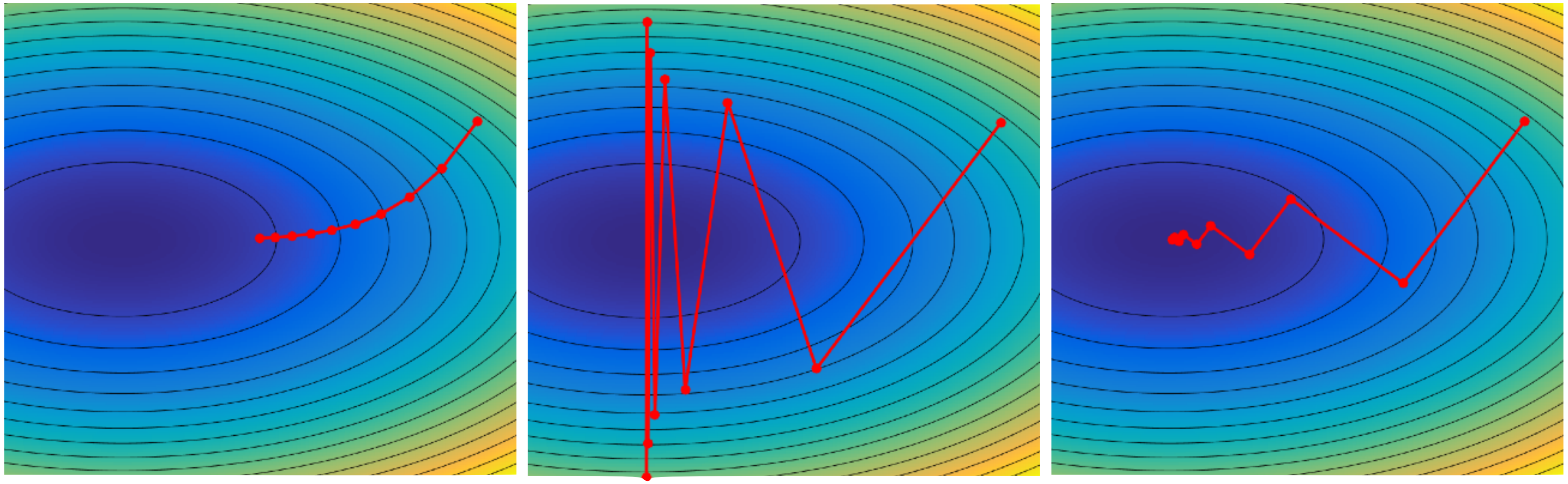
Overview

- Empirical Risk Minimization
- Perceptrons
- **Optimization**
- Convolutional Networks
- Residual Networks
- Transformers

Gradient Descent

$$\min_{\theta} \mathcal{E}(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

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



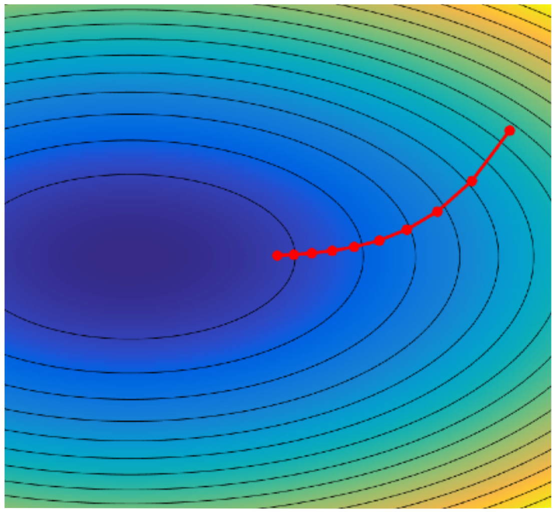
Small τ_ℓ | Large τ_ℓ | Optimal $\tau_\ell = \tau_\ell^\star$

Gradient Descent

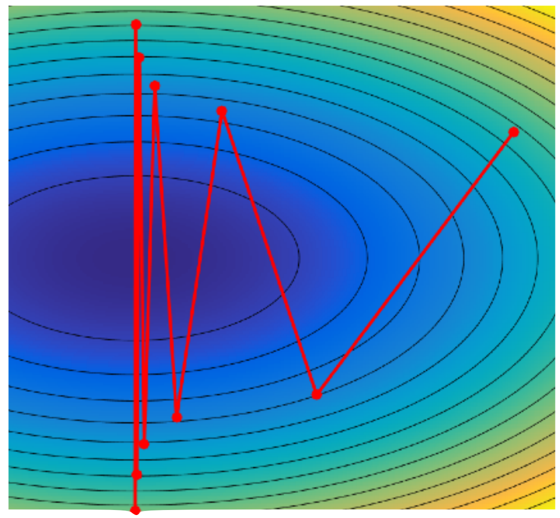
$$\min_{\theta} \mathcal{E}(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

Gradient descent:

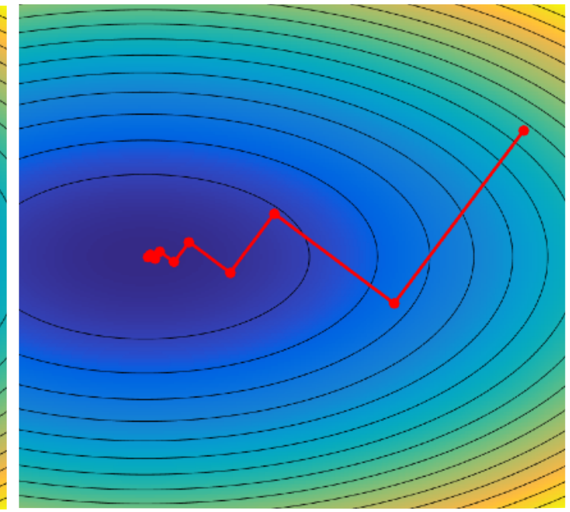
$$\theta_{\ell+1} = \theta_\ell - \tau_\ell \nabla \mathcal{E}(\theta_\ell)$$



Small τ_ℓ



Large τ_ℓ



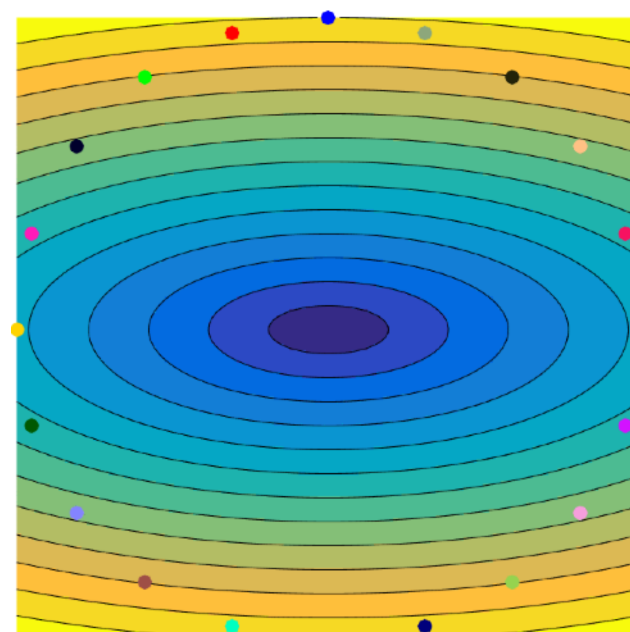
Optimal $\tau_\ell = \tau_\ell^\star$

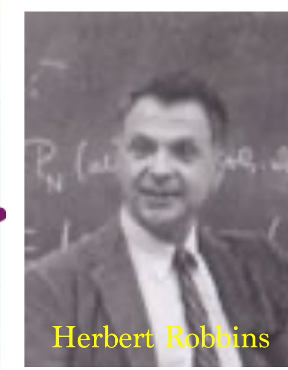
Stochastic gradient descent:

$$\theta_{\ell+1} = \theta_\ell - \frac{\tau}{\ell} \nabla \mathcal{E}_\ell(\theta_\ell)$$

$$i \leftarrow \texttt{rand}$$

$$\mathcal{E}_\ell(\theta) \triangleq \ell(f_\theta(x_i), y_i)$$





Sutton Monro

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^d \to \mathbb{R}$ 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 $\nabla \mathcal{E} : \mathbb{R}^d \to \mathbb{R}^d$?

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^d \to \mathbb{R}$ 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 $(a \times b, \log(a), \sqrt{a} \dots)$ and their derivatives cost $O(1)$.

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

Finite differences:

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

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

The Complexity of Gradient Computation

Setup: $\mathcal{E} : \mathbb{R}^d \to \mathbb{R}$ 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 $\nabla\mathcal{E} : \mathbb{R}^d \to \mathbb{R}^d$?

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

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

This algorithm is reverse mode automatic differentiation

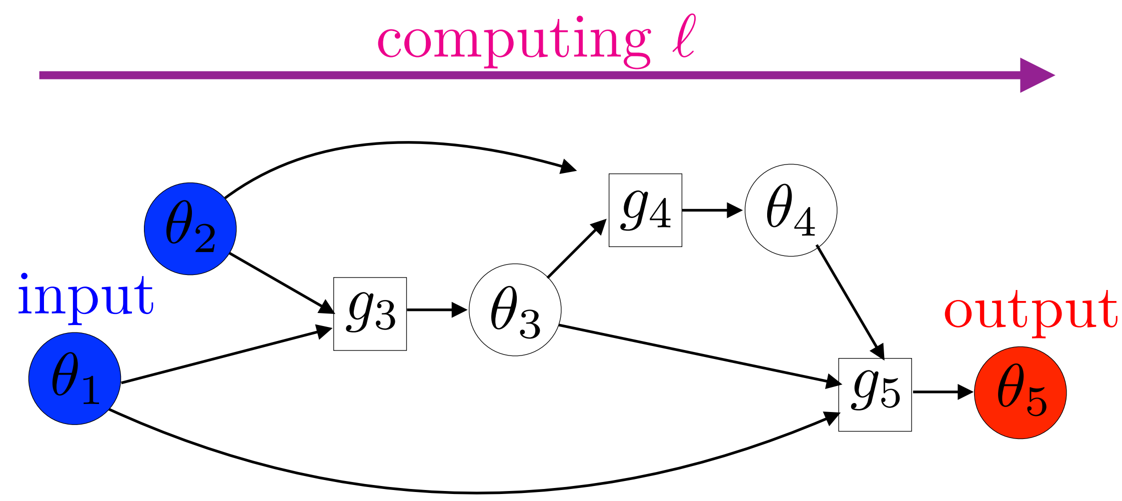
```
def BackwardNN(A,b,X):
    gx = lossG(X[R],Y) # initialize the gradient
    for r in arange(R-1,-1,-1):
        M = rhoG( A[r].dot(X[r]) + tile(b[r],[1,n]) ) * gx
        gx = A[r].transpose().dot(M)
        gA[r] =  M.dot(X[r].transpose())
        gb[r] =  MakeCol(M.sum(axis=1))
    return [gA,gb]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

Computational Graph

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

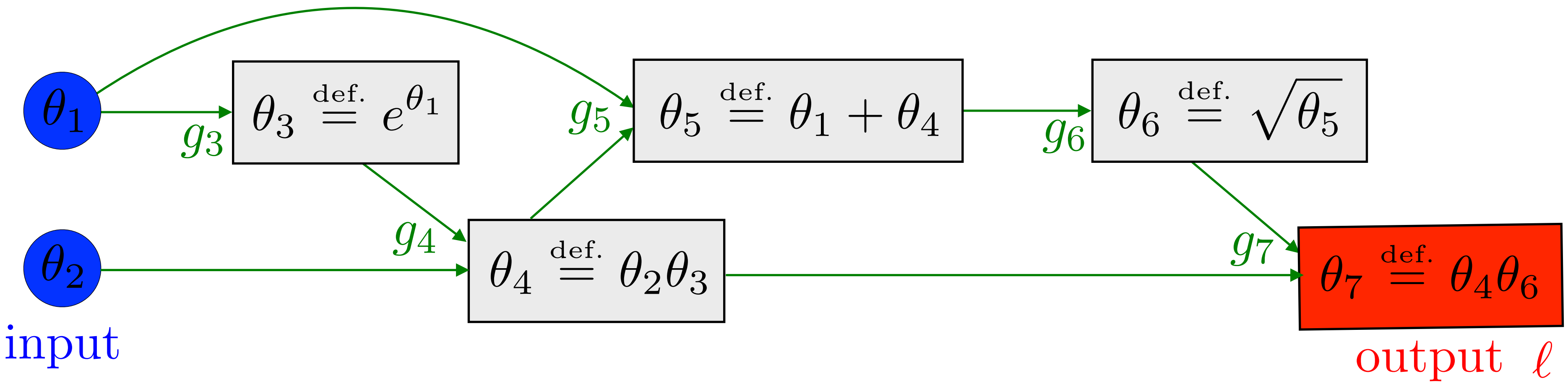
forward

```
function ℓ(θ_1, ..., θ_M)
  for r = M+1, ..., R
  |   θ_r = g_r(θ_Parents(r))
  return θ_R
```



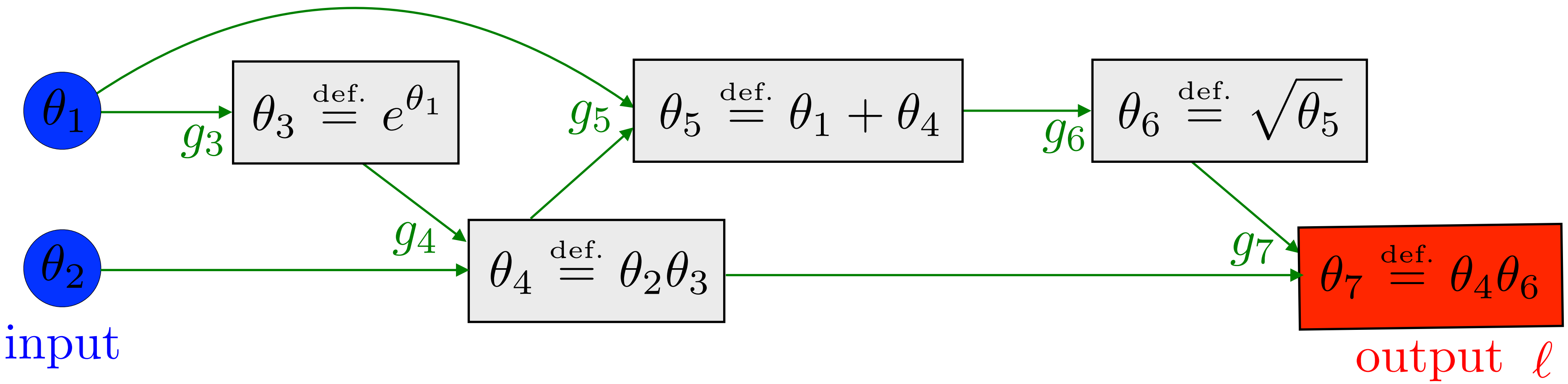
Example

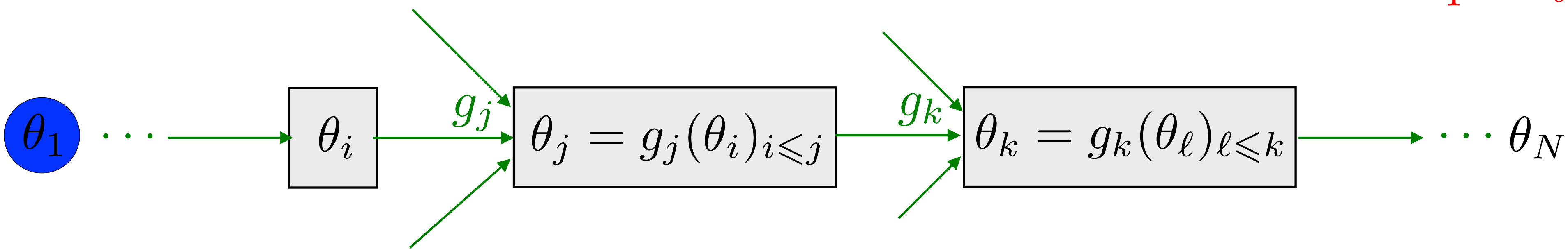
$$\ell(\theta_1, \theta_2) \stackrel{\text{def.}}{=} \theta_2 e^{\theta_1} \sqrt{\theta_1 + \theta_2 e^{\theta_1}}$$



Example

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





Chain rules:

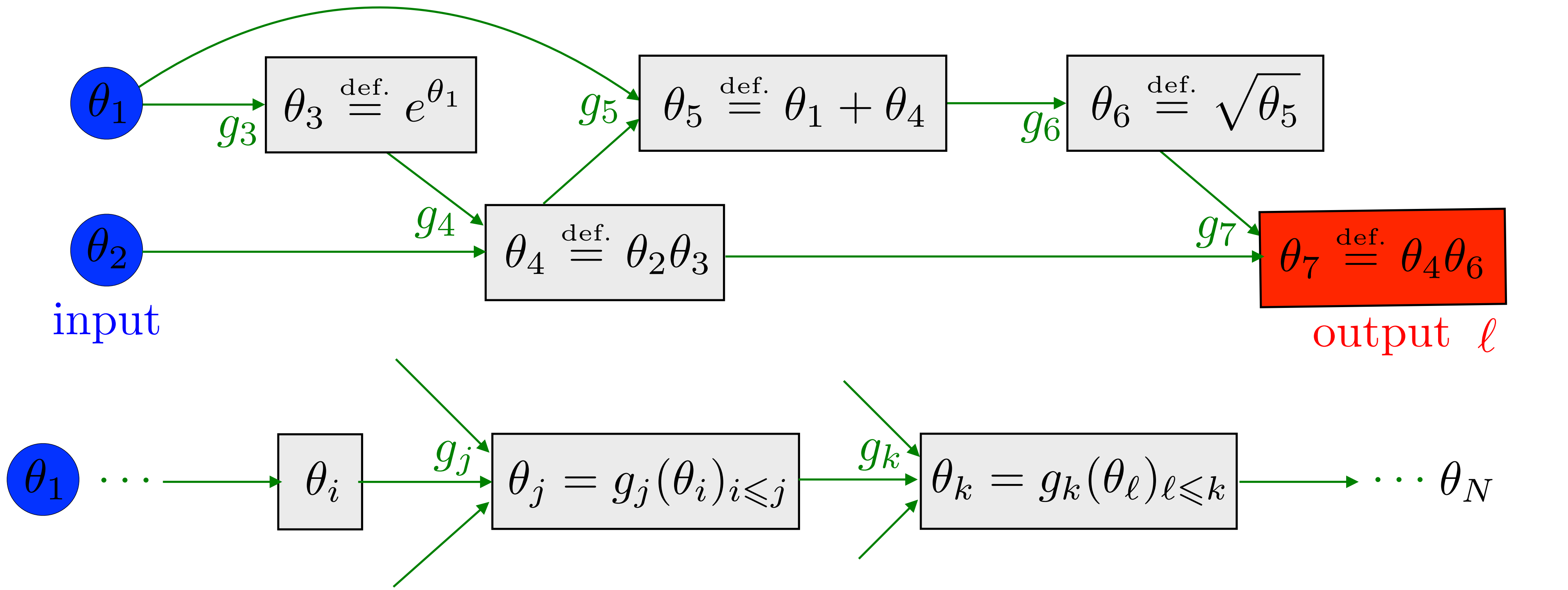
$$\text{“}\frac{\partial \theta_j}{\partial \theta_1} = \sum_{i \in \text{Parent}(j)} \frac{\partial \theta_j}{\partial \theta_i} \frac{\partial \theta_i}{\partial \theta_1}\text{”}$$

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

“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:

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

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

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

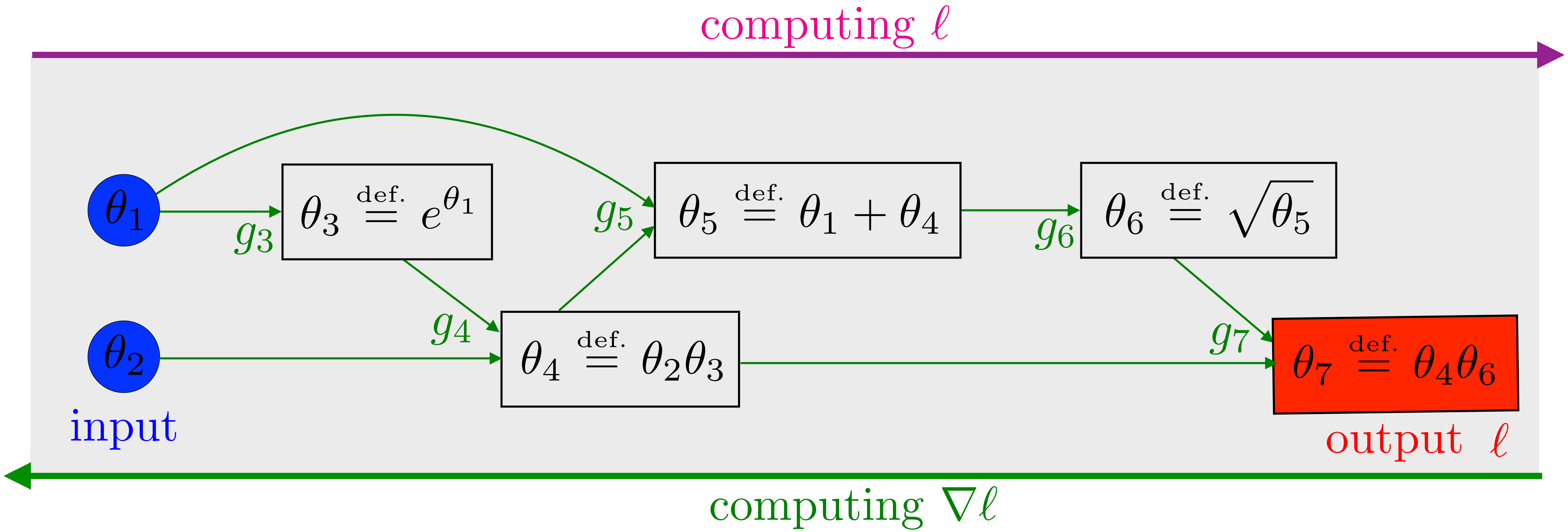
$$\text{“} \frac{\partial \theta_N}{\partial \theta_j} = \sum_{k \in \text{Child}(j)} \frac{\partial \theta_N}{\partial \theta_k} \frac{\partial \theta_k}{\partial \theta_j} \text{”}$$

$\frac{\partial \theta_N}{\partial \theta_j} \to \nabla_j \ell(\theta)$, $\frac{\partial \theta_N}{\partial \theta_k} \to \nabla_k \ell(\theta)$, $\frac{\partial \theta_k}{\partial \theta_j} \to \partial_j g_k(\theta)$

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

Backward Automatic Differentiation

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



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

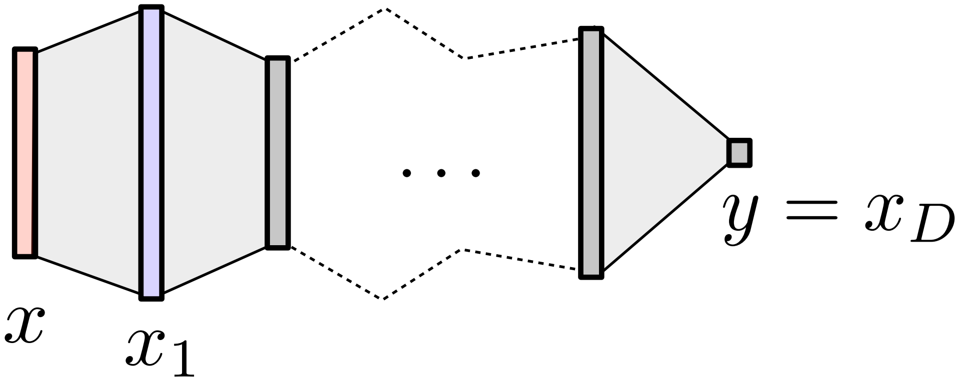
backward

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 $(\nabla_1 \ell, \ldots, \nabla_M \ell)$

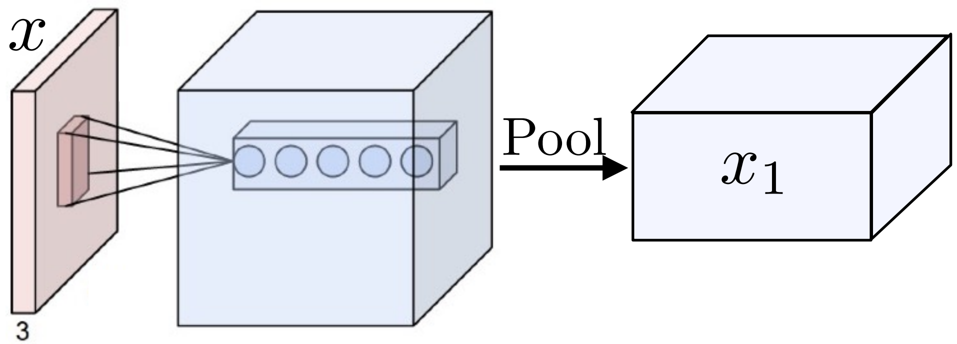
Overview

- Empirical Risk Minimization
- Perceptrons
- Optimization
- **Convolutional Networks**
- Residual Networks
- Transformers

Convolutional CNN



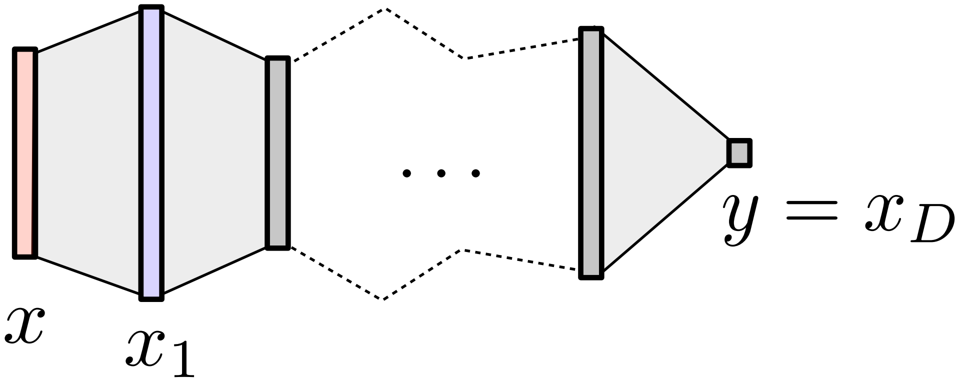
$$x_{k+1} \triangleq \sigma(W_k x_k + b_k)$$



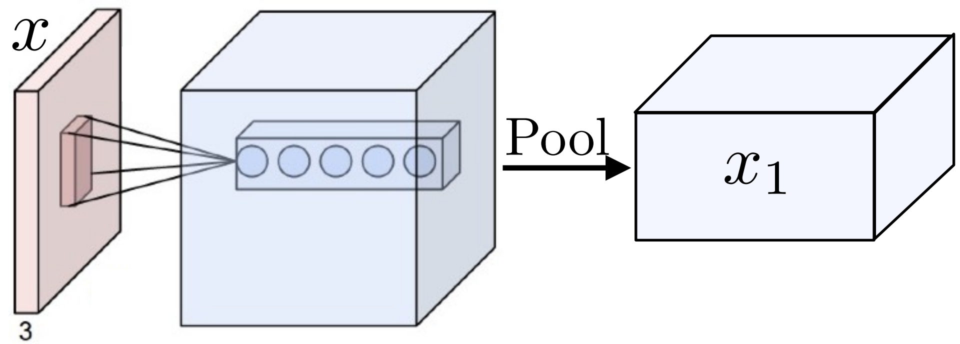
→ Leverage translation invariance of images.

→ Sub-sampling: breaks invariance but increase receptive fields.

Convolutional CNN

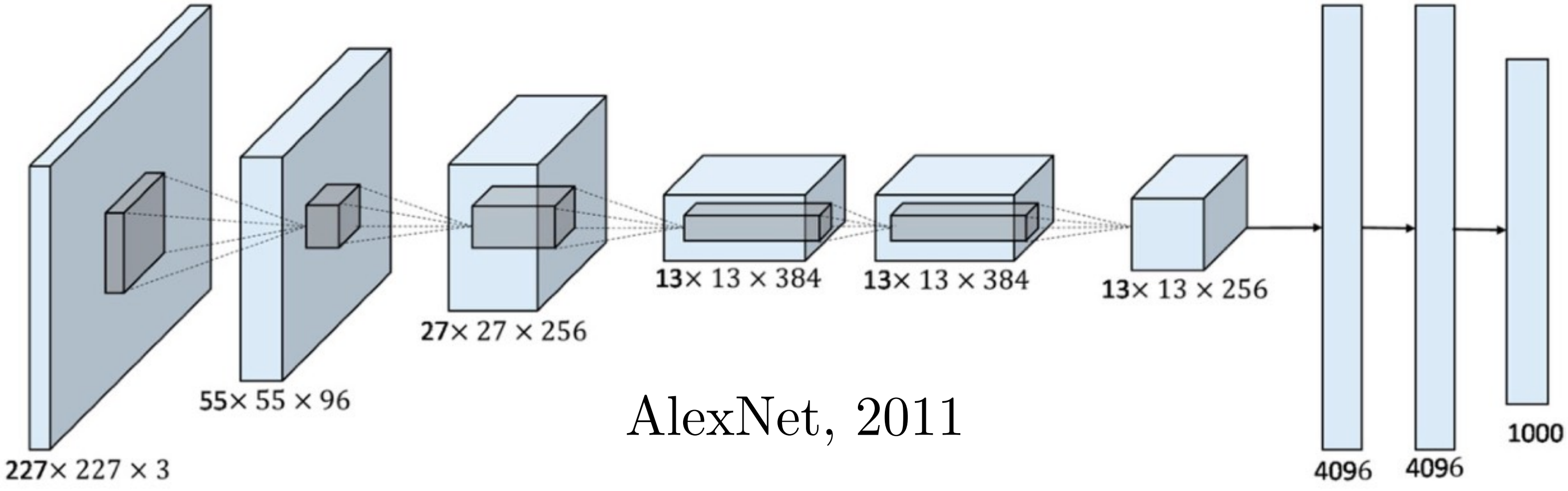


$$x_{k+1} \triangleq \sigma(W_k x_k + b_k)$$



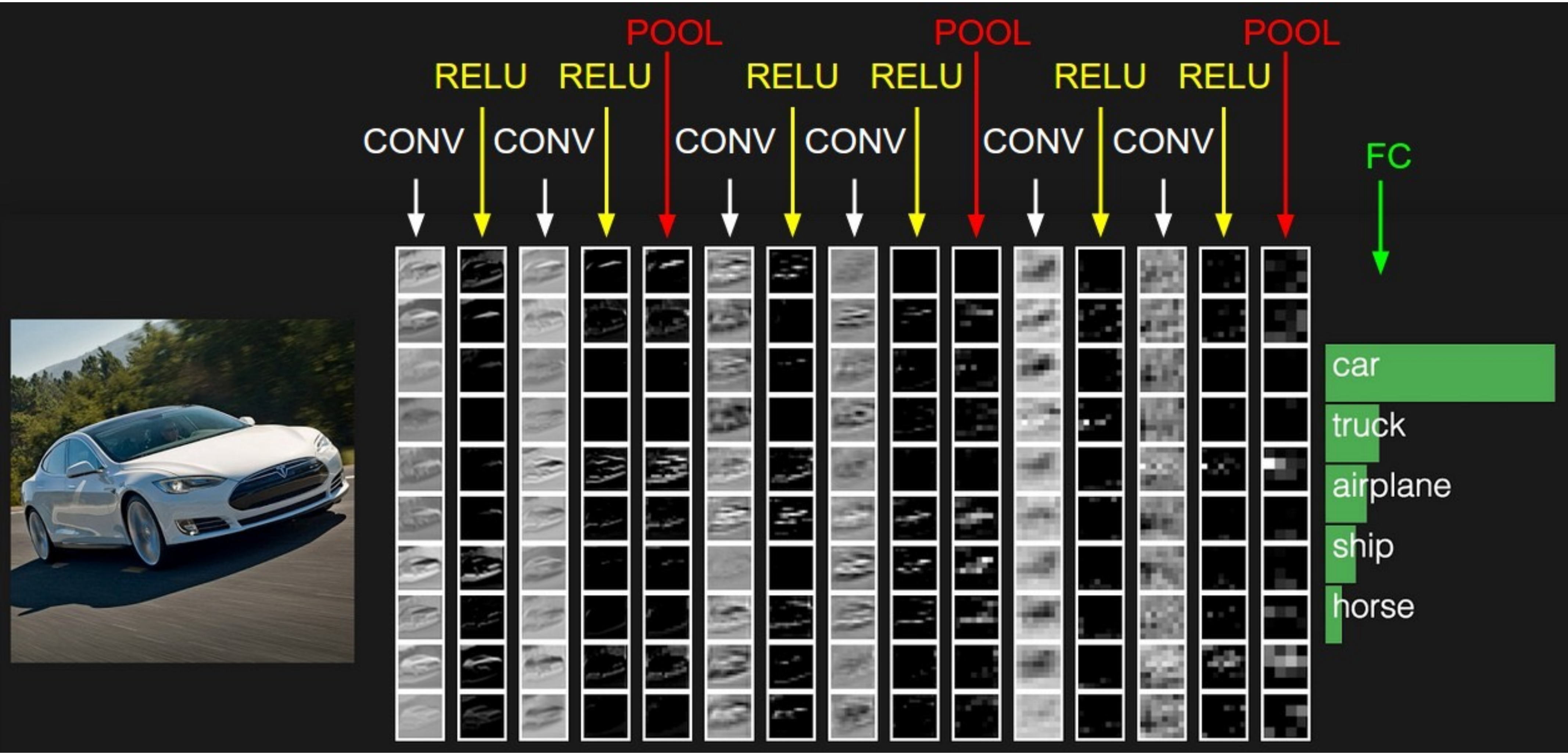
→ Leverage translation invariance of images.

→ Sub-sampling: breaks invariance but increase receptive fields.



AlexNet, 2011

Example of Activations

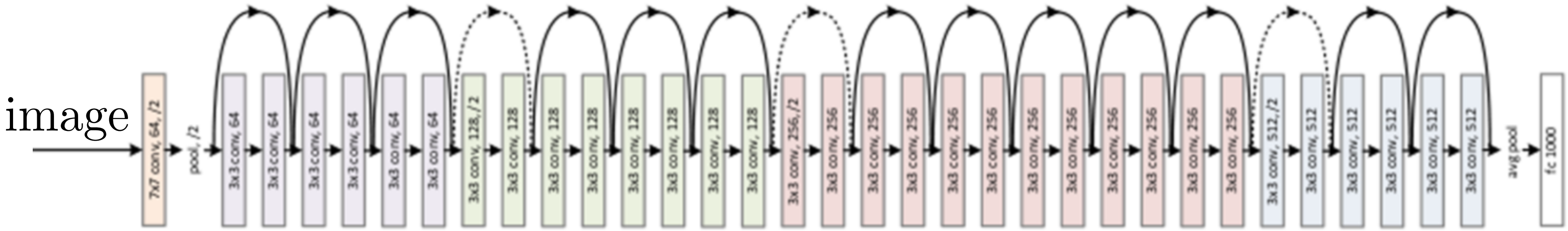


Overview

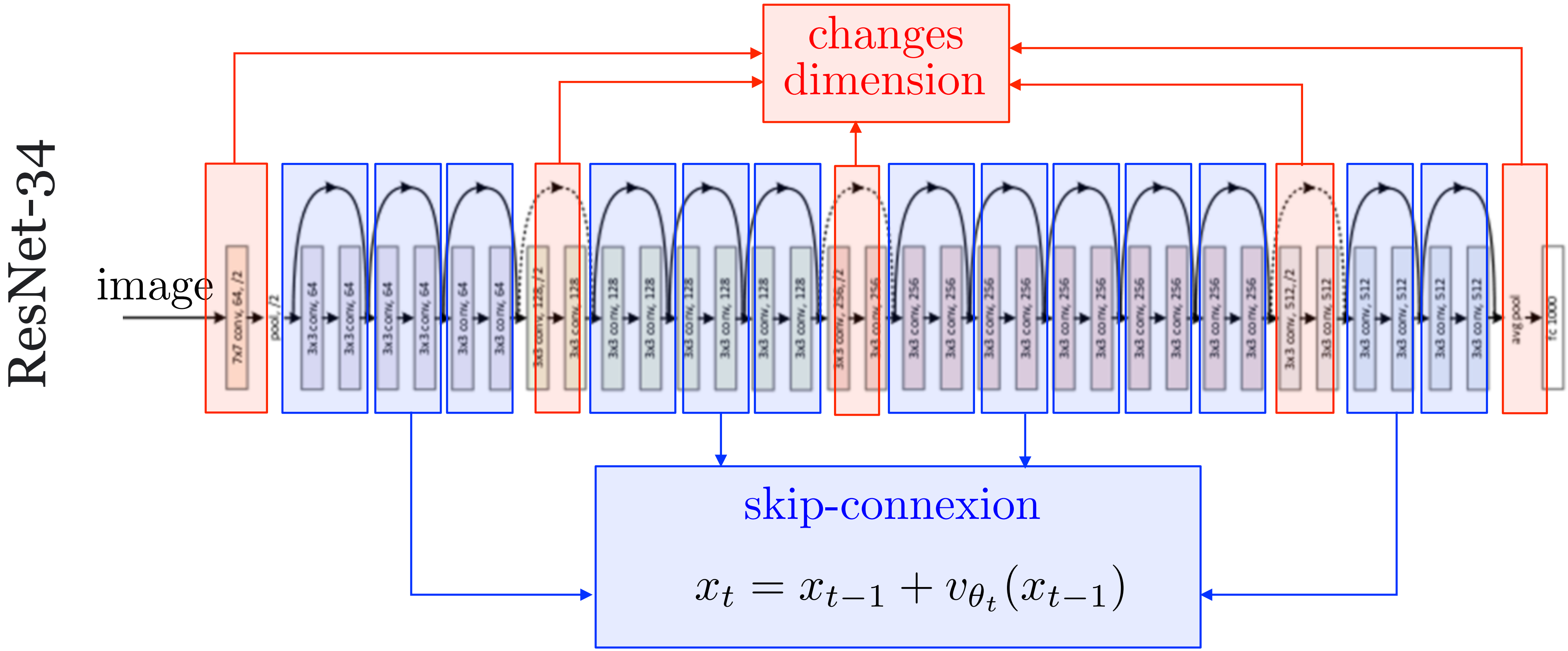
- Empirical Risk Minimization
- Perceptrons
- Optimization
- Convolutional Networks
- **Residual Networks**
- Transformers

ResNet-type Architectures [He et al' 16]

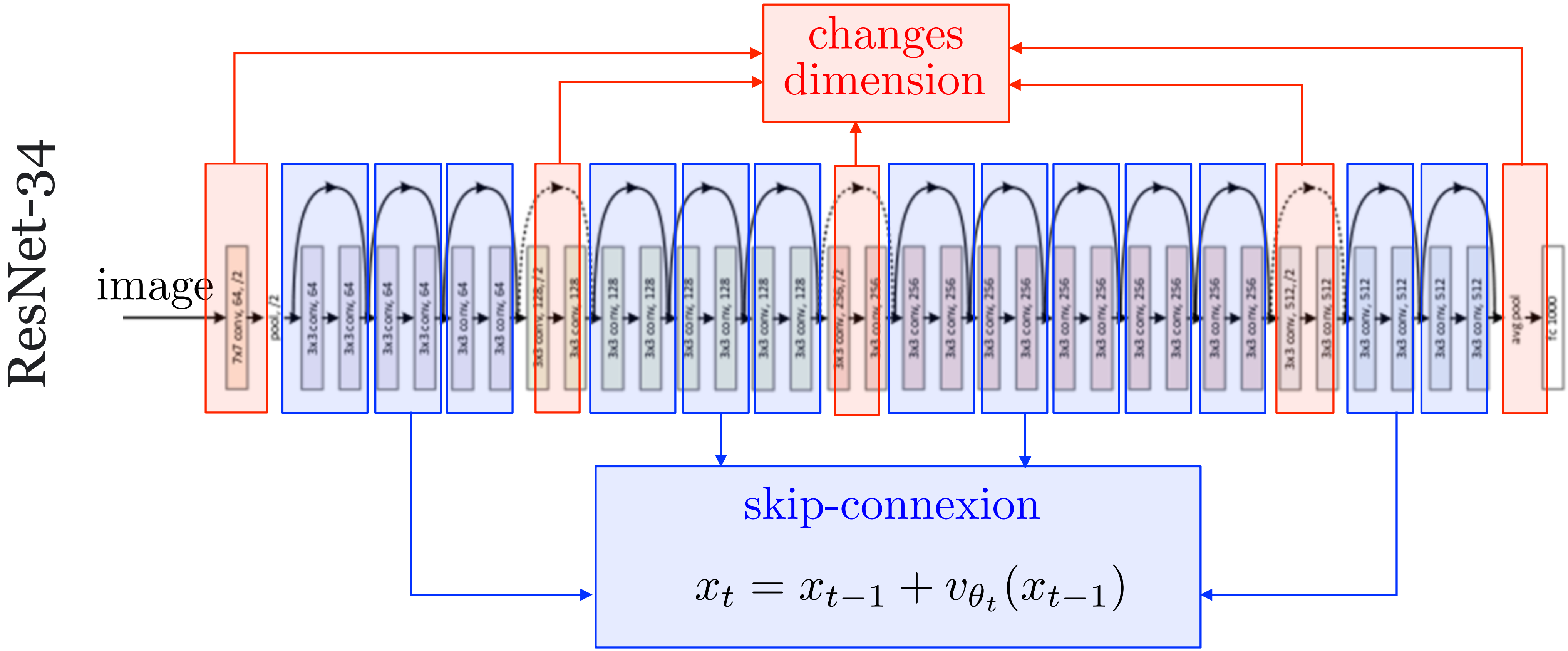
ResNet-34



ResNet-type Architectures [He et al' 16]

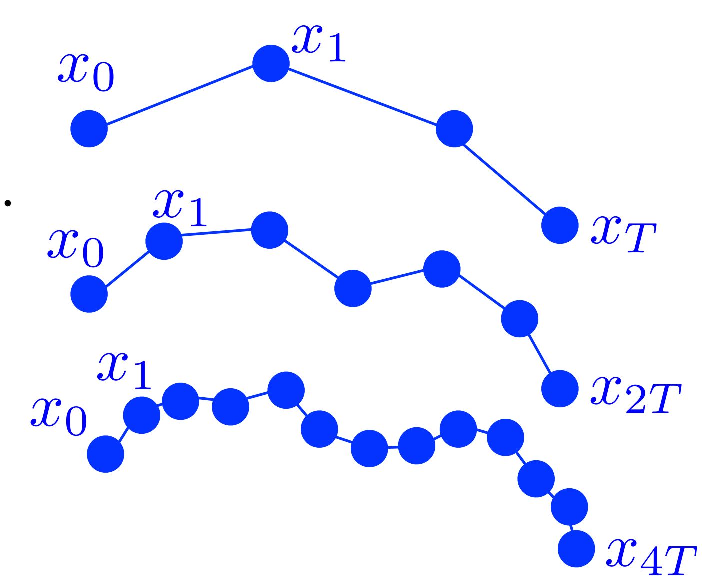


ResNet-type Architectures [He et al’ 16]



→ Makes the “infinite depth” limit non-degenerate.

→ Enable $v_\theta = 0$ initialization, i.e. identity map.

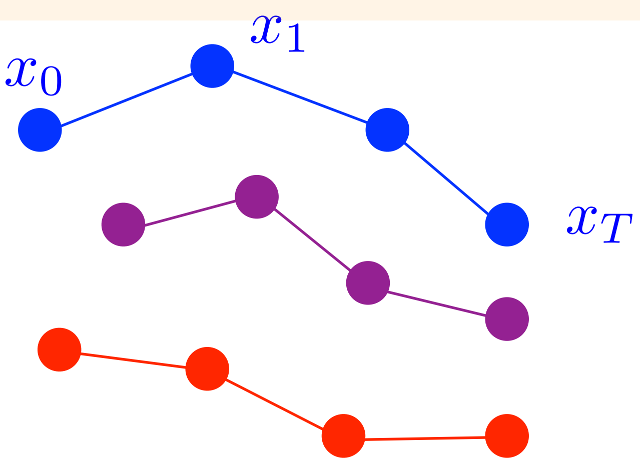


Infinite Depth and Neural-ODEs

ResNet [He et al, 2016]

$$\Phi_\theta(x_0) \triangleq x_T \quad \text{where}$$

$$x_{t+1} = x_t + \frac{1}{T} v_{\theta_t}(x_t)$$



Infinite Depth and Neural-ODEs

ResNet [He et al, 2016]

$$\Phi_\theta(x_0) \triangleq x_T \quad \text{where}$$

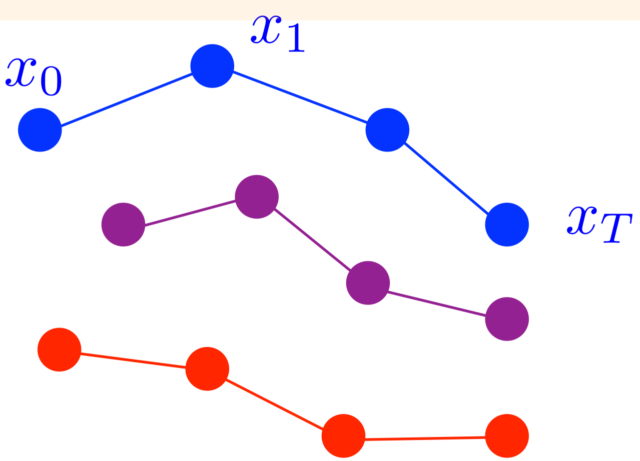
$$x_{t+1} = x_t + \frac{1}{T} v_{\theta_t}(x_t)$$

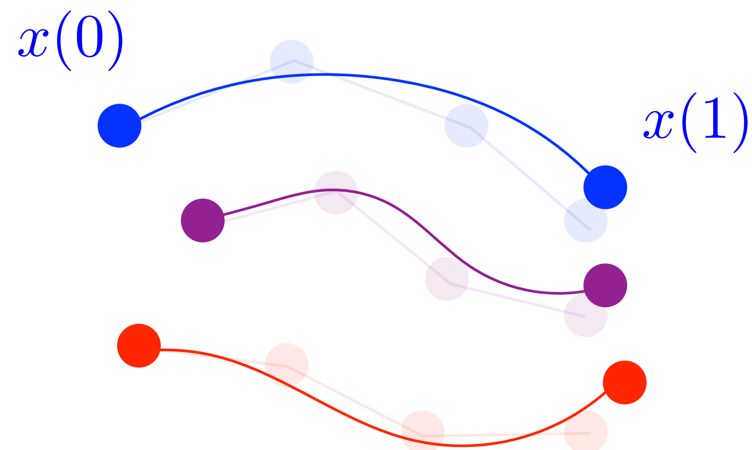
$T \to +\infty$

Neural ODE [Chen et al, 2018]

$$\Phi_\theta(x(0)) \triangleq x(1) \quad \text{where}$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v_{\theta(t)}(x(t))$$





Infinite Depth and Neural-ODEs

ResNet [He et al, 2016]

$$\Phi_\theta(x_0) \triangleq x_T \quad \text{where}$$

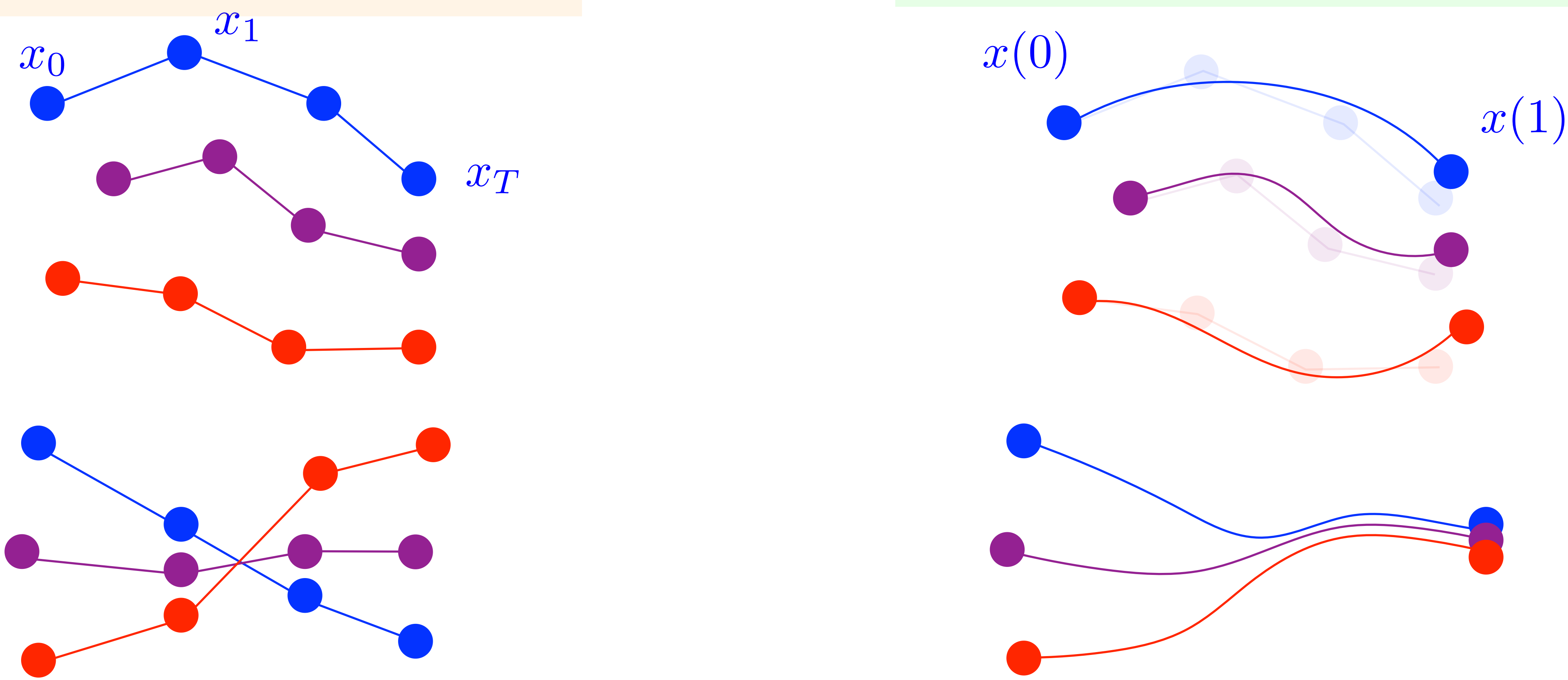
$$x_{t+1} = x_t + \frac{1}{T} v_{\theta_t}(x_t)$$

$T \to +\infty$

Neural ODE [Chen et al, 2018]

$$\Phi_\theta(x(0)) \triangleq x(1) \quad \text{where}$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v_{\theta(t)}(x(t))$$



Trajectories cannot cross: Φ_θ defines a diffeomorphism.

$T \to +\infty$ is a singular limit (θ can “explodes” during training)

On the importance of scale and initialization

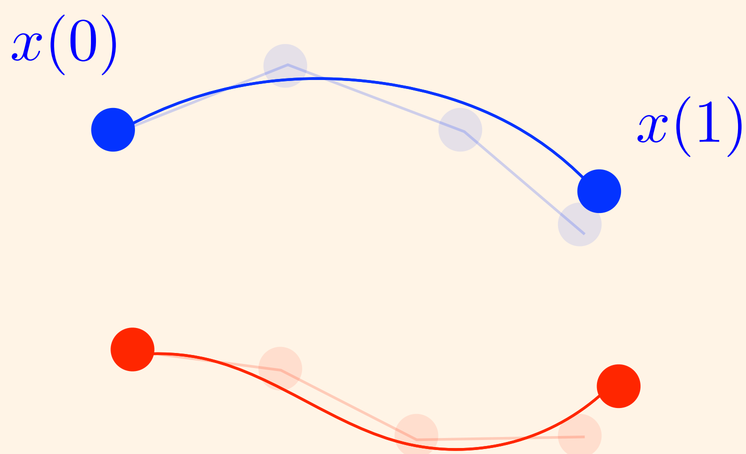
$$x_{t+1} = x_t + \frac{1}{T}\, v_{\theta_t}(x_t)$$

Zero/smooth initialization of $(\theta_t)_t$

$\downarrow$ $T \to +\infty$

Deterministic ODE

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v_{\theta(t)}(x(t))$$



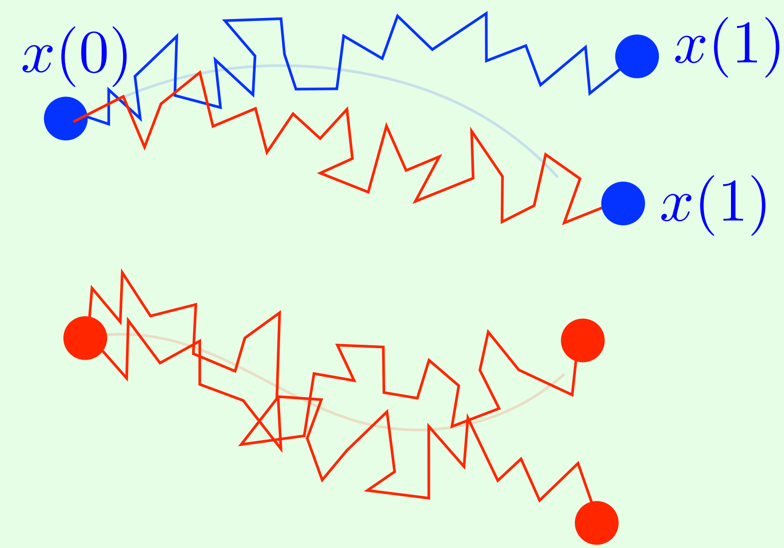
$$x_{t+1} = x_t + \frac{1}{\sqrt{T}}\, v_{\theta_t}(x_t)$$

Random initialization of $(\theta_t)_t$

$\downarrow$ $T \to +\infty$

Stochastic ODE

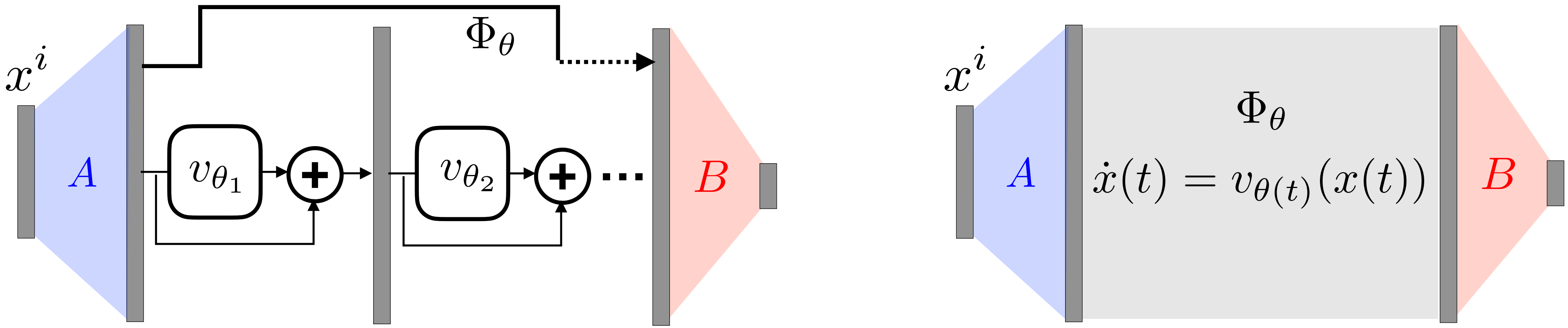
$$\mathrm{d}x(t) = v_{\theta(t)}(x(t))\mathrm{d}t + \mathrm{dW}(t)$$



[R. Cont, A. Rossier, R. Xu, 2022]

[P. Marion, Fermanian, Biau, Vert, 2022]

Training Dynamic

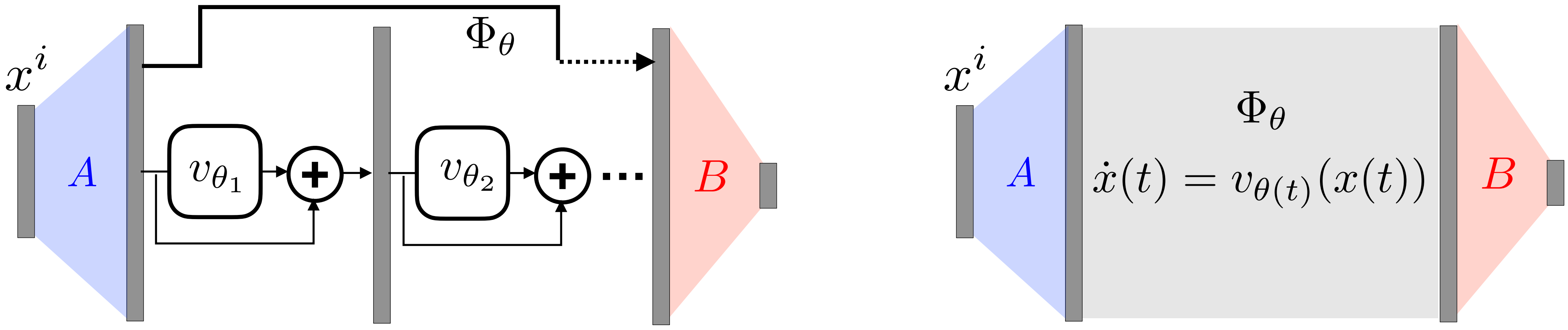


Training: $\min_\theta f(\theta) \triangleq \frac{1}{N} \sum_{i=1}^N \|B\Phi_\theta(Ax^i) - y^i\|^2$

Gradient descent: $\theta^{(k+1)} = \theta^{(k)} - \tau \nabla f(\theta^{(k)})$

$\rightarrow$ **No explicit regularization!**

Training Dynamic



Training: $\min_\theta f(\theta) \triangleq \frac{1}{N}\sum_{i=1}^N \|B\Phi_\theta(Ax^i) - y^i\|^2$

Gradient descent: $\theta^{(k+1)} = \theta^{(k)} - \tau\nabla f(\theta^{(k)})$

$\rightarrow$ **No explicit regularization!**

Question: convergence of θ^k toward global minimum?

Neural tangent kernel [Jacot et al'18]: local linear expansion.

Polyak-Łojasiewicz inequality [Liu, Zhu, Belkin 2021]:

$\rightarrow$ conditionning might explodes as $T \to +\infty$.

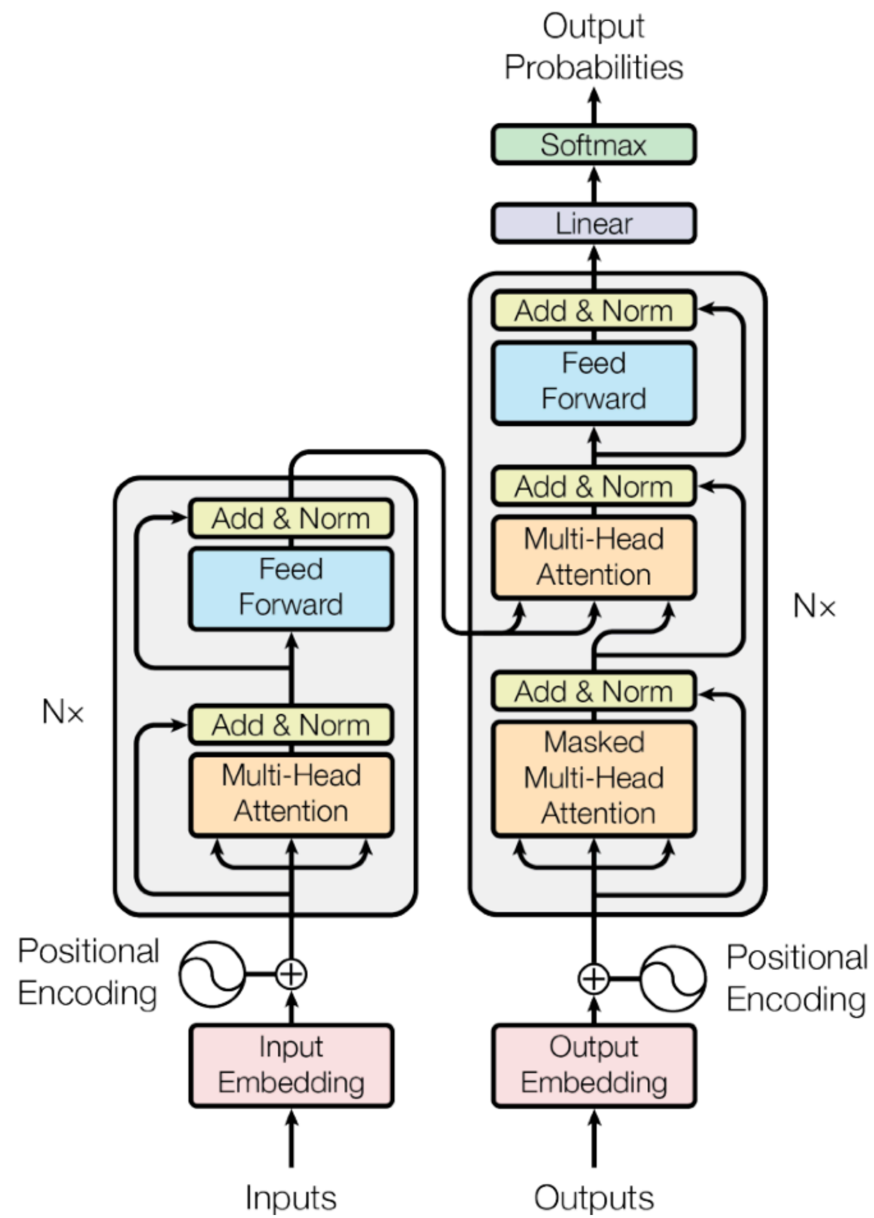
$\rightarrow$ find a suitable limit model and show "implicit" regularization effect.

Simplified analysis: [Barboni et al. 2022]

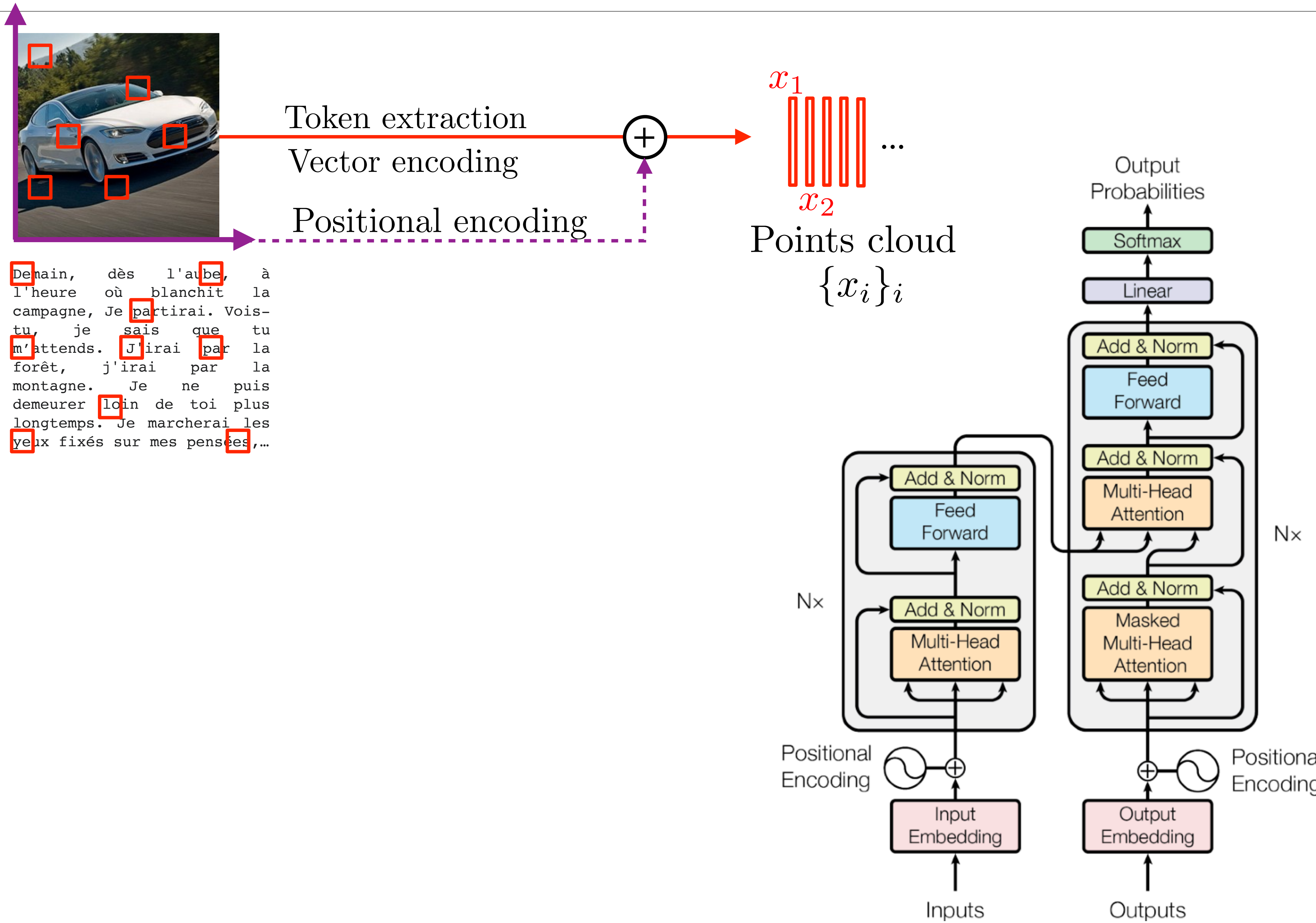
Overview

- Empirical Risk Minimization
- Perceptrons
- Optimization
- Convolutional Networks
- Residual Networks
- **Transformers**

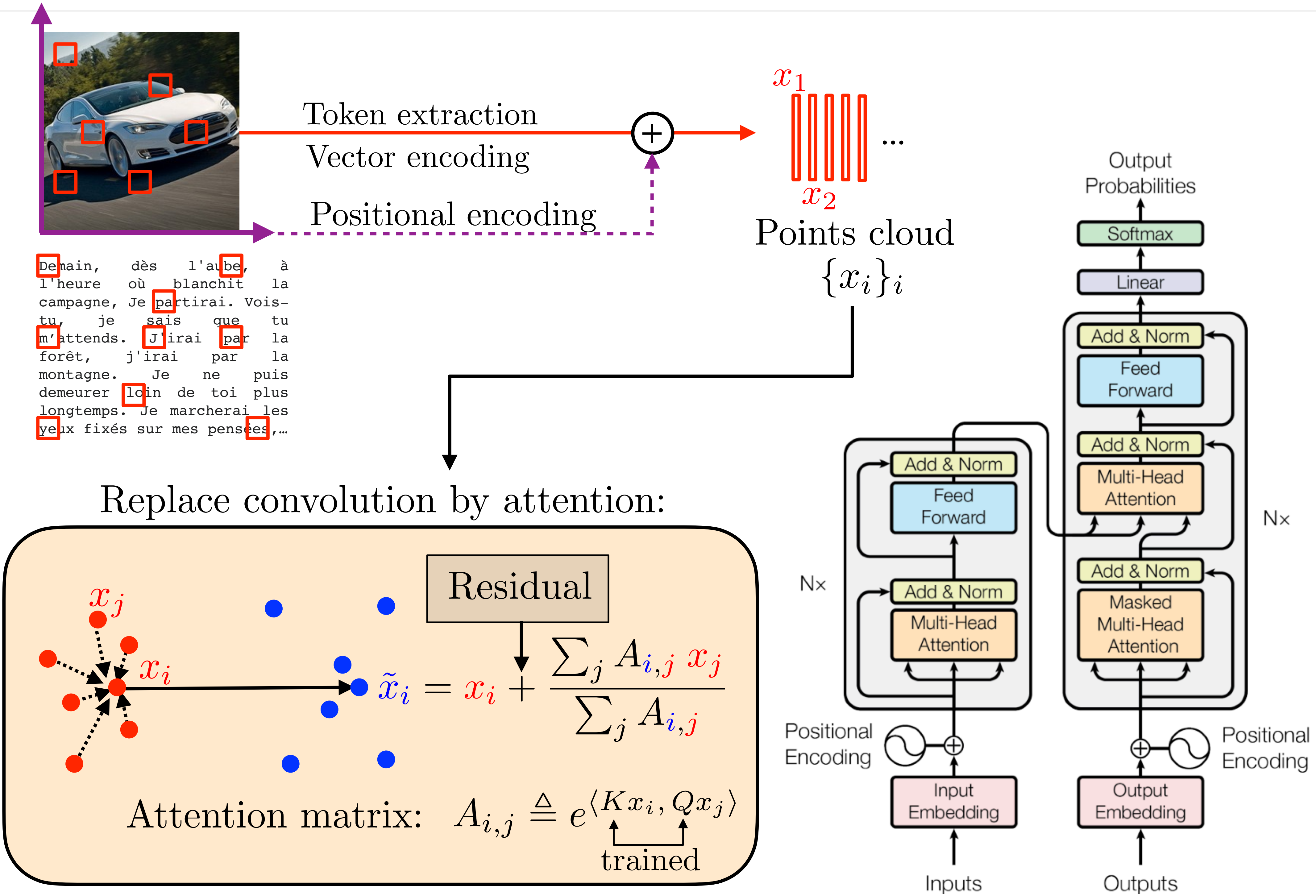
Transformers



Transformers



Transformers



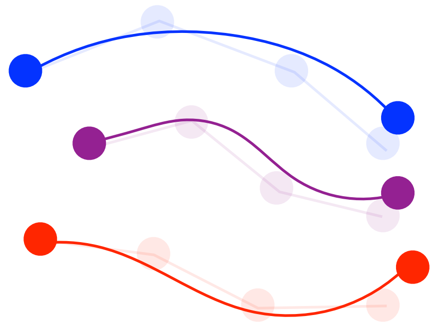
Conclusion

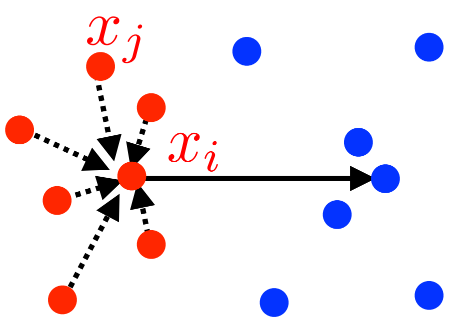
Strong connexion with mathematical concepts:

– Going wider $\sim$ function approximation.

– Going deeper $\sim$ differential equations.

– Attention $\sim$ interacting particles.





Conclusion

Strong connexion with mathematical concepts:

– Going wider ∼ function approximation.

– Going deeper ∼ differential equations.

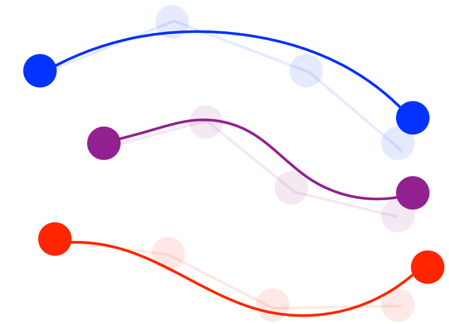
– Attention ∼ interacting particles.

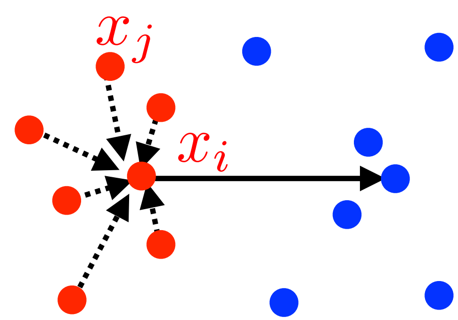
Very limited theoretical understanding:

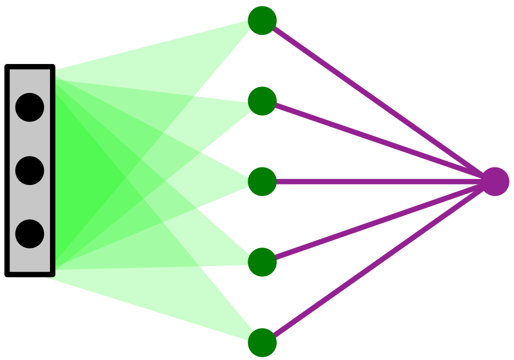
– Why gradient descent works?

– Implicit bias of architectures.

– Implicit bias of optimizers.







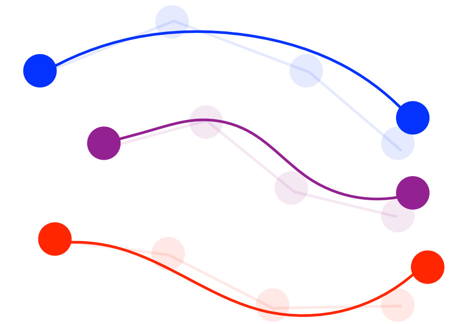
Conclusion

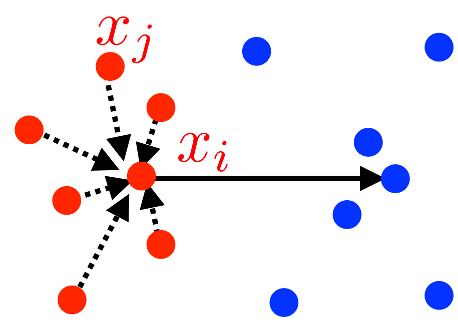
Strong connexion with mathematical concepts:

– Going wider ∼ function approximation.

– Going deeper ∼ differential equations.

– Attention ∼ interacting particles.



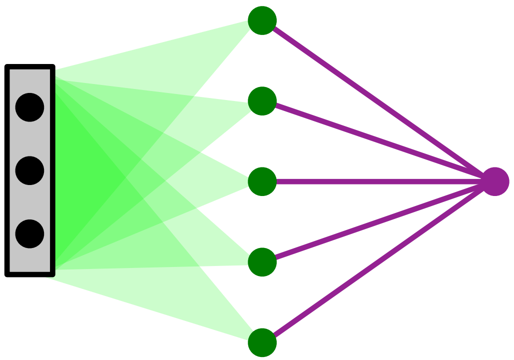


Very limited theoretical understanding:

– Why gradient descent works?

– Implicit bias of architectures.

– Implicit bias of optimizers.



Examples of open problems:

– Why some optimizers (e.g. Adam) works for transformers?

– Mean field analysis of transformers (optimal transport?)