IFT6390
Fondements de l'apprentissage machine

Linear regression
and
Regularized linear regression

Professor: Ioannis Mitliagkas
Slides: Pascal Vincent

Supervised task

predict y from x

input $\mathbf{x} \in \mathbb{R}^d$

target (label) $\mathbf{y}$

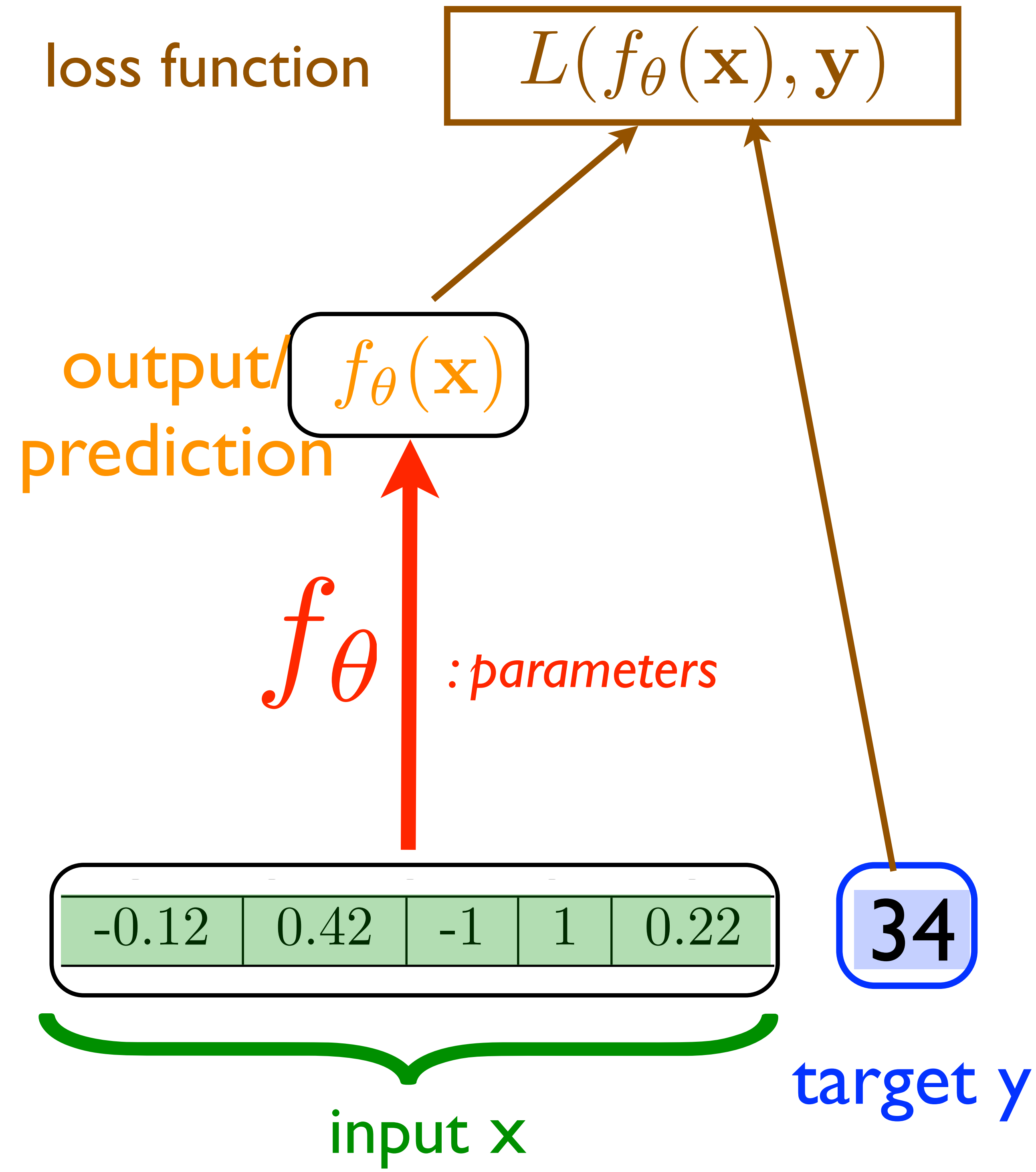
n examples

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	t
0.32	-0.27	+1	0	0.82	113
-0.12	0.42	-1	1	0.22	34
0.06	0.35	-1	1	-0.37	56
0.91	-0.72	+1	0	-0.63	77
...	...	...	...	...	...

Training set Dn

$$D_n = \{(x^{(1)}, t^{(1)}), \ldots, (x^{(n)}, t^{(n)})\}$$

Learn a function $f = f_\theta$ which minimizes a cost (loss).



Empirical risk minimization

We must specify:

- A parametric form for our functions, f_θ
- A specific cost (loss) function $L(y,t)$

So we define the empirical risk as:

$$\hat{R}(f_\theta, D_n) \quad = \quad \sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})$$

i.e. total loss on the training set

Learning amounts to finding the optimal values for the parameters:

$$\theta^\star \quad = \quad \arg\min_{\theta} \hat{R}(f_\theta, D_n)$$

It is the principle of empirical risk minimization.

Eg: Linear regression

A very simple learning algorithm

We select

A linear (affine) form for the function:

$$f_\theta(\mathbf{x}) = \underbrace{\langle \mathbf{w}, \mathbf{x} \rangle}_{\text{scalar product (inner product)}} + b$$

$$\theta = \{\mathbf{w}, b\}, \underbrace{\mathbf{w}}_{\text{"weight vector"}} \in \mathbb{R}^d, \underbrace{b}_{\text{bias}} \in \mathbb{R}$$

Cost: quadratic error:

$$L(y,t) = (y-t)^2$$

Principle of empirical risk minimization (ERM)

We look for the parameters that minimize the empirical risk

$$\theta^\star = \arg\min_\theta \hat{R}(f_\theta, D_n)$$

Linear regression

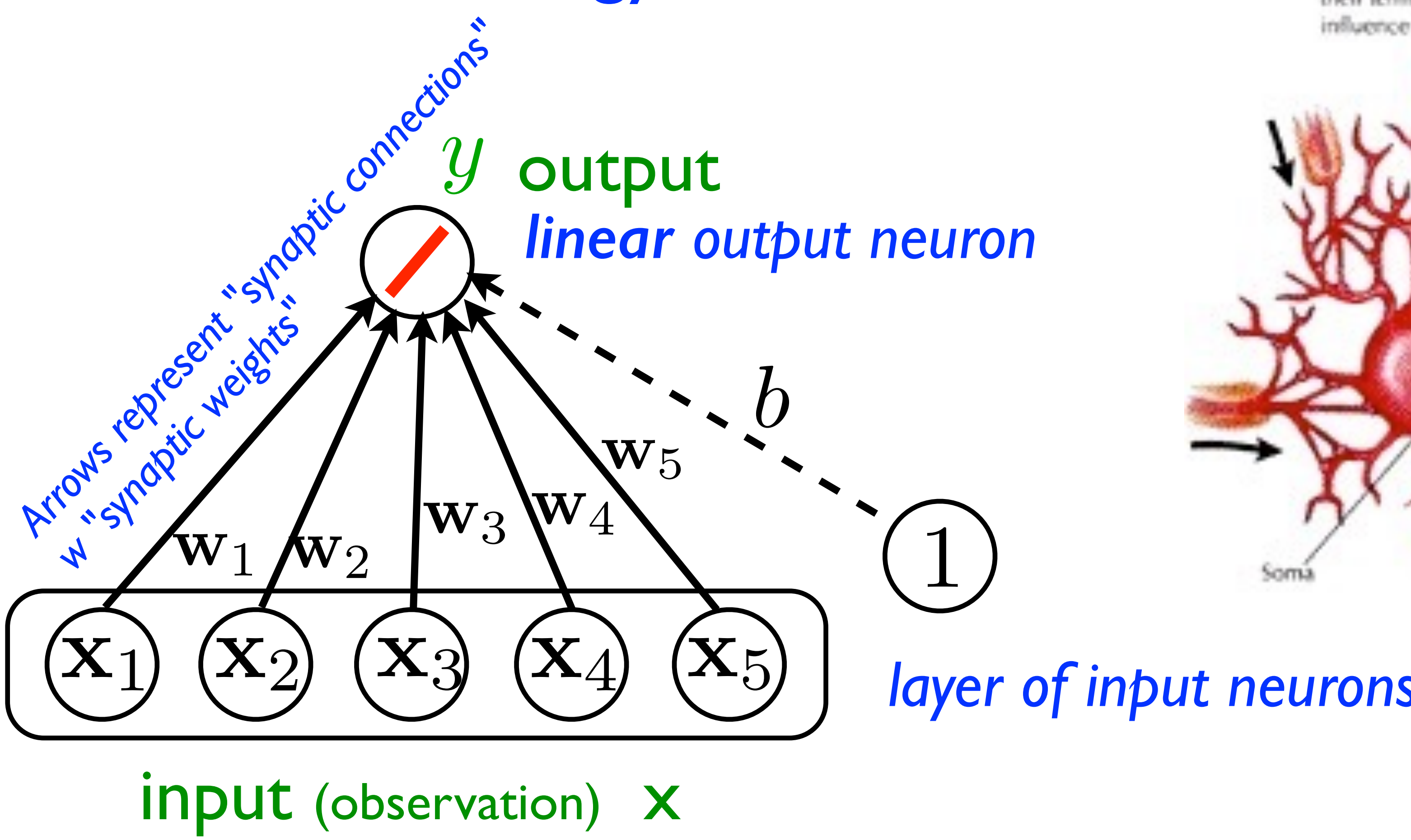
Neural inspiration

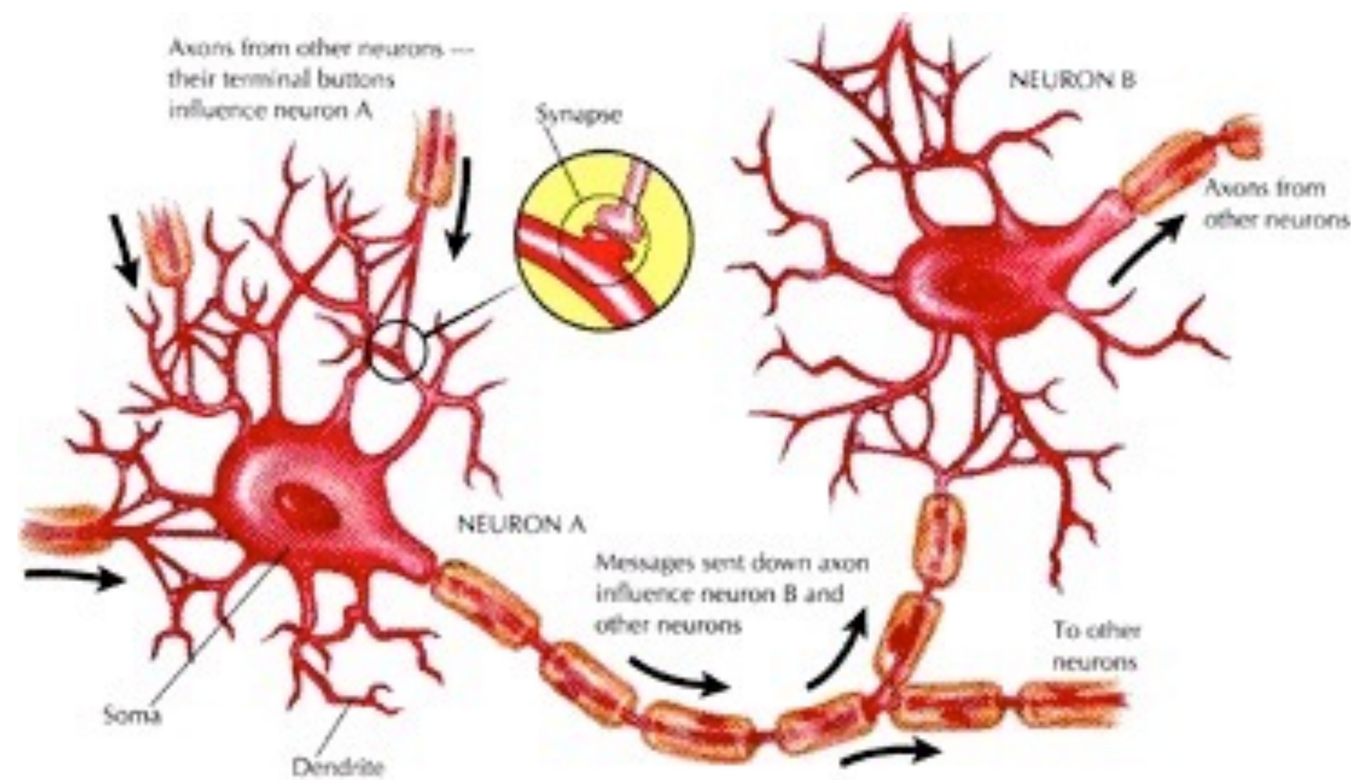
Intuitive understanding of the scalar product

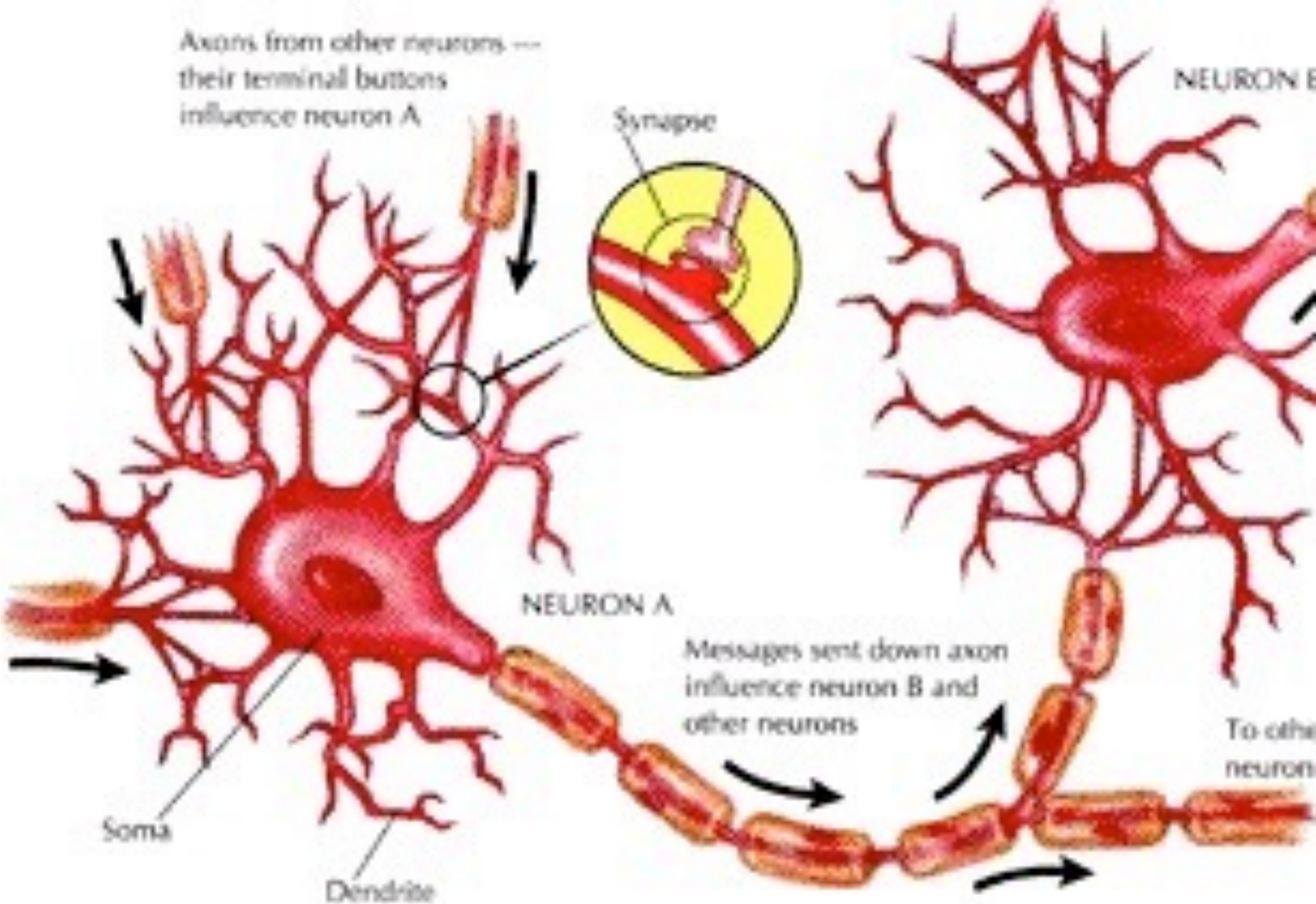
each component of x has a weighted influence on the output y

$$y = f_\theta(\mathbf{x}) = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \ldots + \mathbf{w}_d\mathbf{x}_d + b$$

Neural terminology






Axons from other neurons —
their terminal buttons
influence neuron A
Synapse
NEURON A
Messages sent down axon
influence neuron B and
other neurons
Soma
Dendrite

Regularized empirical risk

It is often necessary to induce a "preference" for some parameter values rather than others to avoid overfitting

We define regularized empirical risk as follows:

$$\hat{R}_\lambda(f_\theta, D_n) \quad = \quad \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})\right)}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\substack{\text{regularization term}\\ \text{(penalty)}}}$$

Ω penalizes more or less the different parameter values.
$\lambda \geq 0$ the importance of this regularization term
(in relation to the empirical risk)

Eg: Ridge Regression

= linear regression + quadratic (L2) regularization

We penalize the large weights

$$\Omega(\theta) = \Omega(\mathbf{w}, b) = \|\mathbf{w}\|^2 = \sum_{j=1}^{d} \mathbf{w}_j^2$$

Neural terminology:
"weight decay" penalty

$$\hat{R}_\lambda(f_\theta, D_n) \quad = \quad \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) \right)}_{\text{empirical risk}} + \underbrace{\lambda \Omega(\theta)}_{\text{regularization term (penalty)}}$$

Eg: Ridge regression

= linear regression + quadratic (L2) regularization

Regularized empirical risk

$$\hat{R}_\lambda(f_\theta, D_n) \quad = \quad \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})\right)}_{\text{Empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\substack{\text{regularization term} \\ \text{(penalty)}}}$$

We are looking for the parameter values that minimize this objective

$$\{\mathbf{w}^*, b^*\} = \theta^* = \underset{\theta}{\operatorname{argmin}} \hat{R}_\lambda(f_\theta, D_n)$$

Eg: Ridge Regression

= linear regression + quadratic (L2) regularization

- For linear regression or ridge regression a little linear algebra gives us an **analytical solution**

we solve for $\theta = \{b, \mathbf{w}\}$: $$\frac{\partial \hat{R}_\lambda(f_\theta, D_n)}{\partial \theta} = 0$$

we obtain: $$\boxed{\begin{pmatrix} b^* \\ \mathbf{w}^* \end{pmatrix} = (\check{X}^T \check{X} + \lambda \check{I})^{-1} \check{X}^T \mathbf{t}}$$

$$\text{où } \check{X} = \begin{pmatrix} 1 & \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{x}_1^{(n)} & \dots & \mathbf{x}_d^{(n)} \end{pmatrix}, \mathbf{t} = \begin{pmatrix} t^{(1)} \\ \vdots \\ t^{(n)} \end{pmatrix} \qquad \check{I} = \begin{pmatrix} 0 & 0 & & & \\ 0 & 1 & 0 & & \\ & 0 & 1 & 0 & \\ & & 0 & \ddots & 0 \\ & & & 0 & 1 \end{pmatrix}$$

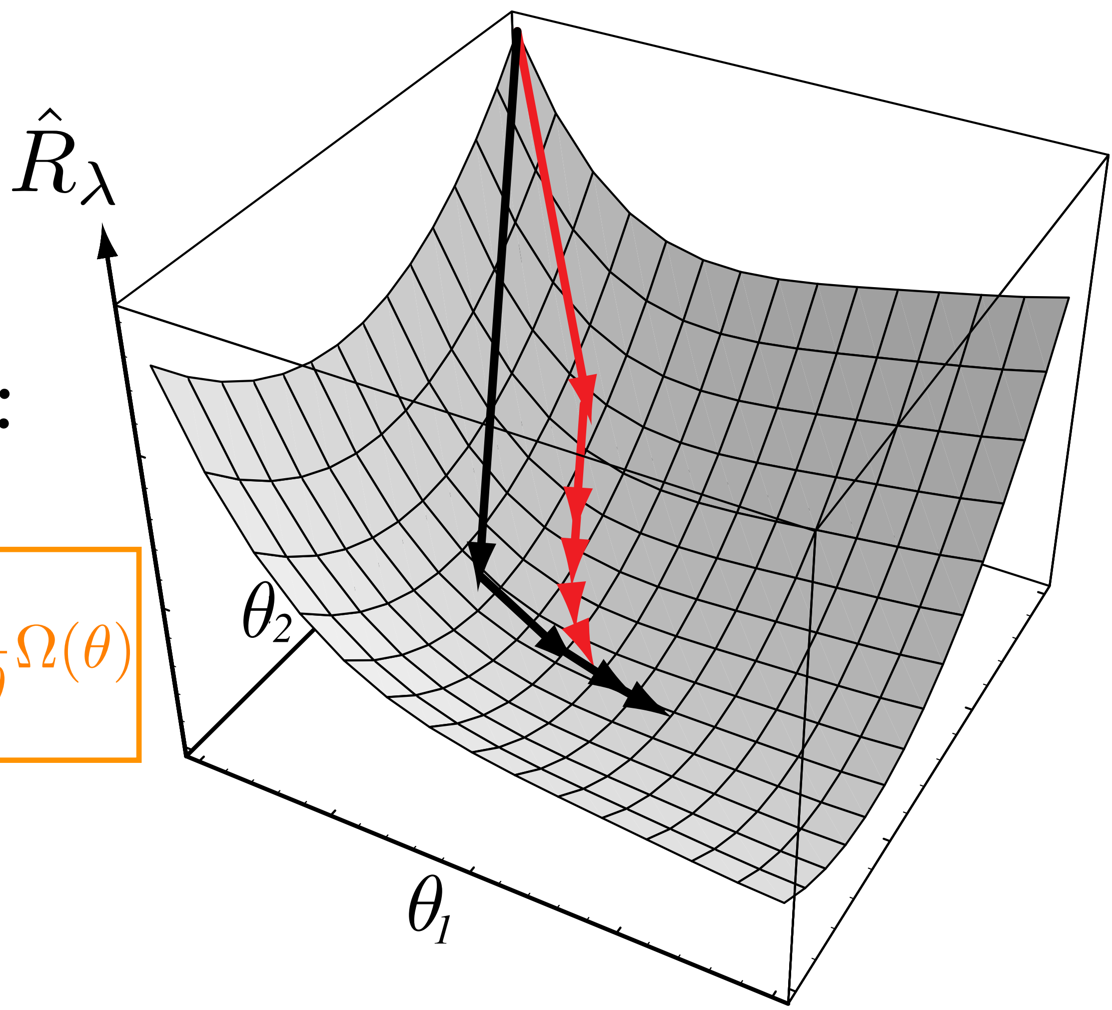
- Most of the time (other choices of f and L) we do not have an analytical solution.
- More generally, we can use a gradient descent method.

Other possibility:

optimization by gradient descent

$$\hat{R}_\lambda(f_\theta, D_n) = \underbrace{\left(\sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})\right)}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\text{regularization term}}$$

- we initialize the parameters randomly
- we update them iteratively following the gradient



Either batch gradient descent (whole dataset):

Loop: $\theta \leftarrow \theta - \eta \frac{\partial \hat{R}_\lambda}{\partial \theta}$

$$= \boxed{\left(\sum_{i=1}^{n} \frac{\partial}{\partial \theta} L(f_\theta(\mathbf{x}^{(i)}), t^{(i)})\right) + \lambda \frac{\partial}{\partial \theta}\Omega(\theta)}$$

Or stochastic gradient descent:

Loop:

For i in 1...n

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta}\left(L(f_\theta(\mathbf{x}^{(i)}), t^{(i)}) + \frac{\lambda}{n}\Omega(\theta)\right)$$

Or other variants of the gradient descent idea

(conjugate gradient, Newton's method, natural gradient, ...)

Various regularizers

«Ridge»: regularization, L_2

In Bayesian terms: corresponds to a Gaussian prior on the weights

$$\Omega(\theta) = \Omega(\mathbf{w}, b) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} \mathbf{w}_j^2$$

«Lasso»: regularization, L_1

In Bayesian terms: corresponds to a Laplacian prior on the weights

$$\Omega(\theta) = \Omega(\mathbf{w}, b) = \|\mathbf{w}\|_1 = \sum_{j=1}^{d} |\mathbf{w}_j|$$

=> automatic selection of components (a number of weights will be zero)

«Elastic net»: combination of the two

$$\Omega(\theta) = \Omega(\mathbf{w}, b) = \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2$$

Etc...