A. Giovanidis 2022

12. Neural Networks

Network Data Analysis – NDA 2022–2023

Anastasios Giovanidis

Sorbonne-LIP6

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

January 2023

Bibliography

B.1 Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, "Dive into Deep Learning", 2021, online: https://d2l.ai/

B.2 https://dataflowr.github.io/website/

B.3 I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning", MIT Press, 2017.

B.4 O. Simeone, "Machine Learning for Engineers", Cambridge University Press, to appear 2023

B.5 Formation Fidle – CNRS: https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/wikis/home

A. Giovanidis 2022

A. Giovanidis 2022

Linear regression

Remember the problem of Linear Regression. Training data available $D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, with index $i = 1, \ldots, n$

- Input data has d dimensions (features) $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d})$
- output data has 1 dimension y_i

Linear regression assumes that the output is an affine function of the input. Find (Θ, b) so that $\hat{y}$ can approximate well y_i, $i = 1, \ldots, n$ and can generalise to unknown inputs

$$\begin{aligned} \hat{y} &= w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b \\ &= \Theta^T \mathbf{x} + b \end{aligned}$$

A. Giovanidis 2022

Least Squares

The loss function is the *Residual Sum of Squares*

$$J_{RSS}(\Theta, b) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \Theta^T \mathbf{x}_i - b)^2$$

which we want to minimise w.r.t. the unknown model parameters

$$\min_{\Theta, b} J_{RSS}(\Theta, b)$$

Using the partial derivatives we get the gradient for each loss item i

$$\frac{\partial J_{RSS,i}}{\partial w_d} = -(y_i - \Theta^T \mathbf{x}_i - b)x_{i,d}$$
$$\frac{\partial J_{RSS,i}}{\partial b} = -(y_i - \Theta^T \mathbf{x}_i - b)$$

☞ In this linear model the solution has a closed form.
But can we solve for other more general models?

A. Giovanidis 2022

Gradient descent

In general we have the finite sum problem

$$\min_{\Theta} J(\Theta) \quad = \quad \min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} J_i(\Theta)$$

Basic gradient descent

$$\begin{aligned} \Theta_{t+1} &= \Theta_t - \eta_t \nabla J(\Theta_t) \\ &= \Theta_t - \eta_t \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(\Theta_t) \end{aligned}$$

At every step t it moves monotonically towards the optimum.

In practice η_t is chosen either as constant, $\eta_t = \eta$, or as a square summable, but not summable function: $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ and $\sum_{t=1}^{\infty} \eta_t = \infty$.

Question:

☞ What is the drawback of this iteration?

A. Giovanidis 2022

Incremental gradient methods

A. Giovanidis 2022

Stochastic Gradient Descent (SGD) At each iteration pick a random data-point $i(t) \in \{1, \ldots, n\}$ and update

$$\Theta_{t+1} \quad = \quad \Theta_t - \eta_t \nabla J_{i(t)}(\Theta_t)$$

Sampling methods

- Random Uniform: Randomly pick an index i with replacement.
- Permutation Based: Pick index i in given order without replacement (`pre-shuffle dataset`)

SGD characteristics

A. Giovanidis 2022

- n-times faster to compute than the full $\nabla J(\Theta_t)$.
- But! it does not decrease monotonically.
- Very sensitive to the step size η_t.
- Converges fast to a region of confusion close to the optimum.
- Fluctuates around the optimum inside the region of confusion.
- The stochastic gradients are unbiased estimates of the true gradient!

A. Giovanidis 2022

Mini-batching

Speed of convergence depends on how noisy the stochastic gradients are:

- The variance controls the convergence rate of the SGD.
- Smaller variance $\rightarrow$ faster convergence to the true gradient.

$$\Theta_{t+1} = \Theta_t - \eta_t \frac{1}{M} \sum_{i=1}^{M} \nabla J_i(\Theta_t)$$

A. Giovanidis 2022

Mini-batching

Speed of convergence depends on how noisy the stochastic gradients are:

- The variance controls the convergence rate of the SGD.
- Smaller variance $\rightarrow$ faster convergence to the true gradient.

$$\Theta_{t+1} = \Theta_t - \eta_t \frac{1}{M} \sum_{i=1}^{M} \nabla J_i(\Theta_t)$$

- $M = 1$: vanilla SGD (`cheap / high variance`)
- $M = n$: full gradient descent (`expensive / low variance`)
- $1 < M < n$: in-between trade-off. Allows multi-core parallelism.

But if the batch is too large, we end up over-fitting the network.

A. Giovanidis 2022

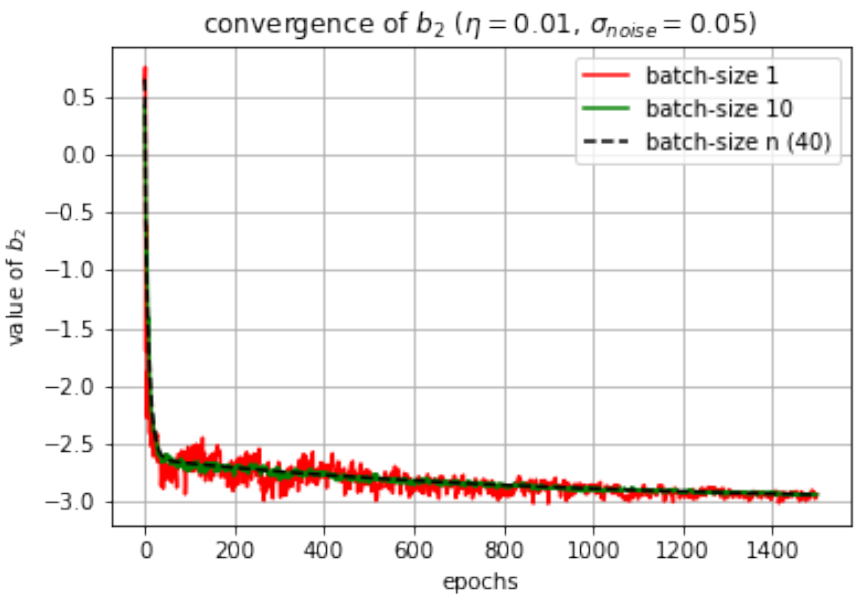
Mini-batch convergence example

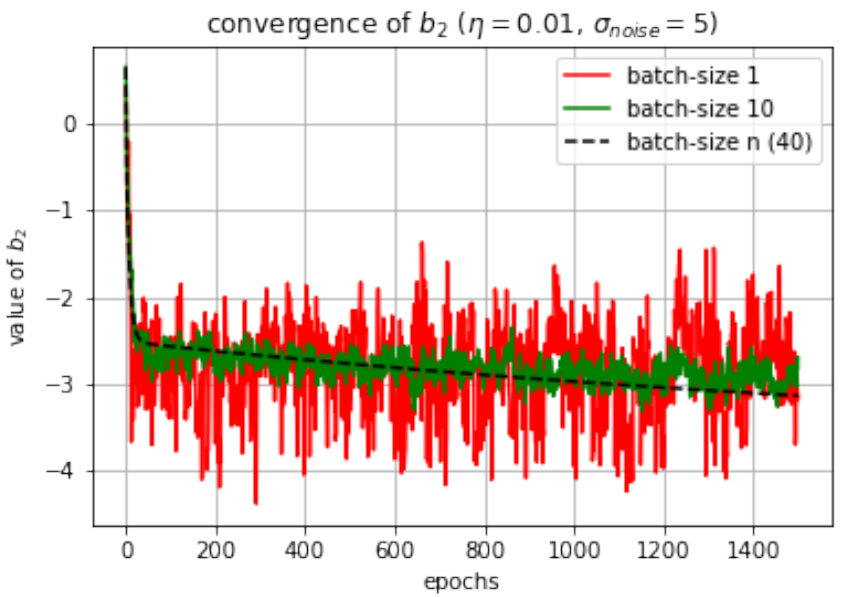
Consider the data comes from the linear model

$$y = 5 + 2x_1 - 3x_2 + \epsilon$$

and we try to estimate with

$$\hat{y} = \hat{b}_0 + \hat{b}_1 x_1 + \hat{b}_2 x_2$$





Hyperparameters choice

A. Giovanidis 2022

Hyperparameters:

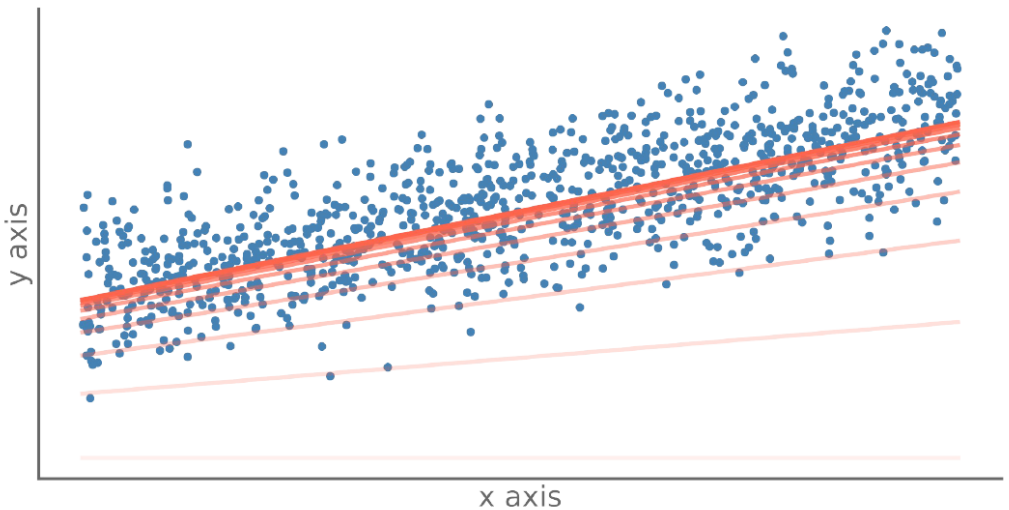
- Learning rate η_t
- Mini-batch size M

In general, the goal is not just to minimise the training loss.
☞ narrow minima of the training loss do not generalise well; wide minima are preferable.

☞ Run SGD with different hyperparameters and choose the pair that minimizes validation loss.

Regression Example

A. Giovanidis 2022



#i	Loss	Gradient		Theta	
0	+12.481	-6.777	-1.732	-3.388	+0.000
20	+4.653	-4.066	-1.039	-2.033	+0.346
40	+1.835	-2.440	-0.624	-1.220	+0.554
60	+0.821	-1.464	-0.374	-0.732	+0.679
80	+0.455	-0.878	-0.224	-0.439	+0.754
100	+0.324	-0.527	-0.135	-0.263	+0.799
120	+0.277	-0.316	-0.081	-0.158	+0.826
140	+0.260	-0.190	-0.048	-0.095	+0.842
160	+0.253	-0.114	-0.029	-0.057	+0.851
180	+0.251	-0.068	-0.017	-0.034	+0.857
200	+0.250	-0.041	-0.010	-0.020	+0.861

From [B.5] 01-history-and-basic-concepts

A. Giovanidis 2022

From Linear Regression to Neural Networks

A. Giovanidis 2022

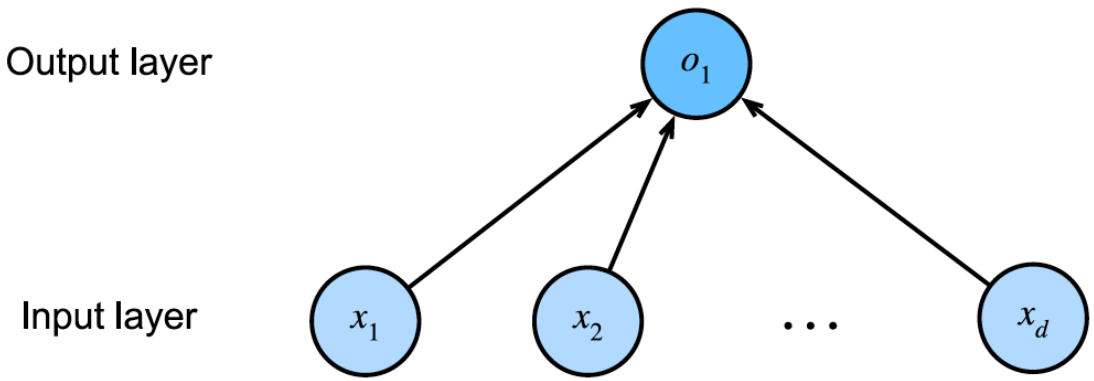
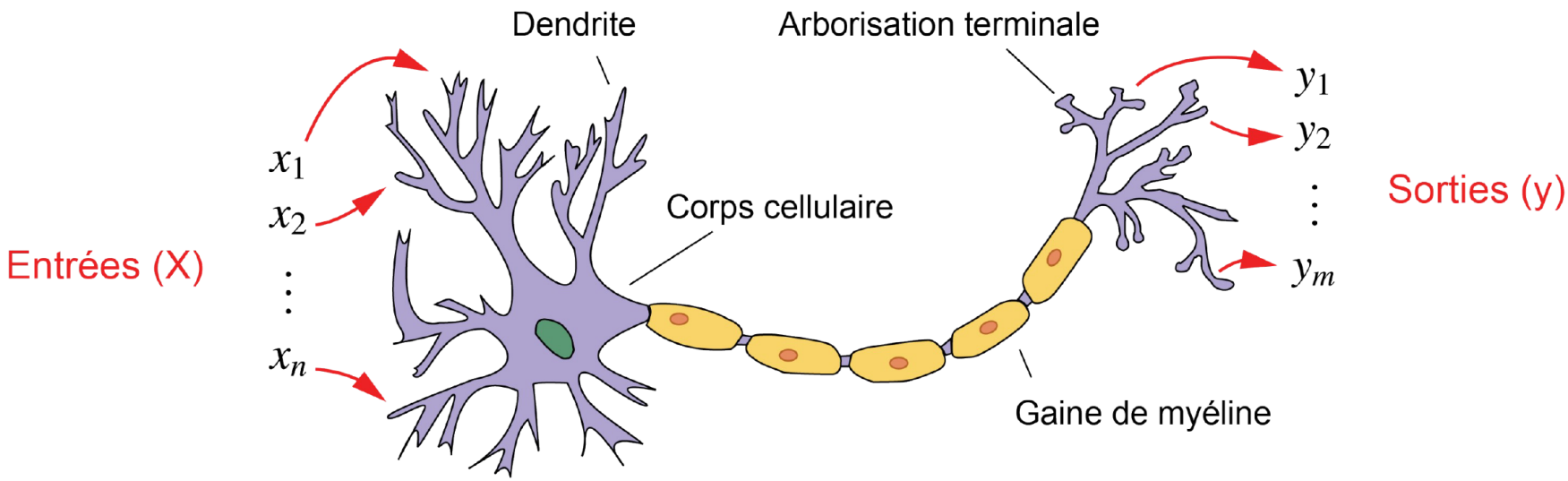


Fig. 3.1.2: Linear regression is a single-layer neural network.

Fully connected single layer network, from [B.1]

Biology inspired neuron

A. Giovanidis 2022

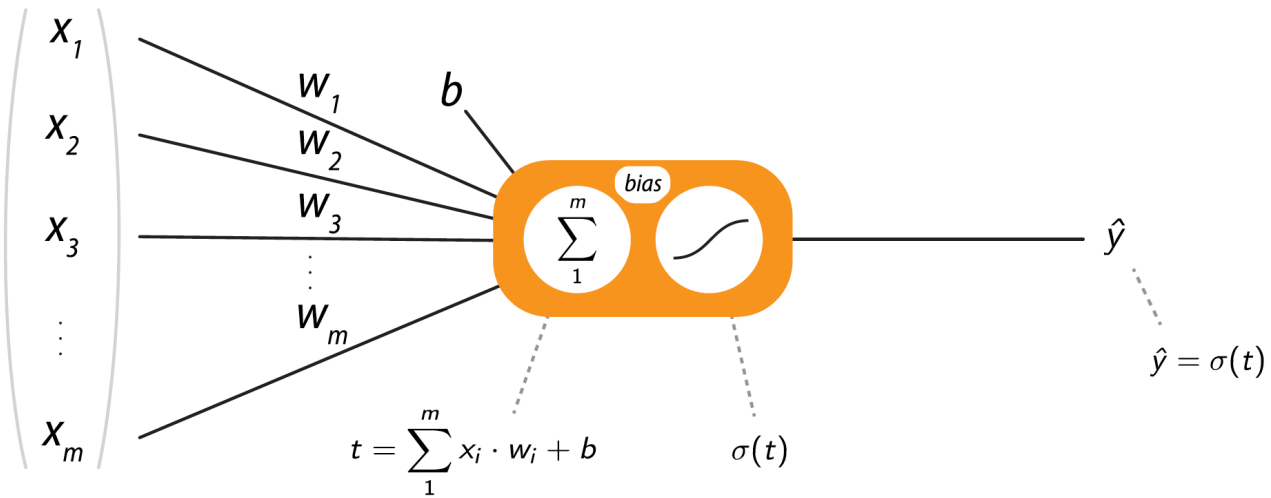


From [B.5] 01-history-and-basic-concepts

Simple artificial neuron

A. Giovanidis 2022

$$\hat{y} = \sigma(\Theta^T \cdot X + b)$$



Input	Bias / Weight	Activation function	Output
X	Θ, b	$\sigma(t)$	$\hat{y}$

From [B.5] 01-history-and-basic-concepts

A. Giovanidis 2022

Reminder: Logistic Regression

Find the probability for y to be in class 1 based on the features $\mathbf{x}$

$$p(\mathbf{x}) = \mathbf{P}\left[y = 1 \mid \mathbf{x}\right] = \frac{e^{\Theta^T \cdot \mathbf{x} + b}}{1 + e^{\Theta^T \cdot \mathbf{x} + b}} = \frac{1}{e^{-\Theta^T \cdot \mathbf{x} - b} + 1}$$

The weights (Θ, b) should minimize the negative log-likelihood cost

$$J(\Theta, b) = -\sum_{i=1}^{n} \left\{ y_i \log p(\mathbf{x}_i) + (1 - y_i) \log(1 - p(\mathbf{x}_i)) \right\}$$

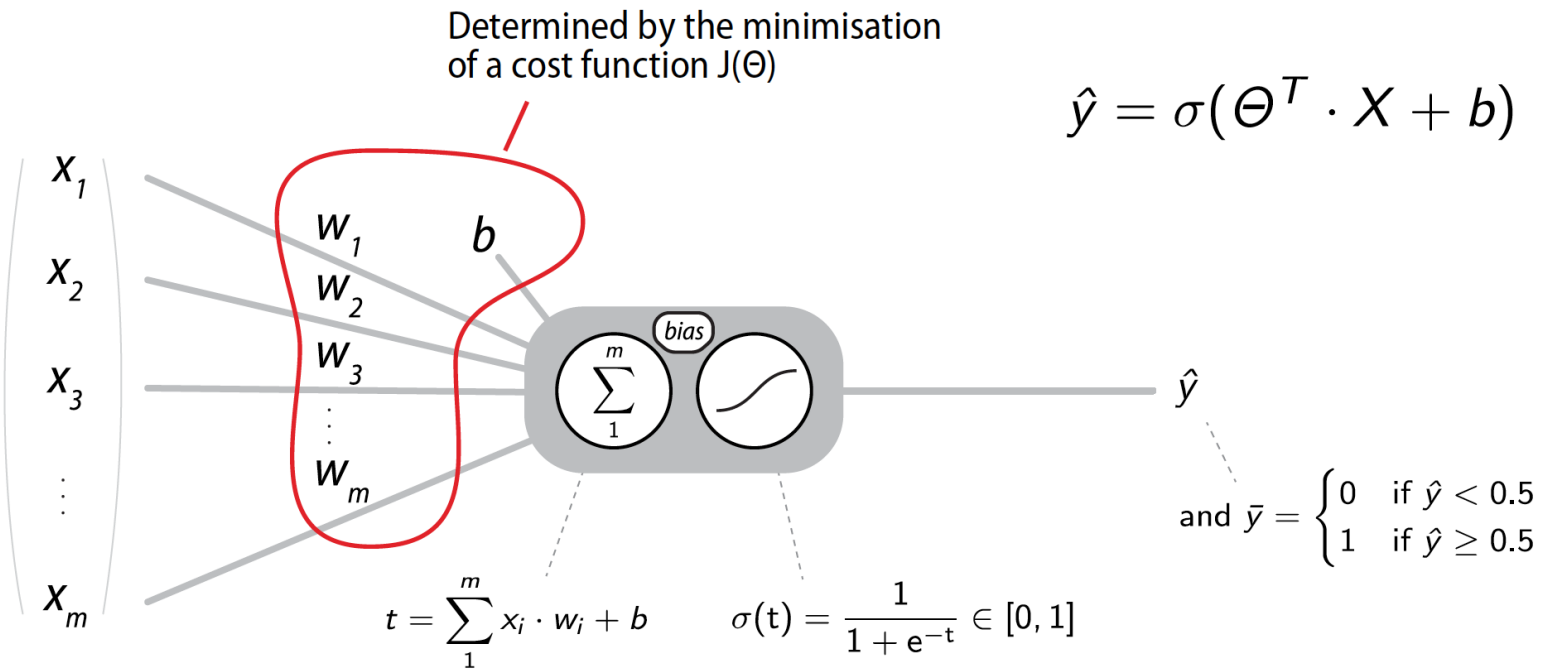
The minimization is achieved by (Stochastic) Gradient Descent

$$\Theta_{t+1} = \Theta_t - \eta_t \frac{1}{M} \sum_{\mathtt{i} \in \mathtt{minibatch}} \nabla_\Theta J(\Theta, b)$$

$$b_{t+1} = b_t - \eta_t \frac{1}{M} \sum_{\mathtt{i} \in \mathtt{minibatch}} \nabla_b J(\Theta, b)$$

Logistic Regression

A. Giovanidis 2022



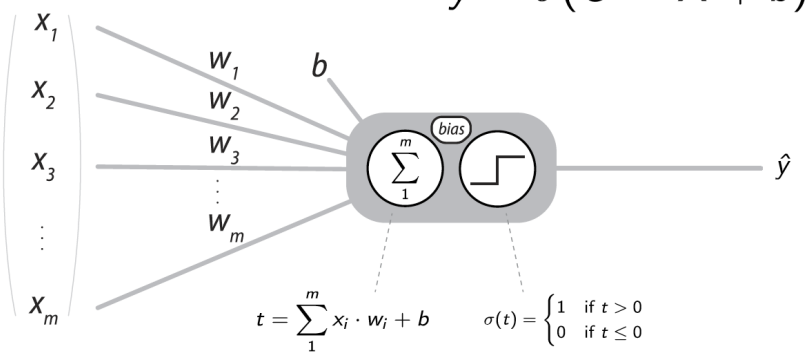
Input	Bias / Weight	Activation function	Output
X	Θ	$\sigma(t)$	$\hat{y}$

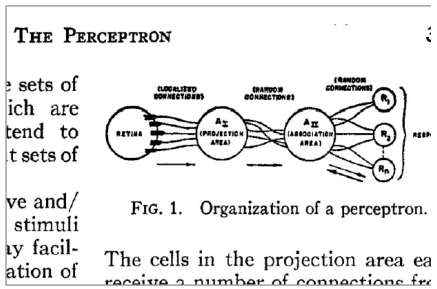
From [B.5] 01-history-and-basic-concepts

The Perceptron

A. Giovanidis 2022

$$\hat{y} = \sigma(\Theta^T \cdot X + b)$$




THE PERCEPTRON

sets of
ich are
tend to
t sets of
ve and/
stimuli
y facil-
ation of

FIG. 1. Organization of a perceptron.

The cells in the projection area ea

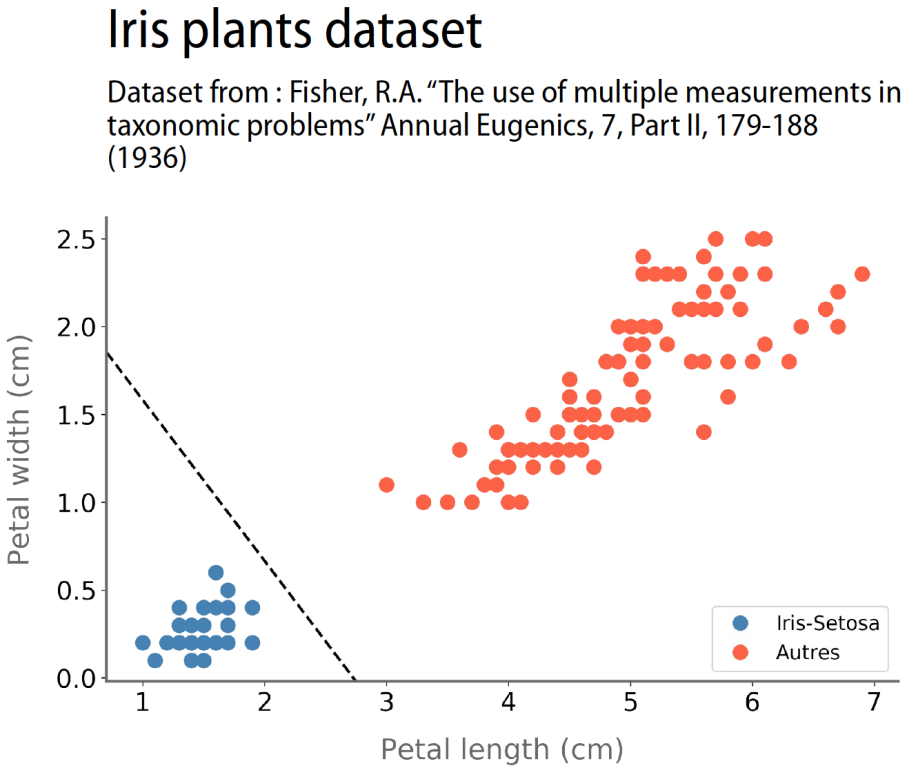
Perceptron
Frank Rosenblatt
1958

Linear and binary classifier

From [B.5] 01-history-and-basic-concepts

Perceptron Example

A. Giovanidis 2022

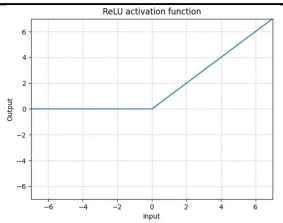
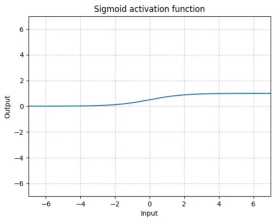
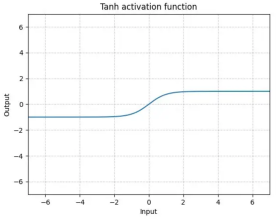


Length x_1	Width x_2	Iris Setosa (0/1) y
1.4	1.4	1
1.6	1.6	1
1.4	1.4	1
1.5	1.5	1
1.4	1.4	1
4.7	4.7	0
4.5	4.5	0
4.9	4.9	0
4.0	4.0	0
4.6	4.6	0
(...)		

From [B.5] 01-history-and-basic-concepts

A. Giovanidis 2022

Activation functions

Function	Description	Plot	Characteristics
`ReLU`	$\max(0, z)$		positive values only
`Sigmoid`	$\frac{1}{1+e^{-z}}$		$[0, +1]$ binary class
`Softmax`	$\frac{e^{z_i}}{\sum_j e^{z_j}}$		$[0, +1]$ multi-class
`Tanh`	$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$		$[-1, +1]$ values

A. Giovanidis 2022

Loss functions

- Regression:
  - Mean Squared Error Loss

$$\texttt{MSELoss}(i) \quad = \quad (y_{i,true} - y_{i,pred})^2$$

- Classification
  - (2 classes): Binary Cross Entropy Loss

$$\texttt{BCELoss}(i) \quad = \quad -[y_i \log(p(\mathbf{x}_i)) + (1 - y_i) \log(1 - p(\mathbf{x}_i))]$$

  - (c classes): Negative Log-Likelihood Loss

$$\texttt{NLLLoss}(i) \quad = \quad \log softmax_{(y_i)} \left(\Theta_1^T \mathbf{x}_i, \ldots, \Theta_c^T \mathbf{x}_i\right)$$

where,

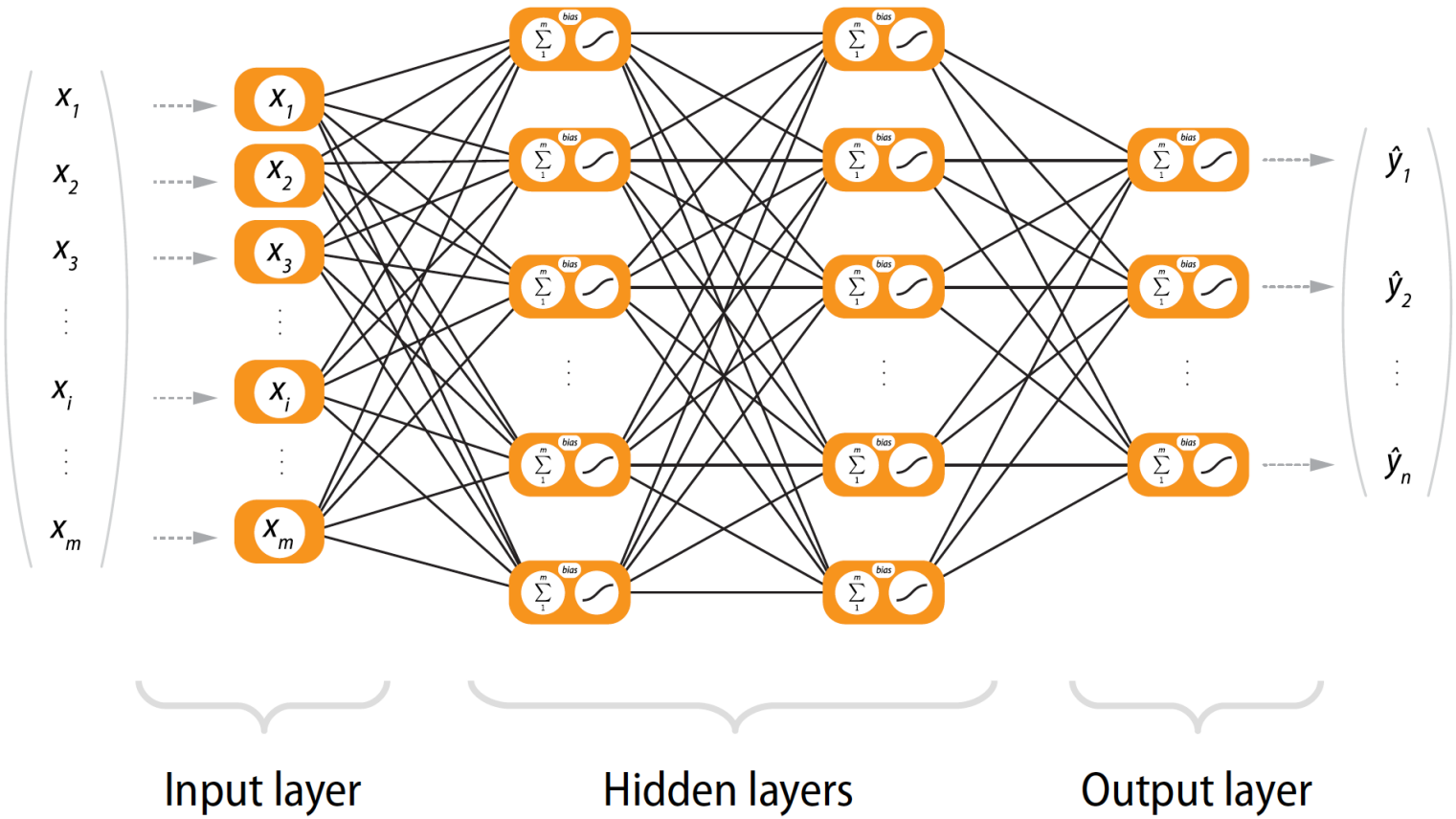
$$softmax_{(y_i=k)} \left(\Theta_1^T \mathbf{x}_i, \ldots, \Theta_c^T \mathbf{x}_i\right) \quad = \quad \frac{e^{\Theta_k^T \mathbf{x}_i}}{\sum_{\ell=1}^{c} e^{\Theta_\ell^T \mathbf{x}_i}}$$

☞ The last layer of your network should be LogSoftmax().

A. Giovanidis 2022

Multi-Layer Perceptron

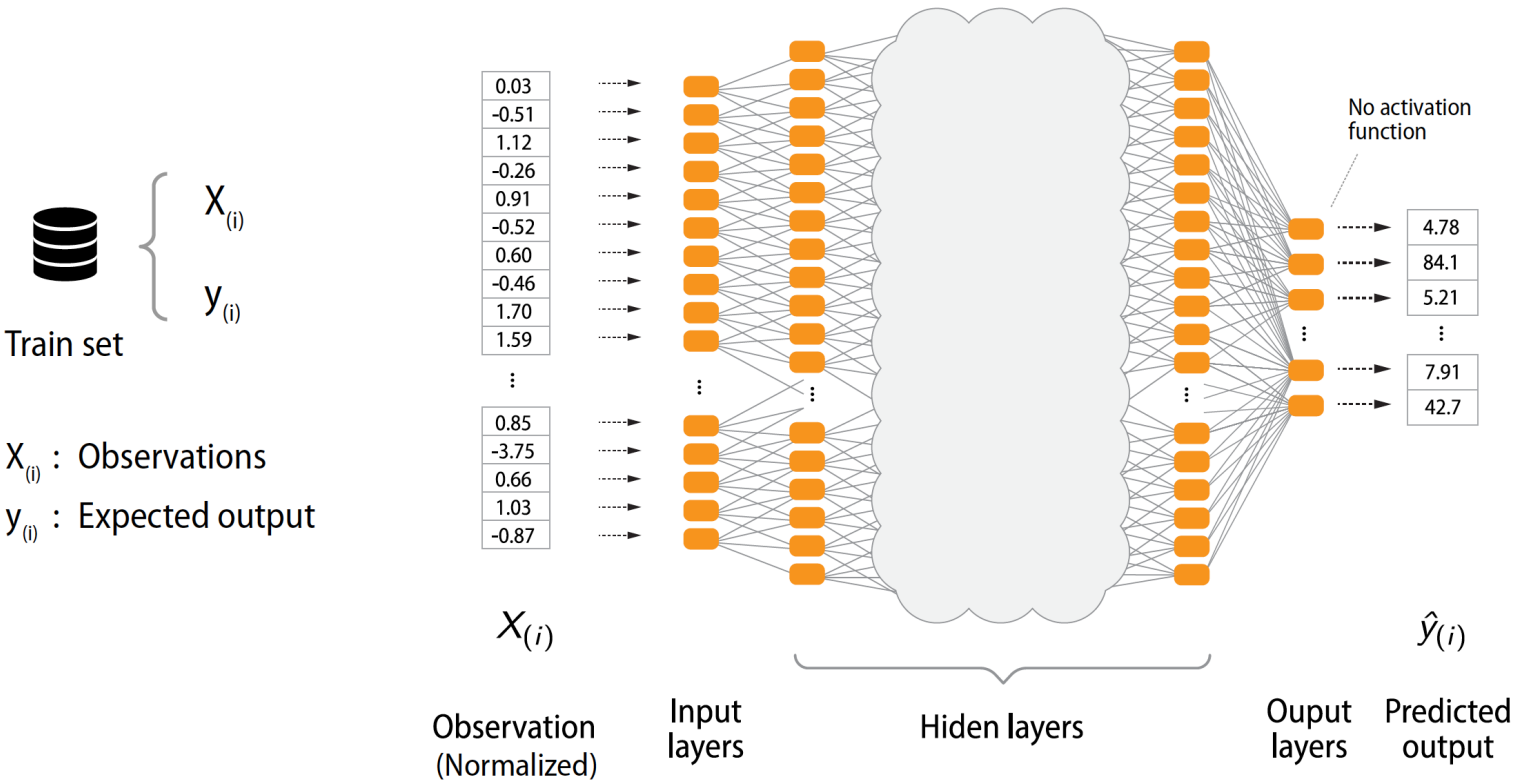
A. Giovanidis 2022



From [B.5] 01-history-and-basic-concepts

Hidden Layers (Regression)

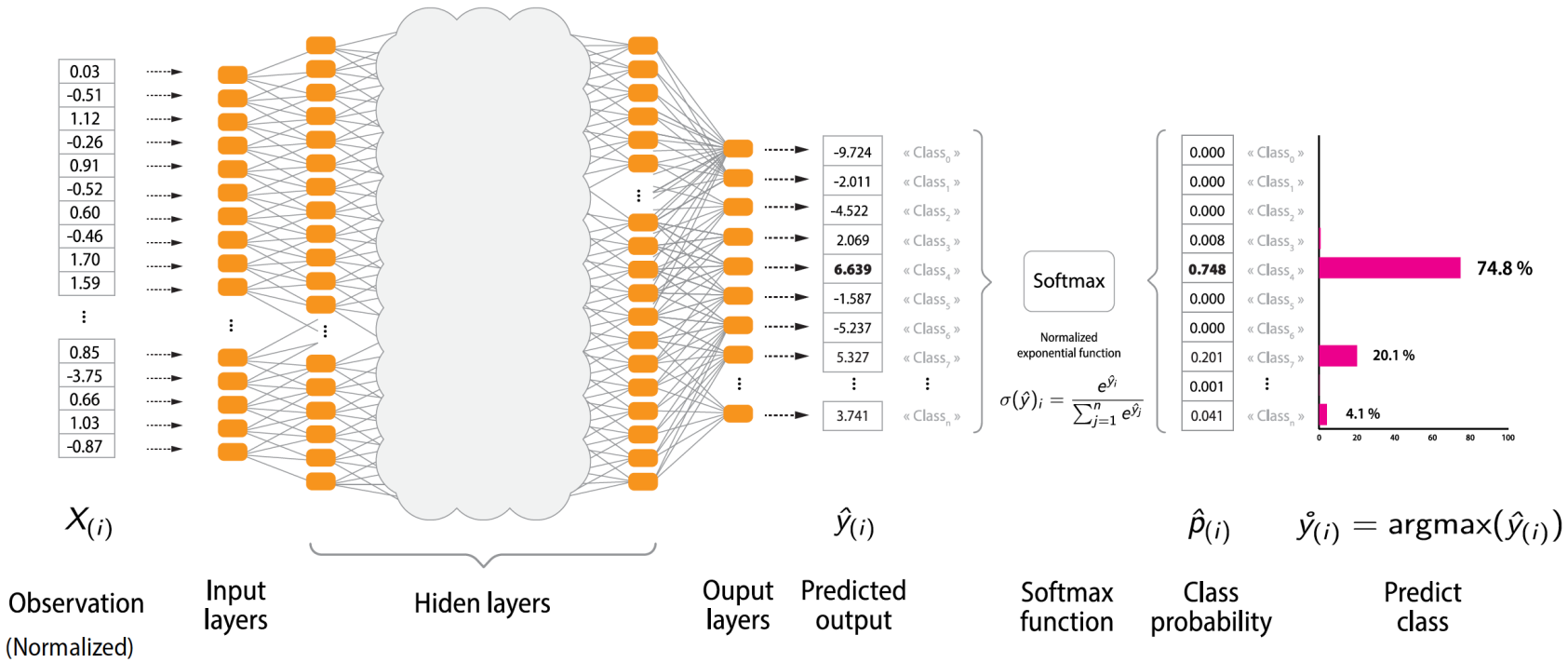
A. Giovanidis 2022



From [B.5] 01-history-and-basic-concepts

Hidden Layers (Classification)

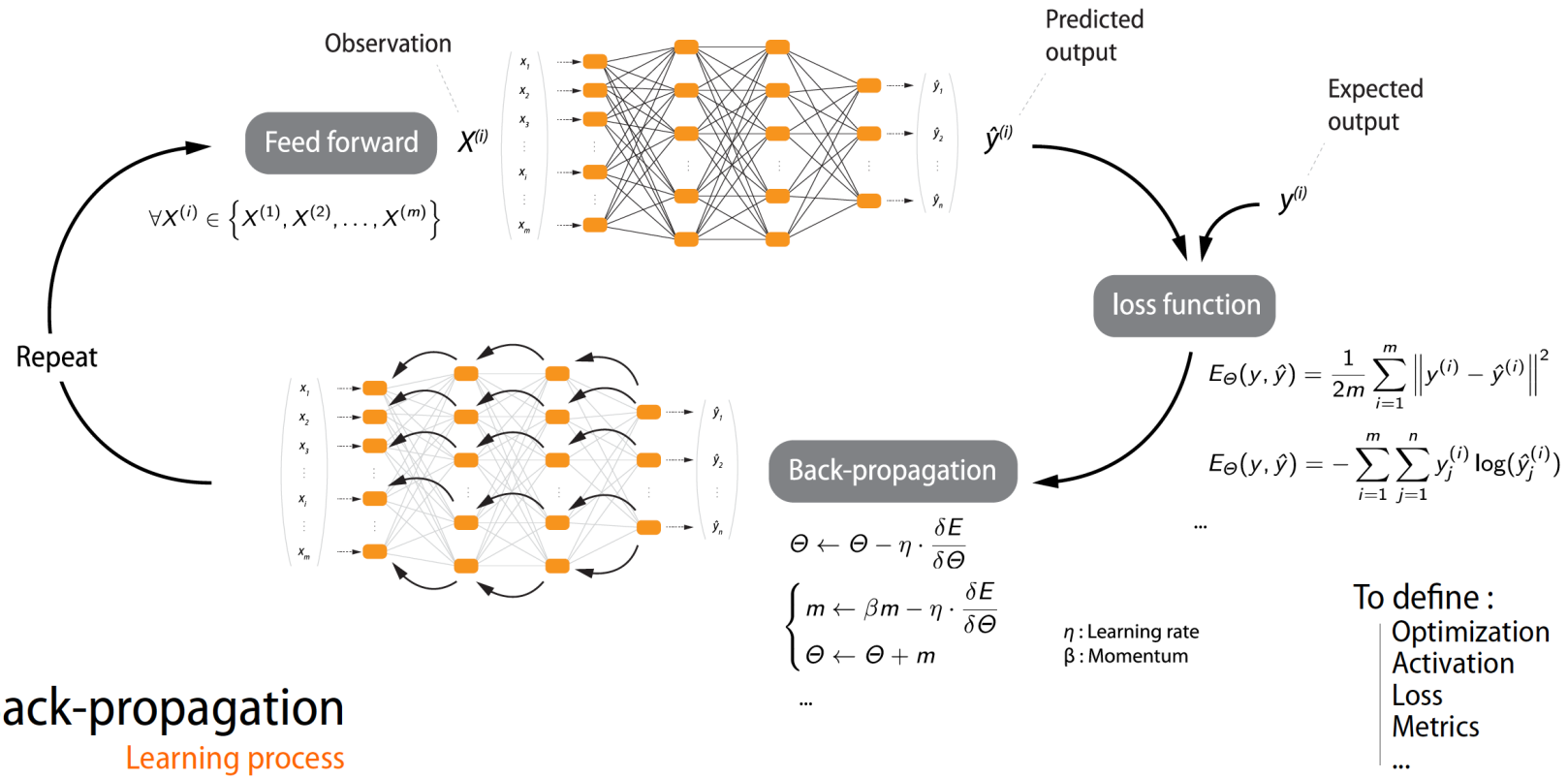
A. Giovanidis 2022



From [B.5] 01-history-and-basic-concepts

Deep Learning Iteration Cycle

A. Giovanidis 2022



Back-propagation

Learning process

From [B.5] 01-history-and-basic-concepts

A. Giovanidis 2022

Automatic Differentiation

A. Giovanidis 2022

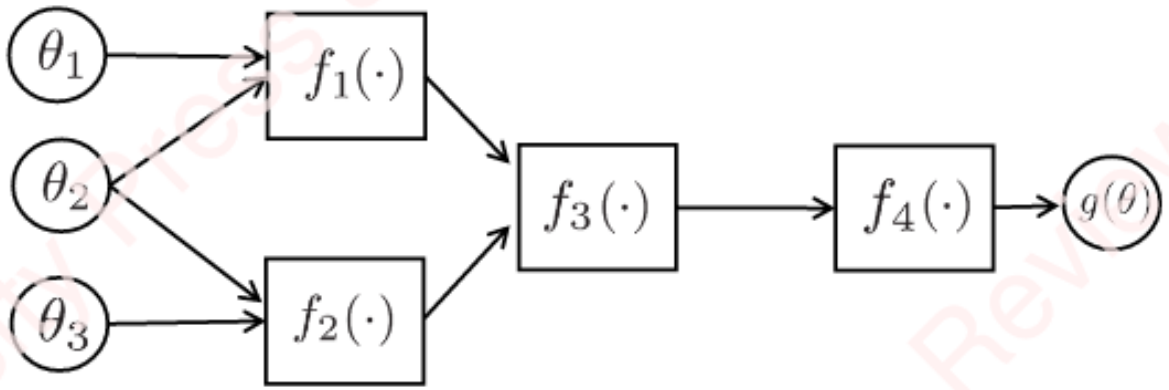
Automatic differentiation is based on the differentiation chain rule.

Backpropagation (or backprop) is key for training neural networks.

- For a given vector θ it computes the numerical value $\nabla g(\theta)$.
- The function $g(\theta)$ is seen as a composition of elementary functions $\{f_i\}$, whose local derivatives are easy to compute.
- All elementary functions output a single scalar.

A. Giovanidis 2022

Computational graph



$$g(\theta) \quad = \quad f_4(f_3(f_1(\theta_1, \theta_2), f_2(\theta_2, \theta_3)))$$

☞ Directed acyclic graph: input nodes $\{\theta_i\}_{i=1}^{d}$, computational blocks $f_i(.)$, and output node $g(\theta)$.

Source [B.4], Chapter 5

Chain rule of differentiation

A. Giovanidis 2022

Consider the function for $\theta \in \mathbb{R}^d$

$$g(\theta) \quad = \quad f_{K+1}(f_1(\theta), f_2(\theta), \ldots, f_K(\theta))$$

Chain Rule

$$\frac{\partial g(\theta)}{\partial \theta_i} \quad = \quad \sum_{k=1}^{K} \frac{\partial f_{K+1}(f_1, \ldots, f_K)}{\partial f_k} \cdot \frac{\partial f_k(\theta)}{\partial \theta_i}$$

☞ The derivatives multiply along each influence path and the contributions of all paths sum.

Forward mode of automatic differentiation

A. Giovanidis 2022

1. Each block $f_k(.)$ computes the local partial derivative $\frac{\partial f_k(\theta)}{\partial \theta_i}$.
2. The message is passed to computational block $f_{K+1}(.)$, which multiplies it by the local partial derivative $\partial f_{K+1}(f_1, \ldots, f_K)/\partial f_k$.
3. The block $f_{K+1}(.)$ sums up all product terms $\partial f_{K+1}/\partial f_k \cdot \partial f_k(\theta)/\partial \theta_i$ to obtain the final result $\partial g(\theta)/\partial \theta_i$.

☞ To compute the full gradient $\nabla g(\theta)$ this procedure should be repeated for every entry $\theta_i \rightarrow$ inefficient for large dimensions d.

A. Giovanidis 2022

Backpropagation

☞ The forward pass is for the calculation of $g(\theta)$ through the computational graph, and the backward pass to compute all partial derivatives $\partial g(\theta)/\partial\theta_i$, simultaneously.

Algorithm 5.3: Backprop to compute all partial derivatives (5.82) for the computational graph in Fig. 5.16

Forward pass:

1. Given an input vector θ, by following the direction of the edges in the computational graph, each kth computational block $f_k(\cdot)$ evaluates its output f_k for $k = 1, \ldots, K+1$.
2. Each computational block $f_k(\cdot)$ also evaluates the local derivatives $\partial f_k(\theta)/\partial\theta_i$ with respect to its inputs $\theta_1, \ldots, \theta_D$ with $k = 1, \ldots, K$.
3. Computational block $f_{K+1}(\cdot)$ evaluates all the local partial derivatives $\partial f_{K+1}(f_1, \ldots, f_K)/\partial f_k$ with respect to all its inputs f_k, with $k = 1, 2, \ldots, K$.

Backward pass:

1. Block $f_{K+1}(\cdot)$ passes each message $\partial f_{K+1}(f_1, \ldots, f_K)/\partial f_k$ back to the respective block $f_k(\cdot)$.
2. Each block $f_k(\cdot)$ multiplies the message $\partial f_{K+1}(f_1, \ldots, f_K)/\partial f_k$ by the corresponding local derivative $\partial f_k(\theta)/\partial\theta_i$.
3. Each block $f_k(\cdot)$ sends the computed message $\partial f_{K+1}(f_1, \ldots, f_K)/\partial f_k \cdot \partial f_k(\theta)/\partial\theta_i$ for $i = 1, \ldots, D$ back to variable node θ_i.
4. Finally, each variable node θ_i sums up all its received upstream messages to obtain (5.82).

Source [B.4], Chapter 5

A. Giovanidis 2022

Example 1

Figure 5.19 Forward pass to compute the gradient $\nabla g(\theta)$ at $\theta = [1, -1, 1]^T$ for function $g(\theta) = \theta_1^2 + 2\theta_2^2 - \theta_3^2$.

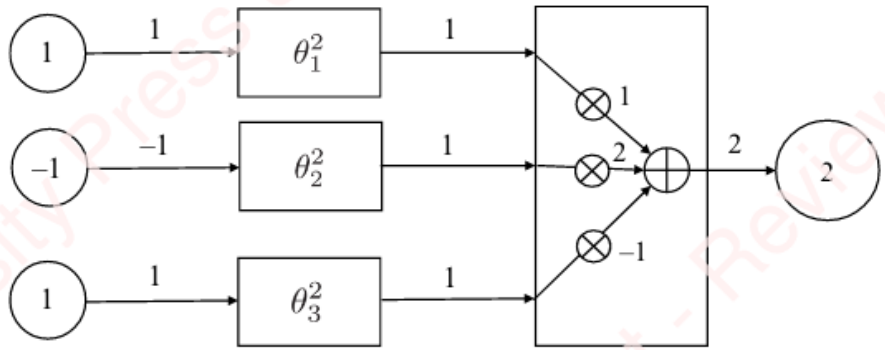
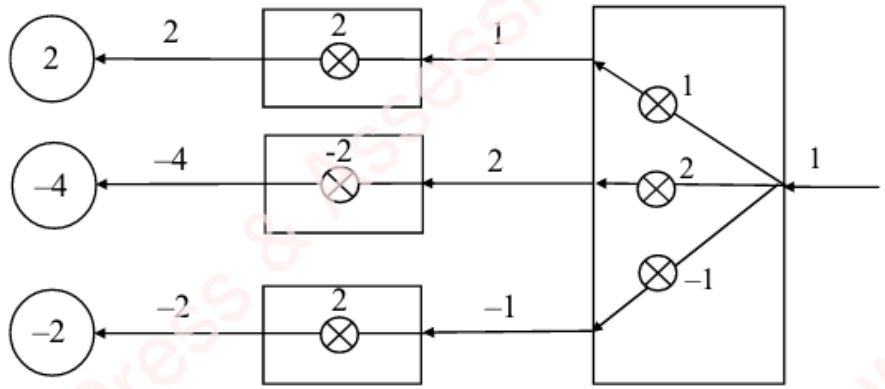


Figure 5.20 Backward pass to compute the gradient $\nabla g(\theta)$ at $\theta = [1, -1, 1]^T$ for function $g(\theta) = \theta_1^2 + 2\theta_2^2 - \theta_3^2$.



Source [B.4], Chapter 5

A. Giovanidis 2022

Example 2

Figure 5.25 Forward pass for the calculation of the gradient $\nabla g(\theta)$ at $\theta = [1, -2, 1]^T$ for the computational graph in Fig. 5.15 with the elementary functions detailed in Example 5.25. Within each computational block, we have also indicated the local partial derivatives to be used in the backward phase by the corresponding input.

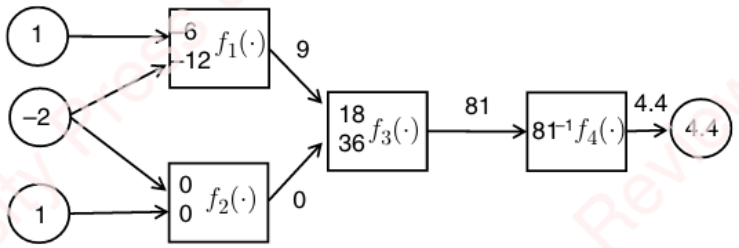
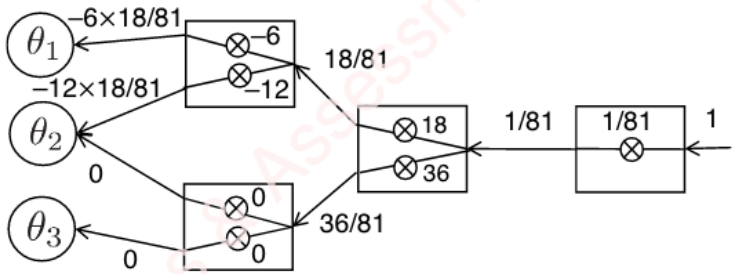


Figure 5.26 Backward pass for the calculation of the gradient $\nabla g(\theta)$ at $\theta = [1, -2, 1]^T$ for the computational graph in Fig. 5.15 with the elementary functions detailed in Example 5.25.



$$f_1(x_1, x_2) = f_2(x_1, x_2) = f_3(x_1, x_2) = (x_1 + 2x_2)^2, \qquad f_4(x_1) = \log(x_1)$$

Source [B.4], Chapter 5

A. Giovanidis 2022

Example 2

Figure 5.25 Forward pass for the calculation of the gradient $\nabla g(\theta)$ at $\theta = [1, -2, 1]^T$ for the computational graph in Fig. 5.15 with the elementary functions detailed in Example 5.25. Within each computational block, we have also indicated the local partial derivatives to be used in the backward phase by the corresponding input.

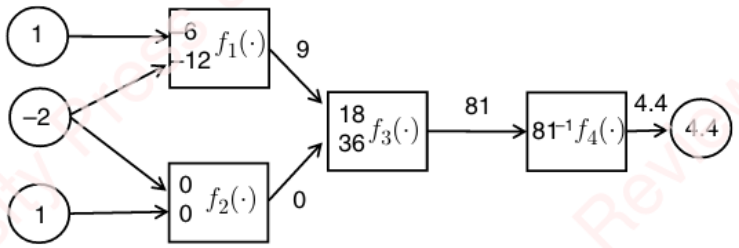
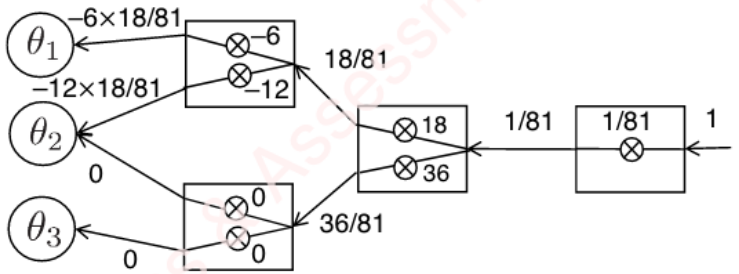


Figure 5.26 Backward pass for the calculation of the gradient $\nabla g(\theta)$ at $\theta = [1, -2, 1]^T$ for the computational graph in Fig. 5.15 with the elementary functions detailed in Example 5.25.

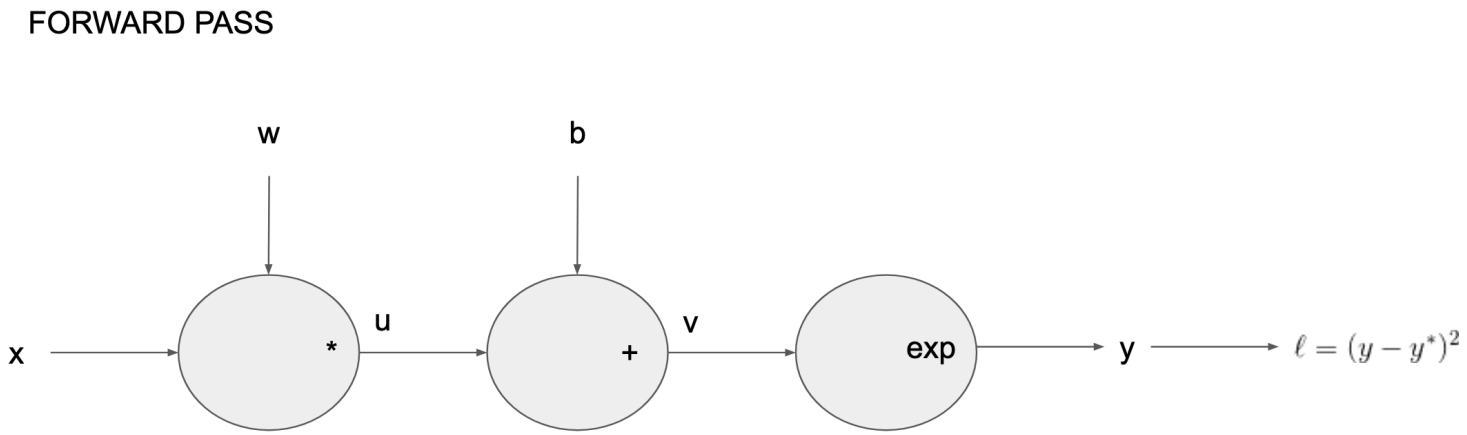


$$f_1(x_1, x_2) = f_2(x_1, x_2) = f_3(x_1, x_2) = (x_1 + 2x_2)^2, \qquad f_4(x_1) = \log(x_1)$$

Source [B.4], Chapter 5

Example 3

A. Giovanidis 2022

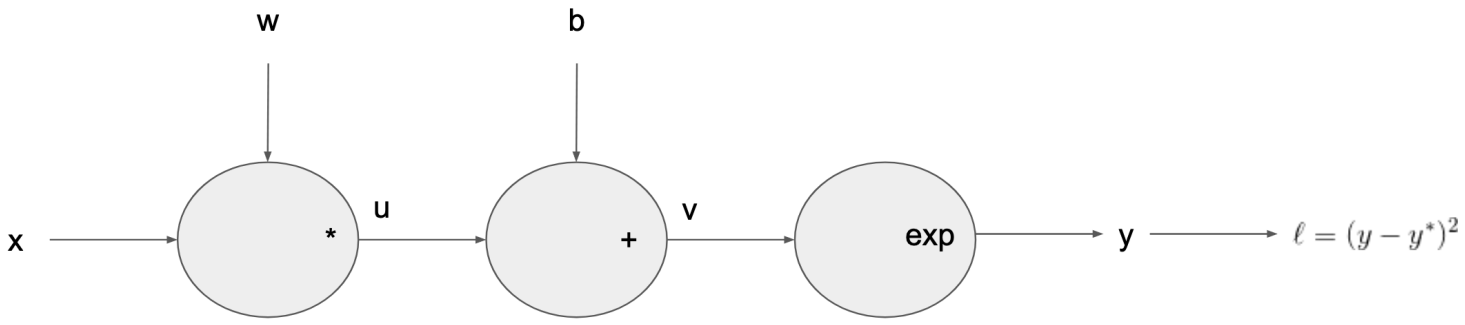


$u = wx$
$v = u + b$
$y = e^v$
$\ell = (y - y^*)^2$

Source [B.2], Module 2b - Automatic differentiation

Example 3

A. Giovanidis 2022



$$u = wx$$
$$v = u + b$$
$$y = e^v$$
$$\ell = (y - y^*)^2$$

$$\frac{\partial u}{\partial w} = x$$
$$\frac{\partial v}{\partial b} = 1, \quad \frac{\partial v}{\partial u} = 1$$
$$\frac{\partial y}{\partial v} = e^v = y.$$

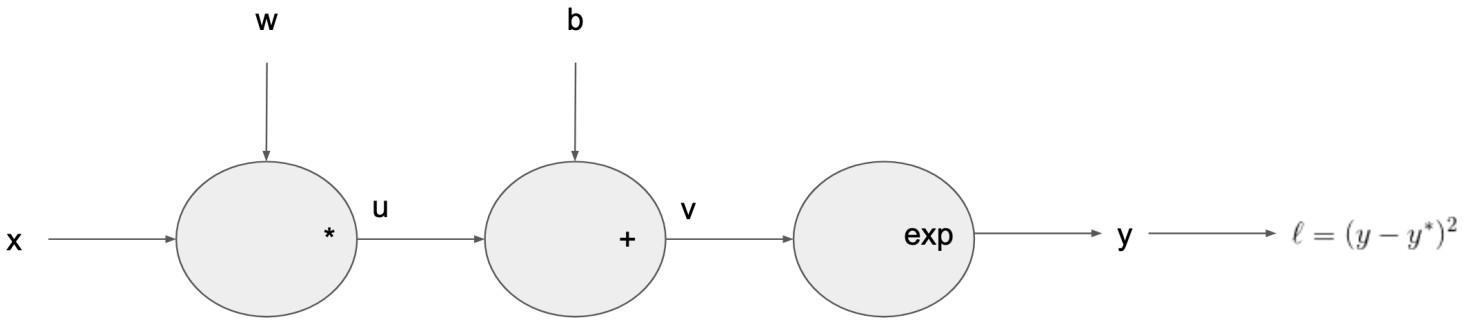
$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w}$$
$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial b}$$

Source [B.2], Module 2b - Automatic differentiation

Example 3

A. Giovanidis 2022

CHAIN RULE



$$u = wx$$
$$v = u + b$$
$$y = e^v$$
$$\ell = (y - y^*)^2$$

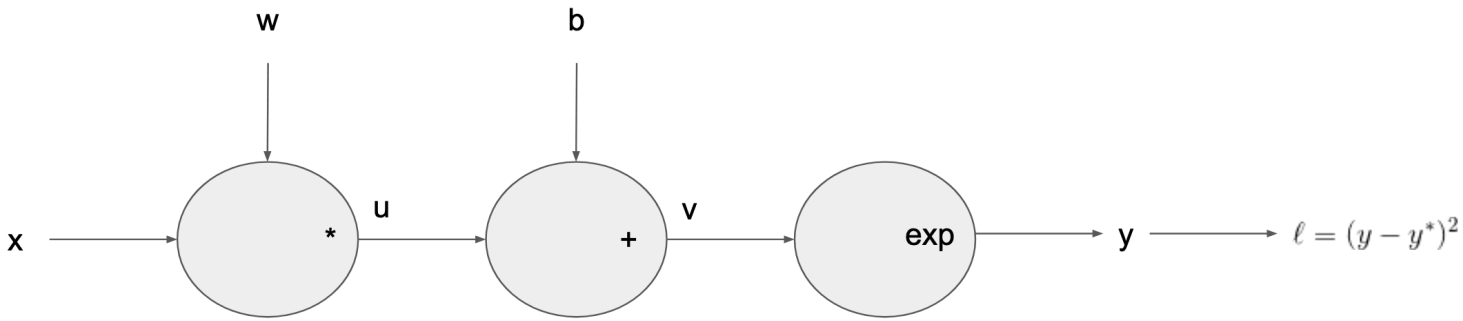
$$\frac{\partial u}{\partial w} = x$$
$$\frac{\partial v}{\partial b} = 1, \quad \frac{\partial v}{\partial u} = 1$$
$$\frac{\partial y}{\partial v} = e^v = y.$$

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w} \qquad \frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w}$$
$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial b} \qquad \frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial b}$$

Source [B.2], Module 2b - Automatic differentiation

Example 3

A. Giovanidis 2022



$$u = wx$$
$$v = u + b$$
$$y = e^v$$
$$\ell = (y - y^*)^2$$

$$\frac{\partial u}{\partial w} = x$$
$$\frac{\partial v}{\partial b} = 1, \quad \frac{\partial v}{\partial u} = 1$$
$$\frac{\partial y}{\partial v} = e^v = y.$$

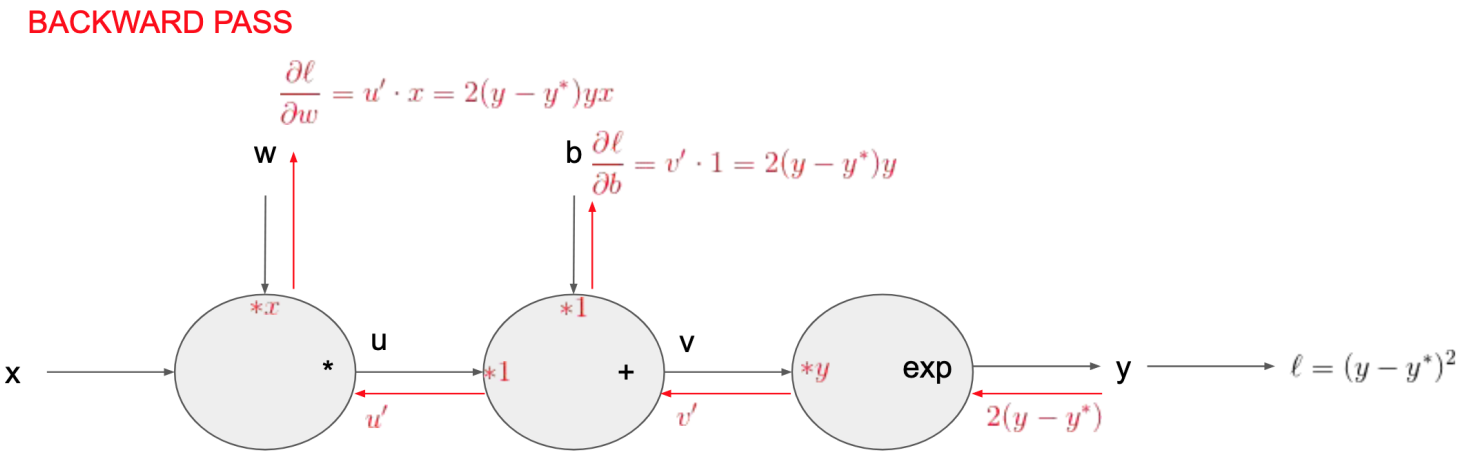
$$\frac{\partial \ell}{\partial w} = 2(y - y^*) \cdot y \cdot 1 \cdot x$$
$$\frac{\partial \ell}{\partial b} = 2(y - y^*) \cdot y \cdot 1$$

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w} \qquad \frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w}$$
$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial b} \qquad \frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial b}$$

Source [B.2], Module 2b - Automatic differentiation

Example 3

A. Giovanidis 2022



Source [B.2], Module 2b - Automatic differentiation

Practical implementation

A. Giovanidis 2022

« Deep learning for humans »

Keras

By François Cholet (Google)
Hight level API
Part on TensorFlow since 2017
MIT licence

TensorFlow

Most used DL framework
Supported by Google
Low level API – an hard way
Apache licence

Widely used in the implementation of practical solutions

PYTORCH

From Torch library
Supported by Facebook
BSD licence

Widely used in the field of AI research

python™ NumPy jupyter

A. Giovanidis 2022

END