

Introduction to Deep Learning

Mnacho Echenim

Grenoble INP-Ensimag

2022-2023



Notes

Evaluating how well an MLP is doing (supervised learning)

Problem we have a set of samples $S = \{(\alpha_{(i)}^0, \rho_{(i)}) \mid i = 1, \dots, N\}$ and an MLP parameterized by a set of weights and biases collectively denoted by θ

Goal find θ^* such that, for all $i = 1, \dots, N$, when the input neurons are fed x_i , the output activation $\widehat{\rho}_{(i)}$ is a **good** approximation of $\rho_{(i)}$

Definition

Cost function: positive function $\mathcal{E}(S, \theta)$ meant to measure how well an MLP is doing

Standard assumption

The cost function can be written as an average of individual cost functions over all samples: $\mathcal{E}(S, \theta) = \frac{1}{N} \sum_{i=1}^N \mathcal{C}(\theta, \rho_{(i)}, \widehat{\rho}_{(i)})$

Reformulation

Goal: solve the optimization problem $\theta^* = \arg \min_{\theta} \mathcal{E}(S, \theta)$



Notes

Examples of individual cost functions

- Mean square error (MSE) $(y, \hat{y}) \mapsto \frac{1}{2} \cdot (y - \hat{y})^2$
- L_1 error $(y, \hat{y}) \mapsto |y - \hat{y}|$ (robust regression)
- $(y, \hat{y}) \mapsto \log(1 + \exp(-y\hat{y}))$ (logistic regression)
- $(y, \hat{y}) \mapsto \max(0, -y\hat{y})$ (binary classification)

Notes

Gradient descent

Goal: find the minimum of $\mathcal{E}(S, \theta)$ using gradient descent: if θ_k denotes the network parameters at iteration k then

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_{\theta} \mathcal{E}(S, \theta_k)$$

The **hyperparameter** η is called the **learning rate**. In our case, the parameter update can be written as:

$$\theta_{k+1} \leftarrow \theta_k - \frac{\eta}{N} \sum_{i=1}^N \nabla_{\theta} \mathcal{C}(\theta_k, \rho_{(i)}, \widehat{\rho}_{(i)})$$

Issues:

- Computing a single gradient can be very long
- Vectorization is not possible for large samples

Notes

Stochastic gradient descent

- **Idea:** why not use the gradient from a single **arbitrary** sample?

The update rule becomes $\theta_{k+1} \leftarrow \theta_k - \eta \cdot \nabla_{\theta} \mathcal{C}(\theta_k, \rho_{(i_k)}, \widehat{\rho_{(i_k)}})$, where i_k is a random variable that follows the discrete uniform distribution on $\{1, \dots, N\}$

- **Why random?**

- **Features**

- ▶ Faster to compute
- ▶ Can be used for online learning
- ▶ Can avoid local minima, saddle points (more on this later)

Notes

Features of SGD

- Convergence can be much slower than for gradient descent
- Problems arise when we are close to the optimal value θ^* : the noise becomes problematic
- Can this noise be reduced?

Notes

A trade-off: Mini-batch gradient descent

- **Principle:** use M arbitrary samples to optimize cost function
The update rule becomes

$$\theta_{k+1} \leftarrow \theta_k - \frac{\eta}{M} \sum_{j=1}^M \nabla_{\theta} \mathcal{C}(\theta_k, \rho_{(i_{k_j})}, \widehat{\rho_{(i_{k_j})}}),$$

where i_{k_j} is a random variable that follows the discrete uniform distribution on $\{1, \dots, N\}$

- **Features**
 - ▶ Less noisy approximation of real gradient
 - ▶ Computation cost can be controlled
 - ★ Mini-batch size

Notes

Summary

Given an MLP and a sample set of size N :

Method	Updates per epoch	Computations per update

Notes

About backpropagation

- Computation of gradients during the training phase
 - ▶ How should weights and biases be updated to take into account that the output of the network is not correct?
 - ▶ Very efficient
 - ▶ One of the reasons deep neural networks are so successful
- A special case of automatic differentiation
 - ▶ Generally presented with **computational graphs**
 - ▶ Modern ML frameworks implement the general case
 - ▶ Here, we focus on the derivation of backpropagation equations for neural networks

Notes



A word of warning

- The goal of this part is to **derive** the backpropagation rules, not simply verify that they work
- The presentation will be quite formal, to ensure each step is well understood
- This makes notations quite heavy
- An interesting exercise is to try to derive the rules by yourselves, using the lighter notations that are more standard

Notes



Some definitions

Definition

Given the matrices M_1, \dots, M_p , we define the bloc matrix

$$\text{diag}(M_1, \dots, M_p) \stackrel{\text{def}}{=} \begin{pmatrix} M_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & M_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & M_p \end{pmatrix}$$

Definition

Given $A, B \in \mathbb{R}^{n \times m}$, the **Hadamard product** of A and B , denoted by $A \odot B$, is the matrix such that $(A \odot B)_{i,j} = A_{i,j} \cdot B_{i,j}$ (pointwise multiplication)

Notes

The chain rule

Definition

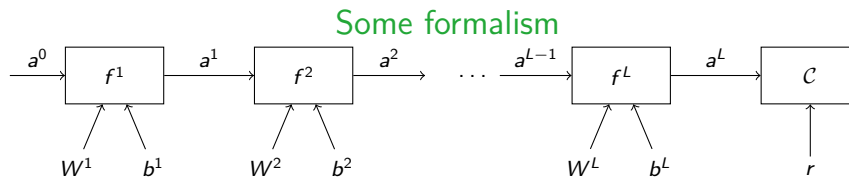
The **Jacobian matrix** of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
 $a \mapsto (f_1(a), \dots, f_m(a))^T$

is the function that associates to $a \in \mathbb{R}^n$ the matrix in $\mathbb{R}^{m \times n}$ denoted by $\frac{\partial f}{\partial a}(a)$, where $\left(\frac{\partial f}{\partial a}(a)\right)_{i,j} = \frac{\partial f_i}{\partial a_j}(a)$

Proposition

Given: $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ $g : \mathbb{R}^m \rightarrow \mathbb{R}^p$ $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$
 $a \mapsto f(a)$ $y \mapsto g(y)$ $a \mapsto g \circ f(a)$

Notes



Where for $j = 1, \dots, L$:

- $a^{j-1} \in \mathbb{R}^{n_{j-1}}$ represents the formal input of layer j
- $b^j \in \mathbb{R}^{n_j}$ and $W^j = (w_1^j, \dots, w_{n_j}^j)$, where for $k = 1, \dots, n_j$, $w_k^j \in \mathbb{R}^{n_{j-1}}$ represents the weights for neuron k at layer j
- f^j represents the computation performed at layer j :

$$f^j: (\mathbb{R}^{n_{j-1}})^{n_j+1} \times \mathbb{R}^{n_j} \rightarrow \mathbb{R}^{n_j}$$

$$a^{j-1}, W^j, b^j \mapsto f^j(a^{j-1}, W^j, b^j)$$

\mathcal{C} is the error function:

$$\mathcal{C}: \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \rightarrow \mathbb{R}^+$$

$$a^L, r \mapsto \mathcal{C}(a^L, r)$$

Notes

Reminders

- Parameters

Name	Formal	Actual	Dimension
Activation of layer j	a^j	α^j	\mathbb{R}^{n_j}
Weights at layer j	W^j	$\Omega^j = (\omega_1^j, \dots, \omega_{n_j}^j)$	$\mathbb{R}^{n_{j-1} \times n_j}$
Bias at layer j	b^j	β^j	\mathbb{R}^{n_j}
Expected output	r	ρ	\mathbb{R}

- Partial derivatives

$$\frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j) \in \mathbb{R}^{n_j \times n_{j-1}}$$

$$\frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j) \in \mathbb{R}^{n_j \times n_{j-1}}, \text{ for } k = 1, \dots, n_j$$

$$\frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j) \in \mathbb{R}^{n_j \times n_j}$$

Notes

What we want to compute

Let g^i represent the computation of the first i layers of the network:

Let \mathcal{E} represent the output of the error function:

Output of i^{th} layer of the network, given the inputs $\alpha^0, \Omega^1, \beta^1, \dots, \Omega^i, \beta^i$:

Error of the network when expected result is ρ :

Goal

Compute $\frac{\partial \mathcal{E}}{\partial w_k}(\alpha^0, \Omega^1, \beta^1, \dots, \Omega^L, \beta^L, \rho)$ and $\frac{\partial \mathcal{E}}{\partial b^l}(\alpha^0, \Omega^1, \beta^1, \dots, \Omega^L, \beta^L, \rho)$

Notes

Using the chain rule

Chain rule on the error function:

$$\frac{\partial \mathcal{E}}{\partial w_k}(\alpha^0, \dots, \Omega^L, \beta^L, \rho) = \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \cdot \frac{\partial g^L}{\partial w_k}(\alpha^0, \dots, \Omega^L, \beta^L)$$

$$\frac{\partial \mathcal{E}}{\partial b^l}(\alpha^0, \dots, \Omega^L, \beta^L, \rho) = \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \cdot \frac{\partial g^L}{\partial b^l}(\alpha^0, \dots, \Omega^L, \beta^L)$$

Chain rule on the output of the network when $j = L$:

$$\frac{\partial g^L}{\partial w_k}(\alpha^0, \dots, \Omega^L, \beta^L) = \frac{\partial f^L}{\partial w_k}(\alpha^{L-1}, \Omega^L, \beta^L)$$

$$\frac{\partial g^L}{\partial b^L}(\alpha^0, \dots, \Omega^L, \beta^L) = \frac{\partial f^L}{\partial b^L}(\alpha^{L-1}, \Omega^L, \beta^L)$$

Notes

Using the chain rule (2)

Chain rule on the output of the network when $j < L$:

$$\begin{aligned}\frac{\partial g^L}{\partial w_k^j}(\alpha^0, \dots, \Omega^L, \beta^L) &= \left[\prod_{i=L}^{j+1} \frac{\partial f^i}{\partial a^{i-1}}(\alpha^{i-1}, \Omega^i, \beta^i) \right] \cdot \frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j) \\ \frac{\partial g^L}{\partial b^j}(\alpha^0, \dots, \Omega^L, \beta^L) &= \left[\prod_{i=L}^{j+1} \frac{\partial f^i}{\partial a^{i-1}}(\alpha^{i-1}, \Omega^i, \beta^i) \right] \cdot \frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)\end{aligned}$$

Notes

Recap

Consider the inputs $\alpha^0, \Omega^1, \beta^1, \dots, \Omega^L, \beta^L$ and the expected result ρ , and let

$$\mathcal{P}^{L+1} \stackrel{\text{def}}{=} \frac{\partial \mathcal{C}}{\partial a^L}(\alpha^L, \rho) \quad \mathcal{P}^j \stackrel{\text{def}}{=} \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j)$$

Then for $j = 1, \dots, L$, we have $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and

$$\frac{\partial \mathcal{E}}{\partial w_k^j}(\alpha^0, \dots, \Omega^L, \beta^L, \rho) = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j)$$

$$\frac{\partial \mathcal{E}}{\partial b^j}(\alpha^0, \dots, \Omega^L, \beta^L, \rho) = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)$$

At layer j , we receive \mathcal{P}^{j+1} and we need to compute $\frac{\partial f^j}{\partial a^{j-1}}(\alpha^{j-1}, \Omega^j, \beta^j)$, $\frac{\partial f^j}{\partial w_k^j}(\alpha^{j-1}, \Omega^j, \beta^j)$ and $\frac{\partial f^j}{\partial b^j}(\alpha^{j-1}, \Omega^j, \beta^j)$

Notes

Same recap, with simplified notations

Consider the inputs $\alpha^0, \Omega^1, \beta^1, \dots, \Omega^L, \beta^L$ and the expected result ρ , and let

$$\mathcal{P}^{L+1} \stackrel{\text{def}}{=} \frac{\partial \mathcal{C}}{\partial a^L} \quad \mathcal{P}^j \stackrel{\text{def}}{=} \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial a^{j-1}}$$

Then for $j = 1, \dots, L$, we have $\mathcal{P}^j \in \mathbb{R}^{1 \times n_{j-1}}$ and

$$\frac{\partial \mathcal{E}}{\partial w_k^j} = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial w_k^j}$$

$$\frac{\partial \mathcal{E}}{\partial b^j} = \mathcal{P}^{j+1} \cdot \frac{\partial f^j}{\partial b^j}$$

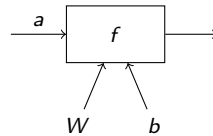
At layer j , we need to compute $\frac{\partial f^j}{\partial a^{j-1}}$, $\frac{\partial f^j}{\partial w_k^j}$ and $\frac{\partial f^j}{\partial b^j}$

Notes

Partial derivatives for a single layer

For a layer with n inputs and m neurons we have the following:

- $a \in \mathbb{R}^n$
- $b = (b_1, \dots, b_m)^T$
and $W = (w_1, \dots, w_m)$,
where for $p = 1, \dots, m$, $w_p \in \mathbb{R}^n$
- f is defined by:



$$f: (\mathbb{R}^n)^{m+1} \times \mathbb{R}^m \rightarrow \mathbb{R}^m$$

$$a, W, b \mapsto [h(a, w_1, b_1), \dots, h(a, w_m, b_m)]^T$$

- $h = \Phi \circ \Psi$, where

$$\begin{aligned} \Phi: \mathbb{R} &\rightarrow \mathbb{R} & \text{and} & & \Psi: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} &\rightarrow \mathbb{R} \\ z &\mapsto \Phi(z) & & & a, w, b &\mapsto w^T a + b \end{aligned}$$

Notes

Partial derivatives for a single neuron

- Input: α (neuron input), β (bias) and ω (weights for the neuron)
 - ▶ Let ζ denote the net input of the neuron
 - ▶ $\zeta = \omega^T \alpha + \beta = \Psi(\alpha, \omega, \beta)$
- Output: $h(\alpha, \omega, \beta) = \Phi(\Psi(\alpha, \omega, \beta)) = \Phi(\zeta) = \Phi(\omega^T \alpha + \beta)$
- We have:

$$\frac{\partial h}{\partial a}$$

$$\frac{\partial h}{\partial w}$$

$$\frac{\partial h}{\partial b}$$

Notes

Back to partial derivatives for a layer

$f: a, W, b \mapsto [h(a, w_1, b_1), \dots, h(a, w_m, b_m)]^T$

Input α, Ω, β , where $\Omega = (\omega_1, \dots, \omega_m)$

$$\frac{\partial f}{\partial a} = \begin{pmatrix} \Phi'(\zeta_1) \cdot \omega_1^T \\ \vdots \\ \Phi'(\zeta_m) \cdot \omega_m^T \end{pmatrix}$$

$$\frac{\partial f}{\partial w_k} = \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k) \cdot \alpha^T \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{line } k$$

$$\frac{\partial f}{\partial b_k} = (0, \dots, \Phi'(\zeta_k), \dots, 0)^T$$

$$\frac{\partial f}{\partial b} = \text{diag}(\Phi'(\zeta_1), \dots, \Phi'(\zeta_m))$$

Notes

Back to partial derivatives for an entire network

$$\mathcal{P}^{L+1} = \frac{\partial \mathcal{C}}{\partial \mathbf{a}^L} \quad \text{and} \quad \mathcal{P}^j = \mathcal{P}^{j+1} \cdot \begin{pmatrix} \Phi'(\zeta_1^j) \cdot [\omega_1^j]^T \\ \vdots \\ \Phi'(\zeta_{n_j}^j) \cdot [\omega_{n_j}^j]^T \end{pmatrix}$$

$$\frac{\partial \mathcal{E}}{\partial w_k^j} = \mathcal{P}^{j+1} \cdot \begin{pmatrix} 0 \\ \vdots \\ \Phi'(\zeta_k^j) \cdot [\alpha^{j-1}]^T \\ \vdots \\ 0 \end{pmatrix} \leftarrow \text{line } k$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{b}^j} = \mathcal{P}^{j+1} \cdot \text{diag}(\Phi'(\zeta_1^j), \dots, \Phi'(\zeta_{n_j}^j))$$

Notes

Back to gradient descent

- We are interested in computing

$$\triangleright \nabla_{w_k^j} \mathcal{E} \stackrel{\text{def}}{=} \left[\frac{\partial \mathcal{E}}{\partial w_k^j} \right]^T$$

$$\triangleright \nabla_{b^j} \mathcal{E} \stackrel{\text{def}}{=} \left[\frac{\partial \mathcal{E}}{\partial b^j} \right]^T$$

- Weights are stored in a matrix: $\Omega^j = (\omega_1^j, \dots, \omega_{n_j}^j) \in \mathbb{R}^{n_{j-1} \times n_j}$

- If we define

$$\nabla_{W^j} \mathcal{E} \stackrel{\text{def}}{=} \begin{pmatrix} \nabla_{w_1^j} \mathcal{E} & \cdots & \nabla_{w_{n_j}^j} \mathcal{E} \end{pmatrix}$$

- Then parameters updates for stochastic gradient descent become

$$\Omega^j \leftarrow \Omega^j - \eta \cdot \nabla_{W^j} \mathcal{E}$$

$$\beta_j \leftarrow \beta_j - \eta \cdot \nabla_{b^j} \mathcal{E}$$

Notes

Effective computations

$$\begin{aligned}\mathcal{P}^j &= \mathcal{P}^{j+1} \cdot \begin{pmatrix} \Phi'(\zeta_1^j) \cdot [\omega_1^j]^T \\ \vdots \\ \Phi'(\zeta_{n_j}^j) \cdot [\omega_{n_j}^j]^T \end{pmatrix} = \mathcal{P}^{j+1} \cdot \text{diag}(\Phi'(\zeta_1^j), \dots, \Phi'(\zeta_{n_j}^j)) \cdot [\Omega^j]^T \\ &= [\Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T]^T \cdot [\Omega^j]^T\end{aligned}$$

$$\begin{aligned}\nabla_{W^j} \mathcal{E} &= \begin{pmatrix} \dots & \underbrace{\begin{pmatrix} 0, \dots, 0, \alpha^{j-1} \cdot \Phi'(\zeta_k^j), 0, \dots, 0 \end{pmatrix}}_{\substack{\text{column } k \\ k^{\text{th}} \text{ vector}}} [\mathcal{P}^{j+1}]^T & \dots \end{pmatrix} \\ &= (\alpha^{j-1} \cdot (\Phi'(\zeta_1^j) \cdot \mathcal{P}_1^{j+1}) \quad \dots \quad \alpha^{j-1} \cdot (\Phi'(\zeta_{n_j}^j) \cdot \mathcal{P}_{n_j}^{j+1})) \\ &= \alpha^{j-1} \cdot [\Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T]^T \\ \nabla_{b^j} \mathcal{E} &= \Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T\end{aligned}$$

Notes

Backpropagation rules

Let $\mathcal{B}^j \stackrel{\text{def}}{=} \Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T \in \mathbb{R}^{n_j \times 1}$ for $j = 1, \dots, L-1$, so that

$$\mathcal{P}^j = [\Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T]^T \cdot [\Omega^j]^T = [\mathcal{B}^j]^T \cdot [\Omega^j]^T = [\Omega^j \mathcal{B}^j]^T$$

Recall also that $\mathcal{P}^{L+1} = \frac{\partial \mathcal{C}}{\partial \mathbf{a}^L}$

Notes

Backpropagation and gradient computation

Input: A network with L layers

Input: α^0 that has been forward propagated

Input: ρ the expected output

```
1  $\mathcal{B}^L \leftarrow \Phi'(\zeta^L) \odot \nabla_{a^L} \mathcal{C}(\alpha^L, \rho);$   
2  $[\mathcal{P}^L]^T \leftarrow \Omega^L \cdot \mathcal{B}^L;$   
3 for  $j \leftarrow L - 1$  to 1 do  
4    $\mathcal{B}^j \leftarrow \Phi'(\zeta^j) \odot [\mathcal{P}^{j+1}]^T;$   
5    $[\mathcal{P}^j]^T \leftarrow \Omega^j \cdot \mathcal{B}^j;$   
6 end
```

Algorithm 1: Backpropagation algorithm

Input: A network with L layers

Input: α^0 that has been forward propagated

Input: $(\mathcal{B}^1, \dots, \mathcal{B}^L)$ that have been updated by backpropagation

```
1 for  $j \in \{1, \dots, L\}$  do  
2    $\text{Gradient}(\Omega^j) \leftarrow \alpha^{j-1} \cdot [\mathcal{B}^j]^T;$   
3    $\text{Gradient}(\beta^j) \leftarrow \mathcal{B}^j;$   
4 end
```

Algorithm 2: Gradient computation

Notes

Bonus: verifying that gradients are correctly computed

Input: Network N with L layers

Input: α^0, ρ

Input: ε, δ

```
1  $N.\text{Propagate}(\alpha^0);$   
2  $N.\text{Learn}(N.\text{Output}, \rho);$   
3 for  $l \in \{1, \dots, L\}$  do  
4   let  $\Omega^l$  be the weight matrix for layer  $l$ ;  
5   for  $(i, j) \in \llbracket 1, n_l \rrbracket \times \llbracket 1, m_l \rrbracket$  do  
6     let  $\omega_{i,j} = \Omega^l[i, j]$  and  $\omega_{i,j}^+ = \omega_{i,j} \cdot (1 + \varepsilon)$  and  $\omega_{i,j}^- = \omega_{i,j} \cdot (1 - \varepsilon);$   
7     let  $N^+ = N[\Omega^l[i, j] \leftarrow \omega_{i,j}^+]$  and  $N^- = N[\Omega^l[i, j] \leftarrow \omega_{i,j}^-];$   
8      $N^+.\text{Propagate}(\alpha^0);$   
9      $N^-.\text{Propagate}(\alpha^0);$   
10    let  $C^+ = \mathcal{C}(N^+.\text{Output}, \rho)$  and  $C^- = \mathcal{C}(N^-.\text{Output}, \rho);$   
11    let  $A_{i,j}^l = \frac{C^+ - C^-}{2\varepsilon \cdot \omega_{i,j}};$   
12    let  $G_{i,j}^l = N.\text{Gradient}(\Omega^l)[i, j];$   
13    Assert  $\frac{|A_{i,j}^l - G_{i,j}^l|}{|A_{i,j}^l| + |G_{i,j}^l|} \leq \delta$   
14  end  
15 end
```

Notes
