Introduction to deep learning

Lecture 2

Sam van Gool
vangool@irif.fr

Master 2 informatique
Université Paris Cité

14 January 2025

Today's lecture

Last week

CM:

- Machine learning basics:
  - supervised vs. unsupervised learning
  - continuous vs. discrete output (regression vs. classification)
- Linear regression model (example: surface vs. price)
- Minimizing the total square loss
- Numpy basics

TP:

- Implementing *linear regression* in Numpy
- A first *learning loop*
- *Exploding gradients* by turning the learning rate too high
- Numerical gradient check

Reminder: Course webpage: `https://samvangool.net/iap/`

TO DO last week

- Finish TP 1.
- Read Chapters 1 & 2 in [P] (Section 1.3 is optional).
- Try Problems 2.1 and 2.2 in [P].
- If you are not yet familiar with `numpy`, work through the start of a tutorial, for example [Numpy quickstart] or [Maximov 2020].
  - We will use some *matrix multiplication* and *broadcasting* later.
  - The same basic skills will apply to `pytorch`.

[P, Problem 2.1]

To walk "downhill" on the loss function:

$$L[\phi_0, \phi_1] = \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

we measure its gradient with respect to the parameters ϕ_0 and ϕ_1.
Calculate expressions for the slopes $\partial L / \partial \phi_0$ and $\partial L / \partial \phi_1$.

[P, Problem 2.2]

Show that we can find the minimum of the loss function in closed form by setting the expression for the derivatives from problem 2.1 to zero and solving for ϕ_0 and ϕ_1.

Note that this works for linear regression but not for more complex models; this is why we use iterative model fitting methods like gradient descent.

This week

- *Softmax* regression: a different model & loss;
- *Minibatch stochastic* gradient descent;
- Application to image classification (TP2).

Today's lecture

1. Recap
2. Softmax regression
3. Stochastic gradient descent

Classification

- In a problem with a *discrete* output, we want to predict a class c between 0 and $C - 1$ where C is the number of classes.
- For example: Recognize an image of a handwritten number.
- The MNIST dataset of handwritten digits (Lecun et al. 1998):

Reflection

How would you do this with a linear regression?

Classification

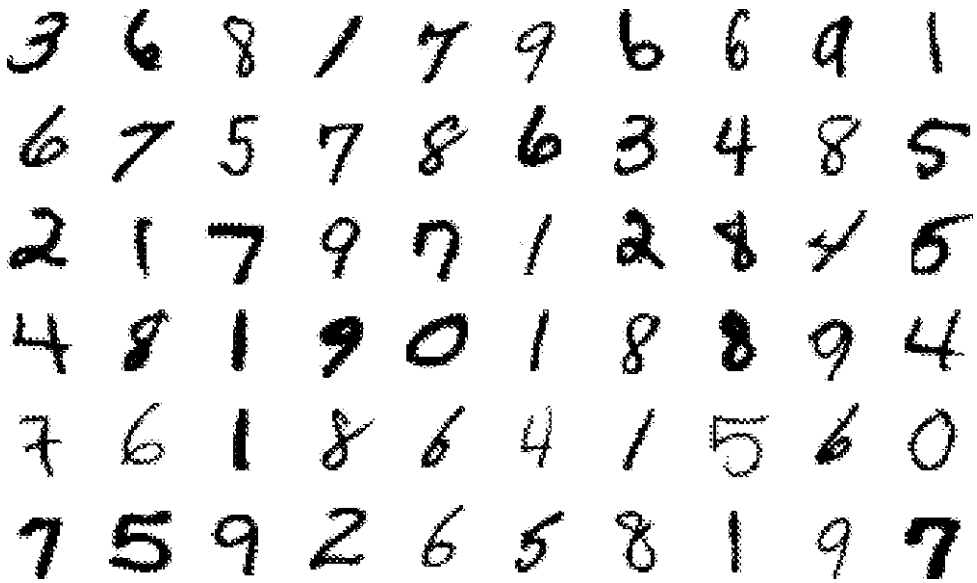
- In a problem with a *discrete* output, we want to predict a class c between 0 and $C - 1$ where C is the number of classes.
- For example: Recognize an image of a handwritten number.
- The MNIST dataset of handwritten digits (Lecun et al. 1998):

Reflection

How would you do this with a linear regression?

Classification

- In a problem with a *discrete* output, we want to predict a class c between 0 and $C-1$ where C is the number of classes.
- For example: Recognize an image of a handwritten number.
- The MNIST dataset of handwritten digits (Lecun et al. 1998):



Reflection

How would you do this with a linear regression?

Why linear regression does not apply here

Compared to last week's linear regression, we have two difficulties:

1. The **input** of a linear regression was one real number, not an image.
2. The **output** of a linear regression is a real number $y \in \mathbb{R}$, not a class.

Why linear regression does not apply here

Compared to last week's linear regression, we have two difficulties:

1. The **input** of a linear regression was one real number, not an image.
2. The **output** of a linear regression is a real number $y \in \mathbb{R}$, not a class.

We will now discuss solutions to these problems.

Fully connected linear layer

- Suppose we have an input $\mathbf{x} = (x_1, x_2, x_3, x_4)$ of 4 real numbers and we want an output $\mathbf{o} = (o_1, o_2, o_3)$ of 3 real numbers.
- Suppose also that we have a parameter ϕ_{ij} for every $1 \leq i \leq 4$ and $0 \leq j \leq 3$.
- We define the output of the fully connected linear layer as:

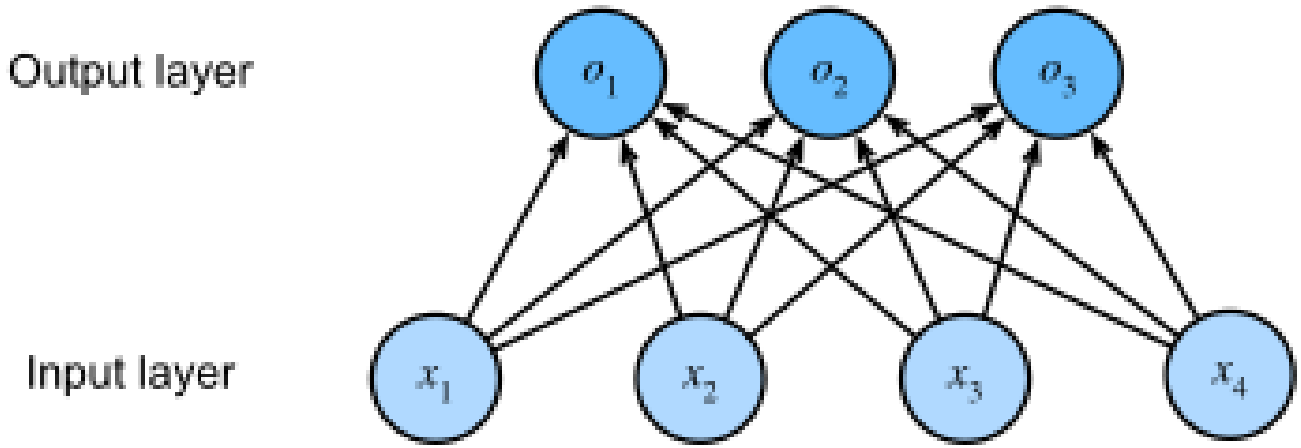
$$o_1 = \phi_{10} + \phi_{11}x_1 + \phi_{12}x_2 + \phi_{13}x_3 + \phi_{14}x_4$$
$$o_2 = \phi_{20} + \phi_{21}x_1 + \phi_{22}x_2 + \phi_{23}x_3 + \phi_{24}x_4$$
$$o_3 = \phi_{30} + \phi_{31}x_1 + \phi_{32}x_2 + \phi_{33}x_3 + \phi_{34}x_4$$

- The numbers ϕ_{10}, ϕ_{20}, and ϕ_{30} are bias parameters, all the other ϕ_{ij} are weight parameters.

Reflection

How can you define the fully connected linear layer using **matrices**?

Fully connected linear layer, visually



[Z, Figure 4.1.1]

Each arrow signifies that the value of this output coordinate *depends* on this value of the input coordinate.

Fully connected linear layer for an image

- Suppose we have an image of 28×28 pixels, and we want to output 10 numbers, one for each of the 10 possible output classes.
- A fully connected linear layer will have input dimension $28 \times 28 = 784$ and output dimension 10.
- The parameters are ϕ_{ij} with $1 \leq i \leq 784$ and $0 \leq j \leq 10$.
- We can organize the parameters in a matrix $\mathbf{\Omega}$ of size 784×10 and a column vector $\boldsymbol{\beta}$ of size 10. Total: 7850 parameters.

$$\mathbf{o} = \mathbf{x}^{\mathrm{T}} \cdot \mathbf{\Omega} + \boldsymbol{\beta}$$

Greek letter alert

Ω is the capital letter 'Omega'.

β is the lowercase 'beta'.

x^{T} is the *transpose* of x.

How does each parameter influence the output?

- The partial derivative $\frac{\partial f}{\partial z}$ measures the influence of the input variable z on the output f.
- The weight parameter ϕ_{23} influences o_2:

$$\frac{\partial o_2}{\partial \phi_{23}} = \frac{\partial}{\partial \phi_{23}}(\phi_{20} + \phi_{21}x_1 + \phi_{22}x_2 + \phi_{23}x_3 + \phi_{24}x_4) = x_3.$$

- The parameter ϕ_{23} does not influence o_1 and o_3.
- The bias parameter ϕ_{30} only influences o_3:

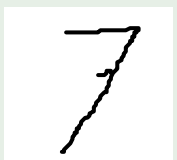
$$\frac{\partial o_3}{\partial \phi_{30}} = \frac{\partial}{\partial \phi_{23}}(\phi_{30} + \phi_{31}x_1 + \phi_{32}x_2 + \phi_{33}x_3 + \phi_{34}x_4) = 1.$$

- We can organize these quantities in matrices, called the gradient.

Hard and soft classification

- Hard assignment: We want to choose one class for an example.
- Soft assignment: We want to assess the probability that an example belongs to class k.
- Practice: First do soft assignment, then answer 'most likely class'.

Example



This image may get *hard* assignment 1 or 7, but *soft* assignment:

$$\mathbf{p} = (.01, .3, .01, .02, .01, .01, .01, .6, .01, .02)$$

expressing a 30% chance that it belongs to class 1 and 60% that it belongs to class 7.

Predicting a probability distribution

- A discrete probability distribution with C classes is a vector

$$\mathbf{p} = (p_0, p_1, \ldots, p_{C-1})$$

such that $0 \leq p_i \leq 1$ for every i, and

$$\sum_{i=0}^{C-1} p_i = 1.$$

- A soft classification model is a function that outputs, given an input $\mathbf{x}$, a discrete probability distribution $\mathbf{p} = f[\mathbf{x}, \phi]$.
- According to this model, the likelihood that input $\mathbf{x}$ has label $y = k$ is

$$\Pr(y = k|\mathbf{x}) = p_k.$$

- Here, $\Pr(A|B)$ is the *conditional probability* of event A, given B.

Softmax

- Let $\mathbf{o} \in \mathbb{R}^C$ be the length C vector output by a linear layer.
- To transform it into a probability distribution $\mathbf{p}$, we **assume**

$$\Pr(y = k) \sim \exp(o_k).$$

- This means that, for each $0 \leq k < C$, p_k is defined as

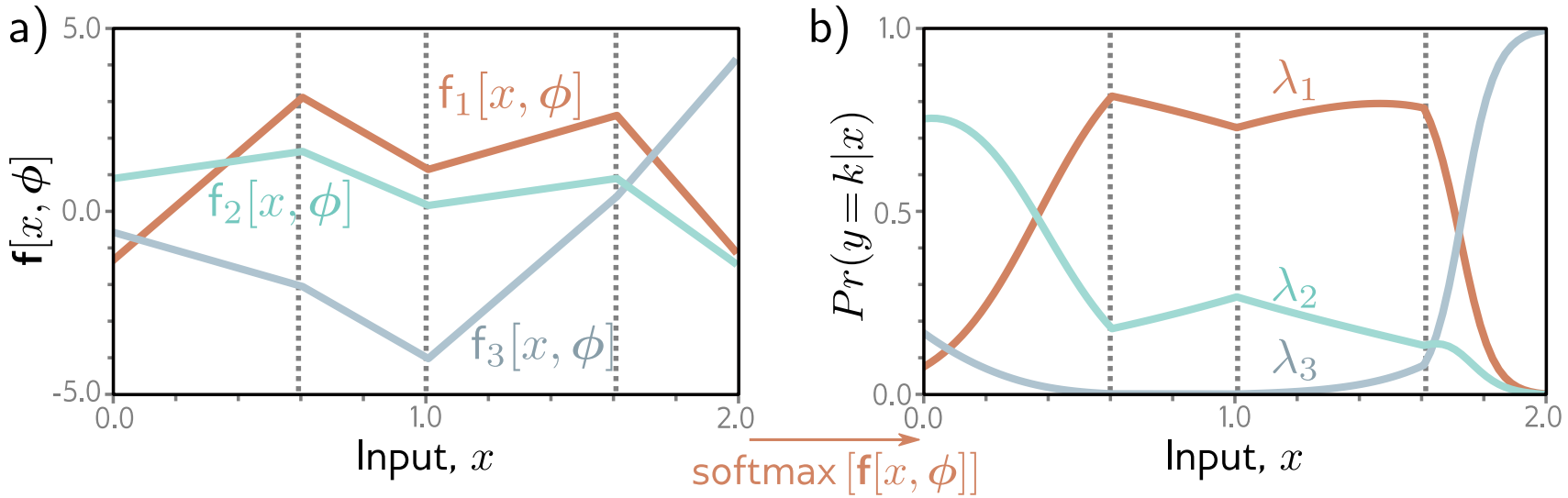
$$p_k \stackrel{\text{def}}{=} \frac{\exp(o_k)}{P(\mathbf{o})}, \text{ where } P(\mathbf{o}) \stackrel{\text{def}}{=} \sum_{i=0}^{C-1} \exp(o_i).$$

Math notation alert

We write $\exp(a)$ for e^a, the a-th power of $e \approx 2.718...$

- The function $\mathbb{R}^C \to [0,1]^C$ that sends $\mathbf{o}$ to $\mathbf{p}$ is called softmax.

Softmax visualized



[P, Figure 5.10]

- In this picture, $\mathbf{o} = (f_1[x, \phi], f_2[x, \phi], f_3[x, \phi])$ and $\mathbf{p} = (\lambda_1, \lambda_2, \lambda_3)$.
- For example, when $x = 1.0$, we have $\mathbf{o} \approx (1, 0, -4)$, so that $\mathbf{p} \approx (.72, .26, 0)$. (Numpy demo)

Maximum likelihood criterion

- How to measure the **quality** of a predicted probability distribution $\mathbf{p}$?
- Suppose the correct class for an input $\mathbf{x}$ is c. We would like p_c to be as close to 1 as possible.
- Recall: $\mathbf{p} = f[\mathbf{x}, \phi]$, for some parameters ϕ that we are trying to learn.
- If we have a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_I$, we would like to maximize the combined likelihood of the parameters. We aim to learn:

$$\underset{\phi}{\operatorname{argmax}} \prod_{i=1}^{I} \Pr(y_i = c_i | \mathbf{x}_i).$$

- This is called the maximum likelihood criterion.

Reflection

What happens to the quantity $\prod_{i=1}^{I} \Pr(y_i = c_i | \mathbf{x}_i)$ if many of the probabilities are close to 0? (Numpy demo)

Negative log-likelihood loss

- Instead, we aim to learn:

$$\underset{\phi}{\operatorname{argmax}} \log \prod_{i=1}^{I} \Pr(y_i = c_i|\mathbf{x}_i) = \underset{\phi}{\operatorname{argmax}} \sum_{i=1}^{I} \log \Pr(y_i = c_i|\mathbf{x}_i).$$

- Or actually, since we always *minimize* instead of maximize:

$$\underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{I} -\log \Pr(y_i = c_i|\mathbf{x}_i).$$

Math notation alert

$\log(a)$ is the **base** e logarithm of a.

So: $\log(2.718) \approx 1$, $\log(100) \approx 4.605$ because $e^{4.605} \approx 100$.

Cross-entropy loss

- Consider just one sample $\mathbf{x}$, with predicted probability vector $\mathbf{p}$. If the correct class is c, the negative log-likelihood loss is $-\log p_c$. It is sometimes called the cross-entropy loss.
- You will sometimes also see this loss function written as

$$\ell \stackrel{\text{def}}{=} -\sum_{j=0}^{C-1} y_j \log p_j$$

with $\mathbf{y} = (y_0, \ldots, y_{C-1})$ the one-hot encoding of the correct class c:

$$y_j = \begin{cases} 0 & \text{if } j \neq c \\ 1 & \text{if } j = c. \end{cases}$$

Reflection

Why is it the same? Which is better?

Gradient calculation

- To summarize, given an input (column vector) $\mathbf{x}$:
  1. We compute the logits $\mathbf{o} = \mathbf{x}^{\mathrm{T}} \cdot \boldsymbol{\Omega} + \boldsymbol{\beta}$.
  2. We compute the probability vector $\mathbf{p} = \mathrm{softmax}(\mathbf{o})$.
  3. If the correct class is c, the loss on $\mathbf{x}$ is $\ell \stackrel{\mathrm{def}}{=} -\log p_c$.
- We need to compute the **gradient** of this new loss function:

$$\mathbf{p}' \stackrel{\mathrm{def}}{=} \frac{\partial \ell}{\partial o_j} = \begin{cases} p_j & \text{if } j \neq c \\ p_j - 1 & \text{if } j = c. \end{cases}$$

- The gradient with respect to the parameters $\boldsymbol{\Omega}$ at input $\mathbf{x}$ is:

$$\frac{\partial \ell}{\partial \Omega} = \mathbf{x} \otimes \mathbf{p}'$$

and with respect to the bias vector: $\frac{\partial \ell}{\partial \boldsymbol{\beta}} = \mathbf{p}'$.

Today's lecture

1. Recap
2. Softmax regression
3. Stochastic gradient descent

Gradient descent

- We have a model f with parameters ϕ.
- We want to reduce a loss function L.
- We update the parameters by doing

$$\phi \leftarrow \phi - \eta \cdot \nabla L$$

where ∇L is the gradient of the loss function: the direction that most increases the loss.
The *hyperparameter* η is the learning rate.

Greek letter alert

The Greek letter η is pronounced 'eta'.
The symbol ∇ is an upside-down Greek letter, pronounced 'del' or 'nabla', which means 'Phoenician harp'.

Naive gradient descent

- Suppose $L\colon \mathbb{R}^n \to \mathbb{R}$ is a function of $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_n)$.
- The gradient ∇L at $\mathbf{u}$ is a vector of length n, with

$$(\nabla L)_i = \frac{\partial f}{\partial \phi_i}(\mathbf{u}),$$

  the partial derivative of L with respect to parameter ϕ_i.
- For example, L the total loss of the model over the dataset.
- Naive gradient descent: Compute the average gradient of the loss function over the entire dataset, and step down.
- Disadvantages:
  - Takes a long time before we make a single update;
  - The data may contain *redundancy*.

Stochastic gradient descent

- Stochastic gradient descent: For each sample, adjust the parameters using the gradient of the individual loss, ℓ.
- Why *stochastic*? Any single sample may be an outlier.
- Disadvantages:
  - May lead to chaotic process.
  - Can be slow: does not use *vectorization*.

Minibatch stochastic gradient descent

- Intermediate solution: Do not take full batch, do not take one sample, but take a minibatch.
- Fix a batch size B (usually 32, 64, 128, ...), and do the following:
  1. **Randomly** choose B samples from the data set, $\mathbf{x}_1, \ldots, \mathbf{x}_B$.
  2. Compute the gradient over the average loss:

$$G = \nabla \left[\frac{1}{B} \sum_{i=1}^{B} \ell(\mathbf{x}_i) \right]$$

  3. Update the parameters with learning rate η:

$$\phi \leftarrow \phi - \eta G.$$

  4. Repeat for a large number of minibatches.
- One epoch of training: Do this until you have used all samples in the dataset (except $\leq B - 1$ that couldn't be used to form a batch).

Coming up...

Next week we train our first *deep* network: the 'multi-layer perceptron'.

TO DO after today, before next Tuesday

- Finish TP2.
- Read [Z, pp. 82-88, pp. 126-148], [P, Sections 5.1–5.2].
- Do [Z, Section 4.4.7], Exercise 1 and [P, Problem 5.2].
- If you need a reminder of Linear Algebra and Calculus: read [Z, Section 2.3 and 2.4], or [P, Appendix B].

References

[P] Prince, S., **Understanding Deep Learning**, https://udlbook.com

[Z] Zhang, A. et al., **Dive into Deep Learning**, https://www.d2l.ai

Image sources:

- https://en.wikipedia.org/wiki/Omega
- https://en.wikipedia.org/wiki/Nevel_(instrument)