Machine learning

1 Introduction

1.1 What is ML?

The successes of ML

1.1.1 Algorithmic advances

- New neural network architectures to handle many kinds of data (text, vision, sounds, physical data, etc...)
- new training techniques, optimization algorithms
- new rules to be able to train these models at a large scale.

1.1.2 Computationnal advances and open source

- Use of efficient hardware to scale up models: gpus. Allows to perform many matrix multiplications efficiently super quickly.
- development of open-source tools that are easy to use for researchers that are not experts in code. Makes it easy to implement complicated neural networks, scale them, and compute gradients. Pytorch, Jax.
- You can code a kind of chat-gpt and train it in about 1k lines of code (cf nano-gpt)

1.1.3 ML breakthroughs

ML and deep learning have allowed many breakthroughs in the last 10 years

- Computer vision (image processing, image recognition, segmentation, etc...): most computer vision systems use neural networks. Also, automatic image generation.
- Natural language processing: automatic translation, and text generation
- Chemistry: protein folding prediction (alpha fold); one nobel prize for this this year

1.2 Two examples of modern machine learning success and their detailed pipelines

1.2.1 Neural networks on imagenet

- We have a dataset of 1M images; each image belongs to one class: cat, dog, boat, etc...
- Goal: train an algorithm that takes as input an image and outputs a predicted class.
- Mindset: we have an expressive function that takes images and outputs class. We want to learn the parameters. Everything is learned from data: the only thing the researcher specifies is the architecture of the network.

- SGD training: feed a few images to the network, penalize errors.
- And that's it.

More formally, we have a dataset of images $(x_1, \dots, x_n)$, where here $n = 10^6$.

Q: how do you represent an image in a computer?

Each image x_i is represented as a $3-D$ tensor of dimension $3 \times l \times h$ where l is the length and h the width. This is a raw representation of the image. We call $\mathcal{X}$ this input space. Each image also comes with a label $y_1, \dots, y_n$, where each y_i is the corresponding class of the image: $y_i \in \{cat, dog, shoe, boat...\}$. There are k different classes. This is a **classification** problem. The researcher then designs a neural network, that is a function of two variables

$$f(x; \theta)$$

where x are inputs belonging to $\mathcal{X}$, $\theta \in \mathbb{R}^p$ are parameters. A very important property of this function is that it is by design differentiable with respect to θ. How to design such functions will be the topic of future courses.

Q: what should be the output space of the neural network, having classification in mind?

Classically, the neural network outputs one value per class: $f(x, \theta) = z \in \mathbb{R}^k$. The idea is that the coordinate z_j should be large if the input x belongs to the class j, and small otherwise.

Q: how do you take a decision with such a network?

The neural network is trained by finding good parameters θ, so that the expected behavior above happens. We therefore need a way to measure how well the output z corresponds to the class y of the sample. Hence, we need to define a cost function $\ell(z, y) \in \mathbb{R}$.

Q: can you think of a way of doing so?

Usually, we use the cross-entropy loss:

$$\ell(z, y) = \log(\sum \exp(z_j)) - z_y$$

Q: can you show that this is a good loss? i.e., ≥ 0, when does it equal 0? show that it is shift-invariant.

Hence, we have a way to tell how well the neural net with parameters θ performs on the image x_i: it is simply $\ell(f(x_i; \theta); y_i)$.

Q: how do you measure the performance of the neural network on the full dataset?

We then find the optimal θ by empirical risk minimization:

$$\min_\theta \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; \theta), y_i).$$

This is done using classical optimization algorithms like SGD, more on that later. Importantly, we made the architecture differentiable to be able to compute the gradient of the loss function; how to compute gradients through a neural network is another important topic that we will discuss later.

Once you have trained the model, i.e. found θ that approximately minimizes the loss, you can use the model for inference. We test the model on held-out data, computing the test accuracy and test loss.

History of imagenet: Before neural nets, 25% top-5 error. First neural net (2012): 15%. Now: 1%

1.2.2 Training large language models like gpt

Large language models are trained in a very similar fashion. The goal is to understand how natural language works. To do so, we have access to billions of sentences on internet.

Q: what is a good supervised task to learn how natural language works?

Usually, these models are trained by doing next-token prediction: the input x_i is now a sentence, and the output target y is the following word – or token – in the sentence.

Q: how can we represent text in the computer? it should be in a vectorized form.

Each sentence is then transformed into a sequence of vectors: $x_i = [v_1, \dots, v_q]$, where each vector corresponds to a word, and the target y is a single word in the vocabulary of size 60k. The length of the sequences varies, so we have an architecture that can take arbitrary-length sequences:

$$z = f([v_1, \dots, v_q]; \theta)$$

More on how to build such architectures later.

Then, the networks is trained once again by empirical risk minimization with the cross-entropy loss. At inference, the network can take a sequence of inputs $[v_1, \dots, v_q]$ and then construct the next token.

Q: how to construct the next token? what if you want randomness?

We usually compute $z = f([v_1, \dots, v_q]; \theta)$ and then sample from the law $\text{softmax}(z/t)$, where t is a temperature: if t goes to 0, the generation becomes non-random, if t is high the network will generate more non-random tokens.

Q: how to generate long sentences?

We simply input $[v_1, \dots, v_{q+1}]$ into the llm.

1.3 Machine learning and optimization

1.3.1 Empirical risk minimization

As we have seen, empirical risk minimization is a classical way to solve a ML problem. It means that we need to be able to minimize a function. This is called optimization.

We consider a function $\mathcal{L} : \mathbb{R}^p \to \mathbb{R}$, and the problem $\min_\theta \mathcal{L}(\theta)$. Usually in ML the function f is differentiable, i.e. it has a gradient such that

$$\mathcal{L}(\theta + d\theta) = \mathcal{L}(\theta) + \langle \nabla \mathcal{L}(\theta), d\theta \rangle + o(d\theta)$$

A simple idea to minimize $\mathcal{L}$ is to proceed iteratively, building a sequence θ^t that hopefully converges to the solution.

Q: At θ^t, in which direction does the function decrease the most?

It is therefore natural to follow the gradient; this is gradient descent:

$$\theta^{t+1} = \theta^t - \rho \nabla \mathcal{L}(\theta^t)$$

note: we will later see how to proceed when the function is not necessarily differentiable, e.g. if there is a $\ell - 1$ norm in it.

1.3.2 The example of least squares

We consider a regression problem, where inputs are $x_1, \dots, x_n \in \mathbb{R}^p$, and the targets are $y_1, \dots, y_n \in \mathbb{R}$. We consider a simple linear model:

$$y = \langle \theta, x \rangle + \varepsilon, \;\; \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

Q: what is the probability of observing y given θ and x?

We have that $p(y|x, \theta) = p(y - \langle \theta, x \rangle | x, \theta) = N(y - \langle \theta, x \rangle, \sigma^2)$.

In order to find the best parameters, we therefore maximize this probability, and find that the corresponding optimization problem is $\min \frac{1}{2n} \sum_{i=1}^{n} \|\langle \theta, x_i \rangle - y_i\|^2$

Q: what kind of function is it? What is the solution?

It is a simple quadratic function, minimized when the gradient cancels.

2 Unsupervised learning

Assume that we have access to a set of unlabeled samples $x_1, \dots, x_n$. We do not have access to any label, yet we want to extract information about this dataset. We therefore want to understand the structure of this data.

This is super common in practice because annotating is usually costly. We will explore two classical methods to do so.

2.1 Clustering: k-means

We want to group examples in the dataset: similar points should belong to the same *cluster*.

Draw pictures in 2d:

- Case where there is an obvious clustering
- Case where there is none

Formally, we want to split the data into k clusters, i.e. build a selection function $\phi : \mathbb{R}^p \to \{1, k\}$.

Q: how would you do this?

We take a list of clusters $C_1, \dots, C_k$, such that $\cup^o C_i = \{1, \dots n\}$.

Q: How would you define the quality of a clustering? We want points in the same cluster to be close:

$$\mathcal{L}(C_1, \dots, C_k) = \sum_{l=1}^{k} \frac{1}{2\#C_l} \sum_{i,j \in C_l} \|x_i - x_j\|^2$$

Problem: this is a combinatorial problem, solving it requires exploring all the possible subsets... too hard to do.

Idea: have a list of k centroids $b_1, \dots, b_k \in \mathbb{R}^p$.

Q: if the centroids are fixed, how do you assign a sample x to a cluster?

Simply take $\arg\min \|x - b_i\|$.

Q: how can you rewrite the loss function with the centroids?

$$\mathcal{L}(C_1, \dots, C_k) = \sum_{l=1}^{k} \sum_{i \in C_l} \|x_i - b_l\|^2$$

Q: how can you minimize this loss?

alternately update clusters and centroids.

Q: define the function of both centroids et clusters

$$\mathcal{G}(C_1, \dots, C_k, b_1, \dots, b_k) = \sum_{l=1}^{k} \sum_{i \in C_l} \|x_i - b_l\|^2$$

what can you say about $\mathcal{L}$ and $\mathcal{G}$?

$$\min_{b_1, \dots, b_k} \mathcal{G} = \mathcal{L}$$

Q: how do you minimize G wrt. b?

Simply take $b_i = \frac{1}{\#C_i} \sum_{i \in C_i} x_i$

Q: how do you minimize G wrt clusters?

Simply take $C_l = \{i| \ \|x_i - b_l\| \text{ is minimal}\}$.

Therefore, we have an alternate minimization scheme for the k-means problem.

Algorithm (Lloyd):

- Initialize centroids $b_1, \dots, b_k$ (Q: how??)
- iterate:
  - Assign clusters $C_l = \{i| \ \|x_i - b_l\| \text{ is minimal}\}$
  - Update centroids

Does this algorithm converge? Hard to prove... saddle points (e.g. if points are all in 1 cluster...)

Q: what is the complexity of the algorithm? Each iteration is $n \times k \times p$.

Q: how to estimate the number of clusters?

Draw an example on the board; look for a knee in the loss function.

Q: l2 distance

2.2 Principal component analysis

k means searches for groups in the data; PCA is a way to find a low rank representation of the data.

We have a dataset $x_1, \dots, x_n \in \mathbb{R}^p$. For simplicity, assume zero-mean.

We want to find one direction of space $u \in \mathbb{R}^p$ that explains as much as possible the dataset, i.e., such that there are factors $z_i \in \mathbb{R}$ such that

$$x_i \simeq z_i u$$

Q: is the problem well posed

there is a scale indeterminacy so we impose $\|u\| = 1$

Q: geometrically, what does it mean?

points are almost in a 1-d space

Q: how can you find the coefficients z_i given u? how to compute u?

$z_i = \arg\min_z \|x_i - zu\|^2$

and then simply

$$\min_{u,z} \frac{1}{n} \sum_{i=1}^{n} \|x_i - z_i u\|^2$$

Q: what is the solution?

$z_i = \langle x_i, u \rangle$ so equivalent to

$$\max \sum_{i=1}^{n} \langle x_i, u \rangle^2$$

variance maximization

In other words,

$$\max u^T C u$$

where C is the covariance matrix.

u: largest eigenvector of the covariance matrix.

Q: How to extend this to more ranks? Simply consider $y_i = x_i - z_i u$, and then try to fit a PCA on this (deflation).

Corresponds to second largest eigenvalue of covariance

Q: how would you use this for data vizualisation?

2.3 SVD

We can write any matrix $X \in \mathbb{R}^{n \times p}$, with $n \geq p$, as $X = U \Sigma V^T$ with $U \in \mathbb{R}^{n \times p}$ s.t. $U^T U = I_p$, Σ diagonal positive, and $V \in O_p$.

The principal components are the rows of V.