202305241229
Tags: #WIP #Course

Advanced Statistical Physics Lecture 1

Statistical mechanics and its tools can be used to model the workings of machine learning (ML) techniques. In this course the focus will be put on modelling Neural Networks (NN) in the context of deep learning.

Introduction

Intelligence can be categorized as *biological* or *artificial*.
Biological intelligence is studied by the field of Neuroscience, Medicine and Mathematics, and the neuron itself, the basic unit of a biological brain, tends to be the main focus of such studies.
The study of artificial intelligence is however more focused on the study of networks of neurons, where artificial neurons are modelled as idealized mathematical versions of their biological counterpart. ML is a subset of artificial intelligence that includes NNs but also many other types of algorithms that "learn" from data. An example of one such algorithm beside NNs is the k-nearest neighbors, which essentially classifies a new data point based on its distance between k given examples of the data already known (the so called *training set*) that are closest to it. If $k = 7$ for example, and 4 points belong to a category labelled "cat", 2 to "dog", and the remaining one to "bird", then the considered point is classified as "cat". Note that the distance doesn't have to be the Euclidean distance in feature space, for example one may use the Wasserstein distance[1] if it is more appropriate for the type of data considered (in image recognition for example).
Deep learning is a particular subset of machine learning that includes most instances of NNs. The learning itself can be:

- supervised, where training data is labeled;
- unsupervised, where training data in unlabeled. Examples can be clustering algorithms, that look at how points cluster in feature space, or generative models (eg. Generative Adversarial Networks[2], Diffusion models[3]);
- reinforcement learning, where training happens from the interaction with the network and an environment (i.e., other networks), which either rewards or punishes the network based on its output[4]. Examples of networks trained like this are Alpha-zero and Alpha-go.

Historically, the hype for artificial intelligence began in 1958, with Rosenblatt's paper on the *perceptron*, which today still constitutes the basic unit of a NN. After this first result, interest in AI began to wane throughout the 70s, until several new results reignited interest in the topic[5]. Finally, a new period of stagnation persisted from 1995 onwards, when Support

Vector Machines were dominating the research on machine learning. Interest was sparked again in 2012, when a Convolutional NN won the ImageNET classification challenge by a huge margin. From then on, the popularity of NN increased exponentially up to now, with the latest results being huge generative models such as GPT-4.

Supervised Learning and NNs

Supervised learning is given by four main ingredients:

1. a training set $D = \{\vec{x}_\mu, \vec{y}_\mu\}$ with $\mu = 1, \cdots, p$. Each example $(\vec{x}_\mu, \vec{y}_\mu) \in D$ is made of an input vector $\vec{x}_\mu \in \mathbb{R}^n$ of n features and an output vector (or scalar) $\vec{y}_\mu \in \mathbb{R}^m$ of m labels. In classification problems, the labels are generally given as an integer[6] or as unit vectors of dimensionality equal to the numbers of categories considered, where the corresponding position in the vector is equal to 1, while all the others are 0. An example could be a set where the features are biometric measurements (eg. height, weight, age, ...) and the corresponding labels are the sex (male or female). The examples are assumed to be *independent and identically distributed* (iid) according to some unknown joint probability density function $P(\vec{x}, \vec{y})$;
2. A model $f(\vec{x}; \vec{\theta})$ that depends on paramters $\vec{\theta}$.
   A possible model can be a NN. The most typical architecture for a NN is a chain of layers $l = 1, \cdots, L$, each layer made of a number N_l of neurons. Neurons in a given layer l have weights, which can be identified $w_i^{(l)}$, $i = 1, \cdots, N_l$. Each layer takes the output from the previous one, and applies the weights to transform the output through an affine transformation $\mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$, the so-called *pre-activation* of the layer. Then a non-linear transformation $\sigma : \mathbb{R}^{N_l} \to \mathbb{R}^{N_l}$ is applied before the output being passed to the next layer. These models are called *feed-forward networks*, the simplest of which are *fully connected networks*, in which each neuron i in the l-th layer is connected by to every other neuron j in the $(l-1)$-th layer. One can define a $W^{(l)} \in \mathbb{R}^{N_l} \times \mathbb{R}^{N_{l-1}}$ matrix that maps how the weights in layer $l-1$ transform the input at that layer into the output fed to the next layer l.
   Mathematically, the pre-activation $h^{(l)}$ and activation $\phi^{(l)}$ at layer l can be written as:

$$h_i^{(l)}(\vec{x}) = \sum_{j=1}^{N_{l-1}} W_{ij}^{(l)} \phi_j^{(l-1)}(\vec{x}) + b_i^{(l)},$$
$$\phi_i^{(l)}(\vec{x}) = \sigma\left(h_i^{(l)}(\vec{x})\right),$$

   where the $b_i^{(l)}$ are called the bias of the layer. The activation and pre-activation can be obtained recursively at each layer starting from $\phi_i^{(0)} = x_i$.
   Generally, a NN defined in this way can be modelled as a chained composition of affine transformation and non-linear activations:

$$f(\vec{x}, \vec{\theta}) = \sigma \circ W^{(L)} \circ \sigma \circ W^{(L-1)} \circ \cdots \circ \sigma \circ W^{(1)}(\vec{x}).$$

   All the hyperparameters of the network (L, N_l, σ, ...) are free and are generally tuned.

3. A loss function $\mathcal{L}(\vec{\theta})$, which some measure of the distance between the training labels y_μ and the outputs of the model.
   There are many possible choices for the loss, but the simplest change based on the type of problems. In regression problems the quadratic loss is the default:

$$\mathcal{L}(\vec{\theta}) = \sum_{\mu=1}^{p} \Big(y_\mu - f(\vec{x}_\mu, \vec{\theta}) \Big)^2,$$

   while in binary classification problems (with $y = \pm 1$) one choice is:

$$\mathcal{L}(\vec{\theta}) = \sum_{\mu=1}^{p} \theta_H \Big(-y_\mu f(\vec{x}_\mu, \vec{\theta}) \Big),$$

   where the θ_H is the Heavyside function.
4. A learning algorithm that tunes the parameters $\vec{\theta}$ of the model by minimizing the loss based on the training set.
   For a NN, the usual approach is to initialize the weights randomly, sampling from a gaussian distribution, and then use some gradient descent algorithm (e.g., stochastic gradient descent) such that $-\nabla_{\vec{\theta}} \mathcal{L} = 0$. NNs tend to have non-convex loss functions, so their optimization is challenging.

The training error in a classification problem tackled through a supervised learning scheme defined as $\epsilon_T = \mathcal{L}(\vec{\theta})/p$, which gives the fraction of wrong guesses by the model $f(\vec{x}, \vec{\theta})$ on the training set. What we are usually interested in, rather than ϵ_T, is the generalization error:

$$\epsilon_G = \iint \mathrm{d}^3 x \, \mathrm{d}y \, \theta_H(-y f(\vec{x}, \vec{\theta})) P(\vec{x}, y).$$

Since the joint probability $P(\vec{x}, y)$ is not known, we usually reserve part of the available data as validation and test sets, and approximate the generalization error by computing the loss on these **after** training.

References

---

1. See Optimal Transport Theory for context.↩
2. See the work of Ian Goodfellow.↩
3. See the work of Jascha Sohl-Dickstein.↩
4. See Control Theory and the Bellman equation.↩
5. eg. Hopfield, 1986, Rumelhart, 1986, LeCun, 1989, ...↩
6. We'll consider integer labels from now on, unless specified otherwise.↩